Early Safety Assessment of Automotive Systems Using Sabotage Simulation-Based Fault Injection Framework

Garazi Juez(✉), Estíbaliz Amparan, Ray Lattarulo, Alejandra Ruíz, Joshué Pérez, and Huáscar Espinoza

TECNALIA Research & Innovation, Derio, Spain
{garazi.juez,estibaliz.amparan,rayalejandro.lattarulo,
alejandra.ruiz,joshue.perez,huascar.espinoza}@tecnalia.com

Abstract. As road vehicles increase their autonomy and the driver reduces his role in the control loop, novel challenges on dependability assessment arise. Model-based design combined with a simulation-based fault injection technique and a virtual vehicle poses as a promising solution for an early safety assessment of automotive systems. To start with, the design, where no safety was considered, is stimulated with a set of fault injection simulations (fault forecasting). By doing so, safety strategies can be evaluated during early development phases estimating the relationship of an individual failure to the degree of misbehaviour on vehicle level. After having decided the most suitable safety concept, a second set of fault injection experiments is used to perform an early safety validation of the chosen architecture. This double-step process avoids late redesigns, leading to significant cost and time savings. This paper presents a simulation-based fault injection approach aimed at finding acceptable safety properties for model-based design of automotive systems. We focus on instrumenting the use of this technique to obtain fault effects and the maximum response time of a system before a hazardous event occurs. Through these tangible outcomes, safety concepts and mechanisms can be more accurately dimensioned. In this work, a prototype tool called Sabotage has been developed to set up, configure, execute and analyse the simulation results. The feasibility of this method is demonstrated by applying it to a Lateral Control system.

Keywords: Fault Injection · Early safety assessment · Vehicle dynamics model

1 Introduction

Automated driving exhibits increasingly complex dependability challenges as the driver reduces his role in the control loop, and the vehicle must operate under exceptional situations, e.g. dealing with sensor noise. Fault Injection (FI) has been recognised as a potentially powerful technique for the safety assessment and corner-case validation of fault-tolerance mechanisms in manual and automated driving systems [1]. The major aim of performing FI is not to validate functionality, but rather to test the fault tolerance or probe how robust the vehicle is or their components are to arbitrary faults. The ISO 26262 standard [2] notably recommends its use across the validation and verification phases of the V-Cycle development process.

S. Tonetta et al. (Eds.): SAFECOMP 2017, LNCS 10488, pp. 255–269, 2017.
DOI: 10.1007/978-3-319-66266-4_17

Despite the growing use of model-based tools in the design phases, FI is a technique that has seen little widespread use in early phases [3]. The potential benefits of using FI during pre-implementation phases of automotive systems range from providing an early evaluation up to a preliminary validation of safety concepts. In the specific case of manual driving, this could help determining (human-) controllability and fault tolerant time interval (FTTI) values in early design phases. FTTI is defined as the time-span in which a fault can be present in a system before a hazardous event occurs [2]. Concerning highly automated driving, we believe that FI could be applied for dimensioning monitoring functions by determining a system maximum response time before a hazardous event occurs.

This paper explores how to use model-based FI to assess safety properties of vehicle systems and how to augment vehicle simulation with appropriate fault models for safety determination. To address these concerns, a simulation-based FI framework (Sabotage) is coupled with the Dynacar vehicle simulator [4]. Dynacar includes a vehicle model (e.g. dynamics), an environment model (e.g. driving circuit) and pre-defined sensor and actuator model libraries for e.g. engine, transmission, steering system, and braking system. The added value of including vehicle and environment models is that the maximum time before the vehicle dynamics are unsafely affected can be identified. In other words, it allows quantitatively estimating the relationship of an individual failure to the degree of misbehaviour on vehicle level.

Our approach has been evaluated on a case study for the model-based design of a Lateral Control system. We have focused on automatically inserting fault injection model blocks (saboteurs), which represent failure modes, and forecast maximum system reaction times based on the critical lateral deviation (maximum lateral error). This value determines the required level of fault tolerance – e.g. redundancy or graceful degradation – without affecting vehicle safety. A good estimation of these values helps engineers to better define appropriate safety goals and requirements as main output of the safety concept [5].

The remainder of this paper is structured as follows. Section 2 presents the related state of the art. Section 3 introduces the Sabotage tool framework and how it can be used for an early safety assessment. Afterwards, Sect. 4 shows how the aforementioned method is applied to the Lateral Control case study. Finally, Sect. 5 presents conclusions and future work.

