Validation of safety necessities for a Safety-Bag component in experimental autonomous vehicles

Manel Brini, Paul Crubille, Benjamin Lussier, Walter Schön

Validation of safety necessities for a Safety-Bag component in experimental autonomous vehicles

Manel Brini*, Paul Crubillé*, Benjamin Lussier* and Walter Schön*
*Sorbonne universités, Université de Technologie de Compiègne, CNRS,
Heudiasyc UMR 7253, CS 60 319, 60203 Compiègne Cedex France
Email: {manel.brini,paul.crubille,benjamin.lussier,walter.schon}@utc.fr

***Summary*—This work presents a study to improve the safety of experimental autonomous vehicles in the Heudiasyc laboratory. This work presents risk analyses showing that the use of our vehicles involves significant risks during experiments, and that integrating an Independent Safety Component called Safety-Bag in the vehicle architecture can significantly reduce these risks. The Safety-Bag carries out the on-line verification of safety necessities by checking the vehicle's current state with safety rules and taking or disabling actions to ensure a safe behavior. In our work, we present and we apply two methods for risk analysis (FMEA and HazOp-UML) to design these safety necessities in the case of experimental autonomous vehicles. We also present the validation of two safety necessities through fault injection experiments with a robotized Fluence vehicle and a vehicle in the loop testbed.**

***Keywords*—Safety-Bag, dependability, safety, fault tolerance, FMEA, HazOp-UML, autonomous vehicles.**

I. INTRODUCTION

Autonomous automotive vehicles are mobile robots which evolve in an open environment and need to respect strict traffic rules. They should also be able to detect and react to dangerous situations on their own. Depending of their autonomy's level, they will have to carry out their mission in unknown surroundings and perform various complex tasks. In order to do that, they use artificial intelligence software based on declarative mechanisms, which cause their validation and verification to be difficult. In addition, autonomous vehicles are fast and powerful mobile robots, able to accumulate a large amount of energy. Thus, they are a source of danger to their operators and their environment. Guaranteeing their safety still remains a technological challenge for their industrialization. In this paper, we integrate an Independent Safety Component called Safety-Bag, to reduce risks from the complex and incompletely validated control systems. The Safety-Bag allows software and hardware fault tolerance but needs implementable safety necessities to detect problems and put the system in a safe state. After this introduction, the second section of this article presents a state of the art first on safety in autonomous vehicles and second on the safety bag component. The third section briefly presents the experimental autonomous vehicle that we are working on, and the implementation of its Safety-Bag. The fourth section presents the design process that we propose for the safety necessities of our Safety-Bag, that is the application dependent rules that will define its behavior. This design is based on using diversified risk analysis methods, in our case FMEA (Failure Mode and Effects Analysis) and HazOp-UML (HAZard and OPerability study-Unified Modeling Language). Section V describes how we determined the safety necessities for autonomous automotive vehicles from the risk analyses, as well as a comparison of the necessities obtained from each analysis. The last section presents our approach to validate our work using fault injection on a dangerous experimental vehicle in the most representative and safe way possible. This validation is done using the real autonomous vehicle in a vehicle-in-the-loop testbed using simulation to send data to the operator and part of the vehicle's sensors, while real motors affixed to the wheels simulate virtual world efforts. The paper ends with conclusions and perspectives for further work.

II. STATE OF THE ART

We present in this section a state of the art on dependability in autonomous automotive vehicles. We then introduce the Safety-Bag component and present previous works and applications of this component. Concerning the dependability domain, its means and its threats (faults, errors, failures), this paper will adopt the concepts and definitions presented in [1].

A. Dependability in autonomous automotive vehicles

In the last decade, autonomous automotive vehicles made significant progress. In 2003, the Defense Advanced Research Projects Agency (DARPA) announced the First Grand Challenge aiming to develop autonomous vehicles capable of navigating desert rails and roads at high speeds. [2] describes the winner of the 2007 DARPA Urban Grand challenge: *Boss* an autonomous vehicle capable of driving safely in traffic at speeds up to 48 km/h. It uses on-boards sensors to track other vehicles, detect static obstacles and localize itself relative to a road model. However, before being used in critical application such as everyday driving, autonomous vehicles need to improve their dependability, that is the ability to deliver service that can be justifiably trusted [1]. In particular, their complexity and the use of declarative programming and AI (Artificial Intelligence), necessary to evolve in an unstructured open and partially unknown environment, raise the problem of their safety, that is the absence of catastrophic consequences due to their behavior on the user and the environment.

According to [3], safety is indeed a technological barrier to the industrialization of autonomous vehicles. To answer this problem, the NHTSA (National Highway Traffic Safety

Administration) agency [4] asks for a process that should include a hazard analysis and safety risk assessment. It should particularly contain design redundancies and safety strategies able to put an autonomous vehicle in a safe state when critical hardware or software failures errors occur. This fall back should also facilitate the transition from autonomous to manual driving when the system is not able to function properly on its own. In [5], dependability techniques are classified as either fault tolerance techniques or robustness techniques according to the class of hazards being addressed.

