Ein Ansatz zur Wiederherstellung ausgefallener Hardwarekomponenten in Fail-Operational Architekturen (A Hardware Redundancy-Recovery Approach for Fail-Operational Architectures)

Eine Erweiterung von FDIRO (An FDIRO Add-On)

Dipl.-Ing. **T. Kain**, B.Sc., Volkswagen AG, Wolfsburg;
T. Horeis, M.Sc., IQZ GmbH, Hamburg;
J. Heinrich, M.Sc., IQZ GmbH, Wuppertal;
a.o. Univ.-Prof. Dr.techn. **H. Tompits**, TU Wien, Vienna;
M. Aguirre Mehlhorn, BSc., Volkswagen AG, Wolfsburg;

Kurzfassung

Autonome Fahrzeuge, d.h. Fahrzeuge der SAE-Stufen 4 und 5, bestehen zu großen Teilen aus komplex verteilten Systemen, die im Falle eines Fehlers nicht auf eine menschliche Rückfallebene zurückgreifen können. Folglich werden Prozesse benötigt, die Fehler behandeln und somit gefährliche Situationen vermeiden können. Ein fail-operational Ansatz zur Erfüllung dieser Anforderungen ist FDIRO, welcher die eigenständige schrittweise Behandlung von Fehlern ermöglicht. Nachdem von FDIRO ein Fehler erkannt wurde erfolgt dessen Isolierung indem die fehlerhafte Komponente deaktiviert und eine redundante Komponente aktiviert wird. Anschließend führt FDIRO die Wiederherstellung des Redundanzgrades durch. Als letzter Schritt erfolgt eine Optimierung der Systemstruktur. Im aktuellen Entwicklungsstadium beinhaltet FDIRO einen Ansatz zur Wiederherstellung des Redundanzgrades, der die Redundanz von Softwareanwendungen durch das Starten neuer Anwendungsinstanzen wiederherstellen kann. Allerdings ist FDIRO nicht in der Lage, ausgefallene Hardwarekomponenten, wie Recheneinheiten oder Sensoren, wiederherzustellen. Folglich kann nur eine begrenzte Anzahl von Hardwareausfällen toleriert werden, bevor die Mission durch einen Notstopp beendet werden muss.

In diesem Beitrag wird als Add-on von FDIRO der Hardware-Redundanz-Wiederherstellungsansatz HRR eingeführt. Im Falle eines Hardwarefehlers weist FDIRO nach der Durchführung des Isolationsschritts HRR an, die fehlerhafte Komponente wiederherzustellen. HRR analysiert den Fehler und bestimmt einen Reparaturplan, der aus einer geordneten Liste von Reparaturansätzen besteht. Im Falle einer erfolgreichen

Reaktivierung sorgt FDIRO abschließend für eine kontrollierte Integration der reparierten Hardwareeinheit.

Abstract

Autonomous vehicles, i.e., vehicles classified as SAE Level 4 and 5, are complex distributed systems which exclude any human takeover action in case of a failure. Consequently, those vehicles have to be equipped with mechanisms that can handle failures to prevent hazardous situations. To address this, in previous work, FDIRO was developed, a fail-operational approach that handles hard- and software faults in a stepwise fashion. After a fault was detected and isolated, by deactivating the faulty component and activating a redundant component, FDIRO performs a redundancy recovery followed by an application-placement optimization. At the current stage of development, FDIRO implements a redundancy-recovery approach that can recover the redundancy of software applications by starting new application instances. However, FDIRO is not capable of recovering failed hardware components like computing nodes or sensors. Consequently, only a limited number of hardware failures can be tolerated before the mission has to be terminated by an emergency stop.

In this paper, we introduce the hardware redundancy-recovery approach HRR, which is an add-on to FDIRO. In case of a hardware fault, after performing the isolation step, FDIRO instructs HRR to recover the faulty component. HRR analyzes the fault and determines a *repair plan*, an ordered list of repair approaches. In case of a successful reactivation, FDIRO ensures a controlled integration of the repaired hardware.

1. Introduction

The development of automated and autonomous driving functions at SAE levels 4 and 5 [7] poses many challenges for the automotive industry. One major challenge is replacing the driver as a controller of the operation of the vehicle. In an autonomous vehicle, the system has to handle failures of functions and components autonomously [4], i.e., the system has to be fail-operational.

