



ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR



Doctoral Dissertation
Doctoral Program in Control and Computer Engineering (XXXIII cycle)

Novel Validation Techniques for Autonomous Vehicles

Jacopo Sini

* * * * *

Supervisor

Prof. Massimo Violante, Supervisor

Doctoral Examination Committee:

Examiners list not defined

Politecnico di Torino
Date not yet defined

This thesis is licensed under a Creative Commons License, Attribution - Noncommercial-NoDerivative Works 4.0 International: see www.creativecommons.org. The text may be reproduced for non-commercial purposes, provided that credit is given to the original author.

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

.....

Jacopo Sini
Turin, Date not yet defined

Summary

The automotive industry is facing challenges in producing electrical, connected, and autonomous vehicles. Even if these challenges are, from a technical point of view, independent from each other, the market and regulatory bodies require them to be developed and integrated simultaneously.

The development of autonomous vehicles implies the development of highly dependable systems. This is a multidisciplinary activity involving knowledge from robotics, computer science, electrical and mechanical engineering, psychology, social studies, and ethics.

Nowadays, many Advanced Driver Assistance Systems (ADAS), like Emergency Braking System, Lane Keep Assistant, and Park Assist, are available. Newer luxury cars can drive by themselves on highways or park automatically, but the end goal is to develop completely autonomous driving vehicles, able to go by themselves, without needing human interventions in any situation.

The more vehicles become autonomous, the greater the difficulty in keeping them reliable. It enhances the challenges in terms of development processes since their misbehaviors can lead to catastrophic consequences and, differently from the past, there is no more a human driver to mitigate the effects of erroneous behaviors.

Primary threats to dependability come from three sources: misuse from the drivers, design systematic errors, and random hardware failures.

These safety threats are addressed under various aspects, considering the particular type of item to be designed. In particular, for the sake of this work, we analyze those related to Functional Safety (FuSa), viewed as the ability of a system to react on time and in the proper way to the external environment.

From the technological point of view, these behaviors are implemented by electrical and electronic items.

Various standards to achieve FuSa have been released over the years. The first, released in 1998, was the IEC 61508. Its last version is the one released in 2010.

This standard defines mainly:

- a Functional Safety Management System (FSMS);
- methods to determine a Safety Integrated Level (SIL);
- methods to determine the probability of failures.

To adapt the IEC61508 to the automotive industry’s peculiarity, a newer standard, the ISO26262, was released in 2011 then updated in 2018.

This standard provides guidelines about FSMS, called in this case *Safety Lifecycle*, describing how to develop software and hardware components suitable for functional safety. It also provides a different way to compute the SIL, called in this case Automotive SIL (ASIL), allowing us to consider the average driver’s abilities to control the vehicle in case of failures. Moreover, it describes a way to determine the probability of random hardware failures through Failure Mode, Effects, and Diagnostic Analysis (FMEDA).

This dissertation contains contributions to three topics:

- random hardware failures mitigation;
- improvement of the ISO26262 Hazard Analysis and Risk Assessment (HARA);
- real-time verification of the embedded software.

As the main contribution of this dissertation, I address the safety threats due to random hardware failures (RHF).

For this purpose, I propose a novel simulation-based approach to aid the Failure Mode, Effects, and Diagnostic Analysis (FMEDA) required by the ISO26262 standard. Thanks to a SPICE-level model of the item, and the adoption of fault injection techniques, it is possible to simulate its behaviors obtaining useful information to classify the various failure modes. The proposed approach evolved from a mere simulation of the item, allowing only an item-level failure mode classification up to a vehicle-level analysis. The propagation of the failure modes’ effects on the whole vehicle enables us to assess the impacts on the vehicle’s drivability, improving the quality of the classifications. It can be advantageous where it is difficult to predict how the item-level misbehaviors propagate to the vehicle level, as in the case of a *virtual differential gear* or the mobility system of a robot. It has been chosen since it can be considered similar to the novel light vehicles, such as electric scooters, that are becoming more and more popular. Moreover, my research group has complete access to its design since it is realized by our university’s DIANA students’ team. When a SPICE-level simulation is too long to be performed, or it is not possible to develop a complete model of the item due to intellectual property protection rules, it is possible to aid this process through behavioral models of the item. A simulation of this kind has been performed on a mobile robotic system. Behavioral models of the electronic components were used, alongside mechanical simulations, to assess the software failure mitigation capabilities.

Another contribution has been obtained by modifying the main one. The idea was to make it possible to aid also the Hazard Analysis and Risk Assessment (HARA).

This assessment is performed during the *concept phase*, so before starting to design the item implementation. Its goal is to determine the hazards involved in the item functionality and their associated levels of risk. The end goal of this phase is a list of *safety goals*. For each one of these safety goals, an ASIL has to be determined. Since HARA relies only on designers expertise and knowledge, it lacks in objectivity and repeatability.

Thanks to the simulation results, it is possible to predict the effects of the failures on the vehicle's drivability, allowing us to improve the severity and controllability assessment, thus improving the objectivity. Moreover, since simulation conditions can be stored, it is possible, at any time, to recheck the results and to add new scenarios, improving the repeatability.

The third group of contributions is about the real-time verification of embedded software. Through Hardware-In-the-Loop (HIL), a software integration verification has been performed to test a fundamental automotive component, mixed-criticality applications, and multi-agent robots.

The first of these contributions is about real-time tests on Body Control Modules (BCM). These modules manage various electronic accessories in the vehicle's body, like power windows and mirrors, air conditioning, immobilizer, central locking. The main characteristics of BCMs are the communications with other embedded computers via the car's vehicle bus (Controller Area Network) and to have a high number (hundreds) of low-speed I/Os.

As the second contribution, I propose a methodology to assess the error recovery system's effects on mixed-criticality applications regarding deadline misses. The system runs two tasks: a critical airplane longitudinal control and a non-critical image compression algorithm. I start by presenting the approach on a benchmark application containing an instrumented bug into the lower criticality task; then, we improved it by injecting random errors inside the lower criticality task's memory space through a debugger. In the latter case, thanks to the HIL, it is possible to pause the time domain simulation when the debugger operates and resume it once the injection is complete. In this way, it is possible to interact with the target without interfering with the simulation results, combining a full control of the target with an accurate time-domain assessment.

The last contribution of this third group is about a methodology to verify, on multi-agent robots, the synchronization between two agents in charge to move the end effector of a delta robot: the correct position and speed of the end effector at any time is strongly affected by a loss of synchronization.

The last two contributions may seem unrelated to the automotive industry, but interest in these applications is gaining. Mixed-criticality systems allow reducing the number of ECUs inside cars (for cost reduction), while the multi-agent approach is helpful to improve the cooperation of the connected cars with respect to other vehicles and the infrastructure.

The fourth contribution, contained in the appendix, is about a machine learning application to improve the social acceptance of autonomous vehicles. The idea is to improve the comfort of the passengers by recognizing their emotions. I started with the idea to modify the vehicle's driving style based on a real-time emotions recognition system but, due to the difficulties of performing such operations in an experimental setup, I move to analyze them offline. The emotions are determined on volunteers' facial expressions recorded while viewing 3D representations showing different calibrations. Thanks to the passengers' emotional responses, it is possible to choose the better calibration from the comfort point of view.

Acknowledgements

I would like to acknowledge my advisor, Prof. Massimo Violante, for offering me this excellent chance, and all the people who collaborated with me during my PhD programme as coauthor, Master's Degree thesisists, and some colleagues who worked closely with me. These people are (in rigorous alphabetical order): Antonio Arena, Concetta Argiri, Serhiy Avramenko, Enea Bagalini, Prof. Radu Bojoi, Ludovica Bozzoli, Sebastiano Campisi, Alessio D'Andrea, Marco D'Auria, Stefano De Caro, Riccardo Dessì, Vincenzo Dodde, Stefano Esposito, Flavio Fusetti, Rubin Gnaniah, Chunying Ma, Francesco Mangiacane, Antonio Costantino Marceddu, Prof. Bartolomeo Montrucchio, Alessandra Mugoni, Andrea Passarino, Luca Pecorella, Davide Piumatti, Aldo Quario, Filippo Santonocito, Mariangela Saracco, Peter Sarson, Prof. Matteo Sonza Reorda, Enrico Zanda.

*I would like to dedicate
this dissertation to my
loving parents Giovanni
Battista and
Vincenzina, and to my
aunt Donatella.*

Contents

List of Tables	XV
List of Figures	XVIII
1 Introduction	7
1.0.1 Software: an opportunity and an issue	8
1.0.2 The software complexity in the automotive industry	9
1.1 Industry trends on automation of vehicles safety and driving functions	9
1.1.1 Commercial applications	10
1.2 Autonomous vehicles classification	13
1.2.1 SAE J3016	13
1.2.2 ECR classification: a responsibilities-based point of view . .	14
1.3 Functional Safety	16
1.3.1 Overview and position of Functional Safety in the autonomous driving framework	16
1.3.2 FIDES	24
1.3.3 AIAG&VDA FMEA manual	24
1.4 Technical contributions on this dissertation	24
1.4.1 Proposals that contribute to improving the FMEDA	25
1.4.2 Proposals that contribute to improving the HARA	25
1.4.3 Proposals that contribute on improving real-time software validation	26
1.5 Structure of this dissertation	27
2 Items development inside the ISO26262 framework	29
2.1 Safety Lifecycle	30
2.1.1 Structure of the ISO26262	30
2.1.2 Functional Safety Management	30
2.1.3 Confirmation measures	33
2.2 Concept Phase	36
2.2.1 Item definition	36
2.2.2 Hazard Analysis and Risk Assessment	36

2.2.3	Safety Element out of Context	43
2.3	Functional safety concept	43
2.3.1	Functional parameters	44
2.3.2	Safety architecture	45
2.3.3	ASIL decomposition	45
2.3.4	Functional Safety Requirements	46
2.4	Implementation phases	50
2.5	Technical safety concept	53
2.6	FMEA	53
2.7	Hardware design and FMEDA	54
2.7.1	Classification rules	55
2.7.2	Rates and metrics	55
2.7.3	Failure metrics	58
2.8	FIDES	58
2.8.1	Guide structure	59
2.8.2	Definitions	59
2.8.3	Model coverage	61
2.8.4	Reliability prediction	61
2.8.5	Predicted reliability evaluation guide	63
2.8.6	General model	64
2.8.7	Life profile	64
2.9	Fault injection	66
2.10	Differences between FMEA and FMEDA	67
2.11	Software Development for Embedded Systems	67
2.11.1	Vocabulary	70
2.11.2	V-model	70
2.11.3	Model-based software design	71
2.11.4	Software unit testing	71
2.11.5	Software integration testing	73
3	Simulation-based FMEDA	77
3.1	Industrial practice	78
3.2	The idea of automating the FME(D)A	78
3.2.1	Fault coverage measurement	79
3.2.2	Fault models	80
3.2.3	Fault injection	81
3.3	Proposals	81
3.3.1	Hypotesis	82
3.3.2	Evolution of the proposed approaches	82
3.4	Research contributions	86
3.5	First proposal - A simulation-based FME(D)A	88
3.5.1	Fault models	88

3.5.2	Fault Injection	89
3.5.3	Failure modes effects assessment strategies	89
3.5.4	Assessment of SW mitigation capabilities on HW failures	91
3.5.5	Experimental setup	92
3.5.6	Simulation results	92
3.6	Second proposal - A real automation of FMEDA	95
3.6.1	Fault models	99
3.6.2	Fault injection	101
3.6.3	Failure modes effects assessment strategies	102
3.6.4	Assessment of the SW mitigation capabilities on HW failures	104
3.6.5	Experimental setup	104
3.6.6	Simulation results for [16]	106
3.6.7	Simulation results for [18]	108
3.7	Third proposal - Handmade vs. simulation-based approaches	112
3.7.1	Fault models	112
3.7.2	Fault injection	112
3.7.3	Failure modes effects assessment strategies	112
3.7.4	Assessment of the SW mitigation capabilities on HW failures	113
3.7.5	Experimental setup	113
3.7.6	Simulation results	114
3.8	Fourth proposal - From an item-level FMs classifications to a vehicle-level one	118
3.8.1	Fault models	122
3.8.2	Fault injection	122
3.8.3	Failure modes effects assessment strategies	122
3.8.4	Assessment of the SW mitigation capabilities on HW failures	123
3.8.5	Experimental setup	123
3.8.6	Simulation results	125
3.9	Fifth proposal - From safety to mission criticality: finding a tradeoff between needs and model precision	131
3.9.1	Fault models	137
3.9.2	Fault injection	138
3.9.3	Failure modes effects assessment strategies	139
3.9.4	Assessment of the SW mitigation capabilities on HW failures	140
3.9.5	Experimental setup	140
3.9.6	Simulation results	141
3.10	Sixth proposal - Application to a mobile robotics case study	144
3.10.1	Description of the rover	147
3.10.2	FMEA of the rover	148
3.10.3	Fault models	151
3.10.4	Fault injection	152
3.10.5	Failure modes effects assessment strategies	152

3.10.6	Assessment of the SW mitigation capabilities on HW failures	153
3.10.7	Experimental setup	163
3.10.8	Simulation results	163
3.11	Summary	169
4	Simulation-based HARA	171
4.1	State of the art	173
4.1.1	Research contribution	174
4.2	Proposed methodology	176
4.2.1	Situation Analysis and Hazard Identification	179
4.2.2	Hazard classification	179
4.2.3	ASIL determination	180
4.2.4	Safety objective definition	180
4.2.5	Review	180
4.2.6	Summary	180
4.3	Benchmark case study	181
4.3.1	AEBS and its integration into the vehicle	181
4.3.2	Fault model	182
4.3.3	The proposed methodology	182
4.4	Simulation results	184
4.4.1	EuroNCAP AEBS test protocol	185
4.4.2	NHTSA tests	187
4.4.3	European Commission Regulation 347/2012 tests	188
4.4.4	ASIL Assignment	189
5	Real-time software validation	191
5.0.1	Research contribution	192
5.1	Automotive Body Control Modules	192
5.1.1	Proposed approach	194
5.1.2	Benchmark proof-of-concept setup	196
5.2	Mixed-criticality systems	200
5.2.1	Proposed approach	202
5.2.2	Experimental results with the triggering of unsolvable known defect in a low criticality software component - [9]	211
5.2.3	Experimental results with FI, pause signal, and eDB - [10]	213
5.3	Multi-agent robotic system (MAS) development	216
5.3.1	Proposed approach	217
5.3.2	Benchmark application	219
5.3.3	Control software and models	221
5.3.4	Experimental results	223

6	Conclusions and Future Work	229
6.1	Contributions on Simulation-based FMEDA	229
6.1.1	Advantages	231
6.1.2	Limitations	232
6.1.3	Future development	232
6.2	Contributions on Simulation-based HARA	233
6.2.1	Advantages	233
6.2.2	Limitations	234
6.2.3	Future development	234
6.3	Contributions on Real-time software validation	234
6.3.1	Automotive Body Control Modules	235
6.3.2	Mixed-criticality avionic systems	236
6.3.3	Multi-agent robotic systems	237
A	Machine learning applications for the automotive industry	239
A.1	A formal model for the human emotions	243
A.1.1	Facial expression databases	243
A.2	Neural networks	245
A.2.1	Generalization capability: underfitting and overfitting	246
A.2.2	Vocabulary	246
A.2.3	Performance assessment metrics	247
A.2.4	Performance improvement techniques	247
A.3	Recognition of emotions from facial expression	249
A.3.1	Facial Expressions Databases Classifier	250
A.3.2	Choice of the neural networks	251
A.4	Neural Networks Training	251
A.4.1	Training Environment Set-Up	252
A.4.2	Training Results from [110]	253
A.4.3	Training results from [111]	263
A.5	Autonomous driving algorithms assessment from [110].	268
A.5.1	Situations Preparation	268
A.5.2	Criteria for Emotion Analysis	268
A.5.3	Experimental Campaign	269
A.5.4	Results Discussion	269
A.6	Road Tests	272
A.7	Conclusions	274
A.7.1	Training results	274
A.7.2	Future works	276
	Bibliography	277

List of Tables

2.1	Overview of verification measure. From [7]. o No requirement and no recommendation for or against X required X* Scope of this review also includes hazardous events rated as QM.	34
2.2	Overview of verification measure. From [7]. o No requirement and no recommendation for or against + Recommended ++ Required	35
2.3	ISO 26262-3 Table B1: Examples of severity classification.	40
2.4	Adapted from ISO 26262-3 Table 6 Class of controllability. The row <i>Driving factors</i> is extracted from table B.6.	40
2.5	Adapted from ISO 26262-3 Table 6 Class of controllability. The row <i>Duration</i> is extracted from table ISO26262:2018-3 B.2 The row <i>Frequency of situation</i> is extracted from table ISO26262:2018-3 B.3.	41
2.6	Mapping of MSIL to ASIL.	42
2.7	Random hardware failure metrics limits.	58
3.1	Simulation results obtained from [15].	94
3.2	FMEDA assessment results obtained by the tool presented in [16] considering 5 different workloads.	107
3.3	FMEDA assessment results obtained by the tool presented in [18].	111
3.4	FMEDA assessment result comparison between the handmade and the automatically performed one.	114
3.5	Comparison between the failure classifications obtained by the automatic tool and from the experts.	115
3.6	Comparison between the handmade and the automatic assessments. The differences between the two classifications are highlighted.	117
3.7	Simulation results. Conditions: FA indicates fault affected without mitigation algorithm, while M indicates fault conditions with the mitigation algorithm enabled	126
3.8	The classification criteria, with their tolerances, as defined in the system design phase (as defined by the complex system designer).	140
3.9	Failure modes classification results.	143

3.10	Failures grouping summary. The characters a , b and x in the previous table indicate what components can fail in each group: components marked as x can fail simultaneously in any number between 0 and 9; components marked as a can fail simultaneously in a maximum number of 2; components marked as b can fail simultaneously in a maximum number of 3 for groups numbered 3,5,6,7,8 or in a maximum number of 2 for groups numbered 10,11,12.	155
3.11	Flags-failures associations.	157
3.12	Flags-failures associations for all the components.	159
3.13	Traction subsystem mitigation levels.	159
3.14	Failures-states-mitigations associations.	160
3.15	Flags-failures associations for all the components.	162
3.16	Simulation results for the traction subsystem.	167
3.17	Simulation results for the steering subsystem.	168
4.1	Severity classification rules. Table from [66].	183
4.2	Severity classification rules. Table from [66].	183
4.3	Results from the simulations inside the EuroNCAP scenarios. Table from [66].	186
4.4	Results from the simulations inside the NHTSA scenarios. Table from [66].	188
4.5	Results from the simulations inside the NHTSA scenarios. Table from [66].	188
4.6	ASIL classification of the various tests. Table from [66].	189
5.1	Average time-domain performance comparison between the results obtained in MIL and SIL. Table from [9].	211
5.2	Average time-domain performances comparison between a full software implementation in the real-time computer and with the loop closed on the DUT in fault-free conditions. Table from [9].	213
5.3	Time-domain performances comparison between the loop closed on the DUT in fault-free and fault-affected situations. Table from [9].	213
5.4	Time-domain performances characteristics in fault-free conditions. Table from [10].	214
5.5	Hardware fault injection classification. Table from [10].	214
5.6	Software fault injection classification. Table from [10].	215
5.7	Interference detection. Table from [10].	215
A.1	Picture available for each emotion in the chosen databases. Table from [110].	254
A.2	Subdivision of the databases. Table from [110].	262
A.3	Test accuracies summary table (best values). Table from [110].	263
A.4	Picture available for each emotion in the databases trained in [111].	264

- A.5 Emotional effects of the benchmark tests. In the columns are indicated the number of people that reacted to the considered *situation* with the emotion on the left. Data obtained by the network in [116], trained with the *Ensemble 1* database using the data augmentation (see section A.2.4) and the z-score normalization. Table from [110]. 271
- A.6 Test accuracies summary table (best values) for all the trainings described in this dissertation. Data retrieved from [110] (for the trainings on *Ensemble 1*, *Ensemble 2*, CK+, and FER2013) and from [111] (for the trainings on *Ensemble 3* and IMFDB). . 275

List of Figures

1.1	An artistic representation of a System on Chip (SoC) Cypress PSoC 5. Figure from Cypress Semiconductor.	8
1.2	A crash (one fatality) involving a Tesla car occurred on March 2018 in California. Image from ABC News.	11
1.3	A Waymo robotaxi.	12
1.4	A Amazon Zoox robotaxi.	13
1.5	SAE J3016 Autonomous vehicles classification.	14
1.6	Operating modes classification proposed by Edge Case Research. . .	15
1.7	Relationship between the standars involved in the development of autonomous driving systems	18
1.8	The HARA process inside the ISO26262 framework. Figure adapted from [6].	20
1.9	The HARA process inside the ISO/PAS 21488 (SOTIF) framework. Figure adapted from [6].	21
1.10	The HARA process inside the UL 4600 framework. Figure adapted from [6].	21
1.11	The structure of a UL 4600 Safety Case, with an EooC. Figure adapted from [6].	22
1.12	Summary of the differencies between KPIs and SPIs.	23
1.13	Topics discussed in this dissertation, with common application on automotive items and their position into the ISO26262 <i>safety lifecycle</i>	25
2.1	The ISO26262:2018 structure	30
2.2	Hierarchy of Safety Goals and Functional Safety Requirements. Figure adapted from [7].	37
2.3	HARA subdivision in phases.	37
2.4	Exposure classification in duration or frequency.	41
2.5	ASIL determination matrix.	42
2.6	Relationship between design assumptions and SEooC development. Figure adapted from [7].	44
2.7	ISO26262-9 Figure 2 Classification scheme of ASILs when decomposing safety requirements. Figure adapted from [7].	46
2.8	Failure classification between <i>systematic</i> and <i>random (hardware)</i> . . .	47

2.9	Failure classification between <i>independent</i> and <i>dependent</i>	48
2.10	Dependent failure analysis. Figure from [7].	49
2.11	Time intervals relevant for safety. Figure from [7].	50
2.12	System level development process with the main involved documents.	52
2.13	The 7 steps described by the AIAG and VDA FMEA manual. Figure from [5].	54
2.14	The content of the outcome result of FMEDA analysis.	55
2.15	The failure mode classificaton criteria.	56
2.16	Relationship between failure cause, mechanism, mode, and reliability contribution factor. Figure adapted from FIDES.	61
2.17	The bathtub curve, showing the three periods of the life of a product.	62
2.18	Relations between the WPs from <i>Configuration management plan to Software verification Report</i>	68
2.19	Relations between the WPs to be prepared during the software implementation process.	69
2.20	The V-diagram of the software development.	71
2.21	HIL conceptual structure compared with a real implementation, based on a National Instruments cRIO as the real-time simulator and a ZedBoard as the controller.	75
3.1	A general three stages structure for an item.	82
3.2	Workflow of the simulation-based methodology evolution. The descriptions near the arrows explain the novelties between the proposals.	83
3.3	Structure of the seat belt reminder.	88
3.4	Schematic representation of the simulation-based approach proposed in [16]	96
3.5	A schematic representation of the item described in [16].	97
3.6	A schematic representation of the item described in [18].	98
3.7	Schematic representation of an inverter connected with a with a star winding brushless direct current (BLDC) permanent magnet motor.	101
3.8	The schematic of the analog conditioning stage described in [16]. The red borders around the resistors R_1 and R_2 indicates the absence of a defined parameter for their resistance values, since it is computed at runtime by the sabouter.	101
3.9	The external interfaces of the SB used to inject stuck-at FMs in [16] and [18]. The red borders around the <code>FaultInjectorSelector</code> constants indicates the absence of a defined value, since it is computed runtime by the sabouter.	102
3.10	The implementation of the SB used to inject stuck-at FMs in [16] and [18].	103
3.11	Examples of SBs, used to inject stuck-at FMs on pins, and simulation-only components (in this case resistors) to inject short circuits between adjacent pins of a commercial microcontroller.	103

3.12	The implementation of the detection algorithm for IGBTs in the [18] paper.	105
3.13	Schematic representation of the simulation-based approach proposed in [17]	113
3.14	Simulink model to perform simulations described in [17]	113
3.15	The block diagram of the proposal. Figure adapted from [19].	120
3.16	The block diagram of the benchmark application.	121
3.17	Mitigation algorithm. Figure from [19].	124
3.18	The triple curving simulated track. The arrows indicates the direction of the ride.	125
3.19	Case b lateral errors with respect to the ideal trajectory. Adapted from [19].	127
3.20	Case b yaw angles measured with respect to the tangent of the ideal trajectory. Adapted from [19].	128
3.21	Case c lateral displacement with respect to the ideal trajectory. Adapted from [19].	129
3.22	Case c yaw angles measured with respect to the tangent of the ideal trajectory. Adapted from [19].	130
3.23	Case e yaw angles measured with respect to the tangent of the ideal trajectory. Adapted from [19].	130
3.24	Representation of the proposed approach. Adapted from [40].	134
3.25	Block diagram representation of the motor control system case study. Adapted from [40].	136
3.26	Schematic of the PSU. Figure from [40].	138
3.27	A boost cell of the PSU instrumented with simulation only switches. Figure from [40].	138
3.28	Schematic of the PSU. Figure from [40].	139
3.29	System level development process with the main involved documents.	146
3.30	A 3D rendering of the Ardito Rover developed by the D.I.A.N.A. student's team of Politecnico di Torino.	149
3.31	System level development process with the main involved documents.	154
3.32	Three stages structure of the a general item.	164
4.1	The structural block diagram of an ADAS device. Figure from [66].	174
4.2	The phases of the HARA, with indication of those ones that can be performed by a simulation-based approach.	174
4.3	Block diagram representation of the proposed approach.	177
4.4	Block diagram representation of the proposed approach, with the HARA phases, mapped to its blocks.	178
4.5	Representation of the situations analized in all the scenarios.	185

4.6	Plot of the relative distance between the VUT and the TV over the time for the case CCRb (12 m, 6 m/s ²). The intersection between the time axis and the relative distance curve represents the TTC. Figure from [66].	187
4.7	Plot of the relative distance between the VUT and the TV over the time for the case CCRb (40 m, 6 m/s ²). The intersection between the time axis and the relative distance curve represents the TTC. Figure from [66].	187
5.1	ECU Simulink™ model, with description of mapping from the HMI to logical resources of the reconfigurable hardware. On the top of the three grayed areas, it is reported the source of the inports/outports labels.	197
5.2	The TXT XHIL Studio simulation editor.	198
5.3	The TXT XHIL Studio simulation player.	198
5.4	Interaction between TXT XHIL Studio and NI VeriStand API.	199
5.5	A benchmark of the proposed approach, performed on simplified ECU implementing a turn indicators control software on an NXP S32K144 reference board, shown in November 2017 at National Instruments Days in Milan, Italy.	200
5.6	The block diagram of the approach presented in the [9] and [10] papers. The blocks regarding the external debugger and the path to perform the fault injection, present only in the approach presented in [10], are bordered in red. In [9] there is only a signal to trigger the known non-solvable defect affecting a low-DAL task, managed by the HIL simulator.	204
5.7	The time domain performances characteristic of interest, for a 2nd order system, from the control theory literature.	205
5.8	The block diagram of the proposed HIL system.	208
5.9	The <i>pause/resume Finite State Machine (FSM)</i> described in the approach presented in [10].	209
5.10	The block diagram of a generic MAS. Adapted from [11].	217
5.11	Flowchart of the proposed approach. Figure from [11].	219
5.12	The mechanical structure of the benchmark MAS. Adapted from [11].	220
5.13	The control software block diagram.	221
5.14	MIL/SIL estimated path. Figure from [11].	224
5.15	Block diagram of the SW architecture adopted for HIL testing.	225
5.16	A photograph of the experimental setup described in this proposal.	226
5.17	HIL worst-case estimated path. Figure from [11].	227
5.18	Results obtained from 65 HIL simulations. Figure from [11].	227
6.1	The mapping of the three proposed approach on the ISO26262:2018 structure.	230
A.1	Common questions on autonomous vehicles development aspects.	240

A.2	Accuracy graph of the network in [116], trained with the CK+ database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	255
A.3	Loss graph of the network in [116], trained with the CK+ database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	255
A.4	Normalized confusion matrix of the network in [116], trained with the CK+ database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	256
A.5	Accuracy graph of the network in [116], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	257
A.6	Loss graph of the network in [116], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	257
A.7	Normalized confusion matrix of the network in [116], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	258
A.8	Normalized confusion matrix of the network in [140], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	259
A.9	Accuracy graph of the network in [116], trained with the <i>Ensemble 1</i> database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	260
A.10	Loss graph of the network in [116], trained with the <i>Ensemble 1</i> using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	260
A.11	Normalized confusion matrix of the network in [116], trained with the <i>Ensemble 1</i> database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].	261
A.12	Accuracy graph of the network proposed in [116], trained with the <i>database ensemble 3</i> with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].	264
A.13	Loss graph of the network proposed in [116], trained with the <i>database ensemble 3</i> with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].	265

A.14 Normalized confusion matrix of the network proposed in [116], trained with the <i>database ensemble 3</i> with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].	266
A.15 Accuracy graph of the network proposed in [116], trained with the <i>database ensemble 3</i> with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].	266
A.16 Loss graph of the network proposed in [116], trained with the <i>database ensemble 3</i> with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].	267
A.17 Normalized confusion matrix of the network proposed in [116], trained with the <i>database ensemble 3</i> with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].	267
A.18 Myself acting a neutral face as recognized by the neural network [116] trained on FER2013 [117] with FER+ annotations [118].Figure from [112].	273
A.19 Antonio Costantino Marceddu acting a neutral face as recognized by the neural network [116] trained on FER2013 [117] with FER+ annotations [118].Figure from [112].	273

Acronyms

A/IS	Autonomous and Intelligent System
ABS	Anti-lock braking system (AntiBlockierSystem)
AC	Alternate Current
ACC	Advanced Cruise Control
ACCS	Adaptive Cruise Control System
ADAS	Advanced Driver Assistance System
ADS	Automated Driving System
AEB/AEBS	Advanced Emergency Braking (System)
AIAG	Automotive Industry Action Group
ASIL	Automotive Safety Integrated Level
BCM	Body Control Module
BFIT	BestFIT
BOM	Bill Of Material
CAD	Computer Aided Design
CAN	Controller Area Network
CCRb	Car-to-Car Rear Braking (from EuroNCAP AEBS test protocol)
CCRm	Car-to-Car Rear Moving (from EuroNCAP AEBS test protocol)
CCRs	Car-to-Car Rear Stationary (from EuroNCAP AEBS test protocol)
CCS	Cruise Control System
CMP	Configuration Management Plan

Acronyms

DAL	Design Assurance Level
DC	Direct Current
DDT	Dynamic Driving Task
E-SM	Encoder Installed on the Steering Motor (SM)
E-SRG	Encoder installed on the Steering Reduction Gear (SRG)
E-TGR	Encoder installed on the Traction Reduction Gear (TRG)
E-TM	Encoder installed on the Traction Motor (TM)
ECR	The company Edge Case Research
ECU	Electronic Control Unit
eDB	external DeBugger
ELECTRIMACS	International conference of the IMACS TC1 Committee
ESP	Electronic Stability Program
ETFA	IEEE International Conference on Emerging Technologies and Factory Automation
FCW	Forward Collision Warning
FDIR	Fault Detection Isolation and Recovery
FI	Fault Injection
FM	Failure Mode
FMEA	Failure Mode Effects Analysis
FMECA	Failure Mode, Effects, and Criticality Analysis
FMEDA	Failure Mode, Effects, and Diagnostic Analysis
FMVSS	Federal Motor Vehicle Safety Standards
FPK	Forward Position Kinematic
FSC	Functional Safety Concept
FSM	Functional Safety Management
FSMS	Functional Safety Management System
FSR	Functional Safety Requirement

Acronyms

FuSa	Functional Safety
GPIO	General Purpose Input Output (port)
HAL	Hardware Abstraction Layer
HARA	Hazard Analysis and Risk Assessment
HAZOP	HAZard and OPerability analysis
HIL	Hardware-In-the-Loop
HMI	Human-Machine Interface
HSI	Hardware/Software Interaction
IEEE	Institute of Electrical and Electronics Engineers
IGBT	Insulated Gate Bipolar Transistor
IKPP	Inverse Kinematic Path Planner
IOLTS	IEEE International Symposium on On-Line Testing and Robust System Design
ISO	International Standard Organization
KPI	Key Performance Indicator
LATS	IEEE Latin-American Test Symposium
LPQF	Low Profile Quad Flat Package
MAS	Multi-Agent System
MBSD	Model-Based Software Design
MC/DC	Modified Conditions/Decision Coverage
MCS	Mixed-Criticality System
MIL	Model-In-the-Loop
ML	Machine Learning
MPSoC	Multiprocessor System on a Chip
NCAP	New Car Assessment Programme
NHTSA	National Highway Traffic Safety Administration
ODD	Operational Design Domains

Acronyms

OEDR	Object and Event Detection and Response
OTA	Over-The-Air
PEIC	Power Electronics Innovation Center (of Politecnico di Torino)
PID	Proportional-Integral-Derivative (control)
PIL	Processor-In-the-Loop
PM	Part Manufacturing
PSU	Power Supply Unit
RADAR	RAdio Detection And Ranging
RCP	Rapid Control Prototyping
RHF	Random Hardware Failure
RMS	Root Mean Square
SA/HI	Situation Analysis and Hazard Identification
SAE	Society of Automotive Engineers
SG	Safety Goal
SGR	Steering Reduction Gear
SIL	Safety Integrated Level
SIL	Software-In-the-Loop
SM	Steering Motor
SoC	System on a Chip
SOTIF	Safety Of The Intended Functionality
SPI	Safety Performance Indicator
SPICE	Simulation Program with Integrated Circuit Emphasis
SSUT	SubSystem Under Test
SUT	System Under Test
TC	Traction Clutch
TGR	Traction Reduction Gear

Acronyms

TM	Traction Motor
TSC	Technical Safety Concept
TSR	Technical Safety Requirement
TTC	Time To Collision
TV	Target Vehicle
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
V&V	Validation and Verification
VDA	Verband der Automobilindustrie (German Association of the Automotive Industry)
VUT	Vehicle Under Test
WCET	Worst-Case Execution Time
WP	Work products

Chapter 1

Introduction

Microcontrollers' adoption to control vehicle functions starts in the late '70s, pushed by the need to save fuel and early pollution control regulations for the automotive sector.

Early application of computer-based systems was for engine control purposes, particularly in the gasoline engine ignition system. These have been followed, in the '80s, by the early electronic fuel injection systems, automatic transmission control, and adaptive suspension, leading to integrated powertrain control.

In the '90s, with the increasing diffusion of computing systems of smaller dimensions and with more computational power per cost unit, more vehicle functions moved from mechanical/electromechanical implementations to a microcontroller, enabling novel functions to improve the safety, through airbags, anti-lock braking system (ABS), traction control, and electronic stability program. Other than these, comfort enhancements start to be deployed, like power windows, power seats, cruise control, automatic climate control, electronic cluster instrumentation, infotainment and navigation systems, and many other advanced functionalities.

The diffusion of Systems-on-Chip (SoC), which are entire computer systems in a single chip package, thanks to their versatility, computational power, and incredible flexibility, started the smartphone era in the mid-2000.

In the automotive industry they enabled more sophisticated functionalities into cars. Initially were introduced systems like parking sensors, cameras to depict the environment surrounding the vehicle and simplifying the maneuvers in small spaces, adaptive cruise control (ACC) followed, in the last ten years, by more complex Advanced Driver Assistance Systems (ADAS) based on RADARs, LiDAR, and computer vision, like advanced emergency braking (AEB), collision avoidance and mitigation, and lane assist systems.

In the last decade, we assisted into a trend of integration of subsystems to answer three contemporary challenges the automotive industry is facing:

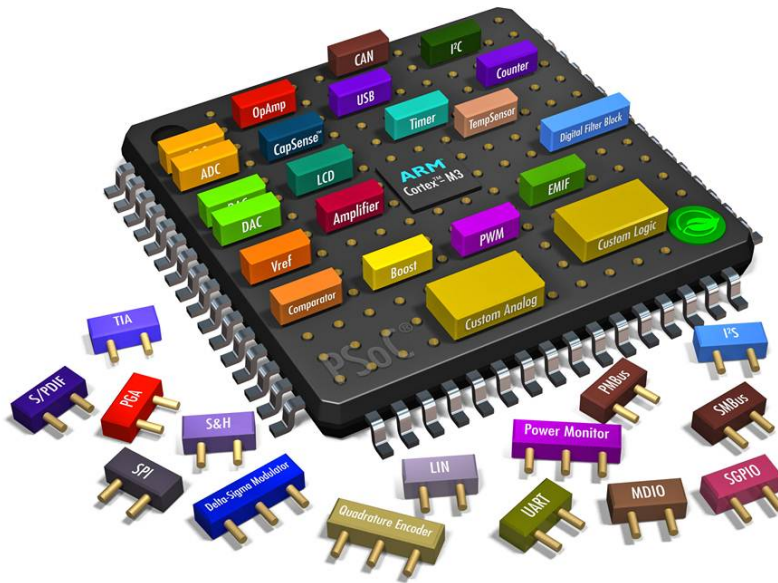


Figure 1.1: An artistic representation of a System on Chip (SoC) Cypress PSoC 5. Figure from Cypress Semiconductor.

- *electrification*: the trend to integrate electric propulsion systems, as the unique motor or to improve the internal combustion engine efficiency;
- *autonomous driving*;
- *connectivity*: to make vehicle able to communicate with other vehicles, with the road infrastructure, and with the cloud to stream music content, update the navigation system with traffic conditions and, more importantly, allow for updating the embedded software, especially for those software components implemented through machine learning approach.

1.0.1 Software: an opportunity and an issue

The usage of computer systems on-board the vehicles offer an enormous variety of benefits. However, there is also a dark side: by substituting physical processes or mechanical links with software-programmed ones, it is possible to introduce reliability problems that can cause misbehaviors. These can become an issue in those cases where they can lead to huge loss of money (mission-critical) or to accidents, affecting people's health or the environment (safety-critical).

To avoid that, a new idea of safety, from a functionality perspective, called *Functional Safety*, shall be adopted. This idea is not a novelty of the automotive

industry. However, it originates from the needs of nuclear, defense, and aerospace industries that emerged in the '60 when computer systems start to be adopted.

1.0.2 The software complexity in the automotive industry

A paper, released from a premium car manufacturer, [1] reports that their cars embedded software is composed of about 100 million high-level language programming statements, of which about 10 million are conditional ones. These are distributed over about three million functions called in 30 million points. Just to allow the reader to figure out the complexity of its architecture, this software is distributed on more than 120 Electronic Control Units (ECU) and manages about 7000 external signals.

Even if part of this complexity can originate itself by the software architecture (like reuse of legacy code and third-part components), it is impossible to reduce it significantly.

To solve the reliability problems originated from such complexity, *quality*¹ of the software shall be achieved by application of strict development policies to guarantee a low function nesting level and, by adopting guidelines as the MISRA one, uniformity and readability of the source code and avoidance of undefined behavior caused by the usage of implementation-specific instructions. Static analysis tools can enforce all these requirements.

Moreover, AUTOSAR has been another step to *simplify* the complexity, at least in the higher software abstraction level [2] by standardization of the interfaces between SW components.

As another solution, adopting the Model-Based Software Design approach allowed to accelerate the software development keeping the same level of quality in terms of the absence of defects. Moreover, it allows the use of the model itself as the code documentation, obtaining platform-independent software unit implementation and simplifying testing activities.

1.1 Industry trends on automation of vehicles safety and driving functions

In the last years, it is possible to observe a consolidation of the *race* to autonomous driving. From the business point of view, the main aspects characterizing this period are:

¹In this case, *quality* indicates the absence of systematic errors leading to behaviors different from the ones required by the specifications. These are also known as defects or, more colloquially, as *bugs*.

- huge investments for the production of newer models;
- car manufacturers tend to teaming to share technologies and divide the huge required investments;
- the small manufacturers fail or be acquired by a bigger one.

1.1.1 Commercial applications

Privately owned passenger cars

Privately owned passenger cars with some level of driving automation have been available since some years. These cars are based on a driver-monitored approach and can be classified as Level 2 of the SAE classification, described in section 1.2.1, or as having a *Supervised* operating mode as described in the ECR classification explained in section 1.2.2.

These vehicles' safety approach is based on the human driver, responsible for the safety and continuously monitoring the vehicle's automation, always keeping his/her eyes on the road, ready to intervene.

In literature are reported:

- Multiple fatal crashes (fig.1.2) due to excessive driver complacency on the automatic driving system and, sometimes, a time lower to 10 seconds between the regular situation and the fatal crash;
- Tempe Arizona fatality in 2018, caused by, as stated in the preliminary report [3] released by the National Transportation Safety Board (NTSB) by a wrong human-machine interface and violation of the rules of the road by the pedestrian, the only fatality of the accident, who crossed the road in an unauthorized zone walking a bicycle. All these conditions created a situation non forecasted during the autonomous driving software development that had difficulties recognizing the obstacle as a pedestrian walking a bicycle. Moreover, in this specific implementation, the vehicle operator's intervention disables the automatic emergency braking system of the car. The human operator, at this point, was unable to brake in time to avoid the collision.

Considering vehicles at least at level 4 of the SAE (see section 1.2.1), or *automated* in the ECR (see section 1.2.2) classification, where the driving safety is in charge of the vehicle itself, we have these applications: low-speed shuttles, parcel delivery, and fleet of vehicles (robotaxis).



Figure 1.2: A crash (one fatality) involving a Tesla car occurred on March 2018 in California. Image from ABC News.

Low-speed shuttles

These public transportation vehicles can take up to 15 passengers on a fixed route at low speed (below 30 km/h), and are under demonstration in various cities worldwide. These shuttles can be classified as Level 5 of the SAE classification (see section 1.2.1) or having a *Automated* operating mode in the ECR one (see section 1.2.2) due to their need of a crew member on-board to manage the non-driving safety aspects.

Parcel delivery

Some lightweight (like tiny cars) vehicles are now under demonstration to transport parcels from stores to houses. These vehicles are designed for short-range delivery inside cities. They can travel, based on the local rules and the chosen size, on roads, sidewalks, bike lanes.

Their safety is based on their low speed and weight (hence low kinetic energy) and on the presence of remote human operators ready to intervene in case a vehicle asks for help. These special-purpose vehicles cannot be classified in the SAE classification (see section 1.2.1) and have a *Automated* operating mode in the ECR one (see section 1.2.2) since they need a remote human pilot in case of necessity.

Some incidents have been reported, like the presence of a sidewalk bot blocking a wheelchair ramp (Pittsburg, 2019) and tensions of pedestrians over the use of sidewalk space.

Fleet of vehicles

Waymo (see fig.1.3) and Amazon Zoox (see fig.1.4) deployed some robotaxis on a limited scale, while middle-mile trucks gained interest in 2020. Many players are pushing into this area.

These robotaxis can be classified as Level 5 in the SAE classification (see section 1.2.1) and have a *Autonomous* operating mode in the ECR one (see section 1.2.2) since they do not need a crew member on-board.

In early times the safety approach of these vehicles was based on the presence on-board of a human safety driver. However, the current trend is to guarantee operators' availability whenever a car asks for help.

In California, USA, have been reported minor incidents during testing.



Figure 1.3: A Waymo robotaxi.



Figure 1.4: A Amazon Zoox robotaxi.

1.2 Autonomous vehicles classification

1.2.1 SAE J3016

The document SAE J3016 *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems* has been initially released in January 2014 and has been updated twice, respectively, in September 2016 and June 2018.

It classifies different levels of automation by keeping into account the extension (in terms of limited/unlimited) of the *Operational Design Domains* (ODD), in where the vehicle is capable of performing the *Dynamic Driving Task* (DDT) autonomously, and on the fallback mechanisms (between driver, fallback ready user, system) chosen to guarantee the DDT safety (*DDT fallback*).

The DDT is split into two subcategories: *Sustained lateral and longitudinal vehicle*

control and Object and Event Detection and Response (OEDR).
The SAE J3016 classification table is shown in fig.1.5.

Summary of Levels of Driving Automation for On-Road Vehicles

This table summarizes SAE International's levels of *driving* automation for on-road vehicles. Information Report J3016 provides full definitions for these levels and for the italicized terms used therein. The levels are descriptive rather than normative and technical rather than legal. Elements indicate minimum rather than maximum capabilities for each level. "System" refers to the driver assistance system, combination of driver assistance systems, or *automated driving system*, as appropriate.

The table also shows how SAE's levels definitively correspond to those developed by the Germany Federal Highway Research Institute (BAST) and approximately correspond to those described by the US National Highway Traffic Safety Administration (NHTSA) in its "Preliminary Statement of Policy Concerning Automated Vehicles" of May 30, 2013.

Level	Name	Narrative definition	Execution of steering and acceleration/deceleration	Monitoring of driving environment	Fallback performance of dynamic driving task	System capability (driving modes)	ES&S level	NHTSA level
Human driver monitors the driving environment								
0	No Automation	the full-time performance by the <i>human driver</i> of all aspects of the <i>dynamic driving task</i> , even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a	Driver only	0
1	Driver Assistance	the <i>driving mode</i> -specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	Human driver and system	Human driver	Human driver	Some driving modes	Assisted	1
2	Partial Automation	the <i>driving mode</i> -specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	System	Human driver	Human driver	Some driving modes	Partially automated	2
Automated driving system ("system") monitors the driving environment								
3	Conditional Automation	the <i>driving mode</i> -specific performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> with the expectation that the <i>human driver</i> will respond appropriately to a <i>request to intervene</i>	System	System	Human driver	Some driving modes	Highly automated	3
4	High Automation	the <i>driving mode</i> -specific performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> , even if a <i>human driver</i> does not respond appropriately to a <i>request to intervene</i>	System	System	System	Some driving modes	Fully automated	3/4
5	Full Automation	the full-time performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> under all roadway and environmental conditions that can be managed by a <i>human driver</i>	System	System	System	All driving modes		













Figure 1.5: SAE J3016 Autonomous vehicles classification.

1.2.2 ECR classification: a responsibilities-based point of view

An alternative classification, proposed by the Edge Case Research in January 2021 [4], is based on a user-centric approach. Four different operating modes (corresponding almost to the levels 2, 3, 4, and 5 of the SAE J3016 classification) are defined and, for each one of them, it is indicated who is responsible (between the driver or the car) of the vehicle functions² (*driving*, *driving safety*, and *other safety*). A summary of these operating modes is shown in fig.1.6.

²*Driving* is the dynamic driving function, *driving safety* is the monitoring of the automation systems, while *other safety* is all the other aspects (like verification of properly door closing and that the seat belts are buckled).

With respect to the SAE one, the advantage of this classification is that the driver’s responsibilities are explicitly defined in each operating mode.

Operating Mode	Human Role	Driving	Driving Safety	Other Safety
Assistive	Driving			
Supervised	Eyes ON the road			
Automated	Eyes OFF the road			
Autonomous	No human driver			

Vehicle Automation Modes

EDGE CASE RESEARCH
www.ecr.ai

Figure 1.6: Operating modes classification proposed by Edge Case Research.

Assistive

In the assistive mode, the driver has to actively drive the car since only Advanced Driver Assistance Systems (ADAS) are available. Examples of ADAS are Adaptive Cruise Control, Advanced Emergency Braking System, Lane Keeping Assist System, etc.

Supervised

In the supervised mode, the vehicle does all the dynamic driving tasks, but the human driver is expected to pay continuous attention to the road and to intervene if required by safety whether or not notified by the driving system that there is a problem.

Examples of supervised driving systems commercially available are Tesla’s Autopilot and General Motor’s Super Cruise.

Automated

In the automated mode, the driver can take his/her eyes out of the road since the car is both in charge of driving and ensuring driving safety. In this case, the human driver cannot be blamed in case of a crash since safety is a responsibility of the driving system.

Other safety operations different from the driving, like verify that the passengers have their seat belts buckled, children are properly seated, manage the vehicle's evacuation in case of fire, are still in charge of an adult person on board the vehicle. If an automated vehicle is used for a public transportation task, a crew member must be on board at any time during the vehicle operation.

Autonomous

In the autonomous mode, there is no human on-site or even continuously monitoring of what is going on during the vehicle operation: in this case, the vehicle itself is responsible for the driving and all the other safety aspects.

1.3 Functional Safety

Even if the proposals described in this dissertation consider only the functional safety (FuSa) aspects considered by the ISO26262, it is useful to describe how these interact with all the other safety aspects.

Moreover, two conceptual instruments are described: the FIDES guide, to evaluate the probability of failure for the hardware components the item is composed of, and an FMEA manual [5], to assess the impact of the possible failures the designed item can embed or avoid the presence of defects it can embed.

1.3.1 Overview and position of Functional Safety in the autonomous driving framework

Achieving Functional Safety is not a stand-alone process, but shall be integrated alongside (autonomous) system safety, vehicle safety, cyber-security, dynamic driving functions intended functionalities, and quality management. The fig.1.7 shows how these aspects interact to each other.

The involved standards are:

- System Safety
 - UL 4600
Standard for Evaluation of Autonomous Products, released in April 2020.
- Vehicle Safety

- FMVSS
National Highway Traffic Safety Administration (NHTSA) issues Federal Motor Vehicle Safety Standards (FMVSS) to implement United States Congress laws.
- NCAP
Rules released by the independent agencies of the New Car Assessment Program.
- Cyber-security
 - SAE J3061
Cybersecurity Guidebook for Cyber-Physical Vehicle Systems, released in January 2016.
 - ISO/SAE DIS 21434
Road Vehicles - Cybersecurity Engineering, released in February 2020.
- Dynamic Driving Function
 - ISO/PAS 21488
Road vehicles — Safety of the intended functionality, released in January 2019.
 - ISO/TR 4804
Road vehicles — Safety and cybersecurity for automated driving systems — Design, verification and validation, released in December 2020.
- Functional Safety
 - ISO 26262
Road vehicles — Functional safety, released in November 2011, lastly updated in December 2018.

Functional Safety (FuSa), which has a central role in this dissertation, considers the absence of unreasonable vehicle operation risks. Moreover, the ISO 26262 considers only the risks associated with possible malfunctions of Electrical and Electronical (E/E) components embedded into road vehicles. In other words, the safety of the nominal functionality is not in the scope of FuSa, which places itself alongside the safety of the nominal behaviors expected from the dynamic driving function (addressed by UL 4600 for *automated* or *autonomous* vehicles and by ISO/PAS 21488 SOTIF for ADAS). The end goal to achieve FuSa is to design a system able to react on time and in the proper way to the external world in every situation, also in case of failures affecting its components.

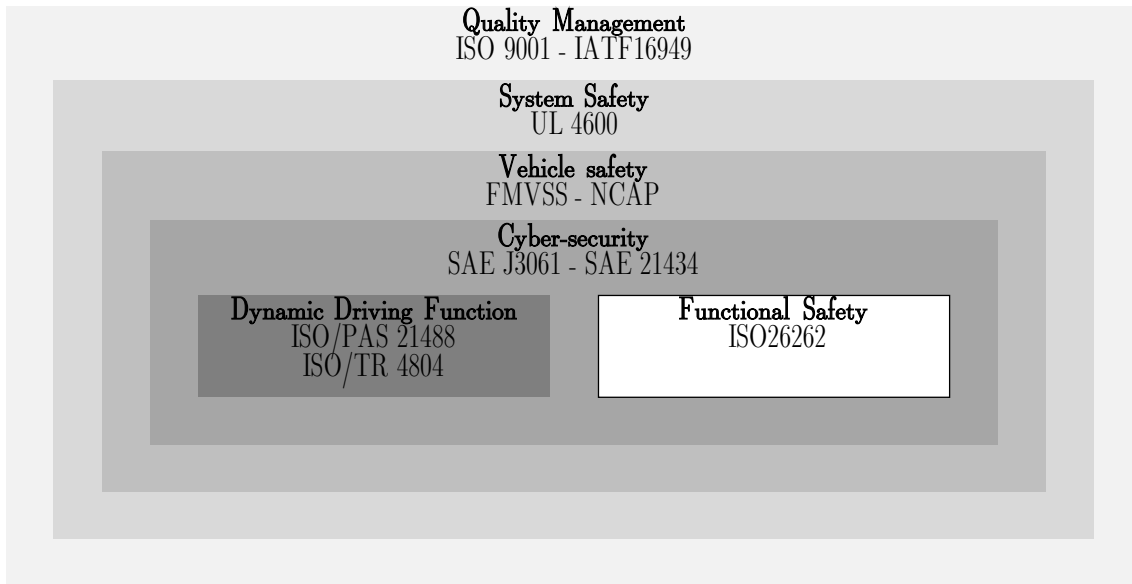


Figure 1.7: Relationship between the standards involved in the development of autonomous driving systems

Mitigation of deliberate tamperings of the system and errors in the application scope hypothesis are not topics of the FuSa, that addresses preventable misuse and the absence of software/hardware defects or random hardware failures that can lead to safety violations.

In any case, also the topics excluded by FuSa have to be taken into consideration, inside the adopted quality management (QM) framework, during all the conceptualization, design, production, maintenance, and decommissioning of safety-relevant systems.

Evolution of the approach to achieve Functional Safety

This section is about the evolution of the process to achieve functional safety. The discussion starts from the ISO26262 approach, released in 2011, targeting FuSa of non-autonomous vehicle. After that, it is considered the one from the ISO/PAS 21488, released in January 2019 to keep into account the safety of the intended functionality (SOTIF) of ADAS and vehicles classified at least as L2 in the SAE (see section 1.2.1) or *supervised* in the ECR (see section 1.2.2) and the newer one, released in April 2020 in the UL 4600, designed for vehicles up to L5 of SAE or *Autonomous* in the ECR one.

The main difference between the ISO26262 concept and the newer ones is that the first considers deterministic software, where each software unit behaves in a deterministic manner. In contrast, the last two addresses software developed through

machine learning approach, in where a complete description of the possible scenarios in the real world is not possible. To better explain the differences in the HARA processes of these standards, figures 1.8, 1.9, and 1.10 show their interaction with design, testing, and deployment. These figures have been adapted from [6].

ISO 26262 The ISO26262 [7] is a standard dedicated to achieving FuSa in the automotive industry.

It was published in its first edition in 2011, then updated in 2018.

Due to its central role in this dissertation, it is described in chapter 2. Section 2.1 describes the *Safety Lifecycle*, section 2.2 details the *Concept Phase*, section 2.3 is about the *Functional Safety Concept* (FSC), section 2.5 tell of the *Technical Safety Concept* (TSC), section 2.4 describes the *development at the system level*, section 2.7 explains how the design of safe hardware, while section 2.11 discuss the development of safe software.

The approach to achieve FuSa proposed by ISO26262 is a traditional one. It is based on a Hazard Analysis and Risk Assessment (HARA), similarly to the approach presented in the IEC 61508.

It bases the required rigor for subsequential activities on an Automotive Safety Integrated Level (ASIL). The assigned level represents the associated risk level and, subsequentially, the safety integrity needed to perform functionalities of the considered item.

The safety analysis is performed as described in the following.

The hazards list is obtained from the engineers' knowledge, literature, or similar items.

After the hazard list has been obtained, a risk assessment process is performed (based again on the engineers' knowledge) to assess the risk level for each one of the hazards in terms of three risk parameters: *severity*, *exposure*, and *controllability*. From the list of hazards, Safety Goals (SGs) have to be derived. Each of these SGs is associated with an ASIL level depending on the risk level associated with the hazards to which the driver or the people surrounding the vehicle can be exposed if the SG is violated.

As the *controllability* risk parameter is taken into account, it is simple to understand how the presence of a human driver is crucial in the risk assessment, making the ISO26262 not suitable for autonomous driving vehicles (the only solution is to associate the most restrictive controllability level), since there is no driver at all.

As shown in fig.1.8, the only feedback to the design phase is the HARA. After that, no other feedbacks are expected. It can be suitable for normal vehicle functions, like ABS, Cruise Control, airbags, where the environment is wholly known a priori and the software developed to behave in a deterministic way. Hence, it does not need any update to expect some newer calibrations or functionalities.

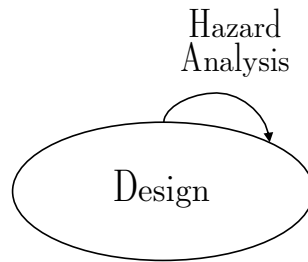


Figure 1.8: The HARA process inside the ISO26262 framework. Figure adapted from [6].

ISO/PAS 21488 The approach proposed by the ISO 26262 is insufficient not only when there is no driver but also when the environment is not completely known a priori. In those cases, the software is designed relying on machine learning approaches. Since it is not possible to keep into account all the possible cases it is possible to encounter in the real world after the deployment, it is necessary to extend the FuSa approach by also keeping into account the so-called Safety of the Intended Functionality (SOTIF).

Three different moments are considered: *design*, *testing*, and *deployment*.

The *design* starts from the results of the HARA phase, as in the ISO26262. However, the testing has a different role concerning the ISO26262, where it is all about proving absence of SGs violations. ISO/PAS 21488, with its concept of *safety of the intended functionality* (SOTIF), requires to question the nominal functionality as well, as shown in fig.1.9.

When a disagreement w.r.t. the safe behavior is found, modifying the software to solve the found issues is necessary. The software update has to be done continuously during the vehicle lifecycle, hence to deploy them, an automatic Over The Air (OTA) updating system is needed. It poses novel challenges on cybersecurity aspects since the involved software is safety-relevant.

In the SOTIF framework, errors are defined as violations of Key Performance Indicators (KPIs).

The KPI-based approach is advantageous when non-deterministic software units, like those based on machine learning approaches, are adopted. An example can be the Lane Assist, which is based on computer vision.

This is sufficient since the driver is still responsible for the vehicle's safety, and he/she cannot rely upon the assistance system.

UL 4600 The approach followed by ISO26262 and ISO/PAS 21488 is not sufficient for those cases in where the vehicle has to drive by itself, without any human intervention.

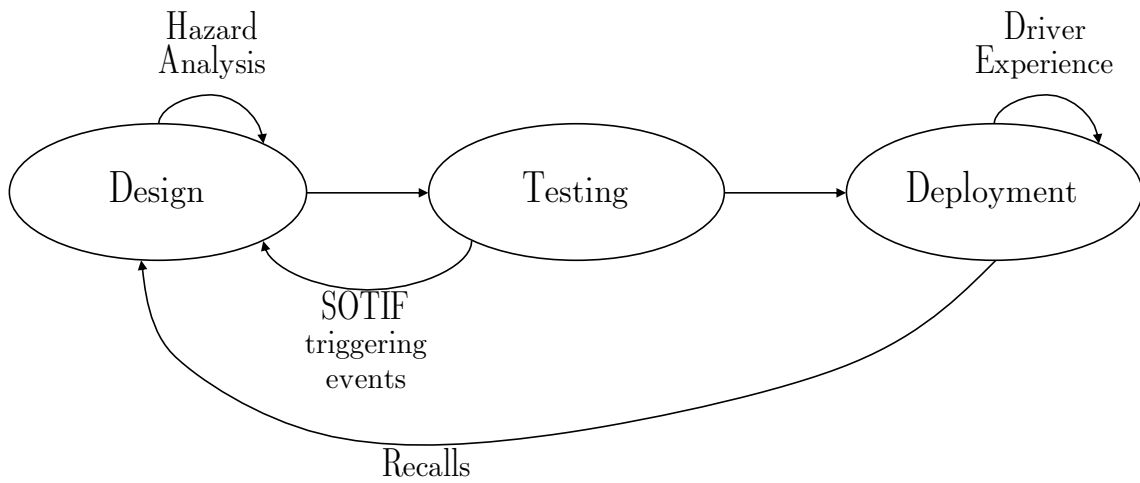


Figure 1.9: The HARA process inside the ISO/PAS 21488 (SOTIF) framework. Figure adapted from [6].

In this case, a new approach, based on *safety case(s)* proposed in the UL 4600, has to be adopted.

In this framework, as shown in fig.1.10, the design, testing, and deployment phases are linked to a Safety Case.

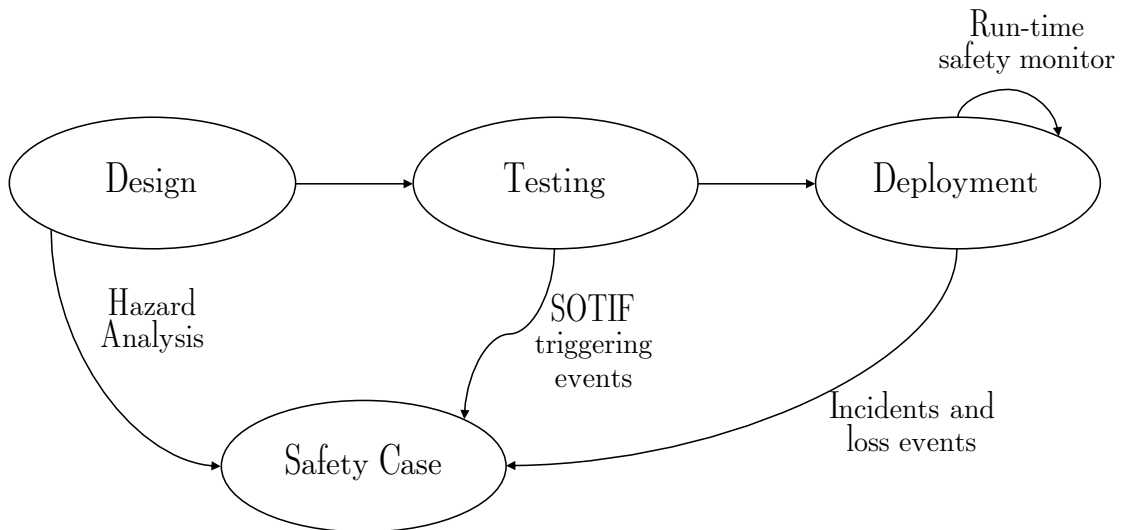


Figure 1.10: The HARA process inside the UL 4600 framework. Figure adapted from [6].

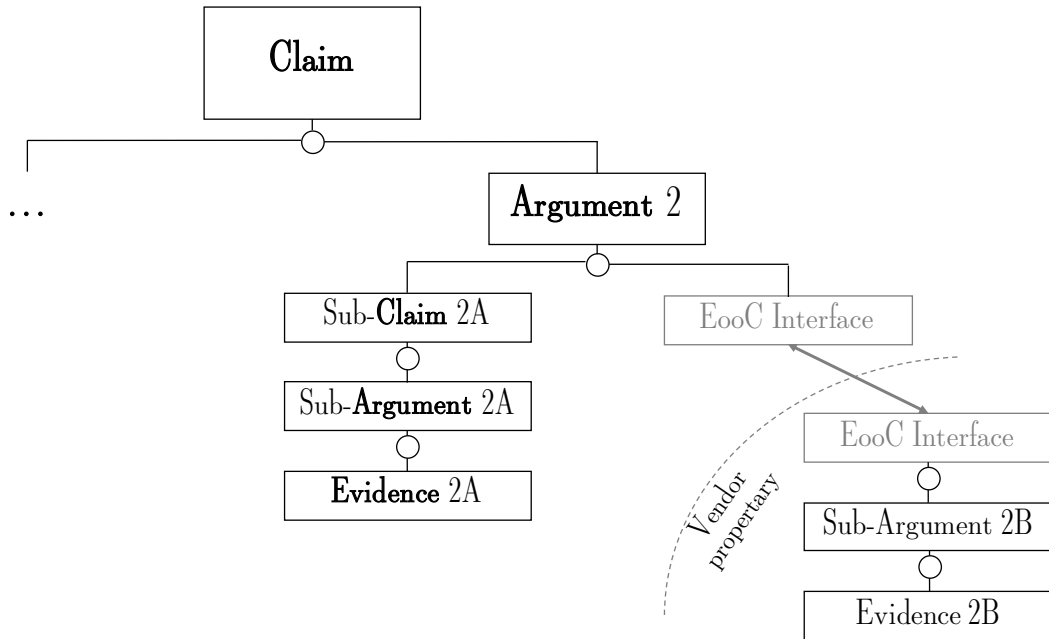


Figure 1.11: The structure of a UL 4600 Safety Case, with an EooC. Figure adapted from [6].

The *safety case*, as shown in fig. 1.11, is a gerarchical structure composed of *Claims*, *Arguments* and *Evidences*.

Each *argument* of a *claim* can be split in *sub-claims* when needed.

Claims (or *sub-claims*) are properties of the system, for example, *keep a sufficient safety distance from the preceding vehicle*.

Arguments are descriptions of why the *claim* is valid, for example, *Detect preceding vehicle(s), measure the distance from it, and act properly on brakes when needed*.

The *Evidence* is a list of supporting arguments, like tests, analysis, simulation, etc, to demonstrate that the considered *claim* is properly implemented.

A concept similar to the Safety Element out of Context (SEooC) (see section 2.2.3) of the ISO 26262 is contained into UL 4600, under the name of Element out of Context (EooC).

It can be applied to the various class of components, like hardware components, software units, sensor, trained neural networks, etc.

EooC is shipped with a safety case fragment. Its vendor does not need to expose the entire safety case but, similarly to the *safety manual* of the ISO26262, it has to provide an interface containing: properties and characteristics, assumptions that the system must complain, fault model(s) used for the assessment, a partial UL 4600 clause coverage, and an assessment report.

The metric to obtain feedbacks on implementations is the Safety Performance Indicators (SPI). SPIs have to be associated to *claims* or *sub-claims* to measure

their validity.

A *claim* can be perfect (required 100% of SPI success), or can be defined as a target percentage.

During tests, an SPI value violation means the safety claim is invalid; hence they monitor the validity of the safety case.

Root causes of safety claims violation can be:

- execution defects into the design process;
- design defects;
- hazards non considered or improperly managed;
- SOTIF analysis gaps;
- presence of bias in training data (for machine learning approaches);
- evidence gaps or defects;
- errors in the assumption.

KPIs and SPIs

The differences between Key Performance Indicators (KPIs) and Safety Performance Indicators (SPIs) are summarized in fig. 1.12.

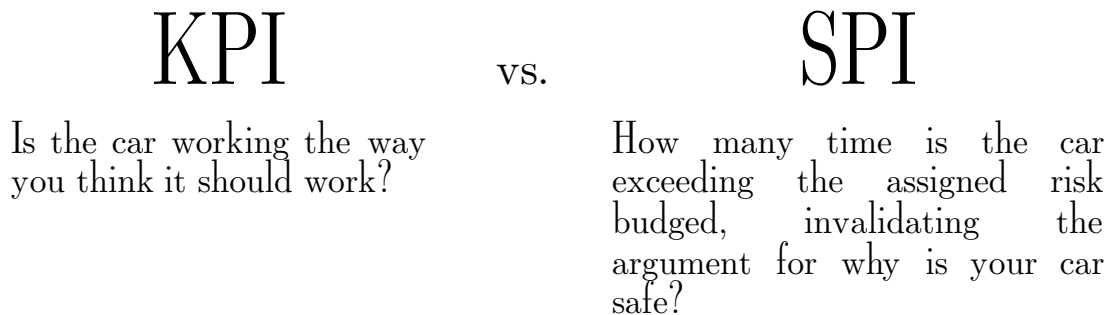


Figure 1.12: Summary of the differences between KPIs and SPIs.

The KPIs are about the accuracy and precision of the *claim* implementation, expressed in statistical terms. For sensors, KPIs can be the detection rate, the signal-to-noise ratio (SNR), expressed in terms of average and variance.

The SPIs, instead, are more focused on safety of the functionality expressed by the

considered *claim*. For example, an SPI can be the number of consecutive frames containing false negatives, like an undetected obstacle on the road, or the number of times the car violates the minimum clearance distance to an obstacle. In these terms, SPIs can also consider concurrent multi-sensor detection failures or losses of calibration.

1.3.2 FIDES

FIDES is a guide, released in 2004 and updated in 2009 by various companies from the aerospace industry, to assess the probability of random hardware failures of the *components*³ included in the considered hardware design. This guide is described in section 2.8.

1.3.3 AIAG&VDA FMEA manual

This manual [5], jointly released by AIAG and VDA in June 2019, describes how to perform the Failure Mode and Effect Analysis (FMEA) for automotive applications.

This process, done during the *concept phase* (see 2.2) of the ISO 26262, has not to be confused with the Failure Mode, Effects, and Diagnostic Analysis (FMEDA) (see section 2.7), performed to assess the reliability of the hardware design, as described in section 2.10.

1.4 Technical contributions on this dissertation

This dissertation proposes various technical contributions on functional safety aspect. These are needed to increase the dependability of microcontroller-based electric and electronics components in charge of performing safety-critical tasks.

The central idea of all these contributions is the simulation. These can be summarized into three different topics (see fig.1.13⁴):

- simulation-based FMEDA, with six proposals;
- simulation-based HARA, with a unique proposal;
- real-time software validation, with three proposals.

³The term *component* is used here with the definition indicated in the ISO 26262 standard. It is possible to find a vocabulary to avoid confusion between ISO 26262 and FIDES in section 2.8.2.

⁴The figure shows the phases of the ISO26262 *safety lifecycle* where the proposals can be applied. A detailed description of the Standard can be found at 2.

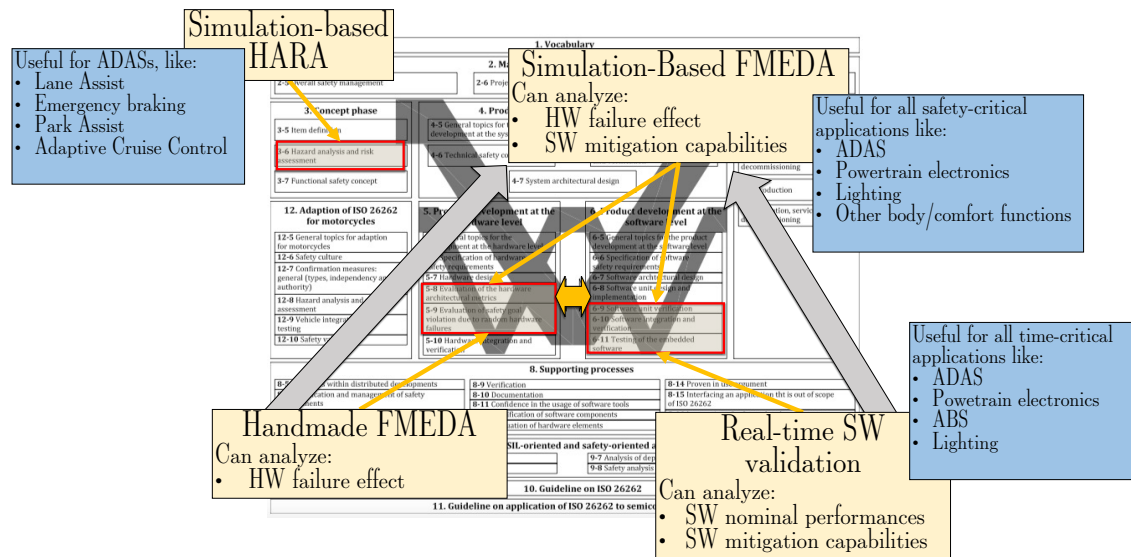


Figure 1.13: Topics discussed in this dissertation, with common application on automotive items and their position into the ISO26262 *safety lifecycle*.

1.4.1 Proposals that contribute to improving the FMEDA

The FMEDA is described, by the ISO26262 part 6 (HW development), as a technique to verify if a hardware design reaches the required reliability metrics (reported in tab.2.7), based on the ASILs (see section 2.2) of the functions it is in charge of.

The contributions presented on this topic are mainly applied to automotive industry case studies. Notable exceptions are the two last proposals, described in sections 3.9 and 3.10, which are about, respectively, to an industrial and a mobile robotic application.

FMEDA is mandatory from the ISO26262 to all the safety-relevant electronics, but industries also apply it to not safety-relevant comfort/body functions of cars to improve the availability of these systems and, consequently, customer satisfaction.

1.4.2 Proposals that contribute to improving the HARA

The hazard analysis and risk assessment is a crucial phase of the ISO26262 safety lifecycle. It is performed during the concept phase (described in part 3 of the Standard). It aims to define the safety goals (see section 2.2) for the item under development and associate an ASIL to each of them. This level is proportional to the maximum risk level the people inside or surrounding the vehicle are exposed to if the safety goal is violated.

The development of this approach started from the methodologies proposed to perform the simulation-based FMEDA. These are described in the section 3.6 of

this dissertation.

They have been adapted to HARA. This analysis is performed during the concept phase so before the item's schematics have been designed. The lack of schematics makes it necessary to obtain models based only on the vehicle's behavior. Moreover, the failure modes are described only in terms of erroneous behaviors of the affected item.

Thanks to these models, and a vehicle-level simulator, it is possible to assess how the effects of the wrong behaviors of the item can affect the entire vehicle and hence driving safety. The ISO26262 requires performing the assessment considering three different risk parameters: *severity*, *controllability*, and *exposure*. Their combination, by means of the determination matrix shown in fig.2.5, leads to an ASIL level associated with the considered hazard. The proposed approach allows determining, thanks to the simulation results, two of them: *severity* and *controllability*.

The main application of these techniques in the automotive industry is to test and validated ADASs.

1.4.3 Proposals that contribute on improving real-time software validation

Real-time software validation is crucial for safety-critical embedded systems with hard real-time requirements.

These tests aim to verify that the embedded software, considered as the application components integrated with the device drivers and other supporting components, can run inside the allotted time slots without deadline misses and provide the right results.

The core of this approach is the real-time simulator, composed of a software infrastructure that can manage the communications between a host computer, providing the HMI, and a real-time computer, that is a special-purpose device in charge to run the plant model in a real-time manner and to provide the plant responses to the device under test.

Real-time software validation has been widely applied to closed-loop control systems in almost all industrial sectors. For this scope, on the market are available plenty of off-the-shelves tools. In the literature, such tests are called as Hardware-In-the-Loop (HIL), as described in the section 2.11.5 of this dissertation. It is evident that for these kinds of applications, there is no room for improvements.

But, following the current market trends, it is possible to find out that, for at least three different applications domains, there are no proposals to perform HIL testing on them:

- automotive body control modules [8], described in section 5.1;
- mixed-criticality avionic systems [9] [10], described in section 5.2;

- multi-agent robotic systems [11], described in section 5.3.

The contributions on mixed-criticality and multi-agent robotic systems may seem unrelated to the automotive industry, but interest in these applications is gaining.

Real-time software validation is mandatory and well discussed in the literature for almost all safety-relevant electronics. Due to the difficulties of finding actual industry applications using these techniques, we decided to demonstrate the approach on benchmark from other market fields. However, mixed-criticality systems allow reducing the number of ECUs inside cars (for cost reduction), while the multi-agent strategy is helpful to improve the cooperation of the connected cars with respect to other vehicles and the infrastructure.

1.5 Structure of this dissertation

The rest of the dissertation is composed as follows.

Chapter 2 describes how items have to be designed inside the ISO26262 framework.

Chapter 3 describes the achieved results on the topic of simulation-based FMEDA.

Chapter 4 describes the achieved results on the topic of simulation-based HARA.

Chapter 5 describes the achieved results on the topic of real-time SW Validation techniques.

Chapter 6 draws the conclusion of this dissertation for the various topics.

Finally, Appendix A describes the achieved results on the topic of Machine Learning (ML) Applications. Due to the choice not to include ML on the main track of the thesis, this chapter also contains its concluding paragraph.

Chapter 2

Items development inside the ISO26262 framework

This chapter describes the state of the art of functional safety in the automotive industry. It is composed as follows.

Section 2.1 explains the ISO26262 *safety lifecycle*, providing information about the structure of the Standard (see section 2.1.1), *functional safety management* (FSM) (see section 2.1.2) and *confirmation measures* (see section 2.1.3).

Section 2.2 describes the *concept phase* (part 3 of the ISO26262) explaining its first two main phases *item definition* and *hazard analysis and risk assessment*. The third and last main phase, about the writing of the *Functional safety concept*, is described in section 2.3.

The section 2.4 explains the workflow for the development at the system level (part 4 of the ISO26262), while section 2.5 describes how to prepare the *Technical Safety Concept*.

Section 2.6 explains the *Failure Mode and Effect Analysis* conducted at the system level, at the same time of the hazard analysis and risk assessment.

Section 2.10 clarifies the differences between the *Failure Mode and Effect Analysis* and the *Failure Mode, Effects, and Diagnostic Analysis* performed during the hardware design phase, described more in details into the sections 2.7 and 2.8 about the *product development at the hardware level* (part 5 of the ISO26262). Section 2.9 describes *fault injection*.

Section 2.11 describes the *product development at the software level* (part 6 of the ISO26262).

2.1 Safety Lifecycle

2.1.1 Structure of the ISO26262

The ISO26262 is structured as shown in fig. 2.1.

