



Martin Stoffel

On-Demand Triple Modular Redundancy for Autonomous Vehicles



Scientific
Publishing

Martin Stoffel

On-Demand Triple Modular Redundancy for Autonomous Vehicles

On-Demand Triple Modular Redundancy for Autonomous Vehicles

by
Martin Stoffel

Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

On-Demand Triple Modular Redundancy for Autonomous Vehicles

Zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.) von der KIT-Fakultät für Elektrotechnik und Informationstechnik des Karlsruher Instituts für Technologie (KIT) genehmigte Dissertation

von Dipl. Wirtsch.-Ing. Martin Stoffel, M.Eng.

Tag der mündlichen Prüfung: 21. März 2025

Hauptreferent: Prof. Dr. Eric Sax

Korreferent: Prof. Dr. Stefan Wagner

Impressum



Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

KIT Scientific Publishing is a registered trademark
of Karlsruhe Institute of Technology.
Reprint using the book cover is not allowed.

www.bibliothek.kit.edu/ksp.php | E-Mail: info@ksp.kit.edu | Shop: www.ksp.kit.edu



This document – excluding the cover, pictures and graphs – is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0): <https://creativecommons.org/licenses/by/4.0/deed.en>



The cover page is licensed under a Creative Commons Attribution-No Derivatives 4.0 International License (CC BY-ND 4.0): <https://creativecommons.org/licenses/by-nd/4.0/deed.en>

Print on Demand 2025 – Gedruckt auf FSC-zertifiziertem Papier

ISBN 978-3-7315-1437-4

DOI 10.5445/KSP/1000181098

Kurzfassung

Ein aktueller Forschungsschwerpunkt für Automobile ist der Wandel von manuell gesteuerten Fahrzeugen zu autonomen Fahrzeugen, die keinen menschlichen Fahrer mehr benötigen. Stattdessen übernehmen Computer die Fahraufgabe, auch in öffentlichen Bereichen mit anderen Verkehrsteilnehmern. Dies bringt hohe Safety-Anforderungen mit sich. Zusätzlich verwenden autonome Fahrzeuge moderne machine-learning basierte Algorithmen, die spezialisierte System on a Chips (SoC) mit entsprechenden Hardwarebeschleunigern in Steuergeräten benötigen.

Derartige SoCs existieren bereits für automobiler Anwendungen, jedoch verfügen sie nicht über das geforderte Safety Level hinsichtlich zufällig auftretender Hardwarefehler. Einen Ausweg bietet der Standard ISO 26262 durch sein Dekompositionsprinzip, welches erlaubt, zwei solche niedrig klassifizierte Elemente miteinander zu verbinden, sodass sie zusammen das höchste Safety Level erreichen.

Die vorliegende Dissertation stellt ein neues Konzept vor, welches das Dekompositionsprinzip verwendet und zwei unabhängige Softwareapplikationen auf zwei unabhängigen Steuergeräten betreibt. Nach dem Vorbild der Luftfahrt wird ein Voting Mechanismus vorgestellt, der die Rechenergebnisse beider Applikationen kontinuierlich vergleicht. Im Falle eines Fehlers, wird die Applikation auf einem dritten Steuergerät on-demand aktiviert und ausgeführt. Dies erlaubt die Identifikation des fehlerhaften Steuergeräts sowie die Rekonfiguration des Systems um uneingeschränkt funktionsfähig zu bleiben.

Im Gegensatz zu aktuellen Konzepten minimiert dieser Vorschlag die Anzahl der redundanten Komponenten, da es keinen voll ausgebildeten dritten Kanal zur Fehlererkennung benötigt. Das System wurde auf einem Prüfstand sowie in einem schweren Nutzfahrzeug getestet und zeigt vielversprechende Ergebnisse hinsichtlich der Fehlererkennungs- und Rekonfigurationszeit.