2 Related Work

Fault injection has been deeply investigated by both academia and industry as surveyed in [6] and described in [7]. The idea of using simulation-based FI in early design phases is not that widely spread. For instance, Svenningson [8] investigated the benefits of applying this technique on Simulink behavioural models, and Vinter et al. [9] developed a similar approach for the SCADE toolset. Even if failures can be derived, those effects and FTTI values cannot be estimated on vehicle level since no vehicle dynamics is considered. The closest to an investigation of simulation-based FI that integrated vehicle

dynamics is presented in [10, 11]. Silveira et al. [10] implemented a seamless co-simulation approach that combines Matlab and CarSim for evaluating the fault impacts on vehicle stability. Also, Jones et al. [11] introduced a similar co-simulation solution using Matlab, CarMaker/TruckMaker and CRUISE tools. This work especially supports the determination of Automotive Safety Integrity Levels (ASIL) during the concept phase as per ISO 26262. The major drawback of these approaches is that the fault library and the solution are language dependent and the automation level could be further developed.

Concerning the use of FI across the range of abstraction levels of ISO 26262, most of the work has been done as a way of verifying the implemented safety mechanisms or safety requirements [12]. Only few works have emphasised its usage during early design phases. Pintard [3] developed guidance for applying FI on both sides of the ISO 26262's V-Cycle, including system and hardware pre-implementation phases; however, the aim of the author was not to develop a fault injection framework.

3 Overview of the Simulation-Based Fault Injection Approach

3.1 The Sabotage Tool Framework

Simulation-based fault injection is a technique that uses a series of high-level abstractions or models representing the system under study to evaluate and validate its dependability in early design phases. Thus the system is simulated on the basis of simplified assumptions to (a) predict its behaviour in the presence of faults, (b) to estimate the failure coverage and timing of fault tolerant mechanisms, or (c) to explore the effects of different workloads, i.e. different activation profiles. Applying FI provides remarkable benefits for designers. On the one hand, fault forecasting is achieved by performing an evaluation of the system behaviour with respect to fault occurrence or activation. On the other hand, as tackled in [14], FI is seen as a dynamic testing technique to achieve fault removal during the development phase of a system (verification, diagnosis, and correction).

Sabotage is a simulation-based fault injection tool framework based on the well-known FARM environment model [13]. The FARM model is composed of: (1) the set of Faults to be injected, (2) the set of Activations exercised during the experiment, (3) the Readouts to define observers of system behaviour, and (4) the Measures obtained to evaluate dependability properties. In the rest of this section, we describe the particularities of the proposed tool framework in light of the FARM constituents.

Figure 1 shows the Sabotage building blocks and the flow of models to perform a safety assessment during vehicle simulation in early design phases. The tightly-coupled simulation environment is constituted as follows: the Sabotage framework is used to set up, configure, run and analyse FI experiments. The Dynacar vehicle simulator [4], integrated as a Matlab/Simulink system function (S-function), includes models to represent some vehicle sensors, actuators and dynamics. An S-function is a computer language description of a Simulink block written in Matlab, C, C++, or Fortran. Besides, Dynacar provides a graphical user interface where the previously configured operational situations are observed. The model of the whole system is completed by including simulation

models representing the Electronic Control Unit (ECU) functions (also known as controller model or control strategies). By co-simulating the three applications we are able to carry out a closed-loop modelling of the vehicle control system in the presence of faults.