- Fault-tolerance techniques aim to avoid system failures in the presence of faults affecting system resources (such as sensor failures, software design faults, etc.).
- Robustness techniques aim to avoid system failures in the presence of external faults, environment uncertainties and contingencies.

In [6], Xu and Yuan propose an active safety system for autonomous vehicle, that has been identified as an effective technique for avoiding accidents. It consists of a motion planner and controller based on robust control rather than on dependable techniques. Another robust approach for the safety assessment of planned trajectories of autonomous vehicles is presented in [7]. This approach considers vehicle dynamics, interaction between traffic participants, and lane changes in multi-lane traffic. However, to our knowledge, fault tolerance is rarely mentioned in autonomous cars. Although some techniques for error detection (such as the Safety-Bag's rules and a watchdog timer) or error recovery (implemented through positioning in a safe state or a software reconfiguration) focus on autonomous robots, they are not yet applied on the autonomous cars. Jamil *& al.*[3] propose a hazard analysis conceptual framework for an Autonomous Automotive Cyber-Physical Systems (A^2CPS). This component continuously checks the system's state with a Hazard Database to detect potential risks and reacts accordingly, in a way similar to a safety bag component. However, no real implementation of such a component is presented yet. Perhaps the most studied dependability mean for autonomous vehicle is fault elimination through testing. For example, [8] presents the limitation of the existing reliability and safety engineering tools for autonomous Unmanned Ground Vehicles (UGVs) and proposes a novel methodology based on statistical testing in simulated environment. However, note that in [9], the authors postulate that testing alone is unfeasible to ensure a sufficient dependability in autonomous vehicles.

B. Safety-Bag for critical complex system

Independent from the system that it supervises, a Safety-Bag or *Independent Safety Component* is responsible for intercepting the system's commands and enforcing safety rules to avoid catastrophic failures. To reduce common cause failures, it must be specified and developed independently from the functional system, and have means of action and detection independent from the faults to be tolerated. It monitors the operational system, and in case of danger puts the system in a safe state [5].

This approach was used effectively for critical applications. For example, the ELEKTRA (Electronic Interlocking System)rail system [10] contains a logic channel that processes commands, and a safety channel that checks the commands according to safety rules. The SPAAS (Software Product Assurance for Autonomy On-board Spacecraft) project for an autonomous spacecraft [11] is composed of a Safety-Bag in charge of monitoring on-board a set of safety properties, and a *plausibility checker* that validates procedures on a ground station before they are uploaded and executed on-board. The SPIN nuclear plant monitor [12] controls physical parameters such as the pressure of the steam or the temperature of the core and has an architecture divided in two stages: acquisition and control. A mobile robot excavator [13] contains a safety manager component, responsible for all safety aspects of the system during operations, and is central to the control system's architecture. The R2C (Requests & Resources Checker) component [14] contains a state checker, that checks acceptable and unacceptable states in the system according to a set of predefined rules. All these Safety-Bag components require a set of application dependent rules in order to check the system's commands safety and determine how to react. Diversified analysis methods are proposed in this article to define these rules. In a complementary way, the SMOF (Safety MOnitoring Framework for Autonomous System) [15] proposes a technique using formal methods to guarantee the validity of each rule and that no emergent behavior may place the system in a dangerous state. However in an open environment with extremely diverse situations, formal analyses may well be a very complex and costly task.

III. SAFETY-BAG COMPONENT

We present in this section the Safety-Bag component designed and implemented for our experimental autonomous cars.

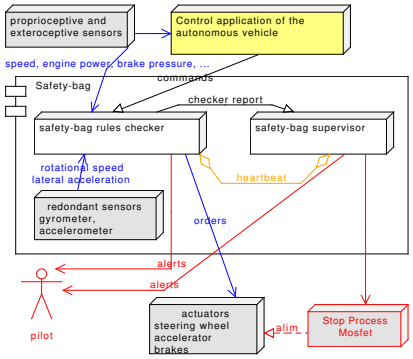

Figure 1. Safety-Bag implementation on our autonomous car

To know the state of the system, the Safety-Bag uses at the same time some of the car's sensors (such as the speed and steering angle sensors) and specific redundant sensors (such as the gyrometer and accelerometer). Hardware faults in the redundant sensor can be detected by the safety bag, so their choice among all of the car's sensors is a classical cost/safety compromise. The Safety-Bag filters the commands from the software components of the control application, verifies the

respect of the safety rules and is thus able to detect anomalies. It then recovers through a safety action by enforcing some commands (such as braking) or rejecting dangerous commands (such as inhibiting the acceleration). Adding any component in a system, even a fault tolerant one, can introduce new possible faults and failures. As a single fault should not cause failure of the whole Safety-Bag, it is composed of two functionally diverse computers monitoring each other. As shown in figure 1, one is called the *Safety-Bag rules checker*, and the other the *Safety-Bag supervisor*. The checker verifies the commands from the control application against a set of safety rules, allowing them if considered safe, while blocking them or even taking safety actions otherwise. We call safety necessity a safety rule combined with its possible safety responses. A safety necessity is typically expressed as a if-then-else condition.