Abstract

One recent focus topic in automotive development is the move from manually driven vehicles to autonomous vehicles (AV) which no longer require human intervention. Instead, computers take over the driving task by maneuvering the AV in public areas what brings exceptional high safety requirements to the system. Additionally, AVs require the usage of machine learning algorithms which need specialized Systems on a Chips (SoCs) as processing units inside automotive Electronic Control Units (ECU).

While those SoCs are already existing, they are limited in their safety level regarding randomly occurring hardware faults. That makes them not suitable for safety critical applications in standalone operation. However, the functional safety standard ISO 26262 allows to decompose a desired safety level by combining two lower rated SoCs to jointly achieve a higher safety level.

This dissertation proposes a novel concept which utilizes the decomposition principle by operating AV software applications redundantly on two connected ECUs. Taking avionic architectures as example, the required fault detection is realized by comparing the output of both applications using distributed voters. In case of a detected fault the system on-demand executes the faulty software application on a third generic back-up ECU again. This allows the identification of the faulty ECU and the reconfiguration of the system to stay operational.

Contrary to existing solutions the proposed approach minimizes the number of required redundant components since it does not need a fully populated third channel as in the traditional triple modular redundancy pattern. The system was tested on a test bench and in a class 8 long haul truck. It showed promising worst case fault handling durations of 3.2s, thereby contributing to the research for safe architectures for AVs.

Acknowledgement

Almost four years have passed since I began my PhD. Back then my most concerning question was if I will be able to successfully finish it since I was working full-time as Software Engineer and I was a dad of two children. Now, while writing the final words of this thesis, I can proudly say: Yes, it was possible!

It was possible out of three reasons: First, the inner positive attitude and the love to the things I did which unleashed exceptional energy. Second, the discipline to continue during times, when things didn't go in a way, I intend them to be. And third and most importantly, due to the people who surrounded and supported me, while working on my PhD.

Therefore, first and foremost I would like to express my deep appreciation towards Prof. Dr. Eric Sax, who gave me the opportunity of pursuing my PhD and continuously supported me with constructive conversations and valuable feedback. I also would like to thank Prof. Dr. Stefan Wagner for taking over the role as second reviewer.

Secondly, I am very grateful to Dr. Axel Gern and Michael Smuda, who gave me the chance to work on my PhD in parallel to my profession. Also, I am grateful to Dr. Janine Guenther, who gave me stability and support during unstable times.

Last and most importantly, I would like to say thank you to my family, especially to my wife Karina and my kids Sofia and David. They gave me the room and time I needed to concentrate on my thesis in my spare time, especially over the weekend, late at night and during vacation. I also would like to thank my parents for being inspiration and motivation throughout my life.

Plochingen, 25th March 2025

Table of Content

Kurzfassung	i
Abstract	iii
Acknowledgement	v
List of Abbreviations	xi
List of Variables	xv
1 Motivation	1
1.1 Towards Autonomous Driving	1
1.2 The Importance of Safety	4
1.3 Autonomous Driving System	5
1.4 Research Questions	7
2 Background	9
2.1 Development of Automotive Systems	9
2.2 ECUs and Operating Systems	10
2.2.1 Electronic Control Units	10
2.2.2 Processes, Scheduling and Real-Time	12
2.3 Network Technologies	15
2.3.1 Topologies and Technologies	15
2.3.2 Data Transportation	18
2.4 Automotive E/E Architectures	19
2.4.1 Service-Oriented Architectures	19
2.4.2 Adaptive AUTOSAR and ROS2	21
2.5 Automotive Safety	24
2.5.1 Safety Parameters	25
2.5.2 Dynamic Reconfiguration	27
2.5.3 Functional Safety - ISO 26262	30
2.6 Other related Technologies	33
2.6.1 Convolutional Neural Networks	33