The system must react to failures by isolating the faulty component and switching to a fallback configuration that can continue the driving task. After a successful switchover, the goal is to restore the original system state or an alternative configuration, which meets all system design requirements, so that the operation of the vehicle can be continued safely.

To address these requirements, the fail-operational process FDIRO [3] was developed. FDIRO performs a series of steps after the occurrence of a failure in order to restore the system. At the current stage of development, FDIRO is able to recover failed software components but lacks a hardware-repair behavior. As a consequence, hardware faults significantly degrade the reliability of the system. In many cases, hardware failures are not irreparable, e.g., a sensor failure can be, for instance, caused by external factors like dirt or internal events like load peaks. Thus it is conceivable to reactivate the failed components by performing certain repair actions.

In this paper, we investigate the effects of reactivating failed hardware components during runtime. As a motivation for our approach, we first show that performing repair actions during runtime improves the reliability of the system. Afterwards, we introduce the hardware redundancy-recovery approach HRR, which attempts to repair hardware components after they failed during runtime autonomously. Furthermore, we illustrate how the HRR approach can be included in the existing FDIRO procedure.

The idea of HRR is to first analyze the error that caused the failure. Based on this analysis, an ordered list of repair actions is determined, whereby the repair actions are defined and ranked at design time by the system engineer. After the execution of a repair action, the HRR procedure checks whether the fault was fixed. If this is the case, the repaired hardware component can again be used by the system. Otherwise, a subsequent repair action is executed.

To the best of our knowledge, no general hardware repair approach like the one presented in this paper exists. However, some strategies focusing on hardware maintenance have been defined in the past, discussed in what follows.

To begin with, Raghavaiah and Hariprasad [6] specified the following four maintenance strategies used for mechanical equipment: (i) breakdown maintenance, (ii) preventive (scheduled) maintenance, (iii) predictive maintenance, and (iv) reliability-centered maintenance. While breakdown maintenance is a reactive maintenance process that fixes components after their failure, preventive maintenance repairs the component after predefined time intervals to prevent failure and extend the lifetime of the components. Predictive maintenance tries to predict the failure occurrence time by monitoring the components to fix the component before the failure occurs. Reliability-centered maintenance is a process, which analyzes all possible failure modes to customize an individual maintenance plan for each component.

Qin and Li [5] defined a preventive maintenance process with a Markov decision process to determine different reliability parameters. The model can consider systems with several

degraded states and preventive maintenance actions between the different states. Even though maintenance always assumes downtime to execute the maintenance actions, this model can consider a repair behavior of hardware components by using degraded states and their transitions. However, the preventive maintenance actions are modeled using a constant repair rate, and the actions are not further specified or modeled.

Selcuk [8] defines some main future research areas for predictive maintenance, including e-maintenance, remote maintenance, and management systems. While e-maintenance extends the predictive maintenance techniques with monitoring and prognostic functions over the internet, remote maintenance and management systems are aiming to restore the system from a distance. The focus of predictive maintenance lies within intelligent predictions of possible failures rather than intelligent maintenance. However, the authors mentioned promising fields that are focusing on the automation of the maintenance process.

Note that none of these strategies, models, and ideas consider a hardware-repair behavior that is executed during the operation of the vehicles.

The paper is organized as follows: In Section 2, we describe the FDIRO process in its current form. Section 3 shows that implementing a hardware-repair behavior can increase the reliability of autonomous vehicles. In Section 4, we introduce the HRR approach. Furthermore, we illustrate the integration of HRR into FDIRO and provide an example use-case. Section 5 concludes the paper with a brief summary and an outline for future work.

2. FDIRO: A Fail-Operational Approach for Autonomous Vehicles

FDIRO [3], standing for “Fault Detection, Isolation, Recovery, and Optimization”, is a fail-operational approach that is specifically designed for the use-case of autonomous driving. The idea of FDIRO is to handle occurring faults in a stepwise manner, as illustrated in Figure 1.