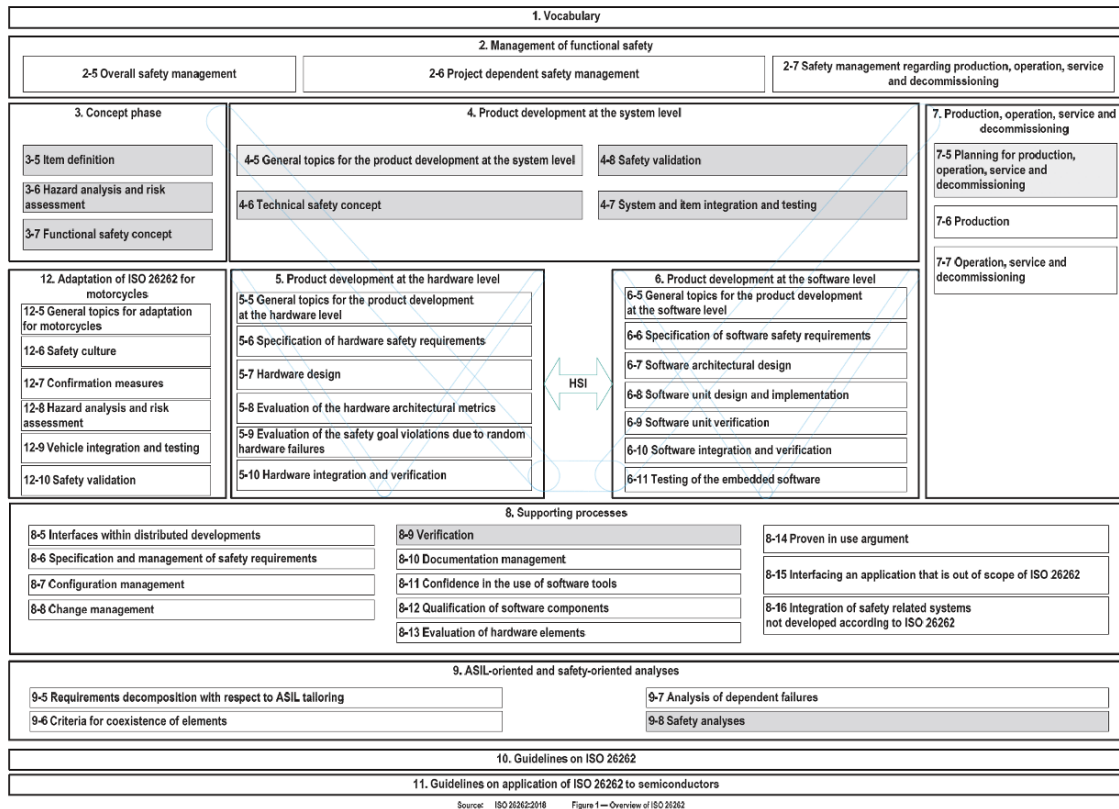


Figure 2.1: The ISO26262:2018 structure

2.1.2 Functional Safety Management

Functional Safety Management (FSM) is the core of the ISO26262 since it is needed to implement the safety lifecycle. It is not a standalone process to be added inside the workflow of a company, but it can be considered as an add-on to the adopted Quality Management (QM)¹ system.

In particular, the availability of a well used QM process based on ISO 9001 and IATF 16949 is a precondition and shall be used in combination with ISO 26262,

¹Most adopted standards on QM are: ISO9001, QM9000, SPICE, CMMI, and IATF16949.

ISO/PAS 21448, ISO/SAE AWI 21434, and UL 4600, depending on the level of automation required by the vehicle. Hence, FSM is the enhancement of the quality requirements needed to consider the Functional Safety aspects.

The FSM can be conceptually split into two sections: a project independent and a project-specific one. About the project independent requirements, we have to consider:

- education and qualification measures with the aspects of functional safety (safety culture);
- how to establish a company-specific safety lifecycle including all the needed supporting processes;
- establish an additional process of continuous improvement on the functional safety;
- the need to implement the ISO9001 and the IATF 16949 before the FSM.

Work products for project-independent FSM

Main WPs to manage the project-independent FSM aspects are:

- organization-specific rules and processes for FuSa;
- evidence to demonstrate the management of employees' competence level;
- evidence to demonstrate the QM system implemented inside the company;
- rules to report safety anomaly when identified.

A detailed description of these documents exulate form the purposes of this dissertation.

Activities for project-specific FSM

For each project the company intends to start, it is required to prepare a safety plan.

The safety plan describes all the activities needed to achieve the FuSa.

Each one of these activities has to be described through:

- the objective of the activity;
- necessary results of other activities such as input document for the phase;
- the list of people responsible for the safety activities;
- date and duration of the safety activities;

- the required resources;
- the interfaces to the suppliers;
- the identification of the corresponding WP documentation.

Other than that, the safety plan has to describe the verification and validation activities² or refer to the Validation and Verification (V&V) plan, specifying:

- verification methods chosen for each safety requirement (in terms of review, analysis, simulation, and test);
- the corresponding verification steps shall be defined for every development phase;
- policies to guarantee that the fulfillment of the requirements is validated by using a test;
- document the test results.

Project-Specific Safety Management Roles

For each project, these roles are required:

- *Safety Manager*. He/she is responsible for the FSM. In particular, he/she prepares and maintains the safety plan and monitors that the activities are in line with it. Moreover, he/she has to plan the safety activities in the development subphases of the safety life cycle, monitor the progress of safety activities, maintain records, assign the safe activities for the team, and manage the deliverables of the safety activities.
- *Project Manager*: He/she appoints to the *Safety Manager* and is responsible for ensuring that the safety activities are performed and verifies that the compliance with the ISO26262 is achieved.
Another task is to allocate, during the project, sufficient resources for the safety-related activities.
- *Safety Team*: the team responsible for performing the safety-related activities.
- *Safety engineer*: a member of the *Safety Team*.

²ISO26262:2018 part 8 clause 10

2.1.3 Confirmation measures

The conformation measures have to be performed keeping into account *degree of independence requirements*. The notations are defined as follows [7]:

- -: No requirements and no recommendations for or against regarding this confirmation measure.
- **I0**: The confirmation measure *should* be performed. However, if it is performed, different people shall perform it in relation to the people responsible for creating the considered work product.
- **I1**: The confirmation measure *shall* be performed by different people in relation to the people responsible for creating the considered work product.
- **I2**: The confirmation measure *shall* be performed by people who are independent of the team that is responsible for the creation of the considered work product (for example, people not reporting to the same direct superior).
- **I3**: The confirmation measure *shall* be performed by a team that is independent regarding management resources and release authority from the department responsible for creating the considered work products.

In tables 2.1 and 2.2 are listed the required verification measure for each one of the Work Product expected from the safety lifecycle.

Verification review subject	Applies to						
	QM	ASIL A	ASIL B	ASIL C	ASIL D		
Hazard analysis and risk assessment of the item			X*				
Functional Safety Concept	o		X				
Technical Safety Concept	o		X				
System architectural design	o		X				
System	o		X				
Item and system integration	o		X				
Integration and verification specification	o		X				
Safety validation specification	o		X				
Hardware safety requirement	o		X				
Hardware design	o		X				

Table 2.1: Overview of verification measure. From [7].

o No requirement and no recommendation for or against
X required

X* Scope of this review also includes hazardous events rated as QM.

Verification review subject	Applies to			
	QM	ASIL A	ASIL B	ASIL C ASIL D
Evaluation of the effectiveness of the architecture of the item to cope with the random hardware failures	o	o	+	++
Evaluation of the safety goal violation due to random hardware failures	o	o	+	++
Software safety requirements and the refined hardware-software interface requirements	o			++
System architectural design	o			++
Software units	o			++
Software integration	o			++
Software component qualification	o			++
Evaluation of hardware elements	o			++
Dependent failure analysis	o			++
Safety analyses	o			++

Table 2.2: Overview of verification measure. From [7].
o No requirement and no recommendation for or against
+ Recommended
++ Required

2.2 Concept Phase

The *concept phase* addresses possible hazards caused by misbehaviors of safety-related E/E systems, keeping into account also the interactions between these systems.

This phase has a Functional Safety scope; hence hazards related to electric shock, fire, smoke, heat, radiation, toxicity, flammability, corrosion, and similar, are not to be considered unless directly caused by malfunctioning behaviors of the considered E/E system, in the following referred as the *item*.

The main phases (and work products to be prepared) during the *concept phase* are:

- Item Definition
- Hazard Analysis
- Functional Safety Concept (FSC)

This phase's primary goals are: to determine the *Hazards*, the *Safety Goals* (SG) associating to each one of them an *Automotive Safety Integrated Level* (ASIL). Moreover, it is necessary to determine a list of *Functional Safety Requirements* (FSR) for our item, and finally to provide a *draft block diagram* indicating the responsibilities, in terms of SG and ASILs, for each one of the blocks of the designed item.

In the rest of the dissertation, the *Hazard Analysis* phase will be called *Hazard Analysis and Risk Assessment* (HARA).

Fig.2.2 illustrates the hierarchical approach by which the SGs are determined as the result of HARA. The Functional Safety Requirements are then derived from the SGs allocated to the System Architectural Design (described in part 4 of the standard).

2.2.1 Item definition

In the *definition*, the item (or the system) under safety analysis is described. The aim of this phase is defined in ISO26262:2018 part 3 clause 5. It has to provide sufficient information about the item, its functionalities, dependencies, and interaction with other items/systems to support an adequate understanding, enough to perform the HARA phase and write the FSC.

2.2.2 Hazard Analysis and Risk Assessment

The HARA has to be performed as described in ISO26262:2018 part 3 clause 6. It aims to identify and classify the hazardous events caused by the item's malfunctioning behaviors and establish their risk potential to formulate the Safety

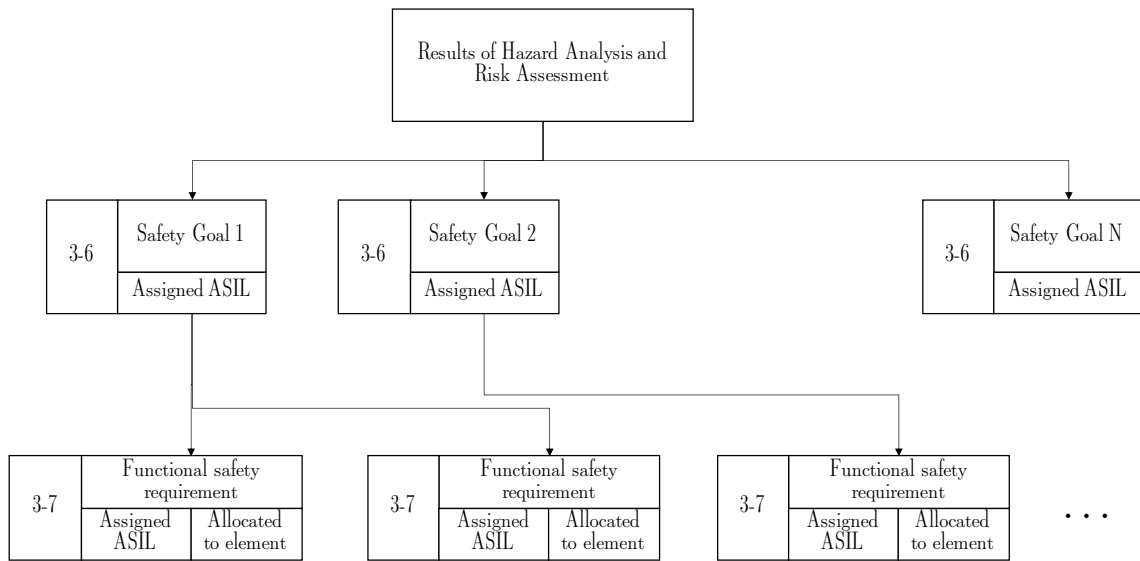


Figure 2.2: Hierarchy of Safety Goals and Functional Safety Requirements. Figure adapted from [7].

Goals with their corresponding ASILs related to preventing or mitigating hazardous events, avoiding unreasonable risks.

HARA can be split into 5 phases, as shown in fig.2.3.

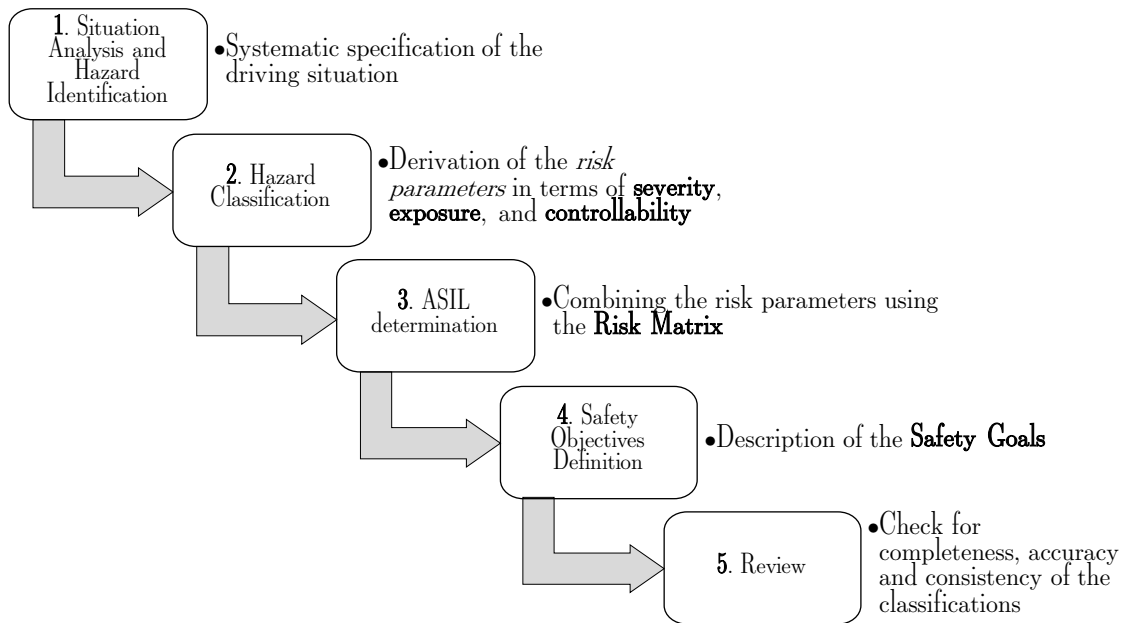


Figure 2.3: HARA subdivision in phases.

Situation Analysis and Hazard Identification

Situation Analysis and Hazard Identification (SI/HI), together with the *Hazard Classification* (see 2.2.2), is performed with the end goal to determine the *risk parameters* (*severity, exposure, and controllability*) for each one of the *hazards* found.

It is performed by a systematic specification of the *driving situations*, followed by an inductive analysis to determine all the hazards caused by the possible predictable misbehavior of the considered item.

To complete the analysis, designers shall provide a list of the failures and hazards considered out of the scope of the SI/HI.

HAZard and Operability analysis (HAZOP) HAZOP is applied to determine potential malfunctions and potential vehicle-level hazards.

This method has been developed in heavy industry to get hazards from functions related to processes (chemical, industrial, etc.) for which the keywords were made. It is widely adopted in the automotive, with modifications of the keywords to adapt them to vehicle functions.

A possible set of guidewords to evaluate malfunctions can be [12]:

- NO OR NOT: the negation of the design intent, function not provided.
- MORE THAN INTENDED: quantitative increase with respect to the design/set intent.
- LESS THAN INTENDED: quantitative decrease with respect to the design/set intent.
- NOT REQUESTED: functionality is provided when not required.
- INCORRECT DIRECTION: opposite functionality is provided.
- EARLY: relative to the expected deadline.
- LATE: relative to the expected deadline.
- LOCKED: functionality is provided even after that the request is removed.

Scenarios To obtain a list of *scenarios*, a set of *Operational Situations* and *Operating Modes* have determined.

Designers can obtain *Operational Situations* and *Operating Modes* by analyzing:

- Local variability: road type, traffic, and environmental characteristics;
- Drivers and vehicle variables: speed, maneuver, driver conditions, etc.

and then, combining them by changing *variable parameters*:

- Location: parking area, intersection, city, country road, highway;
- Speed: low, medium, high, very high (> speed limits in highways);
- Traffic conditions: the presence of pedestrians, other vehicles preceding, oncoming, or following.
- Maneuvers: light/heavy braking, emergency stop, light/heavy acceleration, overtaking, cornering, turning, starting to move, driving straight.
- Driver on board or off board.

The cartesian combinations of these *Operational Situations* and *Operating Modes* with the *variable parameters* lead to an enormous number (thousands) of possible *Scenarios*. Still, some of them are incoherent (like starting to move/emergency braking, braking/acceleration, high speed/driver off board, etc.) or forbidden by traffic laws (like intersection/overtaking).

It would seem that the more detailed the list of *Scenarios*, the more you will be able to provide a better description. However, unfortunately, designers must take into account that, by fragmenting them excessively, it would be challenging to evaluate the correct exposure, incurring the so-called *exposure trick*.

To avoid this, it is better to provide only evaluations of meaningful *Scenarios* where a possible hazard can impact. The hazards have to be considered one at a time, and a rationale shall be provided to describe the qualitative analysis of all variables.

Hazard Classification

The *Hazard Classification* derives the *risk parameters* in terms of *severity*, *exposure*, and *controllability*.

To determine the parameters, similar public functional safety analysis and other literature material shall be considered, adapting them, thanks to the analysis performers' knowledge, to the specific case under analysis.

The standard itself provides a list of examples.

Severity To determine the severity risk parameter, it is possible to use the definition reported in table 2.3.

Controllability To determine the controllability risk parameter, it is possible to use the definition reported in table 2.4.

Exposure There are two ways of determining the probability, as shown in fig. 2.4. Hence, the standard contains two different tables, one considering probability classes of exposure in time, the other the probability of exposure inside a specific operational situation.

Class	S0	S1	S2	S3
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival uncertain) or fatal injuries	Life-threatening injuries (survival uncertain) or fatal injuries
Reference for single injuries (from AIS scale)	AIS 0 and less than 10% probability of IAS 1-6	More than 10% probability of AIS 1-6	More than 10% probability of AIS 3-6	More than 10% probability of AIS 5-6

Table 2.3: ISO 26262-3 Table B1: Examples of severity classification.

Class	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable
Driving factors		More than 99 % of the average drivers or other traffic participants are able to avoid harm	Between 90 % and 99% of the average drivers or other traffic participants are able to avoid harm	Less than 90 % of the average drivers or other traffic participants are able to avoid harm

Table 2.4: Adapted from ISO 26262-3 Table 6 Class of controllability. The row *Driving factors* is extracted from table B.6.

To better explain the difference, in the table 2.5 are reported the definitions in terms of *Duration* and *frequency* from tables B.2 and B.3 of part 3 of the ISO26262.

ASIL determination

The technical risk reduction measures to achieve an acceptable residual risk are classified into four classes called Automotive Safety Integrity Level (ASIL): A, B, C, D.

A critical remark is that the ASIL always refers to a specific safety goal (and safety requirement).

The level is obtained by combining the three risk parameters using the risk matrix shown in fig.2.5.

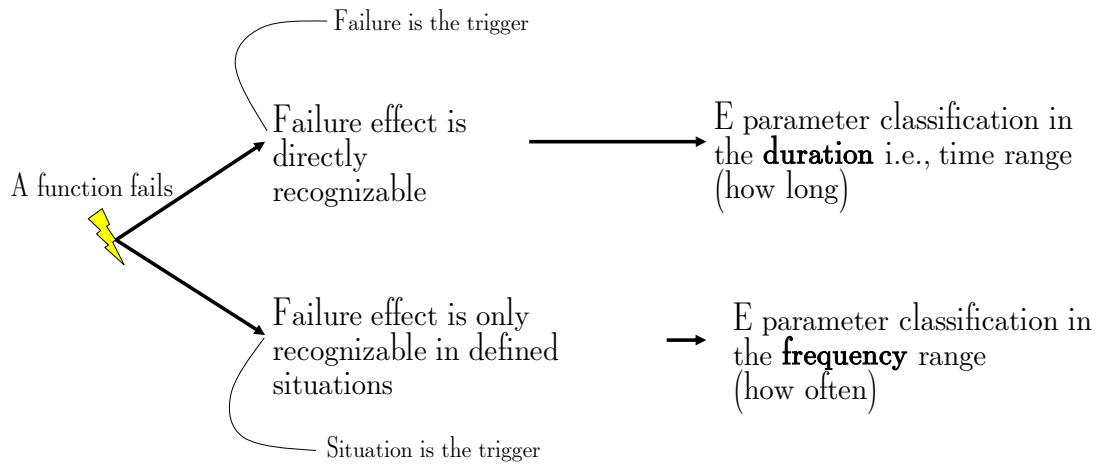


Figure 2.4: Exposure classification in duration or frequency.

Class	E1	E2	E3	E4
Description	Very low probability	Low probability	Medium probability	High probability
Duration (% of average operating time)	Not specified	≤ 1% of average operating time	1% to 10% of average operating time	≥ 10% of average operating time
Frequency of situation	Occurs less often than once a year for the great majority of drivers	Occurs a few times a year for the great majority of drivers	Occurs once a month or more often for an average driver	Occurs during almost every drive on average

Table 2.5: Adapted from ISO 26262-3 Table 6 Class of controllability.

The row *Duration* is extracted from table ISO26262:2018-3 B.2

The row *Frequency of situation* is extracted from table ISO26262:2018-3 B.3.

MSIL for motorvehicles

The second revision of the ISO26262 standard defines, for motorcycles, a qualitative way to assess the *technical risk reduction measures* to achieve an acceptable residual risk subdivided into 4 classes. These are called *Motocycle Safety Integrity Level* (MSIL).

MSIL can be: A, B, C, D where MSIL D is comparable with ASIL C.

Like ASILs, also MSILs always refers to a specific safety requirement (and safety goal). The mapping between MSIL and ASIL is shown in table 2.6.

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

Figure 2.5: ASIL determination matrix.

MSIL	ASIL
QM	QM
A	QM
B	A
C	B
D	C

Table 2.6: Mapping of MSIL to ASIL.

Safety goals definition

Once all the hazards have been assessed, assigning them an ASIL level, it is necessary to define the *item's safety goals* (SG).

These are statements of what shall not happen, for example, *the item does not apply unintended acceleration to the vehicle*.

To each one of the defined safety goals it has to be assigned an ASIL (the ASIL is assigned to the safety goals and not to the item). This ASIL shall correspond to the higher one given to the hazards to which the people inside or surrounding the vehicle are exposed if the considered safety goal is violated.

Independent review

The HARA phase has to be checked for completeness, accuracy, and consistency of the classifications.

This review, as required by table D.1 of the ISO26262:2018 part 2 clause 2, shall be performed, independently from the ASIL determined, by an independent person, regarding management, resources, and release authority, from the department responsible for the HARA phase itself (I3 level as described in 2.1.3).

2.2.3 Safety Element out of Context

Until now has been considered an item/system to be integrated into the context of a particular vehicle. This assumption is not always valid, since to realize a safety-related item, Commercial Off-the-Shelf (COTS) components can be adopted, or the designed item can be a COTS by itself.

In these cases, we can consider them as *Safety Element out of Context* (SEooC). A SEooC is a safety-related element not developed for a specific application (intended as functionality/item/vehicle). Examples can be microcontroller (elaboration units not designed in the context of a particular item) or entire systems, like transaxles and instrument clusters (not designed in the context of a specific vehicle).

In other words, a SEooC is designed under the assumption that it is intended to be used in multiple different applications, as long as the designers that are integrating it can establish the validity of its design hypothesis during the integration of the SEooC.

In developing a SEooC, its ASIL capabilities are determined by considering its compliance with the assumed safety requirements assigned with a given ASIL. Therefore, the SEooC must be designed and produced following all the requirements specified by the ISO26262 standard for the assigned ASIL.

The hypothesis must be made regarding the intended functionality and use context, including the expected external interfaces. These shall address a superset of items so that the SEooC can be used later in multiple different, still similar, applications. The validity of the assumptions and the compatibility of the external interfaces will be established in the actual item/system/vehicle context while integrating the SEooC. The relationship between assumptions and SEooC is shown in fig.2.6.

2.3 Functional safety concept

The *functional safety concept* (FSC) describes how to achieve the safety goals.

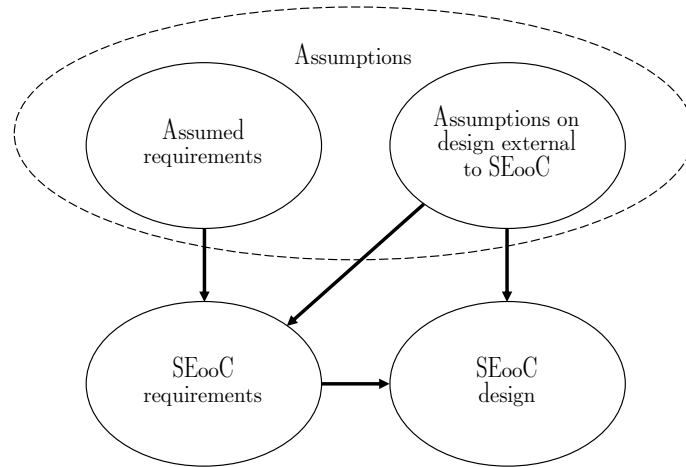


Figure 2.6: Relationship between design assumptions and SEooC development. Figure adapted from [7].

The FSC contains conceptual descriptions of the functional interactions required to achieve the safety goals. These are written in the form of *Functional Safety Requirements* (FSR).

FSC goal is done, from a practical point of view, to derive a set of FSRs and allocate them to a (preliminary) architecture of the item or external measures.

It is convenient to provide also a state chart describing the various states in where the item can found itself. These states should be classified as:

- nominal operating modes;
- emergency operating modes;
- degraded operating modes;
- safe states.

A good description includes explanations about the transitions between the states.

2.3.1 Functional parameters

The functional parameters to be specified are:

- Operating conditions;
- Fault tolerance times and intervals;
- Transitions to safe states or degraded/emergency operation modes;
- Functional redundancies.

Warning and degradation concepts

Safety engineers can define a functionality as fail-safe or fail-operational. In a fail-safe system, the safe state consists of disabling the functionality, while in a fail-operational one, it continues to provide the functionality with reduced (degraded) performances.

The process to provide a degraded performance without losing the functionality is called *graceful degradation*.

Emergency running operation modes When by design, the item has to reach a safe state for a specific failure, it is necessary to specify how to reach and maintain it (when operations are needed).

Degraded operation modes When by design, for a specific failure, the item has to react in a fail-operational manner, it is necessary to specify how to reach the degraded state itself, its limitation with respect to the nominal operation state, and how to maintain it (when operations to maintain the degraded state are needed). Moreover, it is necessary to describe how to reach the safe state if another failure occurs, preventing the degraded operation mode from continuing.

Driver's actions that contribute to achieving the safety goal

If some driver's operations are expected, these shall be described in the FSC. Moreover, it is necessary to provide evidence about the fact that the average driver reacts in this way, the effects in case he/she not react at all or reacts in an improper way, and eventually how to train drivers to respond appropriately.

2.3.2 Safety architecture

The FSC has to specify, through a block diagram representation, the functional redundancies and the individual functional block's independence level.

For each of these blocks, it has to be specified its ASIL requirements and, when required, the chosen ASIL decomposition.

2.3.3 ASIL decomposition

The ASIL decomposition is the distribution of the ASIL requirements over several elements of the item. It requires sufficient independence of the decomposed element. The rules for the decomposition are provided by the standard as shown in [fig.2.7](#).

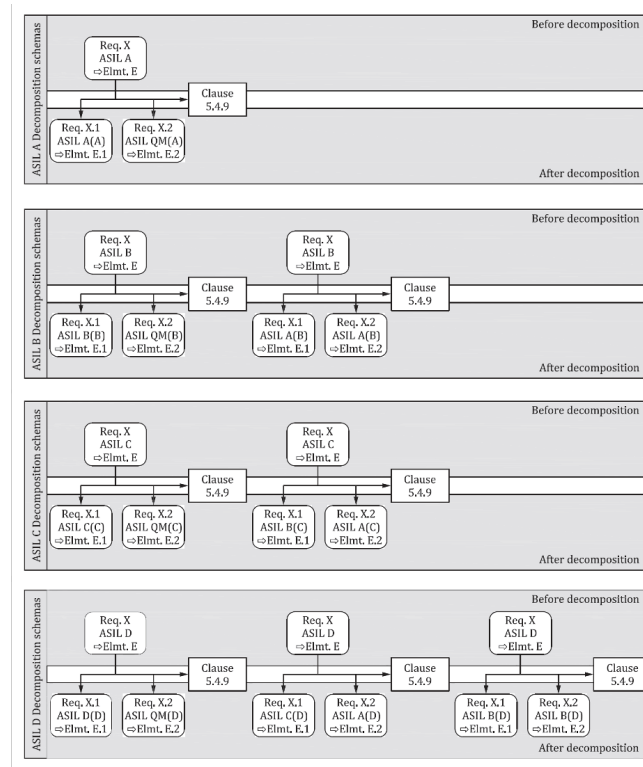


Figure 2.7: ISO26262-9 Figure 2 Classification scheme of ASILs when decomposing safety requirements. Figure adapted from [7].

2.3.4 Functional Safety Requirements

The objective to keep in mind while describing the FSRs is to maximize the vehicle’s availability, ensuring, at the same time, the safety of its operation. The particular critical points for this purpose are selecting the safe states based on the identified potential failure modes and specifying how and when trigger transitions between states, degraded operating modes, and safe states have to be performed.

Failure Classification

A *failure* can be classified (see fig.2.8) as *systematic* or *random (hardware)*. More in detail:

- *Systematic failures* are those, existing in the manufactured product, related in a deterministic way to a certain cause, that can only be eliminated by a change of design, manufacturing process, operational procedures, documentation, or

other relevant factors.

Software defects (see section 2.11.1) can raise only systematic failure.

- *Random failures* are those that did not exist in the manufactured product and can occur unpredictably during the lifetime of a hardware element following a probability distribution (see section 2.8.4).

These failures can be *permanent* if the affected component is damaged, so it is impossible to restore its characteristics, or *transient*, when the affected component restarts to work correctly after a certain amount of time. Soft errors of memories are examples of transient faults.

The FuSa approaches to *systematic* and *random* failures are different.

For the first group, the goal is prevention, thanks to a rigorous application of the safety plan activities to avoid defects. For the latter, the goal is to implement safe reactions when a random fault occurs.

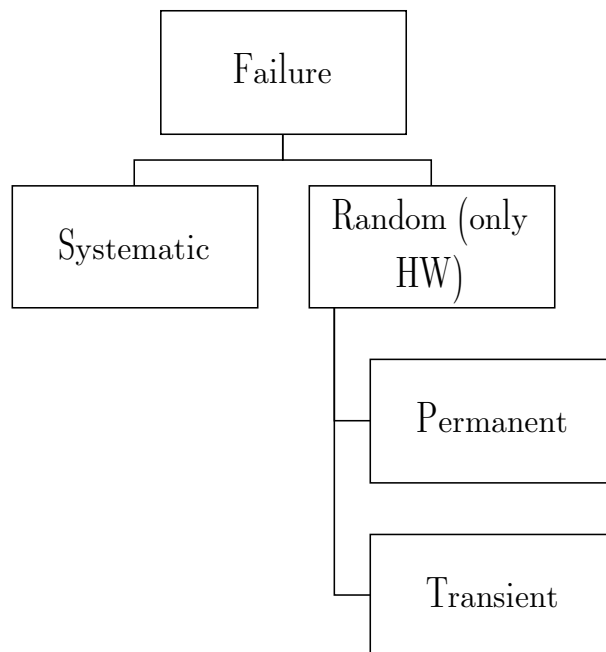


Figure 2.8: Failure classification between *systematic* and *random (hardware)*.

Moreover, as shown in fig.2.9 two or more failures can be *Independent* or *Dependent*.

The *independent* ones are those whose probability of simultaneous or successive occurrence $P_{A|B}$ can be expressed as the simple product of their unconditional probabilities P_A and P_B , hence $P_{A|B} = P_A \cdot P_B$.

On the other hand, the *dependent* one are those that are not statistically independent, i.e., the probability of the combined occurrence $P_{A|B}$ of the failures is not equal to the product of the probabilities of occurrence of all considered independent failures P_A and P_B , hence $P_{A|B} \neq P_A \cdot P_B$.

The *dependent* failures can depend on each other in two different ways. If a failure of an element of an item resulting from a root cause, inside or outside of the element, and then it causes a failure of another element or elements of the same or different item, the failures have a *cascading* relations. As opposite, they can share a *common cause*, when the failure of two or more elements of an item resulting directly from a single specific event or root cause is either internal or external to all of these elements.

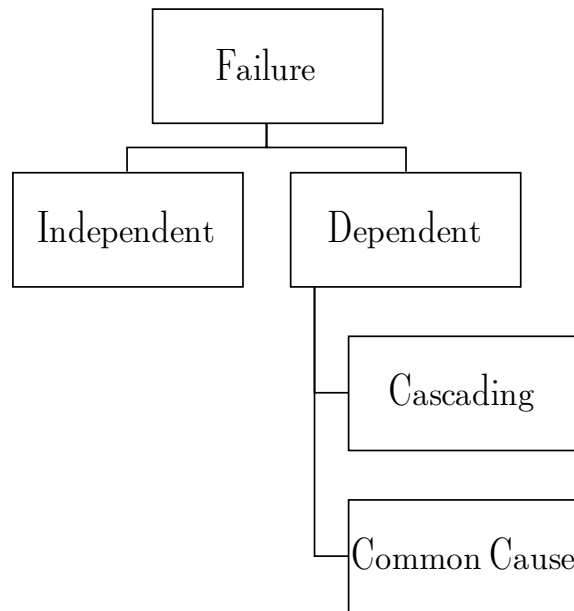


Figure 2.9: Failure classification between *independent* and *dependent*.

While cascade failure analysis can only be conducted by going into the merits of the specific implementation chosen for the item (both in hardware and software terms), the presence of common cause failures can be avoided by conducting dependent failure analysis, as shown in fig.2.10.

Timing aspects of the Functional Safety

To achieve functional safety, the designed item has to react appropriately (by avoiding errors, as described in section 2.3.4), and on time: we have to consider the timing aspects needed to deal appropriately with the system we want to control.

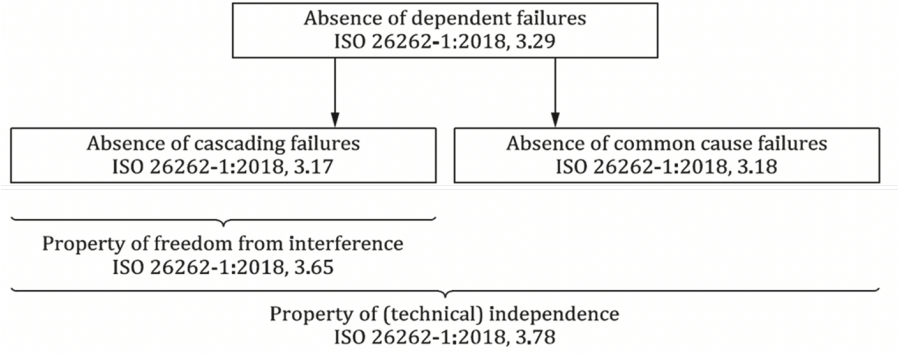


Figure 2.10: Dependent failure analysis. Figure from [7].

The important time intervals from the safety point of view are:

- *Fault detection Time Interval* (FDTI): Maximum time-span from the occurrence of a fault to the detection of a fault.
- *Diagnostic Test Time Interval* (DTTI): the amount of time between the executions of online diagnostic tests by a safety mechanism (incl. duration of the execution of an online diagnostic test).
- *Fault Reaction Time Interval* (FRTI): time-span from the detection of a fault to reaching a safe state or to reaching emergency operation.
- *Fault Handling Time Interval* (FHTI): $FHTI = FDTI + FRTI$.
- *Fault Time Tolerance Interval* (FTTI) [7]: time-span in which a fault or faults can be present in a system before a hazardous event occurs.
- *Fault Reaction Time Interval* (FRTI) [7]: the time span from the detection of a fault to reaching the safe state . Injection time t_i : the time (from the start of the simulation) in seconds when the considered fault has been injected.
- *Detection time* t_d : the time (from the start of the simulation) in seconds when the system has detected the considered fault.
- *Diagnostic Time Interval* t_{dti} : the difference between the detection time and the injection time, hence $t_{dd} = t_d - t_i$.
- *Mitigation time* t_m : the time when the mitigation algorithm has brought the system to a safe state.
- *Mitigation delay* t_{md} : the difference between the time when the mitigation algorithm has been put the system in a safe state and when the fault has been injected, hence $t_{md} = t_m - t_i$. It is needed that $t_{md} \leq FTTI$.

A graphical representation of these time intervals is shown in fig.2.11.

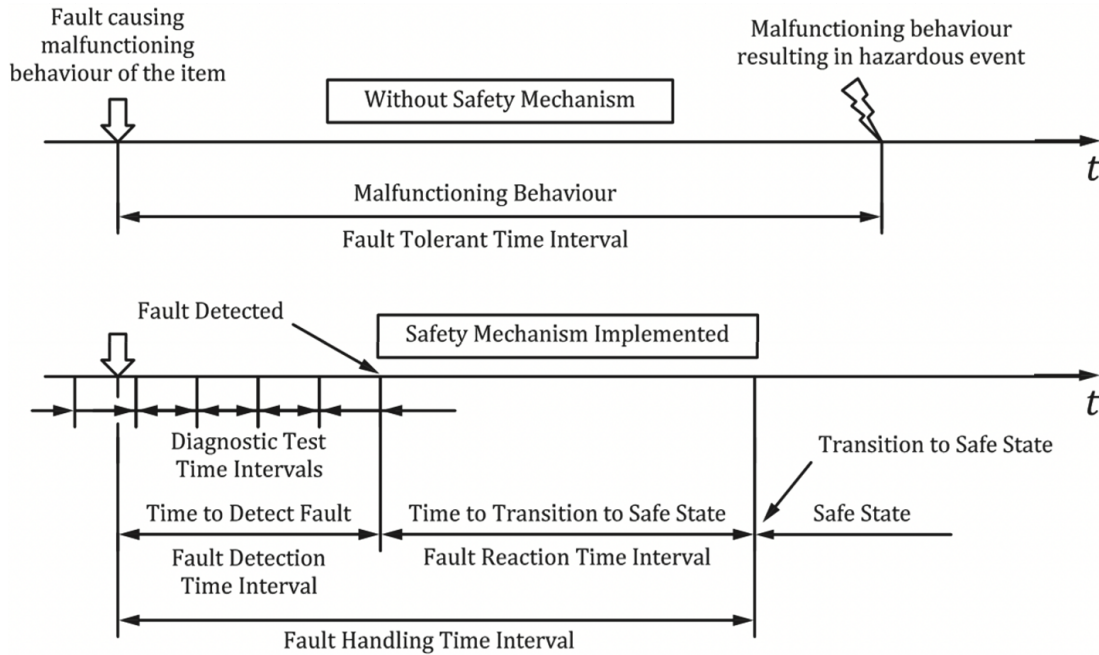


Figure 2.11: Time intervals relevant for safety. Figure from [7].

2.4 Implementation phases

Once the *Technical Safety concept* (see section 2.5) has been prepared and verified through the confirmation measures, it is possible to start the product implementation phases.

The product implementation is managed by parts 4 (system design), 5 (hardware design), and 6 (software development) of the ISO26262 standard. As shown in fig.2.1 the hardware and software development proceed in parallel, except for the Hardware/Software Interaction (HSI) analysis, which is the interface between the hardware and software world and has to be performed during the system-level analysis by both hardware design and software development team.

HSI is the most critical point regarding item reliability, especially when random hardware failure detection and mitigation are implemented in software.

Moreover, also the hardware-implemented protection mechanisms (like overcurrent protection in motor control) can expect some software action after their intervention.

A summary of the Concept Phase (see section 2.2) documents, their interaction to each other, with the Functional Safety Concept (FSC) (see section 2.3), the Technical Safety Concept (TSC) (see section 2.5) and the development at the system, hardware, and software level are shown in fig.2.12.

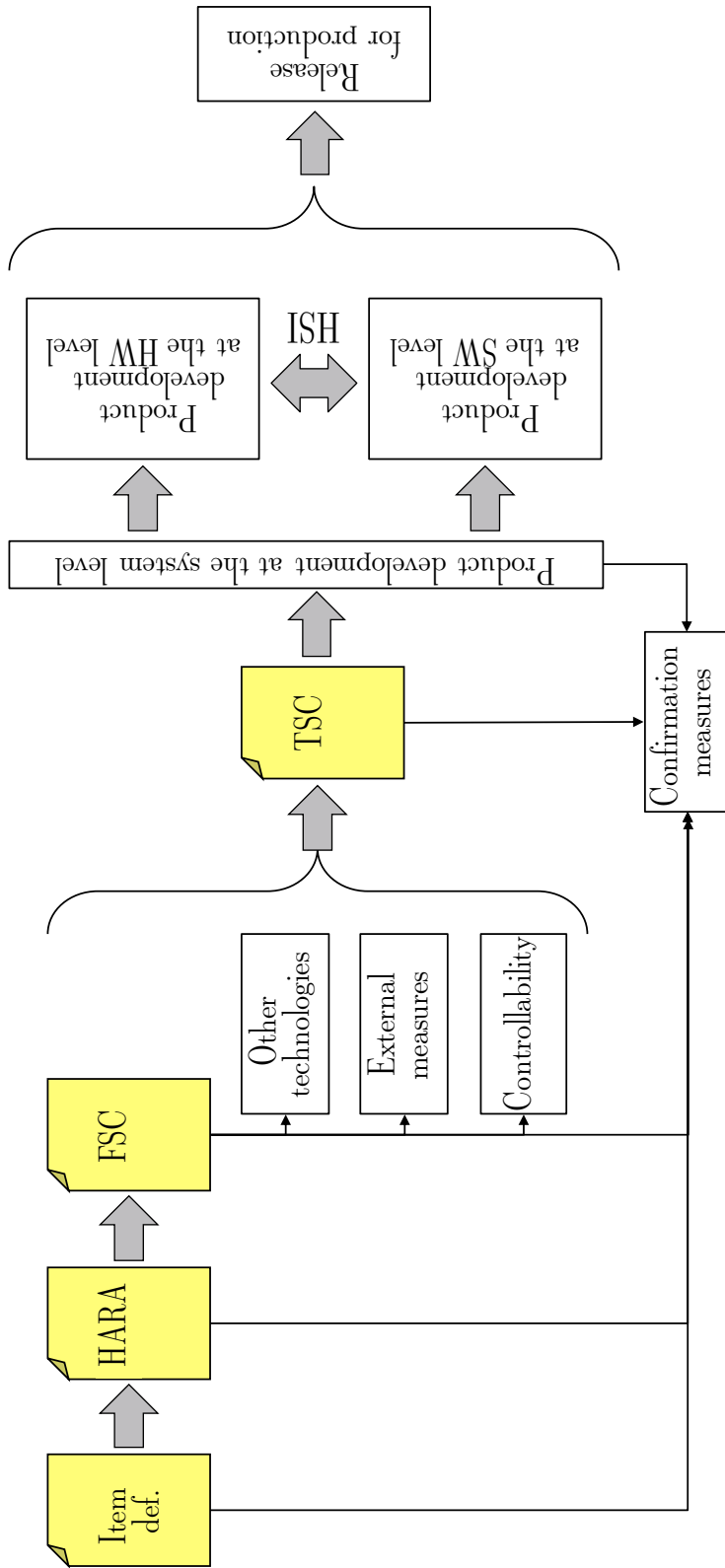


Figure 2.12: System level development process with the main involved documents.

2.5 Technical safety concept

Once we have the FSC, it is possible to start the development at the system level, following the guidelines described by the ISO26262 part 4. In particular, part 4-5 describes the general topics for the product development at the system level, the planning of validation, and activities needed for safety audits and assessment.

From the design point of view, the main document to produce during this phase is the *Technical Safety Concept* (TSC).

This document derives from the FSR and the draft architecture reported in the FSC.

In particular, it has to list, in the form of *Technical Safety Requirements* (TSR):

- mechanism to identify and control faults in the item/system itself;
- mechanism to identify and control fault in other systems;
- measures to achieve or maintain the safe state (transition, fault tolerance, and emergency running interval);
- measures for implementing the warning and degradation concepts.

Other than these main points, it has also to describe:

- specifications for the item validation, in particular, separate validation plans related to the safety requirements;
- mechanisms for latent faults avoidance, with a particular emphasis on the test intervals;
- control mechanisms for latent faults (safety measures for dual-point failures). In particular, these mechanisms must satisfy ASIL B for ASIL D safety goals, ASIL A for ASIL B and C SGs, QM for ASIL A safety goals.

2.6 FMEA

The FMEA manual [5] considered into this dissertation has been jointly released by AIAG and VDA in June 2019.

The process proposed into it is divided into seven steps.

The first three are about system analysis. Those are *planning and preparation* (1), *structure analysis* (2), and *function analysis* (3). The next three steps regard the failure analysis and risk mitigation and are *failure analysis* (4), *risk analysis* (5), and *optimization* (6).

The last phase is about *risk communication* (7) through results and safety-related documentation. The flow is shown in fig.2.13.



Figure 2.13: The 7 steps described by the AIAG and VDA FMEA manual. Figure from [5].

The manual describes the *design* FMEA to assess designs, and the *process* FMEA, to evaluate production processes. For this dissertation’s sake, we are interested only in the *design* one. Phases 2, 3, 4, 5, and 6 differ between the two flavors.

To avoid duplicating the description, the FMEA process of this manual and how to aid it by a simulation-based approach are described in section 3.10.2.

2.7 Hardware design and FMEDA

The standard requires in its part 4, table 1, to perform a System Architectural design analysis by means of *deductive* and *inductive* methodologies³.


The Failure Mode, Effects, and Diagnostic Analysis (FMEDA) is an *inductive* methodology adopted to assess the possible consequences of *random hardware failures* that can affect the item. Another techniques, FIDES, is needed to determine the random hardware failure probability (called *failure rate* in the following) for each one of the components of the design, and described in section 2.8, have to be integrated into the ISO26262 framework.

In particular, the adoption of inductive methodologies is required to be performed for all the ASIL levels.


Of these, the Failure Mode, Effect, and Diagnostic Analysis (FMEDA) has the

³Deductive and inductive techniques to examine the relationships between fault (or defect), error and failure effects differs from each other by the methodology applied to the examination. In the *deductive* techniques, the analysis starts from the failure effects up to their cause, while in the *inductive* ones, the analysis starts from fault models up to a prediction of their effects. Commonly used *inductive* approaches are FMEA, FMECA, and FMEDA, while the most used *deductive* methodology is the *Fault Tree Analysis* (FTA). In this dissertation are analyzed only *inductive* techniques.


Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Failure mode effect
R1	2.25	Open	84%	?
		Increase x2	8%	?
		Decrease x2	8%	?
C1	0.26	Interruption	40%	?
		Short circuit	10%	?
		Decrease x2	50%	?
R2	2.25	Open	84%	?
		Increase x2	8%	?
		Decrease x2	8%	?
U1	0.75	Interruption of any pin	50%	?
		Short of adjacent pins	50%	?
		Internal calculation error	50%	?
U2	0.59	Interruption of any pin	25%	?
		Short of adjacent pins	25%	?




Bill of materials (BOM)



Evaluated with a failure rate determination method (e.g. FIDES)



From failure mode catalog (e.g. IEC 62380, MIL-HDBK-217, others...)



Based on the designer knowledge of the circuit's functionalities

Figure 2.14: The content of the outcome result of FMEDA analysis.

purpose of determining the hardware reliability metrics to verify that the design fulfills the ASIL requirements.

2.7.1 Classification rules

The core of the FMEDA is the failure mode effect classification. The end result of FMEDA is a table (see fig.2.14), where for each *component* of the *item's* Bill Of Material (BOM) is indicated its *failure rate* (computed by FIDES, see section 2.8) and are listed its *failure modes*, the *failure modes rate of occurrence* and the *failure modes effects*.

The goal of the FMEDA is to classify the *failure mode effects* by assessing if they can be detected and can violate a Safety Goal by themselves, as shown in fig.2.15.

2.7.2 Rates and metrics

Failure rate (of a component)

The *failure rate*, indicated in the following as λ^c , is the occurrence rate of *random hardware failures* that can affect a component c expressed in FIT⁴.

⁴For the definition of FIT, see 2.8.4.

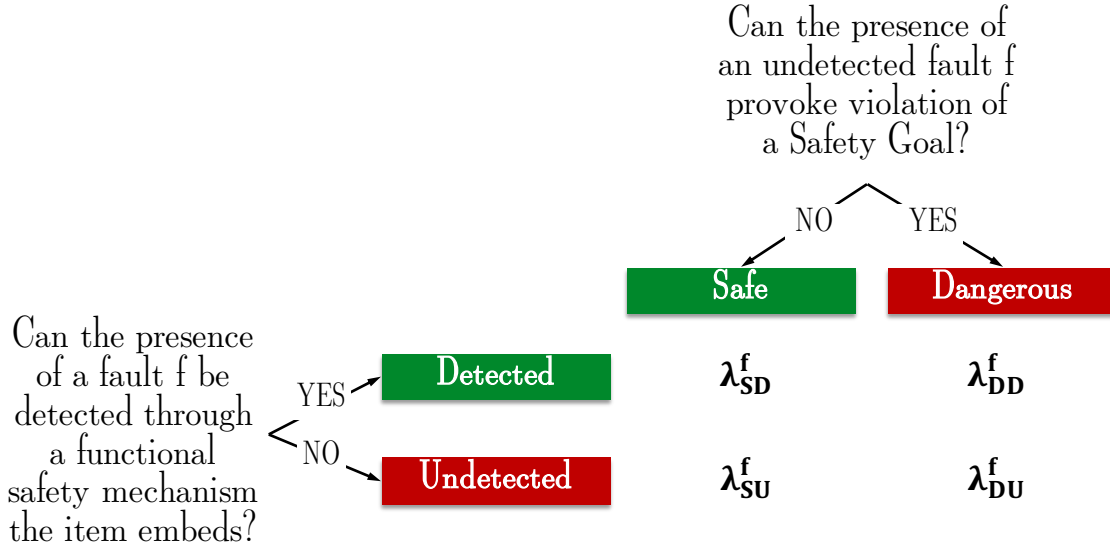


Figure 2.15: The failure mode classification criteria.

From λ^c of all the components, as the output of the hardware verification, engineers shall provide evidence of their item’s robustness by computing the *random hardware*, *single point*, and the *latent fault metrics* and showing that the metrics fulfill the item ASIL’s target values.

Since a single component can fail in various ways, each one of them called a *failure mode*, it is convenient to define an item failure mode rate of occurrence λ^f , from which some rates and metrics are defined for a given failure mode f .

The λ^f for a component c are computed by multiplying λ^f by the considered *failure mode rate of occurrence* f .

Item’s failure rate

The item failure rate, λ is the occurrence rate of the entire item expressed in FIT. It is defined as the sum of the failure rates of all the components (or of all the components failure modes) of the item Bill Of Material (BOM), as in the eq.2.1.

$$\lambda = \sum_{\forall c}^{\text{Components}} \lambda^c = \sum_{\forall f}^{\text{Components failure modes}} \lambda^f \quad (2.1)$$

Dangerous Undetected (Single point fault) rate

The dangerous undetected rate λ^f_{DU} is the occurrence rate of faults that are not detected through any of the functional safety mechanisms the item embeds, and

that can lead by themselves (hence *single point*) to a violation of one or more safety goals, thus causing harms to the item users.

The *single point fault rate* spf is defined as the sum of all the dangerous undetected rates, as in the eq.2.2.

$$\text{spf} = \sum_{\forall f}^{\text{Components failure modes}} \lambda_{DU}^f \quad (2.2)$$

Dangerous detected (Residual fault) rate

The dangerous detected rate λ_{DD}^f is the occurrence rate of faults that are detected through the functional safety mechanisms the item embeds. Anyway, if undetected, the detection mechanisms obviously cannot trigger their mitigation strategies, and these can lead to a violation of one or more safety goals (hence *residual*), thus causing harm to the item users.

The *residual fault rate* rf is defined as the sum of all the dangerous detected rates, as in the eq.2.3.

$$\text{rf} = \sum_{\forall f}^{\text{Components failure modes}} \lambda_{DD}^f \quad (2.3)$$

Safe Undetected (Latent fault) rate

The safe undetected rate λ_{SU}^f is the occurrence rate of faults that are not detected (hence *latent*) through any of the functional safety mechanisms the item embeds and that cannot lead by themselves to a violation of one or more safety goals. Anyway, items with faults of this type are penalized in the FMEDA since in the event another one or even more faults occurs, the combination can violate one or more safety goals, thus causing harm to the item users.

The *latent fault rate* lf is defined as the sum of all the safe undetected rates, as in the eq.2.4.

$$\text{lf} = \sum_{\forall f}^{\text{Components failure modes}} \lambda_{SU}^f \quad (2.4)$$

Safe Detected rate

The safe detected rate λ_{SD}^f is the occurrence rate of faults detected through the functional safety mechanisms the item embeds and that cannot lead by themselves to a violation of one or more safety goals. It is defined as in the eq.2.5.

$$\text{lf} = \sum_{\forall f}^{\text{Components failure modes}} \lambda_{SD}^f \quad (2.5)$$

This rate is not used in the computation of the failure metrics.

2.7.3 Failure metrics

Given the above *single point*, *residual* and *latent fault rates*, three metrics are defined.

Random hardware fault metric

$$\text{rhf} = \text{spf} + \text{rf} \quad (2.6)$$

Single point fault metric

$$\text{spfm} = 1 - \frac{\text{spf}}{\lambda} \quad (2.7)$$

Latent fault metric

$$\text{lfm} = 1 - \frac{\text{lf}}{\lambda} \quad (2.8)$$

ASIL-related metrics limits

The hardware design verification is completed when the computed metrics fulfill the requirements indicated in table 2.7 for rhf, spfm, and lfm based on the target ASIL level.

Metric	ASIL B	ASIL C	ASIL D
rhf	$\leq 10^{-7}h^{-1}$ $\leq 100 \text{ FIT}$	$\leq 10^{-7}h^{-1}$ $\leq 100 \text{ FIT}$	$\leq 10^{-8}h^{-1}$ $\leq 10 \text{ FIT}$
spfm	$\leq 90\%$	$\leq 97\%$	$\leq 99\%$
lfm	$\leq 60\%$	$\leq 80\%$	$\leq 90\%$

Table 2.7: Random hardware failure metrics limits.

2.8 FIDES

The methodology to compute the probability of hardware random hardware failures has been obtained from the FIDES guide 2009, released in September 2010.

2.8.1 Guide structure

The FIDES guide is composed of two parts:

- predicted reliability evaluation guide;
- reliability process control and audit guide.

For the sake of this dissertation, only the first part of the guide is considered.

It has two objectives:

- make a realistic evaluation of the reliability of electronic products;
- provides a specific tool for the construction and control of the reliability model.

2.8.2 Definitions

In this subsection are indicate some definitions useful to understand the rest of the discussion.

These are: *reliability, system, subsystem, equipment, subassembly, electronic component, product, item, failure cause, failure mechanism, reliability contribution factor, and failure mode.*

Please note that some FIDES definitions are *false friends* with respect to the ISO26262 ones. In these cases, the nearest concept inside the ISO26262 is indicated.

Reliability

It is defined as the capability of an item to perform a required function under given conditions for a given time interval.

Reliability is usually expressed qualitatively by appropriate characteristics. In some applications, one of these characteristics is an expression of this capability by a probability called reliability.

System

It is a set of equipment capable of making or supporting an operational role. A complete system includes all equipment, hardware, software, service, and personnel necessary for its operation so that it is sufficient to itself in its usage environment. For example, automobile, aircraft, a microcomputer.

Subsystem

A *subsystem*⁵ is a set of equipment capable of performing an operational function of a system. The subsystem is a major subdivision of the system. For example, an ABS in an automobile, a GPS in an aircraft.

Equipment

This term denotes a group of items capable of performing a complete function. For example, a computer in the ABS systems or a screen in a GPS.

Subassembly

This term denotes an item or an assembled group of items capable of performing a function of the equipment. For example, an electronic board in a computer, a hard disk.

Electronic component

This term denotes an element that will be assembled with other elements to perform one or several functions. For example, a transistor, resistor, capacitor. This definition also includes the printed circuit board (PCB).

Product

It is the assembled entity for which reliability is being studied.

Item

An *item*⁶ is an elementary entity, not broken down, for which designers can study the reliability.

Failure cause, mechanism, mode, and reliability contribution factors

The definitions of *Failure cause*, *mechanism*, *mode*, and *reliability contribution factors* are shown in fig.2.16.

⁵It is almost equivalent to the item in the ISO2626.

⁶It is almost equivalent to the *component* in the ISO2626.

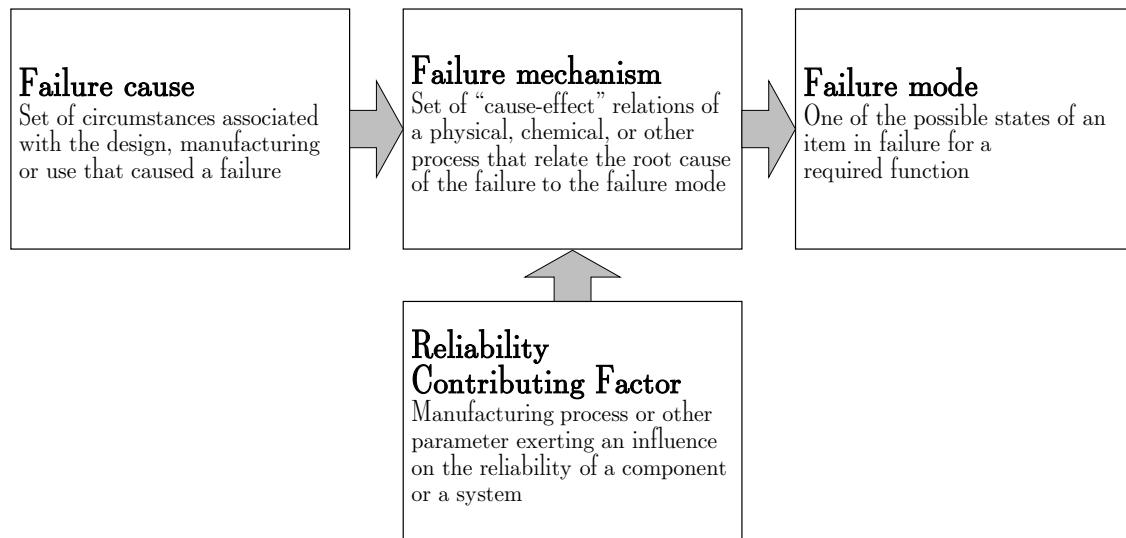


Figure 2.16: Relationship between failure cause, mechanism, mode, and reliability contribution factor. Figure adapted from FIDES.

2.8.3 Model coverage

The FIDES methodology takes into account:

- failures derived from development or manufacturing errors;
- overstresses (electrical, mechanical, thermal) related to the application and not listed as such by the user (the occurrence of the overstress remained concealed);

while not deal with:

- software failures;
- unconfirmed failures;
- failures related to preventive maintenance operations that were not carried out;
- failures related to accidental aggressions when identified or proven, like failure propagations, use outside its specifications, bad manipulations).

2.8.4 Reliability prediction

The reliability prediction given by the FIDES methodology is a failure rate λ . The life of a product can be broken down into three periods, as shown in fig.2.17:

- *infant mortality*, early failures;

- *useful life*, characterized by an approximately constant failure rate;
- *wear-out*, characterized by an increase of the failure probability due to wear.

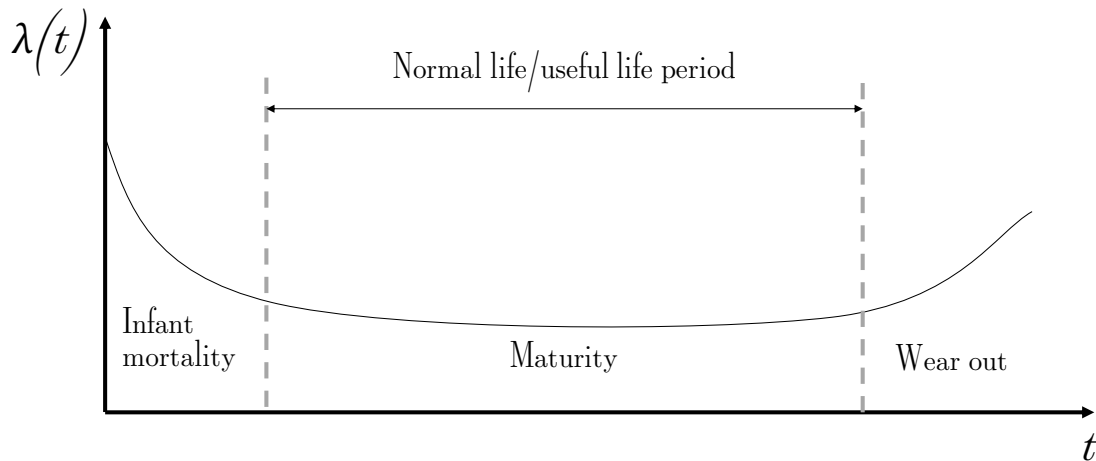


Figure 2.17: The bathtub curve, showing the three periods of the life of a product.

The assumption done into the FIDES manual is that during useful life, the failure rate of E/E components is independent of the functioning hours of the product. The concept of *random failure* describes this assumption. Since only *random failures* are taken into account, *infant mortality* and *wear-out* periods are excluded from the prediction.

- The infant mortality period can be avoided thanks to burn-in processes during the production phase if properly managed during the development of the equipment or system;
- The wear-out period of the E/E components covered by FIDES is sufficiently far in the future compared with the useful life of the equipment or the system. Of course, checking this assumption is a key point during the design of a product. With substitutions of components every certain time of life/usage, preventative services can be required to fulfill this assumption.

Failure mechanism

Microscopically, very few failure mechanisms strictly satisfy a constant rate type occurrence law. However:

- the dispersion of many failure mechanisms, although they are accumulative and therefore increasing with time, is such that they can be deemed to be constant over the life period considered;

- the accumulation of large number and diverse components, even on a single board, will be close to a constant;
- age differences between equipment in the same system or stock of pieces of equipment will tend to make the rate constant for an observer at the system level.

Time To Fail

In some exceptional cases, designers can use the physics of failures to predict the probabilistic life values for a component called *Time to Fail*. This type of prediction is complementary to the reliability prediction but cannot replace it.

Failure In Time

The random hardware failure probability is expressed, coherently as in the FMEDA, in Failure In Time (FIT).

1 FIT corresponds to a probability of 1 failure over 1 billion hours, hence

$$1FIT = \frac{1}{10^{-9}}h$$

2.8.5 Predicted reliability evaluation guide

Inputs for the FIDES analysis

To compute the *component*⁷ FITs, these information are required.

Environment and usage conditions

- Operating temperature.
- Amplitude and frequency of temperature cycles.
- Vibration amplitude.
- Relative humidity
- Ambient pollution level.
- Exposure to accidental overstress (depending on the application type).

⁷FIDES speaks about *item* but, to avoid confusion, I will use the term *component* following the ISO26262 vocabulary.

Data on product definition

- Part list.
- Technical or technological characteristics of items⁸.

Application-related information

- Stress or overstress levels on *component* (dissipated power, stress under power, etc.).
- Local aggravation (or moderation) to temperature or another environmental parameter.

2.8.6 General model

The general model (see eq.2.9) computes the *component* failure rate taking into account *physical* and *process contributions*.

$$\lambda = \left(\sum \text{Physical contributions} \right) \cdot \left(\prod \text{Process contributions} \right) \quad (2.9)$$

λ is the failure rate, \sum Physical contributions represents a mainly additive construction term comprising physical and technological contributing factors to reliability, while \prod Process contributions represents a multiplication term, that represents the impact of the development, production and operation process on reliability.

The equation eq. 2.9 can be rewrite as:

$$\lambda = \lambda_{\text{Physical}} \cdot \Pi_{\text{PM}} \cdot \Pi_{\text{Process}} \quad (2.10)$$

where $\lambda_{\text{Physical}}$ represents the physical contribution, Π_{PM} (PM for Part Manufacturing) represents the quality and the technical control over manufacturing of the item, while Π_{Process} represents the quality and technical control over the development, manufacturing and usage process for the product containing the item.

2.8.7 Life profile

A complete description of the FIDES guide is outside of the purposes of this dissertation.

A central concept of the FIDES guide is the *life profile*. In determining a *life profile* for a reliability prediction, it is necessary to think about

⁸ *Items* are elementary entities, not broken down, for which designers can study the reliability. It almost corresponds to the *components* in the ISO 26262.

what are causes that can produce failure during the item⁹ life.

It requires an engineering approach to reliability, and it is crucial for the reliability evaluation since it strongly influences the prediction accuracy.

The FIDES model has been designed to be sensitive to physical contributing factors. Choosing high or severe values when the life profile is being constructed to remain conservative will eliminate a large proportion of the result's predictive value.

Safety engineers can limit the detail level and the accuracy of the life profile description to the accuracy level with which the product life can be predicted.

General description of the life profile

The first step is to describe it from a qualitative point of view, identifying:

- the precise type of platform when the *product*¹⁰ is integrated into a *system*¹¹;
- the location in the platform, if applicable;
- the geographic or climatic region considered;
- the type of use.

Choice of phases

The choices of phases must be sufficient to describe the different usage situations as completely as possible.

To enable a good understanding of a complex life profile, it may be useful to indicate, for each phase:

- a clear title;
- a descriptive paragraph.

A specific phase must be determined every time environmental conditions change significantly in terms of the stresses encountered.

There is no universal method for breaking it down into phases. Usually is relevant to perform the analysis by considering a typical product usage day.

⁹Component in the ISO26262

¹⁰*Item* or *System* in ISO26262

¹¹*Vehicle* in the ISO26262

Phase duration

It is recommended that designers should build up life profiles with a total duration of 1 year, namely 8760 hours.

The objective is to determine failure rates expressed in calendar FITs, which is recommended rather than using the failure rate expressed in other units, like *per hour of operation* or *per hour of mission*. The phase duration must be expressed in hours and chosen to describe the product's activity as realistically as possible.

Since the failure rate λ has to be expressed in FIT (1 failure per 10^9 hours), it is necessary to weigh the various phases duration in terms of hours per year, as in eq. 2.11.

$$\lambda_{\text{Physical}} = \sum_i^{\text{Phases}} \left(\frac{\text{Annual time}_i}{8760} \cdot \lambda_i \right) \quad (2.11)$$

Applicability domain

The applicability domain, hence the physical contributing factors to be identified for each one of the phases, are:

- temperature (and thermo-electrical) stress;
- temperature cycling¹²;
- humidity;
- vibration;
- chemical stress;
- application type.

In general, the reliability prediction is only applicable in the environment domain for which the component is qualified.

2.9 Fault injection

A widely-used technique for evaluating the sensitivity of a system to faults (in other words, to assess if a fault can lead to an error and hence to a failure) is fault injection.

¹²To determine the thermal fatigue of the item (component in the ISO26262).

Fault injection (FI) can be defined as the deliberate introduction of a fault into a working system in order to evaluate the effects of such fault on it. From an operative point of view, it consists of inserting misbehaviors that emulate the fault (by a fault model) of choice and observing how the target system reacts to it when processing a workload.

FI techniques are used in different fields, from aerospace to automotive, with different goals: validation of fault-tolerant solution, dependability validation, failure prediction, and estimation of the fault tolerance level.

FI simulation is used to simulate a fault in a system even such fault is not actually present or to inject the fault into a simulated system.

In addition to hardware faults, in recent years, the increasing complexity of software, even in terms of code lines, led software faults to become the leading cause of computer systems' failure. The software fault injection was first adopted in [13], but it is possible to find subsequent works in literature, like [14], that often focus on the impact that a software defect (*bug*) might have on a critical system and obtain the certifications when required.

2.10 Differences between FMEA and FMEDA

In this dissertation, the terms FMEA and FMEDA, although often confused in the literature due to the blurred boundary between the two processes, are used differently.

FMEA is the process carried out before and during the ISO 26262 concept phase (and therefore during the design at a high level of abstraction of the item considered) to predict what its failures may be (described at the behavioral level) and which measures can be taken into consideration already from this phase to try to mitigate them.

FMEDA, on the other hand, is the process described in the section 2.7 and required by ISO 26262 part 5, to evaluate the reliability of the designed hardware (in conjunction with the embedded software used within its computing components).

2.11 Software Development for Embedded Systems

Developing software in charge to perform safety-critical tasks inside the framework of the ISO 26262 requires following the strict development process described in its part 6.

The main specification work product involved in SW development is the *Configuration management plan*. From it, a *Software Safety Requirements Specification* document is prepared.

Once we have the latter document, a *Software Verification Plan* is written to describe how to test the software and, after the software is implemented and tested, a *Software verification Report* is prepared as a proof of the conducted test and verification activities. The relations between these documents are shown in fig.2.18.

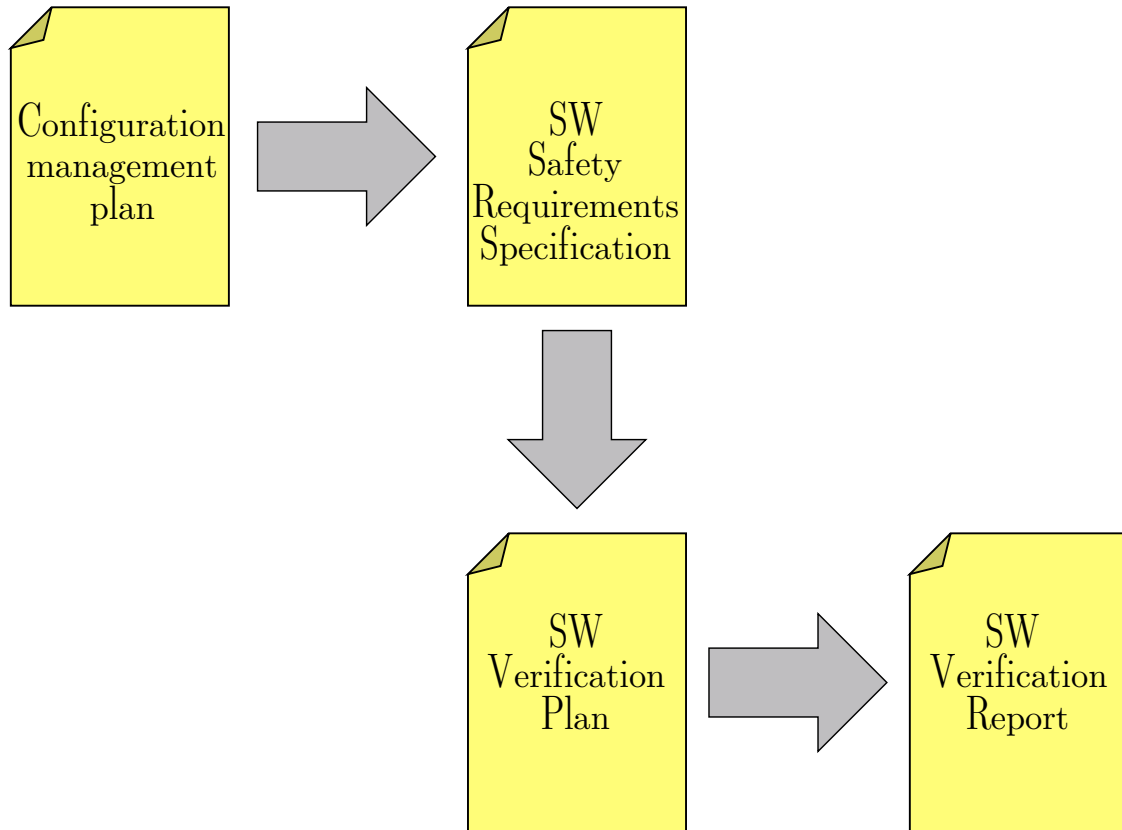


Figure 2.18: Relations between the WPs from *Configuration management plan* to *Software verification Report*.

The *Configuration management plan* (CMP) is prepared taking into account the *Safety plan*, the *Organization-specific rules for functional safety*, and the applicable prerequisites of the lifecycle phases.

The CMP specifies the work products needed to reproduce the SW:

- models (and/or) Source Code¹³;
- software development environment (including its configurations);
- software deliveries (both source code and binaries);

¹³Models in case the Model-Based Software Design approach is adopted.

- drivers;
- operating system;
- evidence of the various development activities.

Focusing on the software implementation process, the expected work products, written alongside the already presented *Software Safety Requirements Specification* are: *Software Architectural Design Specification*, *Hardware/Software Interaction Specification* and a *SW Unit Design Specification* for each one of the software unit envisaged in the architecture. The relations between these documents are shown in fig.2.19.

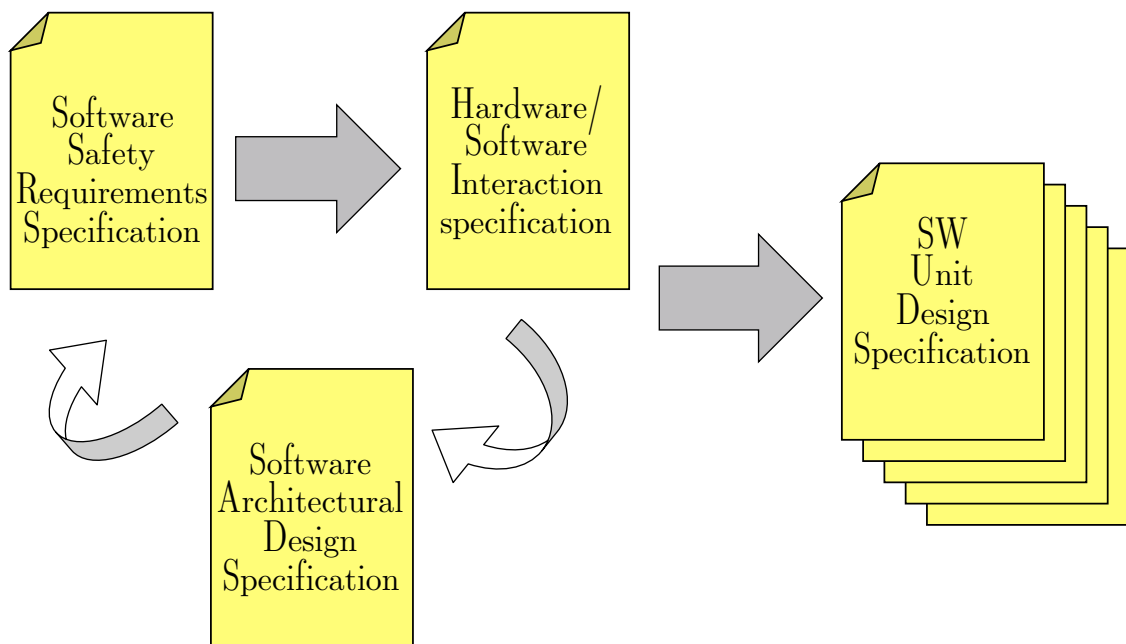


Figure 2.19: Relations between the WPs to be prepared during the software implementation process.

The software is a particular product, different from the hardware components since it cannot be affected by manufacturing errors, defective components, faults, or aging. However, it can contain defects or systematic errors¹⁴.

¹⁴The commonly used term to define software defects or systematic errors is *bug*, but it is not used in the ISO 26262 standard. In the rest of the dissertation, regarding software, the terms *fault*, *systematic error*, *defect*, and *bug*, are used as synonyms to indicate a flaw in a component or system that can cause it to fail systematically in performing its required function, or an incorrect statement or data definition.

2.11.1 Vocabulary

These terms are defined in a specific way when we are dealing with software development inside the ISO26262 framework.

Component

It is a non-system level element that is logically or technically separable and comprises more than one HW part or SW unit.

HW part

It is a portion of an *HW component* at the first level of hierarchical decomposition (example: resistor, CPU of a microcontroller).

Element

It can be a *system*, a *component* (HW or SW), a *HW part*, or a *SW unit*.

Embedded SW

It is the fully integrated software to be executed on a processing element.

Software Component (SWC)

A composition of one or more *software units*.

Software Unit

An atomic-level SW component of the SW architecture that *can be subjected to standalone testing*¹⁵.

2.11.2 V-model

The V-diagram, shown in fig.2.20, is a widely adopted flow to manage software development.

It is beneficial for developing safety-critical software since it clearly shows how to map the development phases (on the left side) with the verification phases (on the right side) and the interaction needed to solve the defects found during the verification phase.

¹⁵The author would like to highlight how the possibility of testing is kept into account into the definition of *Software Unit* due to the central role of testing in the ISO 26262.