2.6.2	Hashing	35
2.6.3	Virtualization	36
2.6.4	Orchestration.....	37
3	State of Science and Technology	39
3.1	Autonomous Driving	39
3.1.1	Physical System Architecture	39
3.1.2	AD Software Stack	42
3.1.3	Dataflow between the Modules.....	44
3.1.4	Applications inside the Detection Module	46
3.2	Related Safety Research	48
3.2.1	Traditional Safety Architectures	48
3.2.2	Service-oriented Architectures.....	51
3.2.3	CPU Lockstep Architectures.....	59
3.3	Avionic System Design Patterns.....	60
3.4	Critical Appraisal	62
4	The On-Demand TMR System.....	65
4.1	Idea and Contribution.....	65
4.1.1	Function Design and Requirements.....	65
4.1.2	System Design Alternatives	67
4.1.3	The novel On-Demand TMR Concept	71
4.2	Architecture and Components.....	75
4.2.1	Distributed Voters	76
4.2.2	On-Demand Channel	78
4.2.3	Reconfigured Operation	80
4.2.4	Recovery Mechanism	82
4.3	Implementation Alternatives.....	83
4.3.1	SOA Technologies	83
4.3.2	Reconfiguration Technologies	85
4.4	Test Cases and Metric.....	87
4.4.1	Component Test Cases	87
4.4.2	System Test Cases and Final Acceptance Testing	95
4.4.3	Acceptance Criteria	97

5	Prototypical Realization.....	101
5.1	Overview.....	101
5.2	System Testing.....	102
5.2.1	System Set-Up	102
5.2.2	Results of not Reconfigured Channel	105
5.2.3	Influence of Network and ROS2	106
5.2.4	Results of Reconfigured Channel.....	113
5.2.5	Behavior of the on-demand TMR State	115
5.2.6	Influence of Historic Samples	117
5.2.7	Interim Summary.....	120
5.3	L4 Lane Detection Testing.....	122
5.3.1	System Set-Up	122
5.3.2	Results and Comparison to System Testing.....	128
5.3.3	Influence of Scheduling and Application Structure	130
5.3.4	Influence of Time Synchronization and Deployment	134
5.3.5	Interim Summary.....	137
6	Performance of FHTI and the Voting Duration	141
6.1	Performance Related to Comparable References	141
6.1.1	Sleep Detection	141
6.1.2	Reaction of a Human Driver	142
6.1.3	Safety Distance	143
6.2	System Enhancing by Software Mapping.....	146
6.3	Performance Related to the State of the Art Technologies.....	147
7	Conclusion and Outlook.....	151
7.1	Test Case and Requirement Compliance	151
7.1.1	System Test Acceptance	151
7.1.2	L4 Lane Detection Test Acceptance.....	153
7.1.3	Compliance to Requirements	155
7.2	Summary and Limitations	157
7.3	Need for Research	161

8 Appendix	165
8.1 Publications.....	165
8.1.1 Journals and Conferences.....	165
8.1.2 Patents.....	165
8.2 Test Results.....	166
8.2.1 System Test.....	166
8.2.2 Final Acceptance Test	174
8.3 Autonomous Driving Sensors.....	180
8.4 AD-Software and Algorithms	181
8.4.1 Sensing.....	181
8.4.2 Detection	183
8.4.3 Localization	184
8.4.4 Prediction	186
8.4.5 Mission	188
8.4.6 Behavior.....	188
8.4.7 Motion	190
8.4.8 Control	191
8.5 Other Artificial Intelligence Technologies.....	192
8.6 Research Vehicles	194
8.6.1 Approach	194
8.6.2 Vehicles.....	196
8.7 Signal-oriented E/E Architecture	211
8.8 List of Figures.....	216
8.9 List of Tables	220
8.10 References	222