The checker and supervisor monitor each other though the exchange of *heartbeats*. If the Safety-Bag supervisor no longer receives the signal from the Safety-Bag rules checker, it raises alarms and disables the control application commands as the autonomous behavior of the system can no longer be guaranteed. If the Safety-Bag rules checker no longer receives the signal from the Safety-Bag supervisor, it advises the driver to stop the experiment even if there is no immediate risk, as the Safety-Bag is now vulnerable to a single fault. The Safety-Bag can trigger visual and auditory alarms. It records all events related to safety, which allows post-experimental analysis.

IV. DETERMINATION OF SAFETY REQUIREMENTS

This section presents our design process for safety requirements, which are needed to define safety necessities. We present both FMEA and HazOp-UML risk analysis methods, but other diversified methods could also be used to diminish the risk of forgetting or improperly formulating safety requirements.

A. Design process

As a design process for safety requirements, we propose to use diversified risk analysis methods, each one implemented by a different team of safety experts when possible. This diversified process has two main reason. First, as risk analyses can only be done by human experts, using diversified methods (and even better different teams) is a well known technique to diminish errors and oversights in an analysis. Second, each risk analysis methods specializes on a particular aspect, and using diverse complementary analyses will allow us to produce as many safety requirements as possible. In this paper, we chose to use the FMEA [16] and the HazOp-UML analysis [17]. The FMEA is a well known method that focuses on every single failure in every component, while the HazOp-UML analysis focuses more on the system's process and its interactions with the environment. Figure 2 presents our design process using these two methods. Other complementary analyses could be added, such as the fault tree analysis that allows to identify the consequences of multiple failures (unlike the FMEA). In the FMEA, we first identify the system's component and subcomponents, and then determine all possible failures of these components and their effects on the system. Safety requirements are deduced from these informations by safety experts, and we propose to identify the Safety-Bag's necessities from the safety requirements as presented latter in section V.

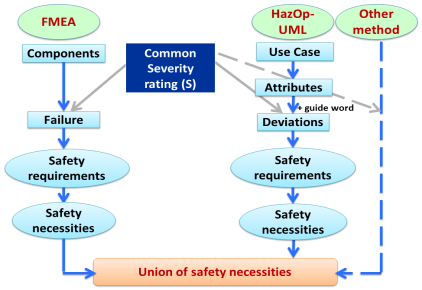

Figure 2. Safety necessities design process

HazOp-UML is a technique based on the description of the system and its actors. In our case, the actors can be: the driver, the pedestrians, the other vehicles, etc. The HazOP-UML analysis focuses on the system's processes and its relations with the different actors. First, we identify a set of use cases, that are the system functionalities. For each use case, we determine its attributes: preconditions, invariants and post-conditions that defines the functional limits of the use case (for example the minimum or maximum speed) but also requirements for a correct behavior (the correct input of a particular sensor, or the absence of a vehicle on a particular lane). Then, we apply the HazOp method by associating each attribute with a list of guide words as mentioned in [18] to guide the process and to identify all the potential dangers that are named *deviations*. Finally, we identify the safety requirements from each deviation and then determine the safety necessities.

B. Risk analysis methods

The first step in risk analysis consists in defining a table of severity rating (shown in figure 2). This is used to rate each component's failure or deviation in the analyses, but also to guide risk reduction approaches.

In our severity table, there are five levels : from nominal level 0 to most severe level 5. In level 0, the system is in nominal operation. In level 1, experimentations should be stopped, but the application is still capable of properly driving the vehicle. In level 2, the time needed for the driver to take over is sufficient to not pose any difficulty. In level 3, the control application has stopped and the vehicle is no longer controlled while an alarm is raised. In level 4, the application has stopped and the vehicle is no longer controlled but no alarms are raised. Finally in level 5, an invalid command is sent or maintained on the actuators; it is extremely difficult for a driver to regain control.

We present now the results from our two risk analyses: FMEA and HazOp-UML.

1) FMEA: Our FMEA focuses on 11 different components of our experimental vehicle and identifies 21 different failures. From these failures 5 are of severity 5, 4 of severity 4 and 3 of severity 4 or 5, which confirms that our experimental vehicle is a dangerous system as numerous single component's failures can lead to catastrophic consequences. Figure 3 is an extract from the FMEA table. We introduced additional columns to distinguish the failure's effects on the control component and the final effects on the vehicle's behavior. We also added a column describing the safety requirements that can protect from this failure, which is a common practice in the industry. In addition, we introduced in the table sub-columns to give indicative values for both response time and detection time. The failure rate λ (*failure probability per hour*) is difficult to identify accurately without performing a huge number of experiments. In most cases, the given values are best guesses from an engineering point of view. For hardware failures, we took them as pessimistic values, from our knowledge of the system and its components. For software failures, experimental components are developed research context and can be tested only briefly before being integrated into the system. The practice leads us to believe the failure rates of such components are significantly high. As an example, in the first line of the figure 3, a locked down failure in the control application leads to an uncontrollable vehicle as the actuators commands are not updated. The accident may become unavoidable before the driver is able to resume manual control. The severity of such a failure is 5. The corresponding safety requirement states that the system needs to be able to detect failures of the control application and subsequently block automated commands while warning the driver to regain manual control.