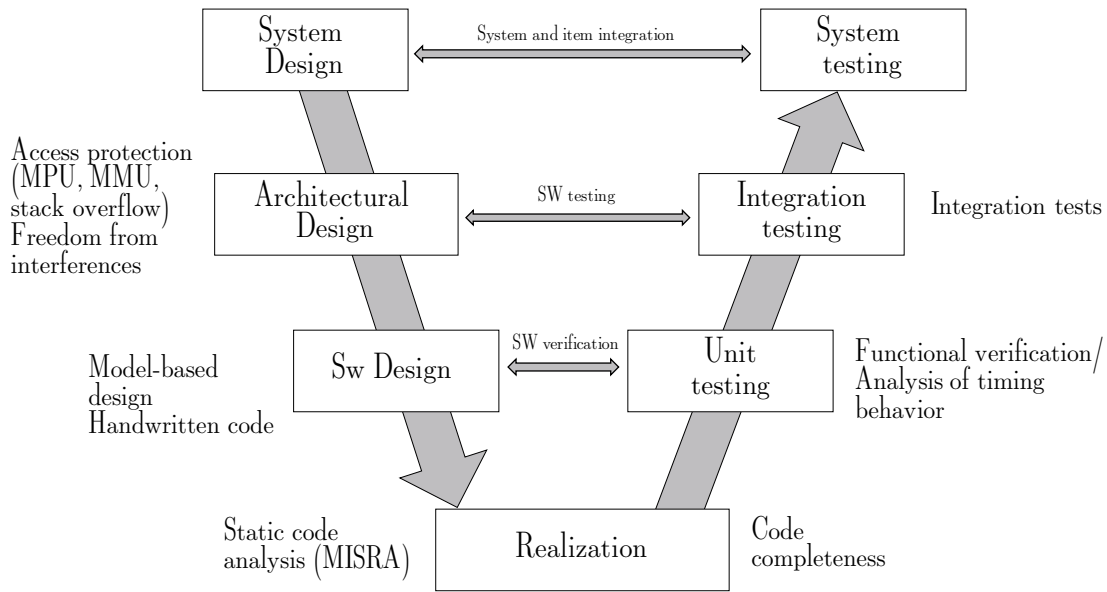


Figure 2.20: The V-diagram of the software development.

2.11.3 Model-based software design

A novel approach adopted to develop automotive software is the Model-Based Software Design (MBSD).

The idea is to realize a semi-formal model of the software units and architectural design in an environment like Mathworks© Simulink©. These models can be run in the modeling environment, perform simulation, and translate thanks to Code Generator Tools, like Mathworks Simulink Coder© and dSpace© Targetlink© into C or C++ source code.

The obtained code can be integrated by hands alongside the handwritten code, or it is possible to generate a complete embedded software thanks to Platform Support Packages. The first approach is widely used in the production-level code, while the latter for Rapid Control Prototyping (RCP). The following describes how to test software inside the FuSa framework, also considering the MBSD approach. Moreover, this description is useful since, in this dissertation, some algorithms developed through MBSD are shown.

2.11.4 Software unit testing

As already described in section 2.11.1, the testing has a central role for FuSa. It is so important that even the definition of *software unit* states that a software unit is the minimum part of code that is standalone testable.

The goal of these tests is to avoid the presence of *defects* in the software that can lead to *systematic failures*.

Various tests at various levels (*unit, integration, system*, see fig.2.20) have to be performed. The lower level is *unit testing*, where a *software unit* is tested against its requirement.

The idea is to define a *software unit*, hence implement it, then test the obtained implementation. To test it, a complete set of stimuli have to be defined. To check if all the cases are taken into account (stimulated), three metrics have been defined, as described in the (see section 2.11.4).

Code coverage metrics

There are three different *code coverage metrics*:

- *Statement coverage*: the percentage of statements of a unit the test executed.
- *Branch coverage*: the percentage of branches of a unit the test executed.
- *Modified condition/decision coverage (MC/DC)*: the percentage of the conditions that affect independently the outcome of a unit the test executed.

Model-In-the-Loop (MIL)

MIL can be performed only on MBSD developed software. It consists of running the SW unit inside the simulation environment on the development PC. It provides information about the correctness of the implementation against its requirements.

Software-In-the-Loop (SIL)

SIL can be done on both MBSD or handwritten code, with different purposes. In the first case, the output obtained in MIL is compared against the one obtained by the automatically generated code, while in the latter, its purpose is to test the handwritten code against its requirements.

Processor-In-the-Loop (PIL)

PIL is done by executing the source code on the target hardware. It can be useful for two different purposes. The first is to verify if the toolchain is capable of generating correct binaries (a particularly critical point is the compiler), while the second is to allow measurements on the runtime required for the developed software unit (see debug and trace), allowing optimization of the code without the risk of introducing defects into the final code. This is also useful for measuring the Worst-Case Execution Time (WCET), fundamental to running the software on real-time computers.

2.11.5 Software integration testing

Debuggers

In the previous paragraph, the main focus was verification, but we also want to know how things are going, at runtime, inside the software. Usually, with integrated development environments (IDE), it is simple to break the code execution and see the memory content, but embedded systems do not have a keyboard or a screen. Hence, the need to use external objects, called *debugger*, to interact with them. Simpler debuggers can break the code execution and monitor a limited number of variables on the programming computer. This approach can be adopted for most of the project, but the need to insert breakpoints leads to break down the real-time behavior of the software, making it not possible to debug while the target is controlling the physical process.

Real-Time Testing

Performing the required operations on time has a crucial role in achieving FuSa, as described in section 2.3.4. Hence, the need to verify the correct timing of embedded software tasks.

Trace tools Thanks to debugging infrastructure embedded in some microcontrollers, more sophisticated debuggers offer the possibility to save the performed operations on external memory and move them to the host pc. In this way, it is possible, by observing what is going on in the data and address buses, analyze them by an analysis tool, and reconstruct all the operations in a specific time window, simplifying the debug operation.

Moreover, it allows to measure the runtime and to avoid instrument the code. The most used device capable of tracing what is going on 32-bit ARM cores is the Lauterbach Trace, with its host software called Trace32.

Hardware-In-the-Loop HIL is done to perform integration testing between the various software units.

The software units are executed, integrated into the target platform alongside the firmware components like device drivers. In this case, since all the embedded software is loaded into the target, it is necessary to stimulate the item with signals identical to the field from both the timing and the electrical points of view.

In other words, the idea is to "trick" the target that it is dealing with the real world. It is possible by running a physical model of the controlled system on a real-time computer and converting the simulation results into electrical signals by output conditioning stages and digital-to-analog converters.

Similarly, the inputs to the simulation are adapted by input conditioning stages and converted by analog-to-digital converters.

In this way, the signals are the same from the electrical point of view, and the simulation of the plant, running in real-time, acts as the physical world.

A schematic representation, alongside a picture of the HIL system used on the experiments described in section 5.2, published in the papers [9] and [10], is shown in fig.2.21.

Based on the physical model of the controlled plant, this approach can appear counterproductive since it requires more software development.

Two factors help to mitigate this disadvantage: on the one hand, models have to be developed in any way to design the control algorithms and test them in MIL/SIL approaches; on the other, it is possible to accelerate the model development by adopting the MBSD to generate the model to be run on the real-time computer. Moreover, since to perform HIL RCP devices are used, Platform Support Packages are available on the market. Standard companion software (see section 5.1) used to perform HIL are NI VeriStand, dSpace ControlDesk, and LHP Panthera.

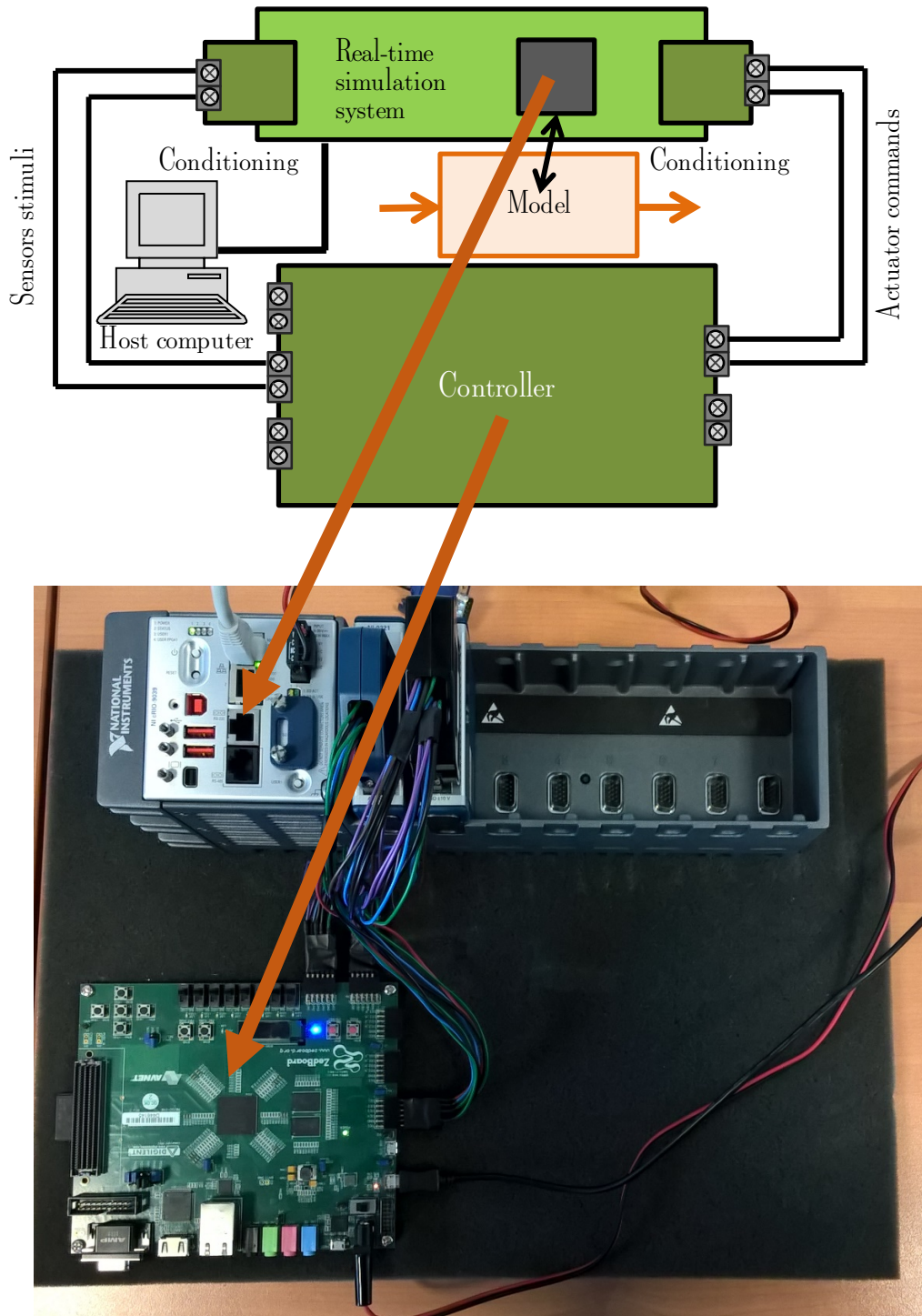


Figure 2.21: HIL conceptual structure compared with a real implementation, based on a National Instruments cRIO as the real-time simulator and a ZedBoard as the controller.

Chapter 3

Simulation-based FMEDA

This chapter describes the contributions proposed during my Ph.D. course to implement simulation-based approaches to perform the Failure Mode, Effects, and Diagnostic Analysis (FMEDA).

The results shown in this chapter has been already published into seven papers released from 2017 to 2021: [15], [16], [17], [18], [19], [20], and [21]. [21] is still under review (april 2021).

Moreover, two Masters' Degree thesis have been obtained from these activities. The first has been written by Marco D'Auria [22] that collaborates to implement the fourth proposal, published in [19], while the second has been written by Andrea Passarino [23], that collaborates to implement the sixth proposal, contained into the already unpublished manuscript [21].

The discussion, for each one of these proposals, have been split into eight points:

- proposed methodology;
- benchmark application;
- fault models;
- fault injection techniques;
- failure mode effects assessment;
- assessment of the embedded software detection and mitigation capabilities on hardware failures;
- experimental setup;
- simulation results.

3.1 Industrial practice

The FMEDA (see section 2.7) is an industrial practice to compute, for a given item implementation, its random hardware failure metrics (see section 2.7.3) required by the ISO26262-part 5. It starts from the Bill Of Material (BOM) and the hardware design schematics to be verified. Then, resorting to hardware components failure modes (FMs) rate catalogs, such as IEC 61709, and reliability calculation models such as FIDES (see section 2.8), the components' failure rates are computed.

To classify failure modes (one component can fail in more than one way) effects among safe/dangerous and detected/undetected (see fig.2.15), designers inspect the hardware schematics through *inductive* analysis. It is based mainly on the designers' knowledge and expertise on the circuit functionalities.

This approach is potentially error-prone and suffers from three shortcomings:

- with the growing complexity of embedded systems employed in safety-critical applications, manual inspection of the schematic can become ineffective in identifying all the possible misbehaviors;
- it is based on worst-case estimations, hence it can lead to overdesign;
- it is not easy to consider the detection and mitigation effects of the embedded software.

Regarding the last point, as a matter of fact, the schematics to be verified often includes integrated circuits (ICs), systems on chip (SoC), microcontrollers, microprocessors, or DSPs. With the manual approach, it is challenging to consider the contribution of the embedded software, which can affect the failure handling (how the software contributes to the fault propagation) positively (mitigation) or negatively.

When the process is completed, each component FIT λ^c is known and their failure modes rates of occurrences, hence λ^f , and each failure mode (FM) has been classified. It is possible to obtain λ_{SD}^f , λ_{SU}^f , λ_{DD}^f , λ_{DU}^f , hence to compute the metrics required by the ISO26262 (see section 2.7.3). In the case the target metrics values for the considered ASIL are not reached (see table 2.7), a redesign of the item is needed.

3.2 The idea of automating the FME(D)A

In this chapter, the term FME(D)A is used instead of the more precise FMEA and FMEDA to refer to the works present in the literature for the reasons explained in the section 2.10. The concept intended by the authors of the presented works is nearest to the FMEDA definition with respect to the FMEA one.

The core idea on automating the FME(D)A process came from the paper [24], where the authors describe their experiences in using fault injection to perform some phases of the FMEA. Starting from the latter reference, the authors of [25] applied a similar approach on a biomedical system, while in [26] it is described as an automatic tool to estimate the hardware failures criticality.

All these three works lack in keeping into account the contribution of the embedded software on the failure handling. This lack was the first reason for developing a methodology capable of simulating both hardware components and embedded software side by side. This aspect is essential as, as already seen in chapter 2, hardware-software interaction analysis is crucial in the ISO26262 safety lifecycle framework.

In the papers [27], and [28] it is presented a framework to perform FME(D)A at the software level.

In particular, in [28], the impact that a software defect (bug) can have on other components that depend on the corrupted code is evaluated, but without an analysis on the validity of the injected *faults*¹, which occurs in [29], where realistic software faults are used, based on the most frequently occurring coding defects. This methodology is multidisciplinary and borrows content from various fields of information engineering (fault injection, fault modeling, simulation, electronic printed circuit board design, embedded software development). Hence, a comparison with similar proposals or against standard benchmarks is not possible.

3.2.1 Fault coverage measurement

Digital tests are graded (and compared to each other) using *fault coverage metrics*. How to compute these metrics is well defined, thanks to the standard IEEE 1804 and the market availability of various EDA tools, like Mentor Graphics DefectSim[™] and Synopsys TextMax[™].

Also for software testing, *statement*, *branches*, and *decision* coverage metrics are well defined and widespread accepted in the scientific community.

Unfortunately, analog tests are intrinsically more complicated with respect to digital ones [30]. Hence, many approaches have been used in the last three decades, yet nothing is standard. A standard is on the way, as the proposal IEEE P2427 [31].

¹The term *fault* is used in this case referring to software, instead of *defect*, due to the use of *fault injection*: in this specific case the text refers to inject *transient* random hardware failures behaving as a wrong specific value inserted in the application text (instruction) or data memory regions, and not in the deliberate instrumentation of a defect (bug) inside the source code. In the rest of the chapter, the source code is considered as defect-free since the presence of systematic failures is not kept into account in the FME(D)A analysis.

3.2.2 Fault models

Nowadays, the IEEE P2427 Working Group is underway to tackle this challenge and define the standard.

The missions of the Working Group [30] are close to the needs of the ISO26262 FMEDA:

- Define the following processes:
 - given a circuit, produce the *failure modes list*²;
 - given a set of tests for that circuit, determine which defects are detected.
- Standardize the following information:
 - definition and communication of *defect models* and *detection criteria*;
 - reporting of *defect coverage*.

The IEEE P2427 Working Group’s fault model is *defect-oriented*, requires a circuit-level (structural) model of the circuit to be applied, keeps into account *defects*³, considers *single failure* at any given time and models failures considering only the manufacturing flaws, under the assumption that a test that can detect zero-time defects can be reused later in the life cycle.

The *defects*⁴ are considered as *permanent* and *unexpected*⁵. These can be modeled in two different ways: [30] [32] [33] [34] :

- *Catastrophic*;
- *Parametric*.

The *catastrophic defects* are those one modelled as open or short circuits affecting both components or the network topology. These can be obtained, in simulation, by inserting switches between the terminals of the components or between nodes that are unconnected by design (topological). Topological defects, in other words, introduce further possible short circuits in the subsystem circuit diagram.

Even if the *catastrophic defect* models cover a significant number of possible scenarios, they are not sufficient to describe the complexity of analog circuit defects.

²The document [30], calls them *universe of defects*.

³A *defect* is an unintended significant change affecting a design intent primitive circuit element or the connection between them. It is possible to consider it, in this dissertation, as a synonym of *failure mode*.

⁴*Failure modes*.

⁵For the sake of this thesis, the term *unexpected* used in [30] can be considered as indicating the concept of *random failures*, as described in the FIDES guide (see 2.8.4).

Hence the need to introduce the *parametric* defect models. These are described as variations of components parameters (e.g., resistance, capacity) outside their nominal ranges (defined as nominal value \pm tolerance).

It is important to remark that in this dissertation only the *catastrophic/parametric* classifications are from the IEEE P2427, while the used models are developed by the authors of the various paper or from the literature, as specified for each proposal in the relative *fault model* section.

At the moment, a widespread accepted standard about coverage computation is not available; hence, no coverage metrics for analog tests are reported on this dissertation.

3.2.3 Fault injection

The fault injection techniques presented into these proposals are based on the papers [35] and [36].

The authors of the following papers highlighted that fault injection make it possible to achieve different goals:

- dependability validation [37];
- failure prediction at the item level [38];
- estimation of the fault tolerance level [39].

Moreover, the authors of [13] also considers when also the embedded software is concerned, while in [14] is kept into account the effects that a software *defect* can have on a safety-critical system.

From the fault injection point of view, the main novelties are the proposals of the various setups to perform the injections inside the chosen simulation environments, as described for each one of the proposals.

3.3 Proposals

To overcome the limitations of manual hardware design inspection, the proposals described in this section have been devised.

When hardware design verification is started, it is therefore very likely that a model of the item and a suitable set of workloads are available since they are used to develop the control routines. These can be used as a starting point to improve hardware verification.

3.3.1 Hypotesis

Safety engineers can make some assumptions about the architecture of the average automotive *item*⁶.

These items can be considered as composed of three stages, as shown in fig.3.1:

- Input conditioning stage, which is typically an analog circuit to adapt the analog inputs the item acquires, from the environment or other items to the input dynamics required by the processing stage embedded in the item;
- Processing stage, which is typically based on a SoC, microprocessor, or microcontroller the item embeds;
- Output conditioning stage, which is typically an analog circuit to adapt the processing stage's output dynamics to that required by actuators or other items.

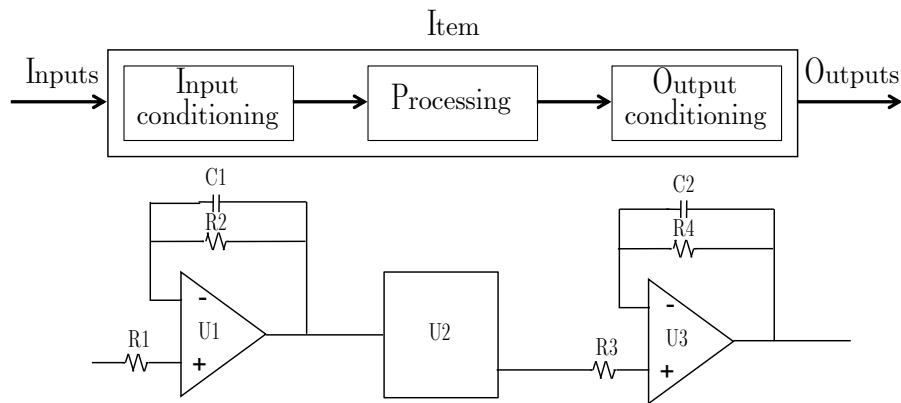


Figure 3.1: A general three stages structure for an item.

3.3.2 Evolution of the proposed approaches

The approach evolved over the years. Hence it can be described as six proposals, from the first more straightforward iteration to the final one applied on the traction system of a mobile robot. Figure 3.2 shows the relations between these proposals in chronological order. The descriptions near the arrows explain the novelties between the proposals.

⁶As described in the fifth and sixth proposals, it can be generalized to other application fields.

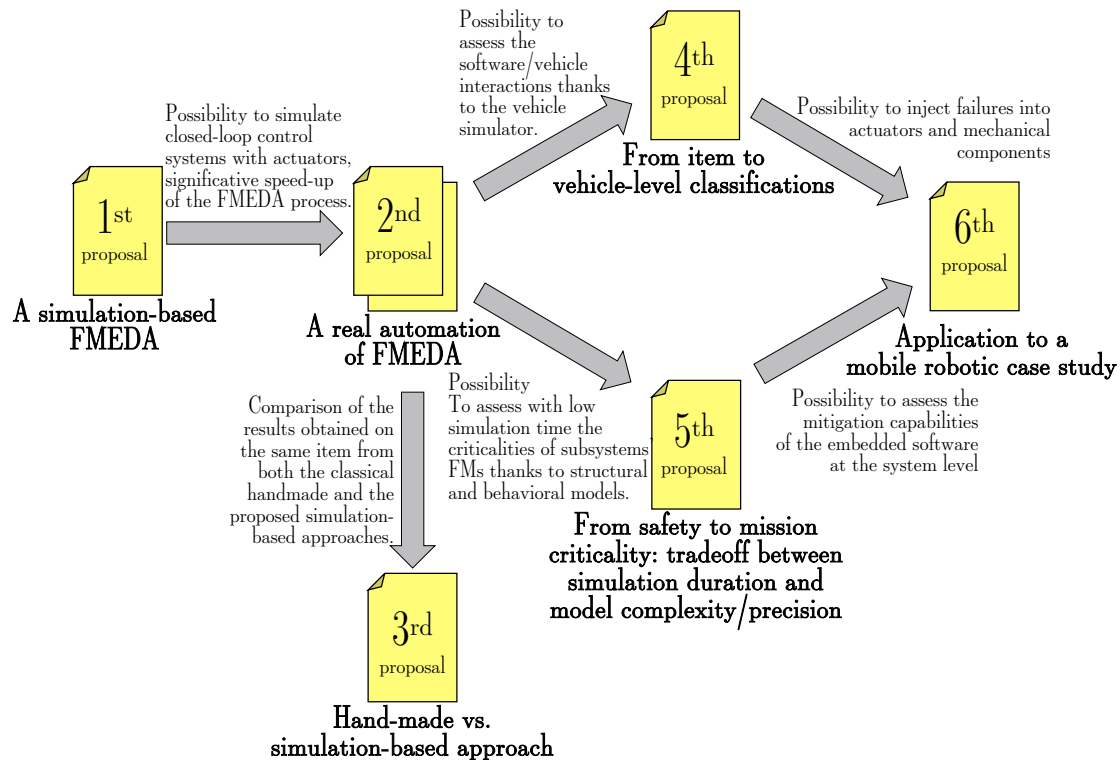


Figure 3.2: Workflow of the simulation-based methodology evolution. The descriptions near the arrows explain the novelties between the proposals.

First proposal - A simulation-based FME(D)A

The first proposal, presented in [15], investigates the technical viability of a simulation-based FME(D)A.

The proposal is intended to analyze how to simulate the *fault* propagation to an eventual *error* and hence a *failure*. The FMs effects are assessed thanks to the simulation results.

For this purpose, the faults affecting input and output conditioning stages are simulated, thanks to the fault models described in the following, resorting to a SPICE-level simulator. Simultaneously, Mathworks Simulink is used to run the software and analyze its contribution to fault propagations.

This propagation can be studied only from the input to the output conditioning stages due to the lack of integration between the software environment and the SPICE-level simulator. It prevents applying this first iteration of the approach to closed-loop control systems.

The results obtained in this first iteration were encouraging but not yet effectively applicable to an industrial project.

Second proposal - A real automation of FME(D)A

The first proposal [15] has many limitations. It demonstrated the viability of aiding safety engineers to perform the ISO26262 FMEDA, but it requires much manual work (to inject the failure and run the simulation) and make it difficult to run the embedded software alongside the simulations in the case the item to be assessed is a closed-loop control system.

To overcome these limitations, this second iteration, presented in the two papers [16] and [18], proposes a new environment for automatic FME(D)A, allowing to obtain:

- significant speedup of the analysis time due to centralizing the simulations' management upon a MATLAB Script, since the SPICE-level simulations are performed resorting to the Simulink Simscape toolbox [16]. This level of integration with all tools running into the MATLAB/Simulink environment, allowing the presence of automatic *sabouter*, *fault list generator*, and a useful *report generator* to document the FME(D)A results automatically according to the ISO26262 requirements;
- the possibility to assess FMs affecting closed-loop control systems and to improve the classification rules [18] thanks to a model of the actuator (in this case, an electrical motor) simulated alongside the item schematics.

Third proposal - Handmade vs. simulation-based approaches

The purpose of this proposal, published in [17], is to compare the results obtained, on the same item (a monitoring circuit to verify the power consumption of a video interface for an autonomous-driving vehicle), from a handmade and a simulation-based FME(D)A performed with the approach presented in the second proposal (paper [16]). The papers [17] and [16] are coeval.

The classification capabilities of the simulation-based approach are based on thresholds between the maximum differences on voltages present in the fault-free and in fault-affected conditions. No other knowledge about the item is used.

As a result, 91% of the FMs classifications obtained with the simulation-based approach are identical or more conservative w.r.t. those obtained with the handmade methodology.

Fourth proposal - From an item-level FMs classification to a vehicle-level one

The purpose of this proposal [19] is to describe a methodology to propagate the FMs effects to the entire vehicle. A set of detection and mitigations algorithms to be applied to a dual-motor axle have been proposed as a benchmark application. Thanks to the vehicle-level simulator, the FMs effects are classified considering

their impact on the vehicle drivability. Due to this, the simulation results are not presented as an FME(D)A table, with the FMs effects classification, but as plots of the vehicle behavior, in terms of speed and yaw angle, in some interesting scenarios. Like the one of this proposal, simulation-based approaches are widely adopted during automotive software development, but to verify whether the system can reach the nominal performance requirement in the early stages. It is proposed to introduce this approach during the FME(D)A to produce results that can aid these analysis also in those cases where the interactions between the item and the vehicles that embeds it are complex.

Some interesting results about how to study the software/vehicle interaction have been obtained, by considering the effects of two straightforward detection and mitigation algorithms on a dual-motor axle. Hence, the approach demonstrated itself able to aid functional safety engineers.

Fifth proposal - From safety to mission criticality: finding a tradeoff between real needs and model precision

The contribution of this proposal is to describe a methodology for effectively performing the *Failure Mode, Effects, and Criticality Analysis* (FMECA)⁷ required by international safety standards in other areas. The benchmark application is a control board designed to drive the electric motor of an industrial compressor. It is considered as a complex cyber-physical system.

The introduced novelty mainly lies in the proposed simulation approach, which allows for injecting FMs into a considered subsystem and analyzing their impact on the whole cyber-physical system.

In particular, the proposed approach can identify the critical faults, whose number is often overestimated by other methods.

The proposed methodology relies on a multilevel simulation of a cyber-physical system that mixes *behavioral* high-level and *structural* SPICE-level models of its different subsystems.

In the proposed simulation methodology, to tradeoff between simulation accuracy and duration, all the subsystems are described with *behavioral* models, except for the target one, called *subsystem under test* (SSUT), which can be affected by the considered failure mode⁸. The adoption of a *structural* models allows high-quality modeling of the FMs, while *behavioral* models allow studying the propagation of

⁷The FMECA is about the criticality of the FMs. Their classification criteria for this specific application are different from the ones of the ISO26262 FME(D)A and are discussed in section 3.9.

⁸In [40] are called faults. However, to be coherent with the terminology adopted in this dissertation, I prefer the term failure mode.

their effects to the other subsystems without making the complex system simulation excessively time-consuming.

Sixth proposal - Development of a high-dependable mobile robot

This proposal, presented in a still unpublished manuscript, is about the FMEA, HARA, and FME(D)A of a mission-critical mobile robot. The complete description is available in

[23]. For the sake of this thesis, I describe only the FME(D)A, in this case performed considering failures of an entire sensor or actuator.

From the safety point of view, the considered system is a fail-safe one; hence all the safety-relevant failures are managed by shutting down the mobility subsystem of the rover. Some non-safety-relevant failures are mitigated thanks to the embedded software.

Moreover, sensors and actuators' failure mitigation algorithms are assessed to demonstrate their effectiveness without the need of a prototype.

This proposal can be considered as the merge of all the proposals discussed into this and 4 chapters. As emerged from [18], safety engineers can also use these models, thanks to their execution speed, to perform real-time validation of the embedded software (see HIL in section 2.11.5 and real-time software validation in chapter 5).

It solves two limitations seen in the previous proposal by making it possible to inject failures in the actuators and mechanical components (limitation of the fourth proposal [19]) and to assess the mitigation triggering capabilities of the embedded software at the system level (limitation of the fifth proposal [40]). Due to the complexity of the analysis, the mitigation effects of the proposed algorithms are not described. No automatic rules are defined, but the DIANA team members assess the simulation results to improve their design.

The results present in this dissertation are limited to the measures of time between the fault injection and the triggering of the opportune mitigation strategies.

3.4 Research contributions

In the *first* [15] and *second* [16] *proposals*, the simulations are performed by implementing the schematics of the item hardware components inside commercial SPICE-level simulators (LTSpice and MathWorks Simulink Simscape) to simulate both *catastrophic* and *parametric* failures.

More in detail, in the first proposal, there are no novelties in the fault injection per se, but as best of my and my coauthor knowledge, it was the first application of such simulation-based approach to perform the failure mode classifications as required by the FME(D)A in compliance with the ISO26262. In the second proposal, it

has been decided to describe a methodology suitable to simulate also closed-loop control systems.

The *third proposal* does not introduce any novelties but tests the effectiveness of the proposed simulation-based approach (in particular the methodology proposed in the second proposal) against the traditional handmade approach [17]. My coauthors and I claim that this is a crucial benchmark since no other objective and repeatable comparisons are possible, as previously described, due to the lack of coverage metrics for analog tests.

The *fourth proposal* propagates the failure mode effects from the actuator (item) level [18] to the whole vehicle, thanks to the presence of a vehicle-level simulator [19]. It allows assessing the failure effects on the vehicle drivability.

The *fifth proposals*, is based on the fourth proposal [19] and two other works focused on FMECA: [41] and [42]. It introduces the simulation-based approach of [19] to the latter two methodologies, allowing to obtain an approach more accurate and complete thanks to the possibility to adopt hybrid models where the fault-affected subsystem is simulated with structural model and the fault-free ones are simulated with behavioral ones. This allows improving the simulation speed without giving up the accuracy of the fault models.

The *sixth proposal* presents a mobile robotic application, in this case, a Mars exploration Rover designed by the student team DIANA of Politecnico di Torino. It allows the possibility to inject mechanical failures thanks to the presence of commercial behavioral models for both the actuators and mechanical components [43] and to assess the effects of the failures at the rover level (as in the fourth proposal).

None of the contributions address ISO26262 FMEDA analysis on integrated circuits or systems-on-chip (SoC), as all semiconductor manufacturers provide ISO26262 metric calculation tools for their products based on the configuration chosen for each specific application. The reader can find an example of a training course for the tool provided by Texas Instruments at [44].

3.5 First proposal - A simulation-based FME(D)A

In this first proposal, presented in 2017 at Latin-America Test Symposium (LATS) and published in its proceedings, starting from the stage schematic, a SPICE-level network is produced for the input and output stages. Then, for each FM of each component, a mutated schematic is prepared by hands, to simulate the faulty circuit according to the considered FM.

The benchmark item chosen to demonstrate the approach, is a seat belt reminder, which is responsible for sensing if a passenger is seated and if the seat-belt is buckled; if not, it shall produce a seat-belt alarm.

Assuming the HARA procedure ranked the item as ASIL C, considering the worst case if the airbag deploys and the passenger is not wearing the seat belt, severe injuries are possible due to the passenger's head crashing into the airbag balloon when being inflated.

The paper is considered only the portion of the item responsible for sensing the passenger on the seat, which is based on the input stage shown in fig.3.3.

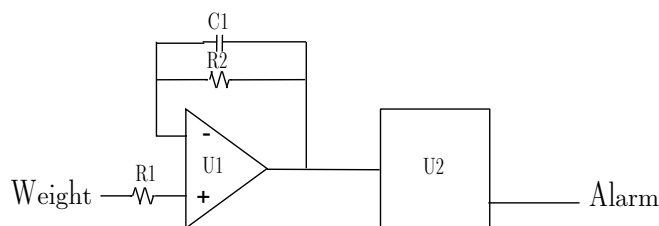


Figure 3.3: Structure of the seat belt reminder.

Its input stage provides to the microcontroller an input voltage that is proportional to the passenger weight so that it can provide an input in the range $[W_{min}, W_{max}]$. The embedded software run by the microcontroller performs a range check:

- in case of an out-of-range input, it signals the detection and an error;
- otherwise, it controls whether the seat belt is buckled when the input value indicates that a passenger is occupying the seat; if the belt is not buckled, an alarm is generated.

3.5.1 Fault models

Considering the components R1 and R2 shown in fig.3.3, which are resistors based on a given technology, it is possible to identify, from failure catalogs, that their relevant failure modes are three:

- open circuit (with an occurrence of 50%, i.e., in the reliability analysis 50% of the time the resistor is faulty, its failure mode corresponds to an open circuit);
- doubled resistance w.r.t. the nominal one (with an occurrence of 25%);
- halved resistance w.r.t. the nominal one (with an occurrence of 25%).

Considering the component C1, which is a capacitor, it is possible another three failure modes:

- Interruption, simulated by opening a switch in series to the capacitor (occurrence of 40%);
- Short circuit, simulated by closing a switch in parallel to the capacitor (occurrence of 10%);
- decrease, simulated by halving its capacity parameter (occurrence of 50%).

Of these FMs, the open circuit/interruption and short circuit can be considered, according to the classification made in the IEEE P2427 proposal, among *catastrophic defects*, while double/halved resistance and decrease (capacity) among the *parametric* ones.

The failure mode effects of U1 and U2 that are ICs are classified as Dangerous Detected(DD) for the motivation described in section 3.5.3.

3.5.2 Fault Injection

In this proposal [15], there is no automatic fault injection strategy: fault injection is performed modifying by hands the schematics before each simulation, with the models described in section 3.5.1.

This process requires creating 10 different circuit schematics, one fault-free and one for each of the considered failure modes of components R1, R2, and C1.

3.5.3 Failure modes effects assessment strategies

The first simple failure mode effects assessment is performed as follows. A failure mode is injected⁹ in the item schematic by changing the parameters of the affected component. It then analyzes how it propagates to the item outputs when the item inputs are simulated with a user-provided workload. Fault propagation is done using a mixed-level simulation environment where;

⁹As described in section 3.5.2, the process is handmade, but since the schematics of the analog conditioning stages are modified, this process is legitimately called *injection*.

- failure mods affecting discrete components are analyzed using SPICE-level simulation, studying the propagation from the item inputs to the processing stage input, and from the processing stage outputs to the external environment (actuators or other items);
- the propagation of fault effects from processing stage inputs to its outputs, i.e., through the item software, is done using Simulink.

The adoption of Simulink to evaluate its contribution on mitigation of failure affecting the input conditioning stage, and to run the software itself, is consistent with the current trend in the automotive industry of adopting model-based software design (MBSD), as described in section 2.11.3.

Analysis for input conditioning stage

For each one of the FMs that can affect the input stage, a SPICE-level simulation is executed.

By using workloads provided by the designers, the SPICE-level simulator computes the outputs that the faulty input stage provides to the microcontroller inputs.

Thus, this step is intended to identify how the failure modes in the input conditioning stage propagate to the microcontroller inputs. Once this step is completed, it makes available the list of faulty inputs that the microcontroller acquires and transforms into outputs through the software it executes.

To analyze how the faulty inputs propagate to the microcontroller outputs under a given workload, a Simulink model runs the embedded software.

For each faulty microcontroller output (even derived from a simulation-based injection as in [45]), a SPICE-level simulation of the fault-free output conditioning stage is performed, to observe how the whole item reacts to faults into its input-conditioning stage.

Analysis for the processing stage

As the FMs that may affect the processing stage are considered: internal calculation errors, interruption of any pin, and short circuits between adjacent pins.

For the sake of [15] paper, they are not considered, and we assume that the processing stage is provided with adequate self-testing capabilities to detect any possible processor failure. Since the processing stage is a single point of failure, any failure mode affecting it is considered as dangerous detected unless proven otherwise (e.g., from the assumptions contained into its safety manual).

Analysis for output conditioning stage

To analyze the impact on the outputs of the item of the FMs affecting the output conditioning stage, it is necessary to obtain its input stimuli. Hence, a SPICE-level simulation of fault-free input conditioning stage, followed by a run of the embedded software, have to be performed. After that, SPICE-level simulations of the faulty output can be performed.

Finally, the failure mode effect classification is done as described in section 3.5.4.

3.5.4 Assessment of SW mitigation capabilities on HW failures

The discrimination between dangerous (leading to a violation of a safety goal) or safe (not leading to a violation of a safety goal) failure modes is based on the produced simulation outputs by comparing the fault-free simulation results with those obtained from the fault-affected one. If there is a discrepancy between the two results, the failure is classified as dangerous.

The simulation has to include the detection mechanisms (hardware and software) to perform the detection assessment.

It is possible to classify the item behavior in the presence of the considered failure modes as follows:

- if the failure mode triggered any of the detection mechanisms (range checks) the item embeds, and the item behavior does not lead to a violation of a safety goal (as derived during the HARA phase, see section 2.2.2), the failure mode is classified as safe undetected;
- if the failure mode is not detected, and the item behavior does not lead to a violation of a safety goal, the failure mode is classified as safe undetected;
- if the failure mode is detected, and the item behavior leads to a violation of a safety goal (even considering the embedded mitigation algorithms), the failure mode is classified as dangerous detected;
- if the failure mode is not detected, and the item behavior leads to a violation of a safety goal, the failure mode is classified as dangerous undetected.

The proposed approach can find out failure mode that propagates to errors and failures. Still, the approach requires running separate simulations of the input conditioning, processing, and output conditioning stage. It prevents the approach from being used when there are feedback signals.

Moreover, it is also difficult to provide accurate rules to perform the assessment different from a threshold of error between the fault-free and fault-affected simulation results.

It is important to remember that the objective is to determine if a failure mode can lead to a violation of one or more safety goals.

This approach, due to the missing integration between the SPICE-level simulator and the embedded software, cannot be used to assess systems with feedbacks between actuators and sensors, like closed-loop controllers. This makes it impossible to assess software mitigation capabilities on failures affecting output conditioning stages.

3.5.5 Experimental setup

This section provides a preliminary evaluation of the approach proposed in [15] when a simple item is considered.

The simulations of the conditioning stages have been performed by Analog Device LTSPICE, then its results about the input conditioning stage are moved to Simulink to run the embedded software.

Workloads

The workload used for the assessment consists of voltage waveform produced by the seat sensor to the weight input in three scenarios:

- no passenger seated;
- child seated of weight 30 kg;
- adult seated of weight 90 kg.

A model for each one of the considered FMs has been prepared and run for each workload.

3.5.6 Simulation results

The obtained results are reported in table 3.1, where for each component we reported:

- failure rate λ^f ;
- failure modes as from a failure mode catalog;
- failure mode rate of occurrence $f_{m.r.o}$;
- fault coverage $f_{coverage}$ defined as the percentage of failure of a specific failure mode that the functional safety mechanism is able to detect;
- residual contribution defined as $= FIT \cdot (1 - f_{coverage})$, (see eq.2.3 for the residual fault rate metric definition);
- failure mode effect classification.

Simulation time

The simulation time for performing the whole process is about 50 minutes when a desktop computer equipped with a 2.6 GHz Intel Core i7 processor and 16 GiB of RAM is used.

Hardware random failure metrics

The FME(D)A metrics obtained by the simulation results shown in table 3.1 for the considered item are:

- random hardware fault metric (see section 2.7.3): $rhf = 0.059 \text{ FIT} \Rightarrow \text{ASIL D}$
- single point fault metric (see section 2.7.3): $spm = 100\% \Rightarrow \text{ASIL D}$
- latent fault metric (see section 2.7.3): $lhf = 92\% \Rightarrow \text{ASIL D}$

As a result, comparing these with the table 2.7, we can conclude that this design is compliant with the ISO26262 requirements for an ASIL C design.

Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Failure mode effect	Fault coverage	Residual contribution [FIT]
R1	2.25	Open	50%	SD	100%	0
		Increase x2	25%	SU	0%	$5.625 \cdot 10^{-1}$
		Decrease x2	25%	SD	0%	$5.625 \cdot 10^{-1}$
C1	0.26	Interruption	40%	SU	0%	$1.040 \cdot 10^{-1}$
		Short circuit	10%	SU	0%	$2.600 \cdot 10^{-1}$
		Decrease x2	50%	SU	0%	$1.300 \cdot 10^{-1}$
R2	2.25	Open	50%	SD	100%	0
		Increase x2	25%	SU	0%	$5.625 \cdot 10^{-1}$
		Decrease x2	25%	SD	100%	0
U1	0.75	Interruption of any pin	50%	DD	100%	0
		Short of adjacent pins	50%	DD	100%	0
		Internal calculation error	50%	DD	90%	$2.950 \cdot 10^{-1}$
U2	0.59	Interruption of any pin	25%	DD	90%	$1.475 \cdot 10^{-2}$
		Short of adjacent pins	25%	DD	90%	$1.475 \cdot 10^{-2}$

Table 3.1: Simulation results obtained from [15].

3.6 Second proposal - A real automation of FMEDA

This section discusses the proposal described in two papers.

The first one has been presented at the 2018 edition of the International Symposium on On-Line Testing And Robust System Design (IOLTS) and published in its proceedings [16], while the second one at the 2019 International Conference of the IMACS TC1 Committee (ELECTRIMACS) conference and published in 2020, as a book chapter, in the Springer's Series Lecture Note on Electrical Engineering [18].

The tool (of both papers) operates as shown in fig.3.4.

The fault list generator module takes the BOM of the item and a fault catalog, containing the components' FIT, failure modes list, and failure modes rates of occurrence.

Combining the BOM and the fault catalog generates the hardware faults list for the considered item.

At this point, thanks to the (instrumented) schematic of the circuit, the tool simulates at SPICE-level firstly the item in fault-free (golden) conditions, and subsequently, it injects the failures one by one.

After each simulation, the classifier compares, by some set of classification rules, the simulation results with the golden ones and assign to each failure mode the related effects as described in fig.2.15.

As the last operation, it computes the metrics and generates a human-readable assessment report.

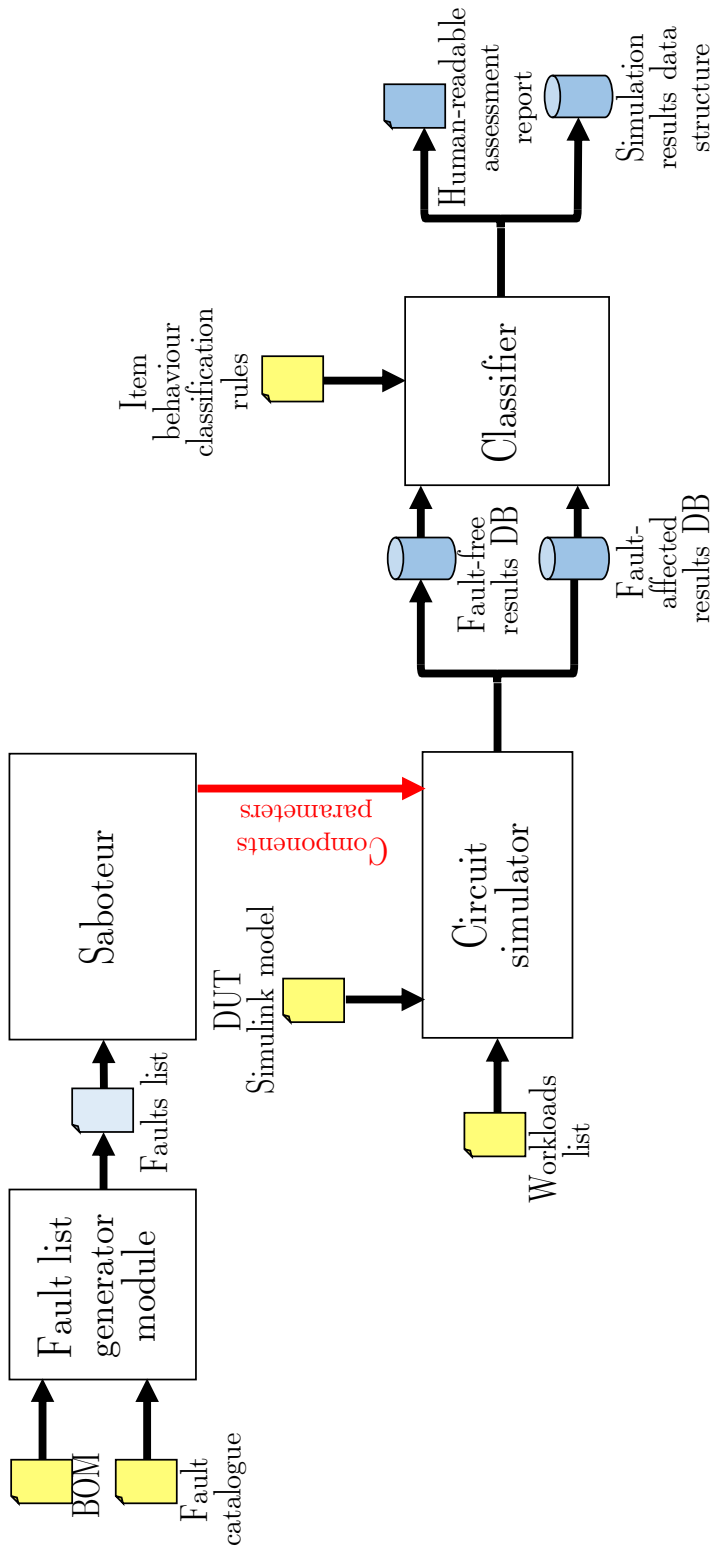


Figure 3.4: Schematic representation of the simulation-based approach proposed in [16]

To demonstrate the methodology, a benchmark item declined into two different versions has been implemented.

It is a control unit for an electric motor in charge to provide traction to a vehicle.

Its nominal functionality is the following.

The throttle pedal provides a voltage signal comprised between 0 and 1.2 Volts.

Its signal has to be amplified, with a gain $G = V_{out}/V_{in} = 4$, to the range 0-5 V to be acquired by an ADC integrated into the microcontroller.

The considered amplifier is an OP-AMP-based unit, implemented on the item PCB through some discrete components, one integrated circuit OP-AMP, and two resistors, as shown in fig.3.5¹⁰.

The ADC embedded into the microcontroller acquires the amplified analog signal

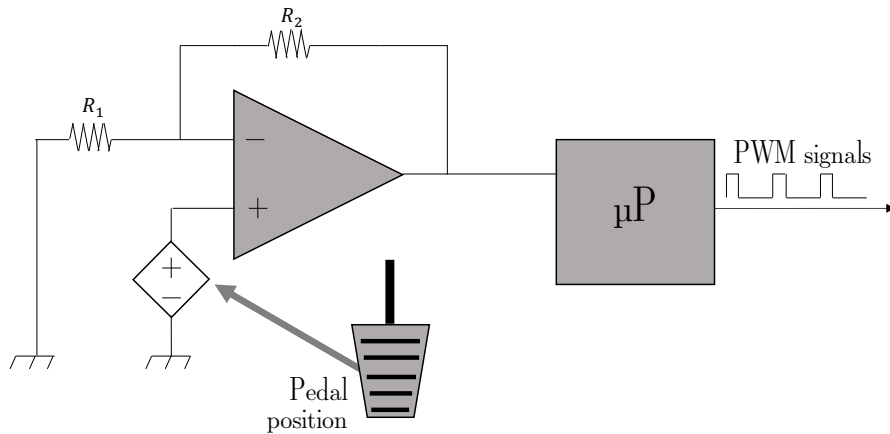


Figure 3.5: A schematic representation of the item described in [16].

and, in the case described in [16], it generates a PWM signal with a duty cycle proportional to the read voltage, while in [18] it provides three different duty cycles to drive the motor with a torque proportional to the pedal position.

By hypothesis in [16], due to the powertrain system frictions and to limits the vehicle acceleration, the minimum duty cycle request is 20%, while the maximum is 80%. If the request is outside this range, the embedded software stops the PWM signal generation.

To obtain an accurate simulation, a model to represent the actual PCB topology is needed. In this case, the layout of an evaluation board for an automotive-grade SoC has been considered. This 32-bit unit is shipped in a 100 pin Low Profile Quad Flat Package (LQPF). The throttle pedal, the inverter, and the motor itself are considered outside the item.

¹⁰In electronics literature, this configuration is called a non-inverting voltage amplifier. Its gain can be computed as in eq.3.1

In [18] simulations are performed in a similar way as in [16] but, in this case, inverter and motor models are alongside the embedded software and a new high-dependable fail-operational input conditioning stage, composed of three independent channels read by separate ADCs, as shown in fig.3.6.

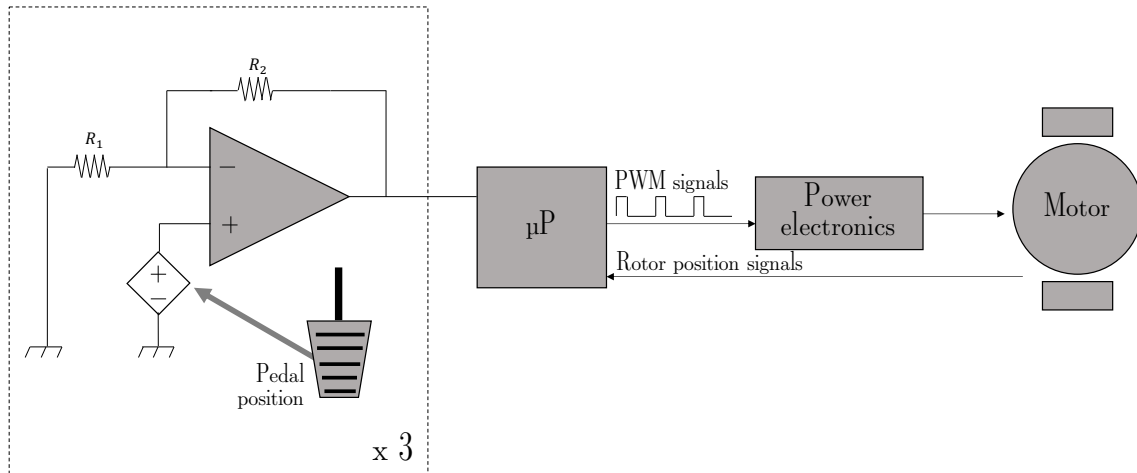


Figure 3.6: A schematic representation of the item described in [18].

These two papers are considered as containing a single proposal since [18] is an improved version of [16].

In particular [18] w.r.t. [16]:

- reports a handmade HARA of the considered item;
- the considered item contains a triple redundancy module (TRM) to improve the reliability of the input conditioning state;
- the item embeds failure detection and mitigation algorithms;
- a simplified model is adopted (for the IGBTs) to tradeoff between simulation quality and required time;
- a model of the driven electrical motor is inserted into the simulation;
- a subset of the inverter components is simulated to assess the implemented detection algorithms.

As described in the following, thanks to the presence of the TRM and the detection and mitigation algorithms, it is possible to see how the random hardware failure rate metrics required by the ISO26262 FMEDA (see section 2.7.3) changes between the two items.

The considered safety goals (SGs) for the item (electrical powertrain) are [18]:

- **SG1**: the motor torque (and speed outside transients) shall correspond to the one requested by the driver by pushing the throttle pedal.
Its most severe violation can be summarized by a condition where the motor torque is zero when a non-zero action is requested (no motor rotation when requested).
- **SG2**: the motor torque when the throttle pedal is completely released shall be zero (or correspond only in a regenerative braking action).
The most severe violation of this safety goal can be described as an unintended vehicle acceleration.

As described in section 2.2, each one of the safety goals must be associated with an ASIL level.

As in [18], an ASIL B is assigned to **SG1**, while an ASIL D¹¹ to **SG2**.

3.6.1 Fault models

Electrical parameter changes

The simplest model to obtain, since it does not require the addition of sabotaging components inside the schematic, consists of modifying the nominal electrical parameter of the affected component according to the FM to be injected. For example, the saboteur can inject a change in the resistance of a resistor to simulate its detachment from the PCB or a short between its pins.

These *electrical parameter changes* models are considered, inside the IEEE P2427 framework, as *parametric defect*.

Simulation-only components

Unfortunately, in some cases, the previous approach is not sufficient to adequately describe the FM; hence it is necessary to add components not present in the design to instrument the simulation with the faulty behavior. For example, to simulate a short circuit between the plates of a capacitor, it is not sufficient to change its capacitance, but it is necessary to add a resistor in parallel to it. A similar approach can be useful to simulate some failures affecting integrated circuits, as short circuits between adjacent pins, that can be simulated by adding resistors between the pins. In this case, we need that the model represents the physical layout of the package. In fault-free conditions, these resistors have a resistance value set to $10^8 \Omega$, while to obtain a short circuit, this value is lowered to about 1Ω . To

¹¹Usually this SG is considered ASIL C by adding assumptions on the mechanical braking system of the vehicle, that allows the driver to reduce the acceleration of the vehicle, and influencing the *controllability* risk parameter.

speed-up the simulation process, it is possible to avoid injecting those failures in which the adjacent pin is unused and configured in a high-impedance state. These *simulation-only* models are considered, inside the IEEE P2427 framework, as *catastrophic defect*. As long as short circuits between unconnected nodes are injected, *topological catastrophic defects* are injected.

Stuck-at simulation-only sabotaging component

A third approach, done by adding a sabotaging component, designed for this specific purpose, between the IC and the rest of the schematics, is applied to simulate permanent stuck-at failures of pins. They are implemented as Simulink sublayers composed of different switches, able to connect the affected pin to the ground or to a controlled voltage generator..

Inverter IGBTs

In [18], also a model to represent FMs affecting the IGBTs of the inverter have been considered.

As described in the literature, a complete and exhaustive failure model is challenging to obtain and cannot cover all possible scenarios.

In any case, all these efforts are not required for this proposal, since its objective is to reproduce the damage of the IGBTs without investigating the boundary conditions which causes the failure (i.e., overcurrent conditions, breakdown of the junctions, ...), but to study their failure modes effects on the behaviors of the item. The interested reader can find a more accurate fault model in section 3.9.2, fig.3.28, and [46].

To implement the model considered in this proposal, the SimScape MOSFET block¹² has been used. This drastically reduces the complexity of the fault model and simulation times. Moreover, designers can easily implement it by adding stuck-at saboteurs between the microcontroller pins in charge of generating the firing signals and the MOSFET gate, which can be set by the *sabouter* at any time during the simulation.

Since this proposal uses the IGBTs simplified fault models with respect to the state of the art [47], the rates of occurrence for the FMs of the microcontroller pin (stuck-at) and IGBT have been combined. It leads to a failure mode rate of occurrence of 86% for the stuck-at open circuit and the remaining 14% for the stuck-at close one. A schematic representation of an inverter is shown in fig.3.7.

For the IGBTs are considered only *catastrophic* failure.

¹²provided into the SimScape ToolBox.

The simulations have been performed with version 2016a of MATLAB/Simulink. MathWorks may change the toolbox's name in more recent versions.

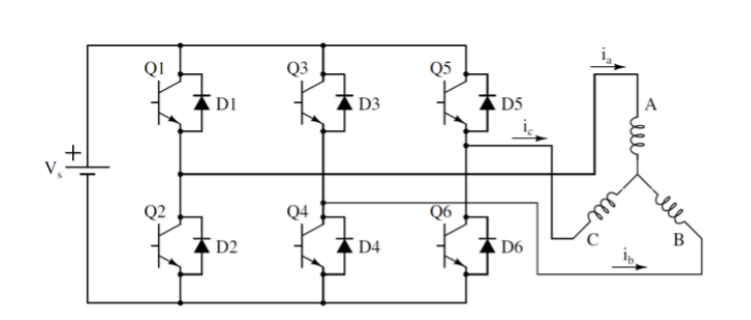


Figure 3.7: Schematic representation of an inverter connected with a with a star winding brushless direct current (BLDC) permanent magnet motor.

3.6.2 Fault injection

The saboteur injects one at a time the failure modes into the system by changing electrical parameters and properly managing the sabotaging and simulation-only components.

Change of nominal parameters

The fault injection strategy for changing nominal parameters consists of letting Simulink get their values, assigned by the saboteur, from the MATLAB workspace. This strategy is used to simulate failures affecting the feedback network of the OP-AMP in the analog conditioning stage needed to acquire the throttle pedal position in both the papers [16], and [18], as shown in fig.3.8.

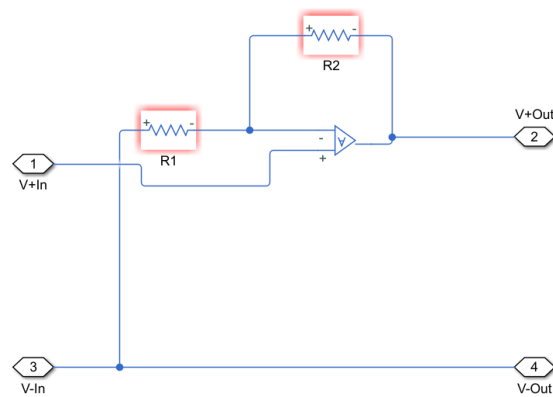


Figure 3.8: The schematic of the analog conditioning stage described in [16]. The red borders around the resistors R_1 and R_2 indicates the absence of a defined parameter for their resistance values, since it is computed at runtime by the saboteur.

Simulation-only components

When necessary, as described in fault models section 3.6.1, it is required to add by hands components not present in the original design to inject the behavior needed to simulate the failure mode. Typical use is to describe short circuits between the plates of capacitors or pins of ICs, as shown in fig.3.11.

Sabotaging blocks

Sabotaging blocks (SBs) are simulation-only components designed for a specific purpose. In this proposal, they are used to inject stuck-at FMs. The SB used in this proposal is shown, colored in magenta, in the fig.3.9. Fig.3.9 shows the interfaces of the SB, while fig.3.10 its implementation, composed of switches to change its configuration between normal (pass-through), stuck-at high level (VCC), or stuck-at low level (GND).

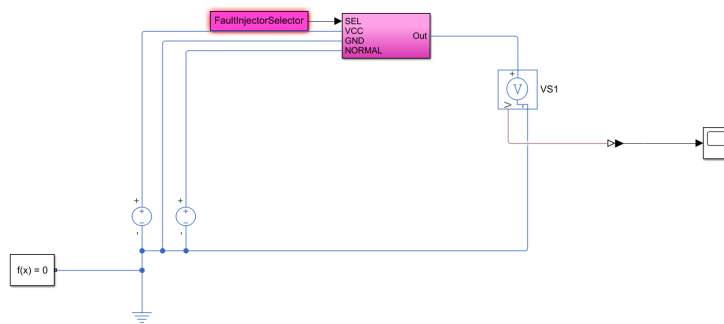


Figure 3.9: The external interfaces of the SB used to inject stuck-at FMs in [16] and [18].

The red borders around the `FaultInjectorSelector` constants indicates the absence of a defined value, since it is computed runtime by the sabouter.

A combined use of *added components* and *sabotaging components* is shown in fig.3.11.

3.6.3 Failure modes effects assessment strategies

For the sake of risk assessment, are considered as dangerous those situations where the motor provides a wrong angular speed or torque¹³, while the motor into zero torque condition is considered as a safe one.

¹³The paper [16] considers a wrong angular speed proportional to the duty cycle. In contrast, in [18], thanks to the presence of a model of the motor and a closed-loop control algorithm (composed of three nested loops, with the inner one controlling the currents, the other the torque,

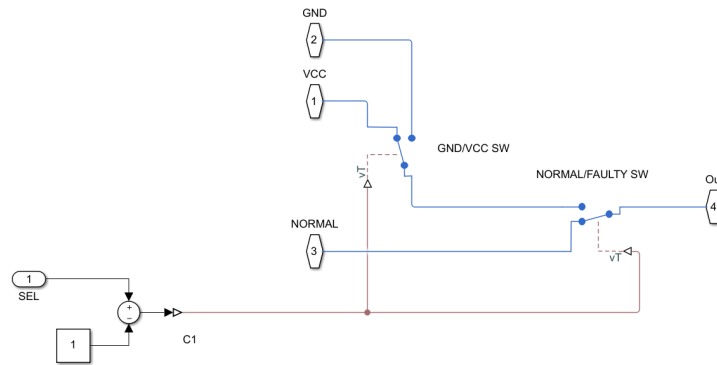


Figure 3.10: The implementation of the SB used to inject stuck-at FMs in [16] and [18].

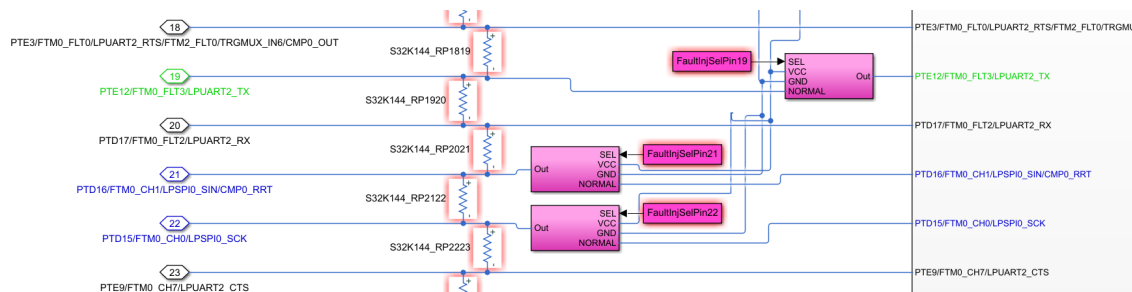


Figure 3.11: Examples of SBs, used to inject stuck-at FMs on pins, and simulation-only components (in this case resistors) to inject short circuits between adjacent pins of a commercial microcontroller.

As in the first case, the detected/undetected classification is obtained from the results from simulations itself that runs the embedded software, but in this case, thanks to the redundancies on the input conditioning stages, the software demonstrated the ability to detect and mitigate the failures, avoiding the presence of dangerous undetected or safe undetected failures on the input conditioning stage. The detection algorithm for the failures affecting the IGBTs is shown in fig.3.12.

and the most external the speed) can control the electrical machine both with a speed or a torque setpoint.

3.6.4 Assessment of the SW mitigation capabilities on HW failures

In the case of [16], the failure mode effects classification in terms of safe/unsafe (see fig.2.15) can be done by comparing the system outputs with the expected ones, obtained from a set of rules or by comparing the system outputs in fault-free (golden condition) with the ones obtained after the failure injection. Instead, the detected/undetected classification is obtained from the simulation results that run the embedded software and contains the failure detection system as part of the item.

On the other hand, in the [18] paper, the safe/unsafe classification is performed by propagating the computed firing signals to the inverter switches, hence to the motor model. The item behavior (propagated on the motor, so motor speed and torque) are stored in databases and compared against the item behavior classification rules.

The algorithm to detect failure modes, and triggers the mitigation measures, is shown in fig.3.12.

The script contained in the `SwitchFailureFromCurrentsDetection` functions is:

```
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
```

3.6.5 Experimental setup

The proposed approach has been implemented through an FME(D)A automation tool. It is fully implemented in the MATLAB/Simulink environment. The schematics of the item under assessment is modeled through the Simulink SimScape Toolbox, while the item software is modeled as a Simulink Matlab Function if it is handwritten code, or by Simulink, if a Model-Based Software Design is adopted¹⁴. *Fault list generator*, *saboteur*, and *classifier* modules are implemented as MATLAB functions. In this way, it is possible to obtain a unique executable model that captures the relevant characteristics of both hardware and software.

¹⁴Model-In-the-Loop (MIL), as described in section 2.11.4.

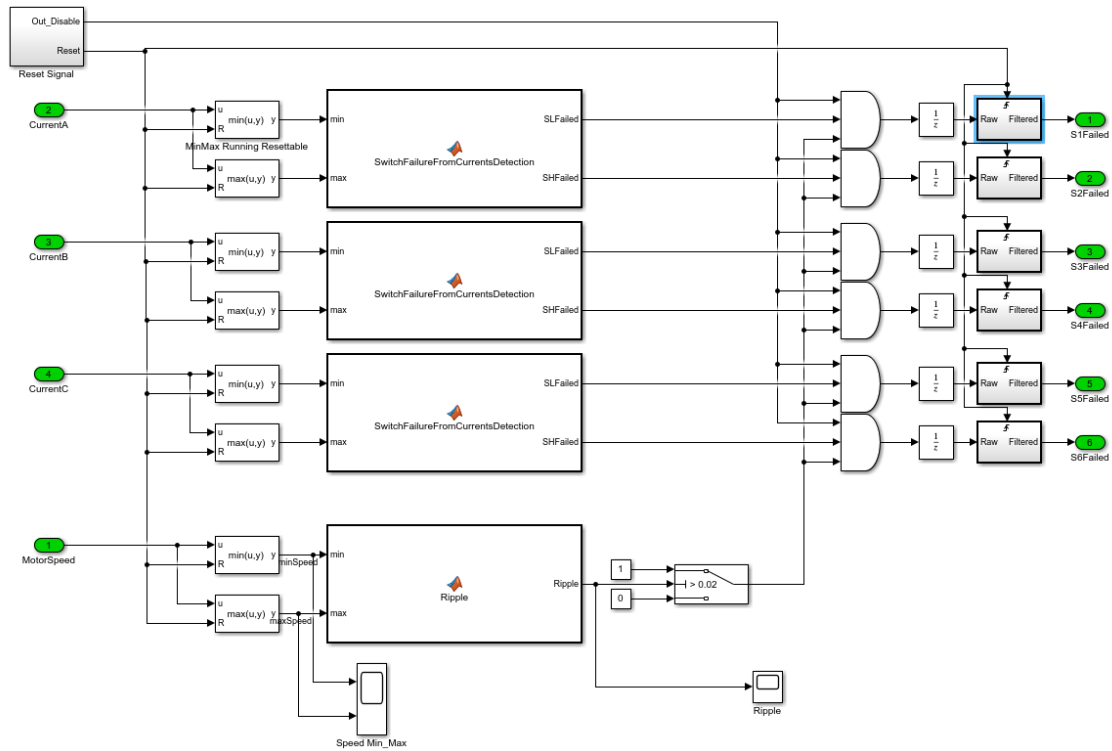


Figure 3.12: The implementation of the detection algorithm for IGBTs in the [18] paper.

In this version of the tool, if needed to add components to instrument the model to simulate some failure modes, these sabotaging components, described in section 3.6.2, have to be added to the BOM.

Simulation runs as shown in fig.3.4, then the results are stored inside a MATLAB array of structures.

Each structure into the array of results corresponds with a workload.

The top-level of each one of these structures has as many rows as the number of BOM components. Each row contains the component name, nominal value, class, FIT, and a second-level structure, with as many rows as the failure modes of the considered components. They contain the failure mode probability, the value set into the component to inject the failure modes itself and the fault coverage, and the fields Safe, Detected, and Residual contribution filled at run-time by the classifier.

Thanks to this structure, when simulations with more than one workload are done, the tool can compute the metrics and assess each workload. After that, it combines all the evaluations in a summary report that allows designers to consider all the worst conditions found in the different workloads.

For each row, the worst-case failure effect is selected, in descending order of severity, as dangerous undetected (DU), safe undetected (SU), dangerous detected (DD), safe

detected (SD).

SUs are considered more critical w.r.t. the DDs because the FMs are assessed one at a time, so the simulation cannot provide information on what happens if two or more FMs are present simultaneously.

In any case, the presence of SUs is discouraged by part 5 of ISO 26262 since they increase the latent fault metric of the design as described in section 2.7.3.

3.6.6 Simulation results for [16]

The FMEDA metrics obtained by the simulation results shown in table 3.2 for the considered item are:

- random hardware fault metric (see section 2.7.3): $rhf = 7.75 \text{ FIT} \Rightarrow \text{ASIL D}$
- single point fault metric (see section 2.7.3): $spfm = 6.06\% \Rightarrow \text{QM}$
- latent fault metric (see section 2.7.3): $lhf = 93.94\% \Rightarrow \text{ASIL D}$

To obtain the simulation result presented in the table, it required about 40 minutes simulating a notebook based on a 2,6 GHz Intel Core i7 (6th gen) CPU.

Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Failure mode effect	Fault coverage	Residual contribution [FIT]
R1	0.75	open	84.0 %	DU	100.0 %	1.89
		increase	8.0 %	DU	100.0 %	0.18
		decrease	8.0 %	DU	100.0 %	0.18
R2	0.75	open	84.0 %	DD	100.0 %	0.00
		increase	8.0 %	DU	100.0 %	0.18
		decrease	8.0 %	DU	100.0 %	0.18
U1	0.75	Interruption of any pin	50.0 %	DD	100.0 %	0.00
		Short of adjacent pins	50.0 %	DD	100.0 %	0.00
FaultInjSelPinPWM	0.75	StuckAtVCC	50.0 %	DU	100.0 %	0.50
		StuckAtGND	50.0 %	DU	100.0 %	0.50
FaultInjSelPinInput	0.75	StuckAtVCC	50.0 %	DD	100.0 %	0.00
		StuckAtGND	50.0 %	DD	100.0 %	0.00
FaultInjSelPowerSupply	0.75	StuckAtVCC	50.0 %	SU	100.0 %	0.50
		StuckAtGND	50.0 %	DU	100.0 %	0.50

Table 3.2: FMEDA assessment results obtained by the tool presented in [16] considering 5 different workloads.

The simulation results show that the metrics are not compatible with the ASIL D metrics requirements (see table 2.7) since as many as 8 failures over the 14 considered lead by themselves to a safety goal violation (in this simplified item, a wrong angular speed provided by the motor) hence are classified as Dangerous Undetected (DU)ones.

By reasoning on these, we can observe that 5 of them are related to the resistors R1 and R2 of the input conditioning stage. Since these resistors compose the feedback network of the conditioning stage amplifier, which gain G is described by the eq.3.1

$$G = \frac{V_{out}}{V_{in}} = 1 + \frac{R_2}{R_1} \quad (3.1)$$

their FMS lead to a change of G . This causes a corruption of the duty cycle request voltage measured by the ADC of the SoC, causing the processor to generate a PWM signal with an erroneous duty cycle. This wrong duty cycle leads the motor to provide a wrong torque level, hence speed.

It highlights that the range check failure detection mechanism implemented in this design is not able to detect failures in the input condition stage in all the possible workload conditions.

Another two failure conditions are related to the pin to which is connected the half-bridge since, in these conditions, the microcontroller cannot control the motor at all. The last DU condition is related to a failure in the power supply that causes a shutdown of the complete set of components installed on the PCB¹⁵.

3.6.7 Simulation results for [18]

The FMEDA metrics obtained by the simulation results shown in table 3.3 for the considered item are:

- random hardware fault metric (see section 2.7.3): $rhf = 1.21 \text{ FIT} \Rightarrow \text{ASIL D}$
- single point fault metric (see section 2.7.3): $spfm = 99\% \Rightarrow \text{ASIL D}$, with random HW failure rate of the power supply unit, threated as a SEooC (see section 2.2.3), $\leq 0.45 \text{ FIT}$.
- latent fault metric (see section 2.7.3): $lhf = 100\% \Rightarrow \text{ASIL D}$

¹⁵From a rigorous point of view, since the zero torque is considered a safe state, these last three failures are Safe Undetected. But, since a simplified classification rule has been implemented, in the term *the microcontroller provides a wrong duty cycle*, these failure modes have been classified as DU.

Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Failure mode effect	Fault coverage
R1 (unit 1)	2.25	open	84%	SD	100%
		increase	8%	SD	100%
R2 (unit 1)	2.25	decrease	8%	SD	100%
		open	84%	SD	100%
OP-AMP (unit 1)	0.75	increase	8%	SD	100%
		decrease	8%	SD	100%
R1 (unit 2)	2.25	Interruption of any pin	50%	SD	100%
		Short of adjacent pins	50%	SD	100%
R2 (unit 2)	2.25	open	84%	SD	100%
		increase	8%	SD	100%
OP-AMP (unit 2)	0.75	decrease	8%	SD	100%
		Interruption of any pin	50%	SD	100%
R1 (unit 3)	2.25	Short of adjacent pins	50%	SD	100%
		open	84%	SD	100%
R2 (unit 3)	2.25	increase	8%	SD	100%
		decrease	8%	SD	100%

Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Failure mode effect	Fault coverage
OP-AMP (unit 3)	0.75	Interruption of any pin	50%	SD	100%
		Short of adjacent pins	50%	SD	100%
AnalogGasPedalCh1	0.75	StuckAtVCC	50%	SD	100%
		StuckAtGND	50%	SD	100%
AnalogGasPedalCh2	0.75	StuckAtVCC	50%	SD	100%
		StuckAtGND	50%	SD	100%
AnalogGasPedalCh3	0.75	StuckAtVCC	50%	SD	100%
		StuckAtGND	50%	SD	100%
MotorEncoderCh1	0.75	StuckAtVCC	50%	SD	100%
		StuckAtGND	50%	SD	100%
MotorEncoderCh2	0.75	StuckAtVCC	50%	SD	100%
		StuckAtGND	50%	SD	100%
MotorEncoderCh3	0.75	StuckAtVCC	50%	SD	100%
		StuckAtGND	50%	SD	100%
FaultInjSelPowerSupply ¹⁶		StuckAtVCC	50%	DU	100%
		StuckAtGND	50%	DU	100%

Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Failure mode effect	Fault coverage
Switch1	1.05	Stuck at open condition	86%	SD	100%
		Stuck at closed condition	14%	DD	100%
Switch2	1.05	Stuck at open condition	86%	SD	100%
		Stuck at closed condition	14%	DD	100%
Switch3	1.05	Stuck at open condition	86%	SD	100%
		Stuck at closed condition	14%	DD	100%
Switch4	1.05	Stuck at open condition	86%	SD	100%
		Stuck at closed condition	14%	DD	100%
Switch5	1.05	Stuck at open condition	86%	SD	100%
		Stuck at closed condition	14%	DD	100%
Switch6	1.05	Stuck at open condition	86%	SD	100%
		Stuck at closed condition	14%	DD	100%

Table 3.3: FMEDA assessment results obtained by the tool presented in [18].

3.7 Third proposal - Handmade vs. simulation-based approaches

The paper [17] has been published in 2018 at the same time as the paper [16] described in the second proposal. Its goal is to assess the effectiveness of the proposed approach by comparing, on an industrial case, the classifications obtained with the simulation-based methodology against the ones handmade by experts.

The methodology is described in the second proposal [16] and represented in fig.3.4.

The considered item, provided by a company, is a monitoring circuit that has to check the power consumption of the video interface of the camera used by the driving algorithms of an autonomous driving car. If the measured power consumption is outside the expected range, it has to detect the failure.

At the end of the HARA, the worst violation of a safety goal due to a missing functionality provided by the video interface has been classified as ASIL D; hence the monitoring device, analyzed in this proposal, shall satisfy the requirements for ASIL B items (see *control mechanisms for latent faults* in section 2.5).

The item schematic, represented inside the MATLAB/Simulink environment, is shown in fig.3.13.

It is possible to see the presence of the *simulation-only component* (see section 3.6.1) C_{1R} to simulate a short circuit between the capacitor plates.

3.7.1 Fault models

The fault models are described in section 3.6.1.

3.7.2 Fault injection

The fault injection is performed as described in section 3.6.2.