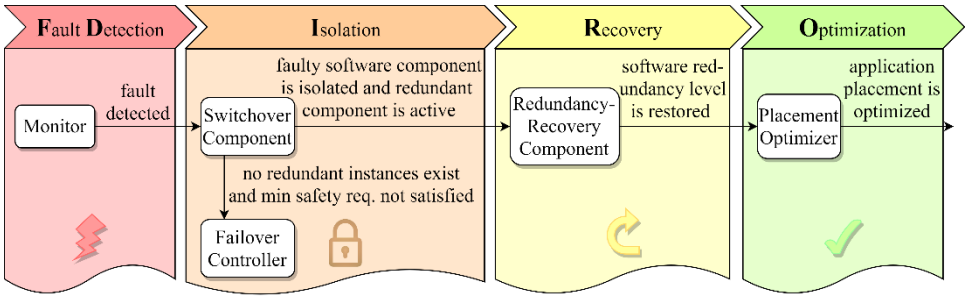

Figure 1: Simplified representation of the FDIRO activities.

The first step defines the detection of the fault. Therefore, so-called *monitors* observe the hardware and software applications of the self-driving system. In case a monitor detects a fault, it reports this fault to the *switchover component*. This component isolates the fault by deactivating the faulty software applications. Note that, in case a monitor reports a hardware fault, all software applications executed by this hardware are considered faulty. Next, the switchover component activates redundant software applications that take over the tasks of the failed applications. If such redundant applications do not exist, the consequences of the fault have to be evaluated. In case this evaluation predicts severe consequences, the *failover controller* takes over the control of the system and brings the vehicle to a stop.
After the isolation of the faulty component and the activation of a redundant component, the *redundancy-recovery component* aims to restore the software-redundancy level. Therefore, the redundancy-recovery component tries to find hardware components that offer enough free resources to run redundant applications.
In the last step, the *placement optimizer* tries to find an optimized application placement that suits the current driving situation.

3. Effects of a Hardware-Repair Behavior

Due to the technological capabilities of autonomous vehicles, it is conceivable that those vehicles take measures to repair failed hardware components during operation. For instance, if the visibility of a sensor is compromised by dirt or ice, the vehicle may activate the cleaning or heating unit, respectively, to reactivate the sensor. In case that the system architecture is designed such that another sensor can compensate for the sensor fault during the repair process, it is assumed that the reliability of the system can be improved.
In this section, we investigate the effects of a hardware-repair behavior using AT-CARS [2], a tool to analyze the reliability of a system. AT-CARS allows modeling the software and hardware components of a given system architecture. Furthermore, for each component, a failure rate can be specified. Additionally, AT-CARS also allows defining repair rates for hardware components, i.e., the probability that a hardware component can be repaired after it failed.
Based on the defined failure rates, AT-CARS injects failures into the software and hardware components of the system architecture. After a failure has been injected, the steps specified by FDIRO are executed. AT-CARS evaluates for each occurring failure its effects on system reliability. As AT-CARS is based on a Monte Carlo simulation, many iterations are required to get reliable output values.

The system architecture, illustrated in Figure 2, that we used to investigate the effects of a hardware-repair behavior is based on the sensor setup as presented by nuTonomy [1]. Furthermore, we added two computing nodes, whereby each computing node is executing three applications, which in turn are arranged redundantly. Each application is once executed in the so-called *active state* and redundantly in the *active-hot state*. Note that since our goal is to investigate the effects of a hardware-repair behavior, a realistic software stack is not required.

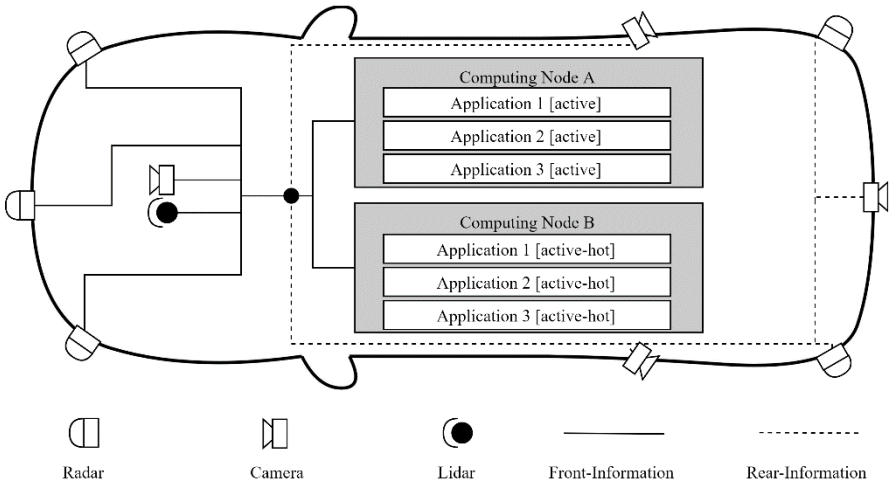

Figure 2: System architecture used for evaluating the effects of a hardware-repair behavior.