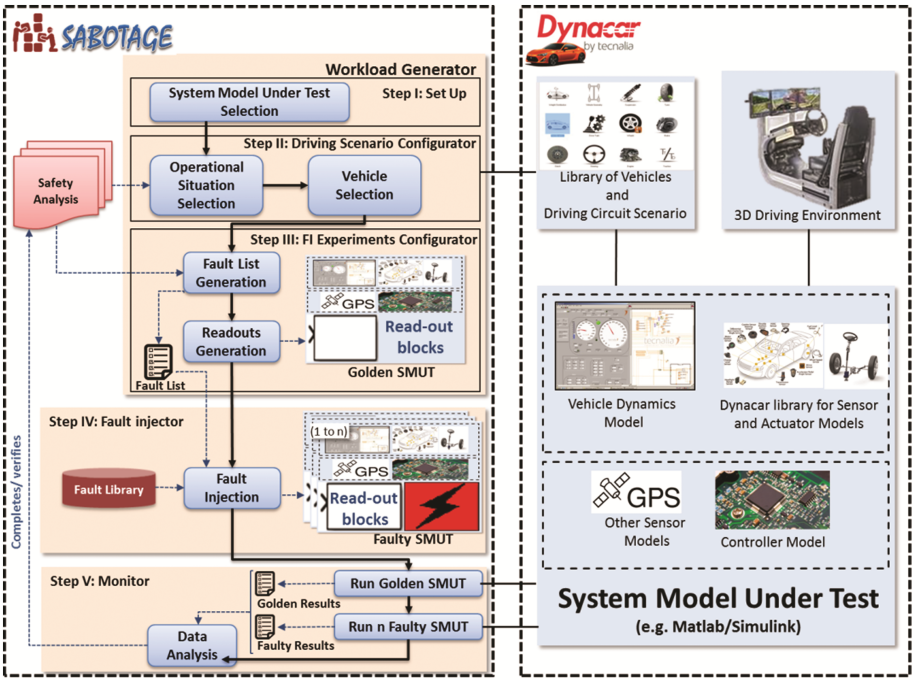

Fig. 1. Sabotage framework for simulation-based fault injection

The Sabotage framework operates as follows. First, a Workload Generator generates the functional inputs to be applied to the system model under test (SMUT). All these models are developed, for instance, using Matlab/Simulink. The Workload Generator consists of (i) selecting the system model under test, (ii) choosing the operational scenario from a driving circuit and environment scenario library, and (iii) configuring fault injection experiments, i.e. creating the fault list and deciding the read-out or observation points (signal monitors). Then a Fault Injector (for now implemented as Matlab code) uses both, the fault list and a fault model library (implemented as C code templates or XML), to create the saboteurs (S-functions) and generate as many Faulty SMUT as the designer needs. Once a fault free version of the SMUT (Golden) and at least one Faulty SMUT are available, the simulation environment is invoked through the Monitor (the Oracle implemented in Java). The Monitor not only runs experiments under the pre-configured vehicle scenario, but also compares and analyses the collected data.

Workload Generator. This block is in charge of three main activities: selecting the SMUT, choosing the most appropriate driving scenario, which represents the operational

situation, and configuring fault injection experiments. Safety analysis provides the basis for specifying the operational situations (i.e. location, road conditions, environment conditions and the like). After that, the designer selects the vehicle and a driving circuit scenario that best symbolises those operational situations to be simulated. Dynacar manages a scenario catalogue that includes up to 150 configurable parameters, thus enabling the emulation of a wide range of vehicles and driving circuit scenarios. The *fault injection experiments configurator* block in Sabotage also addresses the inclusion of extra readout model blocks (signal monitors) in the target system (SMUT) to facilitate the logging process of output data.

Moreover, the activity of *fault list generation* creates a subset of faults that can be injected in a reasonable time but are still able to provide significant results. Our strategy to identify a representative fault subset is to use the target system malfunctions or failure modes, e.g. omission or commission, instead of injecting an exhaustive or random fault set. The kinds of faults in the subset include permanent, intermittent and transient faults.

Fault Injector. The fault list is used to produce a Faulty SMUT only in terms of reproducible and prearranged fault models. Fault models are characterised by a type (e.g. omission, frozen, delay, invert, oscillation or random), target location, injection triggering (e.g. driving circuit position or time driven), and duration. To create a Faulty SMUT, the Fault Injector injects an additional saboteur model block per fault entry from the fault list together with the associated fault models which are coded as templates in a fault library. Saboteurs are extra components added as part of the model-based design for the sole purpose of FI experiments. Algorithm 1 depicts a generic fault model for omission represented by a stuck-at last value.

```
Require:  input, pos,simutime,faultdur;
1         If pos== triggerpos then
2           Freeze=input;
3           enable=1;
4         While enable==1 && simutime<=faultdur do return freeze;
5         return input;
```

Algorithm 1. Stuck-at Last Value.

Monitor. After setting up the FI scenarios and having conceived the required amount of Faulty SMUT, the Monitor starts the simulation process. It tracks the execution flow of the Golden and Faulty simulation runs via *the readouts collection* activity. The Monitor compares Golden and Faulty SMUT results by the *data analysis* activity. The pass/fail criterion of the tests, which was established by the designer as part of Step III (cf. Fig. 1), is used to compute and finalise the results. This criterion includes different properties like the maximum acceptable distance from optimal path considering the vehicle behaviour is acceptable in terms of vehicle dynamics. This way, acceptable maximum system reaction times are obtained. In brief, we are able to report the corruption effects for fault forecasting and fault removal, as described in the next section.

3.2 Using Sabotage for an Early Safety Assessment

In this section, we explore a safety assessment process focused on supporting the creation and an early validation of a safety concept by using Sabotage. Figure 2 shows how the proposed approach can help to dimension the safety concept and to achieve its early safety validation. The approach is discussed in the following sequence in which the proposed safety assessment process is performed.