3.7.3 Failure modes effects assessment strategies

The safe/dangerous classification was based on comparisons with the results obtained in fault-free conditions. The FM is considered as Safe if:

- the output signal produced by the faulty circuitry is within a tolerance of 5% from that produced by the fault-free one;
- the output current flowing into the OP-AMP is not more than double of the one measured in fault-free conditions.

If these conditions are not met, the FM is considered as Dangerous.

3.7.4 Assessment of the SW mitigation capabilities on HW failures

There are no implemented mitigation capabilities in this particular setup since there are no failure detection mechanisms (this item is a monitoring system).

3.7.5 Experimental setup

The tool in the one described in [16], already described in section 3.6. Its architecture is shown in fig.3.4.

The Simulink model of the item is shown in fig.3.14, while in fig.3.13 it is represented the analog circuit schematic represented in Simulink (corresponding to the content of the green sublayer shown in fig.3.14).

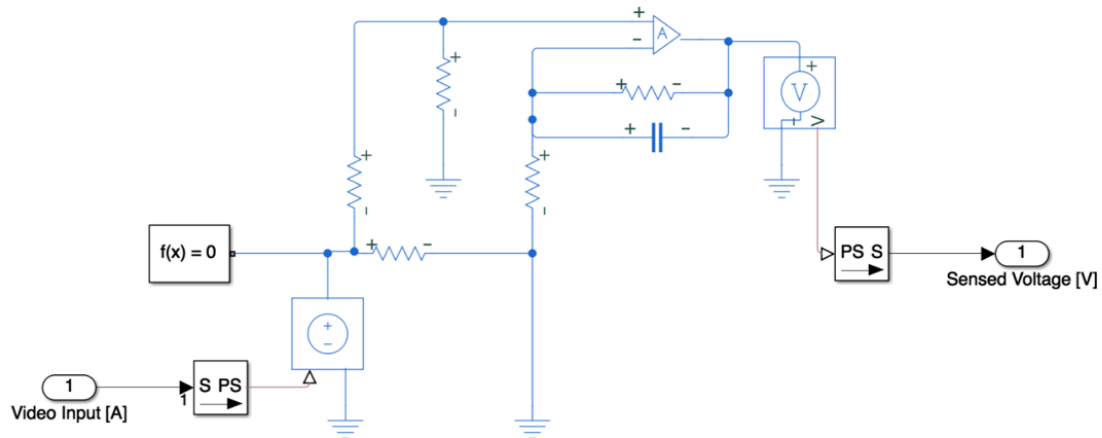


Figure 3.13: Schematic representation of the simulation-based approach proposed in [17]

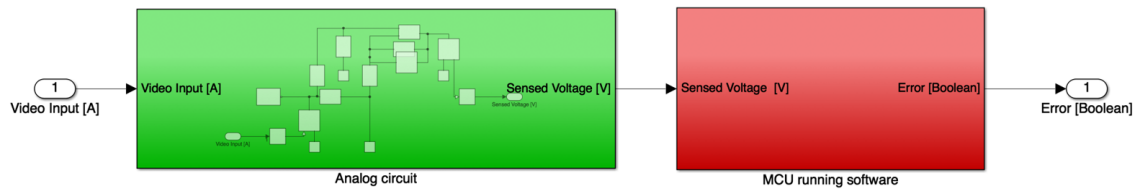


Figure 3.14: Simulink model to perform simulations described in [17]

3.7.6 Simulation results

The hardware failure metrics (see section 2.7.3) obtained from both the automatic tool and the experts' analysis are reported in table 3.4 while the classification for each of the failure modes are written in the table 3.6. For the handmade ones, the experts reported their motivations.

Metric	Sim-based	Handmade
Random hardware fault metric [FIT]	9.9	10.1
Single point fault rate	22%	19%
Latent fault rate	78%	76%

Table 3.4: FMEDA assessment result comparison between the handmade and the automatically performed one.

By comparing the FMs classifications (see section 2.15) table, it is possible to observe that 11 of 18 FMs (the 5 ones related to the two ICs are assessed by hand) obtained the same classification in both handmade and simulation-based FME(D)A.

To better analyze the disagreements, a comparison is shown in the form of the contingency table 3.5. It reports in horizontal the count of classifications obtained from the simulation-based analysis, while in vertical the experts' ones.

There are 2 cases where the sim-based approach has classified the failure mode as an SU, while the experts as DU, and 5 cases for the vice-versa. On the diagonal, as in every contingency table, the cases in which the classification obtained in both the classifications are the same.

Summarizing the results, the simulation-based approach:

- agreed with the experts 61% of times;
- considers as dangerous conditions classified as safe by the experts in 28% of the cases (the tool is more conservative w.r.t experts);
- considers as safe conditions classified as dangerous by the experts in 11% of the cases (these are more problematic since these classifications are less conservative).

By better analyzing the less conservative classification¹⁷, it is possible to observe that:

- in 1 (6%) case the classification rules cannot detect that the monitoring system cannot properly work when affected by the considered FM.

¹⁷Less conservative classifications are those involving FMs classified as DU by the experts and as SU by the simulation-based methodology.

- in another case (6%), the simulation cannot reach those conditions in which the monitoring circuits generate a current readout lower than the real one.

It is possible to say that there is room to improve the failure modes classification since rules based only on comparisons with the fault-free simulation disagree in 39% with the experts' ones¹⁸.

Sim-based Handmade	SD	SU	DD	DU
SD	0	0	0	0
SU	0	2 (equal) [11 %]	0	5 (worst) [28 %]
DD	0	0	0	0
DU	0	2 (better) [11 %]	0	9 (equal) [50 %]

Table 3.5: Comparison between the failure classifications obtained by the automatic tool and from the experts.

¹⁸In this paper the results obtained by the experts are considered as the ground truth so, in this analysis, it is not possible to question the results obtained by the experts. Moreover, to avoid biasing the results, the authors of [17], only know about the item: its schematic, the HARA results, and the textual description of its function as reported in this section.

Component	Failure rate [FIT]	Failure mode	Failure mode of occurrence	Sim-based classification	Handmade classification	Experts' motivation
R1	2.23	open	50.00%	DU	DU	Current value not available
		increase	25.00%	DU	DU	Lower value detected
		decrease	25.00%	DU	SU	Higher value detected
R2	2.23	open	50.00%	DU	DU	Lower value detected
		increase	25.00%	DU	SU	Higher value detected
		decrease	25.00%	DU	DU	Lower value detected
R3	2.23	open	50.00%	SU	DU	Current value not available
		increase	25.00%	SU	DU	Lower value detected
		decrease	25.00%	SU	SU	Higher value detected
R4	2.23	open	50.00%	DU	DU	Lower value detected
		increase	25.00%	DU	SU	Higher value detected
		decrease	25.00%	DU	DU	Lower value detected
R5	2.23	open	50.00%	DU	DU	Current value not available
		increase	25.00%	SU	SU	Higher value detected
		decrease	25.00%	DU	DU	Lower value detected
C1	2.23	interruption	40.00%	DU	SU	Current filter not available
		short circuit ¹⁹	10.00%	DU	DU	Lower value detected
		decrease	50.00%	DU	SU	System not available

Component	Failure rate [FIT]	Failure mode	Failure mode rate of occurrence	Sim-based classification	Handmade classification	Experts' motivation
U1	7.51	Interruption of any pin	50.00%	DU	DU	Current value not available
		Short of adjacent pins	50.00%	DU	DU	Lower value detected
U2	5.94	Internal calculation error	50.00%	DD	DD	Lower value detected
		Interruption of any pin	25.00%	DU	DU	Lower value detected
		Short of adjacent pins	25.00%	DU	DU	Lower value detected

Table 3.6: Comparison between the handmade and the automatic assessments. The differences between the two classifications are highlighted.

3.8 Fourth proposal - From an item-level FMs classifications to a vehicle-level one

The main contributions of this paper [19], based on the papers [15], presented in the first proposal, described in the section 3.5, [16], and [18], presented in the second proposal described in the section 3.6, are:

- to propagate the FMs effects to the entire vehicle, assessing their impact on its drivability;
- to allow the assessment of the embedded software mitigation capabilities, at the vehicle level.

The methodology proceeds as follows.

It starts with the bill of material (BOM) and a fault catalog. FITs are computed by hands following the FIDES methodology as described in section 2.8.

Combining these two documents makes it possible to obtain the failure modes list to be used by the saboteur to inject the faults during the simulations.

The *structural*-level simulator takes the SPICE-level model of the item, that can be instrumented with simulation-only components as described in section 3.6.1.

At this point, the vehicle-level simulator loads the scenario required for the simulation and starts its simulation. During this phase, the *structural*-level simulator interacts with the vehicle-level simulator, taking the input signals from the latter and generating the outputs for the actuators, closing the control loops the item is in charge of.

At the beginning of each simulation (or during a defined simulation time), the saboteur injects the FM into the affected component. The silicon-level faults that could affect the microcontroller are not considered in this proposal since modern automotive-grade MCUs integrate failure detection and mitigation mechanisms [48] [49], hence are considered as SEooC (see section 2.2.3).

At the end of each simulation, its results are stored and then classified according to classification rules.

To apply the proposal, designers needs:

- the embedded software of the item;
- the structural²⁰ models of the item;
- physical models of the car and the surrounding environment, provided by a commercial vehicle-level simulator;

²⁰In [19] it is called *physical* model. In this dissertation, I prefer the name *structural* to be coherent with the names reported in the article published in 2020 a few months after this [40] and analyzed as the fifth proposal in section 3.9.

- scenarios in which test the failure effects;
- vehicle behavior classification rules.

The benchmark application is a transaxle for an electric vehicle. The considered vehicle is a rear-traction sedan car, with an independent motor for each of the two rear wheels. There is no mechanical differential gear between the two motors; hence the control software has to emulate its behavior to allow the vehicle to drive on curves and properly manage all the possible FMs that can prevent the motors from generating the right amount of torque since a torque disparity can make the vehicle turning against the driver's will, causing safety issues.

The block diagram of the considered item is shown in fig. [3.16](#).

The whole methodology is represented in fig.[3.15](#).

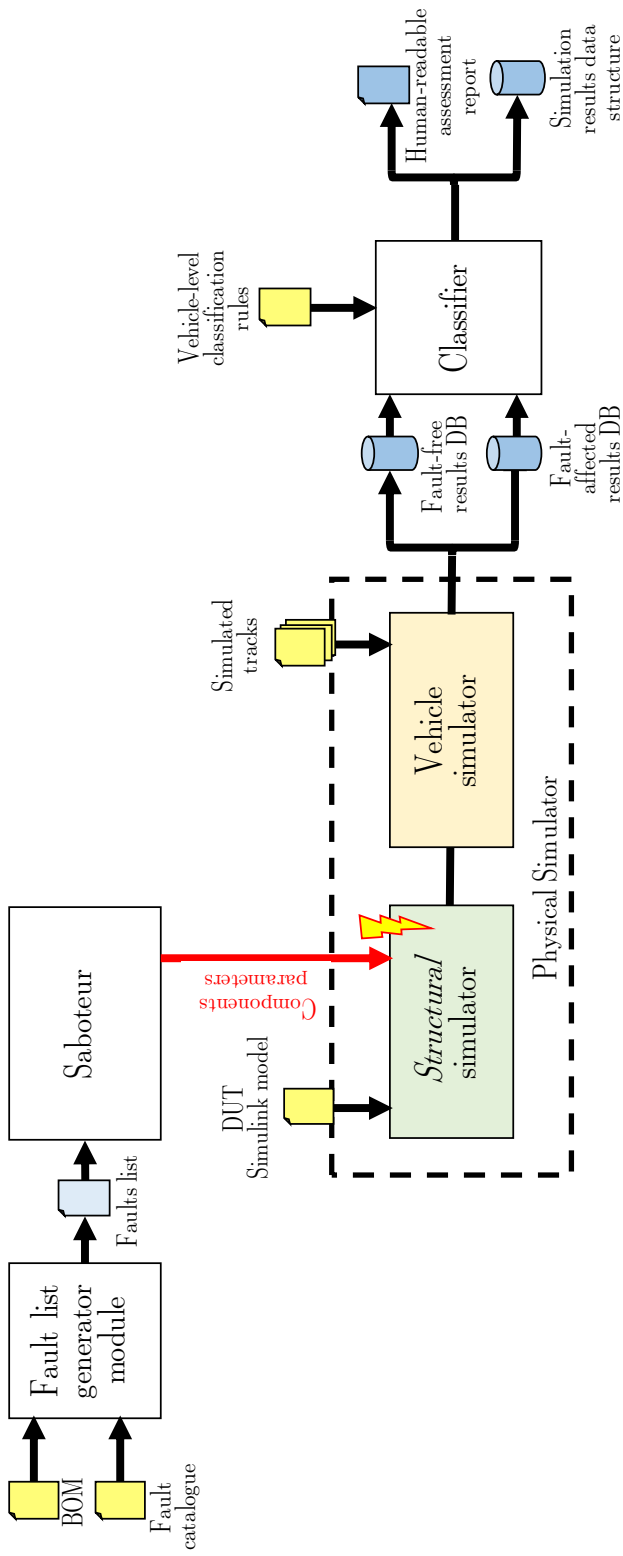


Figure 3.15: The block diagram of the proposal. Figure adapted from [19].

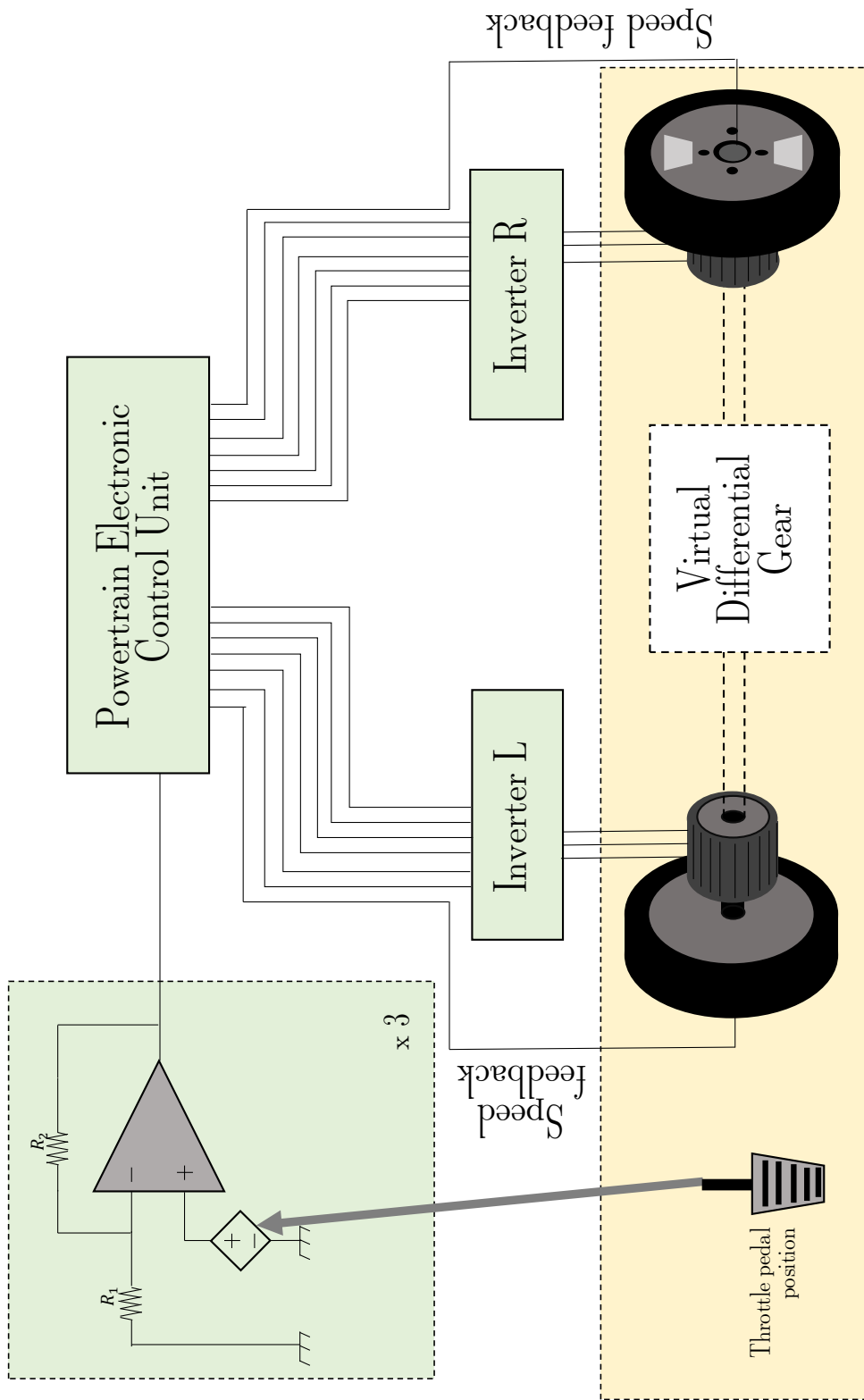


Figure 3.16: The block diagram of the benchmark application.

3.8.1 Fault models

The fault models are described in section 3.6.1.

For the sake of this proposal, have been considered only the faults that could affect the analog components installed on the PCBs of the Powertrain Electronic Control Unit and of the two inverters (see fig.3.16). The faults affecting both the wirings between the boards and the conductive tracks of the PCBs themselves have not to be considered.

3.8.2 Fault injection

The fault injection is performed as described in section 3.6.2.

A limitation of this proposal is the lack of fault injection, possible only at the vehicle-level, on sensors and electromechanical actuators. Even if these components are considered outside the item, they can prevent it from providing the required functionalities. How to inject and assess these failures is described in the sixth proposal (see section 3.10).

3.8.3 Failure modes effects assessment strategies

There are 68 failure modes for the considered item. Of these, 30 regards the throttle pedal position acquisition chain circuitry (the assessments of the FMs affecting each one of the three identical models, without redundancies, are described in section 3.6.6), 2 the power supply, and the remaining 36 are about the two twin motor driver chains. Hence, for each of the motors, we have 18 possible failure modes. 6 of them regard the triple redundancy encoders installed to monitor the speed of the wheels, while the remaining 12 the power electronics.

Each inverter is composed of 6 IGBTs with two stuck-at FMs: open (no current between its collector and emitter) or closed (short circuit between its collector and emitter). Since in the case of a stuck-at closed failure (leading to a short circuit of the affected inverter leg), no software mitigations are possible (the protection fuse melts down, permanently disconnecting the involved phase from the battery) only the open failure mode has been injected on an IGBT of the left motor. This single injection is sufficient to cover all the 12 possible cases due to the symmetries of the considered system.

The IGBTs failure detection algorithm, applied at the single inverter level, is the same described in section 3.6.3 and its semi-formal Simulink model is shown in fig.3.12. It has to trigger the mitigation algorithm described in the following.

No assessments on the throttle pedal - microcontroller chains have been analyzed in this proposal since yet assessed in the second proposal as described in the section 3.6.

As already said, the key point of the proposed approach regards how to improve the evaluation of the FMs' effects on vehicle drivability.

Since, in the analyzed system, a disparity in the torque due to a FM affecting only one of the motors could cause a sudden car turn. The embedded software has to be able to *vector* (similarly, the pilot can trim an aircraft rudder on dual-engine aircraft) automatically the torque to avoid this effect.

The risk level associated with a vehicle function is determined, keeping into account the capability of an average driver (*controllability*) to mitigate the failure effect. For this reason, the driver has been represented, inside the vehicle-level simulation, like a PID controller with a delay to keep into account the human reaction time. Its target behavior is represented by a predetermined trajectory to be followed on the track.

In each of the considered driving situations, two parameters have been analyzed: the lateral distance from the ideal centerline (called in the following *lateral error*) and the difference in the yaw angle between the fault-free and fault-affected ride.

3.8.4 Assessment of the SW mitigation capabilities on HW failures

The fail-operational vehicle-level mitigation strategy is based on the following assumption: the motor driven by the inverter with a failed IGBT cannot provide the full torque it is expected to produce. Hence, since only one of the two motors is affected by this FM²¹, and the asymmetrical torque causes dangerous situations, it is necessary to intervene on the fault-free motor. The idea adopted in this benchmark application is to threshold its speed setpoint up to the fault-affected speed to limit the torque disparity on the wheels.

A semi-formal representation of the mitigation algorithm is shown in fig.3.17

`UpperLimit` and `LowerLimit` signals are equal to the speed of the fault-affected motor, `Detection` comes from the detection algorithm, and `NormalReference` is the speed request from the driver. Once a failure is detected, the `Detection` input signal is put to `true` and the driver's speed request is saturated up to the `UpperLimit` in case of forward direction, or `LowerLimit` in case of a revers (back-word) direction.

Once triggered, the mitigation algorithms remains in the *degraded* state even if the detection algorithm stops to perceive a failure.

3.8.5 Experimental setup

The benchmark application is composed of:

²¹The FMs are injected one at a time.

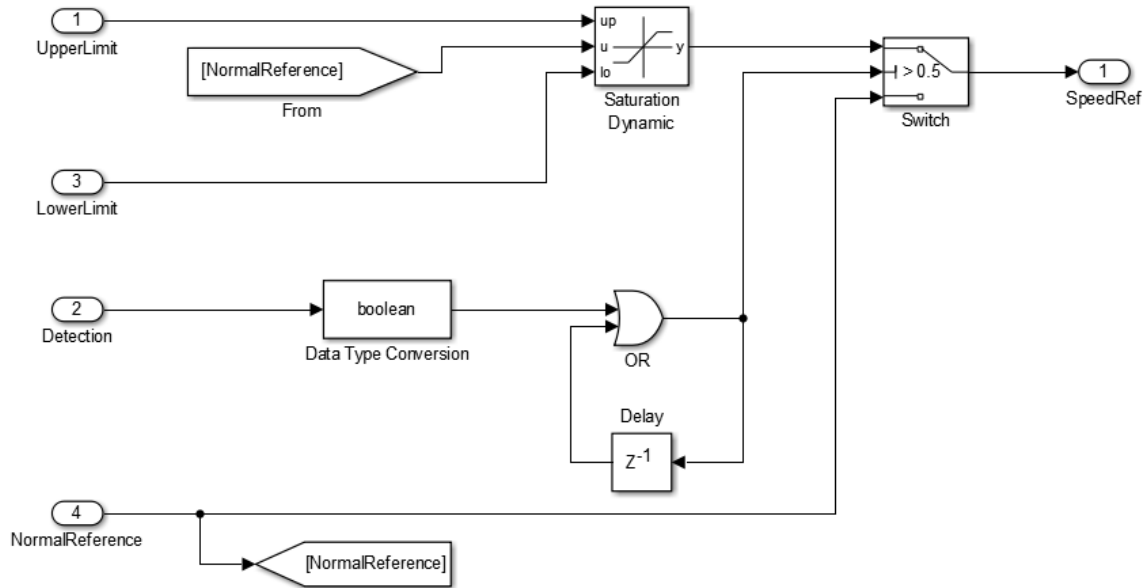


Figure 3.17: Mitigation algorithm. Figure from [19].

- the embedded software of the dual inverter system
- the *structural* model of the inverters, with their fault models to be injected;
- the model of the electric motor;
- the model of the driver, in the form of a PID controller;
- physical models of the car and surrounding environments (tracks) provided by the vehicle-level simulator;
- scenarios in which test the failure effects;
- vehicle behavior classification rules.

The elements indicated in fig.3.15 are implemented by:

- fault list generator module \Rightarrow MathWorksTM MATLABTM script;
- sabouter \Rightarrow MathWorksTM MATLABTM script;
- *structural*-level simulator \Rightarrow MathWorksTM SimulinkTM with SimScapeTM toolbox to perform SPICE-level simulation of the design;
- vehicle-level simulator \Rightarrow CarSimTM [50];
- fault list classifier \Rightarrow MathWorksTM MATLABTM script;

3.8.6 Simulation results

Five different situations have been taken into account to assess the vehicle-level mitigation algorithm:

- a) driving straight at 130 km/h;
- b) acceleration from 0 to 130 km/h;
- c) triple curving at 100 km/h;
- d) regenerative braking on a straight road from 130 km/h to 0 km/h;
- e) regenerative braking on triple curving from 100 km/h to 0 km/h;

The curving track is shown in fig.3.18 Among these cases, the three most interesting are the b), c) and e).

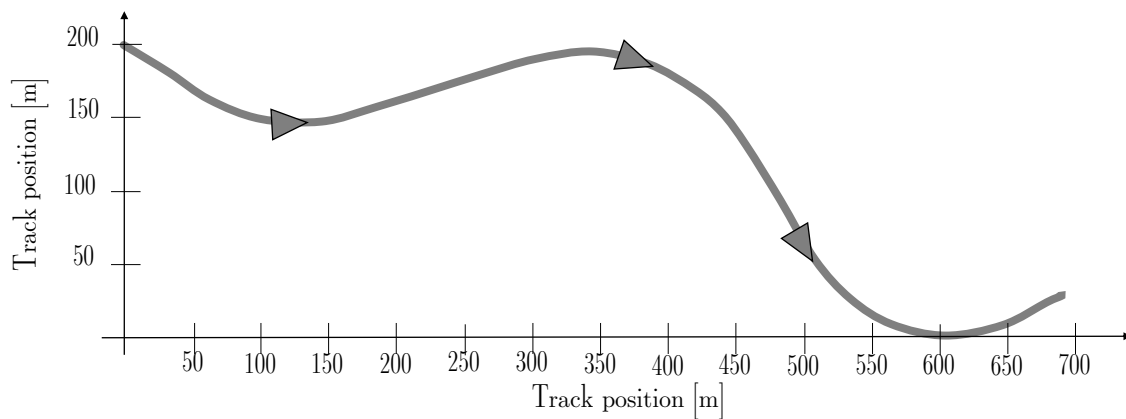


Figure 3.18: The triple curving simulated track. The arrows indicates the direction of the ride.

Cases a and d

In the cases a) and d), the results obtained with the mitigation algorithm enabled or disabled are too close to each other to comment the results: in table 3.7 are reported the values of the maximum errors in terms of yaw angle and lateral displacement errors for the cases a) and d).

Case	Conditions	Lateral displacement [m]	Yaw angle error [deg]
a)	FA	$[-0.117, 0.100]$	$[-0.720, 0.385]$
	M	$[-0.096, 0.080]$	$[-0.289, 0.248]$
d)	FA	$[-0.026, 0.032]$	$[-1.477, 0.813]$
	M	$[-0.026, 0.031]$	$[-1.453, 0.796]$

Table 3.7: Simulation results. Conditions: **FA** indicates fault affected without mitigation algorithm, while **M** indicates fault conditions with the mitigation algorithm enabled

Case b: acceleration from 0 to 130 km/h

This situation represents a full-throttle acceleration from 0 to 130 km/h. The car took 16 s in fault-free condition to reach the target speed.

As shown in fig.3.19, the mitigation algorithm is quite useful to limit the failure effects in terms of lateral error. When the failure is injected and hence detected (at about 2 s from the start of the simulation), the benchmark mitigation algorithm starts to limit the lateral error, especially at high speed (when it is more difficult for a human driver to intervene).

Analyzing the error in terms of yaw angle (see fig.3.20), it is possible to see that the mitigation algorithm can reduce the error from the range -0.5 to 0.6 deg to -0.2 to 0.2 deg.

In this situation, the mitigation algorithm, even if it is straightforward, has demonstrated itself capable of reducing both the lateral and the yaw angle errors of the car.

Case c: triple curving

In the straight acceleration, we obtained quite good results from the chosen algorithm. So, to keep into account a different and more challenging condition, the experiments have been repeated on a curving track, shown in fig.3.18.

As shown in fig.3.21 and fig.3.22, the adopted mitigation strategy improves the lateral error and worsens the yaw angle performances. This is an expected result since it bounds the speed of the fault-free wheel to the fault-affected one: the chosen control strategy is set with a small proportional gain in the speed loop, limiting the effects of the transient.

Case e: regenerative braking on triple curving

In this case, where regenerative braking from 100 km/h to 0 km/h is simulated on the track of fig.3.18, the mitigation algorithm does not improve the lateral error and worsen the yaw angle, as shown in fig.3.23.

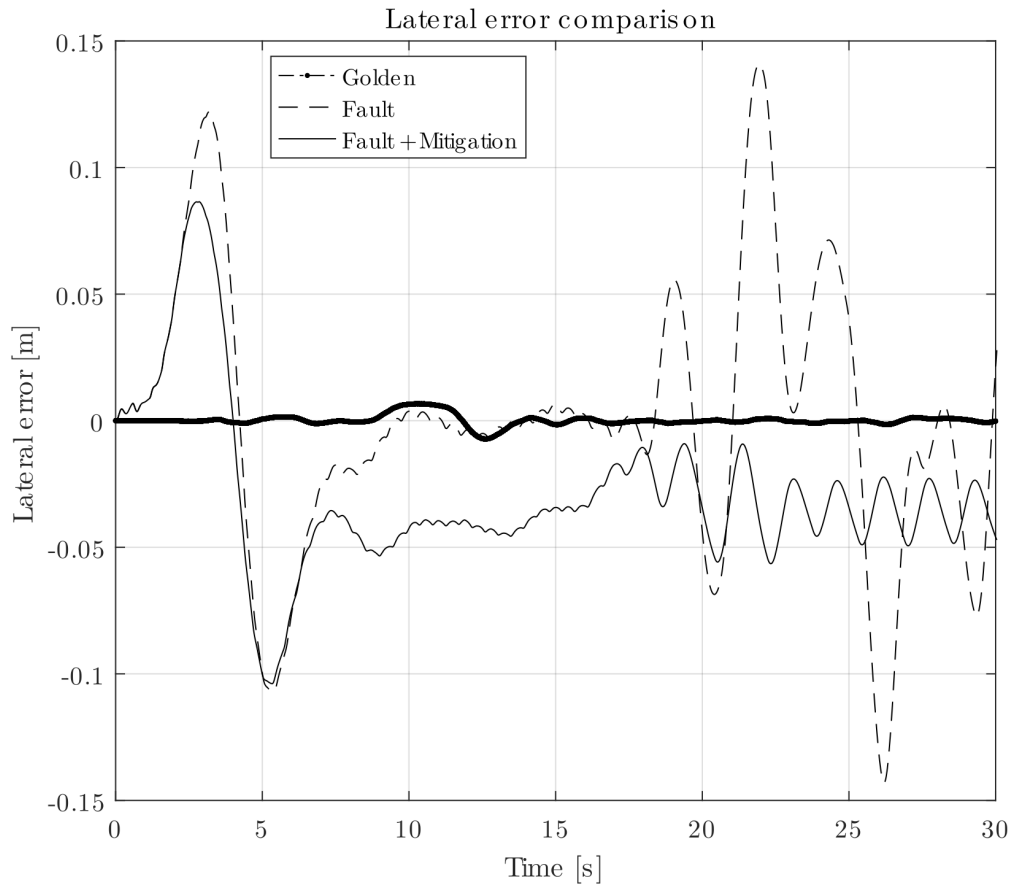


Figure 3.19: Case b lateral errors with respect to the ideal trajectory. Adapted from [19].

These errors are inside the acceptable range, so in a tradeoff, it remains convenient mitigation algorithm adoption.

In any case, this worsening effect warns about the side-effects the mitigation algorithms can add at the vehicle-level: these, in some cases, can become ineffective or even worsen the situation. It highlights the need for extensive testing in different conditions.

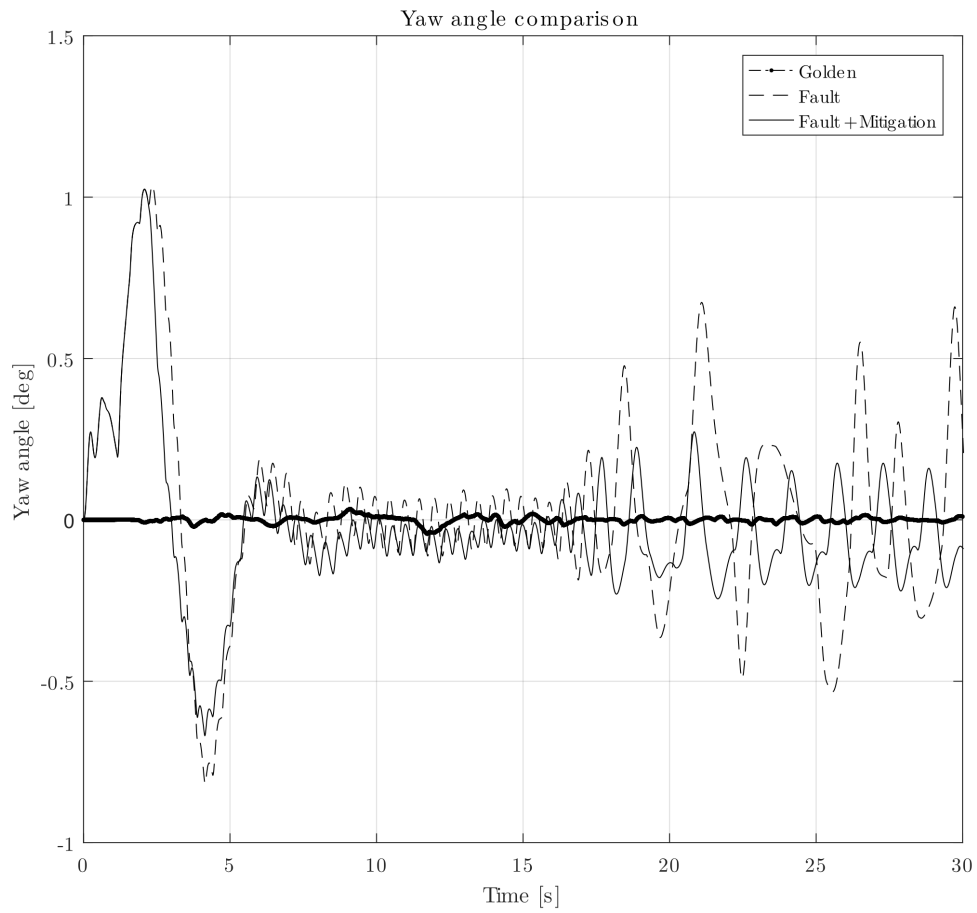


Figure 3.20: Case b yaw angles measured with respect to the tangent of the ideal trajectory. Adapted from [19].

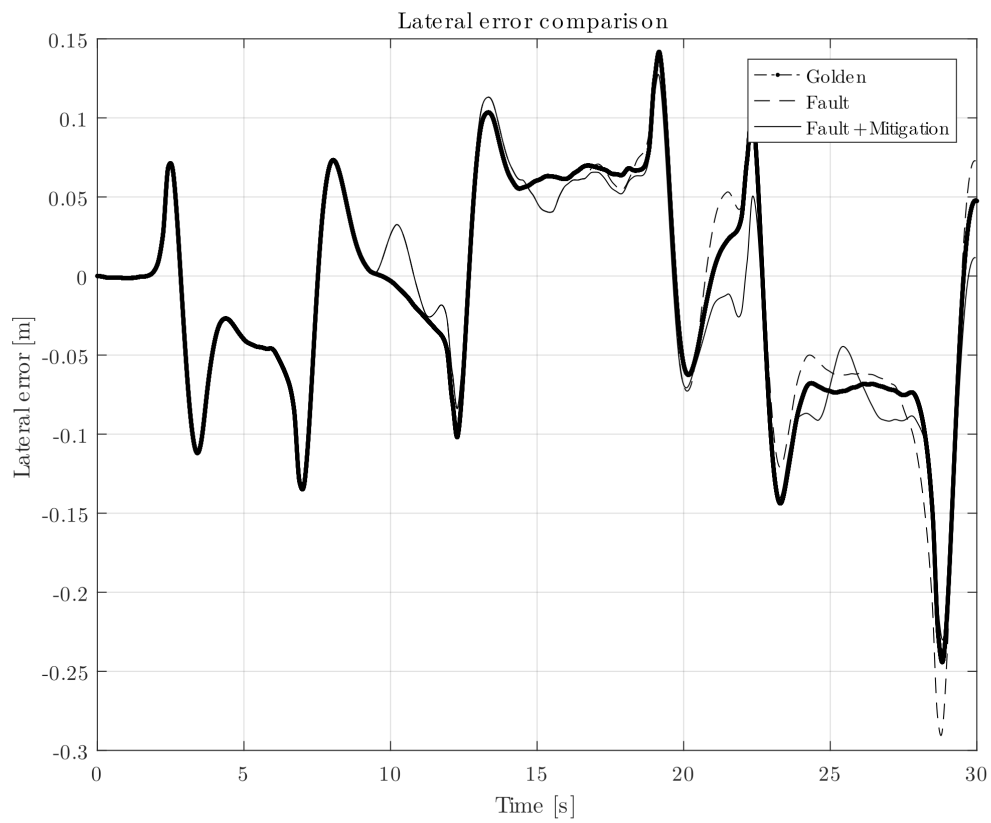


Figure 3.21: Case c lateral displacement with respect to the ideal trajectory. Adapted from [19].

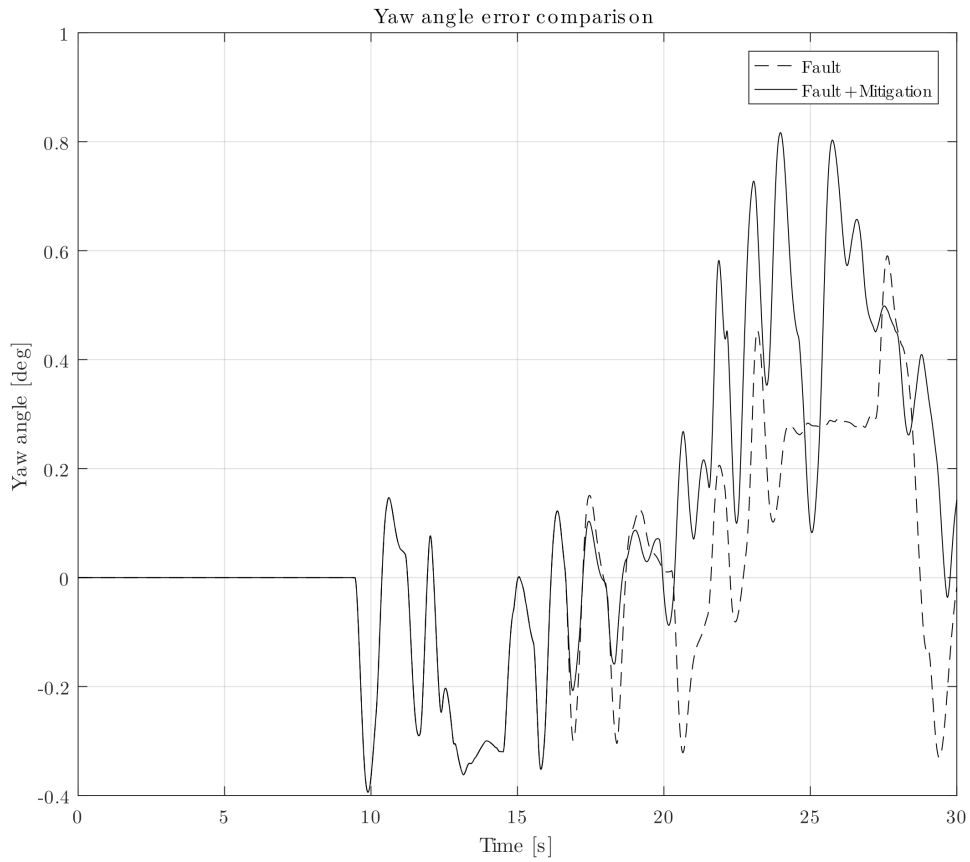


Figure 3.22: Case c yaw angles measured with respect to the tangent of the ideal trajectory. Adapted from [19].

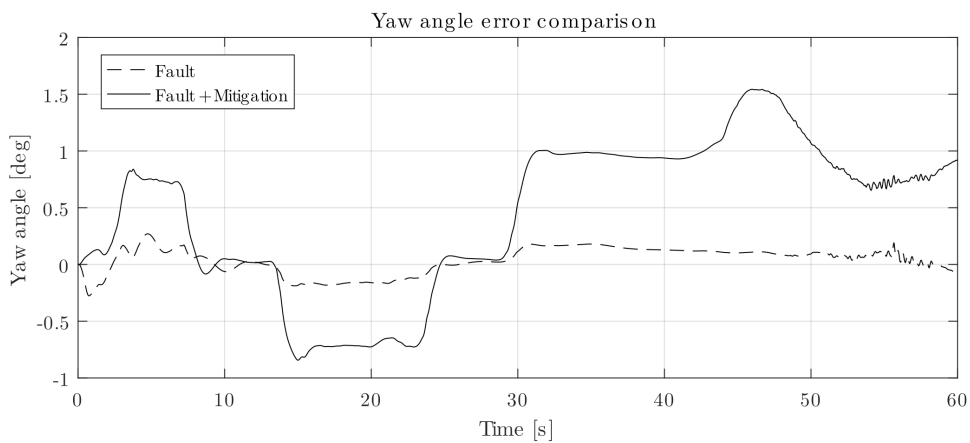


Figure 3.23: Case e yaw angles measured with respect to the tangent of the ideal trajectory. Adapted from [19].

3.9 Fifth proposal - From safety to mission criticality: finding a tradeoff between needs and model precision

The system considered in this proposal is a *complex* cyber-physical system: it is composed of several interconnected subsystems, designed independently from each other to perform a specific part of the system functionality.

The main contributions of this proposal, with respect to the other methodologies described in this dissertation ([15] presented in the section 3.5, and [16] [18] presented in the section 3.6) are the following:

- makes use of electronic design automation (EDA) tools, already available on the market and usually licensed by hardware design companies, to perform analysis of complex cyber-physical systems composed of analog, power, digital, and electromechanical subsystems.
- allows to evaluate, in detail, the effects on the whole system of FMs affecting a target subsystem: for this purpose, the affected subsystem is simulated thanks to a *structural* model, while the stimuli to be applied to the subsystem and the effects of the faults at the system level are computed resorting to *behavioral* (high-level of abstraction) models; recent EDA tools allow to easily combine *structural* and *behavioral* models and to perform their combined simulation effectively;
- automates and speeds-up the FMECA process, supporting a critical step in today's design flow of many systems;

The experimental results, gathered on the case study, show that, thanks to the proposed approach, the safety engineer not only can more easily identify the critical faults affecting the system, but their number is significantly reduced, mainly due to the spontaneous compensation effects emerging from the effects of the closed-loop control systems²².

This structure is particularly interesting since designers can use it alongside the general structure of items (sensors-computing unit-actuators) to put the failure modes effects assessment in a general perspective.

While [15], [16], and [18] the main focus was the Functional Safety of the designed item, i.e., to avoid unreasonable risk due to misbehaviors on its critical

²²Even in case a *compensation* spontaneously emerges from the closed-loop controllers, these can lead to overstresses on other components, founding themselves outside their nominal specifications. In those cases, designers can adopt this simulation-based approach to aid the development of detection mechanism to avoid that these effects trigger other FMs on the overstressed components leading to critical failures at the system level.

functionalities, while in this paper, the purpose is to increase the complex system availability by considering it as a mission-critical one.

In [15], [16], and [18] the FMECA is performed for a single analog subsystem by injecting the FMs at the low level (see section 3.5.1) and their effects are not propagated to the other subsystems²³.

The FM assessment is based on two classes: critical and non-critical. Critical FMs are those that make it impossible for the complex system to provide its functionality, while the non-critical ones are those not affecting the functionality, or *masked*. In [42], [51], [52], [53], [41], and [54], the faults effects are propagated to other subsystems, but the failure modes are simulated by injecting *behavioral* models, hence by input-output relationship of the affected subsystem. In this proposal, instead, the FMs are instead injected at the *structural* level, as discussed in the fault models sections.

It is possible to summarize this proposal, by saying that the low-level fault injection system is the one proposed in the first two ones (see sections 3.5 and 3.6), the system-level classifier is similar to the one proposed in the fourth (see section 3.8), where the assessment of the failure effects is performed at the system level (in the specific case applied to the entire vehicle dynamics).

The assessment of the embedded software effects mitigation capabilities was present in all these older proposals and kept in this one.

The purpose of this proposal is to assess the effects of the FMs affecting one of the subsystems (SSUT) composing the complex system. The FMs list is generated accordingly to the *Structural* (low-level) fault models.

The proposed approach for a generic SSUT of a complex system is performed in eight steps, as described below, and indicated in gray circles in fig.3.24.

1. *Block diagram generation.* The block diagram of the overall complex system is obtained.
2. *Preparation of the behavioral models for the subsystems..* Behavioral models of each subsystem present in the complex are prepared. They can be obtained from the design phase of the complex system or by theoretical or identified transfer functions between the inputs and outputs of the subsystems.
3. *Behavioral system simulation*²⁴. When all the behavioral models are connected to each other accordingly to the complex system topology, it is possible to perform a simulation of the complex system in fault-free conditions. To assess the quality of the obtained model, stimuli compliant with its design specifications are applied to the complex system inputs. The responses must

²³Items in the ISO26262 vocabulary.

²⁴In the paper [40] this phase is called *High-level system simulation*.

comply with the complex system design specifications. Generally, in a cyber-physical system, the system's stimuli are electrical, while the response to the stimulus is obtained on the mechanical actuator.

4. *SSUT structural model*²⁵. The subsystem in which the faults are injected is now chosen. Its *behavioral* model is replaced with a *structural* one, where the FMs will be injected.
5. *Model check*. It is performed a new simulation in fault-free conditions to verify that the *structural model* of the SSUT works properly.
In the case the obtained responses are compatible with the one obtained with the *behavioral* model. It is stored as the *golden response*.
6. *FMs list generation*²⁶. The list of the FMs to be injected is generated.
7. *FM effects simulation*²⁷. For each one of the injected FMs, a simulation is performed by applying an opportune set of stimuli to test various working conditions of the complex system. The simulation results are stored.
8. *FM effects classification*²⁸. The *classifier* [16] [19] compares the responses obtained from the fault-affects complex system with the *golden* one obtained during the *model check* phase and against the rules defined in the section [3.9.3](#).

²⁵In the paper [40] this phase is called *Subsystem under test*

²⁶In the paper [40] this phase is called *Fault level simulation*.

²⁷In the paper [40] this phase is called *Fault effect simulation*.

²⁸In the paper [40] this phase is called *Fault effect evaluation*.

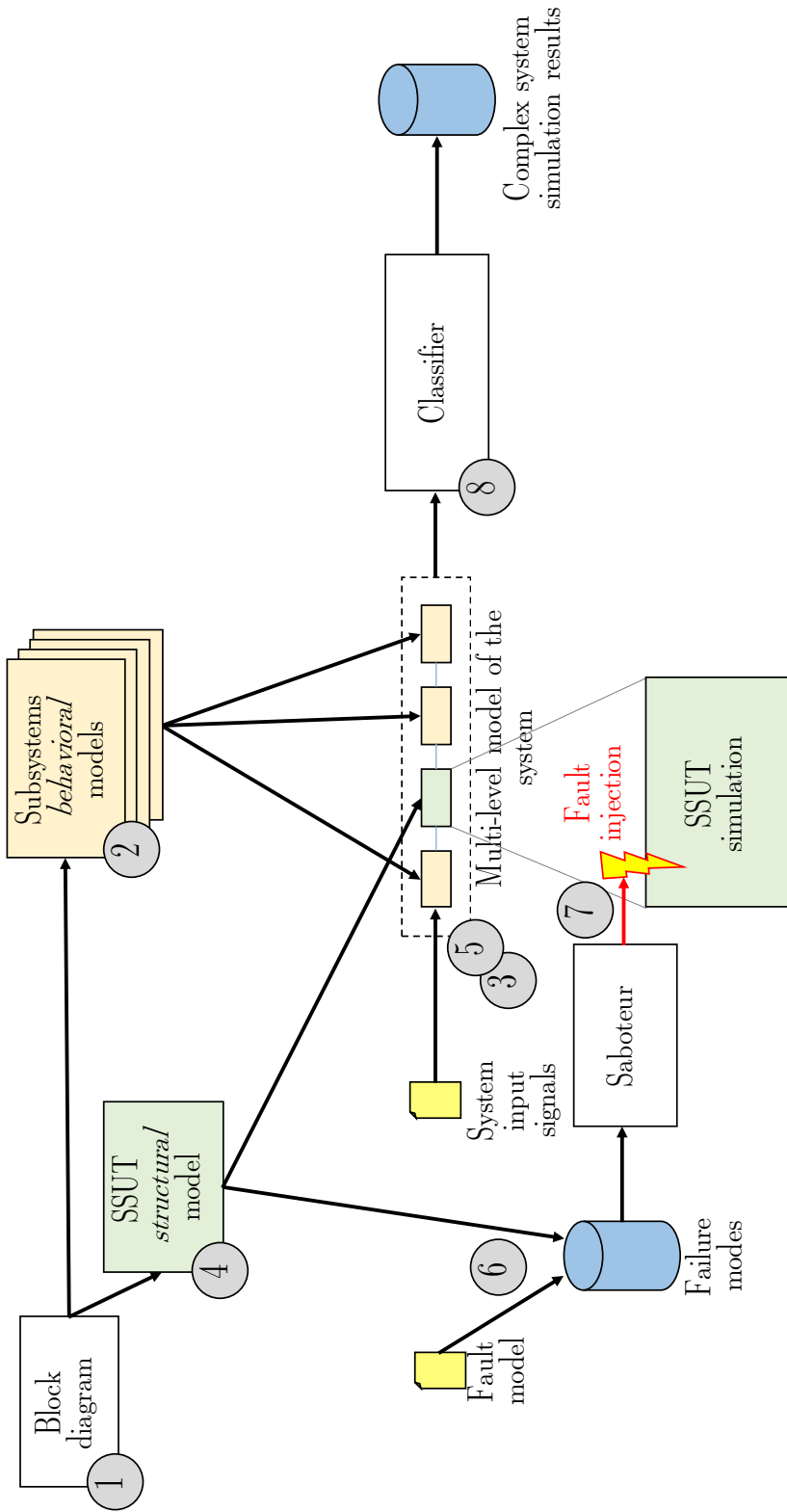


Figure 3.24: Representation of the proposed approach. Adapted from [40].

The considered case study is a three-phase motor control system used for industrial applications, such as compressors and forced ventilation systems. It does not provide safety-critical functionalities but can lead to a stop of production and a loss of money (mission-criticality). Sometimes the availability of these systems can be considered as safety-critical, but in this paper, no assumptions of this kind are considered. In any case, being able to accurately and automatically estimate the FMs effects is of paramount importance.

The control system is composed as shown in fig.3.25. The SSUT is the Power Supply Unit (PSU), in charge of supplying the high voltage needed for the electrical motor. In particular, it has to provide a DC voltage of 400 V with a maximum ripple of ± 7 V with a maximum current of 12 A. It accepts as input a single-phase AC voltage from 110 to 250 V (RMS), with a 50 or 60 Hz frequency. This voltage is used to drive the motor and a PSU-low power that generates the low voltages needed by the conditioning and elaboration units. A *structural* model of the PSU is required to perform fault injection. Its circuit diagram is shown in fig.3.26.

The PSU is composed of:

- a diode (Graetz) bridge (D_{w1} , D_{w2} , D_{w3} , D_{w4});
- three boost cells;
- an analog FAN9673 controller (implemented as an IC).

The schematic of the PSU is shown in fig.3.26.

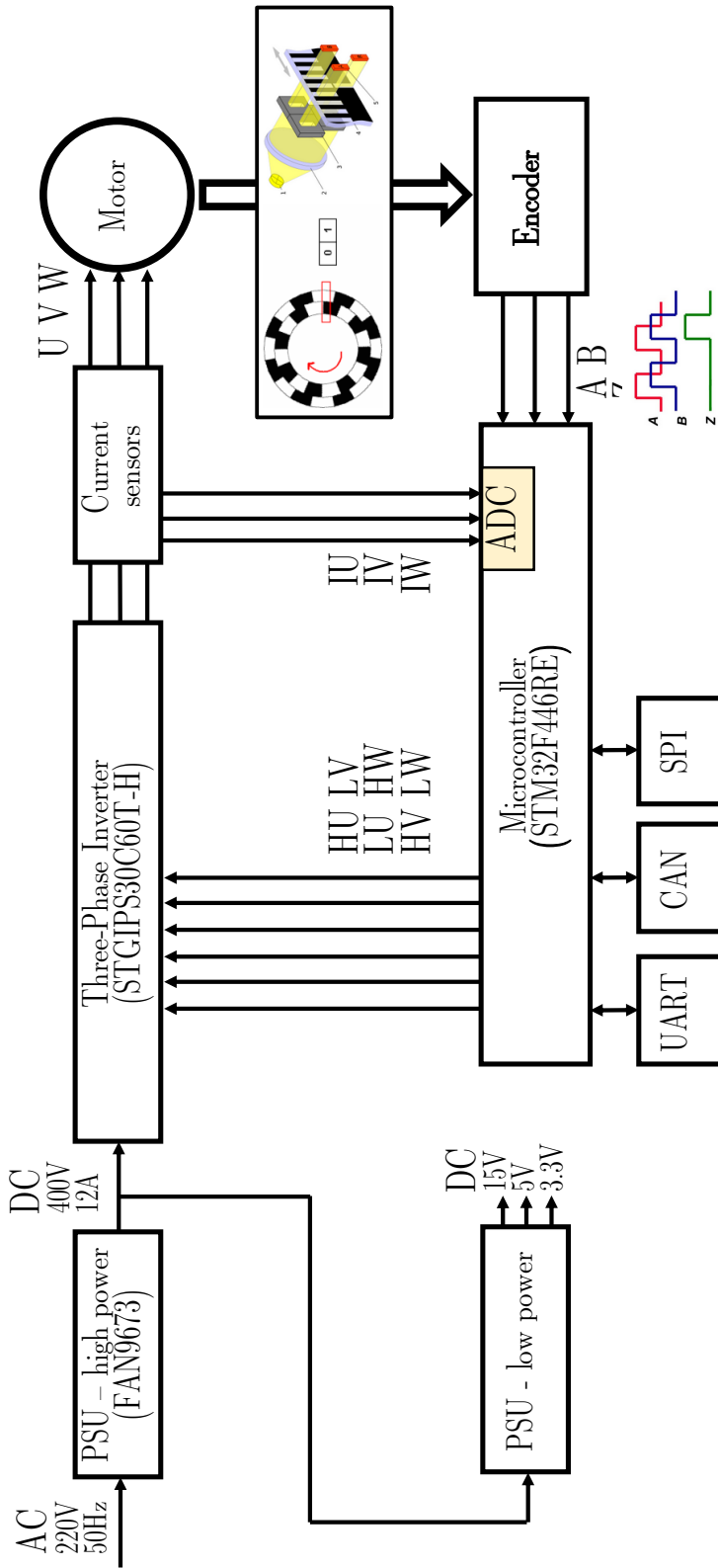


Figure 3.25: Block diagram representation of the motor control system case study. Adapted from [40].

3.9.1 Fault models

Peculiarity of this proposal is the presence of two different levels of models:

- *behavioral* high-level;
- *structural* low-level.

The *behavioral* models are adopted only for the fault-free subsystems; hence, in this proposal, no *behavioral*-level fault models are considered.

The FMs are represented only on the *structural* models domain.

The saboteur injects FMs (defined as *catastrophic failures* in [47]) into the *structural* model of the SSUT. These are modeled, differently from the previous proposals, (compare them with *simulation-only components* in the section 3.6.1) by adding electrical switches (see section 3.9.2) in the circuit diagram. This choice has been made due to the possibility of automating this phase. In this case, a human assessment to exclude the impossible FMs added in this way is required, but, in any case, it is reasonable to expect that it requires less time with respect to adding by hands the *simulation-only components*.

For the sake of this proposal, are assessed only the FMs capable of affecting the power devices assembled on its PCB. The FMs list is generated in accordance with the PCOLA/SOQ [55] standard.

Boost Cell Faults

Following the rules, nine different FMs are considered in a single boost cell of the PSU. The three boost cells present in the PSU, shown in fig.3.26 are identical and placed in parallel; therefore, it is possible to study the effect of the faults in one of the three boost cells indistinctly. The boost cell considered is composed of three elements: an inductor L_1 , a diode D_1 and an IGBT T_1 . As simulation-only components, nine electrical switches (leading to 9 failure modes for each one of the boost cells) are placed between these three components, obtaining the schematics shown on the right side of fig.3.27.

IGBT Faults

The IGBTs have their fault model.

How to derive an equivalent model is discussed in [47].

Fig.3.28 shows the equivalent electrical model of the IGB with 23 switches corresponding to 23 *catastrophic defect* under the IEEE P2427 classification.

The interested reader can find a discussion about physical meaning of these faults in [47] and [56].

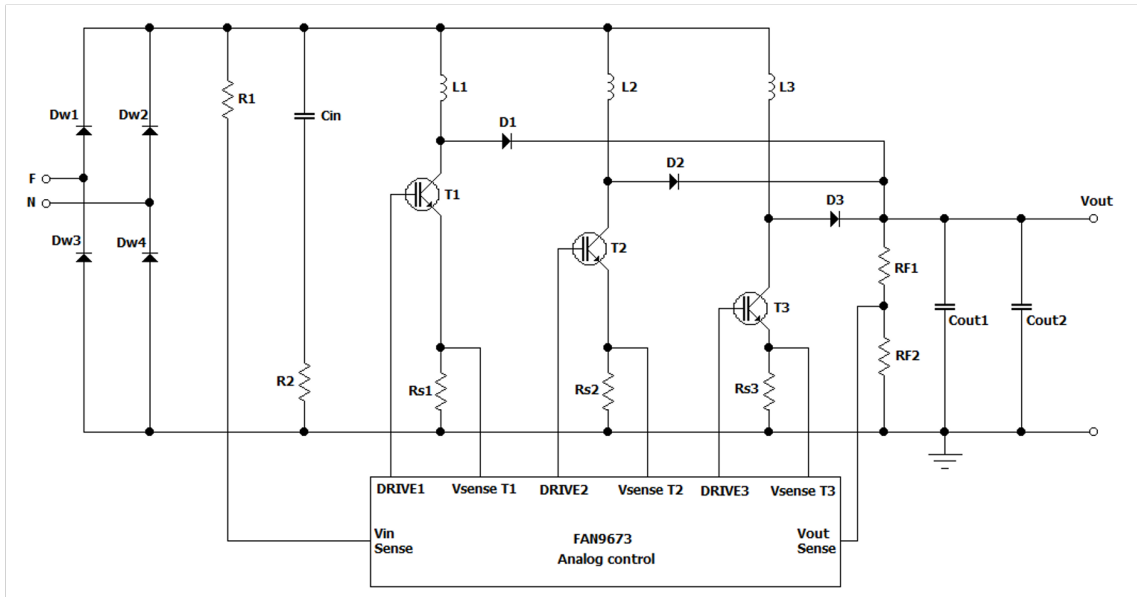


Figure 3.26: Schematic of the PSU. Figure from [40].

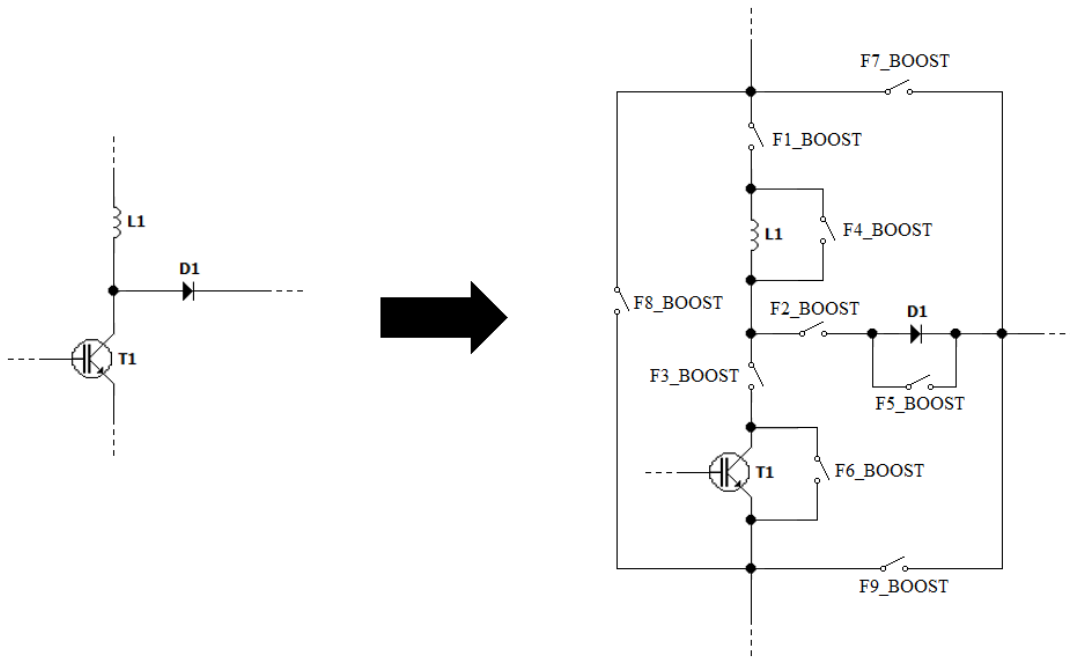


Figure 3.27: A boost cell of the PSU instrumented with simulation only switches. Figure from [40].

3.9.2 Fault injection

The fault injection is performed by properly changing states of:

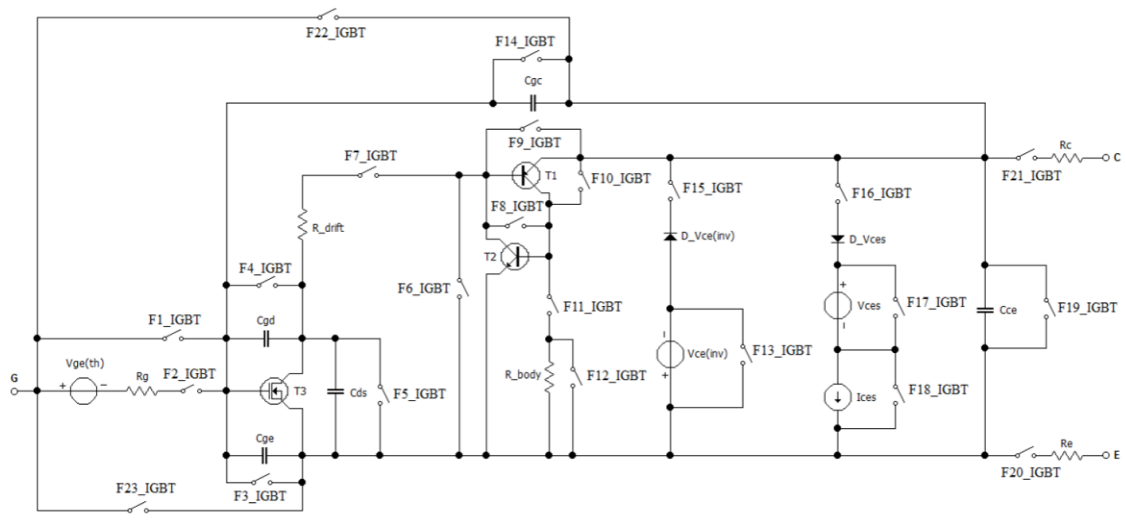


Figure 3.28: Schematic of the PSU. Figure from [40].

- switches inserted in series to each device present in the circuit diagram;
- switches inserted in parallel to each device present in the circuit diagram;
- switches inserted between different nodes of the circuit diagram, in particular between nodes that are normally unconnected in the SSUT circuit diagram.

generated as described in the fault model section 3.9.1.

3.9.3 Failure modes effects assessment strategies

The proposed approach is designed to identify the critical FMs, i.e., those ones that modify the behavior of the complex system, leading to a violation of its specification, hence to an impossibility to provide its functionality (in this case, to drive the electrical motor).

In the proposed approach, the effects of the FMs can be observed only in some specific accessible points of the system. In particular, the electric signals present on the output ports of the complex system (voltages or currents), or the physical quantities handled by the mechanical actuators.

The injected FMs are considered as critical one²⁹ if the response from the complex system is not compliant with the design specification or coherently with the

²⁹Differently from [15], [16], [18], and [19] the considered case study is not a safety-critical item for a vehicle, so the purpose is not to determine if the considered FM is dangerous or safe by assessing if there is or not a violation of safety goals, but an evaluation of their capability to affect the complex system availability.

definitions found in manuals [5] and [57] if it produces a difference w.r.t. the complex system requirements.

Since a numerical criterion is defined, maximum tolerance values are established for each of the physical (mechanical and electrical) quantities present during the design phase.

The FM is classified as critical if the value obtained in the simulation exceeds the maximum accepted tolerance, reported in table 3.8.

	Nominal value	Tolerance *	Tolerance range
U, V, W voltage	400 V	1%	396 V – 404 V
U, V, W current	6 A	2%	5.88 A – 6.12 A
Angular speed	3000 RPM	5%	2850 RPM – 3150 RPM
Vout high-voltage PSU	400 V	1%	396 V – 404 V

Table 3.8: The classification criteria, with their tolerances, as defined in the system design phase (as defined by the complex system designer).

3.9.4 Assessment of the SW mitigation capabilities on HW failures

In the considered complex system, the embedded software³⁰, is in charge of managing all the subsystem, including their failures (detected by hardware paths), and generating the appropriate microcontrollers’ peripherals registers configuration to generate the firing signals for the inverter IGBTs.

Since this system is fail-safe, it does not embeds routines to mitigate or react with a *graceful degradation* strategy to failures, but if a critical failure happens, it shuts down the motor and the PSU.

3.9.5 Experimental setup

From the technical point of view, the approach represented in fig.3.26 has been implemented with the PLECS simulator [58], which is incorporated in, and handled by, the Mathworks Simulink [59] environment. The whole simulation environment is managed with numerous MATLAB scripts. Therefore, different steps of the

³⁰The embedded software considered for this complex system has been provided by the PEIC laboratory of the DENERG Department of Politecnico di Torino. It is a production-level C language implementation, so it has not been modified to perform these simulations.

proposed approach shown in the figure are performed automatically: simulation management, fault injections, data collection, and classification.

PLECS is specifically designed for simulating power and analog electronic circuits and mechanical actuators. Moreover, it allows C code execution through a particular functional block, called *C-Scripts*. Thanks to this block, it is possible to run the embedded software within the simulation environment.

In the real system, a timer, integrated into the microcontroller, is configured for executing the motor control software as a 16 kHz³¹ periodic task.

This behavior is also replicated in the simulation: every 62.5 μ s PLECS calls the control routines, modifies the outputs of the simulated microcontroller, and resumes the simulation.

3.9.6 Simulation results

A total of 32 FMs related to the PSU have been considered. Each simulation is performed by injecting a single FM at a time.

The results are automatically processed with some MATLAB scripts³² to classify the FM. The obtained classification are reported in table 3.9.

As far as the real-world simulation duration is concerned, simulating 20 s of the whole system with all the electrical subsystems represented by *structural* model (SPICE-level) requires approximately 170 minutes³³.

Conversely, when using the proposed multilevel simulation, only about 30 minutes of CPU time is needed.

This highlights the effectiveness, in terms of performance speed-up, of the multilevel simulation approach proposed. On the case study, it is improved about six times.

In table 3.9, it is possible to identify 6 FMs classified as critical and 26 as non-critical. The impact of the six faults classified as critical on the overall complex system is particularly significant. Designers must implement mitigation strategies to detect FMs, even the non-critical ones, due to their compensation by the FAN9673 analog controller, as discussed in [56]. In other words, the PSU control system acts on the IGBTs to compensate for the effect of the injected fault.

This behavior improves the availability of the item for a certain amount of time, but it is necessary to detect all the FMs to let the user know that service of the complex system is required before the *masked* failure triggers a critical FM on an overstressed component.

³¹This frequency has been chosen by the control software developers.

³²Implementation of the *classifier*.

³³CPU times measured on a workstation based on an AMD FX-8370 8 core processor at 4 GHz, equipped with 32 GB of 1333 MHz RAM

Faults	U, V, W voltage [V]	U, V, W current [A]	Angular speed [RPM]	Vout high-voltage PSU	Critical
Fault free	402	6.08	2979	400 V with ± 7 V of ripple	-
F1_BOOST	402	5.98	2979	400 V with ± 7 V of ripple	NO
F2_BOOST	263	4.26	1222	265 V with ± 25 V of ripple	YES
F3_BOOST	402	5.98	2979	400 V with ± 7 V of ripple	NO
F4_BOOST	402	5.98	2979	400 V with ± 7 V of ripple	NO
F5_BOOST	377	7.90	1718	Vout instable	YES
F6_BOOST	398	5.89	2866	397 V with ± 10 V of ripple	NO
F7_BOOST	398	5.89	2866	397 V with ± 10 V of ripple	NO
F8_BOOST	0	0	0	0V	YES
F9_BOOST	0	0	0	0V	YES
F1_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F2_IGBT	398	5.89	2866	397 V with ± 10 V of ripple	NO
F3_IGBT	398	5.89	2866	397 V with ± 10 V of ripple	NO
F4_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F5_IGBT	398	5.89	2866	397 V with ± 10 V of ripple	NO
F6_IGBT	398	5.89	2866	397 V with ± 10 V of ripple	NO
F7_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F8_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F9_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F10_IGBT	398	5.89	2866	397 V with ± 10 V of ripple	NO
F11_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F12_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F13_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F14_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F15_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F16_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO

Faults	U, V, W voltage [V]	U, V, W current [A]	Angular speed [RPM]	Vout high-voltage PSU	Critical
F17_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F18_IGBT	402	5.98	2979	400 V with ± 7 V of ripple	NO
F19_IGBT	398	5.89	2866	397 V with ± 10 V of ripple	NO
F20_IGBT	398	5.87	2866	397 V with ± 10 V of ripple	NO
F21_IGBT	398	5.93	2866	399 V with ± 8 V of ripple	NO
F22_IGBT	312	4.90	1585	300 V with ± 20 V of ripple	YES
F23_IGBT	312	4.90	1585	300 V with ± 20 V of ripple	YES

Table 3.9: Failure modes classification results.

3.10 Sixth proposal - Application to a mobile robotics case study

Due to intellectual property protection, it is difficult to find a complex case study to apply the FMEA, HARA, and FME(D)A processes. Hence, a mobile robot (rover), designed by the students' team D.I.A.N.A. of Politecnico di Torino and already analyzed in a Master's Degree thesis [23], has been considered as the benchmark application of this proposal.

Due to its mechanical design, this rover needs a microcontroller-based system capable of harmoniously moving the traction and steering motors to be simple to use in manual mode, via a stick controller, and through autonomous driving algorithms.

The main focus of this proposal is the assessment of detection and mitigation capabilities of the embedded software over the *failures*³⁴ affecting its sensors and actuators.

The detection (and mitigation triggering) capabilities are assessed in an automatic way, while the mitigation ones are assessed by hands thanks to the aid offered by the simulation results.

Two industrial standards have been followed to perform the analysis: the FMEA manual from AIAG and VDA [5], the HARA process 2.2.2 of the ISO26262 and the ECSS-Q-ST-30-02C for what regards the hardware-software interaction analysis.

The main contributions of this proposal are threefold:

- it proposes a methodology to improve the objectivity of the FMEA and HARA phases by a simulation-based approach;
- it contains a way to continuously verify the fulfillment of the safety requirements, from the early development phases, as further subsystems are implemented step by step to the final design;
- it proposes a way to simplify the safety assessment of the HSIA, especially in those cases where mechanical, electrical/electronic components, and the embedded software, are involved in determining complex behaviors of the entire rover.

The approach proceeds as shown in fig.3.29. It loads the *failure list* for the considered subsystem.

At this point, thanks to models of the electronics and mechanical components of the rover, the physical simulator simulates the rover, for each one of the *mission*

³⁴The models used in this proposal are all described at the *behavioral* level, hence for this proposal, I prefer to use the word *failure* instead of *failure mode*.

contained in the *mission database* in fault-free (golden) conditions, and subsequently, it injects the failures one by one in both the electronics and mechanics. After each simulation, the *system-level classifier* compares, by some set of classification rules, the failure detection signals with the ones indicated in the *detection capability requirements* database, and indicates if the failure has been detected or not. The content As the last operation, it computes the metrics and generates a *human-readable assessment report*.

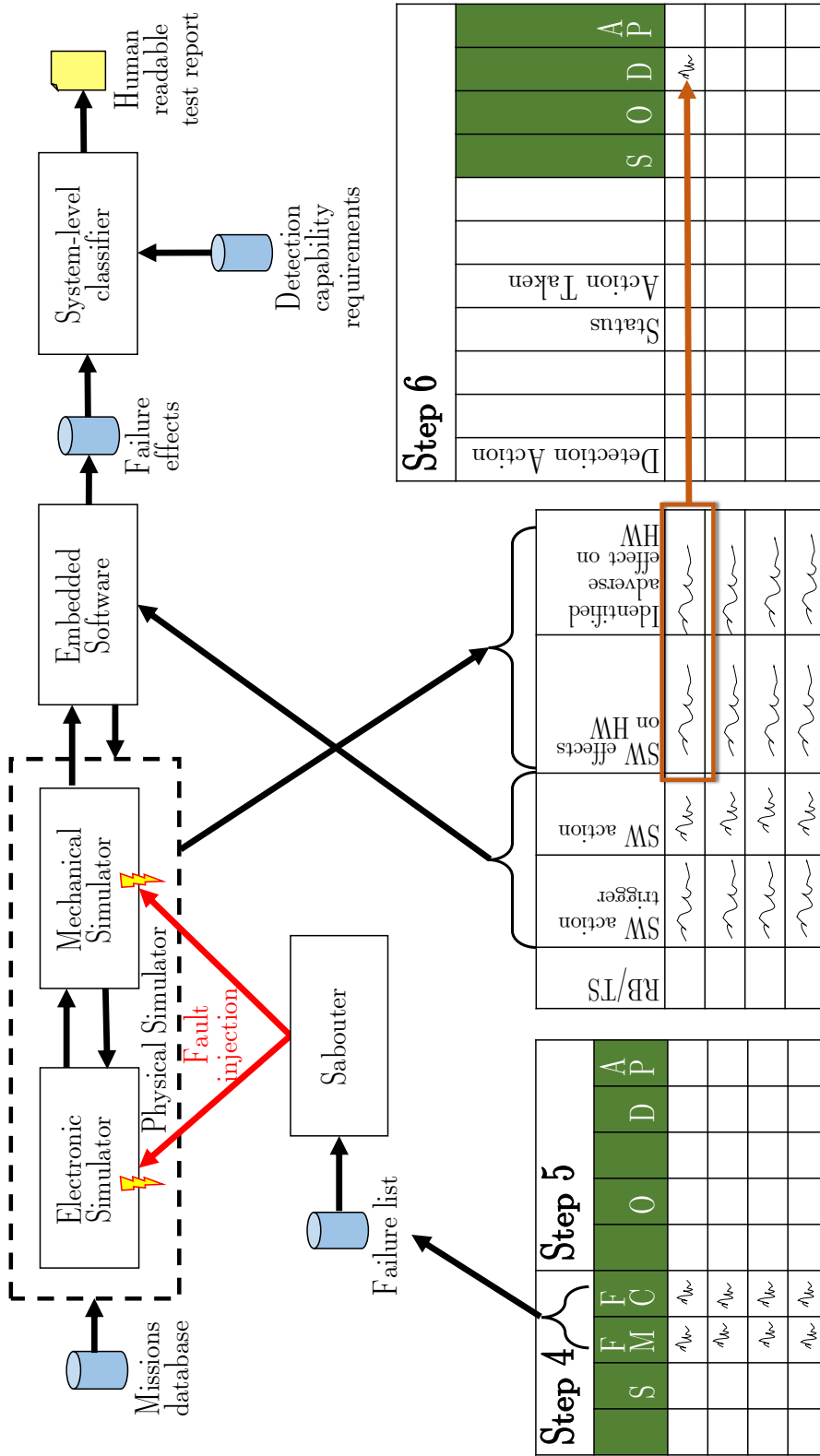


Figure 3.29: System level development process with the main involved documents.

3.10.1 Description of the rover

To better understand the rest of this proposal, it is useful to describe the rover. It is a prototype designed in 2020 to participate to international challenges, with in mind the futuristic purpose of aiding astronauts on Mars's surface.

Its frame adopts a rocker-bogie suspension system that has six wheels and allows it to climb over obstacles that are up to twice the diameter of its wheels without the need for springs [60].

This structure has five joints: two between the rockers and bogies, two between the rockers and the fixed frame, and one in the middle of the torsion bar.

The fixed frame supports an arm and contains scientific instrumentation and control computers.