For the sensors, we assume a failure rate of $5 \cdot 10^{-5}$ failures/hour and for the computing nodes a failure rate of 10^{-5}. Furthermore, we divided the sensors into two groups: front sensors and rear sensors. We define that at least four sensors from each of these two groups must be fully functional for the safe operation of the vehicle. Furthermore, we define that at least one instance of each application has to be executed by the system. However, since in this example we focus on the hardware-repair behavior, we set the failure rate of the software to 0 failures/hour.

To show the effects of a hardware-repair behavior, we evaluated the system architecture using different repair probabilities, whereby we assumed in each simulation run the same repair probability for all sensors and computing nodes. The probabilities vary between 0%, 5%, 10%, 30%, 50%, and 100%. As can be seen in Figure 3, the reliability of the system increases as the repair probability increases. This example shows that even small repair probabilities can increase the system reliability.

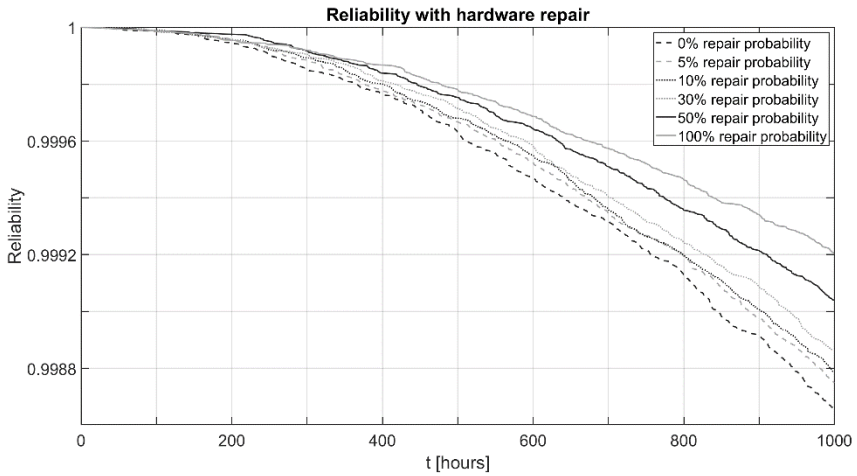

Figure 3: Evaluation of the effects of different repair probabilities on the system reliability.

4. HRR: A Hardware-Redundancy Recovery Approach

The previous section illustrated that integrating a hardware-repair behavior into FDIRO can improve the system reliability. As the traditional FDIRO procedure does not include a hardware-repair behavior, we introduce in this section the hardware-redundancy recovery approach HRR, which is an extension to FDIRO.

In what follows, we describe the HRR procedure as well as its integration into FDIRO. Furthermore, we provide a use-case to illustrate the introduced procedure.

4.1 Approach Overview

The procedure of the hardware-redundancy recovery approach is executed by the so-called *HRR component*. As illustrated in Figure 4, the HRR component first analyses the *error report*, i.e., a report created by the monitor in which the hardware component under investigation has been classified as malfunctioning. Based on that error report, the HRR component determines a so-called *fault category*. The latter contains a pre-sorted list of several faults that might have caused the component under investigation to fail. The faults listed in a fault category can be, for instance, ranked using an FMEA-like ("failure mode and effects analysis") approach [9]. The idea is that for each fault f, its severity, $S(f)$, probability of occurrence, $O(f)$, and probability of detection, $D(f)$, is determined at design time by the system engineer. The values of those parameters are bounded between 1 and 10, whereby 1 is considered insignificant/unlikely, and 10 corresponds to very significant/probable. By multiplying those three parameters, the so-called *risk-priority number*, $RPN(f)$, is calculated:

$$RPN(f) = S(f) \cdot O(f) \cdot D(f).$$

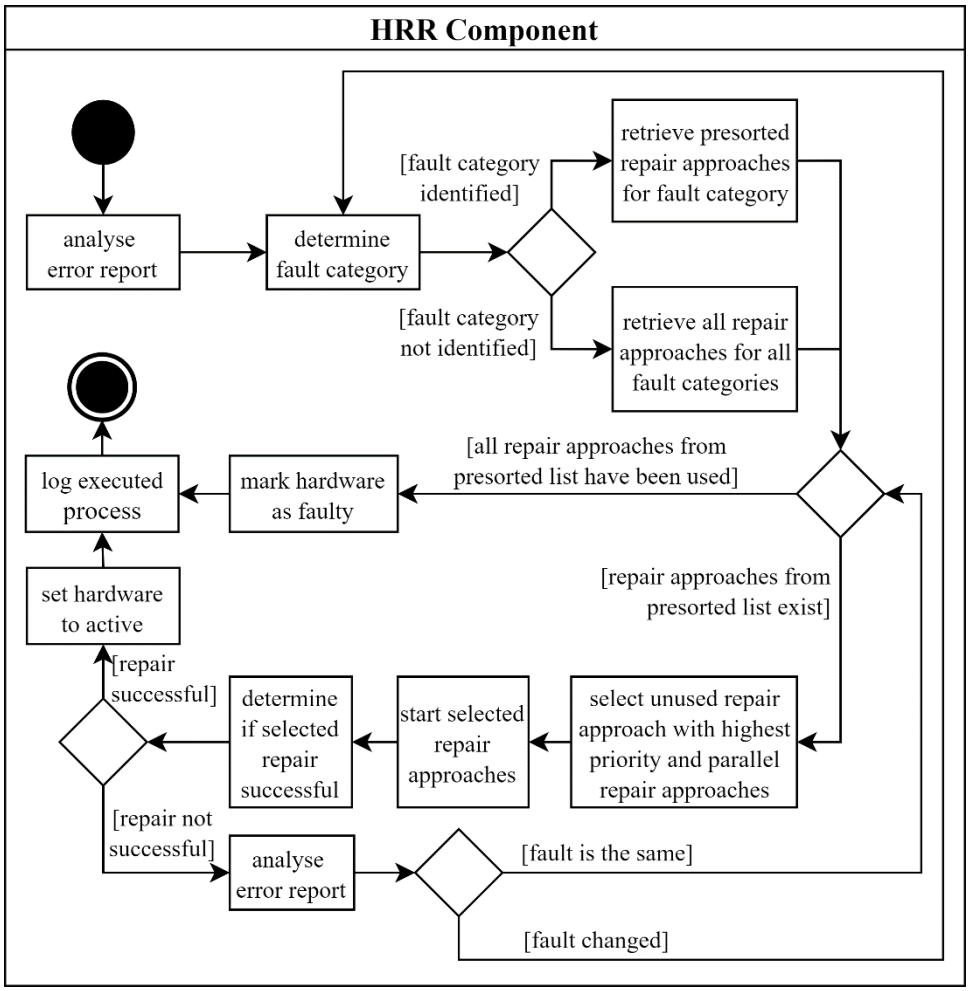

Figure 4: The activity diagram of the HRR component.

The risk priority number of the individual faults is used to rank them in a descending fashion. Each fault entry of an error category holds a predefined list of so-called *repair approaches*, i.e., approaches to fix the fault that caused the hardware component under investigation to fail. The repair approaches are determined by the system engineer at design time and are ordered hierarchically concerning, for example, their effectiveness, estimated computational effort, and execution time. Furthermore, the system engineer defines which repair approaches can be carried out in parallel.

According to the priority of the corresponding fault, the repair approaches of all faults that are part of the identified error category are sequentially added to the so-called *repair plan*.

In case the HRR component cannot identify a fault category based on the provided error report, the repair approaches of all fault categories of the component under investigation are added to the repair plan.

As the error categories, the faults, and the corresponding repair approaches are determined at design time by the system engineer, they are considered as being *static*. However, using, for instance, an OTA (“over the air”) update mechanism, the set of error categories, faults, and repair approaches can be adjusted.

After the list of repair approaches that might fix the fault is identified, the processing of those approaches starts. The HRR component first selects the approach which is ranked the highest. Additionally, all repair approaches, which can be executed in parallel to that approach, are selected as well. Next, the selected repair approaches are executed. Once all selected repair approaches terminated, the HRR component determines whether the repair was successful. A repair is considered successful if the monitor of the component under investigation does not detect any error. In that case, the hardware component under investigation is set to active and can therefore be again used by the system.

Before the HRR component terminates, the executed procedure, including, for instance, the determined fault category, the executed repair approaches as well as their results, are logged. The logs of several vehicles can then be used to optimize, for example, the prioritization of repair approaches.