List of Abbreviations

AD	Autonomous Driving
ADAS	Advanced Driver Assistant Systems
ALU	Arithmetic Logical Unit
API	Application Programming Interface
ARA	AUTOSAR Runtime for Adaptive Applications
ASIL	Automotive Safety Integrity Level
AutoKonf	Automatisch rekonfigurierbare Aktoriksteuerungen für ausfallsichere automatisierte Fahrfunktionen
AR	Architecture Requirements
AV	Autonomous Vehicle
AVR	Autonomous Vehicle specific Requirement
CAN	Controller Area Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DDS	Data Distribution Service
DGPS	Differential GPS
DL	Deep Learning
DSP	Digital Signal Processor
DRL	Deep Reinforcement Learning
ECU	Electric Control Unit
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FAT	Final Acceptance Test

FHTI	Fault Handling Time Interval
FPGA	Field Programmable Gate Arrays
GPU	Graphics Processing Unit
GNSS	Global Navigation Satellite System
HDR	High Dynamic Range
IARA	Intelligent Autonomous Robotics Automobile
IL	Imitation Learning
ILC	Iterative Learning Control
IP	Internet Protocol
IPC	Iterative Closest Point
INCT-SEC	Brazilian National Institute of Science and Technology for Critical Embedded Systems
IR	Infrared
IMU	Inertial Measurement Unit
KIT	Karlsruhe Institute for Technology
L4	Level 4 (Automation Grade)
LCM	Lightweight Communications and Marshalling
LIDAR	Light Detection and Ranging
LIN	Local Interconnect Network
MD5	Message Digest 5
ML	Machine Learning
MTTF	Mean Time to Failure
OODA	Observe Orient Decide Act
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
POSIX	Portable Operating System Interface

PROUD	Public Road Urban Driverless-Car Test
RAM	Random Access Memory
RTK	Real-Time Kinematics
RCL	ROS Client Library
RMW	ROS Middleware
RNN	Recurrent Neural Network
ROM	Read Only Memory
SAE	Society of Automotive Engineers
SDC	Self-Driving Car
SOTIF	Safety of the Intended Functionality
QoS	Quality of Service
RADAR	Radio Detection and Ranging
RTOS	Real-Time Operating System
RQ	Research Question
SAE	Society of Automotive Engineers
SHA	Secure Hashing Algorithm
SIFT	Scale Invariant Feature Transform
SoC	System on a Chip
SR	Safety Requirement
SURF	Speeded Up Robust Features
STC	System Test Case
TC	Test Case
TCP	Transmission Control Protocol
TIEV	Tongji Intelligent Electric Vehicle
TOPS	Trillion Operations Per Second
UDP	User Datagram Protocol
UFES	Federal University of Espirito Santo

VM	Virtual Machine
WCET	Worst Case Execution Time

List of Variables

t_{activate}	The duration to activate L4 App'' as recovery for the faulty instance of L4 App.
t_{add}	The duration which a voter needs from the moment it receives a hash from a L4 App until it replies. It can be either $t_{\text{add_ch1}}$, $t_{\text{add_ch2}}$, or $t_{\text{add_ch3}}$.
$t_{\text{add_51}}$	The duration required to reply the voting result to the L4 App in the faulty 51 iteration.
$t_{\text{add_200}}$	The duration required to identify the faulty channel in FAT testing.
$t_{\text{add_201}}$	The duration to reconfigure the faulty application in FAT testing.
$t_{\text{add_ch1}}$	The duration which V_1 needs from the moment it receives a hash from an L4 App until it replies.
$t_{\text{add_ch2}}$	The duration which V_2 needs from the moment it receives a hash from an L4 App until it replies.
$t_{\text{add_ch3}}$	The duration which V_1 needs during reconfigured operation from the moment it receives the hash from the L4 App until it replies.
$t_{\text{call_back}}$	The duration required until the first historic sample was received by L4 App''.
t_{calc}	The duration to calculate the third result in on-demand TRM state by L4 App''.
$t_{\text{calc_r}}$	The duration which is required to calculate the first result within reconfigured operation.
t_{context}	The duration required until L4 App'' is dispatched to the CPU.
t_{cycle}	The duration after which the L4 App receives the next sample as input.
t_{hash}	The duration required to create the third hash.
t_{loop}	The duration to loop through the historic samples for faulty channel detection purposes.
$t_{\text{loop_start}}$	The duration which is required to loop through the historic samples for reconfiguration purposes.
t_{network}	The duration required to transport a hash over Ethernet.