2) HazOp-UML: To identify a relevant set of use cases to perform a HazOp-UML analysis for autonomous vehicles, we relied on the list of conditions cited on pages 28 and 29 of [4], pending the production of similar recommendations by the European authorities. From 28 situations described in this document, we have identified a total of 25 use cases: 8 generic use cases related to our autonomous vehicle, and 17 use cases extending 3 of these generic use cases. In this article, we detail in particular the use case *Follow a kinetic trajectory*[1]. The list of associated attributes is composed by:

- 4 pre-conditions (e.g., the kinetic state [2] estimated by the vehicle is equivalent to the real kinetic state.),
- 3 invariants (e.g., the vehicle kinetic state estimated by the vehicle remains at a limited distance from the kinetic trajectory.),
- And, 3 post-conditions (e.g., the vehicle knows the next kinetic trajectory to be followed.).

In the HazOp-UML table figure 4, we have associated with each deviation its consequences and possible causes, their severity and finally the safety requirements needed to protect from those consequences. Unlike the method proposed in [18], we do not determine directly the safety necessities for the Safety-Bag, but we begin by identifying safety requirements related to the vehicle behavior's deviation, in a similar way to safety analysis carried out by engineers in the FMEA method. Indeed, all safety requirements may not be implemented by the Safety-Bag component, and other mechanisms will be required when it can not. For instance, verifying the car's available driving space can only be carried out with a data fusion mechanism, which at the moment can not be justifiably trusted enough to be implemented in the Safety Bag.

[1]Kinetic trajectory: The kinetic trajectory is successive sets of position, attitude and speed of the vehicle that describes its intended immediate movement on the road.

[2]Vehicle kinetic state: The vehicle kinetic state consists of the position, the attitude and the speed of the vehicle.

V. DETERMINATION OF SAFETY NECESSITIES, COMPARISON AND SYNTHESIS

Once risk analyses of the system have been performed, we need to determine the safety necessities from the safety requirements. Safety necessities are derived from safety requirements using the knowledge and experience of safety experts, and will subsequently be implemented as safety rules and safety actions in the Safety-Bag.

1) With FMEA: For each safety requirement (as shown in figure 5), the means of its implementation are detailed, and if the Safety-Bag is considered as one of these means, a safety necessity is identified. A safety necessity specifies what the Safety-Bag should observe, how it identifies failures from observations, and what it should do once a fault is detected.

Note than all safety requirements cannot be implemented by the Safety-Bag. The Safety-Bag may for example lack the needed sensors to detect a component's failure. Particularly, the checks to be carried out may be too complex to be performed by the Safety-bag. Indeed, the Safety-Bag must remain sufficiently simple to be validated easily, and thus have a similar form to conditional blocks in an imperative language (if-then-else clauses).

2) With HazOp-UML: Safety necessities are deduced from the safety requirements in the exact same way than with the FMEA.

As stated before, some requirements cannot be implemented by the Safety-Bag. For example, ref. 3 from figure 6 would require complex comparisons between the sum of the vehicle perception and the inboard map. Guaranteeing the dependability of such a function is a hard (or even near impossible) task, and it thus cannot be incorporated into the Safety-Bag.

3) Comparison: In this section, we compare the results obtained by the two methods, using the full risk analyses from which Figures 3 to 5 are extracted, and their quantitative results are summarized in Figure 7.

Indeed, FMEA and HazOp-UML have the same main goal to identify unacceptable risks to the system and its environment, as part of a process to reduce them to an acceptable level. However, the FMEA method focuses on internal components of the system (in our case, experimental software and hardware components), while HazOp-UML focuses more on the vehicle's functions process and its interactions with the environment. This explains the fact that there are common

Elements	Types of failure	Computing effects	Vehicle effects	λ (/h)	Detection		Action		Severity of consequences		Safety requirements	Ref
					mean	t	mean	t	S	Comments		
Control Application's hardware	Locked down	The last outputs are maintained.	No control: acceleration, brake and steering torque unchanged.	~ 10^{-3}	driver	>2s	driver	+ 2s	5	Uncontrollable vehicle	The system must detect the control application failure and put the system in a safe state.	1
	Functional error	Outputs set to 0	- No acceleration - No autonomous brake - Electric Power Steering is in default, dropped wheel	~ 10^{-5}	driver	>4s	driver + Stop Process Button	+ 2s	4	The Electric Power Steering stays in a failure state.	The system must detect the functional errors and put the system in a safe state.	2