In case that the executed repair approaches did not fix the fault, the HRR component determines whether the error report of the monitor still indicates the same fault. If this is the case, the next repair approach in the repair plan that has not been conducted yet is selected. If no more repair approaches that have not been executed yet exist, the hardware component under investigation is marked as *faulty* and can therefore not be used by the system. After logging the executed procedure, the HRR component terminates.

However, in case the analysis of the error report yields a fault that is different from the one determined at the beginning of the HRR procedure, the procedure has to start over.

4.2 Integration of the HRR component into FDIRO

At the current stage of development FDIRO is capable, due to its redundancy-recovery component, to recover software applications, but it lacks a procedure to recover hardware. To extend FDIRO towards a fail-operational approach that can recover failed software as well as hardware, we integrate the HRR component into the FDIRO procedure.

As illustrated in Figure 5, the procedure of the HRR component is initiated by the redundancy-recovery component of FDIRO. Once the switchover component successfully isolated the faulty component, the redundancy-recovery component checks whether the failure is due to a software or hardware component. In case a software instance failed, the redundancy-recovery component tries to find a computing node that offers enough resources

to start a redundant instance of the failed software. Note that this procedure is part of the standard FDIRO procedure.

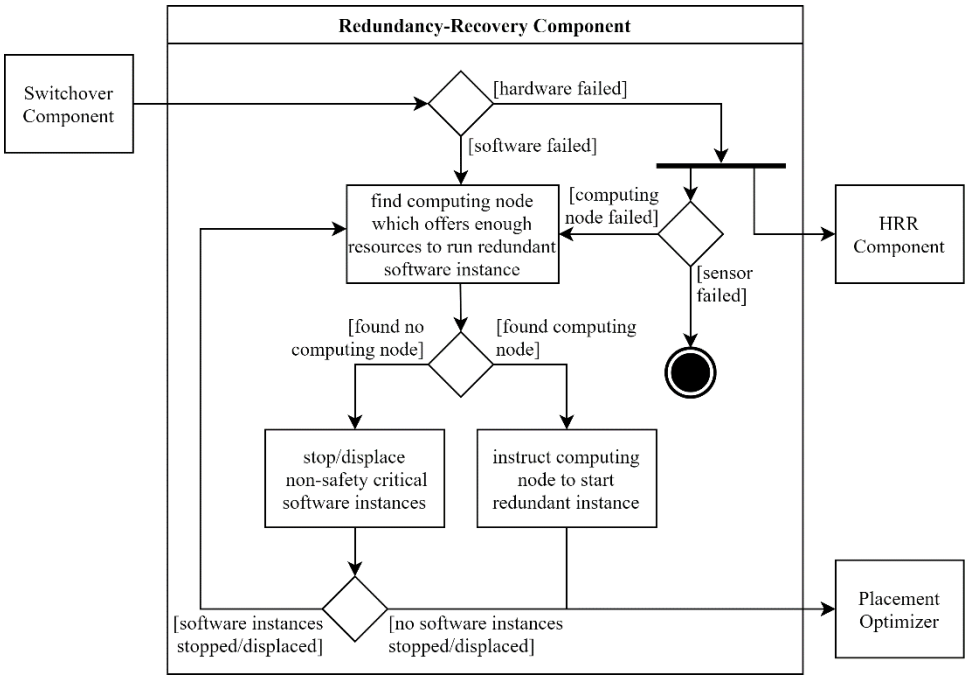

Figure 5: The integration of the HRR component into FDIRO.

However, in case the failed component is a hardware part, the redundancy-recovery component triggers the execution of the procedure defined by the HRR component. Furthermore, in parallel, the redundancy recovery component checks whether the failed hardware is a computing node or a sensor. If it is a sensor, no further actions are required. In case, however, a computing node failed, all the software applications executed by this computing node failed as well. Consequently, the redundancy recovery procedure has to be executed in order to restore the software redundancy level.

4.3 Use-Case

Let us assume a scenario where a monitor detects that the front-right radar sensor failed. Consequently, FDIRO isolates that component so that the data provided by this radar sensor is no longer used as input for any software application of the self-driving system. Once the front-right radar is isolated, FDIRO instructs the HRR component to start its repair procedure. A subset of the error categories for radar sensors is illustrated in Figure 6. In this use-case, we assume that according to the error report provided by the monitor, the error category

“value constant” is the most fitting, i.e., the data points of the radar sensor are constant, e.g., in terms of the measured distance or relative velocity.