t_{third}	The duration to compare hash_1 and hash_2 with hash_3 to identify the faulty channel.
$t_{\text{third_hash}}$	The duration required to generate the first hash, starting from t_{context} until $t_{\text{loop_start}}$ including t_{vbu} .
$t_{\text{v_bu1}}$	The duration until V_{bu} activates L4 App'' including the duration V_{bu} needs to respond to V_1 and V_2 .
t_{voting}	The duration required for the comparison of the hashes.

1 Motivation

1.1 Towards Autonomous Driving

Autonomous driving came strongly into focus of science and research in the year 2003, when the Defense Advanced Research Projects Agency (DARPA) announced the first Grand Challenge. The goal of this challenge was to develop Autonomous Vehicles (AVs), which are capable of navigating on unknown desert trails and roads. It was the response to a U.S. congressional mandate that a third of US military ground vehicles shall be unmanned by 2015 [1].

The first Grand Challenge with more than 100 registered teams took place in 2004. Even though no team was able to complete the given route, the vehicle named Sandstorm went the furthest and served as role model how to win the next challenge. One year later, five vehicles were able to complete a similar track in the second Grand Challenge. The winning car, Stanley, was developed by a team from Stanford University under the supervision of Sebastian Thrun. As last event within that series, DARPA executed the Urban Challenge in the year 2007. The focus changed from desert trails to civil traffic. The teams had to pass a 97 km drive through an urban environment interacting with other moving vehicles and obeying the traffic rules of California [1]. This was the start of the development of multiple academic and industrial research vehicles (see Figure 1) which brought up a variety of automatized vehicles with different automation capabilities.

The Society of Automotive Engineers (SAE) brought structure regarding the automated driving capabilities into this diverse landscape by releasing the SAE J3016 standard in 2014. Accordingly, “autonomous vehicle” and a “self-driving vehicle” are synonyms for either a high (Level 4) or fully automated

road traffic vehicle (Level 5). The focus of this thesis will reside on level 4 civil road traffic vehicles which are defined in the SAE J3016 as following:

Definition 1.1: *A highly automated vehicle performs the driving task automatically within an Operational Design Domain (ODD) without any expectation that a human driver may intervene [2].*

Definition 1.2: *The ODD describes conditions under which the automated vehicle is designed to perform, such as geographic restrictions, weather, temperatures or lightning conditions [2].*

In contrast to level 4, a fully automated vehicle offers a higher level of autonomy by independently finding its way from a departure to a destination in an unknown environment. It is capable to cope with all occurring situations along the way and is not restricted to a specific ODD [3].

Starting with the DARPA Urban challenge, a variety of academic and industrial research vehicles has been developed resulting in pilot programs which are available for public use (see Figure 1). While the focus until the middle of the last decade was on academic research vehicles (see upper row of Figure 1), a pivot to industrial research vehicles took place during the second half of the last decade (see lower row of Figure 1). Beside the traditional automotive companies such as Daimler, Audi or Delphi, new companies such as Waymo, Cruise and Torc Robotics were founded with the sole purpose to realize autonomous driving. Existing small fleet public tests such as operated by Waymo and Cruise without a human driver show, that the maturity of autonomous driving software is high enough for operation in public areas. While in 2015 only rare activities for autonomous commercial vehicles were observable, at the end of the decade multiple companies either added them to their portfolio or concentrated solely on autonomous driving (AD) commercial vehicles. For instance, companies as Waymo and Cruise teamed up with existing truck manufacturers to develop autonomous commercial vehicles [4].