The mobility system of the considered rover is composed of:

- 6 traction BLDC³⁵ motors (TM)
- 6 traction reduction gears (TRG)
- 6 clutches between the traction motors (TC)
- 4 steering stepper motors (SM)
- 4 reduction gears for the steering motors (SGR)
- 6 relative encoders installed on the traction motors (speed of the motor) (E-TM)
- 6 relative encoders installed on the traction reduction gears (speed of the wheel) (E-TGR)
- 6 hall effect speed sensors installed on the wheels (H-TGR)
- 3 absolute encoders, 2 installed on both rocker-bogie hinges, the other on the torsion bar one.
- 4 absolute encoders installed on the steering motors (position of the steering motor) (E-SM)
- 4 absolute encoders installed on the steering after the reduction gears (position of the wheel) (E-SGR).

³⁵BrushLess Direct Current.

Actuators

It is equipped with six traction wheels provided with independent permanent magnet motors (TM) and a reduction gear (TRG). Four of these (at the extremities) can steer thanks to four separate stepper motors (SM) and reduction gears (SRG). Due to the absence of mechanical differential gears or mechanical steering links, the embedded software is in charge of properly moving all actuators to allow the rover to move smoothly.

Sensors

Traction system These sensors are in charge of measuring angles and angular velocities. There are absolute encoders (E-SM and E-SRG) to measure the angles the steering wheels have with respect to the frame and relative encoders (H-TM, E-TM, and E-TRG) to measure the angular velocities of the wheels.

The encoders on the traction motor (regardless they perform an absolute or a relative measure) have been installed both upstream and downstream of the considered motor and reduction gear, with the first measuring the speed of the motor and the latter the speed of the wheel.

Passive joints Three absolute encoders measure the rocker-bogie passive joints positions: two of them are installed on the joints between rockers and bogies structure and the remaining one on the torsion bar.

Steering system Since there are no mechanical links between the steering wheels, the correct configuration of the steering system is guaranteed, thanks to the absolute encoders (E-SM and E-SRG) readouts, only by the mobility subsystem control software [61]. It is calculated by an Ackermann model adapted for the rover structure [23].

3.10.2 FMEA of the rover

This proposal aims to analyze how the embedded software can detect and mitigate some of the possible hardware failures, considering as the affected components: the encoders, the motors, and the clutches between the traction motors and the reduction gears. In this last case, the main goal is to reduce the damage to the rover caused by the failure, with a good compromise on its availability: since it is designed to aid astronauts in a hostile environment, fail-operational behaviors are preferred every time has been found an acceptable trade-off between the possible damages of the rover itself and simplification of the operations for the astronauts.

Different from the other proposals, this one also involves an FMEA phase (see section 2.10). The process starts as described by phases 2 to 5 of [5] (see fig.2.13).

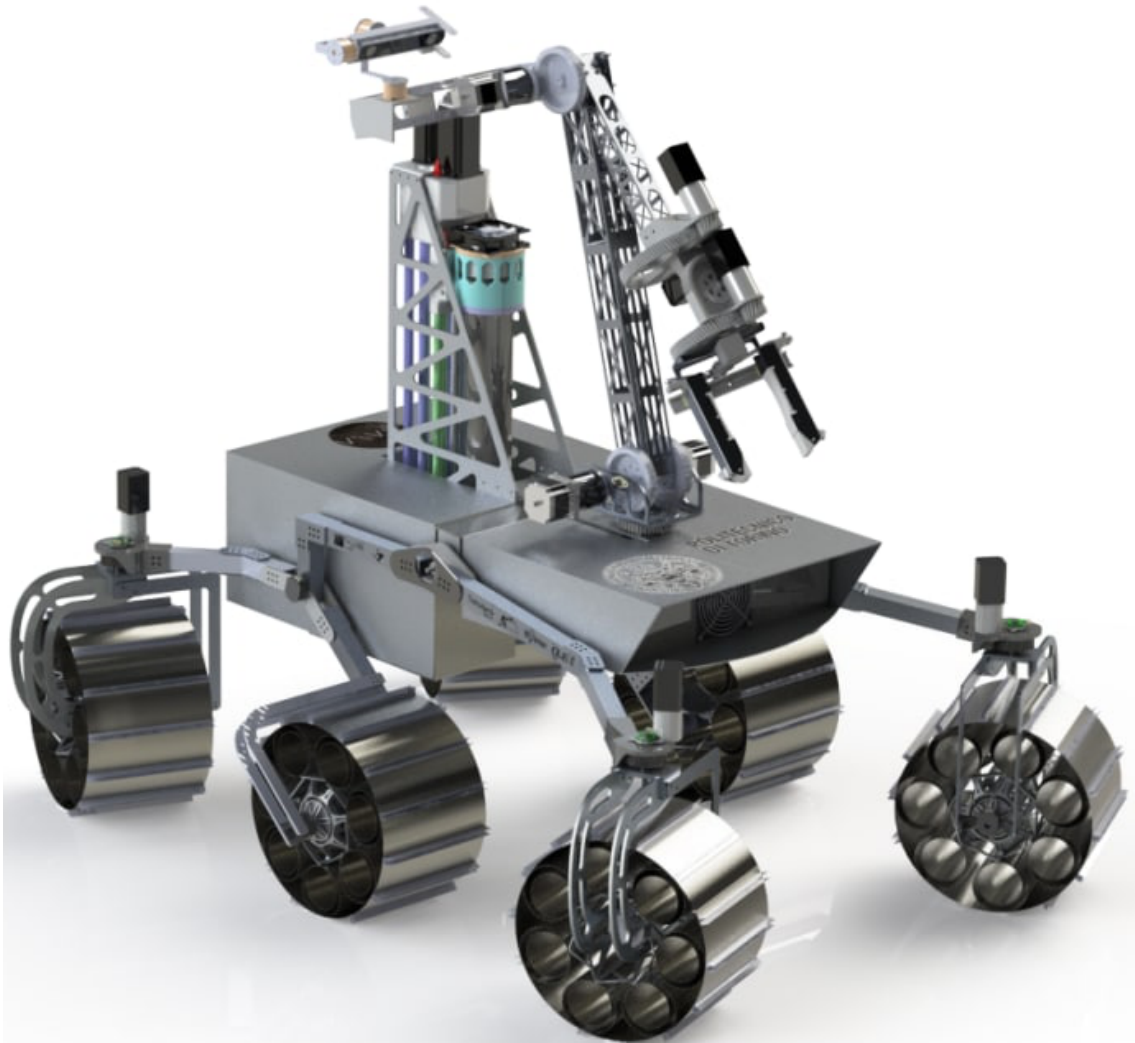


Figure 3.30: A 3D rendering of the Ardito Rover developed by the D.I.A.N.A. student's team of Politecnico di Torino.

Phases 2 and 3 are performed by hands.

Starting from phase 4, there are two options:

- perform it by hands as usual, for those failures that have clear effects at the system level;
- adopts a simulation-based approach for the one with no trivial failure effects. Thanks to the simulator, it is possible to analyze and score the effects of the failure modes by injecting them one at a time.

Failure mode scoring

The manual requires to score the failures in terms of three parameters.

- *Severity*: based on the worst possible consequences when the failure occurs. This parameter is scored, regarding the non-safety critical one (like S0 leading to QM classification in the ISO 26262 as described in section 2.2.2) from 1, which indicates that the failure has little consequences on the system, up to 8, where a primary function is lost. For the safety-critical one, 9 indicates that the failure mode provokes non-compliance with regulation while 10 indicates that it can have consequences on people's health and the environment. Thanks to simulation results, it is possible to obtain useful data to determine this parameter.
- *Detectability*: assess the capacity of the system to detect the considered failure. On this parameter, we have the key advance, since thanks to the simulation results, it is possible to observe if the embedded software can detect the failures and hence to trigger the appropriate mitigation algorithms (*SW action* in the [57] terminology), as described in section 3.10.6.
- *Occurrence*: is determined by hands based on statistical evidence since the simulation cannot help find out this parameter.

All these three parameters' scores range from 1 to 10.

Action priorities After the Severity, Detectability, and Occurrence parameters have been obtained, it is necessary to determine the Action Priority (AP), that correlates:

- Severity (S) of the failure, keeping into account the implemented mitigation measures;
- Occurrence (O) of the failure, keeping into account the implemented prevention measures;
- Detectability (D) of the failure, keeping into account the implemented detection measures.

The AP determination is designed to highlights in a first chance the severity, then the occurrence and the detectability. The AP can assume only three values: high (H), medium (M), and low (L).

The list of the APs found for the entire rover are reported in [23].

3.10.3 Fault models

To allow fault injection into the simulations are needed fault models able to represent them with a sufficient level of detail. They are described, for each class of components, in the following. These models are all defined at the *behavioral* level, hence the *catastrophic/parametric* classification of the IEEE P2427 is not applicable (since it is limited only to the structural models for analog electronic components).

Sensors

Absolute encoders When an absolute encoder fails, it provides a position readout equal to its zero position (0 deg in the simulation).

Relative encoders When a relative encoder fails, it provides a speed readout equal to zero (hence 0 rpm).

Hall effect speed sensors Hall effects speed sensors, like the relative encoders, when one of them fails, it provides a random speed readout.

Mechanical components

Reduction gears When a reduction gear fails, it remains stuck, preventing the affected wheel from rotating.

Actuators

Clutches The considered clutches are normally open. They are monostable, so only when the solenoid is operated they are closed. When one of them fails, the traction control is no more able to engage it, so the traction motor cannot provide torque to the affected wheel.

Traction motors When a traction motor fails, it remains stuck. Due to the presence of the reduction gear, the affected wheel remains stuck until the mitigation algorithm of the embedded software opens the clutch by removing the power supply to its solenoid.

Steering motors When a steering motor fails, it remains stuck. Due to the presence of the reduction gear, it is not possible to move the wheel, which remains with the relative angle applied by the motor the last time it was able to operate.

3.10.4 Fault injection

To test the controller response inside the simulation environment, it is necessary to implement the ability to inject faults in the mobility system. The different failure modes have been injected by replacing the signal values inside the simulation with the ones described by the fault model of the affected component.

3.10.5 Failure modes effects assessment strategies

Steps 1 and 7 of the FMEA manual will not be taken into consideration. Step 5 is briefly described, while the focus for the sake of this proposal is on phase 6, since the goal is to assess the software capabilities to detect, isolate, and mitigate the failure modes, classified with a M or H AP (see section 3.10.2) in phase 5. The Hardware/Software Interaction analysis from [57] plays its role between the phases *risk analysis* (5) and *optimization* (6), where have to be determined the *severity*³⁶ and *detection* capabilities of the embedded software capabilities for the considered failure.

Phase 5

During phase 5 the software detection and mitigation capabilities are not assessed since the manual requires not to consider advanced detection mechanisms, like the one based on the embedded software, assessed in this proposal. To comply with this requirement, benches of simulations are run, one for each considered failure mode, without mitigations. Moreover, a simulation in fault-free conditions is performed, to get the expected nominal behavior of the system and verify that it is able to perform its functionality. This tests the correctness of the model.

Hardware-software interaction analysis (HSIA) and phase 6

After the development of detection and mitigation algorithms, it is possible to assess them during the *optimization* (sixth) phase of the FMEA. Now, the considered system is simulated as did in phase 5, but including the mitigation algorithms effects. These simulations are repeated over and over again, allowing an iterative design approach until the expected detection³⁷ and mitigation capabilities have been obtained. In the terminology of the HSIA, as specified in [57], the requirements for

³⁶Not confuse this *severity*, described in section 3.10.2, with its namesake risk parameter of the ISO 26262, since it considers also, from 1 to 8 score, also mission-critical aspects (see section 1.0.1).

³⁷To trigger mitigation algorithms (SW actions), detection measures are needed.

our detection algorithms are the ones described in the *SW action trigger* columns, while the mitigation algorithms are described in the *SW action* columns of the central table shown in fig.3.29.

A *SW action* can be:

- an emergency procedure to move the system to a safe state (fail-safe behavior);
- an action to isolate the fault avoiding it propagating to the rest of the system (fail-operational behaviors);
- an algorithm to mitigate the failure effect obtaining a degraded³⁸ version of the affected functionality.

3.10.6 Assessment of the SW mitigation capabilities on HW failures

Thanks to the simulation results, it is possible to assess the effects of the *SW action* on the hardware.

Detection and mitigation algorithms have the purpose of improving the reliability, but unfortunately, sometimes can also introduce adverse side effects that can affect the hardware unexpectedly, for example, causing an increase in vibrations.

A good assessment process should also determine that the pros/cons balances of the proposed algorithms are positive.

The rover has 55 components whose failures can affect the mobility subsystem reliability.

It leads to 2^{55} possible failure combination. Moreover, in the case of unintended acceleration, the rover can hit the astronauts leading to the need to define functional safety requirements to continuously monitor that the correct torque is provided on the traction wheels.

Such complexity is no manageable, so the problem has been simplified by considering subdomains of it.

The detection algorithms have been divided following the hierarchy shown in fig.3.31. The active part has been divided into six subdomains, each one representing a single wheel with its steering system. From the conceptual perspective, the central wheels without steering capabilities have been considered a system where the associated failures never happen.

It simplifies the software architecture since designers can replicate the detection algorithm into 6 identical instances.

For each wheel, there are up to 10 components (see 3.10.1): 3 actuators (TM, TC,

³⁸Fail-operational behaviors with degraded operation level are often referred to as *graceful degradation*.

SM), 2 passive elements (TRG, SRG), and 5 sensors (H-TM, E-TM, E-SM, E-TGR, E-SGR). The possible 2^{10} (1024) possible failures combinations have been grouped into 13 *failure groups*, as defined in table 3.10. For each one of these, are indicated the involved components and the expected action at the rover level.

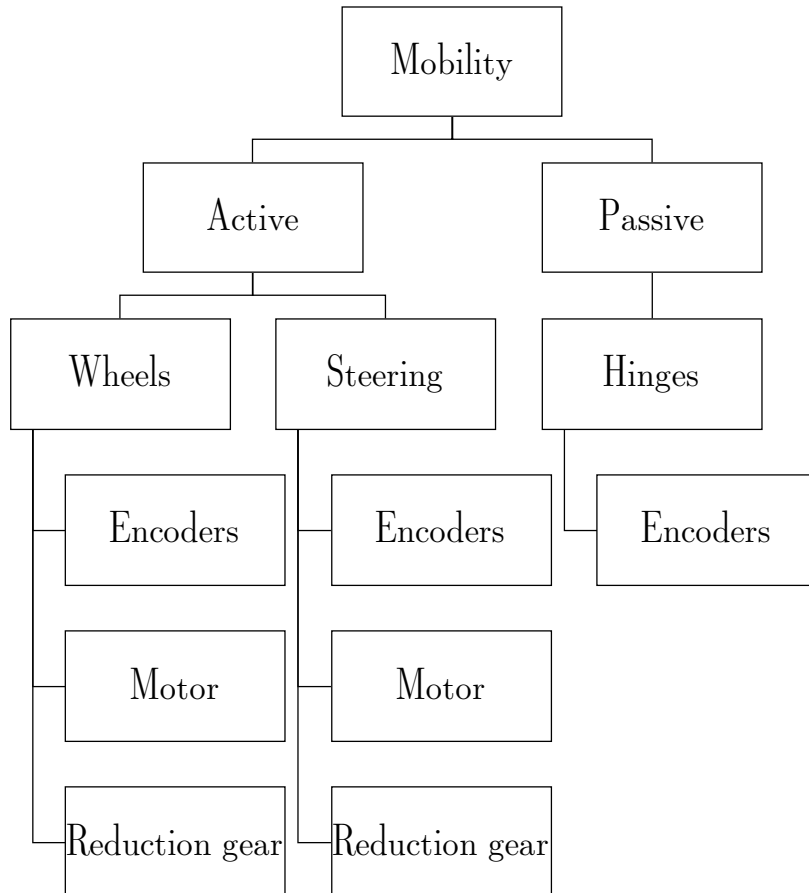


Figure 3.31: System level development process with the main involved documents.

In this proposal, the encoders on the rocker-bogie joints are not considered since, on the one hand, their relative positions depend on the terrain and, on the other, there are no redundancies of these measurements, so it is challenging to check onboard the rover if they measure correctly [62]. In any case, these encoders are installed in a better position with respect to the ones on the wheels and steering systems; hence their probability of failure is lower.

GroupTM #	TRG	TC	H-TM	E-TM	E-TRG	SM	SRG	E-SM	E-SRG	Action at the rover level
1	x	x	x	x	x	1	x	x	x	Fail
2	x	x	x	x	x	1	x	x	x	Fail
3	1	0	x	b	b	0	0	a	a	Fail (mitigation)
4	x	1	x	x	x	x	x	x	x	Fail
5	x	0	1	b	b	0	0	a	a	Fail (rover can work)
6	0	0	0	b	b	0	0	1	0	No degradation
7	0	0	0	b	b	0	0	0	1	No degradation
8	0	0	0	b	b	0	0	1	1	Fail
9	0	0	0	1	1	0	0	a	a	Fail
10	0	0	0	b	b	1	0	0	a	No degradation (but check configuration of the rover in order to find in the reduction is ok or not)
11	0	0	0	1	b	0	0	a	a	No degradation
12	0	0	0	b	1	0	0	a	a	No degradation
13	0	0	0	1	1	0	0	a	a	No degradation (use as motor feedback the reduction feed-back multiply by 50)

Table 3.10: Failures grouping summary.

The characters **a**, **b** and **x** in the previous table indicate what components can fail in each group:

components marked as **x** can fail simultaneously in any number between 0 and 9;

components marked as **a** can fail simultaneously in a maximum number of 2;

components marked as **b** can fail simultaneously in a maximum number of 3 for groups numbered 3,5,6,7,8 or in a maximum number of 2 for groups numbered 10,11,12.

Detection algorithms (traction subsystem)

To determine what are the affected components, detection algorithms are needed, while to define the rover state and hence to trigger the appropriate mitigations, it is needed to choose one of the 13 different groups described in table 3.10. Only the failures affecting the traction subsystem (for the steering subsystem see section 3.10.6) have been considered.

First level detection algorithms (traction subsystem)

For this purpose, we developed 6 different first-level detection algorithms, able to detect non-plausible measurements between couples of the three sensors involved (E-TM, H-TM, E-TRG) and the considered wheel velocity command. There are $C_{4,2} = \frac{4!}{2!(4-2)!} = 6$ possible combinations obtained by considering two sensors at a time, as shown in a matrix form in table 3.11. A sensor cannot disagree with itself, and the order does not make any difference, so the matrix is symmetrical. Checks are defined as comparisons between a difference and a threshold. When a check fails, the relative flag is set.

These differences and the thresholds can be different, coherently with the considered sensors couple:

- if both the inputs are equal to 0, it has no sense to perform a comparison;
- if one of the inputs is equal to 0 and the other not, their difference is compared with a threshold equal to 2 rpm³⁹;
- if both the inputs are different from 0, their difference, normalized by the first input, is compared with a threshold equal to 0.2 (corresponding to a 20% disparity).

For each one of those algorithms are indicated the flag name and the two sensors (or commands) involved in the comparison.

TF1 (E-TM/Wheel velocity command) The flag TF1 is raised when the difference between the wheel velocity command and the velocity measured by the encoder E-TM installed on the motor shaft is greater than the threshold.

TF2 (H-TM/Wheel velocity command) The flag TF2 is raised when the difference between the wheel velocity command and the velocity measured by the hall sensor H-TM installed inside the motor is greater than the threshold.

³⁹The 2 rpm threshold has been chosen to avoid possible divisions by 0

TF3 (E-TM/H-TM) The flag TF3 is raised when the difference between the velocity measured by the encoder E-TM installed on the wheel compared with the one measured by the hall sensor H-TM installed inside the motor is greater than the threshold.

TF4 (E-TM/E-TRG) The flag TF4 is raised when the difference between the velocity measured by the encoder E-TM installed on the wheel compared with the one measured by the encoder E-TRG installed on the reduction gear (divided by the reduction ratio of the gear itself to obtain a comparable value) is greater than the threshold.

TF5 (H-TM/E-TRG) The flag TF5 is raised when the difference between the velocity measured by the hall sensor H-TM installed on the wheel compared with the one measured by the encoder E-TRG installed on the reduction gear (divided by the reduction ratio of the gear itself to obtain a comparable value) is greater than the threshold.

TF6 (E-TRG/Wheel velocity command) The flag TF6 is raised when the difference between the wheel velocity command, compared with the one measured by the encoder E-TRG installed on the reduction gear (divided by the reduction ratio of the gear itself to obtain a comparable value) is greater than the threshold.

Mitigations (SW actions) triggers The table 3.11 recaps how the traction flags (TFs) are map.

	Wheel velocity command	E-TM	H-TM	E-TRG
Wheel velocity command	-	TF1	TF2	TF6
E-TM	TF1	-	TF3	TF4
H-TM	TF2	TF3	-	TF5
E-TRG	TF6	TF4	TF5	-

Table 3.11: Flags-failures associations.

These algorithms generate 6 flags, so we have 64 (2^6) possible combinations. We can group these, obtaining seven conditions, based on the number of set flags:

- All flags cleared: no plausibility problems are detected; hence there are no failures affecting the sensors; there is 1 combination of this kind.
- 1 flag set: the state is incoherent but tolerated with no actions; there are 6 combinations of this kind.
- 2 flags set: the state is incoherent but tolerated with no actions; there are 15 combinations of this kind.
- 3 flags set: 6 of them represent with certainty the failure of a component, while 14 are not sufficient to determine the affected sensors hence tolerated with no actions; there are 20 combinations of this kind.
- 4 flags set: no one of them is sufficient to represent with certainty the failure of a component but, if one or more subsets of three flags are one of the 6 sufficient to represent a failure, these failures are kept into account. If not, the wheel is considered as in the all flag set condition; there are 15 combinations of this kind.
- 5 flags set: each one of the possible combinations is sufficient to determine the components affected by faults. Hence, they are managed without considering the subsets of three flags; there are 6 possible combinations of this kind.
- All flags set: it is impossible to reconstruct the state from the sensors, so the affected wheel is no more monitored; there is 1 combination of this kind.

A list of the 14 managed combinations, with the indication of the determined components, can be found in table [3.12](#).

SW actions (mitigation algorithms)

To improve the reliability of the mobility system of the rover, designers have to develop algorithms able to choose the better strategy to obtain feedbacks from the encoders and hall sensors installed on the motor.

There are two kinds of mitigation strategies:

- *internal*: applied when it is possible to *isolate* the failure at the wheel level without involving the whole rover behavior or/and sensors of the other wheels;
- *external*, applied when it is not possible to manage this failure without involving behavior changes at the rover level and without use sensors feedbacks external from the affected wheel.

From these two kinds of strategies, four levels of mitigation have been defined, as shown in table [3.13](#).

Components	TF1	TF2	TF3	TF4	TF5	TF6
None	0	0	0	0	0	0
H-TM	0	1	1	0	1	0
E-TM	1	0	1	1	0	0
E-TRG	0	1	1	0	1	0
TM	1	1	0	0	0	1
TRG	1	1	0	0	0	1
TC	0	0	0	1	1	1
E-TM + H-TM	1	1	1	1	1	0
E-TM + E-TRG	1	0	1	1	1	1
E-TRG + H-TM	0	1	1	1	1	1
E-TM + TC	1	0	1	1	1	1
H-TM + TC	1	0	1	1	1	1
TM + TC	1	1	0	1	1	1
All encoders	1	1	1	1	1	1

Table 3.12: Flags-failures associations for all the components.

Legend	Type of mitigation algorithm triggered
0	no mitigation
1.1	internal mitigation (use another feedback value)
1.2	internal mitigation (clutch opening)
2.1	external mitigation (use external data, e.g. Slip)
2.2	external mitigation (change behaviour of the rover, e.g. Stop of the rover)

Table 3.13: Traction subsystem mitigation levels.

When one of the 13 managed groups is found by the detection algorithms, a state request and a mitigation level requirement are generated to trigger the appropriate mitigation algorithm.

These states (see table 3.14) are numbered to have the first four corresponding to single failures and consequentially the double and triple ones (all encoders failures).

In some cases, more than one fault-affected components group is associated with the same state. It happens for those having the same effects or required mitigation strategies.

Failure	State	Mitigation required
None	0	0
H-TM	1	1.1
E-TM	1	1.1
E-TGR	2	2.1
TM	3	1.2
TRG	4	2.2
TC	5	0
E-TM + H-TM	6	1.1
E-TM + E-TRG	6	2.1
E-TRG + H-TM	6	2.1
E-TM + TC	7	0
H-TM + TC	7	0
TM + TC	5	0
All encoders	8	2.1 or 2.2

Table 3.14: Failures-states-mitigations associations.

Internal mitigation algorithms (traction subsystem)

The internal mitigation algorithms are those that make use only of feedbacks coming only from sensors installed on the affected wheel. Three strategies are adopted to manage failures of H-TM, E-TM, and E-TRG.

H-TM It is propagated the best feedback, in terms of precision, between E-TM, divided by the gear ratio and E-TRG.

E-TM It is propagated the best feedback, in terms of precision, between H-TM, divided by the gear ratio and E-TRG.

E-TRG It is propagated the best feedback, in terms of precision, between E-TM and H-TM, both divided by the gear ratio.

External mitigation algorithms (traction subsystem)

The external mitigation algorithms are those that make use also of feedbacks coming from the sensors installed in the other wheels on the rover. Hence, it is needed to specify both the internal and the external actions.

TRG or TM (Internal) Opens the clutch (TC) to disengage the motor from the wheel.

(External) The traction control recomputes the torques to the other wheels considering the impossibility for the affected wheel to generate torque to move the rover.

TC (Internal) Stops the motor (only to save battery energy) since the TC failure prevents the motor from applying torque on the wheel.

(External) The traction control recomputes the torques to the other wheels considering the impossibility for the affected wheel to generate torque to move the rover.

H-TM + E-TM or E-TRG + H-TM or H-TM + E-MT (only one of the three) (Internal) It uses the still working speed sensor to feed the rover control loop and the motor speed control loop.

(External) Applies the "inverse kinematic" model of the rover to estimate the speed of the affected wheel to choose the encoder still working correctly and formulate the dual failure diagnosis.

All encoders (Internal) Uses the motor control in sensorless mode.

(External) Applies the "inverse kinematic" model to estimate the speed of the affected wheel to check if the motor is working correctly.

All encoders + TC and/or + TM and/or TRG (Internal) Stops the motor

(External) The traction control recomputes the torques to the other wheels taking into account the impossibility for the affected wheel to generate torque to move the rover.

Detection algorithms (steering subsystem)

For the steering subsystem, the detection algorithms are 3. There are only two encoders, one installed upstream (E-SM) the SRG, the other downstream (E-SRG).

SF1 The flag SF1 is raised when the difference between the angle required to the motor, the position read from the encoder on the motor (E-SM) and the angular position of the motor is greater to 2 rad.

SF2 The flag SF2 is raised when the difference between the angle measured from the encoder on the motor (E-SM) divided by 50 and the angle measured by the encoder on the reduction gear (E-SGR) is greater than 0.04 rad. We highlight that the value 0.04 rad is equal to 2 rad keeping into account the gear ratio of the steering system, so $2/50 = 0.04$.

SF3 The flag SF3 is raised when the difference between the angle measured by the encoder on the motor (E-SM) and the encoder on the reduction gear (E-SGR) is greater than 0.04 rad.

Mitigation (SW effects) triggers Unfortunately, it is possible to trigger a mitigation algorithm only for the cases of the first two rows of the table 3.15, while in the others is not possible, since the flags do not allow discrimination between the cases SM, SGR, E-SM + SM, E-SM + SGR, E-SGR + SM, E-SGR + SGR, and E-SM + E-SRG.

Component	SF1	SF2	SF3
E-SM	1	1	0
E-SGR	0	1	1
SM	1	0	1
SGR	1	0	1
E-SM + SM	1	0	1
E-SM + SGR	1	0	1
E-SGR + SM	1	0	1
E-SGR + SGR	1	0	1
both encoders unreadable, hence firmware reads 0)	1	0	1
Encoders (disagree between each other and with the command)	1	1	1

Table 3.15: Flags-failures associations for all the components.

Internal mitigations algorithms (steering subsystem)

E-SM Propagates the feedback given by steering gear reduction encoder E-SGR.

E-SGR Propagates the feedback given by steering motor encoder E-SM.

External mitigations algorithms (steering subsystem)

The absence of the clutch in the steering system collapses the external mitigations only into the rover stop action. To avoid false failure detections we decided to check also the coherence between the steering angles of the wheels. The failures with external mitigations are:

- SM
- SGR
- E-SM + SM
- E-SM + SGR
- E-SGR + SM
- E-SGR + SGR
- E-SM + E-SGR

3.10.7 Experimental setup

A suitable simulation environment to assess the embedded software detection capabilities is composed of CoppeliaSim [43] alongside MathWorks MATLAB/Simulink [59]. The interactions between them are shown in fig.3.32. Both mechanical and electronic components can be affected by faults, so their signals are modified by the *sabouter*.

CoppeliaSim is in charge to simulate the multibody (mechanical) components and the interaction between the wheels and the terrain [63].

Mathworks Simulink simulates sensors and electronics. The control, detection, and mitigation software have been developed through Model-Based Software Design (MBSD). It simplifies the integration between the environments: thanks to the C++ APIs exposed by CoppeliaSim, the algorithms can control the simulation. The Message Queue Telemetry Transport (MQTT) protocol has been adopted since it is used in the real rover to communicate with the base station.

3.10.8 Simulation results

Thanks to the simulation results, it is possible to analyze the software effects on the hardware, as required by [57]⁴⁰.

The focus of these simulations is to verify that the developed algorithms can detect

⁴⁰See the central table of the fig.3.32.

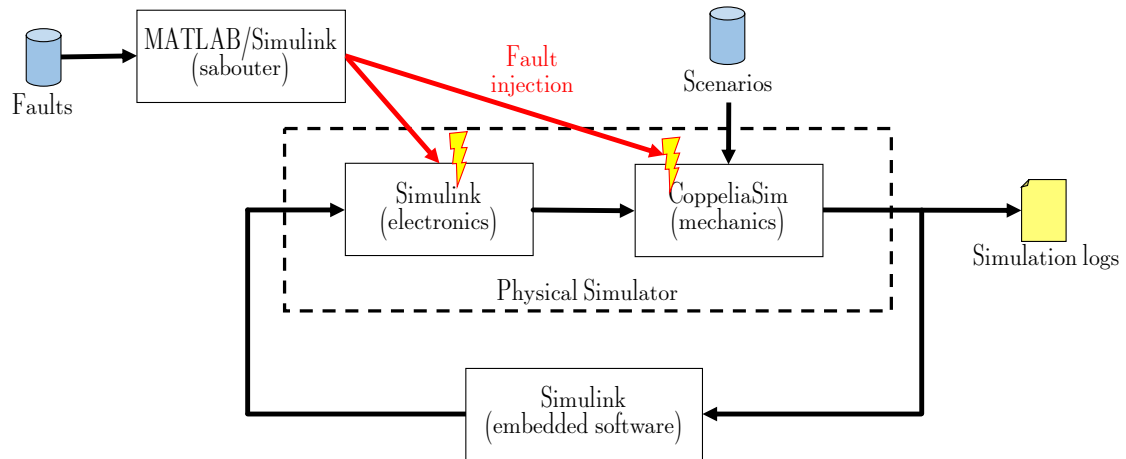


Figure 3.32: Three stages structure of the a general item.

the failures, while the mitigation effects are assessed manually by the D.I.A.N.A. team members and not shown in this dissertation.

Of course, it is possible to determine if the detection algorithms are capable of detecting the failures (and thanks to various scenarios, to determine if in some cases there are false positive or negative) and if the requirements in terms of Failure Time Tolerance Interval (*FTTI*) [7] are met.

The analysis is based on the time intervals defined in the *Timing aspects of functional safety* 2.3.4 section.

Achieved performances

In this subsection, the discussion is divided between the traction and the steering subsystems and, for each one of them, between the two detection levels (raise of flags and mitigation triggers). The detection algorithms that "raise the flags" are called with the name of the raised flag itself.

In the table 3.16 and table 3.17 are shown the results obtained from the simulations, respectively for the traction and the steering subsystem.

Injected Failure (s)	Injection time t_i [s]	Detection Time t_d [s]	Diagnostic time interval $T_{dti} = t_i - t_d$ [s]	Expected mitigation time $(t_i + FTTI)$ [s]	Mitigation time t_m [s]	Mitigation delay t_{md} [s]	Test results
H-TM	11.9	12	0.1	0	0	0	DETECTED
E-TM	10.5	10.7	0.2	0	0	0	DETECTED
E-TRG	8	8.1	0.1	0	0	0	DETECTED
TM	14.6	14.7	0.1	15.2	15.2	0	DETECTED
TRG	13.2	13.9	0.7	14.4	14.4	0	DETECTED
TC	7.7	12.5	4.8	13.5	0	∞	FAIL
H-TM + TM	12.4 s H-TM + 17.4 s TM	12.5 s H-TM + 17.5 s TM	0.1 s H-TM + 0.1 s TM	0 s H-TM + 18 s TM	0 s H-TM + 18 s TM	0 s H-TM + 0 s TM	DETECTED
H-TM + TRG	18.6 s H-TM + 20.6 s TRG	18.7 s H-TM + 20.9 s TRG	0.1 s H-TM + 0.3 s TRG	0 s H-TM + 21.4 s TM	0 s H-TM + 21.4 s TRG	0 s H-TM + 0 s TRG	DETECTED
H-TM + TC	5 s H-TM + 13.6 s TC	5.3 s H-TM + 15.2 s TC	0.3 s H-TM + 1.6 s TC	0 s H-TM + 16.2 s TC	0 s H-TM + 0 TC	0 s H-TM + ∞ TC	DETECTED
H-TM + E-TRG	4.7 s H-TM + 22.1 s E-TRG	4.8 s H-TM + 22.2 s E-TRG	0.1 s H-TM + 0.1 s E-TRG	0 s H-TM + 0 s E-TRG	0 s all + 0 E-TRG	0 s H-TM + 0 s E-TRG	FAIL

Injected Failure (s)	Injection time t_i [s]	Detection Time t_d [s]	Diagnostic time interval $T^{dti} = t_i - t_d$ [s]	Expected mitigation time $(t_i + FTTI)$ [s]	Mitigation time t_m [s]	Mitigation delay t_{md} [s]	Test results
E-TM + E-TRG	13.8 s E-TM + 21.9 s E-TRG	13.9 s E-TM + 22 s E-TRG	0.1 s E-TM + 0.1 s E-TRG	0 s E-TM + 0 s E-TRG	0 s E-TM + 0 s E-TRG	0 s E-TM + 0 s E-TRG	DETECTED
H-TM + E-TM	9 s H-TM + 15.2 s E-TM	9.1 s H-TM + 15.3 s E-TM	0.1 s H-TM + 0.1 s E-TM	0 s H-TM + 0 s E-TM	0 s H-TM + 0 s E-TM	0 s H-TM + 0 s E-TM	DETECTED
E-TM + TM	7.5 s E-TM + 14.4 s TM	7.6 s E-TM + 14.5 s TM	0.1 s E-TM + 0.1 s TM	0 s E-TM + 15 s TM	0 s E-TM + 15 s TM	0 s E-TM + 0 s TM	DETECTED
E-TM + TC	10.7 s E-TM + 17.8 s TC	10.9 s E-TM + 17.8 s TC	0.2 s E-TM + ∞ TC	0 s E-TM + 18.8 s TC	0 s E-TM + 0 s TC	0 s E-TM + ∞ TC	FAIL
E-TM + TRG	6.8 s E-TM + 13.1 s TRG	6.9 s E-TM + 13.8 s TRG	0.1 s E-TM + 0.7 s TRG	0 s E-TM + 14.3 s TRG	0 s E-TM + 14.3 s TRG	0 s E-TM + 0 s TRG	DETECTED
E-TRG + TM	4.5 s E-TRG + 15.3 s TM	4.6 s E-TRG + 15.4 s TM	0.1 s E-TRG + 0 s TM	0 s E-TRG + 15.9 s TM	0 s E-TRG + 15.9 s TM	0 s E-TRG + 0 s TM	DETECTED
E-TRG + TRG	7.2 s E-TRG + 12.6 s TRG	7.3 s E-TRG + 12.7 s TRG	0.1 s E-TRG + 0 s TRG	0 s E-TRG + 13.7 s TRG	0 s E-TRG + 13.8 s TRG	0 s E-TRG + 0.1 s TRG	DETECTED

Injected Failure (s)	Injection time t_i [s]	Detection Time t_d [s]	Diagnostic time terval $T_{dti} = t_i - t_d$ [s]	Expected mitigation time $(t_i + FTTI)$ [s]	Mitigation time t_m [s]	Mitigation delay t_{md} [s]	Test results
E-TRG + TC	8.7 s	E- 8.8 s	E- 0.1 s	0 E-TRG + 16 s TC	0 s E-TRG + 0 s TC	0 s E-TRG + ∞ TC	FAIL
TM + TC	7.9 s	TM + 8 s	0.1 s	8.5 s TM + 14.2 s TC	8.5 s TM + 0 s TC	0 s TM + ∞ TC	DETECTED
TC + TRG	2.4 s	TC + 2.4 s	0 s	3.4 s TC + 16.4 s TRG	15.9 s TC + 16.9 s TRG	12.5 s TC + 0.5 s TRG	FAIL
TRG + TC	13.9 s	TRG + 14.6 s	0.7 s	15.1 s TRG + 17.8 s TC	15.1 s TRG + 0 s TC	0 s TRG + ∞ TC (not needed)	DETECTED
All sensors	10.8 s	H- 10.9 s	0.1 s	0 H-TM + 0 E-TM + 16.4 s E-TRG	0 H-TM + 0 E-TM + 17.6 s E-TRG	0 H-TM + 0 E-TM + 0 s E-TRG	DETECTED

Table 3.16: Simulation results for the traction subsystem.

Injected Failure (s)	Injection time t_i [s]	Detection Time t_d [s]	Diagnostic time interval $T_{dthi} = t_i - t_d$ [s]	Expected mitigation time $(t_i + FTTI)$ [s]	Mitigation time t_m [s]	Mitigation delay t_{md} [s]	Test results
SM	7.1	12.8 (an- gle quite constant)	5.7	13.3	15.8	2.5	DETECTED
E-SM	11.6	14.2 (an- gle quite constant)	2.6	0	0	0	DETECTED
E-SRG	7.5	13.1 (an- gle quite constant)	5.6	0	0	0	DETECTED
E-SM + E-SRG	8.3 s E-SM + 15.7 s E-SRG	10.6 s E-SM + 17.8 s E-SRG	2.6 s E-SM + 2.1 s E-SRG	16.2	18.6	2.4	DETECTED
E-SM + SM	8.4 s E-SM + 15.8 s SM	11.1 s E-SM + 17.2 s SM	2.7 s E-SM + 1.4 s SM	0 s E-SM + 17.7 s SM	0 s E-SM + 20 s SM	0 s E-SM + 2.3 s SM	DETECTED
E-SRG + SM	6.9 s E-SRG + 12.8 s SM	9.1 s E-SRG + 16 s SM	2.2 s E-SRG + 3.2 s SM	0 s E-SRG + 16.5 s SM	0 s E-SRG + 17.6 s SM	0 s E-SRG + 1.1 s SM	DETECTED

Table 3.17: Simulation results for the steering subsystem.

Discussion of the results

For each one of the wheels, we found 9 (6 for the central ones) possible failures (physical components that can misbehave) and 18 (15 for the central one) combinations of failures.

From the detection point of view, simulations show that 22 of them (75 %) are correctly detected, while 5 of them (25 %) not. These 5 undetected ones are all from the traction subsystem and are the ones where the TC is involved. In any case, thanks to the simulation results, we found out that these failures cannot cause critical misbehaviors. The traction clutch (TC) failure makes it impossible to engage it and connect the motor to the reduction gear (TRG), hence to the wheel itself. The loss of the ability of a motor to deliver torque does not make it impossible to keep moving the rover since it has an advanced traction control system, 5 other running wheels, and moves at low speed.

3.11 Summary

The six methodologies proposed in this section evolved from the first three proposals⁴¹ where only the *item* is simulated, to the fourth [19], fifth [40], and sixth [21] where the failure effects are propagated to the whole vehicle (or system).

In the first proposal [15], the simulations are performed at the item-level by manually injecting the failure modes inside the SPICE-level circuit schematic, then using a *classifier* script to classify the FMs effects based on the simulation results.

The second proposal, described in [16], introduces a *fault list generator module*, an automatic *sabouter* to perform the injection of FMs, and a *report generator*. The results obtained by adopting this methodology have been compared with the ones obtained by a company following the usual handmade process, as described in the third proposal based on the paper [17].

To improve the classification quality, a model of the *actuator* (in this specific case, a BLDC motor) has been added to the simulation, as described in the paper [18] contained in the second proposal. This level of integration allows performing the simulation-based FMEDA also on items in charge of closed-loop control systems. It allowed assessing the failure detection capabilities of the embedded software, but the analysis on the effectiveness of the embedded mitigation strategies is limited to fault isolation ones since the approach cannot predict their vehicle-level effects.

In the fourth proposal, published in the paper [18], the methodology has been extended to propagate the actuators actions to the entire vehicle, thanks to the

⁴¹The proposal [15] is described in section 3.5 while [16] and [18] are described in section 3.6 and have been tested against an handmade one in [17]. This comparison between the simulation-based and handmade FMEDA is discussed in the section 3.7

introduction of a vehicle(*system*)-level simulator, allowing modification of the classification rules, moving them from the item-level up to the vehicle(*system*)-level. Thanks to this approach, it is possible to assess, other than the detection (structurally limited within an item), also the mitigation capabilities (at the vehicle-level) of the embedded software. A limitation of this proposal is that it is not possible to assess the embedded software capabilities to mitigate failures external to the item itself since only its components are subjected to fault injection.

In the fifth proposal, the approach proposed in the fourth one [19] has been adapted to an industrial case [40]. The novelty of this proposal is the adoption of a mixed-level simulation system, with *behavioral* and *structural* models running alongside to simulate the cyber-physical system as a whole. The classification rules have been modified to keep into account the criticality of the failures since the considered system is mission-critical.

This proposal lacks an objective assessment of the software mitigation capabilities at the system level since the focus of this paper is to investigate the failure masking effects due to compensations provided by the nominal functionality closed-loop controllers.

In the last, sixth methodology [21], conceptually obtained by merging the fourth and fifth proposals, two limitations have been overcome: it is possible to inject failures in the actuators and mechanical components (limitation of the fourth) and assess⁴² the mitigation capabilities of the embedded software at the system level (limitation of the fifth).

A future improvement can be implementing an automatic FMs classification system based on rules defined at the system level.

⁴²At the moment this is a manual process aided by the simulation results.

Chapter 4

Simulation-based HARA

The Hazard Analysis and Risk Assessment (HARA) is a phase, required during the concept phase (see section 2.2) of the ISO26262, is known to lack in *objectivity* and *repeatability* [64].

To improve these weak points, a novel approach based on a vehicle-level simulator has been proposed. It has been published into two papers: [65] and [66]. [66] has been released in 2020 on the journal Elsevier Microelectronics Reliability and it is an extended version of [65], presented at the International Symposium on On-Line Testing and Robust System Design (IOLTS) in 2019 and published in its proceedings.

HARA is crucial to determine the *hazards* that the *item* may face during its operations and assess their associated *risk* level. As already discussed in section 2.2.2, for each hazard, three risk parameters have to be determined: *severity* (S), *exposure* (E), and *controllability* (C)¹. Once these have been obtained, an Automotive Safety Integrated Level (ASIL) has to be assigned to each of the safety goals considering, for each of them, the most severe hazard to which its violations expose the people inside or surrounding the vehicle.

Like autonomous and semiautonomous vehicles, those equipped with Advanced Driver Assistance Systems (ADAS) can be considered a subset of autonomous and intelligent systems (A/IS), so their testing is more challenging with respect to the usual cars.

On the one hand, it is needed to test if the sensing systems are able to perceive,

¹For those vehicles classified at level 4 of the SAE classification (see section 1.2.1), or as *automated* in the Edge Case Research classification (see section 1.2.2), the *controllability* parameter is not significative since the human driver is not involved in guaranteeing the safety of the driving function.

with sufficient precision and *situational awareness*², the external world.

On the other hand, the algorithms in charge of making safety-relevant decisions are usually developed following the machine learning (ML) approach. It makes them not completely deterministic. An example can be neural networks trained for a computer vision application like obstacle recognition. There is no single, correct outcome for these algorithms, but a set of slightly different but equally correct output set. It makes the execution of unit tests (see section 2.11.4) difficult when not impossible. In this case, a test has to be considered as passed when the chosen behavior keeps the car into a safe situation, failed otherwise.

The application of the ISO 26262 requirements is not sufficient: it considers only the functional safety aspects, but it is also necessary to verify the risk level associated with the nominal fault-free³ behavior. To help safety engineers dealing with non-deterministic strategies, as presented in sections 1.9 and 1.10, two other standards have been released: the ISO/PAS 21448 (SOTIF) and the UL 4600.

SOTIF covers how to avoid dangerous behaviors emerging from non-deterministic outcomes of the system, due to situations not foreseen in drafting or requirements definition for which the neural network is not trained. Other sources of failures can be the aging of the sensors, incorrectly designed HMI, or users' excessive confidence in these systems. The UL 4600 better describes how to obtain a *Safety Case* as completely as possible.

The increasing complexity of the ADAS is making it more challenging to perform HARA. These items require high-performance Electronic Control Units (ECU) with complex software to perform their functionalities.

To operate correctly, they interact with the driver, environment, and other vehicle functions through high-speed in-vehicle networks, as well as a wide range of sensors and actuators. As a result, they implement complex behaviors whose outcome, in the presence of failures, is not trivial to identify and classify as requested by the functional safety standards.

The methodology proposed in this chapter aims to:

- improve the *objectivity* of the risk assessment process, thanks to the combined usage of *risk parameters classification table* (current state of the art to increase the HARA reliability) and simulation results;

²I borrowed this term from psychology. It has to be intended as the level of adherence between the external world surrounding the vehicle and its representation provided by the various data fusion algorithms the vehicle is provided.

³Biased or incomplete training data, or lack of training in some situations, can be considered as a systematic error, but their meaning is different with respect to the one described in the ISO 26262, since, from the technical point of view, the specifications are fully satisfied. Hence, how to deal with these situations is not, strictly speaking, a Functional Safety problem.

- increases the *repeatability* of the overall HARA process since the vehicle-level simulator allows to make it less dependent on the safety engineers knowledge;

To demonstrate, benchmark, and better explain the methodology, it has been applied to a well-known industrial case study by performing simulations with the same tests that are used for the validation of items of the same kind in road tests⁴. As described in the following, the benchmark case study is an *Advanced Emergency Braking System* (AEBS).

The rest of the chapter is organized as follows. Section 4.1 describes the state of the art for the topics not already discussed in section 2.2.2 and the research contributions of this proposal. Section 4.3 discusses about the AEBS benchmark results, while section 4.4 shows the simulation results.

The conclusions, as already did for the proposals about the simulation-based FMEDA, are reported in section 6.2 of the concluding chapter of this dissertation.

4.1 State of the art

Before starting with the description of the literature on which this approach is based, it is useful to provide a generalized ADAS device model. These systems are composed of:

- sensing systems (sensors);
- a *data fusion* (DF) algorithm to merge the information, coming from all the sensors, into a unique and coherent virtual representation of the surrounding environment;
- the *logic* algorithm that takes the opportune responses based on the virtual representation of the external world;
- actuators that physically apply the responses from the *logic* algorithm on the physical environment.

This typical structure is shown in fig.4.1:

The main issue about the HARA regards its validity (repeatability) and reliability (objectivity) [64].

As already discussed in the section 2.2.2, HARA is conducted in five phases, as shown in fig.4.2:

1. Situation analysis and hazard identification (SA/HI);

⁴It has been decided to apply the novel methodology to a well-known case to make it possible comparing the results obtained with this proposal with the ones found in the literature.

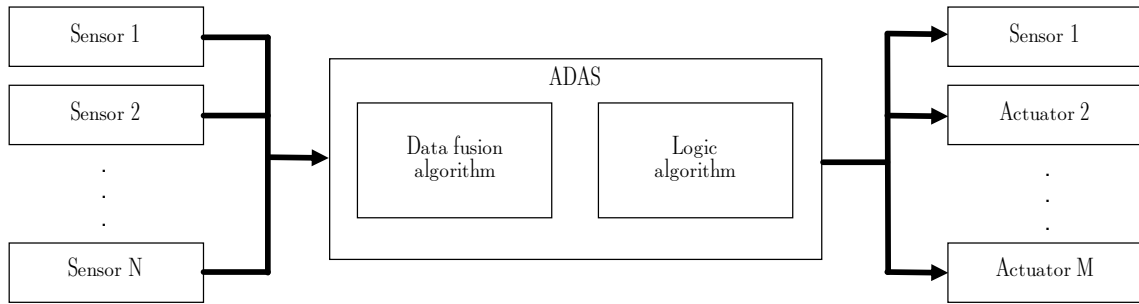


Figure 4.1: The structural block diagram of an ADAS device. Figure from [66].

2. Hazard classification (HC);
3. ASIL determination;
4. Safety objective definition;
5. Review.

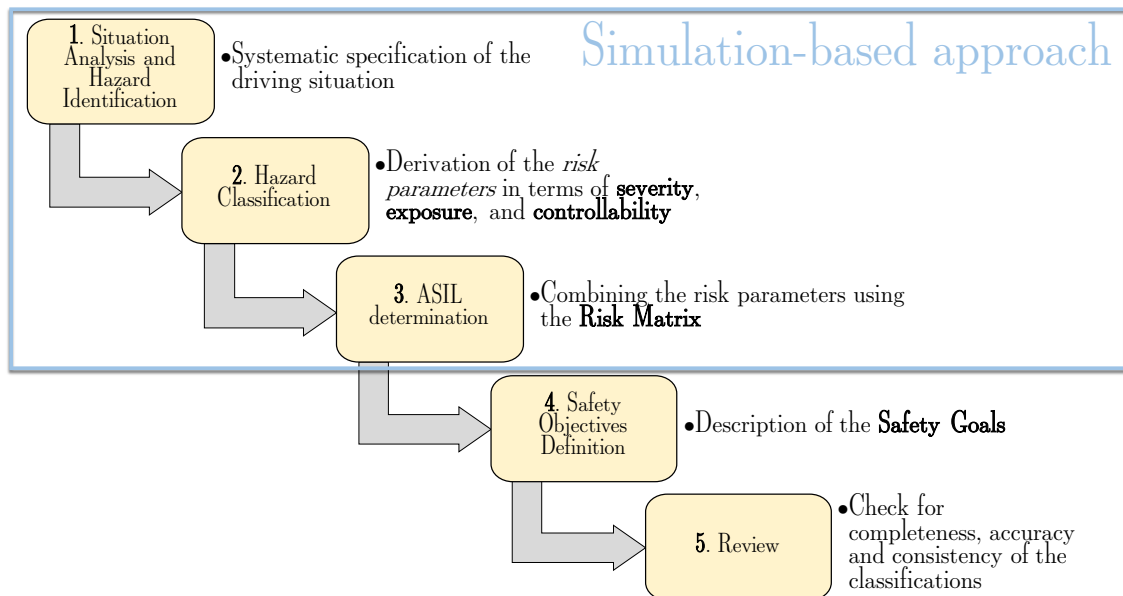


Figure 4.2: The phases of the HARA, with indication of those ones that can be performed by a simulation-based approach.

4.1.1 Research contribution

Structured methodologies to improve validity (repeatability) and reliability (objectivity) of the HARA analysis have been proposed in [67] and [68]. Even if these

works propose different approaches, they share the common goal of making HARA more repeatable and objective by making it less dependent on the background of the involved safety engineers.

Vehicle-level simulators are adopted in the automotive industry for software verification [69]. There are various off-the-shelf tools capable to aid designers during the concept phase of the development, like IPG CarMaker™ [70], AVL™ Vehicle Simulator (VSM™) [71], FEV™ VirtualDynamics™ [72], and CARLA [73], an open-source simulator for autonomous driving research that allows to use the Scenic language [74] to model and generate static scenes.

The novelty of this paper is focused on how it is possible to aid the first three phases (SA/HI, HC, and ASIL determination) thanks to a simulation-based approach.

Regarding the SA/HI phase, it had been shown that, since only the actuators can act on the environment, a good way to obtain a suitable hazard list for an item is to analyze the actuator-level possible misbehaviors [75], regardless of the intermediate stage that caused it. Relying on this hypothesis, it is possible to perform simulations even when the item is defined only at its behavioral level. The interested reader can find a similar way, based on the high-level item description, in the paper [76].

Considering the HC phase, all the papers cited until now use *risk parameters classification tables*. These associate numerical values ranges of the physical dimensions of interest, obtained from the simulations, with the ISO 26262 risk parameters levels, in terms of severity (see table 2.3) and controllability (see table 2.4), adapted to classify for the peculiarities of the considered application. The same approach is adopted in this proposal by comparing the ranges indicated in the *classification tables* with the values obtained from the simulations.

This proposal has been helpful to other scholars.

In [77] the methodology has been proposed as a solution to determine architectural and design requirements methods suitable to assure that the examined system operates in a safe state by preventing faults propagations, in particular in those cases where erroneous values affect critical signals.

Moreover, a survey on methods for the Safety Assurance of Machine Learning Based Systems [78] indicates this proposal as a solution to perform HARA through the use of vehicle-level simulators to test initial specification while, in the SAE technical paper [79] it is included into a safety analysis and verification framework for autonomous vehicles based on the identification of *SOTIF triggering events* (see fig.1.9).

4.2 Proposed methodology

The proposed methodology derives from the ones published in [10] and [18], already presented in section 3.6, and [19], discussed in section 3.8.

These two proposals aim to aid, by a simulation-based approach, the FMEDA. The first proposal discuss the fault simulation within the item boundaries, while the latter describes how propagate misbehaviors⁵ from the outputs of the item to the entire vehicle by a vehicle-level simulator.

The FMEDA is performed at the end of the hardware design phase⁶ to verify that the obtained schematics are suitable to reach the required reliability level. This level depends on the most strict ASIL, between the ones assigned to the considered safety goal, that the item can violate due to a random hardware failure.

The proposed methodology is represented as a block diagram in fig.4.3. Moreover, it can be helpful to map it with the HARA phases described in the section 2.2.2, and shown in fig.4.2. This mapping is shown, superimposed on the block diagram, in fig.4.4.

⁵In the case of the FMEDA it is more precise to call them *ailure modes effects*.

⁶Regulated by the part 5 of the ISO26262.

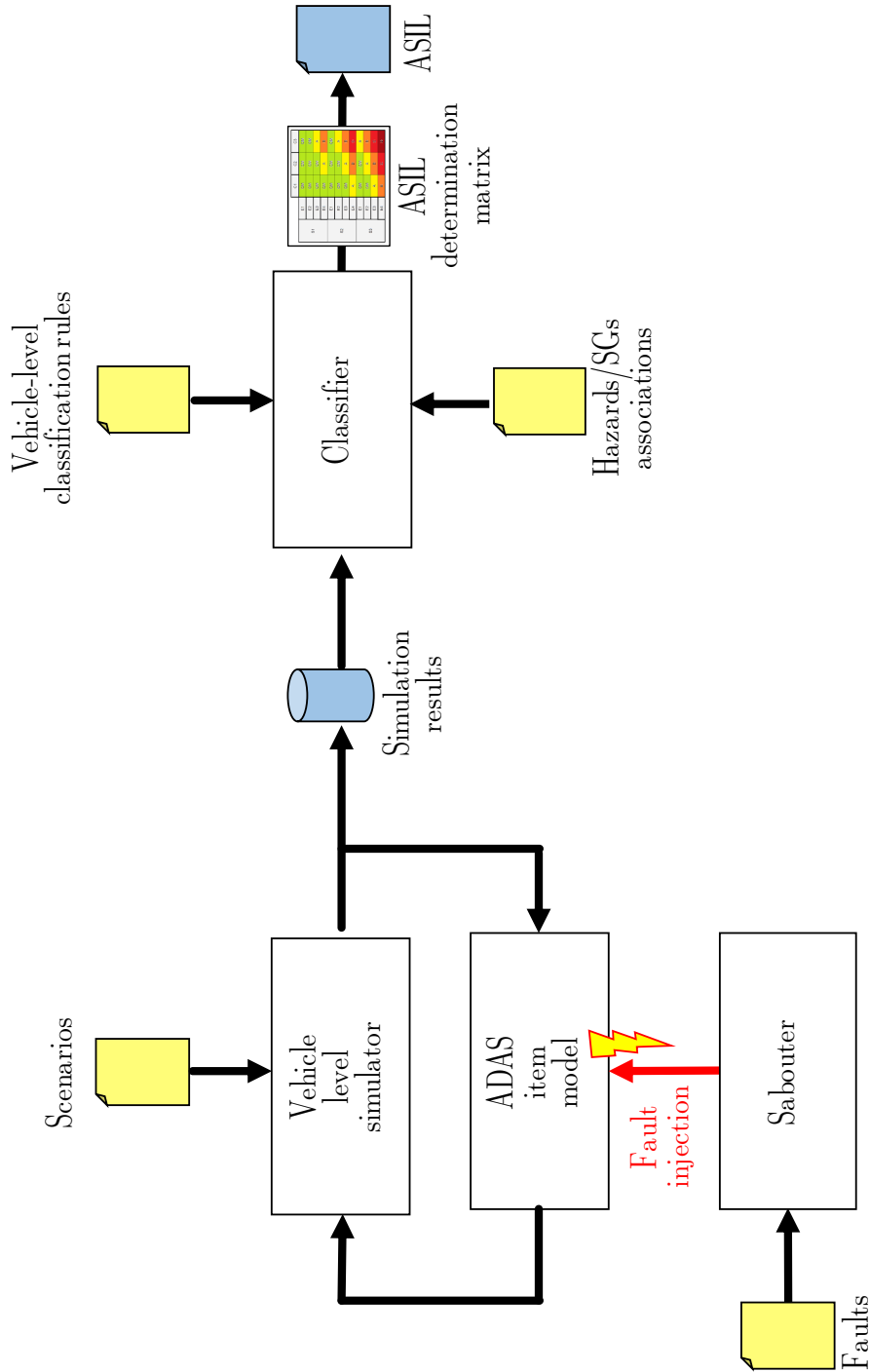


Figure 4.3: Block diagram representation of the proposed approach.

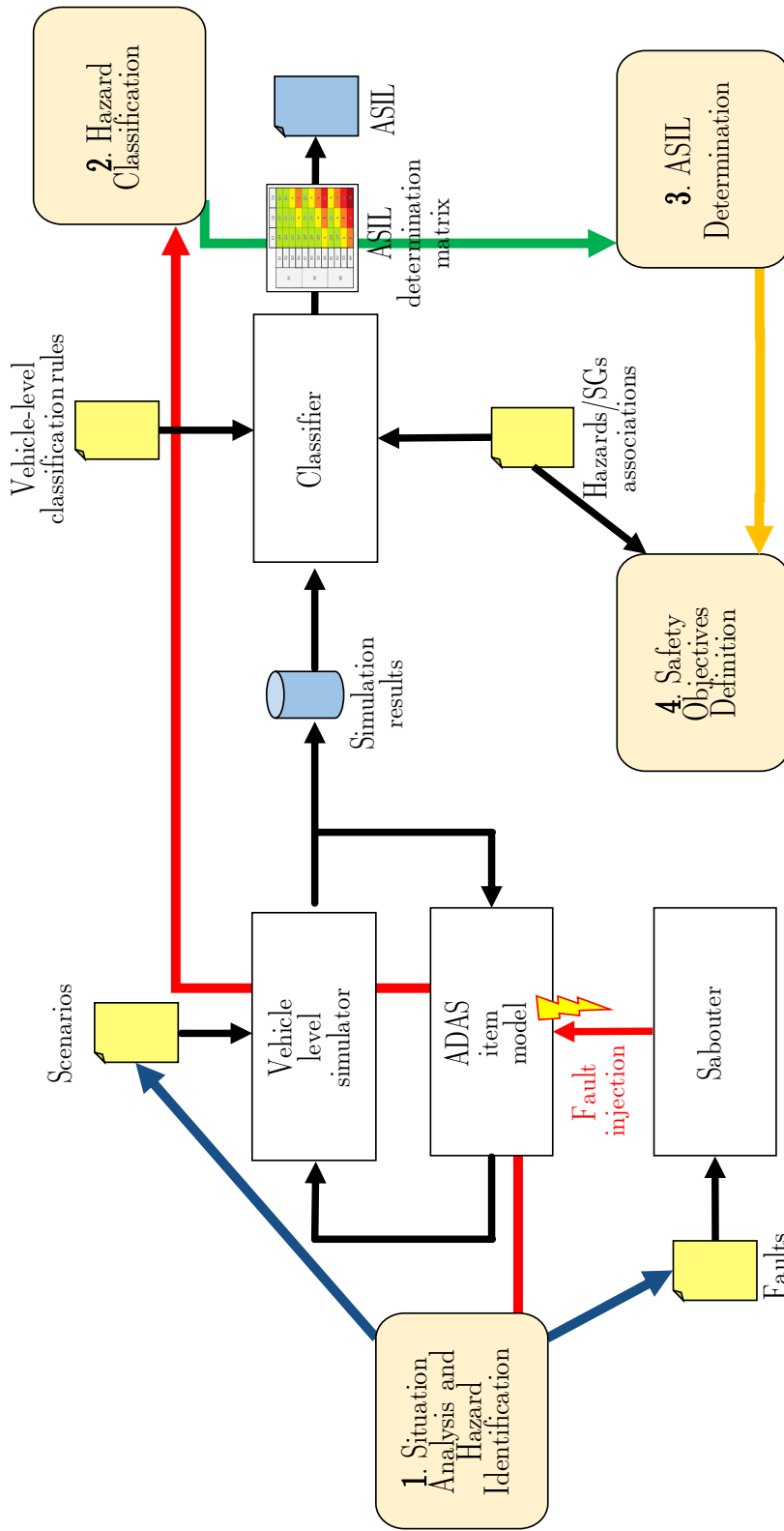


Figure 4.4: Block diagram representation of the proposed approach, with the HARA phases, mapped to its blocks.

4.2.1 Situation Analysis and Hazard Identification

The first phase is the SA/HI. The SA sub-phase is performed by preparing various *scenarios* in a format suitable to the chosen vehicle-level simulator. The behaviors of the simulated vehicle, obtained by simulating the item, thanks to its *behavioral model* in nominal (non-faulty) conditions, are stored into a *simulation results* database. These results can be useful, during the situation analysis, to aid safety engineers to imagine, and add, newer possible situations not already considered but that it is possible to find, in the real world, during the vehicle lifetime.

Moreover, this information is useful during the HC phase to define classification rules based on the differences between the fault-free and fault-affected behaviors. The interested reader can find a better description of *behavioral* models in section 3.9, where differences-based rules are adopted to accelerate the FMEDA.

4.2.2 Hazard classification

It is important to remark that the HARA phase is completed during the concept phase before the item is designed. The lack of schematics makes it impossible to know the exact way (in terms of components' failure modes) and hence the actual behavior in the case a random hardware failure happens⁷.

Thanks to *behavioral* models of the item and the vehicle-level simulator, it is possible to assess the worst-case consequences of some of the failures (those for which it is possible to describe the behavior of the item in the case they happen), on the vehicle, evaluating their severity. The failures are injected by the *Sabouter*, following the instructions contained in the *faults* descriptions.

Simultaneously, by analyzing the effects of the failures on the vehicle acceleration and trajectory, it is possible to assess the average human driver's capability to mitigate them, aiding the controllability determination.

A similar approach has been proposed in [80], where it is described how to assess the controllability considering, as the benchmark application, an electric power steering device.

The hazards are classified by comparing the simulation results with the classification tables for controllability and severity risk parameters. Considering what is shown on the figures 4.3 and 4.4, these classification tables are part of the *vehicle-level classification rules*.

⁷To determine the failure modes list, it is necessary the BOM, not yet available

4.2.3 ASIL determination

After the risk parameters classifications have been obtained, ASILs can be preliminarily assigned to the hazards. At this point, by hands, the hazards are associated with safety goals violations, and finally assigned to SGs⁸ considering their most dangerous associated hazard.

The ASIL classification is obtained combining the three risk parameters S, C, and E, through the so-called *ASIL determination matrix*, shown in fig.2.5.

4.2.4 Safety objective definition

In this phase, performed manually, the safety goals are defined and associated with the hazards. This association is represented as the *Hazards/SGs associations* in the figures 4.3 and 4.4.

The proposed simulation-based approach cannot aid the determination of the exposure parameter. In any case, section 4.4 discusses how the exposure levels have been obtained for the considered benchmark application.

4.2.5 Review

The review phase is entirely performed by hand. The ISO 26262 confirmation measures (see section 2.1.3) require, for this phase, other safety engineers, different from those who performed the HARA.

4.2.6 Summary

The key points of the proposed approach are the following:

- it bases the hazard analysis on simulation scenarios, thanks to the capability of the vehicle-level simulator to represent and simulate different driving situations and surrounding environments;
- it allows, thanks to the simulation results, the decoupling between the knowledge of the safety engineers and the final assessment, improving the *objectivity* of the result;
- thanks to the semi-formal models and the determinism of the simulation results, it is possible to obtain numerical values that improve the *repeatability* of the severity and controllability assessment.

⁸The ASILs, as stated by the ISO26262, are assigned to Safety Goals. Any other preliminary association can be done just in function of the assignment to a safety goal.

From the implementation point of view to set-up an environment suitable for this methodology, this minimum set of components is necessary (see fig.4.3):

- the *behavioral* model of the item (in this case an ADAS) under assessment;
- a vehicle-level simulator;
- a model of the vehicle under test (VUT);
- a software layer to put in communication the model of the ADAS with the vehicle-level simulator;
- a set of *scenarios*, as semi-formal descriptions provided in a format compatible with the chosen vehicle-level simulator⁹;
- a semi-formal *behavioral* description of the possible failures of the item under assessment;
- *vehicle level classification rules*, in this case expressed as classification tables for the *severity* and *controllability* assessment.

4.3 Benchmark case study

4.3.1 AEBS and its integration into the vehicle

AEBSs are designed to reduce the risk of a collision. Their functionality is based on sensors, able to measure the gap between the vehicle and the nearest object in front of it, usually implemented by RADAR technology.

It can work as a standalone system, usually at urban street speeds (< 60 km/h) or, alongside the cruise control system, at every speed, helping the driver to keep the right safety distance.

When it works in standalone mode, it provides the driver visual and sound warning and, if the driver does not react, it performs an emergency brake.

A regular cruise control, a.k.a. *cruise control system* (CCS), can maintain a constant speed, chosen by the driver, without requesting intervention on the throttle pedal. When the driver presses the brake or the clutch pedal, the CCS is suspended, and the system does not provide torque requests anymore. A more complex version, commercially called *Adaptive Cruise Control System* (ACCS), is able to maintain the safety distance, from the preceding vehicle, by automatically adapting its speed setpoint. The braking requirements of ACCSs are defined into the ISO 15622 [81]. The ACCS does not guarantee safety: it is considered a comfort function. Its only

⁹Usually, vehicle-level simulators embed a tool to design the scenarios.

safety requirements is related to its disengagement, which is usually classified as ASIL C.

In the considered case study, the vehicle-level simulator generates the nearest object distance measurement and receives the braking force required by the ADAS model. The driver behavior is part of the scenario file. All the physics simulations are in charge of the vehicle level simulator.

4.3.2 Fault model

A *fault model* is needed to perform fault injections. Since the item considered in this proposal is still in its concept phase, it is impossible to describe detailed fault models, but only to provide *behavioral* faults descriptions. In this case, the misbehavior of the item is described as a condition in which it stops applying braking force on the vehicle without providing any warning to the driver.

4.3.3 The proposed methodology

Situation Analysis and Hazard Identification

During the first stage of the methodology (corresponding to the SA/HI phase), a system composed of a vehicle-level simulator and a *behavioral* semi-formal model of an ACC with AEBS capabilities (ACC/AEBS) is set up.

These scenarios have been prepared for the considered benchmark application, starting from standard tests for the AEBS: for this class of item, are available road tests from European New Car Assessment Programme (EuroNCAP) [82], NHTSA [83], and European Commission [84]. All of them provide descriptions of significant driving situations, i.e., operational situations.

Simultaneously, a *behavioral* model of the item, capable of performing its functionalities inside the simulated environment, is prepared. Moreover, if needed, it has to be instrumented to allow faults injections. Possible ways to instrument the item model can be *sabotaging blocks*, similar to those presented in section 3.6.1. In the considered case, the only considered faulty behavior is described as a complete lack of the item functionality, so it is injected by forcing the braking force it requests to 0.

The first bench of simulations, in fault-free conditions, is run by the *vehicle-level simulator* alongside the *ADAS item model*. The obtained simulation results can be helpful to the HC stage if necessary to compare these golden behaviors with the ones obtained in faulty conditions. In this benchmark application, this kind of classification rules is not exploited.

Hazard Classification

During the second phase, *hazard classification* (HC), thanks to the vehicle behavior¹⁰ obtained from the simulations, the *severity* and the *controllability risk parameters* are evaluated by comparing the simulation results with the *classification tables*. The preparation of the tables is a manual activity, while the application of the vehicle-level classification rules is performed automatically by the *Classifier*, based on the content of the classification tables 4.1 and 4.2.

Classification tables These classification tables have been adapted from [64] to the considered case: since it is an ACC/AEBS, are considered only those involving two vehicles driving in the same direction. Associations from relative speeds of the vehicles and the *severity* are summarized in table 4.1.

Severity	Relative speed [km/h]
S0	< 21
S1	≥ 21 and < 26
S2	≥ 26 and < 36
S3	≥ 36

Table 4.1: Severity classification rules. Table from [66].

From the controllability point of view, the time to collision (TTC) is considered the classification parameter, considering those cases where the driver misuses the emergency braking functionality: he/she relies on the system without looking at the preceding vehicle behavior.

These rules are summarized in table 4.2

Severity	TTC [s]
C1	≥ 3 (from [84])
C2	≥ 3 and < 4
C3	≥ 4

Table 4.2: Severity classification rules. Table from [66].

¹⁰As prescribed by the ISO 26262 part 3 (Concept phase), during the HARA, is forbidden to consider failure effects mitigation systems.

To the best of my knowledge, the assessment of the *exposure risk parameter* cannot be performed by a simulation-based approach, so the values have to be associated, by hands, with each one of the scenarios.

ASIL determination

Once all the risk parameters have been assessed by comparing the simulation results with the values contained in the *classification tables*¹¹, it is possible to determine an ASIL level, through the so-called *ASIL determination matrix*, shown in fig.2.5.

This ASIL level is then associated, by hands, with the involved hazards.

Safety objective definition

The only SG considered in this case study is defined as *the ACC/AEBS brakes when there are obstacles in front of the vehicle*. The hazard associated with this SG is the possibility of hitting the preceding vehicle due to the driver's overconfidence in this system.

The other SG, associated with the hazard in which the item leads to an unintended braking action, has not been analyzed.

4.4 Simulation results

The simulation system implemented to obtain the data contained in this chapter is composed of:

- the vehicle level simulator (IPG Automotive Carmaker [70]);
- a model of a full-size sedan car;
- the ACC/AEBS semi-formal model (provided as a MathWorks Simulink model [59]), instrumented to force its breaking force request to 0;
- a classifier to extract, from the simulation logs, the relative speed between the vehicles at the moment of the crash and the time to collision (TTC). It has also to assign the correct levels, in term of *severity* (see table 2.3) and *controllability* (see table 2.4), by comparing the numerical simulation results with the levels contained in the *classification tables*, respectively from tables 4.1 and 4.2.

¹¹As prescribed by the ISO 26262 part 3 (concept phase), during the HARA it is forbidden to consider failure effects mitigation system

It is unnecessary to develop a software layer able to put in communication MathWorks Simulink and IPG Automotive CarMaker since it provides suitable libraries out of the box.

All the scenarios chosen for the test of the considered AEBS case study can be represented by fig.4.5. Two cars are driving on a straight road in the same direction. The considered item is installed on the Vehicle Under Test (VUT), while a Target Vehicle (TV), preceding the VUT, acts as the obstacle. The differences between the situations are:

- the distance between the two cars at the start of the simulation;
- the speeds of the two vehicles, v_{vut} and v_{tv} at the start of the simulation;
- the TV braking acceleration function $a_{tv}(t)$.

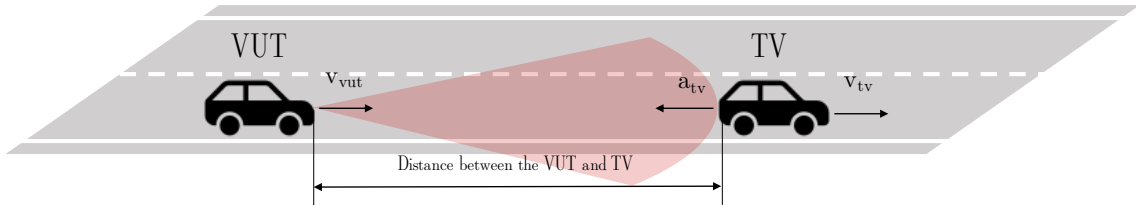


Figure 4.5: Representation of the situations analyzed in all the scenarios.

The severity assessment is based on the relative speed of the vehicles at the moment of the crash, while the controllability is classified by taking into account the time elapsed from the start of the hazardous condition, as described in the test, until the crash. This time interval is called in the following Time To Collision (TTC).

Since the item is represented by a semi-formal *behavioral* model, the fault-free model is a runnable model able to complain the requirements defined in [81], while the fault affected one lacks entirely of the braking capabilities.

The simulated driver does not brake in any condition, so only the AEBS can avoid the crash. Driver behavior, in terms of speed and trajectory, is defined in the test cases. In all the defined scenarios, the benchmark fault-free behavioral model of the item can avoid the crash.

4.4.1 EuroNCAP AEBS test protocol

The EuroNCAP test protocol [82] has been revised in 2017 and specifies two different test procedures:

- AEB City, considered in the assessment of the adult occupant protection;
- AEB Inter-Urban, considered in the assessment of safety assist.

There are three different scenarios:

- Car-to-Car Rear Stationary (CCRs)
The vehicle under test (VUT) is 120 m away from the target vehicle (TV). The TV is still. The simulation starts with the VUT in one case at 80 km/h, and in the other at 50 km/h.
- Car-to-Car Moving (CCRM)
The VUT and the TV are at speeds inside the range 50 - 80 km/h: the test starts at 50 km/h and, by increasing speed step of 5 km/h, it reaches 80 km/h. In this work, the simulation is performed at only 50 km/h.
- Car-to-Car Braking (CCRB)
The VUT and the TV have the same speed equal to 50 km/h. The test is performed with all the combinations of 2 and 6 m/s² decelerations of the TV. The relative distance plot over the time of the two combinations with the 6 m/s² deceleration of the TV are plot in fig.4.6 and fig.4.7. The behavior is similar for all the cases presented in this paper (except, of course, for the initial distance and the collision time).

This test protocol also explains how to check if the Forward Collision Warning (FCW) system works appropriately, but this functionality is not considered in this dissertation.

The obtained results are shown in table 4.3.

Test	Relative speed [km/h]	TTC [s]
CCRs (50 km/h)	42 (S3)	8.9 (C1)
CCRs (80 km/h)	74 (S3)	5.7 (C1)
CCRB (12 m, 2 m/s ²)	22 (S1)	3.8 (C2)
CCRB (12 m, 6 m/s ²)	41 (S3)	1.0 (C3)
CCRB (40 m, 2 m/s ²)	42 (S3)	7.0 (C1)
CCRB (40 m, 6 m/s ²)	44 (S3)	4.3 (C1)
CCRM (50 km/h)	21 (S1)	28.1 (C1)

Table 4.3: Results from the simulations inside the EuroNCAP scenarios. Table from [66].

Two interesting plots, obtained from the simulation results for the cases CCRB (12 m, 6 m/s²) and CCRB (40 m, 6 m/s²) are shown in figures 4.6 and 4.7.

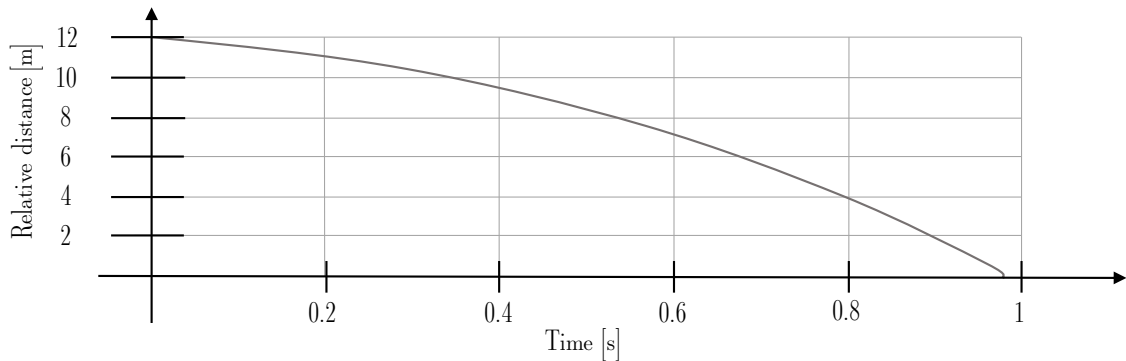


Figure 4.6: Plot of the relative distance between the VUT and the TV over the time for the case CCRb (12 m, 6 m/s^2). The intersection between the time axis and the relative distance curve represents the TTC. Figure from [66].