Figure 3. An extract of FMEA for autonomous vehicles without Safety-Bag

Project: Safety-Bag HazOp table		Use case 1: Follow a kinetic trajectory 4 Preconditions ; 3 Invariants ; 3 Post-conditions • PrC1: The kinetic state estimated by the vehicle is equivalent to the real kinetic state. • PrC2: The kinetic state used by the vehicle's software is not erroneous.						
attribute	Guide Word	Deviation	System effects	Use case effects	S	Possible causes	Safety requirement	Ref
PrC1	No/None	The vehicle kinetic state estimated by the vehicle is unknown.	The control application can return an error, the vehicle may lose control.	The vehicle can no longer follow the kinetic trajectory.	5	Hardware or software failure	The system should verify and ensure that the vehicle kinetic state is determined.	1
PrC2	Other than	The vehicle kinetic state estimated by the vehicle is not at the beginning of the kinetic trajectory.	The vehicle does not know which command to apply or tries to join brutally the trajectory.	Impossible to follow the kinetic trajectory	4 or 5	Actuator or software failure	The actuators and the control application must correctly operate.	5

Figure 4. An extract of HazOp-UML

Elements	Types of failure	Safety requirements	Means of safety requirements implementation	Safety necessities	Comments	Ref
Control Application's hardware	Locked down	The system must be able to detect the control application failure and put the system in a safe state.	The Safety-Bag can supervise the update of control application.	The Safety-Bag should check the liveliness of the control application. If the last command update is too old, the Safety-Bag triggers the alarms, inhibits the controls and brakes the vehicle.	To ensure redundancy, the Safety-bag consists of two computers that exchange a heartbeat.	1
	Functional error	The system must detect the functional errors and put the system in a safe state.	The Safety-Bag can supervise the control application with coherence tests.	• The Safety-Bag should check the temporal consistency commands to be applied. Otherwise, it must put the system in a safe state by raising alarms and returning to manual mode. • The Safety-Bag should check that a safe speed limit is not exceeded. Otherwise, it raises alarms and inhibits the acceleration commands.	The Safety-Bag will be able to detect only a subset of the control application's functional failures or later their consequences on the vehicles' dynamics.	2

Figure 5. Safety requirements and necessities from FMEA

Ref	Safety requirements	Means of safety requirements implementation	Safety necessities	Comments
1	The system should verify and ensure that the vehicle kinetic state is determined.	The Safety-Bag can verify that the vehicle kinetic state is updated.	The Safety-Bag should verify that the vehicle kinetic state is updated. Otherwise, it raises alarms and gives back control to the driver.	The vehicle kinetic state supplied to the Safety Bag must be regularly updated.
3	Ensure that what the vehicle perceives and the road maps are consistent.	Comparison between what the vehicle perceives and road maps.	_	It is complex to verify this safety requirement with the Safety-Bag.
5	The actuators and the control application must correctly operate.	Diagnostic mechanisms of actuators and Control Application, and put the system in the safe state. A part is performed by the Safety-Bag.	A) The Safety-Bag shall check that the brake, the acceleration and the direction correctly operate. B) The Safety-Bag must verify for the control application the : • liveliness, • the temporal coherence, • the speed limit Otherwise, the Safety-Bag raises alarms and gives back control to the driver. In the last case, it also inhibits the acceleration and stops the vehicle.	A) The Safety-Bag can directly verify the intensity in the electric motor of the vehicle and the steering assist motor. B) The Safety-Bag will only detect a subset of the control application's faults.

Figure 6. Safety requirements and necessities from HazOp-UML

elements in both analyses, but also different ones. For example, the safety requirements from HazOp-UML, ref. 1 figure 4, does not correspond to any safety requirements derived from the FMEA. Conversely, some components that are mentioned in the FMEA (ref. 1 figure 3) are not included in HazOp-UML. In regards to common elements, we identified six similar necessities, such as ref. 2 figure 3 for FMEA and ref. 5 figure 4 for HazOp-UML. Finally, it should be noted that the results of the HazOp-UML analysis depend in part on design choices in representing and expressing the UML use cases and attributes, and thus on the person who realizes it. Another difference is that HazOp-UML explicitly studies the system's processes and their evolution in time, while the FMEA focuses on failures of the system's components at a given instant. As previously stated, we found 8 similar safety necessities in both analyses. Those safety necessities correspond to the failure of components that were explicitly targeted in the HazOp-UML analysis, and naturally covered in the FMEA. However, note that these safety necessities needed an analysis similar to the FMEA and that the FMEA still identified three

safety necessities on other components. Therefore, the two techniques appear to us as complementary.

	FMEA	HazOp-UML *(1 use case)*
Number of failures/deviations	*21*	*42*
Safety requirements	*10*	*21*
Safety necessities	*11*	*14*
Safety necessities similar to the other method	*8*	*8*

Figure 7. Quantitative results of both analyses

VI. EXPERIMENTAL VALIDATION

In this section, we validate our Safety-Bag using fault injection and a VITL (Vehicle-In-The-Loop) testbed. Autonomous vehicles are integrated systems in which the interactions between components (both software and hardware) have a significant impact on the system's behavior (communication delays, time needed for the mechanical brake to operate, etc.). We thus believe that VITL validations have a significantly more realistic representativity than SIL (software-in-the-loop) and MIL (modeling-in-the-loop). We implemented in our safety-bag the five following safety necessities from the 18 obtained in the previously presented analyses:

1) The Safety-Bag checks the liveliness of the control application ; when the latter is not responding, the Safety-Bag raises alarms for the driver to take back the controls and brakes while slowly reducing the steering wheel angle.
2) The Safety-Bag checks the temporal consistency of the control application's commands. When a significant number of commands have been successively received in a wrong order, it warns the operator and reduces the vehicle's speed in the same way that in the safety necessity 1.
3) The Safety-Bag checks whether the vehicle's speed does not exceed a safety speed of 50 km/h. When it happens, the Safety-Bag inhibits any acceleration commands from the control application until the vehicle's speed is again under the safety speed value.
4) The Safety-Bag checks whether the steering angle and the vehicle's speed are consistent with the vehicle dynamics. If the steering angle exceeds a value depending of the vehicle's speed, the Safety-Bag reacts by alerting the operator and trying to reduce speed and angular velocity by braking.
5) The Safety-Bag checks that the speed information from the control application and the speed known by the Safety-Bag (via the CAN bus) are compatible. The Safety-Bag reacts by raising alarms and inhibiting the acceleration until the operator takes back the control of the vehicle.

In the following sections, because of a lack of space, we focus only on the first safety necessity. We first present our experimental platform: a real robotized car on a VITL testbed. We then present the nominal activity of our experiment, then the injected fault, and finally our validation results.

A. VILAD testbed presentation

As shown in figure 8, the VILAD (the VITL of the Heudiasyc laboratory) testbed integrates a vehicle simulator controlling force feedback motors which opposes the power of the engine and those of the brakes to simulate a real vehicle's efforts. It allows either a human driver or the command

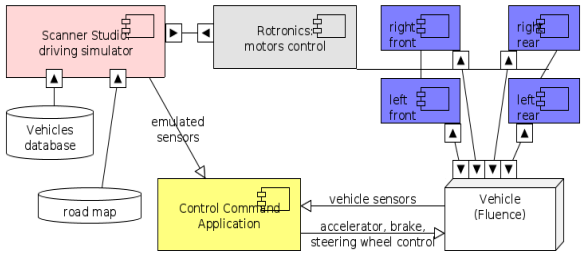

Figure 8. VILAD testbed

application to control the automated IRIS car in a virtual world. IRIS is based on Renault's Fluence electric car. The brake and acceleration are controlled by analog signals. The vehicle is able to accelerate from 0 km/h up to 50 km/h in 4 seconds and can reach 130 km/h. Part of the sensors such as the inertial sensors and lidars are simulated, while others sensors data such as the car's or the motor's speed can be used as such. In these experiments, we used scripted commands rather than a real command architecture for three reasons. First, we had not developed the complete control architecture on our experimental vehicle yet. Second, fault injection is more easily done with scripted commands, as we can simply modify the script to simulate frozen or locked applications. Third, even though our testbed is much more safe than the real robotized vehicle, injecting fault in such a system is still a very dangerous task, especially for the operator, and using scripted commands gives much more controllability and safety in our experiments. We focus here on the longitudinal control: accelerations and brakes.

B. Runway followed by the vehicle and nominal scenario

Figure 9 represents the virtual runway for our experiments. The nominal scenario follows the white trajectory, from the starting point to the destination. Figure 10 presents the command applied for the longitudinal control of the vehicle during the nominal scenario. A value between 0 and 1.8 volts commands the Fluence acceleration. A value between 0 and -2 volts is transmitted to the braking system.

C. Faults injection experiments

We inject two different faults, each in one experiment, by modifying the commands sent by our simulated command application. The first fault consists in simulating a locked application that do not change the last applied command, while the second fault consists in sending an erroneously high acceleration command. The injected faults can be seen on figure 10.

The faults are injected at time 93s into the experiments (at the place seen on figure 9), but do not propagate into failures

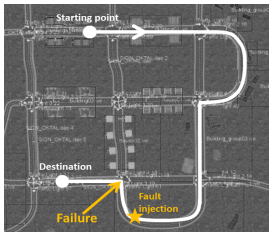

Figure 9. Simulated environment and vehicle trajectory

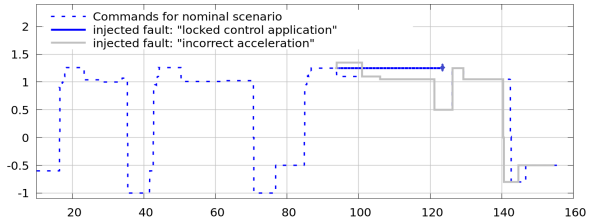

Figure 10. Nominal commands and injected faults from the control-command bloc

until the next left turn. Note that in the first injection (locked

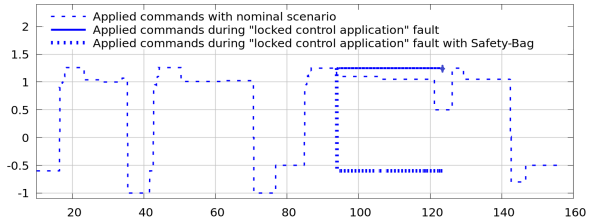

Figure 11. Commands applied to the actuators for the locked control application fault

control application), the commands stop at 123s because the operator had to stop automated control of the vehicle due to its dangerous behavior. In the second injection (incorrect

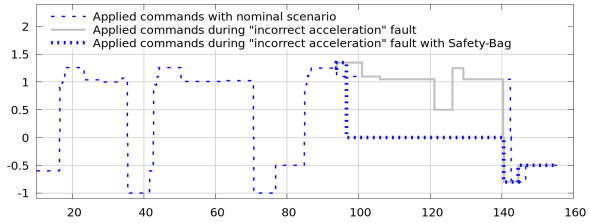

Figure 12. Commands applied to the actuators for the incorrect acceleration fault

acceleration), we had to change the script at 140s to brake before the nominal scenario because the vehicle was farther on the road due to the incorrect acceleration.