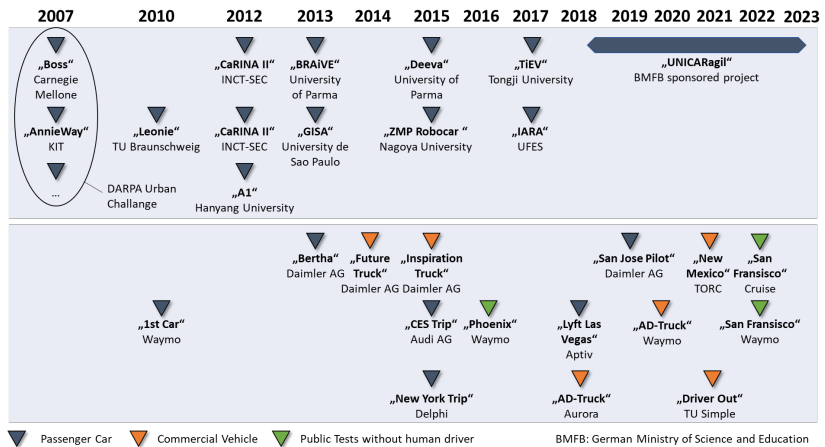


Figure 1: Academic (upper row) and industrial (lower row) research vehicles

Autonomous trucking has two big advantages compared to the passenger car use-case. First, the business case which foresees to take out the human driver is convincing. Studies say, that the market for autonomous trucking is up to 20 times higher than the one for robotic taxis [5]. The second advantage is the reduction of the technical challenge. In fact, the initial announcement of the above mentioned AD-Companies to have multiple vehicles in the field for customer use in 2020 came only partly to reality [6]. It was realized that inner urban areas are still hard to handle. Unsolved software problems lead to crashing processes or hung-up systems which is unacceptable on production level [7]. Another problem is the prediction of the intention of pedestrians, bicyclists, other drivers or animals which is necessary to derive robust driving decisions [8]. Those structural problems are reduced in autonomous trucking since it mainly runs on highways where other traffic participants travel into the same direction, thereby making their behavior more predictable.

The identified AVs (see Figure 1) are only representing a subset of existing industrial vehicles. Embark, Einride, Uber, PlusAI and other companies are also developing self-driving technology for autonomous commercial vehicles on public roads.

1.2 The Importance of Safety

However, the technological advancement of the development and operation of AVs came along with a series of safety critical incidents. The first deadly accident happened in march 2018 where an Uber self-driven car was traveling in Tempe, Arizona. Elaine Herzberg crossed the road with her bike when the self-driving car fatally struck her. Later investigations found out, that the safety culture within Uber was inadequate what lead to an improper design of the system. Additionally, the safety driver was not paying attention to the road through most of the drive [9].

Another incident has taken place with a vehicle from Cruise which struck a pedestrian and dragged her for 6 m in October 2023. She suffered injuries but survived. The report identified a couple of technical errors to be responsible for the misbehavior of the AV and made clear, that a human would have done better in this situation [10]. Due to this event the California Department of Motor Vehicles and the Public Utilities Commission pulled all permits for commercial operation of AV in San Francisco for Cruise. Ever since then the company is in crisis mode since new reports have emerged pointing to the companies safety practice [11].

Another incident happened with an AV from Waymo, which hit a cyclist in San Francisco on February 7th, 2024. While the cyclist had only minor injuries and the analysis is yet not closed, also those smaller incidents heat up the debate whether or not AVs are safe enough to operate in public areas [12].

Safety has always played a curtail and inviolable role within automotive research and development. It was initially coming from the chemical industry, where a series of accidents in the 19th century killed humans due to explosions while working with chemical elements. To avoid those deadly accidents the first safety concepts were formulated and implemented [13].