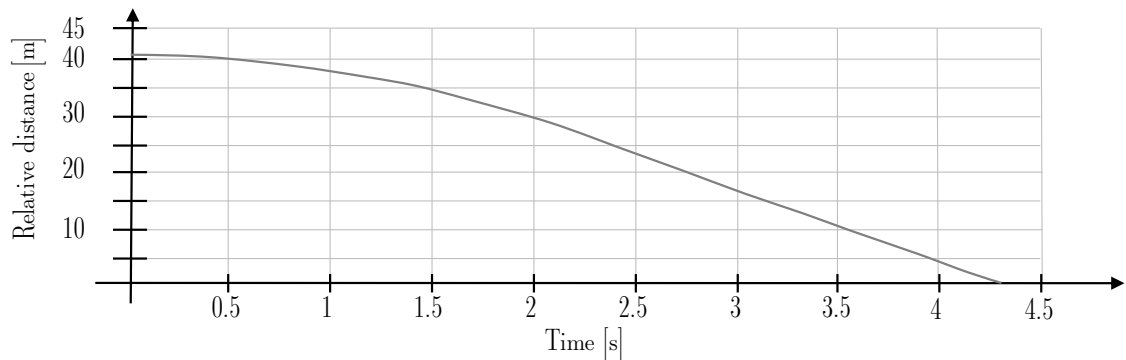


Figure 4.7: Plot of the relative distance between the VUT and the TV over the time for the case CCRb (40 m, 6 m/s^2). The intersection between the time axis and the relative distance curve represents the TTC. Figure from [66].

4.4.2 NHTSA tests

These tests [83] have been published in 1999 to test prototype ACC systems. Their primary purpose is to characterize the entire prototype system, composed of sensors, data fusion and control algorithms, and a vehicle platform.

These tests describe the following operational situations:

- Test 1 (headway control mode): closing in on a preceding vehicle from a long-range. VUT speed is 112.7 km/h, while TV speed is 96.5 km/h.
- Test 2 (aborted passing maneuver): responding to a close approach to a preceding vehicle. In this case, the speed of the VUT is initially 96.7 km/h. The speed of the TV remains 96.7 km/h for the whole test. When the VUT and TV gap is 37.5 m, the driver accelerates to 112.7 km/h. At 2/3 of the

original gap, the driver releases the throttle pedal. The test continues until the steady-state is reestablished.

Test	Relative speed [km/h]	TTC [s]
US Test 1	25.0 (S1)	33.5 (C1)
US Test 2	3.6 (S0)	10.9 (C1)

Table 4.4: Results from the simulations inside the NHTSA scenarios. Table from [66].

4.4.3 European Commission Regulation 347/2012 tests

These three tests come from the European Commission Regulation 347/2012 [84]. It describes mandatory homologation tests for the AEBS in the European Union.

Three tests are interesting for our purposes:

- EU Test 1 The VUT travels against a still target, representing another car, at 80 ± 2 km/h;
- EU Test 2: As the test 1, but the TV moves at 32 ± 2 km/h;
- EU Test 3: As the test 1, but the TV moves at 12 ± 2 km/h.

All the tests start when the distance between VUT and TV is at least 120 m. In this work, the initial space is 120 m for all the simulations.

Test	Relative speed [km/h]	TTC [s]
EU Test 1	80.0 (S3)	5.6 (C1)
EU Test 2	45.0 (S3)	9.3 (C1)
EU Test 3	68.0 (S3)	6.6 (C1)

Table 4.5: Results from the simulations inside the NHTSA scenarios. Table from [66].

4.4.4 ASIL Assignment

After the simulation data have been obtained, it is possible to summarize all the results in table 4.6.

From the ASIL assignment, it is possible to observe that most of the cases obtained an ASIL B.

The only case with a different classification is *EuroNCAP CCRb with 12 m of gab between the VUT and TV, with the TV braking at 6 m/s²*, indicated as *CCRb (12 m, 6 m/s²)* in the table 4.3.

This case can be considered a predictable misuse of the system, as the driver forces the vehicle, with a deliberate action on the throttle pedal, not to respect the safety distance.

A typical AEBS system is designed only to assist the driver, and it is not in charge of the safety of the driving task (see section 1.2.2), so it cannot override the driver's decisions.

To avoid this kind of misuse, it is possible to implement some strategies on the Human Machine Interface (HMI), like a sound warning, triggered when the driver's action prevents the system from respecting the safety distance, from making he/she to desists from this dangerous behavior.

Due to these premises, it appears reasonable to assign the ASIL B to the safety goal *the ACC/AEBS brakes when there are obstacles in front of the vehicle*.

Test	S	C	E	ASIL
CCRs (50 km/h)	3	1	4	B
CCRs (80 km/h)	3	1	4	B
CCRb (12 m, 2 m/s ²)	1	2	4	A
CCRb (12 m, 6 m/s ²)	3	3	4	D
CCRb (40 m, 2 m/s ²)	3	1	4	B
CCRb (40 m, 6 m/s ²)	3	1	4	B
CCRM (50 km/h)	1	1	4	QM
US Test 1	1	1	4	QM
US Test 2	0	1	4	QM
US Test 1	3	1	4	B
EU Test 2	3	1	4	B
EU Test 3	3	1	4	B

Table 4.6: ASIL classification of the various tests. Table from [66].

As said in the approach description, the assessment of the exposure risk parameter is performed manually.

To reduce the exposure level from E4, statistical evidence demonstrating that the considered situation is uncommon has to be provided. For this benchmark application, since all these scenarios have a high probability level, the most reasonable choice is to assign them an E4 level, so no statistical evidence has to be provided.

Chapter 5

Real-time software validation

All the methodologies on the real-time software validation presented in this chapter are based on the Hardware-In-the-Loop (HIL).

HIL is a software integration verification technique. It is performed by connecting a real-time simulation system to the controller (target), (running a physical model) through transceivers, to obtain identical signals, from the electrical point of view to the one of the real world. HIL is explained in section 2.11.5.

In this way, it is possible to test the embedded software on its target device, keeping into account the timings and the interaction with the needed device drivers.

The results shown in this chapter have already been published into four papers. They can be classified based on their topic:

- Tests on Automotive Body Control Modules (BCM) [8]. In this case, in collaboration with a company, a new software tool has been developed to simplify the HIL for items characterized by CAN network access and a high number of low-speed I/Os.
- Tests on real-time mixed-criticality applications with instrumented bug in the lowest DAL [9] (the *safety level* of the DO-178B/C) application or with the bug injected by a debugger [10] (in this case pausing and resuming the real-time simulation to allow access to the controller without interfering with the real-time simulation).

This proposal describes a HIL approach to the validation of mixed-criticality systems (MCS). In this approach, the HIL testing is combined with fault injection to prove safety of the nominal application and of the fault detection, isolation, and recovery (FDIR) mechanisms implemented either in software or hardware.

- Tests on the multi-agent robotic system (MAS). [11] The particularity of these robots is that their end effectors are moved by independent agents that cannot communicate with a link reliable from the timing point of view. Hence, they need to maintain synchronization relying only on their internal clock sources.

5.0.1 Research contribution

The novelties of these contributions are not contained in the HIL methodology by itself but on how is applied.

Multi-agent robotic systems (MAS) may seem not related to the classic paradigms of the automotive industry. However, the challenges on *connected vehicle* (one of the three challenges already discussed alongside *electrification* and *autonomous driving*) make it a field of interest. In automotive applications, other agents can be the other cars or pedestrians carrying a cell phone (vehicle to vehicle, V2V) or the infrastructure, like traffic lights (vehicle to infrastructure communication, V2I).

Different levels of cooperation are possible: while the communication with the pedestrians' cell phones can be used only to aid the obstacle detection algorithm of the autonomous driving, improving safety by allowing detection of pedestrians even outside the long-range radar limitations¹, communication with other vehicles can help optimize transit inside road intersections or, for example, to give the right to pass to an emergency vehicle.

Due to the difficulties of obtaining access to an industrial application of this kind and taking into account the research field of my group (functional safety and development and validation of real-time systems), it was therefore chosen to test the HIL application for the software validation of a hard-real time robotic application. This application could be considered similar from the point of view of functional requirements to the coordination of vehicles within an intersection, making sure that they can cross it at low speed without the need to stop and at the same time guaranteeing a sufficient safety level from the risk of collisions.

5.1 Automotive Body Control Modules

The approach proposed in this paper was presented in 2018 at the IEEE 13th International Conference on Design & Technology of Integrated Systems in Nanoscale Era [8]. It is a technology transfer project, implemented with the company TXT E-Solution S.p.A.

My contribution to this proposal started from the TXT XHIL Studio [85]. When the project started, this product has been already released, so it was yet commercial software. The agreement between Politecnico di Torino and the company allowed the author to access and modify its source code.

The goals of this project are:

- the inclusion of CAN communication both at the protocol and HMI level;

¹A long-range (76 GHz) radar can detect obstacles up to 150 m [86].

- the porting from the proprietary TXT XHIL hardware to the NI PXI platform.

The latter activity has been done in collaboration with the Masters' Degree candidate Alessandra Mugoni, who described the porting actions in her thesis [87].

Modern vehicles are managed by Electronic Control Units (ECUs), which today execute software composed of millions of statements, responsible for a broad spectrum of functionalities, ranging from vehicle occupant comfort (e.g., HVAC) to safety (e.g., ABS, ESP, ADAS). To guarantee an adequate level of quality for the automotive embedded software is has been proposed the ISO 26262 standard. It defines a strict development and validation process (see section 2.11) that must be fulfilled in order to produce software according to state-of-the-art automotive practices.

Among the different steps, the ISO 26262 Standard foresees, software test is one of the most well established, requesting several activities aiming to discover potential deviations from the expected functionalities before the items are shipped to the car manufacturers.

In terms of input/output requirements and the number of implemented functionalities, one of the most complex ECU is the Body Control Module (BCM). It serves multiple purposes: interacts with the user by reading some discrete inputs (e.g., the turn indicators and the ignition key), actuate some discrete outputs (e.g., the window defroster, windscreen wipers, etc.), and communicate over several networks the vehicle accommodate (e.g., CAN, LIN, and FlexRay). A typical BCM ECU has about 100 I/Os, two or three CAN networks, and its software provides more than 100 distinct functions. The usual approach followed by Tier-1 while testing ECUs such as BCM is outlined in [88]. Following the procurement of a HIL hardware providing the adequate set of I/O to connect with the ECU hardware, the test environment is firstly prepared, configuring the companion software of the chosen hardware platform to interact with the ECU to be adequately tested, called as *Device Under Test (DUT)*, or just as *BCM* to highlight the purposes of this proposal, in the following. After that, the test cases are developed according to the item under test specifications.

Two different professional roles are involved in these activities [89]:

- the tool engineer, who is expert in the configuration of the HIL hardware and its companion software;
- the test operator, responsible for implementing the test cases to be performed on the DUT and analyzing the collected output responses.

Today it is possible to recognize two types of HIL solutions for BCM testing:

- *Low-end HIL solutions*, also referred to as *static simulators*, where the inputs to the BCM are provided through a set of switches, knobs, and network simulators that are wired to the BCM via break-out-boxes, while outputs from the BCM are connected to bulbs and dial indicators. Inputs are manually operated by the test operator who follows the defined test cases to apply the required commands (stimuli) to the DUT. Similarly, output responses of the BCM are collected and evaluated manually by the test operator. This solution has two significant limitations. On the one hand, it lacks determinism: since it is based on manual operations, it is nearly impossible to repeat the same test case. On the other hand, the operators cannot automate the tests. Nevertheless, this approach is today still in use with some Tier-1 companies.
- *High-end HIL solutions*, where dedicated reprogrammable hardware is used to automatically interface with the BCM under test, apply test cases, and automatically collect output responses. This approach benefits from the automation capabilities that make the test activity deterministic. However, due to the HIL hardware's intrinsic complexity, the test requires a tool engineer and a test operator. Moreover, the ordinarily available tools to configure and operate HIL systems are mostly intended for real-time validation of control algorithms, rather than testing software based on Finite State Machines (FMS) like the BCM one. As such, they provide a rich set of features that are not specifically designed to ease the work of test case developers.

5.1.1 Proposed approach

The main driver for the development of the proposed approach is to devise a software test instrument, defined in the following as *XHIL Studio* or the *tool*, that is closer to the skill sets and know-how of the software test case developers and test operators of BCMs, which are quite different from that generally required for operating the HIL hardware and its companion software.

As far as the HIL is concerned, several suppliers are on the market, like dSpace, National Instruments, and ETAS. These companies have proprietary reconfigurable hardware platforms and software environments, mainly intended for rapid control prototyping and HIL testing, where the main focus is the real-time validation of newly developed embedded software. As such, the capabilities of the hardware machines are comparable to those of high-end servers, as they shall enable the real-time execution of complex plant models (e.g., a detailed model of a vehicle dynamic or an engine), while the accompanying software environments are mostly intended for supporting the development of validation test cases, and activity that is often performed by the same developers of the control software under test.

In developing the methodology here proposed, it has been adopted a radically different approach:

- the HIL hardware and its companion software are used as a reconfigurable interface to connect with the BCM (or a simplified ECU running only a subset of the software components) hosting the software under test. In this context, the simulation hardware takes care of the electrical interfaces with the BCM and the network connectivity; it is not intended for running complex plant models, although still possible for future extensions. The HIL hardware is used to run, in real-time, a special model, defined in the following as the *BCM model*, that defines an abstraction of the BCM hardware, exposing to the companion software the BCM I/Os as a set of logical channels;
- the proposed HIL software operating the simulation hardware offers a simplified interface to establish the correspondence between the logical I/O channels and a set of test cases developer-friendly objects (e.g., buttons, dials, graphs). Through these objects, the test case developer can define a test instrument panel to operate on the BCM I/Os; moreover, he/she can perform a sequence of operations on the objects (e.g., setting a switch to the desired position) to build test cases that are recorded, and that can be replicated. In the case the TXT proprietary hardware platform is used, XHIL Studio operates it (it acts as both the test and the companion software), while in the cases third part hardware is used, XHIL Studio works through the APIs exposed by the companion software.

Thanks to this approach, it is possible to get two main benefits:

- high flexibility, thanks to the adoption of a reconfigurable HIL system, which makes it easier to interface it with different BCMs;
- significantly improved usability for the test operator, which is not required to be a high-end HIL expert, since he/she can operate, thanks to the proposed tool, as accustomed in the already adopted low-end solutions.

The proposed approach is based on an approach involving a test operator and seven layers:

- *Test operator* is the person who has to perform the tests.
- *Human machine interface (HMI)* of the tool. It allows the test operator to prepare test cases, run tests, and collect output responses with minimum effort.
- *Hardware abstraction layer (HAL)*, it is placed in between the HMI and the *companion software*. It translates the HMI actions into the corresponding actions to be performed by the HIL hardware and delivers them to the hardware driver.

- *Companion software* is a collection of software running on a host PC connected to the HIL hardware to handle the communications with the HIL hardware.
- *BCM model* it is the model that represents the I/O structure of the ECU. It is composed of two different elements. The first one is a table (implemented as a Microsoft Excel file) containing the I/O of the HIL hardware, the relative voltage levels, and the corresponding identification labels. The second one is a model of the BCM. As shown in fig.5.1, the left side of the model is composed of inports and outports, which are named using the same labels reported in the HIL I/O table. Instead, its right side connects the model inports/outports with the test harness. In the current implementation, the BCM model is implemented resorting to Mathworks Simulink. This choice makes it compatible with any HIL software environment; moreover, it makes it possible to support tests at different development process stages. Indeed, the test harness can be a HIL hardware or only a software simulation. In this way, safety engineers can use the tool to implement an X-in-the-loop (XIL) testing setup [90].
- *HIL hardware* is responsible for running the vehicle model (in the case of a BCM regarding the lightning and other supporting feature and not its dynamics), to provide test stimuli as instructed by the test operator through the HMI, and to provide test responses to the operator by the HMI.
- *Breakout box* is a physical apparatus used to wire the HIL to the BCM hardware.
- *Device Under Test (DUT)* is the ECU running the software under test.

5.1.2 Benchmark proof-of-concept setup

This section describes a proof-of-concept implementation of the proposed methodology.

The test operator interacts with the HMI implemented, resorting to TXT XHIL Studio, specifically designed to perform test case development for the software integration test of BCMS. It offers a user-friendly interface specifically devised for test case preparation and execution.

XHIL Studio offers an editor that allows the tool operator to prepare test cases. The editor, shown in fig.5.2, is composed of:

- a list of simulation panels, useful to manage a set of test cases, one for each of them;
- a toolbox panel, which contains the objects, called *widgets*, which can be used to build test cases;

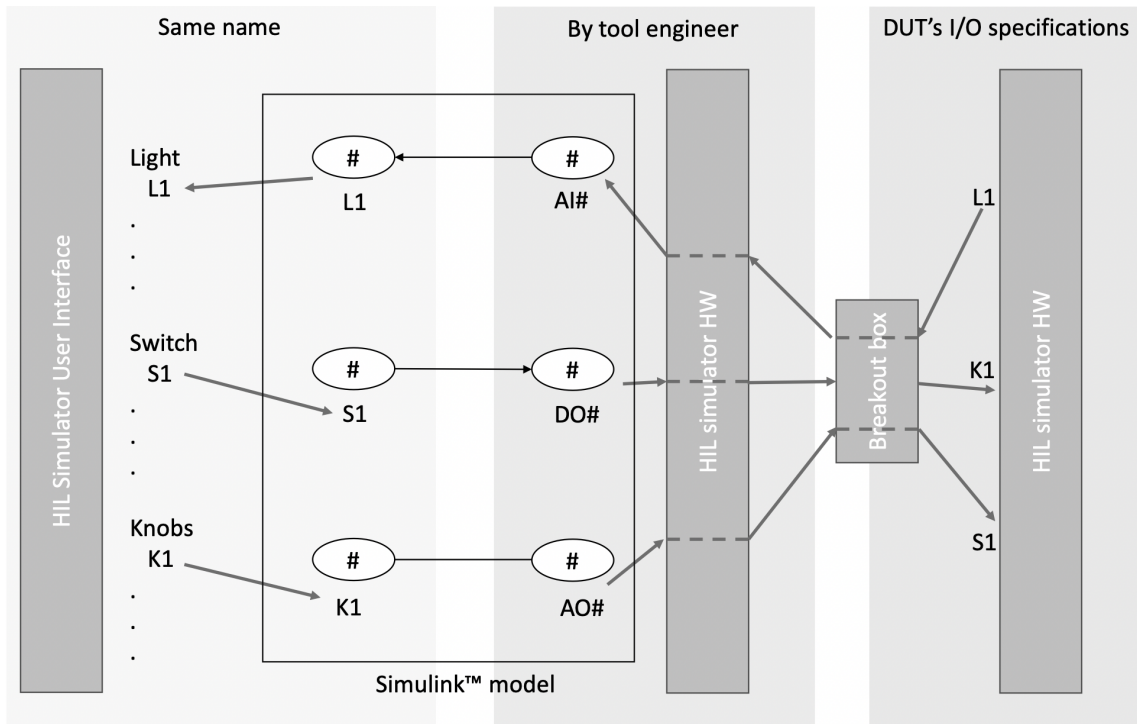


Figure 5.1: ECU Simulink™ model, with description of mapping from the HMI to logical resources of the reconfigurable hardware. On the top of the three grayed areas, it is reported the source of the imports/outputs labels.

- a simulation panel editor, populated by the tool operator populated with widgets, including both discrete I/O signals as well as network messages;
- a list of ECU inputs and outputs, their properties, and a list of the network (e.g., CAN) messages the DUT handles.

After the test case developer prepared the needed simulation panel, it can be operated through a simulation panel player. An example of the player is shown in fig.5.3.

Through the player, the tool operator can define which stimuli have to be applied to the DUT (by acting on the widgets) and its expected correct responses (by taking snapshots of situations in where all requirements are met). The sequences of performed operations can be recorded for later playbacks, allowing them to repeat the tests automatically. The recorded test sequence is deployed to the test harness, and its responses are collected and compared with the expected ones (obtained from the snapshots).

Finally, a test report is generated highlighting any deviations between observed and expected outputs.

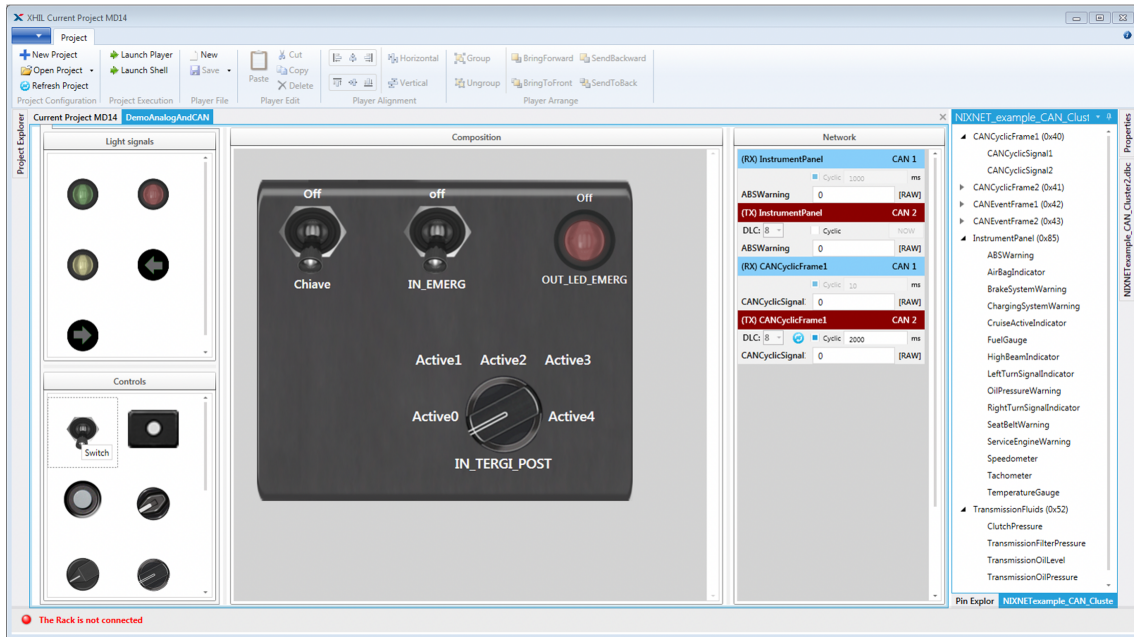


Figure 5.2: The TXT XHIL Studio simulation editor.

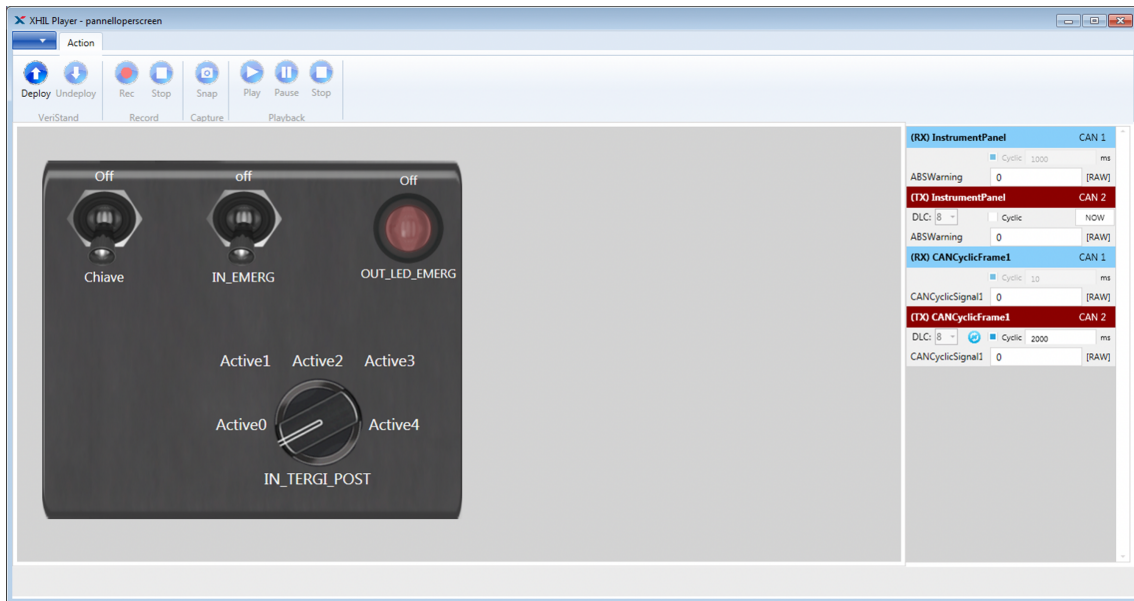


Figure 5.3: The TXT XHIL Studio simulation player.

The HMI interacts with the test harness through the HAL that, in this implementation, manages the communication with the test harness of choice, composed of National Instruments hardware and its companion software, National Instruments VeriStand. Communication between HMI and VeriStand is made possible

through the VeriStand Application Program Interface (API) [91], which provides two services: the Execution API, which controls the operations of the HIL hardware, and the Real-Time Sequence Definition API, which automates the process of the real-time stimuli generation. The *hardware driver* is implemented through the VeriStand Gateway, which manages the communications from the host computer to the HIL hardware through a LAN connection. The interaction between TXT XHIL Studio and NI VeriStand API is shown in fig.5.4.

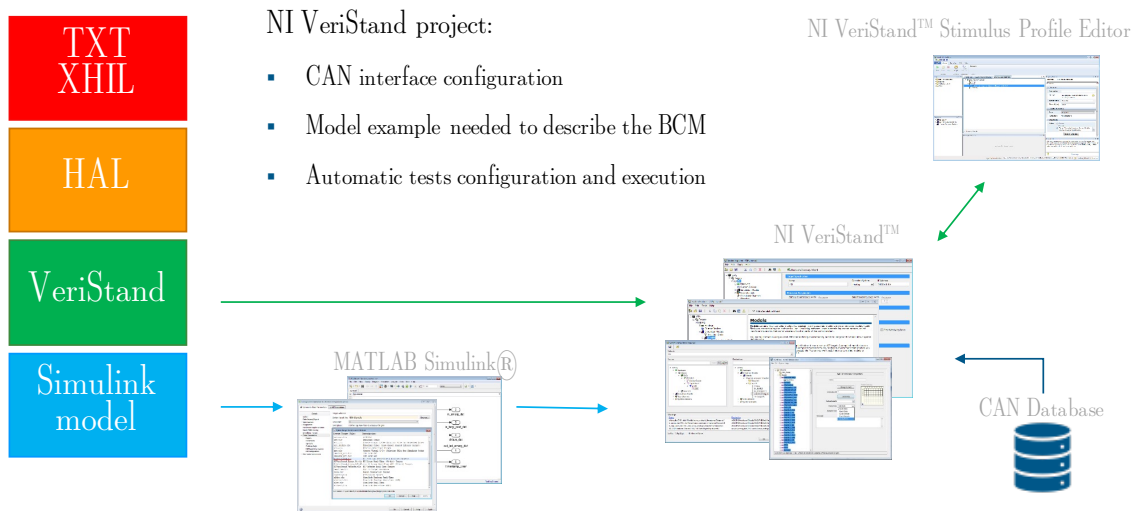


Figure 5.4: Interaction between TXT XHIL Studio and NI VeriStand API.

As the benchmark DUT, it has been considered a commercial BCM provided by an Italian Tier-1 supplier, which entails 72 analog inputs, 80 analog outputs, 4 digital outputs, and 2 can network adapters with their relative messages databases. It implements more than 120 functionalities, and it accounts for more 1200 C functions and about 300000 C lines of code.

Due to intellectual protection issues, for public presentations, a simplified ECU has been obtained by implementing a simplified turn indicators control software on an NXP S32K144 reference board, embedding also CAN connectivity. A photograph of this setup, presented to customers in November 2017 at National Instruments Days held in Milan, Italy, is shown in fig.5.5.

The chose HIL hardware is a National Instruments PXI machine, which runs the NI VeriStand Engine for real-time execution of the ECU model. The PXI has been adopted since it features a powerful Intel Core i7 multicore CPU and allows, thanks to its modular architecture based on the PXI bus, to be easily configured to match the I/O connectivity requirements of the ECU under test. The wiring to the BCM is implemented, resorting to a breakout box for discrete analog and digital signals and the NI XNET cabling for the CAN network.

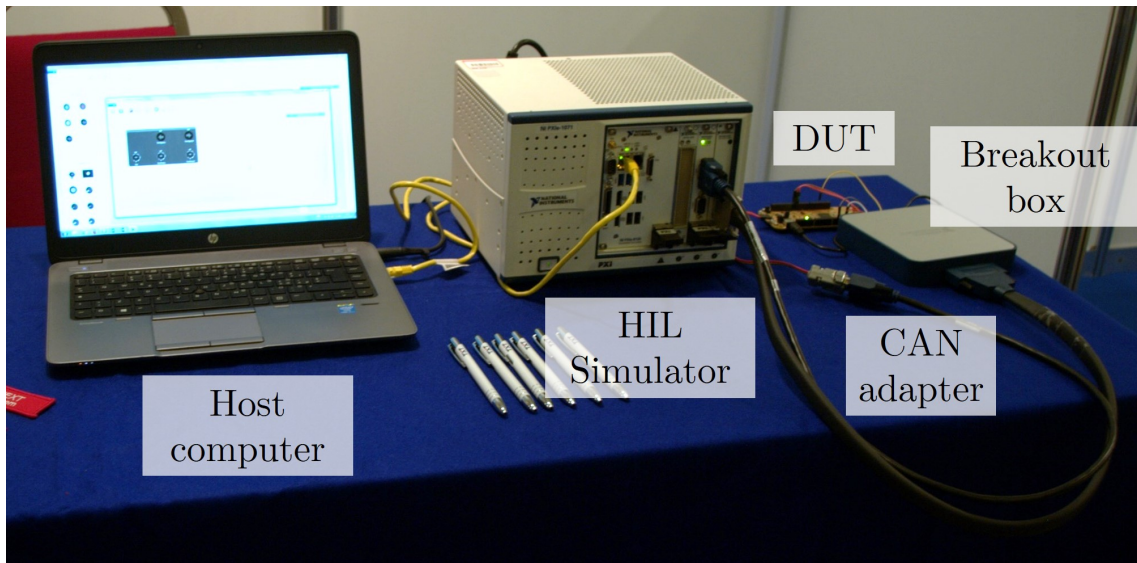


Figure 5.5: A benchmark of the proposed approach, performed on simplified ECU implementing a turn indicators control software on an NXP S32K144 reference board, shown in November 2017 at National Instruments Days in Milan, Italy.

5.2 Mixed-criticality systems

This proposal has been published into two papers: the first one have been presented at the 19th IEEE Latin-American Test Symposium, while the second at the conference 24th IEEE International Symposium on On-Line Testing and Robust System Design. Both the conferences were held in 2018, and the papers were released into their proceedings.

Modern aircraft are managed by Electronic Control Units (ECU), tasked with the execution of several processes. A particular challenge is involved in the design of Unmanned Aerial Vehicles (UAVs): due to safety reasons for the people surrounding the flight zone and the high costs involved in building a UAV, producers have to guarantee the reliability of their avionics ECUs.

To guarantee that the software running on these ECUs shall fulfill hard real-time performance requirements, there have been proposed many techniques to compute the worst-case execution time (WCET) of each process. However, not all software components in an aircraft have to be considered critical to the safety or success of the mission in an equal way. This criticality difference is described, in the standard DO-178B (or C), by defining different Design Assurance Levels (DALs). This concept is similar to the ASIL of the ISO26262, which determination has been described in section 2.2.2.

In the past, certification agencies required that applications running on the same

hardware should all be designed at the highest DAL mandated by their functions. To avoid the cost of developing applications at the maximum DAL (A), avionics OEMs resorted to distribute these application components over several onboard types of equipment, in what was called the federated architecture or, more recently, into the integrated modular avionic (IMA) [92] [93] architecture.

To reduce the size, weight, and power (SWaP) consumption of such architectures, the implementation of mixed-criticality applications is a viable solution. Unfortunately, while developing mixed-criticality application, it is necessary to deal with tasks developed following different DALs requirements that share the same elaboration unit. Therefore, special provisions should be made to avoid catastrophic consequences in the case applications developed at a low DAL interfere with higher DAL ones.

Several solutions have been proposed in the literature for implementing a mixed-criticality system (MCS), some focusing on the system architecture [94] [95], other focusing on the key aspect of the system scheduling [96] [97]. However, independently from the chosen implementation method, developers should validate their solution to prove its safety. This is especially important for MCS: software components developed at the lower DAL might carry defects² which could interfere, when excited, with the execution of the highest DAL ones.

In the literature, there are several proposals about this issue. To classify such solutions, it can be useful to subdivide the interference problems into two classes [95]:

- *temporal interference*: is the ability of a software component to change the execution time of another ones, possibly resulting in deadline misses of real-time safety-critical tasks;
- *spatial interference*: is the ability of a software component to modify resources intended as data provided by either the memory or some peripherals, used by different ones, causing them to misbehave.

Several approaches have been proposed to avoid interferences [98]. These are called as *partitioning* methods. Partitioning is defined as *appropriate hardware and software mechanisms to restore strong fault containment* in [99]. The main trend in literature is to apply system scheduling theory and practices to solve the issue [100] [101] [102].

To make the problem even more complex, a formal validation approach could be unfeasible due to the impossibility of proving the absence of bugs in a software component developed not following the prescriptions of higher DALs. Therefore,

²Systematic failures, or *bugs*.

functional testing is used. It allows to choose a subset of the most probable scenarios and to use them to test partitioning on the target platform. Due to the nature of avionic applications and to the high dependency of the WCET on the target hardware architecture, reliable results can be obtained only when the software is running in a real-time manner on its target platform.

The main technique to perform such integration tests is the hardware in the loop (HIL), already described in section 2.11.5. The proposed approach is implemented on a benchmark MCS. Experiments are performed to prove that the HIL can either detect a failure (in the considered case a deadline-miss of a safety-critical task) or prove fault tolerance³, demonstrating that the chosen partitioning systems prevent the lower DAL tasks [9] or a transient random hardware failure [10] to interfere with the critical higher-DAL ones, by analyzing their effects on the generated UAV control commands.

5.2.1 Proposed approach

The proposed approach requires a benchmark system capable to:

- simulate the physical system in real-time;
- generate stimuli with the same electrical characteristics as those provided by the physical system to the DUT;
- ensure the test repeatability by logging both the stimuli generated by the physical system model and corresponding DUT outputs;
- verify that the control software is correctly implemented in the DUT by checking that it satisfies its functional requirements;
- trigger a known unsolvable defect affecting a low-DAL task [9] or perform fault-injection into the DUT [10];
- suspend the physical system simulation execution during fault injection operations, and resume it safely after such operations are completed (only for the proposal described in [10]);
- produce a human-readable test report, by processing the data collected during the test campaign.

To implement all these requirements, the following components are needed:

³Fault tolerance in this context means that even in the presence of a fault the safety-critical software component performs its task in the expected time frame.

- a *controller* implemented on the DUT, representing the safety-critical module in charge to control the physical system;
- one or more *non-critical tasks* to represent the faulty non-critical software components, running on the DUT, that could interfere with the controller’s task;
- a *fixed-step*⁴ model of the physical system, controlled by the safety-critical controller, to be run on the real-time simulator;
- a *real-time computer* to run the physical model simulation;
- a set of *input/output conditioning stages*, to produce stimuli to the DUT (output), and to obtain the responses from the DUT (input), equivalent from the electrical point of view with the ones of the physical plant;
- a *software infrastructure to manage the simulation environment* on both the workstation and the real-time computer to manage the physical system simulation, log the stimuli and the responses, and produce the test reports;
- an *external debugger (eDB)* (only for [10]);
- a *fault injection system* composed of a *fault-list generator* and a *saboteur* (only for [10]);
- a reliable synchronization method between the *real-time computer* and the *eDB* (only for [10]).

The physical model has been implemented on a high-end commercial real-time computer designed for both HIL and rapid control prototyping, with a huge availability of modular conditioning stages. The considered physical system is the longitudinal model (pitch angle behavior) of an UAV airplane. To implement an efficient test environment, the plane model should consider the trade-off between precision and speed of the simulation.

A proper implementation of the HIL system provides means to perform functional verification of the DUT, besides the validation of *partitioning* [9] and Fault Detection, Isolation and Recovery FDIR [10] mechanisms and other non-functional requirements. An issue, in this context, is the verification of the temporal behavior of the DUT: test engineers should verify the real-time requirements of the whole DUT by evaluating the time-domain performance characteristics of the system (see fig.5.7). Therefore, a verification is needed to prove that, in fault-free conditions,

⁴While in MIL 2.11.4 and SIL (see section 2.11.4) also variable-step models can be adopted, HIL is possible only with fixed-step ones, since it requires to run the model in real-time manner, hence at a constant rate and within a certain execution time range.

both the critical and the non-critical subsystems are achieving their specifications. The proper way to perform such verification depends on the application. However, the architecture proposed in fig.5.6 is general enough to be adapted for any real-time application.

The benchmark architecture, able to grant the properties of temporal and spatial partitioning and implementing the FDIR mechanisms, has been obtained from the literature [94] [95].

In [94] and [95] partitioning has been subdivided into two types, each related to one of the two kinds of interference described above: *spatial partitioning*, for issues relate to *spatial interferences* [94]; *temporal partitioning*, for issues related to *temporal interferences* [95].

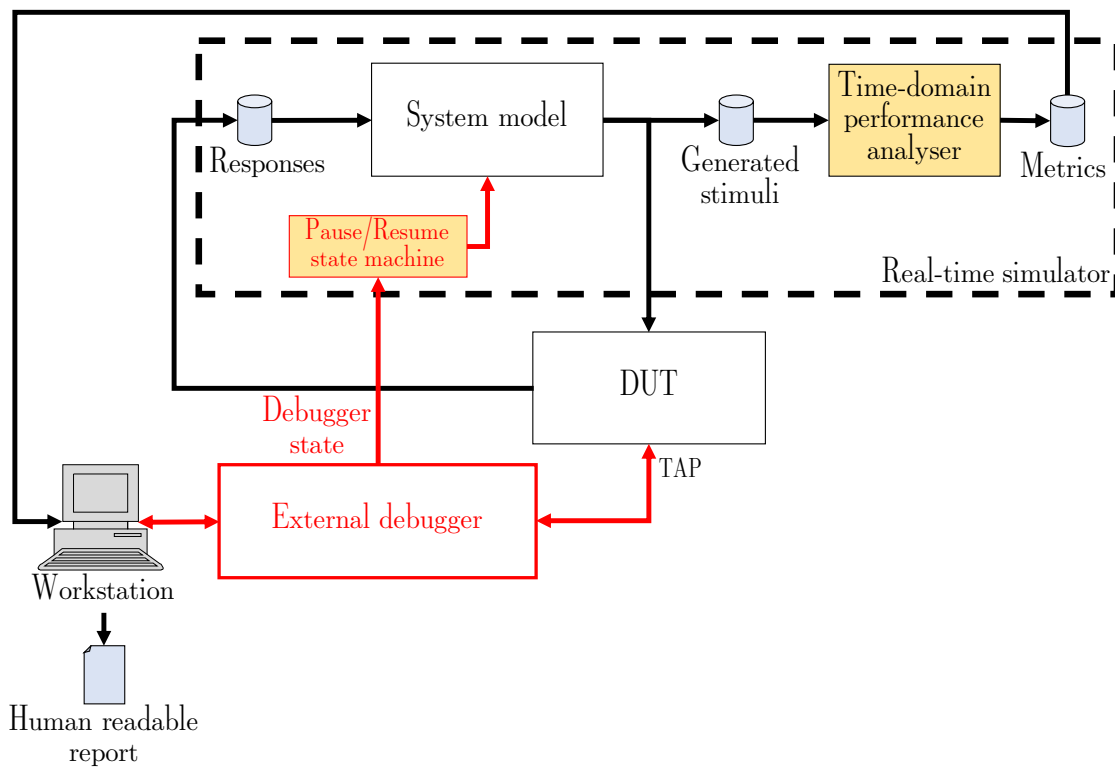


Figure 5.6: The block diagram of the approach presented in the [9] and [10] papers. The blocks regarding the external debugger and the path to perform the fault injection, present only in the approach presented in [10], are bordered in red. In [9] there is only a signal to trigger the known non-solvable defect affecting a low-DAL task, managed by the HIL simulator.

During the tests, *time domain performance analyser*, represented in fig.5.6, is used to characterize the time-domain physical system the step response in different situations. In the first paper [19], the performances are evaluated with the low-DAL

task not affected or affected by an unsolvable defect, while in the second one [10], these are compared, by the workstation, with the results of the fault classification, to obtain a final report which would indicate what faults, if any, caused a significant difference in the step response of the system.

The time-domain responses are characterized according to the following metrics (see fig.5.7 for their graphical representation):

- *rise time*, it is defined as the time between application of the step input and the reaching of the desired system response;
- *overshoot*, is the ratio between the desired value of the system output and the maximum of the measured response;
- *settling time*, is the time required by the system to reach a stable condition in which the response stays within a 5% error from the desired value.

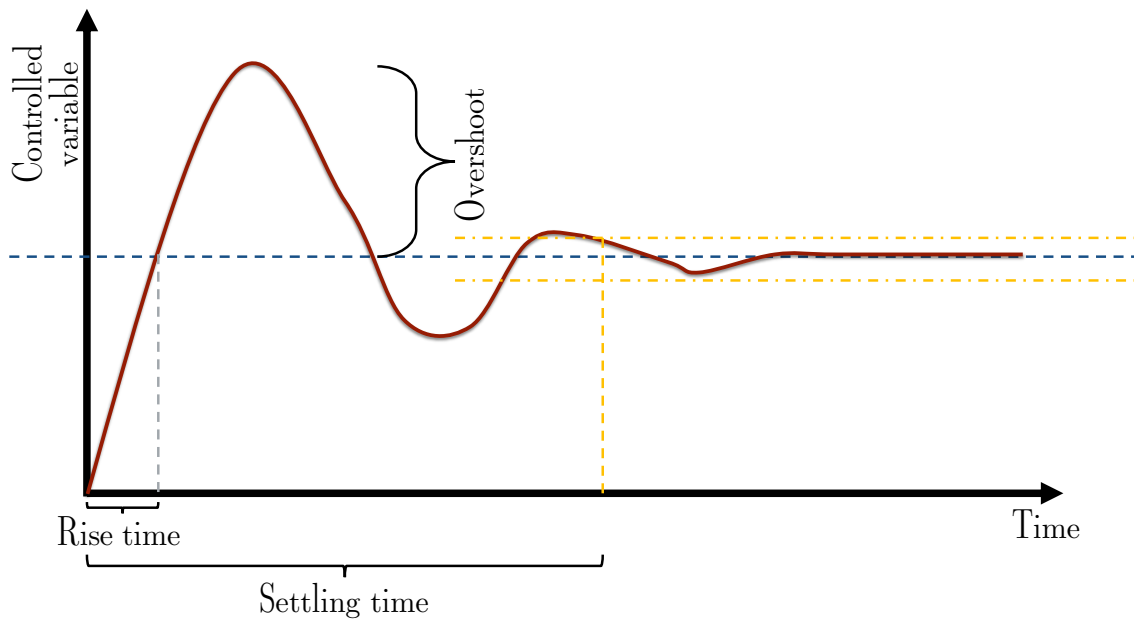


Figure 5.7: The time domain performances characteristic of interest, for a 2nd order system, from the control theory literature.

From a practical point of view, test engineers should prepare the HIL test according to the following to-do list:

1. *Acquire physical system model and a suitable controller.* The physical system (in this case, an airplane UAV) is a model of the longitudinal (pitch) behavior of a plane. Models can be described either in a model-based language, such

as Mathworks Simulink or by a programming language, such as C. Model-based languages should be preferred, as they simplify the implementation of the following steps. In this case the scope is to test the effectiveness of the partitioning and FDIR mechanisms, and not the control software, as it is usually did with the HIL. This model shall adopt a fixed-step solver to allow it to be run in a real-time manner.

2. *Define the interfaces between the real-time simulator and the DUT.* Such interfaces depend on the application specifications. It shall be used, for each signal, the same interface that the DUT uses in the real UAV. It allows to consider the device driver contributions on real-time performances, and their timing requirements (i.e., signals to be acquired in parallel at the exact same time). From the practical point of view, this is a crucial aspect since it allows to choose the needed analog conditioning modules to be installed into the real-time computer.
3. *Prepare and test the physical connections.* In this phase, the wiring harness between the real-time simulator analog conditioning modules, and the DUT are put in place and properly tested, for example by transmitting known periodic patterns.
4. *Integration test.* This phase is the core of the HIL. It is needed to test the communication between the DUT and the real-time computer and to ensure that the safety-critical module has been properly implemented on the DUT. To ensure valid results, the DUT shall run its nominal mixed-criticality workload. During this phase are collected the response of the DUT to the applied stimuli.
5. *Time-domain performance analysis.* In this phase, that can be performed online by the HIL simulator or offline by the workstation (for the sake of this work it has been adopted the online implementation) the real-time behavior of the DUT, while running its nominal workload, is validated. In the scope of this paper, the plant is a system of the second order. Therefore the time-domain profiling is performed by checking its step response.

Once the system has been properly integrated and verified, it is possible to also integrate, as did in [10], a FI system. In this way, it is possible to assess, other than the partitioning between critical and non-critical tasks, also the FDIR systems capabilities to react to transient random hardware failures that could affect both the memory or elaboration unit.

The FI system presented in [10] has been developed according to the model described in [104], with the following mapping:

- the fault injection manager, the fault list generator, and the fault classification module are implemented on a host workstation, connected through a proper interface to an external debugger (eDB);
- the saboteur and data collector are implemented by an eDB, with its management software running on the workstation;
- the workload generator is implemented on the real-time simulator. Also its management software runs on the workstation.

The eDB is connected, through a test access port (TAP), to the DUT. A second interface is designed to connect the DUT to the HIL platform and provide a synchronization signal between the two subsystems. This synchronization allows the HIL platform to pause the model simulation during FI, thus freezing the whole test status during this operation. This synchronization is fundamental to perform a meaningful FI experiment in the context outlined in this paper: if the physical model is not frozen during FI operations, the delay so introduced could be enough to affect the control system real-time performances, making the tests useless for system analysis. This synchronization interface has been implemented on the benchmark system through an active-low signal managed directly by the eDB. Such signal is forced low at the beginning of the fault injection and is released at the end of such operations; therefore, the model simulation is stopped for as long as the DUT processor is stopped during fault injection and granting synchronization.

After implementing and testing all said interfaces, the test environment can be used to apply any suitable test. The main objective of the two proposed test environments is to validate FDIR mechanisms against defects of low-DAL tasks [9] or transient random hardware failures [10] affecting the elaboration unit. To do so, tests should be devised to excite such mechanisms. The fault list generator is able to generate a fault list of a given cardinality and needs as input a list of all possible fault injection locations in terms of address and length for memory-mapped configuration registers or in terms of register name and length for CPU internal registers.

The fault injection manager collects faults from the generated fault list and, one at a time, sends the details to the fault injection module, which performs the following operations:

1. stops the execution of the embedded software on the DUT;
2. forces to low the synchronization signal to the HIL simulator;
3. injects the fault;
4. releases the synchronization signal;
5. releases the DUT, restoring the embedded software execution.

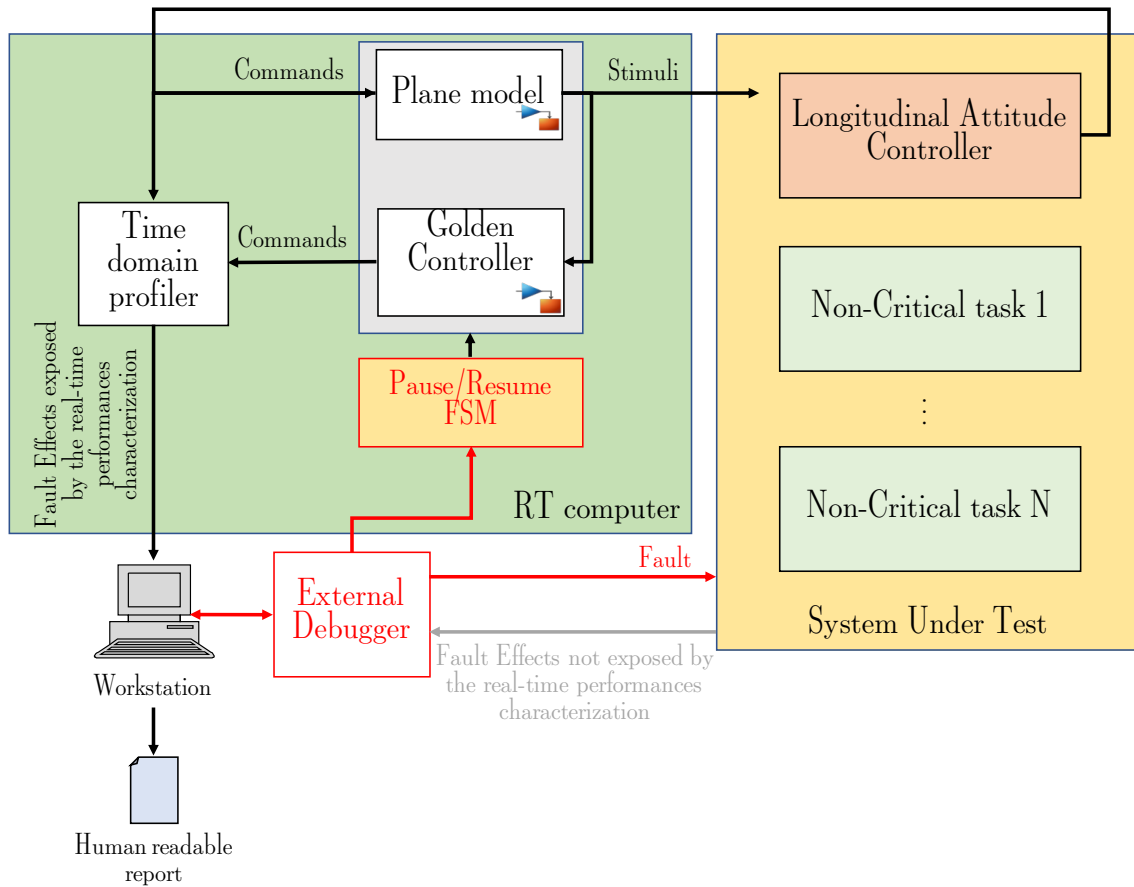


Figure 5.8: The block diagram of the proposed HIL system.

The *pause/resume finite state machine (FSM)*, representing the possible states of the proposed HIL system, is shown in fig.5.9.

Once the execution of the workload has been completed – which in the scope of this paper corresponds to a fixed flight pattern describing a steep pitch angle change request from the simulated pilot – the data collector module reads the output generated by the DUT and forwards them to the fault classification module, along with a set of flags used to determine if any FDIR mechanism has been activated in response to the fault.

The fault classification module separates faults in the following classes:

- Silent or no effect (**NE**): the fault had no effect on the system;
- Silent data corruption or Failure (**F**): the fault was not detected by any FDIR mechanism but caused a wrong output being produced by the DUT;

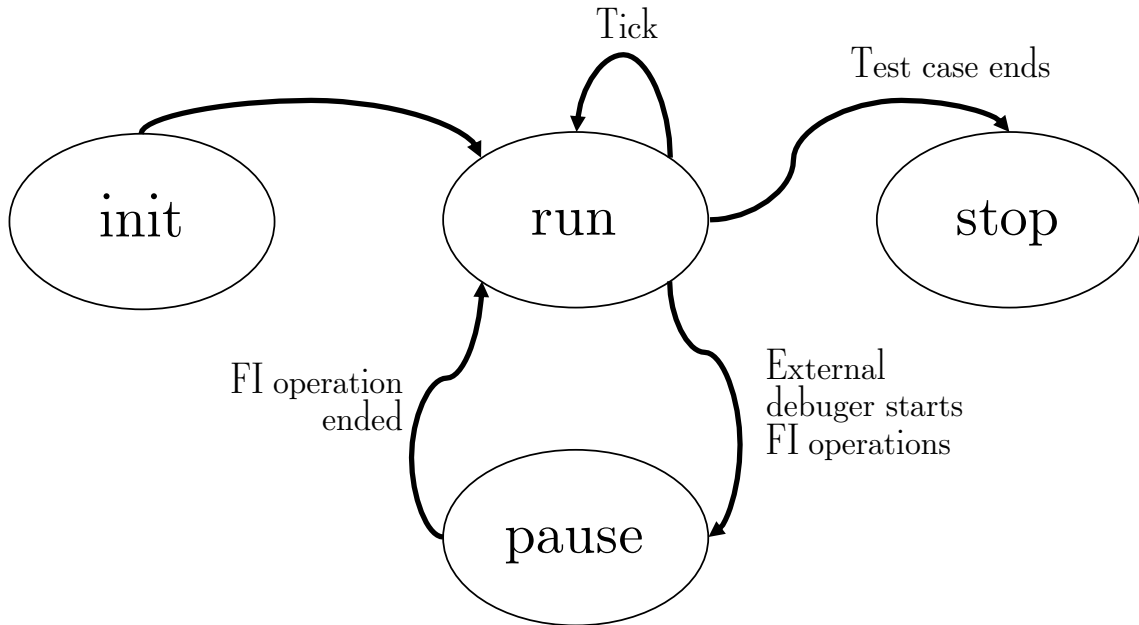


Figure 5.9: The *pause/resume Finite State Machine (FSM)* described in the approach presented in [10].

- Timeout (**TO**): the fault was detected by a watchdog timer (WDT). Such faults are often non-recoverable and require a switch to a hot stand-by spare computer in order to avoid deadline miss;
- Corrected (**C**): the fault was detected and corrected by an FDIR mechanism besides WDTs.

Such classification report is then compared to the HIL platform test report to check if the activation time of an FDIR mechanism, besides watchdog timers, caused a significant modification in the time domain performance metrics; finally, a report is generated containing details on the impact of FDIR mechanisms on the time behavior of the DUT. The data path from the DUT to the debugger is used to allow classification of faults not affecting the real-time behavior of the system.

HIL testing is a common practice in several industries; therefore, HIL testing devices are currently available on the market. In the scope of this paper, a commercial HIL platform has been used for the validation of the proposed approach. Such HIL platform features the following components:

1. a real-time computer based on a 64-bit Intel® x86 processor;
2. an interconnect system based on a Xilinx® Kintex- 7TM;
3. several I/O modules capable of managing both digital and analog signals.

The models used in this experimental evaluation have been implemented using the capabilities of the Simulink software by Mathworks®. This is a common workflow in the model-based software design (MBSD) in which a controller that has to be implemented in software is first implemented as a model and simulated together with a model of the controlled plant in order to validate the design. Once the design is thus validated, it can be translated into a low-level language. In the scope of this paper, the translation has been performed using Matlab's Embedded Coder. The controller software module thus obtained has been integrated into a MCS designed according to a combination of the recommendations contained in [94] and [95].

Thanks to the MBSD approach, the HIL platform is able to run exactly the same controller as the one implemented in the DUT, which allows for easier validation of the system as previously described. It is worth noting that, though it may be convenient, the use of a MBSD approach is not mandatory for implementation of the proposed approach; several other methods exist and can be used to perform the time domain profiling and validation required by the proposed approach.

Model implementation and verification are performed according to the following steps:

1. low-level code is generated starting from the model of the whole closed-loop system, i.e., both the plant model and the controller are translated into software that should be deployed on the HIL platform;
2. The test task thus obtained is compared to the simulation running in Simulink, thus complying with the tool validation required by safety standards;
3. The next step is to check that the HIL platform is able to run both the plant model and the controller model in real-time. This is needed in order to implement the real-time check for the DUT. If this step fails, the model design should be revised in order to speed up its execution. In this configuration, the HIL should be running all the module that it will use during the proper test phase, i.e., the communication protocols between the HIL and the DUT, the pause/resume finite state machine (FSM), a time-domain profiler which compares DUT outputs with the outputs of the golden controller, a logging service to store results before forwarding them to the workstation.

At the end of this checklist, the HIL platform is ready for the proper testing phase. The controller model should now be compiled for the DUT and deployed on it. The communication protocol between the DUT and the HIL platform allows the controller running on the DUT to send commands to the plane model running on the HIL platform and receive sensor data generated by such a model.

The system implemented on the DUT is designed according to a combination of the solutions proposed in [94] and [95]. In such a system, spatial partitioning is granted by using a type-1 hypervisor [94]. Temporal partitioning is implemented

through a statistical approach based on a profiling phase, described in detail in [95]. The DUT has been implemented on a Xilinx® Zynq-7020 APSoC, featuring a dual-core ARM® Cortex-A9 processor. Each core was used to run a task, using core 0 for the controller task and core 1 for a non-critical task, thus implementing a mixed-criticality workload.

5.2.2 Experimental results with the triggering of unsolvable known defect in a low criticality software component - [9]

Thanks to the *time-domain performance analyzer*, in charge to save the *generated stimuli* (see fig.5.6), it is possible to characterize the system’s time-domain step response characteristics, shown in fig.5.7. Statistics were obtained by processing log data recorded during the execution of 500 transitions representing the system’s step response behavior. In this case, the simulated pilot requires the UAV to perform steep pitch angle changes.

Table 5.1 shows a comparison between the responses obtained with the simulation run by Simulink to perform model-in-the-loop (MIL), as described in section 2.11.4), or to perform software-in-the-loop (SIL), as described in section 2.11.4. In this case, SIL tests have been performed by running both the plant and the controller simulation on the real-time computer, but it is possible to run them also on the workstation. It has been chosen to run them on the real-time computer to verify that the HIL simulator is capable to run the physical model in real-time without deadline misses.

	MIL	SIL
Overshoot [%]	2.31	2.33
Rise time [s]	1.6	1.6
Settling time [s]	1.6	1.6

Table 5.1: Average time-domain performance comparison between the results obtained in MIL and SIL. Table from [9].

Performance comparison between the SIL implementation and the HIL one, with the DUT running the control software (without any faulty tasks), and the real-time computer the plant model, is shown in table 5.2. These comparisons are essential to verify that the controller runs properly in nominal non-faulty cases; this process is contained in the usual digital control systems implementation workflow using the MBSD approach. Table 5.3 compares the performance obtained by performing the HIL simulations with the control software running alongside a faulty task or not.

The results shown in the latter two tables have been obtained with HIL implementations, hence the signals are transmitted on the wiring needed to connect DUT to the HIL simulator: test engineers should consider the effects of the physical disturbances on these analog signals. Therefore, we report the performance variance as well as the average collected on 500 repetitions. To simulate the behavior of the system in the presence of a fault, we use a fault injection system, able to excite a bug in the non-critical application.

To simulate the behavior of the system on the presence of a fault, a known defects affecting the non-critical task is excited. Once the defect has been excited, the non-critical application behaves unexpectedly. Separation mechanisms detect this misbehavior, and the proper recovery action is performed. To account for the low probability of the event in which the bug is excited twice in one transaction, the defect has been excited exactly once per transaction⁵. The results show that the controller can run as expected even in the presence of misbehavior in the non-critical task running in parallel on a different core of the DUT. This conclusion is supported by the absence of performance degradation in the time-domain performances, demonstrating that the delay introduced by the FDIR mechanism is sufficiently small to not affect the controlled UAV dynamic.

Indeed, all three performance parameters for the controller have negligible differences in the fault-free and fault-affected HIL simulations. Moreover, variance is low in both the cases, and very similar: its slight difference can be entirely attributed to the effects of white noise on the physical I/O signals.

Therefore, we report the performance variance as well as the average collected on 500 repetitions.

Separation mechanisms detect this misbehavior, and the proper recovery action is implemented.

To account for the low probability of the event in which the bug is excited twice in one transaction, the fault is injected exactly once per transaction. The results show that the controller can run as expected even in the presence of misbehavior in the non-critical task running in parallel on a different core of the DUT. This conclusion is supported by the evidence in the data of the absence of performance degradation. Indeed, all three performance parameters for the controller have negligible differences in the two considered cases. Moreover, variance is low in both cases and very similar. The little difference in the variance can be entirely attributed to the effects of white noise and random measurement error.

⁵A *transaction* is a steep pitch angle changes required by the simulated pilot.

	SIL	HIL (fault-free situation)
Overshoot [%]	2.33	2.34
Rise time [s]	1.6	1.6
Settling time [s]	1.6	1.6

Table 5.2: Average time-domain performances comparison between a full software implementation in the real-time computer and with the loop closed on the DUT in fault-free conditions. Table from [9].

	HIL (defect not triggered)		HIL (defect triggered)	
	Average	Variance	Average	Variance
Overshoot [%]	2.34	0.12	2.37	0.09
Rise time [s]	1.6	$1.1 \cdot 10^{-3}$	1.6	$0.7 \cdot 10^{-3}$
Settling time [s]	1.6	$1.1 \cdot 10^{-3}$	1.6	$0.7 \cdot 10^{-3}$

Table 5.3: Time-domain performances comparison between the loop closed on the DUT in fault-free and fault-affected situations. Table from [9].

5.2.3 Experimental results with FI, pause signal, and eDB - [10]

The authors of both [94] and [95] validated their solutions through fault injection (FI) simulation experiments.

The key component of these testing system, with respect to the one proposed in [9], is the presence of an eDB. The key requirements of this component have previously been outlined; such requirements can be satisfied by several commercial systems dedicated to debugging and profiling embedded systems. In the scope of this proposal, one such system has been selected. The selected design also features a proprietary protocol for interfacing with a workstation through a universal serial bus (USB) cable. The workstation should be able to run the companion software of the hardware debug system in order to implement the fault injection manager.

Synchronization between the HIL platform and the eDB is achieved through a dedicated interface. In the scope of this experimental evaluation, the interface between the eDB and the HIL platform has been implemented through an unused general-purpose input/output (GPIO) interface on the DUT, managed by a register configuration, performed by the eDB, to not modify the embedded software of the DUT, and connected to the HIL platform through one of the available I/O modules. To stop the execution of the model on the HIL platform, the eDB stops the execution of the DUT and puts its MPSoC in debug mode. Through the facilities provided by the MPSoC debug mode, the eDB can write the correct value

on the GPIO pin used as a freeze signal. The HIL platform samples the value of the freeze pin and stops the model simulation. At the end of the fault injection, the eDB must reset the GPIO pin to the original value in order to release the physical model simulation, just before releasing the MPSoC.

Synchronization between the fault injection system and the DUT is performed by means of breakpoints. Several MPSoCs provide some facilities for system debugging; one such facility are the hardware breakpoints, which allow the eDB to require that the software execution is stopped before fetching an instruction at a given address. When a breakpoint is reached, the MPSoC automatically puts itself in debug mode, allowing the eDB to control and perform necessary actions. There are two synchronization breakpoints, one at the beginning of an actuation cycle, the other at the end of an actuation cycle. The first synchronization point is used as a reference for the fault injection, which is performed at a random time relative to such reference; the second synchronization point is used as a data collector.

The profiling results, reported in table 5.4, obtained in MIL constitute the time-domain comparison baseline for the experiments described in the following section.

	MIL	Variance
Overshoot [%]	2.34	0.12
Rise time [s]	1.6	$1.1 \cdot 10^{-3}$
Settling time [s]	1.6	$1.1 \cdot 10^{-3}$

Table 5.4: Time-domain performances characteristics in fault-free conditions. Table from [10].

Test engineers performed four fault injection campaigns, each injecting a different type of fault.

The first campaign was performed to inject faults on the CPU register file (CPU RF). Its target was the architectural register file inside each core on the DUT. A second campaign was performed to inject fault inside the configuration registers of the MPSoC (Cfg. Regs.). Fault classification for these two types of faults has been performed according to the previously described classes described, and it is reported in table 5.5.

Target	NE	C	TO	F	Inj
CPU RF	1755 (87.75%)	119 (5.95%)	126 (6.30%)	0	2,000
Cfg. Regs.	3866 (96.65%)	0	134 (3.35%)	0	4,000

Table 5.5: Hardware fault injection classification. Table from [10].

A third fault injection campaign was performed to inject random software bugs in the code of the non-critical application. Results are reported in table 5.6.

NE	C	TO	F	Inj
8880 (88.80%)	1120 (11.20%)	0	0	10,000

Table 5.6: Software fault injection classification. Table from [10].

The fourth and final campaign was directed at the injection of a single specific bug in the system, similar to the one described in [95]. This campaign was directed at estimating the effects of the interference detection mechanism on the DUT temporal behavior; to collect a statistically significant result, this injection was repeated 15,000 times. In this case, the fault was classified according to the activated interference detection rule associated with one of two recovery mechanisms. The first is the alarm rule, which is associated with panic recovery. Such recovery mechanism triggers a switch to a hot stand-by spare computer; therefore, it can be considered equivalent to the TO classification used for the previously described campaigns. The second is the warning rule, which is associated with graceful degradation recovery. Such recovery mechanism reboots the non-critical task, possibly deleting the fault causing the interference from the system. If this mechanism is not successful, the watchdog timer is eventually triggered within the observation window; therefore, part of these faults is classified as TO as well. Therefore, results were classified similarly to those already presented for the other fault campaigns and are reported in table 5.7.

NE	C	TO	F	Inj.
139 (~0.93%)	2 (~0.01%)	14859 (~99.06%)	0	15000

Table 5.7: Interference detection. Table from [10].

The effects of the faults on the system’s real-time behavior have been evaluated by observing the step response of the system and comparing it to the original profiling performed in a fault-free condition.

Each fault was injected once, after waiting that the system was left running freely for a sufficient time to obtain a stable response to the last applied step transition. After such time elapsed, the DUT was reset to ensure that the previous fault did not persist in the system, and the next fault injection was performed. Using this approach, the testing system allows identifying the deviations from the expected behavior introduced by either the fault itself or by the FDIR mechanisms. The HIL platform produced an intermediate report detailing the listed characteristics for each fault injection. Such information is not human-readable but is intended to be processed by the workstation, which, by crossing it with the fault list, and the fault classification criteria, evaluates the impact of each fault on the final behavior

of the DUT, including the contribution of any activated FDIR mechanism, if any. The result of this operation is a human-readable report that summarizes the effects of the injected faults on the DUT and contains the specific impact of each fault. In the performed experiments, we considered the DUT as part of a hot stand-by spare configuration; therefore, effects on its time behavior of a fault classified as TO are neglected in the scope of this evaluation since they would cause an activation of the hot stand-by spare and a negligible system-wide effect. Faults classified as C are instead of interest. This includes those faults detected by a mechanism that causes either skipping of one actuation cycle or reset of the non-critical task. Both such events have a negligible effect on the system's time-domain behavior and are not observable by the HIL platform.

5.3 Multi-agent robotic system (MAS) development

The proposed methodology, presented in 2018 at the 23rd International Conference on Emerging Technologies and Factory Automation (ETFFA) and published in its proceedings, relies on Hardware-In-the-Loop (HIL), described in section 2.11.5, and it is accelerated by Model-Based Software Design (MBSD), described in section 2.11.3. In this proposal, the HIL validation techniques, widely adopted in the automotive and avionic industry, are used as a Computer-Aided Design (CAD) to develop a multi-agent robotic system (MAS).

Roboticians develop and use physical models during the whole development process of a new cyber-physical system. This approach is ideal for industrial and mobile robotic applications. The hardware is bought from specialized suppliers, which provides an appropriate firmware for the various peripherals installed on the robot and a suitable central control software that executes the planned end effector movements. In this case, the customers have only to develop custom software modules suitable for their application requirements. Usually, these robots are controlled by a unique computation system.

But some robotics applications are based on the multi-agent approach: the manipulator is composed of different subsystems, independent from the computational point of view, and responsible for managing a simple individual task (agent level [103]). All the individual tasks share the common goal to perform, collaboratively, a single task at the entire robot level.

This approach requires a very flexible and modular hardware/software architecture involving a central control system to distribute the agent-level tasks, wireless communications, battery management, and an accurate synchronization between the various agents. To benchmark the capability of the proposed methodology and verify the fulfillment of a set of real-time requirements on the behavior of a robotic system, it has been assessed on a customized multi-agent system.

5.3.1 Proposed approach

Starting from the models developed during the previous design phases, it is possible to run them on real hardware components and in a real-time manner.

Fig.5.10 shows a possible, simple architecture for a MAS, where some agents and an external centralized controller are involved.

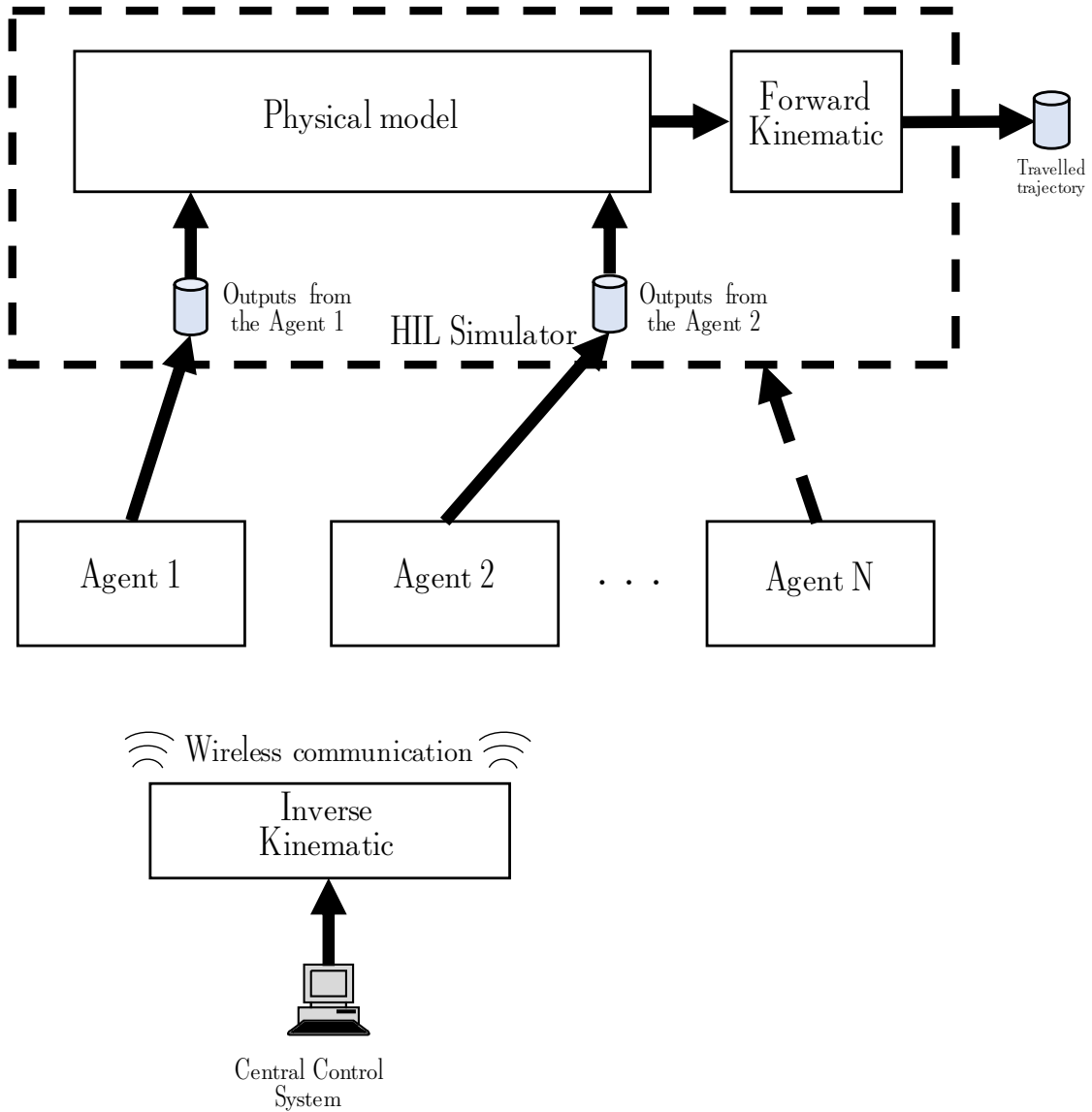


Figure 5.10: The block diagram of a generic MAS. Adapted from [11].

Testing the software components of a such a system is a very challenging. Their architecture is complex and error-prone: they involve a central controller implemented on a workstation (not real-time) that provides the Human-Machine Interface (HMI) and has to control, with non real-time or asynchronous radio-frequency protocols [107] (like Bluetooth, Wi-Fi, ZigBee) the agents, whose software is implemented on low power embedded systems.

HIL technique, due to its intrinsic modularity, can be used to test the various components of the complete system. It can also simulate failures and analyze how the embedded software is capable of reacting to them. MASs are intrinsically prone to failures due to loss of wireless communication, a discharge of the batteries, or a caught agent.

From these premises, it is possible to think about an appropriate methodology to aid MAS developers.

The proposed methodology combines, in the most useful way in terms of data and models availability, the three levels of test verification (MIL, SIL, and HIL, as described in section 2.11.3). A flow chart for the proposed methodology is shown in fig.5.11. In this way, this proposal aims not only to obtain an end-of-line verification system to test the initial release and successive software updates, but to obtain a CAD tool able to aid the software development during all the development phases.

For each release of the software, developers can:

1. check, through a MIL simulation, if the developed or upgraded algorithm can control the system properly;
2. generate the code and integrate it into a test target to run the automatically generated code alongside the various software components in a SIL situation. At this stage, in particular, it is possible to check the various application-level communication protocol without synchronization problems.
3. HIL is performed: the agent-level control software is run on the agents, the system-level control software on the central controller, while a real-time simulator runs the physical model of the system.

The adoption of MBSD (see section 2.11.3) accelerates these steps, allowing to modify the software and run the tests, over and over again, until all the real-time requirements are met.

To implement such a system, these components are required:

- a model-based software design tool;
- an automatic code generator, to *translate* the models into source code;
- a real-time computer;

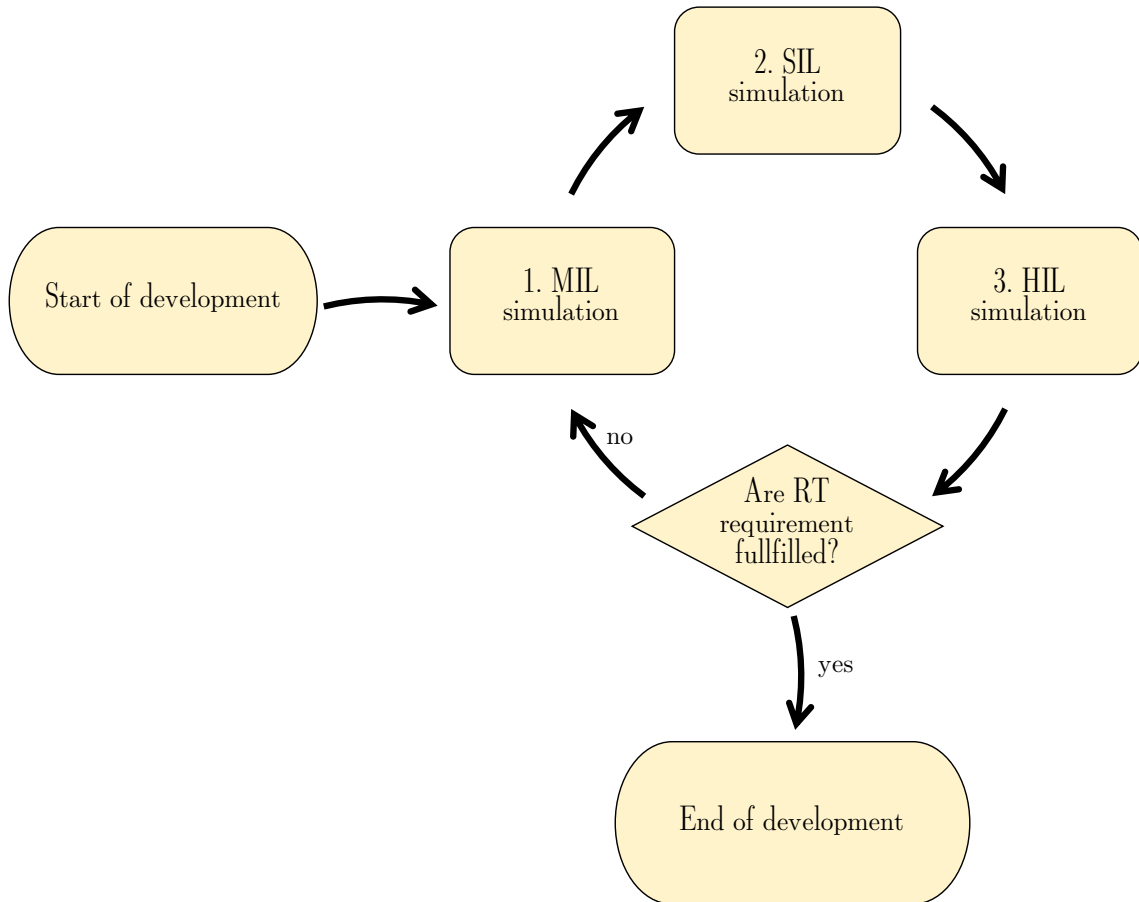


Figure 5.11: Flowchart of the proposed approach. Figure from [11].