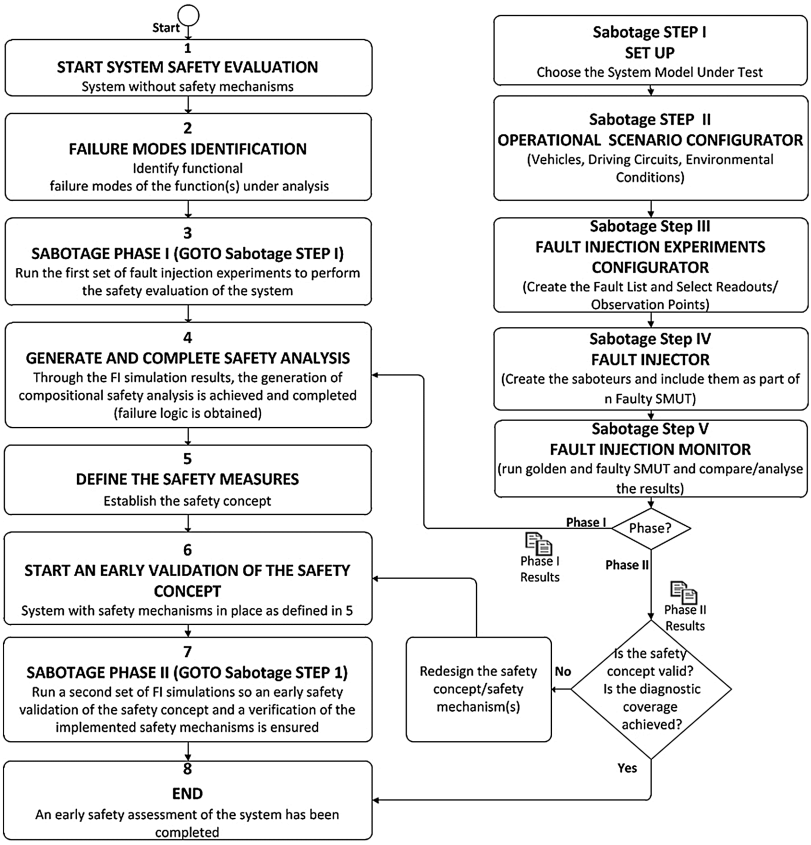

Fig. 2. Proposed early safety assessment flow by means of Sabotage

Sabotage Phase I: Start System Safety Evaluation. One assumption in this approach is that we start from existing system models (i.e. vehicle dynamics and system functions), which did not implement safety mechanisms before. The purpose of the process approach is to assist when creating the safety concept by including the vehicle dynamics. By applying this method, fault effects and the maximum response time of the system before a hazardous event occurs can be obtained. Furthermore, the severity of the injected failure modes can be quantified in terms of vehicle dynamics effect. Through these tangible outcomes, safety concepts and mechanisms can be more accurately

dimensioned. Before completing any kind of fault injection simulation, the target system must be defined and its main functions and failure modes (e.g. omission) have to be stated. Likewise, a preliminary architecture is mandatory in order to know which functional failure modes could lead to system or vehicle failure and hazards.

By following the approach explained in Sect. 3.1, a first run of fault injection simulations needs to be configured. For that, the vehicle and the operational situations shall be specified to select the vehicle and the driving scenarios. Some typical situation scenarios that must be analysed are for example:

- Location: highway, urban;
- Road conditions: uphill, on a curve;
- Environment conditions: good conditions, heavy rain;
- Traffic situations: fluent;
- Vehicle speed (kph);
- Manoeuvres: parking, overtaking, lane keeping;
- People at risk: driver, passenger, pedestrians.

We foster the idea of complementing traditional safety analysis techniques by performing fault injection already at the concept phase. Those traditional analysis techniques include FMEA (Failure Modes and Effect Analysis), FTA (Fault Tree Analysis), DFA (Dependent Failure Analysis) or Preliminary Hazard Analysis (PHA). All of them require not only knowing the failure modes but also to have a clear vision of the failure effect. Yet sometimes those failure effects might not be clearly known in advance. Our approach aims at deriving potential effects or hazards based on the FI simulation results. As a starting point of such preliminary analysis, the malfunctions or failure modes are clearly identified in this process step. As a result of this activity, a fault list can be created by the Sabotage framework. This activity addresses the following questions:

- Where should the faults be injected?
- What is the most appropriate fault model representing the functional failure modes?
- How should the faults be triggered within the system?
- Where should the fault effect be observed?

After generating simulation settings and creating the fault list, Sabotage executes the Golden (fault free) and Faulty simulations, as described in Sect. 3.1. A set of scripts have been developed to achieve the required level of automation. The pass/fail criterion of the simulation is defined as part of the Workload (cf. Fig. 2 Sabotage Step III). The user can set that requirement as a vehicle dynamics property violation. The main goal of comparing the Golden versus Faulty SMUT is to obtain the required results in an automated way.

The results of the simulation experiments (c.f. Fig. 2 Phase I Results) can then complete the safety analysis and help dimensioning the safety concept through the maximum system reaction time. In other words, it can be used to better determine the required level of fault tolerance (e.g. redundancy or graceful degradation). In brief, Sabotage helps to identify those hazards (e.g. vehicle does not turn when it should) or to rank the failure modes with respect to fault occurrence. Dynacar is used to visually observe these system failures through its 3D virtual environment. To sum up, by using

FI approach, it is possible to get data in order to forecast how the system will behave under the effects of real faults for situations in which no previous data was available. Considering engineering needs to prove that fault reaction times are shorter than the FTTI, a good estimation of those values is relevant. This ensures that any fault reaction is completed before a hazardous event occurs.