D. Result and analysis

This section presents results for both injected faults on our VILAD testbed. For each fault, figures 11 and 12 show the longitudinal commands applied to the car's actuators (acceleration and brake) in three cases: (1) the nominal case, (2) with the specified injected fault and no Safety-Bag, (3) with the specified injected fault in presence of a Safety-Bag. The difference in the applied commands between (2) and (3) is due to safety actions taken by the Safety-Bag after a safety rule has been identified as violated.

1) Locked control application fault: Without the Safety-Bag, we see in the figures 13 and 14 that the vehicle speeds up to 80 km/h before the turn. The following brutal slowdown

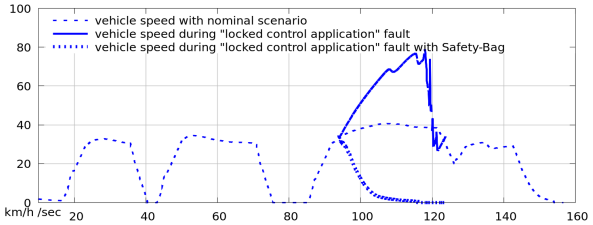

Figure 13. Vehicle speed in the case of locked control application fault

is due to the vehicle slipping on the road before being stopped by the operator. The distance to the trajectory jumps to almost 6 meters, showing that the vehicle clearly went out of control. With the Safety Bag, the frozen application

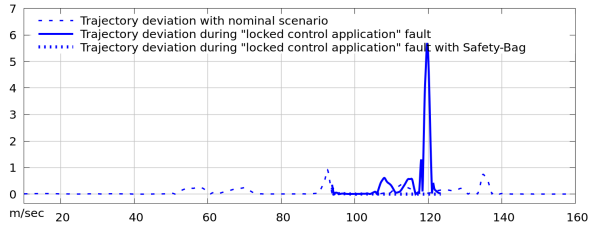

Figure 14. Vehicle trajectory deviation in the case of locked control application fault

is immediately detected as an identification number on the sent commands is no longer incremented. This violates the safety rule stipulating that the control application liveliness must be ensured, and enforces the safety action of rejecting all acceleration commands from the control application while alerting the operator of its failure. The system thus stops at 117s.

2) Incorrect acceleration fault: Without the Safety-Bag, we see in figure 15 that the vehicles speeds up to almost 70 km/h before the left turn. The consequences are not as dire as in

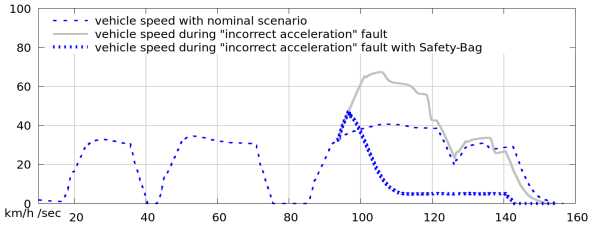

Figure 15. Vehicle speed in the case of incorrect acceleration fault

the previous fault, because the acceleration is not maintained as long (and especially not during the turn), but figure 16 still shows significant deviations from the nominal trajectory, which could be dangerous on a road with opposing traffic.

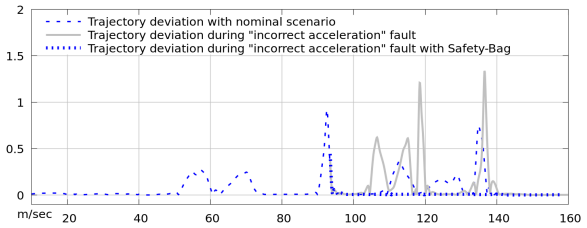

Figure 16. Vehicle trajectory deviation in the case of incorrect acceleration fault

With the Safety-Bag, we see that the vehicle does not go faster than 50km/h. Indeed, the incorrect acceleration violates the safety rule stipulating that the vehicle must not go farther than 47km/h, and the corresponding safety action inhibits all acceleration commands while raising an alarm for the operator, in a similar way than in the previous fault injection. In our current implementation, the Safety-Bag moves to a degraded state from which it can not recover as soon as a safety rule is violated. Therefore, when the car's speed excesses a safety rule's threshold, the acceleration remains inhibited until the end of the experiment and the vehicle stops (figures 13 and 14 for the first fault, 15 and 16 for the second). This behavior is well suited for an experimental vehicle, but for better availability in industrialized cars, the safety-bag could allow acceleration commands again when the car's speed drops below the safety rule's threshold.