- a software environment to log, in a real-time manner, the physical models simulation results.

5.3.2 Benchmark application

The physical structure of the benchmark MAS is shown in fig.5.12.

It is a Delta robot [108] manipulator where two independent agents are constrained to move along the horizontal rail scaffold. The end effector is linked to the agents with two rigid arms. Consequently, its *pose*, w.r.t. the world reference frame R_0 (task space), can be computed from knowing the two agents' positions on the rail (joint space).

By denoting as p the dimension of the joint space and m the task space dimension, it results that $p = m$. It means that its kinematic is fully determined, and it is possible to demonstrate that a biunivocal correspondence between the two spaces exists [105].

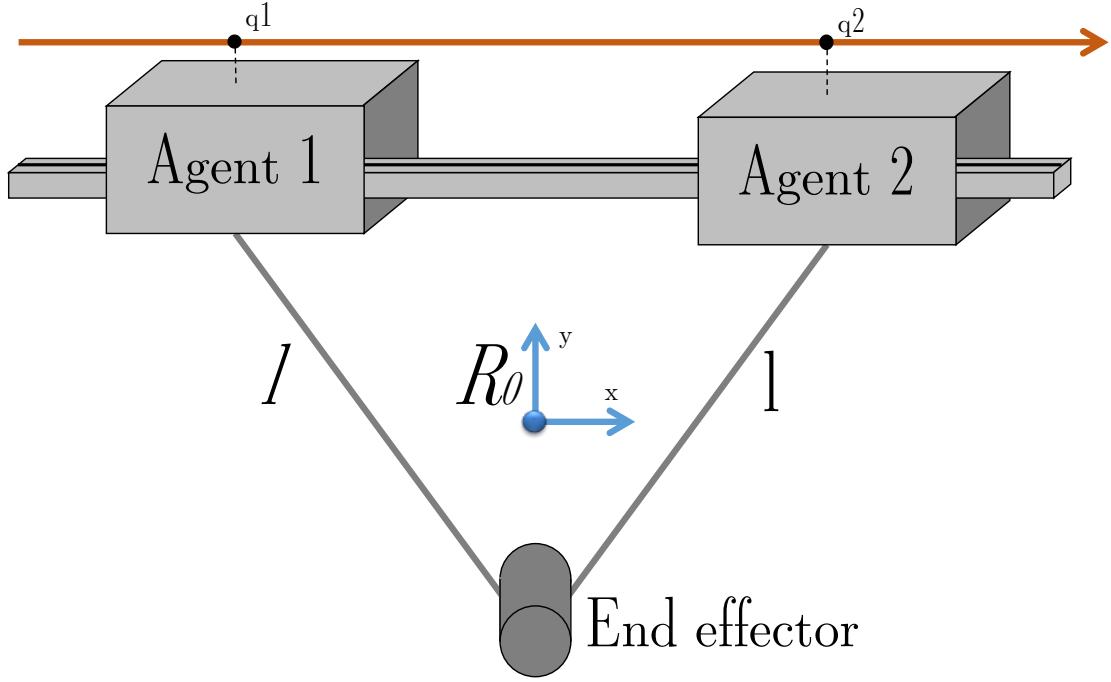


Figure 5.12: The mechanical structure of the benchmark MAS. Adapted from [11].

The proposed geometry is a simplified version of the robot already described in paper [107].

Every agent is equipped with two DC motors. Motion is guaranteed by the direct coupling of their gears with the rail rack system attached directly to the scaffold. The agents can measure their positions q_1 and q_2 on the rail (joint space), by linear magnetic *hall effect* encoders.

The Forward Position Kinematic (FPK) solution, in this particular case, is very simple since it is possible to compute the position $\bar{r} = [x \ y]$ of the end effector into task space, starting from the joint-space coordinates q_1 and q_2 in a closed form, by using the three sphere intersection principle [106]. It is possible to find the entire mathematical description on the paper [11].

The x coordinate it is computed as:

$$x = \frac{1}{2} \cdot \frac{q_1^2 - q_2^2}{q_1 - q_2} \quad (5.1)$$

while the y one can be computed as:

$$y = \sqrt{l^2 - (x - q_1^2)} \text{ or } y = \sqrt{l^2 - (x - q_2^2)} \quad (5.2)$$

5.3.3 Control software and models

For what concerns the simulation of our manipulator, a complete model has been developed. Its block diagram is shown in fig.5.13.

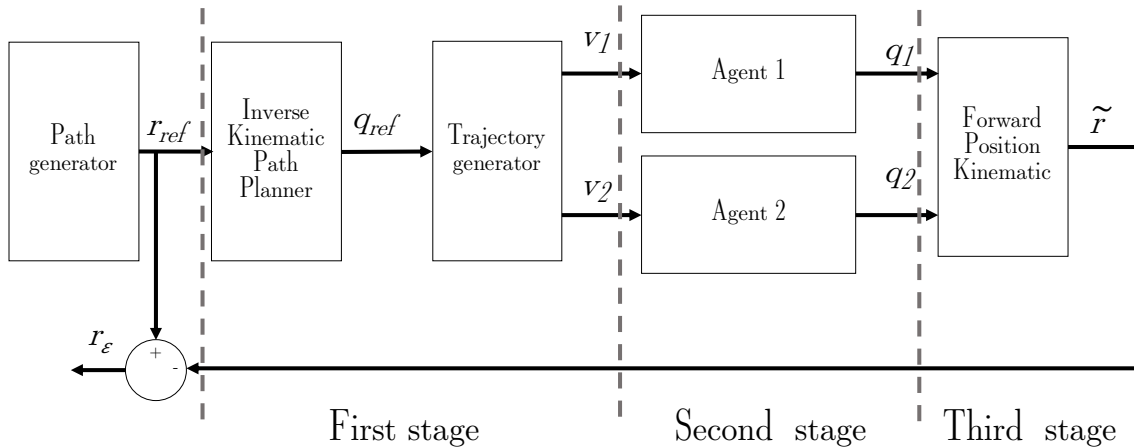


Figure 5.13: The control software block diagram.

The control system can be split up into three main stages.

The first one receives the open-loop path in the task space, and it is responsible for both the point-to-point path determination, and the trajectory⁶ generation, for the two agents.

The second stage is the agent level closed-loop control system. It is responsible for actuating on the task of the agent by following the trajectory determined by the first stage.

The third and last stage takes as input the positions of the agents and produces the real path performed by the end effector.

This subdivision may seem not appear relevant, from the simulation point of view since, in this condition, all the blocks are executed on the same target computer. But since they will be integrated into different target devices (the two agents and the real-time simulator), the interfaces between them are critical, and the system architecture shall be capable of properly managing their possible failures.

The path generator provides the open-loop reference signal the manipulator has to follow. It is expressed in the task space reference system. It has been programmed to generate a circular path with a radius of 1 m centered, in the task space, at coordinates $x = 0$ m and $y = 2$ m. This path is then transformed by the Inverse Kinematic Path Planner (IKPP), which provides the reference set of the

⁶A trajectory is the combination of the point-to-point path with the indication of the velocity to keep in each of the points.

agents' generalized (joint space) coordinates q_1 and q_2 . These could be used, from a theoretical point of view, as the reference positions signals for the agents but, in general, these are not used as direct input of any actuator to avoid issues regarding two important general requirements aspects:

1. Coordination between the agents. The generalized coordinates, transformed point by point by the IKPP, lead to trajectories different from each other in terms of their duration. Therefore, if these are directly used as the reference signal for all agents, even supposing they are identical from the dynamic point of view, they will have a non-coordinate movement behavior, producing vibrations and instability in the end effector position. This aspect is crucial when a continuous path needs to be performed, but good coordination remains preferable even in point-to-point applications, like pick & place tasks. It is generally done using 2-1-2 velocity profiles generators [105].
2. Time constraints on the path reproduction. It is possible that, for a given task, a particular path, which is the union of several other small parts, needs to be executed at a specific speed in order to respect fabrication process requirements (for example, welding). Apart from the path design, a trajectory definition is needed, which is, in simplified words, the same path with the indication of velocities needed to achieve the given temporal constraints.

These issues have been described to justify that q_1 and q_2 cannot be directly used as the input reference signals. Hence, after the IKPP block, a trajectory generator is included to provide reference speed profiles v_1 and v_2 for the agents. It takes a generalized coordinate q_i and compute v_i as a differentiation of the first order:

$$v_i = \frac{q_i - q_{i-1}}{\eta}$$

The agent blocks, which represent the second stage of the block diagram, follow with a closed-loop control system the velocity (and not the position) reference according to the agents' dynamic performance. The outputs provided by these blocks are the positions q_1 and q_2 of the agents on the rails, needed to compute the task space error \bar{r}_ε and hence close the external position closed-loop control system (not discussed in this proposal).

These positions become the input of the third stage block that computes the Forward Position Kinematic (FPK) solution to obtain the real path $\tilde{r}(t)$ followed by the end effector of the manipulator, which can be used to close an external position closed-loop controller.

5.3.4 Experimental results

The proposed results have been obtained by assessing the benchmark MAS performances while its end effector runs through a circular path with a radius of 1 m at the constant speed of 1 m/s.

The agents receive the speed profile offline, so they start to run across the requested path with their expected trajectories already loaded. To avoid the development of a trajectory loader, and since the path in the task space is a simple circle, the path generator, the IKPP, and the trajectory planner have been implemented into the agents themselves. For the same reason, during the HIL simulations, the target also runs the agent's physical model. Since the application deployed on the agents is more demanding with respect to the real one, it is possible to say that if this benchmark is capable of running in a real-time manner on the chosen microcontroller based on 120 MHz ARM Cortex-M, also the simpler real implementation will be.

In both the iterations, during HIL simulations, the agents communicate their position to the real-time simulator by a non-synchronized 7-bit parallel bus. The position is transmitted on the bus in two consequent time steps. By doing so, it is possible to extend the position resolution from 7 to 12 bits (the seventh bit is used to indicate if we are transmitting the higher or the lower part of the word), having evident benefits on the accuracy of the computed path traveled by the end effector $\tilde{r}(t)$.

To assess the obtained results, these numerical metrics have been computed:

- Root Mean Square (RMS).
- Best Fit (BFIT), which gives an estimation of how much the estimated path from the generalized coordinates \bar{r}_{ref} and the FPK solutions are close to the real synthetic path \tilde{r} . It is computed as

$$\text{BFIT} = 1 - \sqrt{\text{RMS}^2 / (1/N_{\text{sample}} \Sigma(r_\epsilon)^2)}$$

where, as shown in fig.5.13, $\bar{r}_\epsilon = \tilde{r} - \bar{r}_{\text{ref}}$

- average speed of the end effector center point, which has been computed always starting from the knowledge of the estimated coordinates x and y coming from the FPK block.

To obtain more realistic results, the MIL simulation models take into account the quantization errors and the downsampling⁷ due to the transmission protocol⁸.

⁷The precision of the joint space positions (generalized coordinates) q_1 and q_2 measures is limited to only 12 bits.

⁸The transmission rate have been halved to obtain 12 bit precision.

The obtained simulation results show how the quantization error and the downsampling phenomena produce an evident error in the estimated path. BFIT is reduced at 70%, from the theoretical 90% obtainable without any protocol, with the joint space positions represented as 64 bits IEEE 754 floating-point numbers.

The average speed of the end effector center point is almost 25% greater than what was expected (1 m/s). This is caused by the quantization error on the generalized coordinates that made the end effector follow a path longer w.r.t. the theoretical one.

This is evident in fig.5.14, where it is possible to see that the actual path is more rough w.r.t. the expected circular one. On the plot it has been superimposed a circular crown to show the error interval on the task space, due to the quantization error affecting the generalized coordinates. The same model has been integrated into the simulation target device to perform SIL simulations, and their results are numerically identical in both MIL and SIL situations, as expected, since the generated code shall behave in the same way to its source model. It proves that the code has been generated correctly.

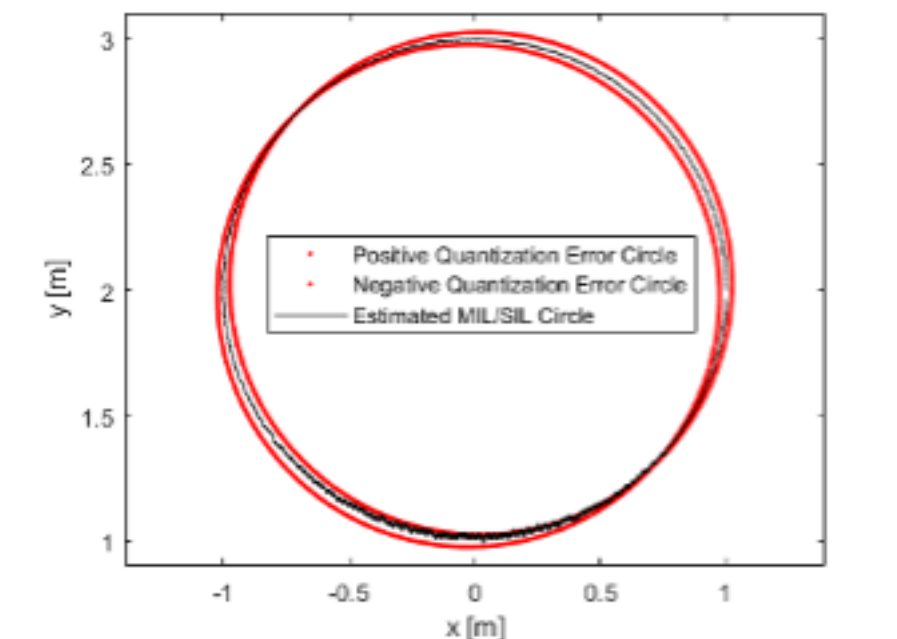


Figure 5.14: MIL/SIL estimated path. Figure from [11].

Now, it is possible to analyze the results obtained in HIL.

Here it is important to remember that three different targets are used: the real-time HIL simulator and the two agents.

The block diagram of the SW architecture adopted for HIL testing is shown in fig.5.15.

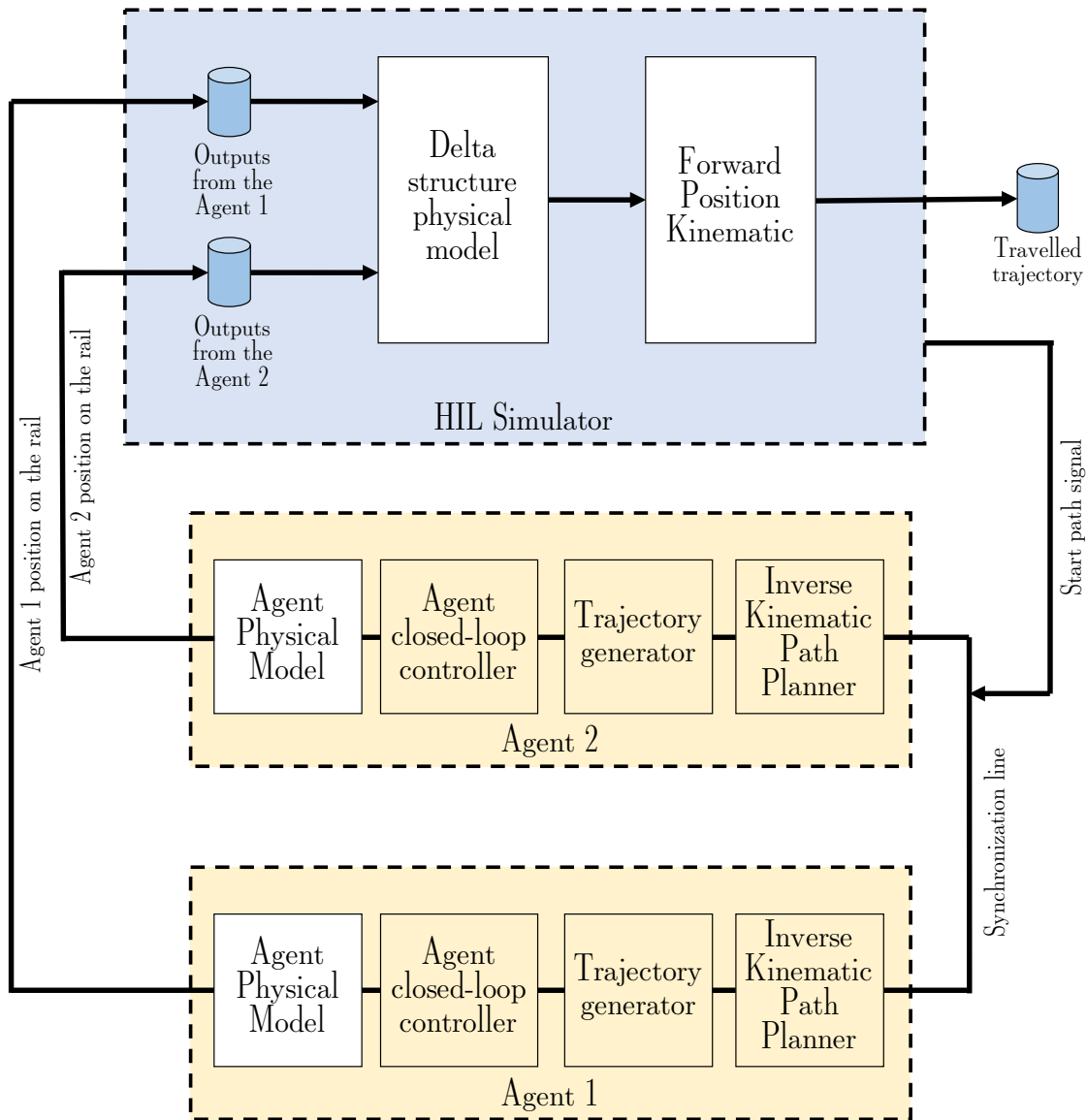


Figure 5.15: Block diagram of the SW architecture adopted for HIL testing.

The agents have been implemented, resorting to a 120 MHz 32-bit ARM Cortex-M unit, making use of two NXP FRDM K64F evaluation boards. All the software, except for the firmware level managing the System on Chip (SoC) peripherals, have been developed by MBSD.

The real-time simulation has been implemented on a National Instruments myRIO-1900 real-time computer, based on a Xilinx Zynq-7010 SoC. A photograph of the system is shown in fig.5.16.

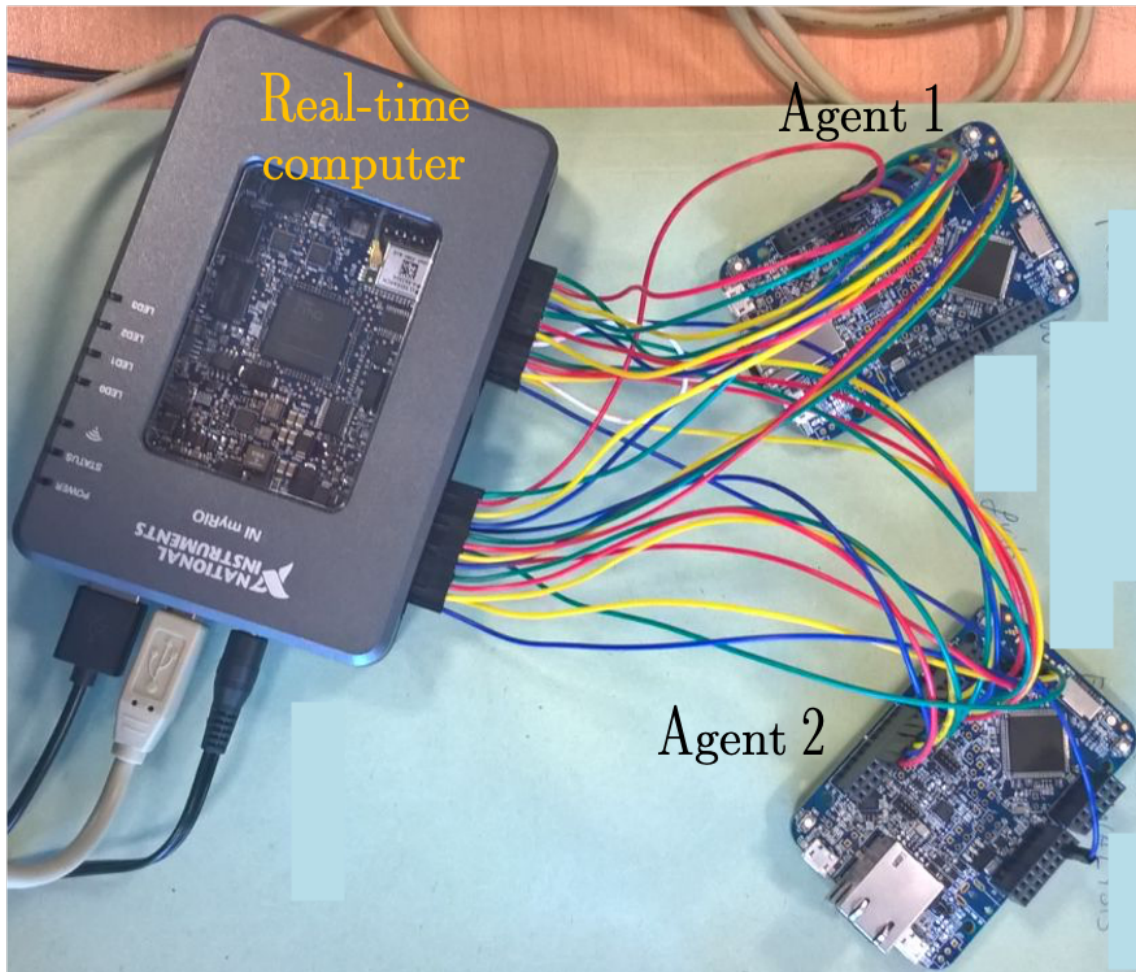


Figure 5.16: A photograph of the experimental setup described in this proposal.

It is also important to remark that the trajectory generator provides the speed profile to the agent closed-loop controller with 64 bits floating-point precision since it is run by the microcontrollers embedded in the agents.

Several (65) runs have been performed to evaluate the performance, from a statistical point of view, in terms of accuracy and repeatability. As shown in fig.5.17, for all the attempts, a uniform distribution proves a positive result in terms of repeatability. In other words, the HIL demonstrates a deterministic behavior of the chosen architecture. Furthermore, in terms of BFIT and RMS, as shown in fig.5.18, the results are similar to those obtained in MIL and SIL simulations (with quantization error and delay): this is not an obvious result, as it can appear at a first time, according to the fact that during the HIL the software is distributed, as in the real world, on systems with different clock sources. From this evidence, it is possible to affirm that the agents are behaving in a real-time manner.

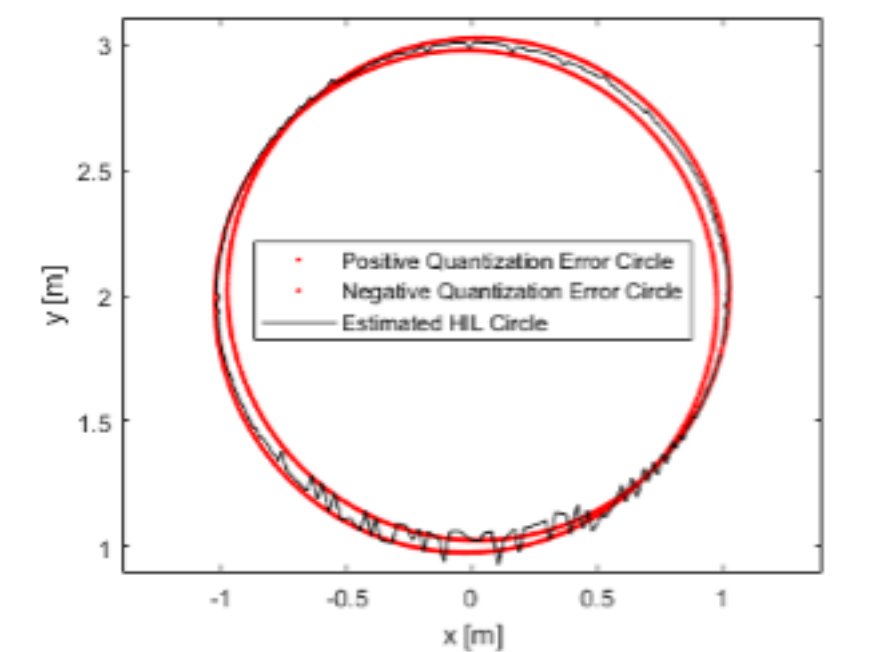


Figure 5.17: HIL worst-case estimated path. Figure from [11].

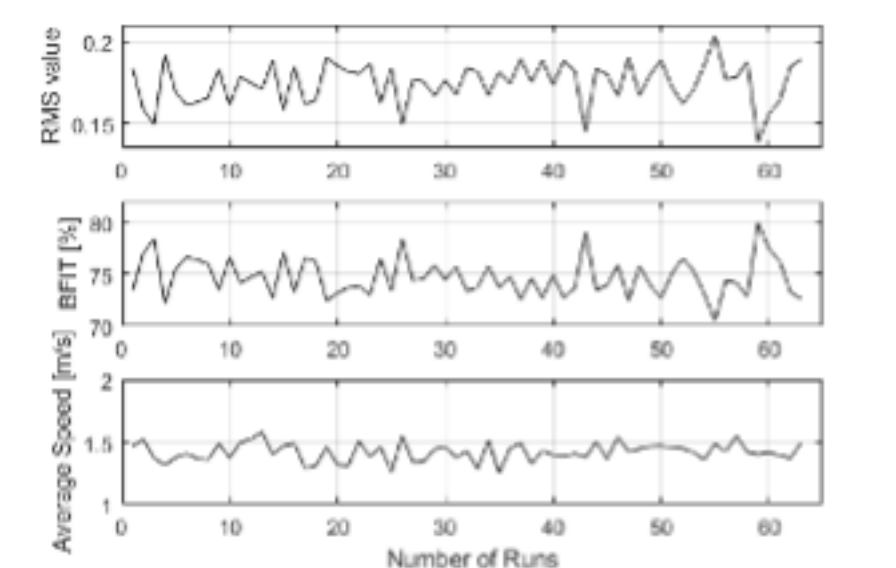


Figure 5.18: Results obtained from 65 HIL simulations. Figure from [11].

Chapter 6

Conclusions and Future Work

This dissertation proposed various improvements on methodologies applied to increase the dependability of microcontroller-based electric and electronic components. This is a key point for those systems in charge of performing safety-critical tasks.

The methodologies can be summarized into three different topics:

- simulation-based FMEDA;
- simulation-based HARA;
- real-time software validation.

Most of the proposed methodologies are designed for automotive applications. Anyway, considering simulation-based FMEDA and real-time software validation, also robotics and avionic (unmanned aerial vehicles) applications have been considered.

It is useful to map the proposed methodologies on the phases of the *safety lifecycle* described by the standard ISO26262, as shown in fig.6.1.

6.1 Contributions on Simulation-based FMEDA

The FMEDA technique is described by the ISO26262 part 6 (HW development) as a technique to verify if a hardware design fulfills the reliability metrics (reported in tab.2.7), based on the ASILs (see section 2.2) of the functions it is in charge of, required by the ISO 26262.

The contributions presented on this topic are mainly applied to automotive industry case studies. Notable exceptions are the two last proposals, described in sections 3.9 and 3.10 which are about, respectively, to an industrial and a mobile robotic application. Thanks to the collaboration of people from the Power Electronics Innovation Center (PEIC) of the Politecnico di Torino, the industrial application

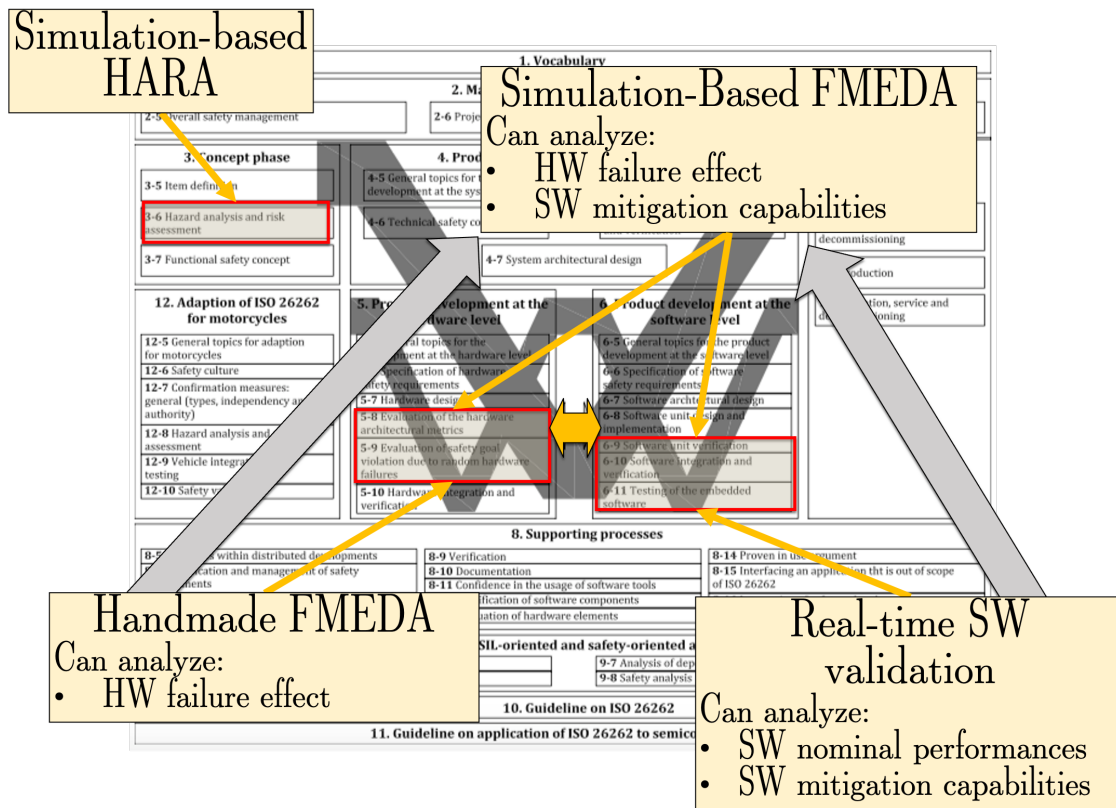


Figure 6.1: The mapping of the three proposed approach on the ISO26262:2018 structure.

benchmark has been chosen to describe how to adapt the methodologies proposed to perform the ISO26262 FMEDA to aid the FMECA. The mobile robotic application, instead, has been chosen for two main reasons:

- it extends the analysis, performed by assessing the failure mode effects assessment at the vehicle-level (described in section 3.8), to a design-FMEA of an entire mobile robotic cyber-physical system;
- it allows to avoid the limitations imposed by the intellectual property protection needs usually applied to a real automotive item. These make it impossible to release publications on this topic, considering one of them as the benchmark application. In this case, the analyzed system has been designed by a students' team of Politecnico di Torino, so there are no such restrictions.

The first iterations, published into the papers [15], [16], [18], described in the sections 3.5 and 3.6, have their assessment capabilities limited to the item-level. It prevents the usage of the simulation results in those cases where safety engineers have to assess the failure effects by considering how they can affect the vehicle

behavior.

This limitation has been overcome in the proposal [19] described in section 3.8, obtained by propagating, thanks to a vehicle-level simulator, the failure modes item-level effects up to the entire vehicle. This approach has been obtained by using structural (SPICE-level) models for both the item and FMs, instead of the behavioral one, as proposed for the simulation-based HARA, published in [65] and [66], as described in chapter 4.

6.1.1 Advantages

By aiding the FMEDA thanks to a simulation-based approach, it is possible:

- to improve its *objectivity* and the *repeatability*.
The *objectivity* is augmented thanks to the presence of the models and the classification rules. These allow decoupling the analysis results from the safety engineers knowledge.
The *repeatability* is improved since the failure mode effects classification, between safe and dangerous, is performed by assessing, through mathematical rules (that can be expressed in both absolute terms, considering the system specification, or w.r.t. the fault-free behaviors), the behaviors of the fault-affected simulated system. If the item model includes the adopted mitigation strategies, it is possible to assess their effectiveness, allowing to keep into account the effect of the embedded software. Moreover, the failure detection capabilities are assessed by simulating the detection systems the item embeds (regardless they are implemented by hardware or software strategies), allowing to perform the detected/undetected classification.
Even if systematic errors are possible in both the model and classification rules, the approach allows to repeat the needed simulations, or modify the classification rules, whenever mistakes are found.
- considering only the proposals where the FMs effects are propagated up to the vehicle level, the methodologies allow assessing the developed detection and mitigation algorithms and, consequently, the FMs effects classifications. This is particularly useful in those cases, like the one described in section 3.8 about a dual-motor axle, or the one described in section 3.10 about the mobility subsystem of a mobile robot, where it is difficult to evaluate the failure consequences at the system (vehicle or robot) level due to the presence of complex interactions between the considered failed item and the whole system;

- consider the FMEDA as a CAD tool, by using it during the schematic design to find the critical components or subsystem and focusing the effort on improving their design (this is true especially when the simulations are performed thanks to multi-level models as proposed in section 3.9 to perform FMECA);
- reduction of the time-to-market: the simulations are faster with respect to a handmade analysis, and the presence of the models and semi-formal classification rules allow a speed-up of the confirmation measures required by the ISO26262 standard.

6.1.2 Limitations

The main limitations of the proposed methodologies are inherited from the FMEDA process. The most relevant one is the difficulty of considering the effects of multiple failures happening at the same time. The number of the required simulations increase exponentially with the number of contemporary failures to be considered, following the rule $s = n_{\text{fm}}^n$, with s is the number of needed simulations, n_{fm} is the number of failure modes, and n is the number of simultaneously present failure modes.

By now, the only way to overcome this limitation is to select, manually, what are the simultaneous failure modes to be assessed. In any case, the proposals described in this dissertation do not investigate this possibility. The only exception has been made for the detection algorithms described in the mobile robotic case detailed in section 3.10, where FMs are grouped.

Another subtler limitation is the need for deep expertise to choose the required fault models' detail level. At the moment, the best tradeoff has been found with the multi-level models described in section 3.9.3.

6.1.3 Future development

By now, the proposed approaches lack two features that can be useful:

- a way to automate the generation of test sets to find and test double-point failures¹;
- a methodology to automatically generate the classification rules.

.

¹See 2.5 for more information on mechanisms for latent faults avoidance.

6.2 Contributions on Simulation-based HARA

The hazard analysis and risk assessment is a crucial phase of the ISO26262 safety lifecycle. It is performed during the concept phase (described in part 3 of the Standard). It aims to define the safety goals (see section 2.2) for the item under development and associate an ASIL to each of them. This level is proportional to the maximum risk level the people inside or surrounding the vehicle are exposed to if the safety goal is violated.

The development of this approach started from the methodologies described in the section 3.6 and proposed in the papers [16] and [18] to perform the simulation-based FMEDA.

They have been adapted to HARA. This analysis is performed during the concept phase, so before the item's schematics have been designed. The lack of schematics makes it necessary to obtain models based only on the vehicle's behavior. Moreover, the failure modes are described only in terms of erroneous behaviors of the affected item.

Thanks to these models, and a vehicle-level simulator, it is possible to assess how the effects of the wrong behaviors of the item can affect the entire vehicle and hence the driving safety. The ISO26262 requires to perform the assessment considering three different risk parameters: *severity*, *controllability*, and *exposure*. Their combination, by means of the determination matrix shown in fig.2.5, leads to an ASIL level associated with the considered hazard. The proposed approach allows determining, thanks to the simulation results, two of them: *severity* and *controllability*.

6.2.1 Advantages

The proposed approach aids the HARA phase in three different aspects:

- improves its objectivity, thanks to the adoption of risk parameters classification tables. It is not a novelty of this approach, but the classification, performed by comparing the ranges defined in these tables with the simulation results, make the tables themselves be systematically adopted;
- improves its repeatability, thanks to the adoption of semi-formal models of the item and its failures, alongside the presence of a commercial vehicle-level simulator. These allow to repeat the simulations in the case there are mistakes in the models or into the risk parameters classification tables;
- the simulation environment can be kept for the next development phases, accelerating the simulation-based FMEDA presented in chapter 3 in this dissertation, MIL (see 2.11.4), SIL (see 2.11.4) software unit verifications, and HIL (see 2.11.5) software integration tests.

6.2.2 Limitations

The main limitation of the proposed approach are:

- it does not describe how to obtain a complete situation analysis;
- it does not describe how to perform the association between hazards and safety goals since they are described only in natural language;
- it cannot aid the exposure determination.

6.2.3 Future development

There are basically two aspects that deserve to be further investigated:

- analyze the possibility to determine the exposure risk parameter from the simulation-based approach, for example, by determining the duration of time when the hazard is present. It has not been investigated since it requires the definitions of a great number of scenarios and the determination of their simulation time over the total simulation time;
- perform a parallel like it has been done for the simulation-based FMEDA (see section 3.7), where the simulation-based results are compared with the ones obtained, by hand, by human experts. Of course, the risk parameters classification tables have to be determined by hands and should be the same for both approaches to perform the assessment.
- adopt a language like Scenic [74] to describe the scenarios in a probabilistic way.

6.3 Contributions on Real-time software validation

Real-time software validation is crucial for safety-critical embedded systems with hard real-time requirements.

These tests aim to verify that the embedded software, considered as the application integrated with the device drivers and other supporting components, can run inside the allotted time slots without deadline misses and providing correct results.

The core of this approach is the real-time simulator, composed of a software infrastructure that can manage the communications between a host computer, providing the HMI, and a real-time computer, that is a special-purpose device in charge of running the plant model in a real-time manner and providing the plant responses to the device under test through signal conditioning peripherals.

Real-time software validation has been widely applied to closed-loop control systems in almost all industrial sectors. For this scope, on the market are available plenty of off-the-shelves tools. In the literature, such tests are called as Hardware-In-the-Loop (HIL), as described in the section 2.11.5 of this dissertation. It is evident that, for these kinds of applications, there is no room for improvements.

But, following the current market trends, it is possible to find out that, for at least three different applications domains, there are no proposals to perform HIL testing on them:

- automotive body control modules [8], described in section 5.1;
- mixed-criticality avionic systems [9] [10], described in section 5.2;
- multi-agent robotic systems [11], described in section 5.3.

6.3.1 Automotive Body Control Modules

Software is a critical element in modern vehicles, as it is responsible for supervising and controlling any function they provide, from comfort to safety, driving, and connectivity.

The software is often responsible for safety-critical functions, which mandate a structured development process, where its validation and verification is a crucial aspect to meet the stringent quality levels required by international standards like the ISO26262. Tier-1s, producing embedded hardware and software for automotive applications, are striving to adapt their development and validation flows to comply with this challenging scenario.

In this proposal, an approach to support efficiently the crucial tests phase known as integration tests, where safety engineers shall develop test cases to check the correct functionalities of the software running on the embedded hardware, is described. The intrinsic complexity of the functionalities the software delivers, the need for systematic test application and output report evaluation, as well as the complexity of interfacing with embedded hardware for automotive applications, motivated the development of a novel approach specifically designed for the validation of the software developed for the so-called Body Control Module (BCM) items. The result of this proposal is a tool capable of significantly improving the established test flows, very close to the needs and know-how of test case developers for this class of items and the adoption of third-party reconfigurable hardware platforms.

Advantages

The proposed approach:

- reproduces the low-end manual testing platforms already used by tier-1s, allowing the test operators to move to the new platform with little or no training;

- makes it possible for the test operators to save and replay tests, improving the repeatability of the tests;
- allows the possibility to perform the tests from a remote location, reducing the needed time and also allowing software developers to have the possibility to test their newly developed software without having to wait for the availability of a test operator;
- the customers can use their already owned HIL platforms, without the need to buy new hardware;
- makes it possible to ask test operators to perform MIL and SIL tests without any training.

Limitations

The proposed approach has the following limitations:

- the test engineer needs to know how to develop on the third-party HIL platform;
- with National Instruments third-part HIL platforms it is necessary to develop a MathWorks Simulink model to put in communication, with a mechanism based on labels, the tool with the companion software;
- the users need to acquire licenses for both the proposed tools and the companion software.

Future development

It should be investigated how to add automatic configuration from the tool to the companion software and the opportunity to adopt as third-part HIL hardware from more vendors.

6.3.2 Mixed-criticality avionic systems

This proposal is about a novel approach for the validation of mixed-criticality systems using a HIL-based technique. It was benchmarked on an application designed for flying an airplane-shaped unmanned aerial vehicle (in particular, it controls the longitudinal behavior of the plane). The purpose is to prove that the HIL can either detect a failure, i.e., a deadline-miss affecting a safety-critical task, or prove fault tolerance, i.e., when the safety-critical task continues working as expected even in the cases the non-critical application sharing the same elaboration unit, or the elaboration unit by itself, are affected by faults. In the case of elaboration unit faults, only transient ones are considered since the permanent faults prevent any possibility of recovery.

Advantages

The proposed approach:

- allows to verify that the partitioning [9] and FDIR [10] mechanisms have an execution time with negligible effects on the real-time performances of the system;
- allows (only in the version presented in [10]), thanks to the presence of an external debugger, to perform fault injections.

Limitations

The proposed approach is not sufficient, by itself, to validate the mixed-criticality system, but it shall be accompanied by formal validation methods as required by DO-178B/C or be used only on already validated partitioning or FDIR mechanisms.

Future development

Regarding the combined use of an external debugger and HIL, it can be investigated the possibility to measure, on a benchmark application, the code coverage metrics reached with the simulated scenarios, allowing to obtain an objective metric about the completeness of the performed tests.

6.3.3 Multi-agent robotic systems

This proposal is a CAD methodology to aid the development of the software intended to equip multi-agent systems (MAS). These systems are challenging since their behavior is determined by independent agents, which needs to be synchronized without a reliable clock distribution system. The goodness of the proposed approach was put to the test on a benchmark application. As proved by the obtained experimental results, the methodology has demonstrated itself capable of aiding designers to reach, with an iterative approach, the best control architecture and parameters. Thanks to adopting a HIL-based technique, this is done without the need to act on the real system, but only resorting to a real-time simulated environment.

Advantages

The proposed approach:

- allows to test multi-agent robots without the need to have a prototype, reducing validation and verification costs, since possible errors do not lead to damages on the real physical system;

- it is modular, allowing to simulate and run software components already integrated on their target, alongside components not already deployed to their final targets, that can be run by the real-time computer: in this way, the methodology is not only an end-of-line tool, but also a CAD system capable of easing the reaching of the expected real-time performances during the embedded software development;
- re-uses the physical model of the plant implemented for the development of the control software, as required by the workflow typically followed during closed-loop control systems design, to verify the real-time performances of the target applications.

Limitations

The proposed approach does not allow to perform fault injection to assess how the MAS is affected in the case its communication systems or one or more agents fail.

Future development

Two future development possibilities should be investigated:

- test the approach on a benchmark application featuring wireless communication between the agents and the centralized controller;
- integration of the proposal with a fault injection system, allowing to evaluate, in a simulated way, how the possible faults can affect the multi-agent system at the society level (or, in other words, if a failure can prevent the MAS to perform the required task). It can be useful to develop mitigation algorithms capable of recomputing the agent-level task exploiting the system's dexterity (if any).

Appendix A

Machine learning applications for the automotive industry

The ideas and the results presented in this appendix have already been published into the three papers [109], [110], and [111], and the Masters' Degree thesis of Antonio Costantino Marceddu [112].

In recent years a rapid evolution of the entire automotive industry happened, pushed by connectivity, electrification, and autonomous driving challenges. Regarding autonomous driving, an increasing number of cars, including cheaper ones, now have various advanced driver assistance system (ADAS), and the manufacturers are racing to obtain vehicles capable of driving by themselves, at least in the most common situations (see sections 1.2.1 and 1.2.2).

Researchers are then concentrating their efforts on all the aspect that could characterize their market success.

Therefore, to analyze autonomous driving, it is needed to involve not only the technical aspects (mainly related to functional safety and cybersecurity) but also ethical and social ones. Fig. A.1 shows these challenges alongside the most common open questions on safety, security, ethical, and social acceptance aspects.

People of the generations Y (millennials)¹, Z², and Alpha³ were born in an era where rapid, sometimes epochal technological changes occurred: for this reason, for most of them, the transition to a fully autonomous driving car it is expected to be easier with respect to the older generations. Therefore, methods will be needed to improve people's response to this type of vehicle to improve their social acceptance.

¹People born between 1981 and 1996.

²People born between the 1997 and 2010.

³People born since 2010.

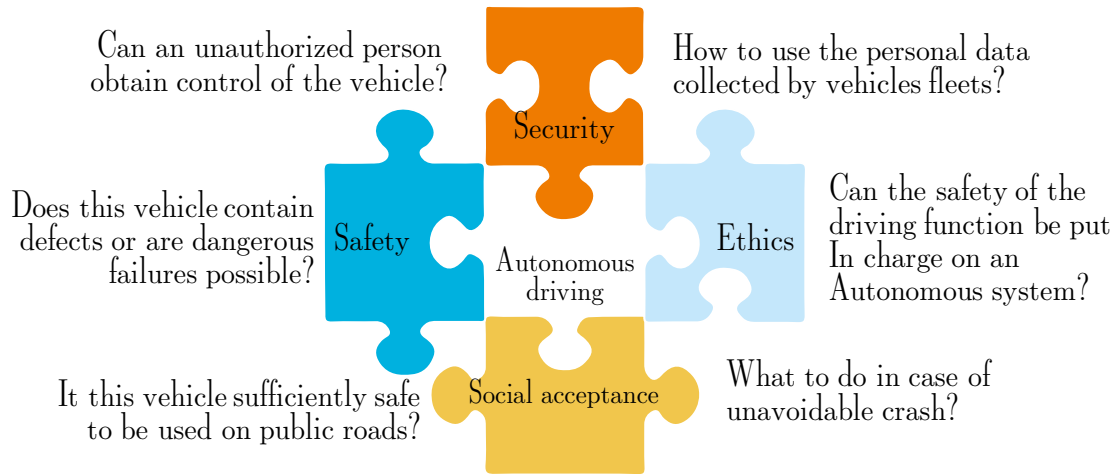


Figure A.1: Common questions on autonomous vehicles development aspects.

The first of these papers [109], have been presented in June 2019. Thanks to properly trained neural networks, it proposes a system able to recognize people's emotions by their facial expressions. The initial idea proposed in this paper was to use the recognized emotions to modify the driving style of the car with the following rules⁴:

- if the passengers experience *sadness* or *fear*, the car should adopt a caring driving style, with lighter accelerations and decelerations during the straights and following paths that minimises the lateral acceleration during the curves;
- if the passengers are *neutral*, the car will adopt a balanced driving style;
- if the passengers on the car are *happy* or *surprised*, the car should adopt a sporty driving style, with faster acceleration and decelerations during the straights and a higher level of lateral accelerations during the curves.

To perform emotion recognition, based on the models developed by the psychologist Paul Ekman (described in section A.1), it was decided to resort on neural networks, since they allow to reach accuracies higher with respect to the ones obtainable with other methods, like support-vector machines or decision trees. Moreover neural networks allow to adopt the machine learning approach, saving development times.

By continuing the analysis, my coauthors and I realized that such a system is not the ideal one, so the activities moved to use it to determine between various

⁴I, and my coauthors, decided for those rules arbitrarily, based on our feelings. Hence, we do not have any demonstrations that these are the best choice

possible calibrations of the autonomous driving algorithms.

The idea is the following: after some suitable neural networks have been chosen and properly trained, it is possible to assess the effects of different *calibrations* on the passengers' feelings considering different *situations* within ordinary driving *scenarios*. To perform emotion recognition, we developed a utility software, called *Emotions Detector* [113], released as open-source on GitHub. To develop this tool are used Java, and the OpenCV, DeepLearning4J (DL4J) [114], and Apache Maven libraries. It can acquire both images from a webcam or frames of a prerecorded video, crop them to the face only, apply the post-processing algorithms needed by the neural network, and run the network on them. At the end of the process, the images themselves and their emotions probability distributions are saved automatically to obtain a test report.

To better understanding the rest of the presentation, it is useful to define these terms:

- *Calibration*: set of parameters that determine the behavior, in terms of the trajectory (acceleration ramps, lateral distances from obstacles, and preferred lateral accelerations) and, in general, all the numerical parameter (not considered in this paper) needed to develop an AV driving algorithm properly.
- *Scenario*: The environment (real or virtual) in which the vehicle's behavior is shown to a person with different calibrations and traffic conditions.
- *Situation*: A combination composed of a calibration, a scenario, and a traffic conditions set, to be shown to testers.

The *situations* can be represented both in simulators and real vehicles. Of course, the use of a real car can give better results, but ensuring the repeatability of the tests requires the use of a closed track and other vehicles for the traffic, making this solution extremely expensive.

The emotions to determine the best calibrations are determined, by the chosen neural network trained for this specific purpose, thanks to volunteers' facial expressions recorded while viewing 3D representations showing the different calibrations. The obtained results have been published in the second paper [110] released in March 2020.

The last paper [111], presented in November 2020, describes an attempt to improve the results in terms of training accuracy already obtained in [110] by increasing the number of databases (and therefore of images available for training). Unfortunately, the addition of these three more databases was not able to improve the already obtained results. However, these data are described in this appendix just in case some researcher wants to try the same path already undertaken in this study.

The software developed to merge (and also perform other operations on) the databases has been released [115] with an open-source license on GitHub.

The activities to implement such a system, in order to also obtain results that can be useful also to other scholars working with emotion recognition from facial expressions, has been slip into these eight steps:

1. finding of a model to describe human emotion, valid for all the cultures;
2. finding of a set of neural networks suitable for emotion recognition;
3. finding of facial expression databases, which images are labeled in order to describe the depicted emotions;
4. develop a tool to simplify the management of the various databases, allowing to merge them to obtain more accurate recognition;
5. train the networks on the found database, taken as single one or merged, to find the best solution;
6. prepare 3D reconstructions of the behaviors generated by the calibrations to be compared on different scenarios;
7. test, thank to pictures acquired by a webcam and a software developed to run the trained neural networks, that they can work in a real scenario;
8. apply the emotion recognition system to chose, between various calibrations, what are the preferred ones.

The rest of the appendix is organized as follows.

Section [A.1](#) describes the state of the emotion recognition by facial expression analysis, thanks to the studies of Prof. P. Ekman and the various databases available in the literature.

Section [A.2](#) describes what the neural networks are, how they work, and provides a vocabulary. Moreover, it describes (subsection [A.2.3](#)) the main performance assessment metrics analyzed in this dissertation to determine the quality of the trained networks and some of the most common performance improvement techniques (subsection [A.2.4](#)).

Section [A.3](#) describes the proposed methodology to recognize the emotions, presenting Facial Expressions Databases Classifier (FEDC), a software tool to prepare the facial expressions databases to be used for neural network training (subsection [A.3.1](#)), and the chosen neural networks (subsection [A.3.2](#)).

Section [A.4](#) presents the technical aspects of the training environment and the training results. Due to the complexity of this section, it contains a description of its own structure at its beginning.

Section [A.5](#) describes the application of the neural network [116] on the database ensemble 1 (see subsection [A.4.2](#)) to chose the best ones between a set of different autonomous driving algorithm calibrations (see the introduction of this appendix).

Section A.6 describes some preliminary tests, performed by my coauthors and myself, to assess the capability on the neural network [116], trained on the FER2013 database [117] with FER+ annotation [118], to recognize emotions.

Finally, section A.7 presents the training results and draws some conclusions on the benchmark application on autonomous driving algorithm assessment application.

A.1 A formal model for the human emotions

As humans, we are instinctively able to determine the emotions that our fellows are feeling. It is well known that facial expressions are a fundamental part of this social ability. In the 1970s, the American psychologist Paul Ekman scientifically studied this phenomenon. In 1972 [119], he published a list containing the six primal emotions he believed are shared among all human groups, independently from their culture. Those are: *anger*, *disgus*, *fear*, *happiness*, *sadness*, and *surprise*. In the following years, he and other researchers added other emotions to this list. For the purposes of this work, and also keeping into account the labels available in the facial expressions databases from the literature, they have been considered only eight basic emotions: the six proposed in 1972 plus *contempt* and *neutrality*. For our automotive application, however, the interest is on recognize only five of them: *fear*, *happiness*, *neutrality*, *sadness*, and *surprise*.

Ekman developed also the *Facial Action Coding System* (FACS) [120]. He observed that facial expressions are performed thanks to facial muscles; hence, they are, from the physical point of view, possible configurations of those when they are moved one by one or in groups. He called these groups of muscular movements as *Action Units* (AUs). Thus, it is possible to classify a facial expression resorting to a weighted evaluation of those AUs. Thanks to these evaluations, it is possible to make facial expression determination more objective. However, to make things even more complex, people can show the same emotion with different AUs. Thus there is a huge intraclass variability. In the available facial expressions databases, when the labeling of the considered facial expression has been performed by analyzing the AUs, the picture is marked as FACS encoded. Furthermore, facial expressions can be posed or spontaneous: while the latter ones are more common to see in everyday life, the formers are a more caricatural, exaggerated version of the same.

A.1.1 Facial expression databases

Various scientists worked on this topic during the years; hence, many pictures of posed and spontaneous facial expressions, organized in databases, are available in the literature. The ten databases selected for this work are:

- The *Extended Cohn–Kanade (CK+) database* [121] [122] contains 593 sequences from 123 subjects portrayed in all eight emotional states considered in this document. Each sequence starts from a neutral state and then gradually reaches the peak of the considered emotion. Overall, 327 of the 593 sequences are FACS coded.
- The *Facial Expression Recognition 2013 (FER2013) Database* [117] is composed of 35,887 pictures of 48×48 pixels retrieved from the Internet. Since the original labeling method has demonstrated itself erroneous in some cases, a newer set of annotations named FER+ [118] was released in 2016. It contains labels for 35,488 images since the remaining 399 do not represent human faces, and it also adds contempt emotion.
- The *Japanese Female Facial Expression (JAFPE) database* [124] contains 213 grayscale photos of posed facial expressions performed by 10 Japanese women. Each image has been rated on six emotional adjectives by 60 Japanese subjects.
- The *Multimedia Understanding Group (MUG) database* [125] contains photos of 86 models posing six emotional states: anger, disgust, fear, happiness, sadness, and surprise. The images of this database are taken inside a photographic studio, thus in controlled illumination conditions.
- The *Radboud Faces Database (RaFD)* [126] is a collection of photos of 67 models, posing all eight emotional states considered in this paper. The database authors took each picture from five different angles simultaneously.
- The *Static Facial Expression in the Wild (SFEW 2.0) database* [127] is composed of frames extracted from different movies depicting people having seven different emotional states: anger, disgust, fear, happiness, neutrality, sadness, and surprise. For this activity, only 1694 labeled aligned images have been used.
- The *FACES database* [128] is a collection of 2052 images taken from 171 actors. They acted two times the following six facial expressions: anger, disgust, fear, happiness, neutrality, and sadness. The actors are further divided into three different age classes.
- Indian Movie Face Database (IMFDB) [129], only used for *Ensemble 3* published in [111].
- NimStim Set Of Facial Expression Database [130], only used for *Ensemble 3* published in [111].

- Real-World Affective Faces Database (RAF-DB) [131] [123], only used for *Ensemble 3* published in [111].

As already said in section 1.2.1, SAE and ECR defined various levels of driving automation. For the sake of this work, are considered only vehicles of at least level 4 of the SAE or *Automated* of the ECR classifications. Various authors studied the interactions between these automations and humans, focusing mainly on how the Advanced Driver Assistance Systems (ADAS) integrated into the car should interact with the driver [132], with a particular focus on the adaptation of the digital cockpit to different driving situations, [133]. Other devices, based also on the analysis of facial expression and yet installed inside cars, are driver fatigue and drowsiness sensors. They work thanks to a sensor for detecting the steering wheel angle, electrocardiogram performed on the steering wheel surface [134], and cameras that, thanks to a computer vision algorithm, can detect the frequency at which the driver blinks [135].

While these applications are applied during the driving, we are interested in the algorithm calibration phase before the vehicle is shipped, especially for the trajectory planning (The reader can find examples of the involved coefficients in [136]). This can help carmakers to choose algorithms and respective calibrations that best suit their customers' expectations. To the best of our knowledge, no author has yet proposed using emotion recognition through computer vision to calibrate autonomous driving algorithms.

A.2 Neural networks

An (artificial) neural network is a mathematical system. The name *neural networks* comes from the conceptual similarity to the biological neural system. From the mathematical point of view, a *neuron* is a mathematical function with a certain number q of *inputs*, u_1, u_2, \dots, u_q , and one *output*, y . Those inputs are linearly combined to determine the activation signal s , by means of the equation $s = \Theta_0 + \Theta_1 u_1 + \dots + \Theta_q u_q$. Θ_0 is usually called the bias parameter. After the sum node, a non-linear function is applied to s , obtaining the output signal $y = \sigma(s)$. $\sigma(s)$ is commonly called *activation function*. Popular activation functions are historically the sigmoidal function and, nowadays, the ELU, ReLU, and LeakyReLU functions.

Various layers of this kind compose a neural network. In the literature, it is possible to find various neural networks designed for multiple purposes.

A.2.1 Generalization capability: underfitting and overfitting

The primary objective of a neural network is to create a model that is able to *generalize*. This goal implies that a good model can work similarly with both already seen and new unseen data. This capability is needed to make the system capable of making classifications and predictions on new data based on what it has *learned* from the training data.

There are two different reasons why the system is unable to achieve *generalization*:

- The model has not *learned* sufficient characteristics from the input data. Hence it will not be able to generalize towards new data. In this situation, the system will *underfit*.
- Conversely, if the model has learned too many features from the training samples, it limits its ability to generalize towards new data. In this case, the system will *overfit*.

Not all the network parameters are chosen during the training. Some of them have to be set before the training or are determined by the neural network structures. These parameters are called *hyperparameters*.

A.2.2 Vocabulary

Before describing the experimental results, it is better to define some terms:

- *Learning rate* defines the update *speed* of the parameters during the training. If it is lower with respect to the ideal one, the learning is slowed down but becomes smoother; on the contrary, if its value is too high, the network can diverge or underfit.
- *Sample* is an element of a database. In our case, it is a picture of a human face with a facial expression properly labeled with the represented emotion.
- *Batch* is a set of N samples processed independently and in parallel. During the training process, a batch corresponds to a single update of the network parameters.
- *Epoch* is usually a passage on the entire dataset and corresponds to a single phase of the training.

A.2.3 Performance assessment metrics

For each experiment, these metrics have been computed:

- *Accuracy* is defined as

$$\alpha = \frac{P_r}{P} \quad (\text{A.1})$$

where P_r is the number of correct predictions and P is the number of total predictions. For this metric, the higher is the better.

- *Loss* represents how bad the model prediction is with respect to a single sample. For this metric, the lower is the better. There are many different methods to compute this parameter in the literature, such as binary cross-entropy, categorical cross-entropy, mean absolute deviation, mean absolute error, mean squared error, Poisson, squared hinge, etc. For our purposes, we chose to compute this metric as a categorical cross-entropy, defined as:

$$L(y, \hat{y}) = \sum_{j=0}^M \sum_{i=0}^N (y_{ij} \cdot \log(\hat{y}_{ij})) \quad (\text{A.2})$$

This loss function must be used for single label categorization, i.e. when only one category is applicable for each data point. It is perfectly suited to our cases, since we formulated the hypothesis that each image (sample) can represent only one of the considered emotions (category).

In particular, the curve composed by the various losses computed in each epoch, called the loss curve in the literature, is important to determine if the model underfits or overfits. If the training dataset loss curve is much greater than the one obtained on the validation dataset, we are into *underfitting* conditions. If the loss curves are near, probably the obtained one is a good model. Finally, if the loss curve of the training dataset is instead much lower than that of the validation dataset, it indicates the presence of *overfitting* [137].

- *Confusion matrix*: Considering that the classification system has been trained to distinguish between eight different emotions, the confusion matrix summarizes the result of the testing of the neural network. It is a particular contingency table in which emotions are listed on both sides. There are the labels of the pictures (ground truths) in the top row while, in the left column, there are the predicted categories (emotions).

A.2.4 Performance improvement techniques

Cross Validation

To reduce *overfitting*, it is possible to adopt the cross-validation technique. It consists of partitioning the dataset into multiple subsets, some of which are used

for training and the remaining for validation/testing purposes. In the literature are described various kinds of techniques, such as Leave-One-Out Cross-Validation (LOOCV), k -Fold, Stratified, and Time-Series. Stratified is used when dealing with binary classification problems, while Time-Series is used when the dataset is composed of observations made at different times; hence, these two are not suitable for our purposes.

For this work, LOOCV or k -Fold can be chosen. It has been decided to chose the latter by putting $k = 9$. In the k -Fold, the dataset is split into k folds (subsets) of approximately the same size: $k - 1$ folds are used for training, while the remaining one is used for validation or test. The database has been divided into two subsets thanks to the FEDC database subdivision function. The first one, which contains 90% of the images, was used for training and validation, while the other one, composed of the remaining 10% of the images, was used to perform the test.

Before the training, the first subset was split into nine smaller subsets: eight were used for training, while the remaining one was used for validation. Changing the validation subset after each training made it possible to perform nine different neural network training to pick the one that performed better.

Data Augmentation

Data augmentation is adopted when a low number of samples is available. The idea is to modify the samples in different ways in order to increase their number artificially. For example, in the case of images, the augmented ones can be obtained by rotating, reflecting, applying translations, and so on. This technique improves the generalization capability of the network without modifying the model.

In all the training, these data augmentations have been applied:

- brightness range from 50% to 100%;
- random horizontal flip enabled;
- rotation interval between ± 2.5 deg;
- shear range of $\pm 2.5\%$;
- width and height shift range of 2.5%;
- zoom transformation interval between $\pm 2.5\%$.

Normalization

To improve the training process, have been applied, alternately, two different normalizations to the grayscale space of the images: $[0, 1]$ and *z-score normalization*.

The [0,1] normalization is a particular case of the *scaling to range normalization*, defined generally by the formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (\text{A.3})$$

in which x_{min} is set to 0 and x_{max} is set to 1. The *z-score normalization*, sometimes called *standardization*, is used to obtain a distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$. The applied formula is:

$$x_z = \frac{x - \mu}{\sigma} \quad (\text{A.4})$$

in which x represent the 8-bit brightness value of the images, x_{min} and x_{max} are, respectively, the minimum and the maximum brightness within the images, μ is the arithmetic average of the brightness of all the pixels of the images, and σ its standard deviation.

A.3 Recognition of emotions from facial expression

As described in section A.1 is possible to determine people's emotions by their facial expressions. Since it is not possible to write "by hand" a software function to analyze the pictures of the passengers' faces and determine their emotions with a good performance, it has been adopted a machine learning approach: thanks to a properly trained neural network, it will be possible to solve this challenge. From the operative point of view, it has been decided to divide the development of this proof-of-concept calibration system into three different phases:

1. Development of a tool, called *Facial Expressions Databases Classifier* (FEDC) [115], to perform different operations on the selected facial expressions databases in order to prepare them for the training of the neural networks. FEDC can also be used to make the supported databases homogeneous so that they can be merged to train the neural networks. These derived datasets are referred to in the rest of this dissertation as *database ensembles* (DE).
2. Choice of the most suitable neural networks available from the literature, and trained them with single databases and some database ensembles to compare them through objective metrics defined in the following.
3. Creation of 3D graphics reconstructed scenarios depicting some driving situations with different calibrations of the autonomous driving algorithm. By showing them to testers and analyzing their facial expressions during the representations, it is possible to determine the calibrations preferred by passengers.

A.3.1 Facial Expressions Databases Classifier

The purpose of FEDC is to simplify the merging of facial expression databases to train the neural network.

More technically, the tool takes the images from the database in the format which it is provided by the databases' editors, creates as many folders as the number of emotions present in the chosen database, and moves the images in the corresponding folder. After that, the selected post-processing operations can be applied to the images.

To simplify operations, FEDC provides an easy to use Graphical User Interface (GUI) that allows the operator to select the database he/she wants to classify, the output directory, and some post-processing options he/she wants to apply to the images, displaying the current operation with a progress bar and an informative label.

Partitioning into datasets

To properly train a neural network, it is a good practice to divide the databases into three smaller datasets:

- the *training* one is used to train the network effectively;
- the *validation* one is used to evaluate the neural network performances during the training.
- the *test* one is used to test the capability of the neural network to generalize by using different samples from the ones involved in the training.

FEDC can create, when its subdivision option is enabled, the train, test, and optionally the validation folders. Each of these contains subfolders holding in the related images of every emotion of the selected database. The user can choose the percentage distribution between the three subdivisions.

Post-processing (performance enhancement) features

The most recent version of FEDC (5.0.0) can also perform the following post-processing operations on the images:

- change image format;
- conversion in grayscale color space;
- histogram equalization (normal or CLAHE);
- face detection to crop the images to face only;

- scaling of horizontal and vertical resolutions;
- subdivision in train, validation and test dataset;
- transformation of rectangular images into square ones (can be activated only if the *Face Detection and Crop* option is not selected).

A.3.2 Choice of the neural networks

It is crucial to choose the best neural networks explicitly made for facial expression recognition to obtain effective results. They have been chosen these two state-of-the-art networks, designed with different approaches.

The one proposed by Ferreira [116] was published in 2018, and it is a deep network that is relatively complex and has a default image size of 120x120 pixels.

The other one, proposed by Miao [140], was published in 2019, and it is a shallow neural network that is much simpler and has a default image resolution of 48x48 pixels.

Both the networks, in their default configurations, have about 9 million parameters, but by setting a resolution of images of 48×48 pixels for the network proposed by Ferreira [116], it is possible to reduce its parameters to about 2 million. This reduction allows performing emotion recognition on single board computers, opening the door to lowering the cost of these tests. In this way, it is possible to run the tests in real-time on autonomous vehicle prototypes. This aspect is essential from the privacy protection point of view since it allows to run the tests without storing face images; hence this algorithm can be deployed without asking for permissions from the depicted people, increasing the number of testers.

A.4 Neural Networks Training

An accurate description of the neural networks training activities, which have been already described in the paper [110] is fundamental since the repeatability of the training, alongside the evaluation of their accuracies is a crucial point to achieve objective emotion detections and hence to properly classify the passengers' feeling, needed to determine which are the best calibrations.

The rest of this section is composed as follows.

The subsection [A.4.1](#) describes the implemented set-up to train the neural networks.

The subsection [A.4.2](#) describes the trainings, including results from the papers [110] and [111]. The neural networks' performances are assessed by the performance metrics described in section [A.2](#).

Some of the performed training are described in detail.

The subsection [A.4.2](#) describes the results obtained training the network [116] on

the CK+ database, the results obtained from the networks [116] and [140] trained on the FER2013 database, and the performances achieved by training both the networks [116] and [140] on the *database ensembles 1 and 2*. It also summarizes all the training results from the paper [110].

Moreover, section A.4.3, thanks to the three databases added to FEDC in its version 5.0.0, released simultaneously to the paper [111], describes the results obtained in the paper [111] training the network [116] on the database ensemble 3 and on the IMFDB [129].

For the reader's convenience, the results presented in both the papers [110] and [111] are summarized in the subsection A.7.1 of the conclusion section of this appendix.

A.4.1 Training Environment Set-Up

The environment described in the following has been adopted to train the networks which performances are described in [110] and [111].

Keras [141] has been adopted as a high-level abstraction API because it is simple to use and, for some years now, it has been one of the most widely used solutions for neural networks training. It can abstract three different frameworks for machine learning: TensorFlow [142], Microsoft Cognitive Toolkit (CNTK) [143], and Theano [144]. All three proposed solutions adopt an open source-like license. For the sake of this work, TensorFlow have been chosen.

Other utility libraries, adopted to speed-up the code writing and to improve the presentation of the experimental results, are:

- *Matplotlib* [145], a 2D plotting library for Python;
- *NumPy* [146], a package for scientific computing for Python;
- *Open Source Computer Vision Library* (OpenCV) [147], a computer vision and machine learning software library for C++, Java, MATLAB, and Python;
- *Pandas* [148], which provides high-performance, easy-to-use data structures and data analysis tools for Python; and
- *Scikit-learn* [139], a library for data mining and data analysis.

The networks have been implemented within the Keras environment. The reader can find the code implemented to train the network at [149].

For the network in [140], there were no problems, while, for the network in [116], it has been faced an ambiguity in the "e-block" layer because, in the paper, it is not clearly described how to implement the relative "crop and resize" operation. To solve this ambiguity, it has been decided to implement it as a single convolutional layer, in which the kernel size is defined according to the resolution of the input

images. For 120×120 pixels images, which is the default input size for the network, the kernel size is 106×106 , while, for 48×48 pixels images, which is the size of the picture of the FER2013 database, the kernel size is 43×43 . In both cases, the number of output filters has been set to 64 in order to make the next multiplication operation possible.

A.4.2 Training Results from [110]

For the training of the aforementioned neural networks (see Section A.3.2), the following databases are chosen in order to compare the results obtained with the yet developed implementation with those obtained by neural networks authors:

- CK+, which was used only for the network in [116] because it was not used by the authors of [140];
- FER2013.

Moreover, also the following two *database ensembles* have been prepared recurring to FEDC:

- *Ensemble 1*, composed of all the labeled images from all the databases supported by FEDC; and
- *Ensemble 2*, composed of all the posed facial expressions images from the databases CK+, FACES, JAFFE, MUG, and RaFD.

The trainings have been performed in 23 different configurations. Table A.1 indicates the number of pictures for each emotion that can be found in the chosen databases.

The networks have been trained with these datasets:

- CK+ database [121] [122] (only for the network in [116]);
- FER2013 [117] [118];
- *Ensemble 1*;
- *Ensemble 2*.

For each training, the "EarlyStopping" callback was used to stop the training if there was no improvement in the loss curve computed on the validation dataset after 18 consecutive epochs. In some training, the callback "ReduceLROnPlateau" has been set to multiply the learning rate by 0.99 or 0.95 in every epoch.

In this dissertation have been reported only the most interesting cases, that the ones also said in the [110]. The reader can find the other cases in [112]. The selected cases are indicated, in Table 3, in bold: for each of them are reported: its accuracy graph, its loss graph, and its confusion matrix.

The choosed hyperparameters are:

Emotion	<i>Ensemble 1</i>	<i>Ensemble 2</i>	CK+	FER2013
Anger	4328	981	45	4953 3111 ^a
Contempt	788	572	18	0 216 ^a
Disgust	1340	1022	59	547 248 ^a
Fear	1887	950	25	5121 819 ^a
Happiness	10,676	1089	69	8989 9355 ^a
Neutrality	14,196	1070	123	6198 12,906 ^a
Sadness	5524	953	28	6077 4371 ^a
Surprise	5254	656	83	4002 4462 ^a

^a FER+ annotations.

Table A.1: Picture available for each emotion in the chosen databases. Table from [110].

- batch size: 100 (except for the network in [116] trained on the CK+ database, where it was set to 50);
- maximum number of epochs: 1000;
- learning rate: 0.001.

CK+ Database

The first training of the network in [116] was performed on the CK+ database, resized to a resolution of 120×120 pixels. With the previously said division, the dataset was split in this way: 364 images were used for training, 46 images were used for validation, and 40 images were used for testing.

The obtained results are shown in figures A.2, A.3, and A.4. These are in line with the one presented in [116], thus the implementation used to get the results described in this dissertation seems to work correctly.

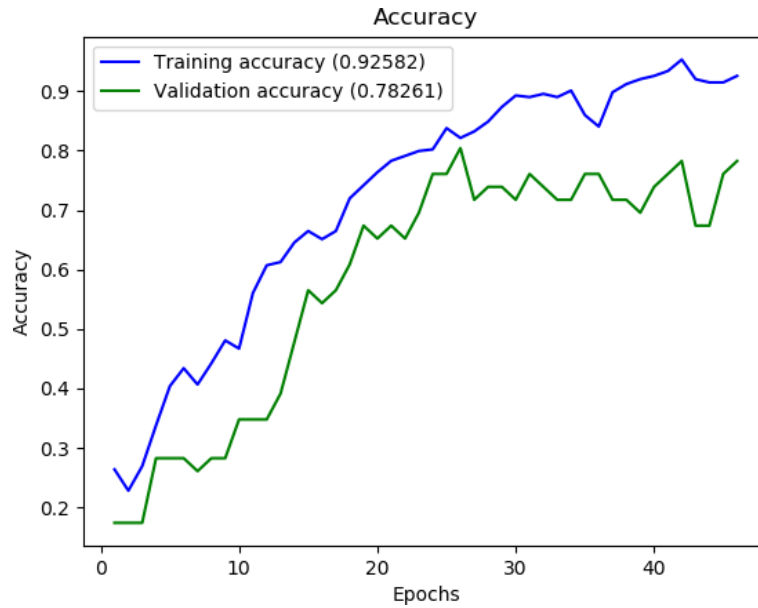


Figure A.2: Accuracy graph of the network in [116], trained with the CK+ database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

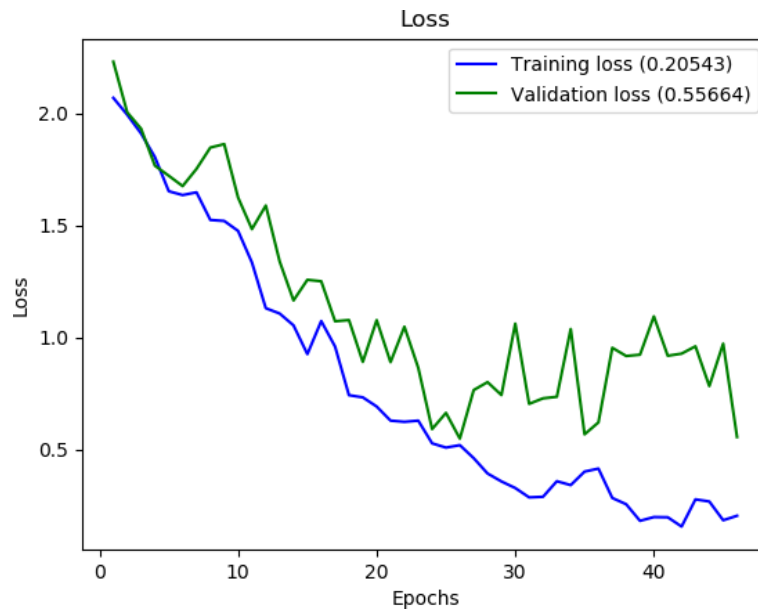


Figure A.3: Loss graph of the network in [116], trained with the CK+ database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

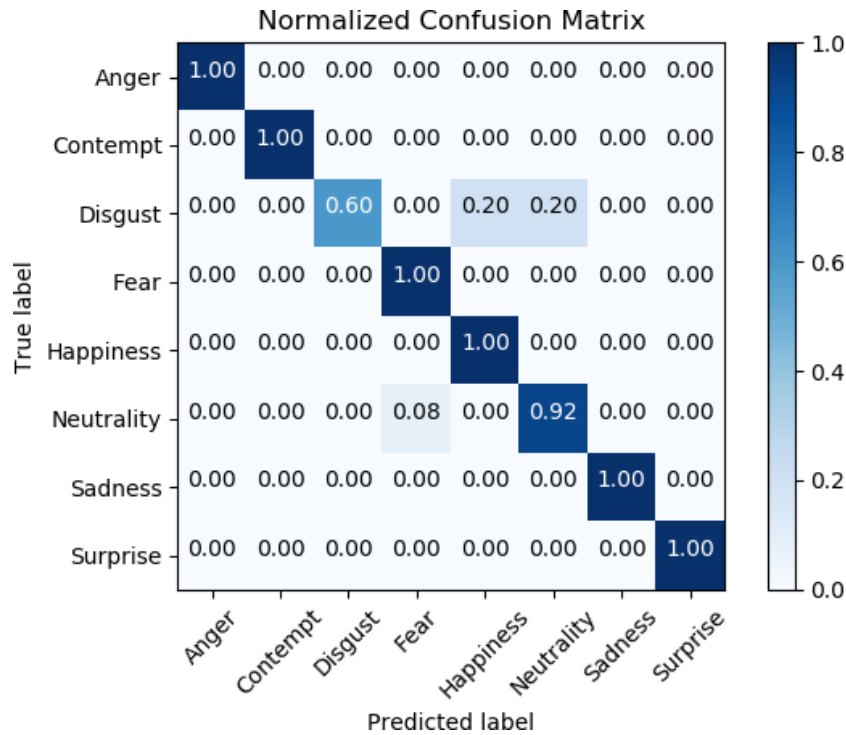


Figure A.4: Normalized confusion matrix of the network in [116], trained with the CK+ database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

FER2013 Database

The FER2013 [117] database has a huge number of pictures, but the resolution of the images is limited to 48×48 pixels. Instead of performing an upscaling of these pictures, we decided to modify the network [116], as described previously, to work with these low-resolution images.