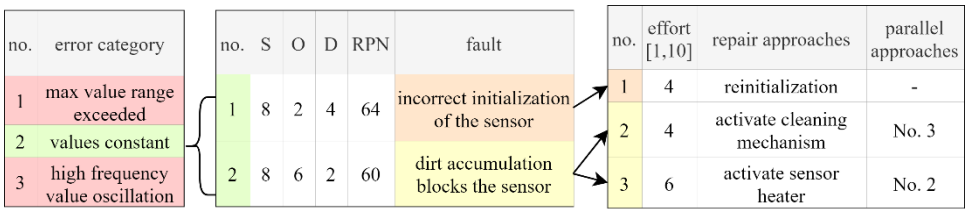

Figure 6: The error categories, faults, and repair approaches of a radar sensor.

As illustrated in Figure 6, two faults, i.e., “incorrect initialization of the sensor” and “dirt accumulation blocks the sensor”, are defined for the considered error class, which are ordered according to their risk-priority number. Furthermore, for both faults, repair approaches are specified.

In order to attempt to repair the faulty radar sensor, the HRR component first initiates a reinitialization of the sensor, as this repair approach is defined as a countermeasure for the fault having the highest risk-priority number. After the reinitialization of the front-right radar sensor terminates, a monitor determines whether the sensor is still faulty. In this use-case, we assume that the reinitialization of the sensor does not fix the fault, and the fault does not change during the repair procedure. Consequently, the HRR component executes the remaining repair approaches.

The next approach conducted by the HRR component in order to fix the fault is to activate the cleaning mechanism of the sensor. While executing this repair approach, the sensor heating is activated as well since those two approaches can be performed in parallel.

After the termination of both repair approaches, the monitor again checks whether the fault was fixed. Assuming this is the case, the HRR component logs the fault as well as the executed repair approaches and activates the sensor again.

5. Conclusion

In this paper, we introduced the hardware-redundancy recovery approach HRR, which is specifically designed for the use in autonomous vehicles. Furthermore, we illustrated that implementing a hardware-repair behavior can improve the system reliability of autonomous vehicles. Consequently, HRR is realized as an add-on to FDIRO, a fail-operational approach for autonomous vehicles, enabling the latter to now recover failures of both software and hardware components.

Concerning future work, we plan to implement the HRR approach in a case study for which we intend to demonstrate that the reliability of autonomous vehicles can be increased if their integrated fail-operational system implements a hardware-repair behavior.

References

[1] Caesar, H. , Bankiti, V., Lang, A., Vora, S. Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O.: nuscenes: A Multimodal Dataset for Autonomous Driving. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020, pages 11618-11628.

[2] Horeis, T. F., Kain, T., Müller, J.-S., Plinke, F., Heinrich, J., Wesche, M., and Decke, H.: A Reliability Engineering Based Approach to Model Complex and Dynamic Autonomous Systems. In Proceedings of the 3rd International Conference on Connected and Autonomous Driving (MetroCAD). 2020.

[3] Kain, T.,Tompits, H., Müller, J.-S., Mundhenk, P., Wesche, M., and Decke, H.: FDIRO: A General Approach for a Fail-Operational System Design. In Proceedings of the 30th European Safety and Reliability Conference (ESREL). 2020.

[4] Koopman, P., and Wagner, M.: Autonomous Vehicle Safety: An Interdisciplinary Challenge. IEEE Intelligent Transportation Systems Magazine. 2017, 9(1):90–96.

[5] Qin, J., and Li, Z.: Reliability Modeling for Multistate System with Preventive Maintenance under Customer Demand. Complexity in Economics and Business. 2020.

[6] Raghavaiah, N., and Hariprasad, I.: Maintenance and Reliability Strategy of Mechanical Equipment in Industry. International Research Journal of Engineering and Technology (IRJET). 2019, 6(6):3430–3432.

[7] SAE International. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. In SAE Standard J3016, pages 1–16, 2018.

[8] Selcuk, S.: Predictive Maintenance, its Implementation and Latest Trends. In Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. 2017, 231(9):1670-1679.

[9] Teng, S.-H. G., and Ho, S.-Y. M.: Failure Mode and Effects Analysis: An Integrated Approach for Product Design and Process Control. International Journal of Quality & Reliability Management. 1996, 13(5):8–26.