Sabotage Phase II: Start an Early Validation of the Safety Concept. Once the model-based safety architecture has been defined, the designer can obtain an early validation of the implemented safety concept. For that, a second run of fault-injection is needed. After having the safety concept defined and its architecture designed as part of the system behavioural model, a second run of the FI simulations can be performed. This allows validating the safety of the system during early design phases. For instance, if the needed diagnostic coverage it is not achieved, the corresponding design part must be rebuilt. Furthermore, possible systematic faults or the robustness of the implemented safety mechanisms can be tested. In this second phase, the user can establish pass/fail criteria upon the defined safety goals and safety requirements.

4 Case Study: Automated Lateral Control

4.1 Automated Driving Control Architecture

After outlining the Sabotage FI simulation framework, the feasibility of this method is demonstrated by applying it to a Lateral Control system. It is worth mentioning that no safety was considered when modelling the system. For that reason, the Sabotage method is applied starting from Phase I (safety evaluation).

The Lateral Control system is part of a complete control architecture for automated driving developed in Matlab/Simulink. This automated control architecture consists of two main systems: Lateral and Longitudinal control. The Lateral Control is the responsible for steering the vehicle along the most appropriate trajectory depending on the vehicle and environment state. This automated function consists of three principal functions.

- Behavioural Planner: It selects the most convenient trajectory depending on the vehicle manoeuvre (i.e. lane keeping, lane changing and obstacle avoidance). Behavioural Planner is composed of another three sub-functions (i.e. Perception, Local Planner and Decision). The Perception function supplies information from the vehicle state sensors and environment state sensor as Differential GPS (DGPS). It has to be pointed out that no sensor fusion is considered in the current design. The Local Planner receives information from the environment sensors to obtain vehicle position. At last, the Decision function creates the optimal trajectory considering the manoeuvre that the vehicle shall perform.
- Trajectory Controller: Keeps the vehicle correctly on the trajectory. Knowing the lateral error, the angular error and the curvature of the path, Trajectory controller calculates the Variation Correction "C_v". The algorithm chosen for the evaluation of this design is the so-called Control Law algorithm and it is defined as formula (1).

$$C_v = K_1 * e_{lat} + K_2 * e_{ang} + K_3 * Curvature \quad (1)$$

- The steering function: Controls the steering wheel to set the vehicle on the trajectory defined by Behavioural Planner. It obtains the input values from the Trajectory Controller.

4.2 Safety Evaluation of the Lateral Control System

Section 4.1 explains how the automated guidance must be performed synchronously with the longitudinal and lateral control. As functional safety is a crucial requirement, this section shows a safety evaluation for an existing preliminary Lateral Control System Behavioural Model. By performing the required simulations, the safety of the system is evaluated so that the most suitable safety concept is obtained. Hence, the following issues have been addressed:

- Simulation-based data acquisition with regards to component failure effects in the presence of real faults observed on vehicle level.
- Dimensioning the functional safety concept by applying the process explained in Sect. 3.2. This implies elaborating a fault tolerant Lateral Control system to avoid possible hazards and to ensure a high level of dependability through fail-operational behaviour or graceful degradation.

In Sect. 4.1, the main functions of the Lateral Control system have been introduced and the Malfunctions related to the Lateral Control system (cf. Fig. 2, Failure Modes Identification) consist of: Behavioural Planner (Unwanted Local Planner, Unwanted Perception, Unwanted Decision), Trajectory Controller (Omission, Commission) and Steering (Omission, Commission). These malfunctions or failure modes are necessary to derive a proper configuration of the required fault models (see Table 2, 2nd column). Those functional failure modes can be reproduced at system level (e.g. steering omission) or even as component level malfunctions (e.g. DGPS information omission). After completing that step, the step one of the first phase of Sabotage is applied. This requires the selection of the Lateral Control SMUT as the design-model for which the safety will be evaluated. In detail, the operational situation is specified: the speed of the vehicle is set to 45 kph maximum in a fluent urban traffic and performing a lane keeping manoeuvre on a curve at a city intersection.

With the aim of seeing the failure effects on vehicle level, Functional Failure Modes associated to the functions have been reproduced. The malfunctions are triggered while the vehicle is driving on a curve. In order to see system and vehicle level effects, functional failure modes related to the DGPS (Differential GPS) and the steering controller have been reproduced. The fault list (cf. Table 1) is specified as by following the template depicted in Sect. 3.1. Fault durations are randomly filled in as multiple of the simulator resolution (1 ms) and triggers are curve positions (X,Y).