VII. CONCLUSION AND PERSPECTIVES

In our study, an *Independent Safety Component* called Safety-Bag is implemented to reduce the risks in an experimental autonomous vehicle. It detects software and hardware faults and recovers by rejecting dangerous inputs, enforcing safe actions, or ultimately giving the control back to the operator. Thus, trained and vigilant pilots remain essential in these experimental vehicles, although the Safety-Bag ensures the detection of some of the most critical errors and allows more response time to the driver.

In this paper, we presented two analysis methods: FMEA and HazOp-UML. These methods are used to determine the safety necessities required by the Safety-Bag to detect and react from problems. They analyze the system from two different points of view and appear complementary for identifying the safety necessities.

However, it should be noted that the results of these analyses (in particular the HazOp-UML analysis) depend greatly on the analyst's skills. In addition, not all safety requirements are necessarily implementable by the Safety-Bag, which must remain simple enough to be easily validated. Moreover, the Safety-Bag should not involve decisions based on risky components (such as data fusion or artificial intelligence mechanisms). Nonetheless, the first tests performed on the virtual runway on the VILAD testbed confirm the interest of using the Safety-Bag in autonomous vehicles, and the interest of FMEA and HazOP-UML to design its safety necessities.

For perspectives, we intend to experimentally validate more safety necessities, and eventually study the absence of conflicts in a way similar to the SMOF approach [15].

ACKNOWLEDGMENTS

This work was carried out and funded under the EQUIPEX ROBOTEX. It was supported by the French Government, through the program *Investissements d'avenir* managed by the *Agence Nationale de la Recherche*. (Reference: ANR-10-EQPX).

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[3] J. K. Naufal, J. B. Camargo, L. F. Vismari, J. R. de Almeida, C. Molina, R. I. R. González, R. Inam, and E. Fersman, "A^{2}cps: A vehicle-centric safety conceptual framework for autonomous transport systems," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[4] N. H. T. S. A, "Federal automated vehicles policy: Accelerating the next revolution in roadway safety," *Washington, DC: US Department of Transportation*, 2016.

[5] B. Lussier, R. Chatila, F. Ingrand, M.-O. Killijian, and D. Powell, "On fault tolerance and robustness in autonomous systems," *Proceedings of the 3rd IARP-IEEE/RAS-EURON joint workshop on technical challenges for dependable robots in human environments*, pp. 351–338, 2004.

[6] T. Xu and H. Yuan, "Autonomous vehicle active safety system based on path planning and predictive control," *Chinese Control Conference (CCC)*, pp. 8889–8895, 2016.

[7] M. Althoff, O. Stursberg, and M. Buss, "Safety assessment of driving behavior in multi-lane traffic for autonomous vehicles," *Intelligent Vehicles Symposium*, pp. 893–900, 2009.

[8] D. Meltz and H. Guterman, "Verification of safety for autonomous unmanned ground vehicles," *Electrical & Electronics Engineers in Israel (IEEEI), IEEE 28th Convention of*, pp. 1–5, 2014.

[9] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 2016-01-0128, pp. 15–24, 2016.

[10] P. Klein, "The safety-bag expert system in the electronic railway interlocking system elektra," *Expert Systems with Applications*, vol. 3, no. 4, pp. 499–506, 1991.

[11] J. Blanquart, S. Fleury, M. Hernerk, C. Honvault, F. Ingrand, J. Poncet, D. Powell, N. Strady-Lécubin, and P. Thévenod, "Software safety supervision on-board autonomous spacecraft," in *Proceedings of the 2nd European Congress Embedded Real Time Software (ERTS'04)*, 2004.

[12] G. Guesnier, J.-F. Hamelin, and J.-M. Peyrouton, "Centrales nucléaires n4: l'informatique au service d'une conduite plus sûre," *Epure*, no. 56, pp. 95–96, 1997.

[13] C. Pace, D. Seward, and I. Sommerville, "A safety integrated architecture for an autonomous excavator," in *IEEE, Proc. 17th Int. Symp. on Automation and Robotics in Construction,(ISARC), Taiwan*, 2000.

[14] F. Py and F. Ingrand, "Dependable execution control for autonomous robots," in *Intelligent Robots and Systems,(IROS). Proceedings. IEEE/RSJ International Conference on*, vol. 2. IEEE, 2004.

[15] M. Machin, J. Guiochet, H. Waeselynck, J.-P. Blanquart, M. Roy, and L. Masson, "Smof: A safety monitoring framework for autonomous systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2016.

[16] MIL-STD-1629A, "Military standard: Procedures for performing a failure mode, effects and criticity analysis," Departement of Defence Washington, Tech. Rep., 1980.

[17] J. Guiochet, "Hazard analysis of human–robot interactions with hazop–uml," *Safety science*, vol. 84, pp. 225–237, 2016.

[18] A. Mekki-Mokhtar, J.-P. Blanquart, J. Guiochet, D. Powell, and M. Roy, "Elicitation of executable safety rules for critical autonomous systems," in *Embedded Real Time Software and Systems (ERTS)*, 2012.