The settings and hyperparameters are the same adopted from the CK+ database, except for the batch size that has been increased to 100. In this way have been obtained 28712 images for training, 3590 for validation, and 3585 for testing.

With this database, they have been obtained accuracies around 60%: a not impressive result, surely improvable but also undermined from the sometimes dubious labels and to the presence, in the database, of some images that do not represent human faces. Thus, we decided to use the FER+ [118] annotations, which allowed us to remove erroneous images hence improving the ground truth.

The best results in terms of test accuracy on this database were obtained from the network in [116], and are shown in figures A.5, A.6, and A.7.

As shown by the confusion matrix (see Figure A.7), the trained network is quite good at detecting happiness, neutrality, and surprise, while it is weak at detecting

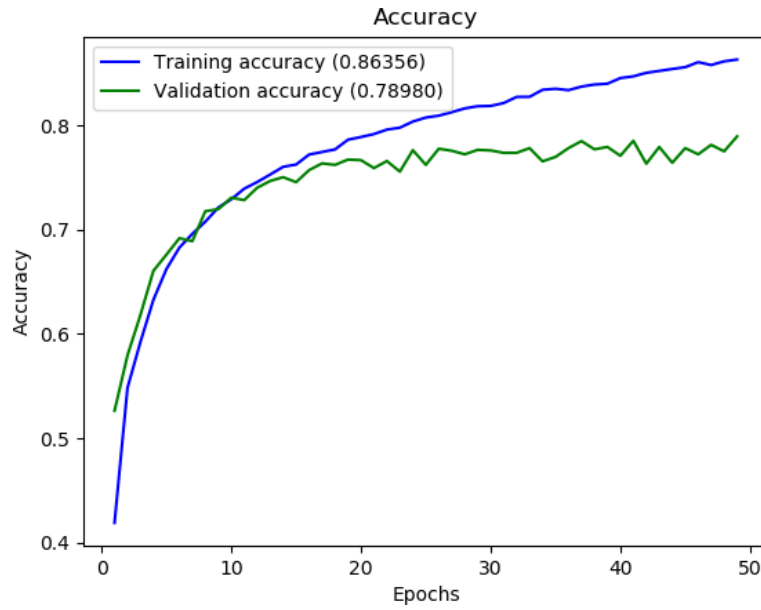


Figure A.5: Accuracy graph of the network in [116], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

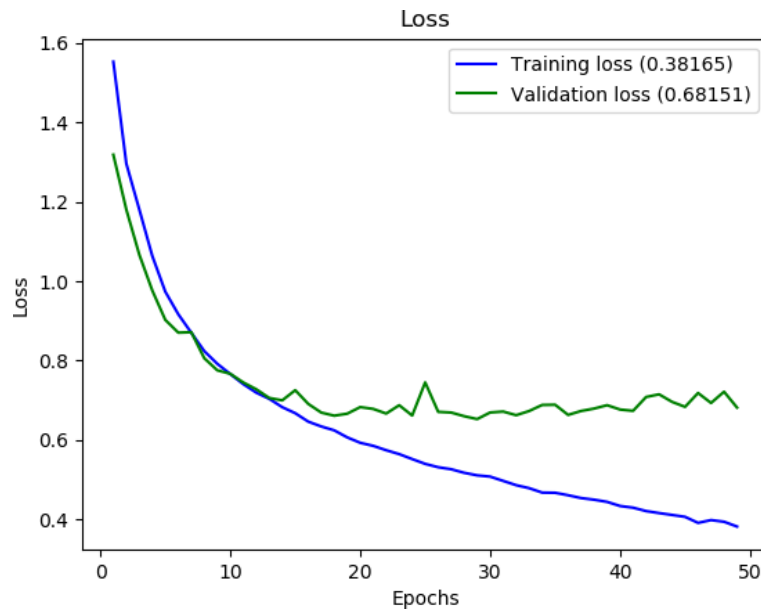


Figure A.6: Loss graph of the network in [116], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

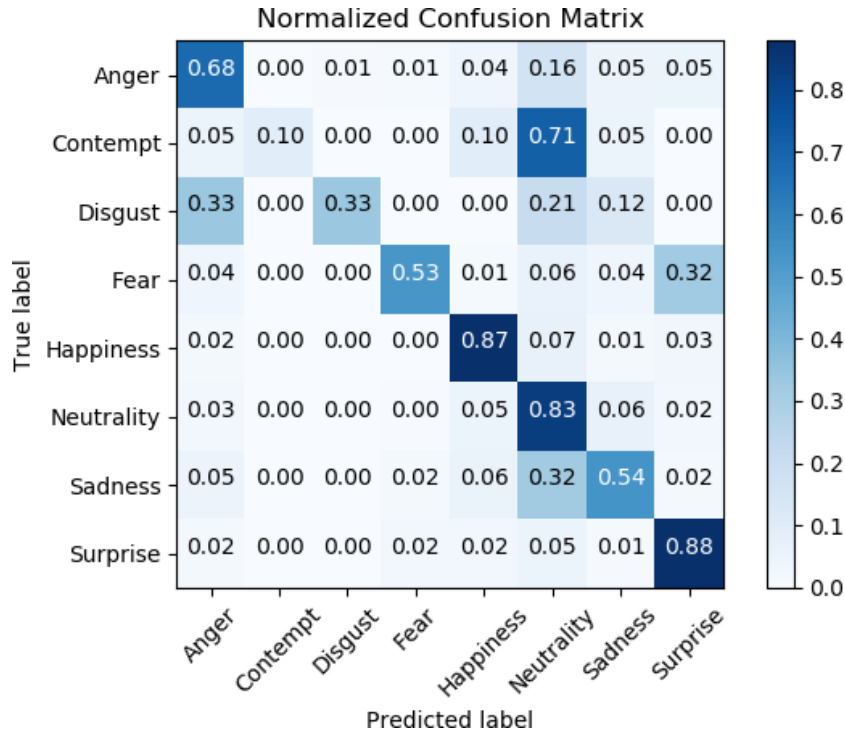


Figure A.7: Normalized confusion matrix of the network in [116], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

fear and sadness. We also have poor performance in recognizing contempt and disgust, but these emotions are not crucial for our purposes. Since FER2013 is known to be a not well-balanced database, and considering that also the network in [140], trained with the same settings and on the same databases, presents a similar confusion matrix (see fig.A.8), a possible explanation is that the FER2013 database does not provide good examples for contempt, disgust, and, more important for our application, fear, and sadness classes.

Database Ensembles

After these preliminary results, the neural networks have been trained using two different *database ensembles*: one containing the images, posed and spontaneous, of all the databases supported by FEDC and one containing only the posed ones. These ensembles were obtained using FEDC, applying a conversion to the grayscale color space and a face detection algorithm in order to crop the images to show only the human faces. Both were created by downscaling all the images to 48×48 pixels to adapt them to the resolution of the FER2013 database and to be able to compare the results of the two databases placed under the same conditions. For the

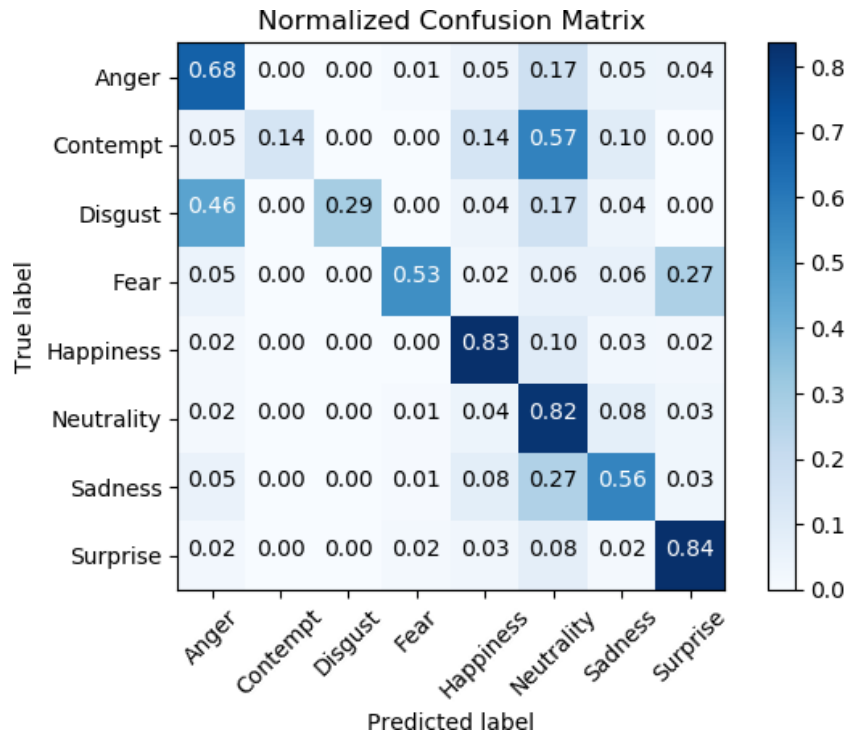


Figure A.8: Normalized confusion matrix of the network in [140], trained with the FER2013 database with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

FER2013 database have been used the FER+ annotations because the improvement in accuracy due to their use is relevant.

The *Ensemble 1* database is composed of all the available images from the database supported, for now, by FEDC. Using the same subdivision procedure used in the previous examples, they have been obtained 35212 images for training, 4402 for validation, and 4379 for testing. Results are shown in figures A.9, A.10, and A.11.

The obtained results are better, in terms of classification errors, than those obtained using the databases individually, especially for the contempt and disgust classes, which had accuracies similar to random ones.

The *Ensemble 2* database is a subset of *Ensemble 1* composed only of posed images. Thanks to the FEDC subdivision procedure, they have been obtained 5847 images for training, 731 for validation, and 715 for testing.

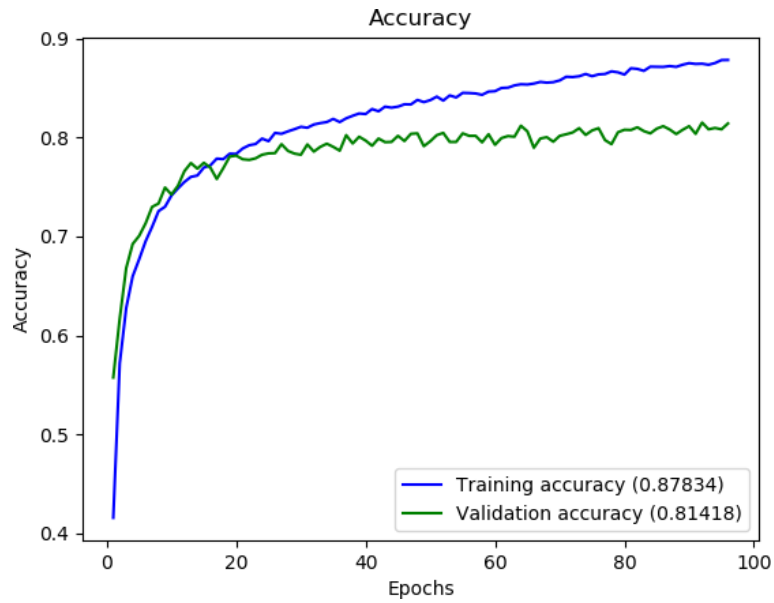


Figure A.9: Accuracy graph of the network in [116], trained with the *Ensemble 1* database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

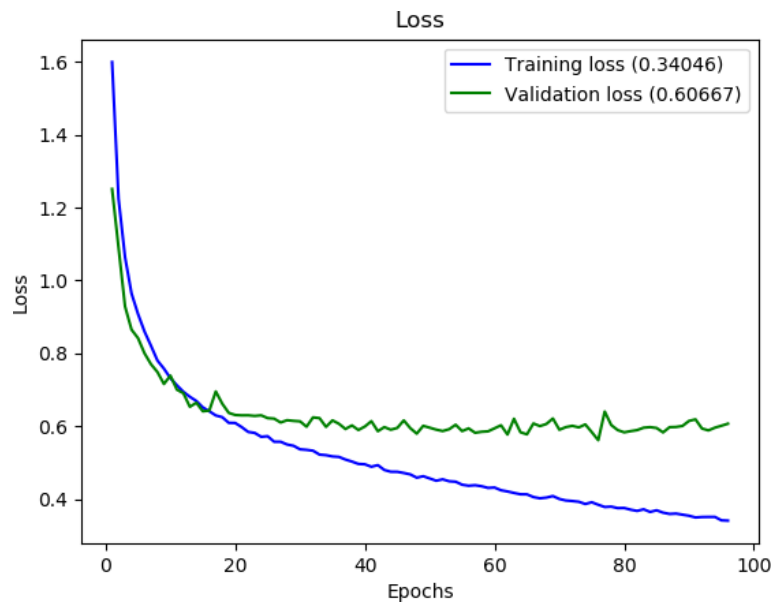


Figure A.10: Loss graph of the network in [116], trained with the *Ensemble 1* using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

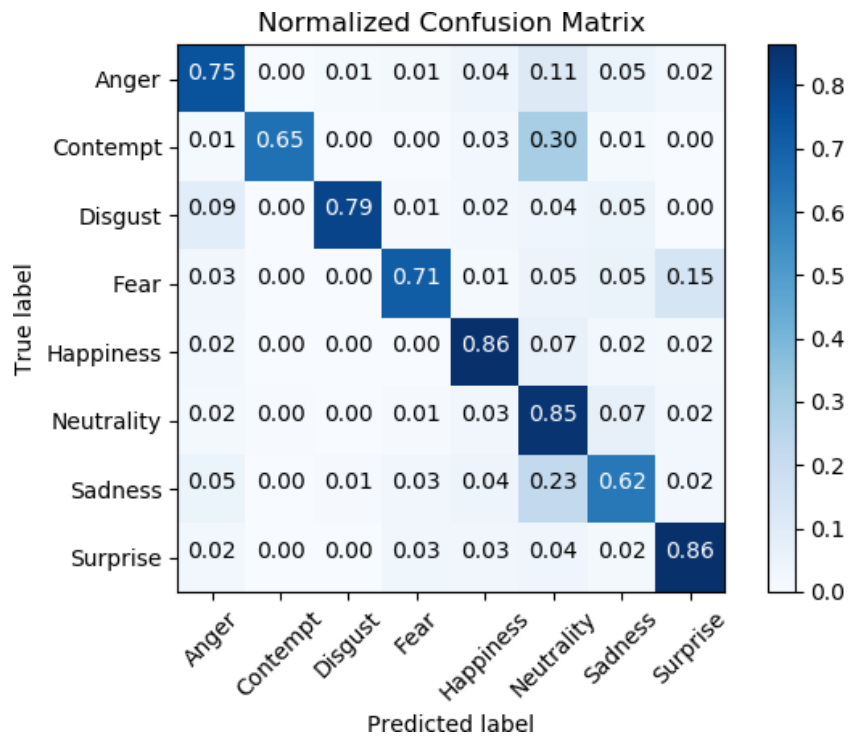


Figure A.11: Normalized confusion matrix of the network in [116], trained with the *Ensemble 1* database using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [112].

Summary of the results

For reader convenience, the training results obtained in [110] have been summarized into two tables. Table A.2 contains the numbers of photos for training, validation, and test datasets, while table A.3 contains the best training accuracies obtained for each database and neural network couple. The test accuracies of the cases shown in detail in the paper are in bold. In general, the network described in [116] requires more time for training but has slightly better performance and, with the same image size, requires fewer parameters than the one in [140]. Thus, if my coauthors and I had to choose a network, we would certainly pick the first one. Before continuing, it is essential to make an observation: even if the test accuracies of the training made with the *Ensemble 2* database are better with respect to the ones obtained with *Ensemble 1*, it is reasonable to expect a better result, on the field, from the networks trained with the latter. This is because the spontaneous expressions are those that we can observe most commonly in everyday life, while those posed are more caricatural with respect to the spontaneous and deliberately exaggerated: this makes the interclass difference more significant, but, at the same time, a network that is trained with these posed images will inevitably experience a bias between the photos on which it is trained and those in which a prediction will actually be requested.

	<i>Ensemble 1</i>	<i>Ensemble 2</i>	CK+	FER2013
Train	35212	5847	364	28712
Validation	4402	731	46	3590
Test	4379	715	40	3585

Table A.2: Subdivision of the databases. Table from [110].

	<i>Ensemble 1</i>	<i>Ensemble 2</i>	CK+	FER2013
[116]	74.79 %	94.69 %	82.50 %	56.57 %
	78.85 ^b %	97.20 ^b %	92.50 ^b %	61.40 ^b %
	80.38 ^c %	97.34 ^c %	92.50 ^c %	62.46 ^c %
				78.36 ^{a,c} %
[140]	71.39 %	92.59 %		54.03 %
	76.91 ^b %	94.83 ^b %	N.A.	61.67 ^b %
	78.47 ^c %	96.78 ^c %		62.71 ^c %
				76.67 ^{a,c} %

^a FER+ annotations. ^b [0,1] normalization. ^c z-score normalization.

Table A.3: Test accuracies summary table (best values). Table from [110].

A.4.3 Training results from [111]

Thanks to the last version of FEDC, in which has been added support for another three databases, Indian Movie Face Database (IMFDB) [129], NimStim Set Of Facial Expressions Database [130], and Real-World Affective Faces Database (RAF-DB) [131] [123] with respect to the version available during the preparation of the *Databases Ensembles* used in [110] and presented in section A.4.

So, to obtain the experimental results described in [111], all the databases yet used in [110] plus these three new ones are merged. The *database ensemble* obtained in this way is composed of 90273 photos, doubling their number with respect to the version presented in [110].

Unfortunately, also in this new *database ensemble 3*, as reported in table A.4, the number of images available for each class is very different: this is a common problem, which often has repercussions on the accuracy that the neural network is able to achieve with classes that have few samples available.

Training on the databases ensemble composed of all the ten databases

Using the same settings reported above, the neural network proposed by Ferreira [116] have been trained on the database ensemble composed of all the ten databases. The maximum obtained test accuracy is of 70.162%, a disappointing value, about ten percent lower with respect to the 80.38% obtained on the *Database Ensemble 1* described in [110] and reported in the section A.4.2 and in table A.3 of this dissertation. The reader can check the obtained results in figures A.12, A.13, and A.14.

Unfortunately, in the IMFDB database has some problems:

Emotion	<i>Ensemble 3</i>	IMFDB
Anger	8065	2688
Contempt	837	0
Disgust	6266	3885
Fear	3016	595
Happiness	24803	8001
Neutrality	26940	9318
Sadness	11601	3435
Surprise	8745	1701

Table A.4: Picture available for each emotion in the databases trained in [111].

- the images distribution with respect to the considered emotions is very different (see table A.4);
- some labels are not correct.
- the images have different resolutions.
- some of the depicted faces are too rotated to be labeled with a sufficient accuracy and to be used for facial expressions recognition.

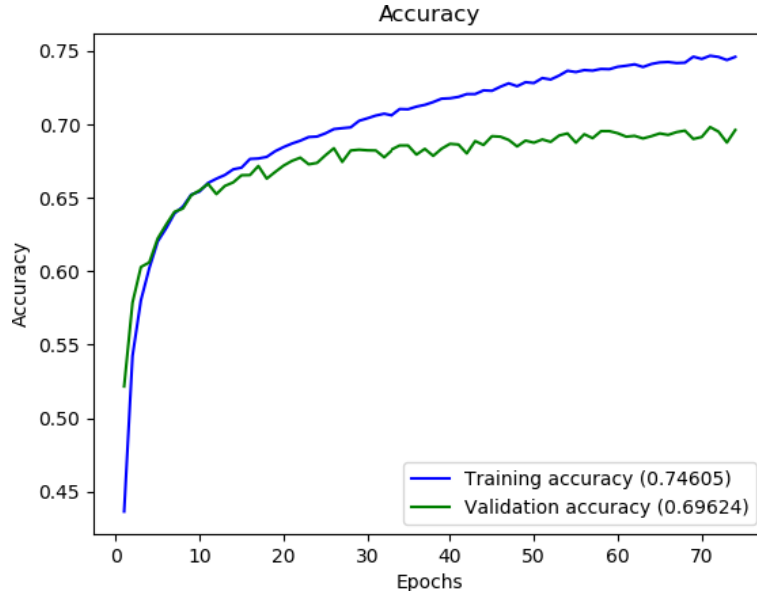


Figure A.12: Accuracy graph of the network proposed in [116], trained with the *database ensemble 3* with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].

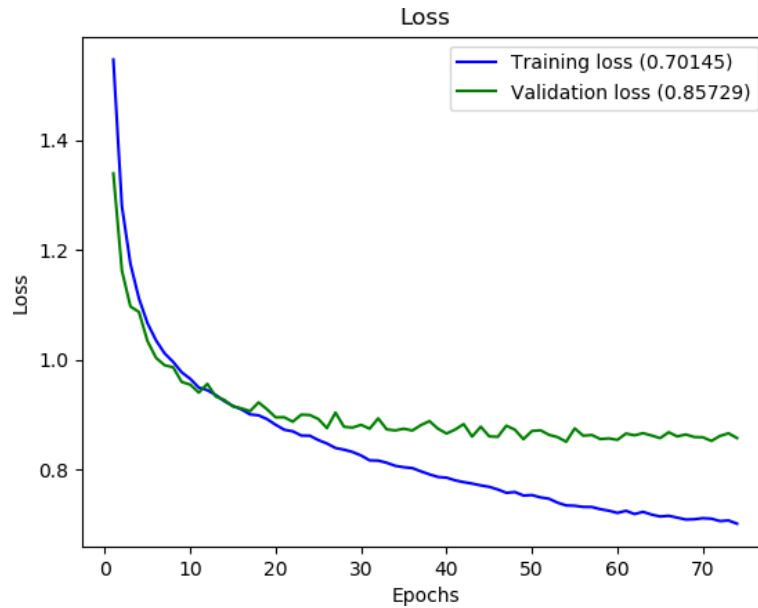


Figure A.13: Loss graph of the network proposed in [116], trained with the *database ensemble 3* with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].

From the confusion matrix shown in fig.A.14 it is possible to see that there is no correlation between the number of images available per class and the relative prediction performance. This counterintuitive result prompted us to analyze the content of the added databases better. As also described in [150], we noticed that the IMFDB has several problems, as described in A.4.3.

Training on the IMFDB database

The neural network proposed by Ferreira [116] have been trained on the IMFDB [129] database: the best fold obtained a test accuracy of 47.908%. This result is 3% better with respect to the one obtained in [150] by training on the plain IMFDB [129] database but surely disappointing, confirming the observation already did in section A.4.3. The reader can check the obtained results in figures A.15, A.16, and A.17.

Among the obtained result, as observed by my coauthor A.C. Marceddu, *the most impressive one is related to the case of fear, in which the neural network has achieved zero precision: therefore, the inadequacy of the images used for the training is evident, since the neural network has not been able to extract essential features for their recognition. Curiously, it can also be noted that the network did not mistakenly exchange the other classes for this one* [111].

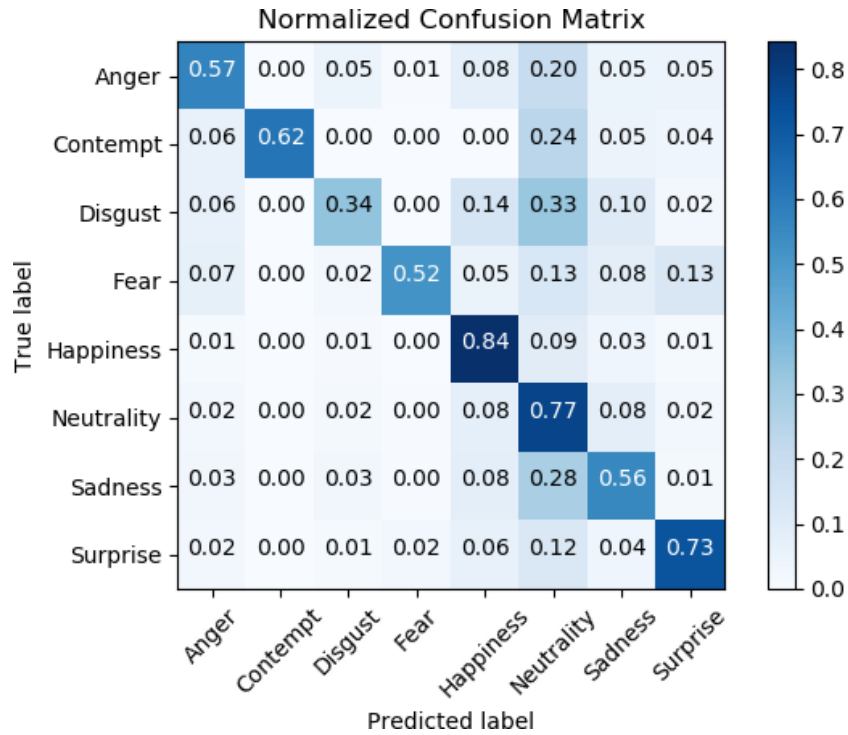


Figure A.14: Normalized confusion matrix of the network proposed in [116], trained with the *database ensemble 3* with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].

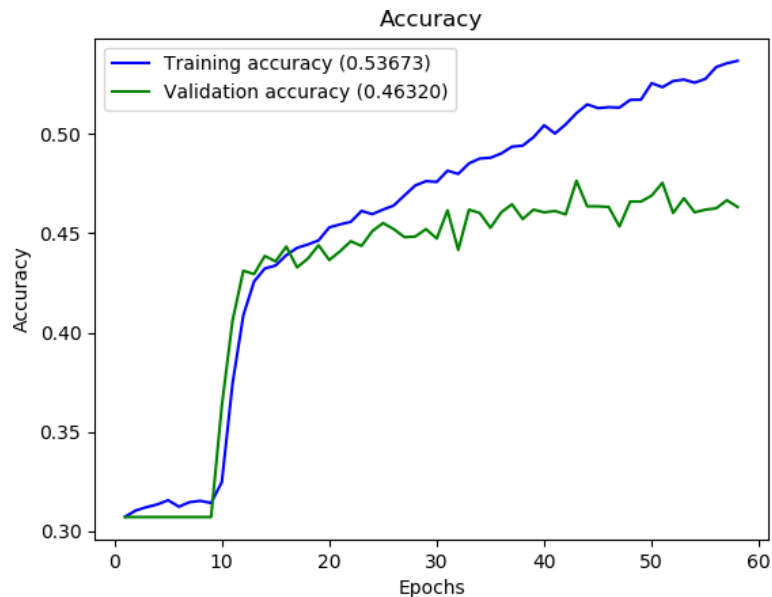


Figure A.15: Accuracy graph of the network proposed in [116], trained with the *database ensemble 3* with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].

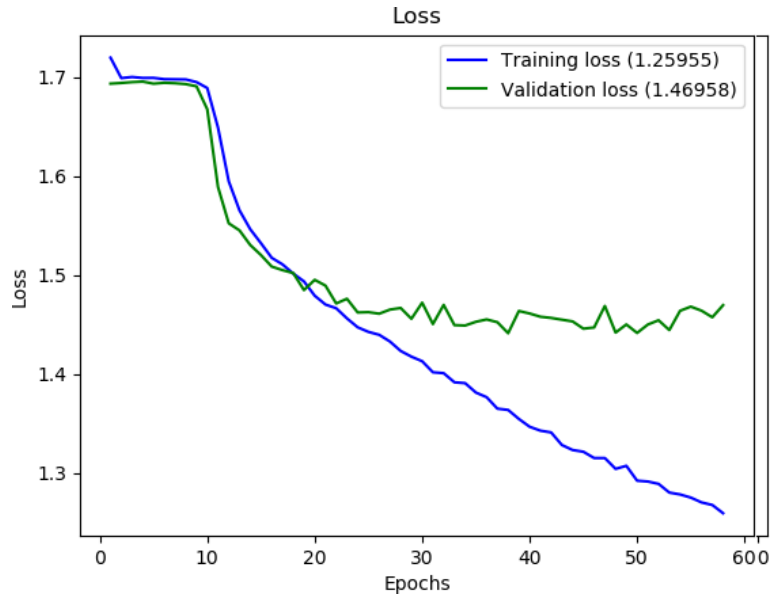


Figure A.16: Loss graph of the network proposed in [116], trained with the *database ensemble 3* with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].

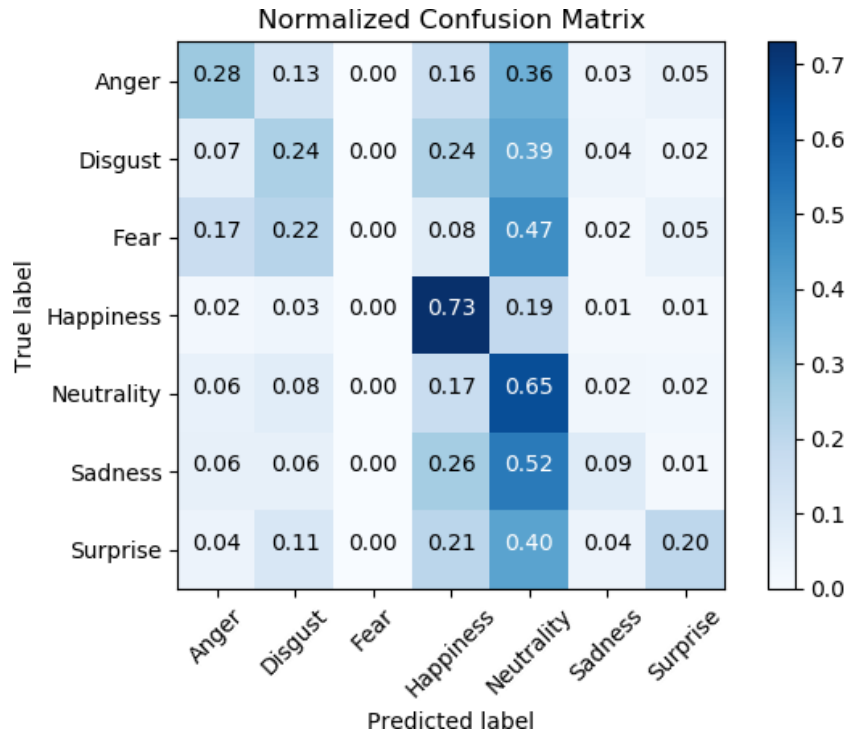


Figure A.17: Normalized confusion matrix of the network proposed in [116], trained with the *database ensemble 3* with the FER+ annotations using the data augmentation (see section A.2.4) and the z-score normalization. Figure from [111].

A.5 Autonomous driving algorithms assessment from [110].

A.5.1 Situations Preparation

In the paper [110] they have been proposed five different *calibrations* of an autonomous driving algorithm behaving in two different *scenarios*.

By combining those *calibrations* and *scenarios*, 11 benchmark *situations* have been prepared.

The first six of them (identified in the following as C_n) involve as *scenario* a curve to the right in a suburban environment. The car can face it with three different *calibrations*, hence following three different trajectories: strictly keeping the right side (*situations* C1 and C4), keeping the center (*situations* C2 and C5) of the lane, or widening at the entrance of the curve to decrease the lateral accelerations (*situations* C3 and C6). Since the vehicle remains within its lane in all these cases, all these behaviors are allowed by most road regulations.

The other five, instead (identified in the following as T_n), have as *scenario* a right turn inside an urban environment. In the road just taken, there is an obstacle that obstructs the rightmost lane. The road has two lanes for each direction of travel. With the first *calibration* (*situations* T1, T2, and T3), the car tries to stay at the right with much decision; therefore, it suddenly discards the obstacle. With the second *calibration* (*situations* T4 and T5), instead, the car decides to widen the turn in advance and to move to the right lane only after it passed the obstacle.

A.5.2 Criteria for Emotion Analysis

As described in the introduction of this appendix, it have been prepared a 3D representation for each of the considered *situations*.

The most relevant emotion, when different from neutrality or sadness, was taken into consideration. *Fear* and *surprise* are considered as negative emotions, while *happiness* as a positive one. *Sadness* and *neutrality* have been considered as intermediate values since the network appears to little appreciate the differences between these moods. In any case, if there was no other emotion than *neutrality* or *sadness*, the one with a greater number of *sadness* outcomes was considered worse with respect to ones that score more *neutrality* outcomes. Since the neural networks can recognize also *anger*, *contempt*, and *disgust*, those outcomes are considered as *experiment failures* since not the expected to be obtained in this kind of tests.

A.5.3 Experimental Campaign

Eight people, called in the following *testers*, who are six males and two females, average ages 25 years, interval 23–31 years, have been asked, in February 2020, to watch 3D reconstructions of these *situations*, starting from a black screen and without describing what they would see to not interfere with their moods. Their emotions have been detected every 2 s.

In the 3D reconstructions, the *situations* are shown in the order: *T2-T4-T3-T1-T5-C1-C5-C2-C6-C3-C4*. It has been chosen to not mix the urban (T) and suburban (C) *scenarios* to not break the immersion in the environment. In the urban *scenario*, those *situations* that are expected to provoke greater emotional reactions are placed the in the middle of the representation, while, in the suburban one, the reconstruction starts from the softer one moving, at the end, to the most critical one.

For the tests, it has been used a flat projection screen to allow to choose the point of view, avoiding, in this way, that the tester could not be able to see the critical moments represented. Using a virtual reality set could improve the environment immersion, but since the adopted emotion recognition technique requires seeing the entire face, using a device of this kind is not possible.

A.5.4 Results Discussion

The experimental results, shown in table A.5, demonstrate that *situations* T2 and C6 are the most stressful from the passengers' points of view. In the urban *scenario*, there are some positive reactions to the *situation* T3, probably due to the capability of the vehicle to make the safest decision by keeping in the right lane and stopping in front of the obstacle. In addition, the *situation* T4, which is the one that minimizes the lateral movement of the car, is appreciated. With traffic, the *calibrations* shown in the *situations* T1 and T5 appears to be equivalent. Regarding the curve *scenario*, the *calibration* shown in situations C3 and C6 is preferred when there is no traffic from the other direction (*situation* C3). Oppositely, for the *calibration* where the car stays at the right side of its lane (C1 and C4), it is preferred the *situation* C4 in which there is traffic in the other direction. The *calibration* shown in C2 and C5 are not appreciated: in our opinion, this is due to the unnatural path that follows the centerline of the lane.

The *scenarios* are the following:

- *T1*: The vehicle takes the right-turn curve in a step way, staying at as possible at the right of the street. The car discards the obstacle when it is really close to keep the right rigorously, then re-enters the rightmost lane immediately after it. No other traffic.

- *T2*: Same *situation* as in *T1*, but with incoming traffic from the opposite direction.
- *T3*: Same *situation* as in *T1*, but traffic from the same direction of the passenger's vehicle supersedes it, preventing the algorithm to move around the obstacle. The vehicle stops in front of it, then starts again moving around the obstacle.
- *T4*: The vehicles take the right-turn curve entering in the left lane, super-seed the obstacle then move to the right lane.
- *T5*: Same *situation* as *T4*, but with incoming traffic from the opposite direction.
- *C1*: The vehicle runs through the curve keeping strictly along the right edge of the road. No other vehicle comes from the other direction of travel.
- *C2*: The vehicle travels the curve keeping in the center of its lane. No other vehicle comes from the other direction of travel.
- *C3*: The vehicle travels the curve widening to the left at the entrance, then closing it to the right, to reduce lateral accelerations.
- *C4*: Same *situation* as in *C1*, but with traffic from the opposite direction.
- *C5*: Same *situation* as in *C2*, but with traffic from the opposite direction.
- *C6*: Same *situation* as in *C3*, but with traffic from the opposite direction.

These preliminary results agree with the experiences reported by the testers when they were interviewed after the tests. In particular, asking about the *situations* C3 and C6, it emerged that the C3 one, in which the curve is traveled keeping the left side of the lane, is more appreciated without traffic in the opposite direction. Instead, following the same trajectory with traffic, as in the *situation* C6, causes inconveniences to the passengers.

	T1	T2	T3	T4	T5	C1	C2	C3	C4	C5	C6
Fear	0	0	0	0	0	0	0	0	0	0	0
Sadness	5	7	2	3	4	1	4	3	4	4	5
Surprise	0	0	1	1	0	0	0	0	0	0	0
Happiness	0	0	2	2	0	3	0	1	3	1	0
Neutrality	3	1	3	2	4	4	4	4	1	3	3
Experiment failure	0	0	0	0	0	0	0	0	0	0	0

Table A.5: Emotional effects of the benchmark tests. In the columns are indicated the number of people that reacted to the considered *situation* with the emotion on the left. Data obtained by the network in [116], trained with the *Ensemble 1* database using the data augmentation (see section A.2.4) and the z-score normalization. Table from [110].

A.6 Road Tests

To check how much the trained network [116] is capable of properly recognizing emotions in a real scenario, my coauthors and I performed in August 2019 some tests inside a car in daylight conditions.

To perform these tests, we acted 20 posed facial expressions for each emotion relevant for the purposed of this work (*fear, happiness, neutrality, sadness, and surprise*). These 100 classifications are for sure not sufficient to obtain statistically relevant results, but they are yet effective to demonstrate the capability of the network to operate in realistic scenarios.

To perform these tests have been used a preliminary version of *Emotion Detector* [113].

These test are made with multiple session involving different actors and cars. The actor/actress sat in the back seat of the car, with a laptop or a table on his/her legs, raised with a pedestal to keep the embedded camere at a right height to frame the actor/actress' face.

The first session was made with the network [116] trained on the FER2013 database [117] with FER+ annotation [118]. This is not a well-balanced database (see table A.1) and, as it has been seen in section A.4.2 aond its confusion matrix shown in A.7 it has low performances with some emotions, like contempt or disgust, but has at the same time a good accuracy in the recognition of happiness and neutrality.

Figures A.18 and A.19 shows some results, with myself and Antonio Costantino Marceddu acting respectively neutrality and happiness.

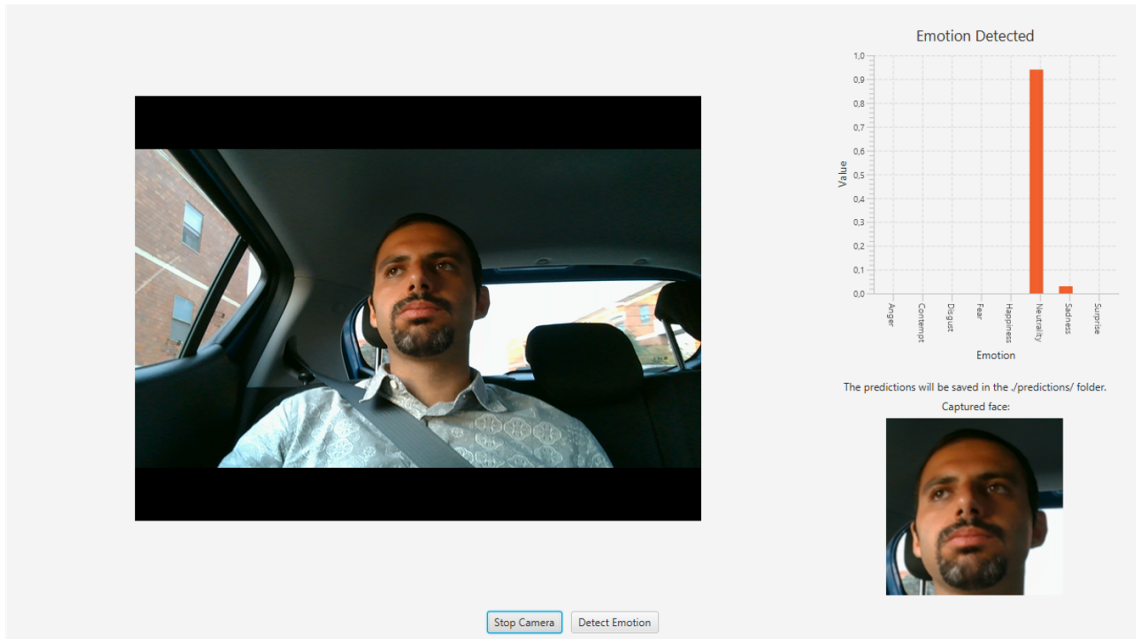


Figure A.18: Myself acting a neutral face as recognized by the neural network [116] trained on FER2013 [117] with FER+ annotations [118].Figure from [112].

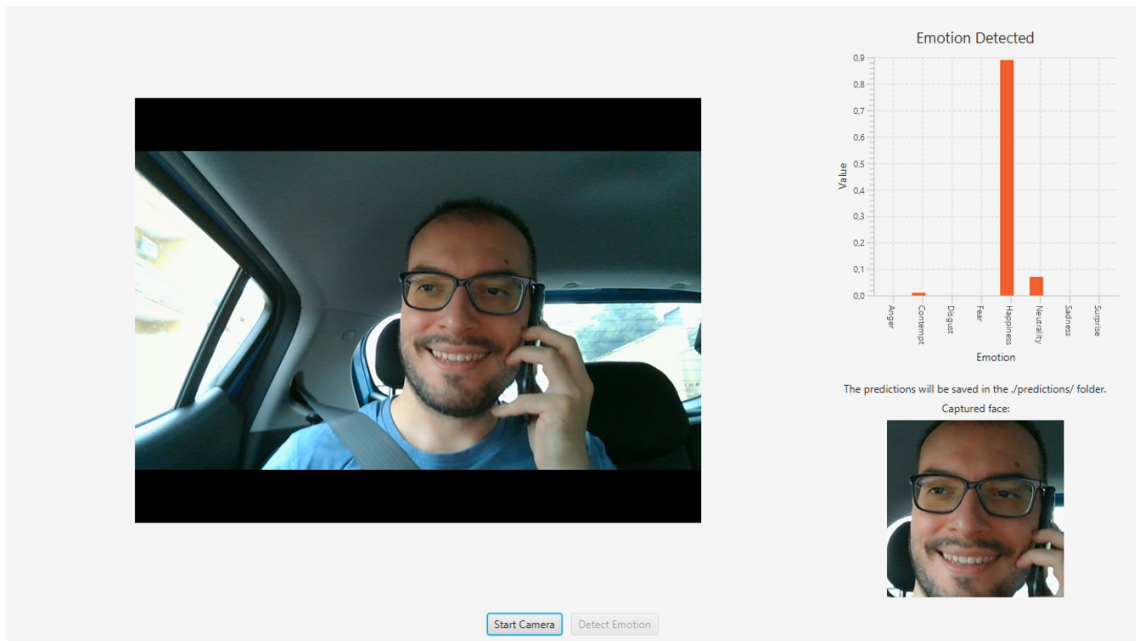


Figure A.19: Antonio Costantino Marceddu acting a neutral face as recognized by the neural network [116] trained on FER2013 [117] with FER+ annotations [118].Figure from [112].

A.7 Conclusions

This appendix describes the results obtained from [110] to implement a proof-of-concept to smooth the transition towards autonomous vehicles. This system aims to improve the passengers' trustiness in these vehicles. A delicate calibration of the driving functions should be performed, making the AV decisions closest to the ones expected by the passengers. To develop such a system, they have been adopted machine learning techniques to recognize passengers' emotions, making it possible to obtain an objective comparison between various driving algorithms *calibrations*. Two state-of-the-art neural networks, [116] and [140], have been chosen to achieve this result, hence implemented, trained, and tested in different conditions.

Moreover, two software tools have been developed and released on GitHub under an open-source licence: *Facial Expressions Databases Classifier (FEDC)* [115] and *Emotions Detector* [113]. The first has been designed to generate large facial expressions pictures databases by merging and processing various databases. The second has been developed for internal use to analyze the testers' emotions during the *situations* representations. The proposed methodology demonstrated itself useful to help designers choose between different calibrations of the trajectory planner when applied considering two different conditions.

A.7.1 Training results

Other than the training results from [110], also the results from [111], obtained by training the network on a database ensemble (*Ensemble 3*) composed with three more databases with respect to *Ensemble 1*: Indian Movie Face Database (IMFDB) [129], NimStim Set Of Facial Expression Database [130], and Real-World Affective Faces Database (RAF-DB) [131] [123]. Unfortunately, the best result is still the one obtained with seven databases in *Ensemble 1*. The network trained on *Ensemble 2* has a greater accuracy with respect to the latter but, since it has been trained only to recognize posed expression, it is less accurate with respect to the one trained on the *Ensemble 1* when used in a real-world application.s

For the reader convenience, the reached accuracies have been collected on table [A.6](#).

	<i>Ensemble 1</i>	<i>Ensemble 2</i>	CK+	FER2013	<i>Ensemble 3</i>	IMFDB
[116]	74.79 %	94.69 %	82.50 %	56.57 %		
	78.85 ^b %	97.20 ^b %	92.50 ^b %	61.40 ^b %	70.162 ^c %	47.908 ^c %
	80.38 ^c %	97.34 ^c %	92.50 ^c %	62.46 ^c %		
			78.36 ^{a,c} %			
[140]	71.39 %	92.59 %		54.03 %		
	76.91 ^b %	94.83 ^b %	N.A.	61.67 ^b %	Not trained	Not trained
	78.47 ^c %	96.78 ^c %		62.71 ^c %		
			76.67 ^{a,c} %			

Table A.6: Test accuracies summary table (best values) for all the trainings described in this dissertation. Data retrieved from [110] (for the trainings on *Ensemble 1*, *Ensemble 2*, CK+, and FER2013) and form [111] (for the trainings on *Ensemble 3* and IMFDB).

A.7.2 Future works

As future work, it can be analyzed the possibility to improve the obtained results by using an improved car simulator, with motion capabilities and a curved screen, to increase the immersion in the simulated environment and the number of testers to obtain analysis with statistically relevant results.

Bibliography

- [1] Vard Antinyan. 2020. Revealing the complexity of automotive software. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). Association for Computing Machinery, New York, NY, USA, 1525–1528. DOI:<https://doi.org/10.1145/3368089.3417038>
- [2] Fürst S, Spokesperson AU. "Autosar the next generation—the adaptive platform." In Proc. Conf. CARS@ EDCC, 2015.
- [3] National Transportation Safety Board (NTSB), "Preliminary Report Highway HWY18MH010", <https://www.nts.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf>, last visited 06/03/2021 (all date in this bibliography are in the format mm/dd/yyyy).
- [4] A user guide to vehicle automation modes, <https://pr-97195.medium.com/a-users-guide-to-vehicle-automation-modes-4bdd49b30dc0>, last visited 06/03/2021.
- [5] Automotive Industry Action Group (AIAG) and Verband Automobilindustrie (VDA), AIAG&VDA FMEA Handbok, ISBN-13: 9781605343679 (June 2019)
- [6] Prof. Phil Koopman of Carnegie Mellon University lectures, <http://users.ece.cmu.edu/~koopman/>, last visited 06/03/2021.
- [7] ISO 26262:2018, Road vehicles — Functional safety
- [8] J. Sini, A. Mugoni, M. Violante, A. Quario, C. Argiri and F. Fusetti, "An automatic approach to integration testing for critical automotive software," 2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), Taormina, Italy, 2018, pp. 1-2, doi: 10.1109/DTIS.2018.8368563.
- [9] Esposito, S.; Sini, J.; Violante, M., "Real-time validation of mixed-criticality applications", 2018 IEEE 19th Latin-American Test Symposium (LATS2018)
- [10] Esposito, Stefano; Sini, Jacopo; Violante, Massimo, "Real-Time Validation of Fault-Tolerant Mixed-Criticality Systems", 24th IEEE International Symposium on On-Line Testing and Robust System Design 2018 (IOLTS2018)

- [11] J. Sini, M. Violante and R. Dessì, "Computer-Aided Design of Multi-Agent Cyber-Physical Systems," 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 2018, pp. 677-684, doi: 10.1109/ETFA.2018.8502448.
- [12] SAFETY ANALYSIS APPROACHES FOR AUTOMOTIVE ELECTRONIC CONTROL SYSTEMS, <https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/2015sae-hommes-safetyanalysisapproaches.pdf>, last visited 06/03/2021.
- [13] Christmansson, J., & Chillarege, R. (1996, June). Generation of an error set that emulates software faults based on field data. In *Fault Tolerant Computing, 1996., Proceedings of Annual Symposium on* (pp. 304-313). IEEE.
- [14] Hsueh, M. C., Tsai, T. K., & Iyer, R. K. (1997). Fault injection techniques and tools. *Computer*, 30(4), 75-82.
- [15] Bagalini, E.; Sini, J.; Reorda, M.S.; Violante, M.; Klimesch, H.; Sarson, P. An automatic approach to perform the verification of hardware designs according to the ISO26262 functional safety standard. In *Proceedings of the 18th IEEE Latin American Test Symposium (LATS)*, Bogota, Colombia, 13–15 March 2017.
- [16] Sini, J., Violante, M. "An Automatic approach to Perform FMEDA Safety Assessment on Hardware Designs", In: *IEEE 24th International Symposium on On-Line Testing And Robust System Design IOLTS* (2018), DOI: 10.1109/IOLTS.2018.8474217
- [17] Sini, Jacopo; Sonza Reorda, Matteo; Violante, Massimo; Sarson, Peter, "Towards an automatic approach for hardware verification according to ISO 26262 functional safety standard", *24th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS2018)*
- [18] SINI, JACOPO; VIOLANTE, MASSIMO; DESSI, RICCARDO, "ISO26262-Compliant Development of a High Dependable Automotive Powertrain Item" In: *LECTURE NOTES IN ELECTRICAL ENGINEERING*
- [19] Sini, J.; D'Auria, M.; Violante, M. Towards Vehicle-Level Simulator Aided Failure Mode, Effect, and Diagnostic Analysis of Automotive Power Electronics Items. In *Proceedings of the 2020 IEEE Latin-American Test Symposium (LATS)*, Maceio, Brazil, 30 March–2 April 2020., doi: 10.1109/LATS49555.2020.9093694.
- [20] Piumatti, Davide; Sini, Jacopo; Borlo, Stefano; Sonza Reorda, Matteo; Bojoi, Radu; Violante, Massimo, "Multilevel Simulation Methodology for FMECA Study Applied to a Complex Cyber-Physical System", *MDPI ELECTRONICS* 2020
- [21] Manoscritto inviato a TETC sul rover marziano!!!
- [22] D'Auria M.; "Novel Hardware Verification Techniques for Electric Vehicles", Link: <https://webthesis.biblio.polito.it/14529/>, last visited 06/03/2021.

- [23] Passarino, A.; "Dependability in a Mission Critical Scenario: D.I.A.N.A. Mars Rover System Analysis", Link: <https://webthesis.biblio.polito.it/15959/>, last visited 06/03/2021.
- [24] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay, "Experience with fault injection experiments for FMEA", *Softw. Pract. Exp.*, vol. 41, no. 11, pp. 1233–1258, Oct. 2011
- [25] H. H. Ammar, S. M. Yacoub, and A. Ibrahim, "A fault model for fault injection analysis of dynamic UML specifications", in *12th International Symposium on Software Reliability Engineering, 2001. ISSRE 2001. proceedings, 2001*, pp. 74–8
- [26] S. M. Yacoub and H. H. Ammar, "A methodology for architecture-level reliability risk analysis", *IEEE Trans. Softw. Eng.*, vol. 28, no. 6, pp. 529–547, Jun. 2002
- [27] D. E. M. Nassar, W. Abdelmoez, M. Shereshevsky, H. H. Ammar, A. Mili, B. Yu, and S. Bogazzi, "Error propagation analysis of software architecture specifications," in *Proc. of the International Conference on Computer and Communication Engineering, ICCCE, 2006*.
- [28] N. Snooke and C. Price, "Model-driven automated software FMEA," in *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual, 2011*, pp. 1–6.
- [29] D. Cotroneo and R. Natella, "Fault Injection for Software Certification," *IEEE Secur. Priv.*, vol. 11, no. 4, pp. 38–45, Jul. 2013
- [30] Rearick J., "IEEE P2427: Proposing the Essential Framework for Measuring Defect Coverage in Analog Circuits" (2018) http://sagroups.ieee.org/2427/wp-content/uploads/sites/302/2018/05/2C3_P2427_framework_vts18_rearick.pdf, last visited 2021/08/26.
- [31] IEEE P2427 - Standard for Analog Defect Modeling and Coverage, available online at <https://standards.ieee.org/project/2427.html>, last visited 2021/08/25.
- [32] M. Slamani and B. Kaminska, "Analog circuit fault diagnosis based on sensitivity computation and functional testing," in *IEEE Design & Test of Computers*, vol. 9, no. 1, 1992
- [33] A. Arabi, N. Bourouba, A. Belaout, M. Ayad, "Catastrophic faults detection of analog circuits," *7th International Conference on Modelling, Identification and Control (ICMIC), Sousse, 2015*
- [34] P. Duhamel, J. Rault, "Automatic test generation techniques for analog circuits and systems: A review," in *IEEE Transactions on Circuits and Systems*, vol. 26, no. 7, 1979
- [35] A. Benso, P. Prinetto, "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation", 2003, Kluwer.
- [36] 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

- [37] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. C. Fabre, J. C. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: A methodology and some applications," *Softw. Eng. IEEE Trans. On*, vol. 16, no. 2, pp. 166–182, 1990
- [38] 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.
- [39] Cukier, M., Powell, D., & Ariat, J. (1999). Coverage estimation methods for stratified fault-injection. *IEEE Transactions on Computers*, 48(7), 707-723
- [40] Piumatti, D.; Sini, J.; Borlo, S.; Sonza Reorda, M.; Bojoi, R.; Violante, M. Multilevel Simulation Methodology for FMECA Study Applied to a Complex Cyber-Physical System. *Electronics* 2020, 9, 1736.
- [41] Sastry, A.; Kulasekaran, S.; Flicker, J.; Ayyanar, R.; Tamizhmani, G.; Roy, J.; Srinivasan, D.; Tilford, I. Failure modes and effect analysis of module level power electronics. In *Proceedings of the 42nd Photovoltaic Specialist Conference (PVSC)*, New Orleans, LA, USA, 14–19 June 2015
- [42] Peyghami, S.; Davari, P.; F-Firuzabad, M.; Blaabjerg, F. Failure Mode, Effects and Criticality Analysis (FMECA) in Power Electronic based Power Systems. In *Proceedings of the 21st European Conference on Power Electronics and Applications (EPE '19 ECCE Europe)*, Genova, Italy, 3–5 September 2019.
- [43] CoppeliaSim - <https://www.coppeliarobotics.com/>, last visited 06/03/2021.
- [44] Example of a training course provided by Texas Instruments about SoC-specific ISO26262 FMEDA metrics computation tools. <https://training.ti.com/basics-fmeda-and-how-it-useful-system-level-safety-analysis-part-1>, last visited 2021/08/31.
- [45] E. Bagalini, M. Violante, H. Hakobyan, "Evaluation of error effects on a biomedical system" , *IEEE East-West Design & Test Symposium*, 2015, pp. 39-42
- [46] G. Cassanelli, et al. "Reliability predictions in electronic industrial applications", In: *Microelectronics Reliability*, 2005, 45.9-11: 1321-1326
- [47] D.Piumatti, M. Sonza Reorda, "Assessing Test Procedure Effectiveness for Power Devices", In: *IEEE 33rd Conference on Design of Circuits and Integrated Circuits* (2018)
- [48] Hsueh, M. C., Tsai, T. K., & Iyer, R. K. (1997). Fault injection techniques and tools. *Computer*, 30(4), 75-82.
- [49] D. Cotroneo & R. Natella (2013). Fault injection for software certification. *IEEE Security & Privacy*, 11(4), 38-45.
- [50] <https://www.carsim.com/>, last visited 06/03/2021.
- [51] Cickaric, L.S.; Katic, V.A.; Milic, S. Failure Modes and Effects Analysis of Urban Rooftop PV Systems—Case Study. In *Proceedings of the International*

- Symposium on Industrial Electronics (INDEL), Banja Luka, Bosnia and Herzegovina, 1–3 November 2018.
- [52] Zhang, Z.; Hao, M. Failure Mode and Effects Analysis of UAV Power System Based on Generalized Dempster-Shafer Structures. In Proceedings of the 2019 IEEE International Conference on Unmanned Systems (ICUS), Beijing, China, 17–19 October 2019.
- [53] Banerjee, P.; Pandey, K. Implementation of Failure Modes and Effect Analysis on the electro-hydraulic servo valve for steam turbine. In Proceedings of the IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, India, 4–6 July 2016.
- [54] Rastayesh, S.; Bahrebar, S.; Bahman, A.S.; Sørensen, J.D.; Blaabjerg, F. Lifetime Estimation and Failure Risk Analysis in a Power Stage Used in Wind-Fuel Cell Hybrid Energy Systems. *Electronics* 2019, 8, 1412.
- [55] Parker, K.P. A New Process for Measuring and Displaying Board Test Coverage. In Proceedings of the Apex 2003, Anaheim, CA, USA, 4 September 2003. Available online: https://www.keysight.com/upload/cmc_upload/All/Apex_KParker_010903.pdf, last visited 06/03/2021.
- [56] Piumatti, D.; Borlo, S.; Mandrile, F.; Reorda, M.S.; Bojoi, R. Assessing the Effectiveness of the Test of Power Devices at the Board Level. In Proceedings of the XXXIV Conference on Design of Circuits and Integrated Systems (DCIS), Bilbao, Spain, 20–22 November 2019.
- [57] European Cooperation for Space Standardization ECSS-Q-ST-30-02C – Failure modes, effects (and criticality) analysis (FMEA/FMECA) – (6 March 2009)
- [58] PLECS Tool, Plexim. Available online: <https://www.plexim.com/plecs>, last visited 06/03/2021..
- [59] MathWorks MATLAB - <https://www.mathworks.com/>, last visited 06/03/2021.
- [60] W. Yongming, Y. Xiaoliu and T. Wencheng, "Analysis of Obstacle-Climbing Capability of Planetary Exploration Rover with Rocker-Bogie Structure," 2009 International Conference on Information Technology and Computer Science, Kiev, 2009, pp. 329-332, doi: 10.1109/ITCS.2009.74.
- [61] O. Toupet et al., "Traction control design and integration onboard the Mars science laboratory curiosity rover," 2018 IEEE Aerospace Conference, Big Sky, MT, 2018, pp. 1-20, doi: 10.1109/AERO.2018.8396761.
- [62] T. P. Setterfield and A. Ellery, "Terrain Response Estimation Using an Instrumented Rocker-Bogie Mobility System," in *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 172-188, Feb. 2013, doi: 10.1109/TRO.2012.2223591.
- [63] D. Michel and K. McIsaac, "New rocker-bogie and terramechanics-based wheel/soil interaction models for planetary rovers," 2012 IEEE International Conference on Mechatronics and Automation, Chengdu, 2012, pp. 2417-2422, doi: 10.1109/ICMA.2012.6285724.

- [64] K. Siddartha, S. Birrell, G. Dhadyalla, H. Sivencrona, P. Jennings, "Towards increased reliability by identification of Hazard Analysis and Risk Assessment (HARA) of automated automotive Systems", In: Safety Science 99 166–177, 2017, DOI: 10.1016/j.ssci.2017.03.024
- [65] J. Sini, M. Violante, V. Dodde, R. Gnaniah and L. Pecorella, "A Novel Simulation-Based Approach for ISO 26262 Hazard Analysis and Risk Assessment," 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS), Rhodes, Greece, 2019, pp. 253-254, doi: 10.1109/IOLTS.2019.8854385.
- [66] Jacopo Sini, Massimo Violante, "A simulation-based methodology for aiding advanced driver assistance systems hazard analysis and risk assessment", Microelectronics Reliability, Volume 109, 2020, 113661, ISSN 0026-2714, <https://doi.org/10.1016/j.microrel.2020.113661>, last visited 06/03/2021.
- [67] Jang. H.A., Kwon H.M., Hong S., Lee, M. K., "A study on Situation Analysis for ASIL Determination", In: Journal of Industrial and Intelligent Information Vol. 3 No. 2, June 2015, DOI: 10.1016/j.ress.2016.09.004
- [68] K Beckers, D. Holling, Coté I.M., Hatebur D., "A structured hazard analysis and risk assessment method for automotive systems – A descriptive study" In: Reliability Engineering and System Safety (2017) pg. 185-195
- [69] National Instruments™ ADAS and Autonomous Driving Validation Test <https://www.ni.com/it-it/solutions/transportation/adas-and-autonomous-driving-validation.html>, last visited 2021/09/01.
- [70] IPGCarMaker, <https://ipg-automotive.com/products-services/simulation-software/carmaker/>, last visited 06/03/2021.
- [71] AVL VSM™ Vehicle Simulation <https://www.avl.com/-/avl-vsm-4->, last visited 2021/09/01.
- [72] FEV™ VirtualDynamics™, <https://virtualdynamics.fev.com/>, last visited 2021/09/01.
- [73] CARLA Open-source simulator for autonomous driving research.<https://carla.org/>, last visited 2021/09/01.
- [74] Scenic, domain-specific probabilistic programming language for modeling the environments of cyber-physical systems like robots and autonomous cars. <https://scenic-lang.readthedocs.io/en/latest/>, last visited 2021/09/01.
- [75] Johanennessen, "Actuator Based Hazard Analysis for Safety Critical Systems", In: Computer Safety, Reliability, and Security SAFECOMP 2004 Proceedings
- [76] H.A. Jang, H.M Kwon., S.H. Hong, M.K. Lee, "A study on situation analysis for ASIL determination" In: Journal of Industrial and Intelligent Information Vol. 3 No. 2, June 2015, DOI: 10.12720/jiii.3.2.152-157
- [77] El-Bayoumi, A. (2020). An enhanced algorithm for memory systematic faults detection in multicore architectures suitable for mixed-critical automotive applications. International Journal of Safety and Security Engineering, Vol. 10, No. 4, pp. 467-474. <https://doi.org/10.18280/ijssse.100405>

- [78] Gesina Schwalbe, Martin Schels. A Survey on Methods for the Safety Assurance of Machine Learning Based Systems. 10th European Congress on Embedded Real Time Software and Systems (ERTS2020), Jan 2020, Toulouse, France. hal-02442819
- [79] Huang, A., Xing, X., Zhou, T., and Chen, J., "A Safety Analysis and Verification Framework for Autonomous Vehicles Based on the Identification of Triggering Events," SAE Technical Paper 2021-01-5010, 2021, <https://doi.org/10.4271/2021-01-5010>.
- [80] H. Kwon, R. Itabashi-Campbell and K. McLaughlin, "ISO26262 application to electric steering development with a focus on Hazard Analysis," 2013 IEEE International Systems Conference (SysCon), Orlando, FL, 2013, pp. 655-661. DOI: 10.1109/SysCon.2013.6549952
- [81] ISO15622:2018 "Intelligent transport systems–Adaptive cruise control systems–Performance requirements and test procedures"
- [82] European New Car Assessment Programme(EuroNCAP), "Test Protocol–AEB systems", November 2017
- [83] U.S. Department of Transportation – National Highway Traffic Safety Administration, "Intelligent Cruise Control Field Operational Test (Final Report), May 1998
- [84] European Commission Regulation 347/2017 Attachment 1
- [85] TXT XHIL Studio <https://www.txtgroup.com/it/mercati/our-markets/automotive-transport/>, last visited 06/03/2021.
- [86] Robert Bosch GmbH (ed.) "Bosch Automotive Electrics and Automotive Electronics Systems and Components, Networking and Hybrid Drive" 5th edition (2014). ISBN: 978-3-658-01783-5
- [87] Alessandra Mugoni, "Development of software tools for functional testing of automotive electronic control units", December 2017, Politecnico di Torino
- [88] NI VeriStand Fundamentals Course Manual, National Instruments, 2011
- [89] Liu, B., Zhang, H., Zhu, S., (2016). An incremental V-Model Process for Automotive Development. 23rd Asia-Pacific Software Engineering Conference.
- [90] Tibba, G., Malz, C., Stoermer, C., Nagarajan N., Zhang, L., Chakraborty S. (2016) Testing Automotive Embedded system under X-in-the-Loop Setups, 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)
- [91] NI VeriStand .NET Reference, http://zone.ni.com/reference/en-XX/help/372846J-01/veristand/vs_net_reference/, last visited 06/03/2021.
- [92] P. J. Prisaznuk, "Integrated modular avionics," Aerosp. Electron. Conf. 1992. NAECON 1992., Proc. IEEE 1992 Natl., pp. 39–45 vol.1, 1992.
- [93] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics," AIAA/IEEE Digit. Avion. Syst. Conf. - Proc., pp. 1–10
- [94] S. Avramenko, S. Esposito, M. Violante, M. Sozzi, M. Traversone, M. Binello, and M. Terrone, "An Hybrid Architecture for Consolidating Mixed Criticality

- Applications on Multicore Systems,” in 2015 IEEE 21st International On-Line Testing Symposium, 2015, pp. 26–29
- [95] S. Esposito, M. Violante, M. Sozzi, M. Terrone, and M. Traversone, “A novel method for online detection of faults affecting execution-time in multicore-based systems,” *ACM Trans. Embed. Comput. Syst.*, 2017, vol. 16, no. 4, pp. 1–19
- [96] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, “Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement,” *Proc. - Euromicro Conf. Real- Time Syst.*, pp. 109–118
- [97] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht, “Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems,” *Des. Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–5
- [98] A. Burns and R. I. Davis, *Mixed Criticality Systems - A Review*, 7th edition. Univerisy of York, 2016.
- [99] J. Rushby, “Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance,” NASA Langley Research Center, NASA CR-1999-209347.
- [100] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” *Proc. - Real-Time Syst. Symp.*, pp. 239–243
- [101] J. Anderson, S. Baruah, and B. Brandenburg, “Multicore operating- system support for mixed criticality,” in *Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.
- [102] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, “Scheduling of mixed-criticality applications on resource-sharing multicore systems,” 2013 *Proc. Int. Conf. Embed. Software, EMSOFT 2013*
- [103] G. Caire, M. Cossentino, A. Negri, A. Poggi, P. Turci, “Multi-Agent Systems Implementation and testing”, In: 4th International Symposium – From Agent Theory to Agent Implementation (AT2AI-4) 2004
- [104] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, M. Violante, *Software-implemented Hardware Fault Tolerance*, Springer Science & Business Media, 2006.
- [105] Corke P., “Robotics, Vision and Control – Fundamental Algorithms in MATLAB”, Springer Tracts in Advanced Robotics Volume 73 (2011)
- [106] Gibson, K. D., and Harold A. Scheraga. "Volume of the intersection of three spheres of unequal size: a simplified formula." *Journal of Physical Chemistry* 91.15 (1987): 4121-4122.
- [107] Viegas C., Tavakoli M., Lopes P., Dessì R. et al. “SCALA-A Scalable Rail-Based Multitrobot System for Large Space Automation, Design and Development”, In: *IEE/ASME Transaction of Mechatronics*, Vol. 22 N. 5 2208-2216 (October 2017)

- [108] Carretero, J. A., et al. "Kinematic analysis and optimization of a new three degree-of-freedom spatial parallel manipulator." *Journal of mechanical design* 122.1 (2000): 17-24.
- [109] Sini J., Marceddu A.C., Violante M., Dessì R. (2021) Passengers' Emotions Recognition to Improve Social Acceptance of Autonomous Driving Vehicles. In: Esposito A., Faundez-Zanuy M., Morabito F., Pasero E. (eds) *Progresses in Artificial Intelligence and Neural Systems. Smart Innovation, Systems and Technologies*, vol 184. Springer, Singapore. https://doi.org/10.1007/978-981-15-5093-5_3, last visited 06/03/2021.
- [110] Sini, J.; Marceddu, A.C.; Violante, M. Automatic Emotion Recognition for the Calibration of Autonomous Driving Functions. *Electronics* 2020, 9, 518. <https://doi.org/10.3390/electronics9030518>, last visited 06/03/2021.
- [111] Antonio Costantino Marceddu, Jacopo Sini, Massimo Violante, Bartolomeo Montrucchio, "A novel approach to improve the social acceptance of autonomous driving vehicles by recognizing the emotions of passengers," *Proc. SPIE 11605*, Thirteenth International Conference on Machine Vision, 116051R (4 January 2021); <https://doi.org/10.1117/12.2586417>, last visited 06/03/2021.
- [112] Antonio Costantino Marceddu, "Automatic Recognition And Classification Of Passengers' Emotions In Autonomous Driving Vehicles." Rel. Massimo Violante, Jacopo Sini. Politecnico di Torino, Corso di laurea magistrale in Ingegneria Informatica (Computer Engineering), 2019 <https://webthesis.biblio.polito.it/12423/>, last visited 06/03/2021.
- [113] Emotion Detector https://github.com/AntonioMarceddu/Emotion_Detector, last visited 06/03/2021.
- [114] Eclipse DeepLearning4j Development Team. DeepLearning4j: Open-sOurce Distributed Deep Learning for the JVM, Apache Software Foundation License 2.0. 2019. Available online: <http://deeplearning4j.org/>, last visited 06/03/2021.
- [115] Facial Expression Database Classifier (FEDC) https://github.com/AntonioMarceddu/Facial_Expressions_Databases_Classifier, last visited 06/03/2021.
- [116] Ferreira, P.M.; Marques, F.; Cardoso, J.S.; Rebelo, A. Physiological inspired deep neural networks for emotion recognition. *IEEE Access* 2018, 6, 53930–53943, doi:10.1109/ACCESS.2018.2870063.
- [117] "Challenges in Representation Learning: A report on three machine learning contests." I. Goodfellow, D. Erhan, P.L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D.H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Ro-maszko, B. Xu, Z. Chuang, and Y. Bengio. arXiv 2013

- [118] Barsoum, Emad & Zhang, Cha & Ferrer, Cristian & Zhang, Zhengyou. (2016). Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution. 279-283. DOI:10.1145/2993148.2993165.
- [119] Ekman, P. "Basic emotions." In *Handbook of Cognition and Emotion*; University of California, San Francisco, CA, USA, 1999; pp. 45–60.
- [120] Ekman, P.; Friesen, W. *Facial Action Coding System (FACS): A Technique for the Measurement of Facial Action*; Consulting: Palo Alto, CA, USA, 1978.
- [121] Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00), Grenoble, France, 46-53.
- [122] Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. (2010). The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified expression. Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010), San Francisco, USA, 94-101.
- [123] S. Li, W. Deng and J. Du, "Reliable Crowdsourcing and Deep Locality-Preserving Learning for Expression Recognition in the Wild," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2584-2593.
- [124] Michael J. Lyons, Shigeru Akamatsu, Miyuki Kamachi, Jiro Gyoba. Coding Facial Expressions with Gabor Wavelets, 3rd IEEE International Conference on Automatic Face and Gesture Recognition, pp. 200-205 (1998). DOI: 10.1109/AFGR.1998.670949
- [125] N. Aifanti, C. Papachristou and A. Delopoulos, "The MUG Facial Expression Database", in Proc. 11th Int. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), Desenzano, Italy, April 12-14, 2010
- [126] Langner, O.; Dotsch, R.; Bijlstra, G.; Wigboldus, D.; Hawk, S.; Knippenberg, A. Presentation and validation of the radboud face database. *Cogn. Emot.* 2010, *24*, 1377–1388, doi:10.1080/02699930903485076.
- [127] Dhall, A.; Goecke, R.; Lucey, S.; Gedeon, T. Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark. In Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), Barcelona, Spain, 6–13 November 2011; pp. 2106–2112, doi:10.1109/ICCVW.2011.6130508.
- [128] Ebner, N.; Riediger, M.; Lindenberger, U. Faces a database of facial expressions in young, middle-aged, and older women and men: Development and validation. *Behav. Res. Methods* 2010, *42*, 351–362, doi:10.3758/BRM.42.1.351.
- [129] Shankar Setty, Moula Husain, Parisa Beham, Jyothi Gudavalli, Menaka Kandasamy, Radhesyam Vaddi, Vidyagouri Hemadri, JC Karure, Raja Raju, Rajan, Vijay Kumar and C V Jawahar. "Indian Movie Face Database: A Benchmark for Face Recognition Under Wide Variations"

- [130] Tottenham, N., Tanaka, J., Leon, A.C., McCarry, T., Nurse, M., Hare, T.A., Marcus, D.J., Westerlund, A., Casey, B.J., Nelson, C.A. (2009). The NimStim set of facial expressions: judgments from untrained research participants. *Psychiatry Research*, 168(3):242-9
- [131] S. Li, W. Deng and J. Du, "Reliable Crowdsourcing and Deep Locality-Preserving Learning for Expression Recognition in the Wild," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2584-2593.
- [132] Hasenjäger, M.; Wersing, H. Personalization in Advanced Driver Assistance Systems and Autonomous Vehicles: A Review. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; doi:10.1109/ITSC.2017.8317803.
- [133] Choi, J.K.; Kim, K.; Kim, D.; Choi, H.; Jang, B. Driver-Adaptive Vehicle Interaction System for the Advanced Digital Cockpit. In Proceedings of the International Conference on Advanced Communications Technology(ICACTION), Chuncheon-si Gangwon-do, Korea, 11–14 February 2018; doi:10.23919/ICACTION.2018.8323736.
- [134] Jung, S.J.; Shin, H.S.; Chung, W.Y. Driver Fatigue and Drowsiness Monitoring System with Embedded Electrocardiogram Sensor on Steering Wheel. *IET Intell. Transp. Syst.* 2014, 8, 43–50, doi:10.1049/iet-its.2012.0032.
- [135] Kusuma, B.M.; Sampada, S.; Ramakanth, P.; Nishant, K.; Atulit, S. Detection of Driver Drowsiness using Eye Blink Sensor. *Int. J. Eng. Technol.* 2018, 7, 498, doi:10.14419/ijet.v7i3.12.16167.
- [136] Levinson, J.; Askeland, J.; Becker, J.; Dolson, J.; Held, D.; Kammel, S.; Kolter, J.Z.; Langer, D.; Pink, O.; Pratt, V.; et al. Towards fully autonomous driving: Systems and algorithms. In Proceedings of the IEEE Intelligent Vehicles Symposium, Baden-Baden, Germany, 5–9 June 2011; pp. 163–168. 10.1109/IVS.2011.5940562.
- [137] Classification Loss Metric. Available online: <https://peltarion.com/knowledge-center/documentation/evaluation-view/classification-loss-metrics>, last visited 06/03/2021.
- [138] Ebner, N., Riediger, M., & Lindenberger, U. (2010). FACES—A database of facial expressions in young, middle-aged, and older women and men: Development and validation. *Behavior Research Methods*, 42, 351-362. DOI:10.3758/BRM.42.1.351.
- [139] Scikit-Learn. Available online: <https://scikit-learn.org/stable/>, last visited 06/03/2021.
- [140] Miao, S.; Xu, H.; Han, Z.; Zhu, Y. Recognizing facial expressions using a shallow convolutional neural network. *IEEE Access* 2019, 7, 78000–78011, doi:10.1109/ACCESS.2019.2921220.
- [141] Keras. Available online: <https://keras.io/>, last visited 06/03/2021.
- [142] TensorFlow. <https://www.tensorflow.org/>, last visited 06/03/2021.

- [143] The Microsoft Cognitive Toolkit. Available online: <https://docs.microsoft.com/en-us/cognitive-toolkit/>, last visited 06/03/2021.
- [144] Theano. Available online: <http://deeplearning.net/software/theano/>. Theano tool is now deprecated in favor of aesara <https://github.com/pymc-devs/aesara>, last visited 06/03/2021.
- [145] Hunter, J.D. Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 2007, 9, 90–95.
- [146] NumPy. Available online: <https://numpy.org/>, last visited 06/03/2021.
- [147] Bradski G. The OpenCV Library. Dr Dobb's Journal of Software Tools, 2000.
- [148] McKinne, W. Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference, Austin, Texas; van der Walt, S., Millman, J., Eds; pp. 51–56. Available online: https://www.researchgate.net/publication/265001241_Data_Structures_for_Statistical_Computing_in_Python, last visited 06/03/2021.
- [149] Available online: https://github.com/AntonioMarceddu/Facial_Expressions_Recognition_With_Keras, last visited 06/03/2021.
- [150] A. N. Navaz, S. M. Adel and S. S. Mathew, "Facial Image Pre-Processing and Emotion Classification: A Deep Learning Approach," 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 2019, pp. 1-8, doi: 10.1109/AICCSA47632.2019.9035268.

This Ph.D. thesis has been typeset by means of the T_EX-system facilities. The typesetting engine was pdfL^AT_EX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete T_EX-system installation.