Table 1. Example of a fault list generation

Component	Fault location (target signals)	Fault model	Fault duration	Fault trigger (X,Y)
DGPS	X,Y	FrozenLastValue	150 ms	10 m, 20 m
DGPS	X,Y	Delay	100 ms	20 m, 30 m
Steering ECU	Steering	FrozenLastValue	70 ms	30 m, 30 m

By employing the process described in Sect. 3.1, the saboteur blocks are automatically injected to the SMUT, through a custom Matlab script, which holds the proper configuration by means of the previously built up fault list. Together with the saboteurs, the read-out blocks are included as well. Then, the fault injection simulations are performed by triggering them at many driving circuit points on a curve to obtain the most critical ones (see Fig. 3).

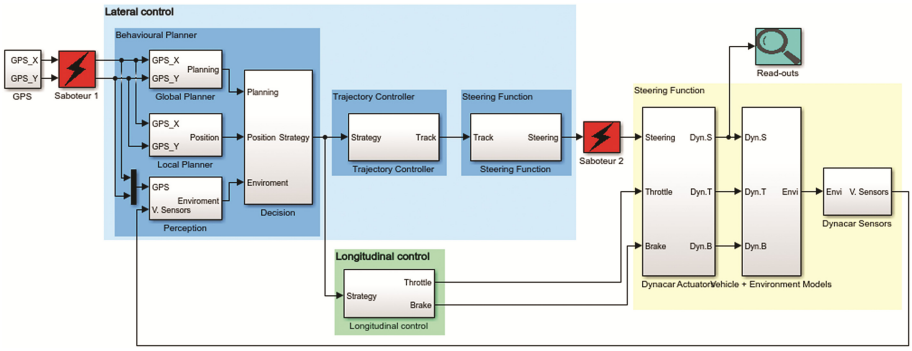

Fig. 3. Faulty system behavioural model

Table 2 lists the most relevant Potential Effects and Hazards obtained by FI simulation. For this purpose, the Lateral Control malfunctions are modelled by means of Fault Models which allow to obtain vehicle level effects and to get a more precise definition of the safety goals.

As depicted in Table 3, based on the identified hazards, appropriate safety goals have been derived. Figure 4 depicts how the maximum time before the vehicle dynamics are unsafely affected is obtained. More precisely, it illustrates a steering omission based on the Yaw Rate and the Lateral Error signals (observation points or read-outs).

Table 2. Hazard and potential effects obtained by FI simulation

Fault target	Fault model	Potential effects	Hazard
Steering	FrozenOutOfRange	Steering shaft is broken	Vehicle may perform a sudden steering and go out of control causing multiple collisions
	Frozen SteeringValueMax	Strong deviation of steering shaft position	Vehicle may perform an oversteering, spin and cause multiple collisions
	Frozen SteeringValueMin	Strong deviation of steering shaft position	Vehicle may perform an understeering and go out of control causing multiple collisions
	Frozen LastValue	Constant steering shaft position	Vehicle may depart lane due to blocked steering angle causing multiple collisions
Trajectory controller	Frozen LastValue	Constant Cv value	Vehicle may depart lane due to unwanted steering angle causing multiple collisions
	Frozen OutOfRange	Controller is saturated	Vehicle may depart lane due to unwanted steering angle causing multiple collisions
Behavioural planner	FrozenLastValue	The trajectory is not updated	Vehicle may follow a not updated trajectory
DGPS	Frozen DGPSLastValue	Behavioural planner is not updated	Vehicle may depart lane because of following an unwanted trajectory and cause multiple collisions
	Frozen RandomValue	Behavioural planner change the trajectory	Vehicle may perform a sudden steering, go out of control and cause multiple collisions

Table 3. Definitions of the lateral control safety goals

Safety goal ID	Safety goal definition
SG1	An unwanted steering angle shall be prevented
SG2	A sudden steering manoeuvre due to an unwanted trajectory shall be prevented
SG3	An unwanted behavioural planner shall be prevented

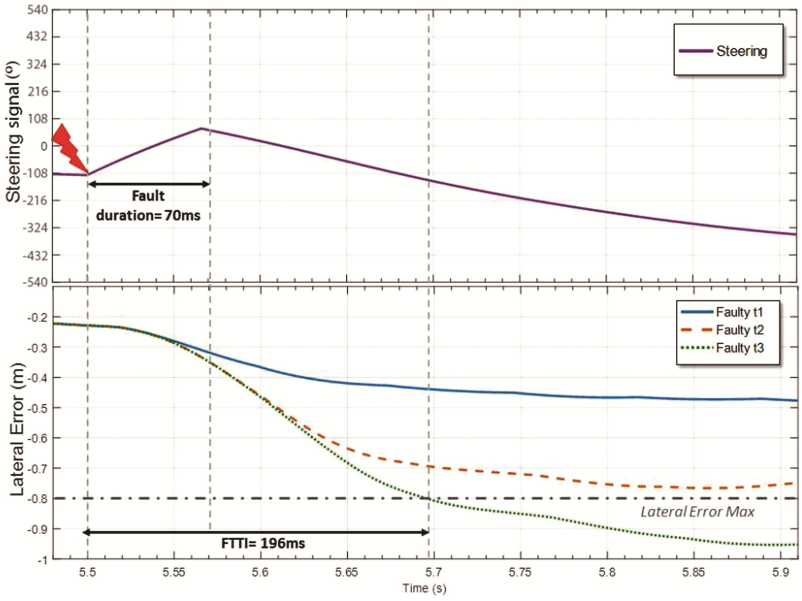

Fig. 4. Basis for FTTI calculation

To calculate the proper FTTI (Fig. 4) value by simulation, the maximum lateral error is taken into account as the pass/fail criterion. This value is analytically calculated using the formula (2). The following formula is used to determine where the vehicle dynamics is certainly affected:

$$LatError_{max} = \frac{(Lane_{width} - Vehicle_{width})}{2} = \frac{3,5 - 1,9}{2} = 0,8\,m \tag{2}$$

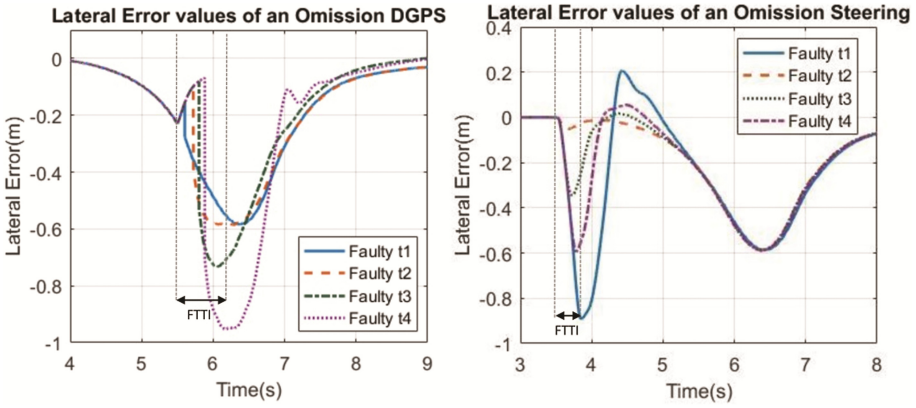

Fig. 5. Lateral error values for different steering and DGPS fault durations

Figure 5 depicts the collection of the results for faults introduced in X,Y DGPS signals and in the steering controller.

In addition, Table 4 represents how the FTTI values, the safety goals (Table 3) and the fault detection times (Fault duration) have been filled in based on FI experiments.

Table 4. Lateral control: hazard analysis

Function	Malfunction	SG	Safe state	FTTI (ms)	Fault duration (ms)
Steering	Commission	SG1	Ensure integrity of command execution	196	70
Trajectory controller	Commission	SG2	Graceful degradation and driver regains control	250	114
Behavioural planner	Commission	SG3	Graceful degradation and driver regains control	327	123

Table 5 lists the Functional Safety Requirements (FSR) based on FI simulations.

Table 5. Lateral control: definition of safety requirements

FRS	Related SG	Definition
FSR1	SG1	The system shall always assure that the yaw rate does not increase more than 16%
FSR2	SG1	The steering shall be fail-operational
FSR3	SG1	The operational state of the steering controller shall be monitored and reported to superordinate controller
FSR4	SG1	The system shall always assure that the range of the steering shall be between [–540, 540] degrees
FRS5	SG2	The system shall always assure that the LateralError value must be less than LateralErrorMax
FSR6	SG2	The system shall always assure that the Controller Cv parameter value is between [–1, 1]
FSR7	SG2	The operational state of the trajectory controller shall be monitored and reported to superordinate controller
FSR8	SG3	The trajectory shall be calculated based on the most appropriate manoeuvre (e.g. lane keeping, lane changing)
FRS9	SG3	The system shall assure that the new position (X_{t+1},Y_{t+1}) of the trajectory based on vehicle speed shall not exceed more than 9% of the current position(X_t,Y_t)
FSR10	SG3	The system shall calculate its trajectory starting from the most appropriate manoeuvre (e.g. lane keeping, lane changing)
FSR11	SG3	The operational state of the behavioural planner shall be monitored and reported to superordinate controller

In the selection of safety concepts, the main outcome is that a redundant steering is necessary in order to achieve the required level of availability. The main reason is that a failure related to the steering shall be detected within 70 ms and the availability must be provided within 196 ms. Regarding failures coming from the Trajectory Controller, the detection time is established at 114 ms and the maximum system failure reaction time in terms of vehicle dynamics is 250 ms. Because of these timing results, graceful degradation might be sufficient in this case. Behavioural Planner malfunctions shall be detected in less than 123 ms and their effect on vehicle level controlled within 327 ms. The same applies to the Behavioural Planner for which graceful degradation might be sufficient.

5 Conclusion and Future Work

We have presented a simulation-based FI approach for an early safety assessment of automotive systems. Our approach has been evaluated in a case study for the model-based design of a Lateral Control system. From a novelty standpoint, we focused on determining of the fault detection time interval for permanent faults based on the maximum lateral error, as a vehicle dynamics property. A major strength of the method introduced in this paper is its usage during early design phases to evaluate the safety of the system. This allows dimensioning and trading-off between safety concepts and performs an early safety validation of the design. The uncertainty related to some automotive systems, such as an automated vehicle, makes traditional safety analysis methods definitely not sufficient, requiring additional virtual and simulation solutions. Forthwith, FI establishes itself as a way of completing and verifying previously carried out safety analyses. Given that analysing system reactions under the effects of real faults can a burden-some issue, these FI experiments arise as a viable solution.

Our future work spans the spectrum from relaxing the fault simulation constraints to instrumenting the automated assessment work. This includes: (1) adding the capability of automatically collapsing the injection of faults to generate optimised fault lists, (2) integrating with contract-based approaches, (3) connecting to other system modelling environments such as Papyrus/SysML, (4) linking to model-based safety analysis tools, and (5) comparing FI simulation results with the results of performing software fault injection for a model-car.

Acknowledgments. The authors have partially received funding from the ECSEL JU AMASS project under H2020 grant agreement No 692474, the UnCoVerCPS project under H2020 grant agreement No 643921 and MINETUR (Spain).

References

1. Koopman, P., Wagner, M.: Challenges in autonomous vehicle testing and validation. In: 2016 SAE World Congress (2016)
2. ISO 26262: Road vehicles – Functional safety, International Organisation for Standardisation (ISO) (2011)

3. Pintard, M.L.: Des analyses de securite a la validation experimentale par injection de fautes - le cas des systemes embarques automobiles. Ph.D, Institut National Polytechnique de Toulouse (2015)
4. Pena, A., Iglesias, I., Valera, J., Martin, A.: Development and validation of Dynacar RT software, a new integrated solution for design of electric and hybrid vehicles. In: EVS26 Los Angeles (2012)
5. Ruiz, A., Juez, G., Schleiss, P., Weiss, G.: A safe generic adaptation mechanism for smart cars. In: IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 161–171, Gaithersbury, MD (2015). doi:10.1109/ISSRE.2015.7381810
6. Ziade, H., Ayoubi, R., Velazco, R.: A survey on fault injection techniques. Int. Arab J. Inf. Technol. **1**(2), 171–186 (2004)
7. Benso, A., Di Carlo, S.: The art of fault injection. J. Control Eng. Appl. Inform. **13**(4), 9–18 (2011)
8. Svenningsson, R.: Model-implemented fault injection for robustness assessment, Licentiate Thesis, Stockholm (2011)
9. Vinter, J., Bromander, L., Raistrick, P., Edler, H.: Fiscade - a fault injection tool for SCADE models. In: Automotive Electronics 2007 3rd Institution of Engineering and Technology Conference, pp. 1–9 (2007)
10. Silveira, A., Araujo, R., De Castro, R.: FIEEV: a co-simulation framework for fault injection in electrical vehicles. In: 2012 IEEE International Conference on Vehicular Electronics and Safety, ICVES 2012, pp. 357–362 (2012)
11. Jones, S., Armengaud, E., Böhm, H.: Safety simulation in the concept phase: advanced co-simulation toolchain for conventional, hybrid and fully electric vehicles. In: Fischer-Wolfarth, J., Meyer, G. (eds.) Advanced Microsystems for Automotive Applications 2014. Lecture Notes in Mobility, pp. 153–164. Springer, Switzerland (2014)
12. Folkesson, P., Ayatolahi, F., Sangchoolie, B., Vinter, J., Islam, M., Karlsson, J.: Back-to-back fault injection testing in model-based development. In: Koornneef, F., Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9337, pp. 135–148. Springer, Cham (2015). doi: 10.1007/978-3-319-24255-2_11
13. Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.C., Laprie, J.C., Martins, E., Powell, D.: Fault injection for dependability validation: a methodology and some applications. IEEE Trans. Softw. Eng. **16**, 166–182 (1990). Fault injection for dependability validation: a methodology and some applications
14. Algirdas, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. **1**, 11–33 (2004). doi:10.1109/TDSC.2004.2