This paradigm was officially transferred to the area of road traffic safety by the country of Sweden in the year 1997. The Swedish parliament passed a bill

of traffic safety which formulated the expression “Vision Zero”, stating that “...no one will be killed or seriously injured within the road transport system” [14]. Inspired by Sweden’s approach, other European countries took up this process and started on its implementation and further development. In Germany, the “Deutsche Verkehrssicherheitsrat” (DSR)¹ decided on 16th October 2007 that “Vision Zero” will be the main guideline for all future work of the council and explicitly stated, that electronic support systems will play a crucial role to achieve this goal [13].

Given this context, the vision zero also applies for AVs what brings even higher safety requirements to the system design due to the missing human driver.

1.3 Autonomous Driving System

To define a generic structure of an AD system, the existing academic research vehicles (see Figure 1) have been analyzed regarding their system design (see Appendix 8.6.2). In summary, each AV contains the three layers sensing, processing and acting (see Figure 2).

The *sensing layer* comprises different types of sensors needed to perceive the environment. A typical AV can contain up to 9 Light Detection and Ranging Sensors (LiDaR), 12 cameras and 10 Radio Detection and Ranging Sensors (radar) [15] (more details in Chapter 3.1).

The majority of the AD-Software is allocated in the *processing layer* of the system. Examples are perception, planning and decision algorithms (see chapter 3.1.4 and appendix 8.4). The identified research vehicles contained up to 10 industrial computes units (see vehicle Boss in appendix 8.6.2) since only one is not performant enough to provide sufficient compute performance.

¹ DSR is a German council with the purpose to increase the safety of all traffic participants. Among others, the ministries of transport of each federal German state are members of the council.

Since automotive Electrical Control Units (ECUs) have less processing performance than the used industrial computes, even more embedded ECUs are required.

The physical steering of the vehicle takes place within the *acting layer*. Commands coming from the ECUs are transferred via a physical bus to the respective actuators.

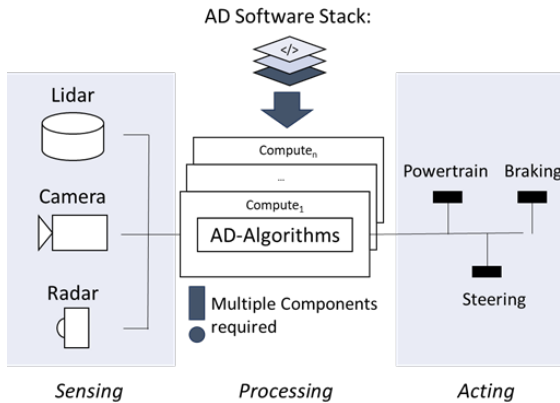


Figure 2: Schematic of an AD-System

To avoid incidents as explained in the previous chapter, the described AD-system must be designed in a manner, that it first is able to detect faults and second is able to react properly. Especially the reaction part is even more important for autonomous trucks compared to the inner urban passenger car use case since trucks travel long distances on highways. A simple stop in lane as reaction to a fault represents a risk for the backwards traffic since other road users must maneuver around while traveling high speed. Out of this reason a fault must not only be detected, but also masked so that the system stays operational, enabling the truck to finish its mission. Especially in north America, where there can be many miles between two exits this brings explicit availability requirements to the AD-system.

1.4 Research Questions

Until now, most of the research for AVs was related to the initial system and software design. Existing vehicles (see Figure 1) and especially the operating public demos of Cruise and Waymo show that the principle closed-loop autonomous driving approach (see Figure 3) is on a mature level.



Figure 3: Autonomous driving closed loop

However, the mentioned incidents show that there are still challenges to be mastered, explicitly around safety to bring AVs into production.

Automotive Electrical and Electronic (E/E) systems such as the introduced AD-System are typically designed in an early phase of the vehicle development process. Over time, architecture patterns have evolved. As of today, most vehicles are designed by using signal-oriented E/E architectures. However, signal-oriented architectures are reaching limits in their capability to transport the required amount of data and the power of their micro controllers for AD-Systems. Furthermore, traditionally inside those signal-oriented E/E architectures safety is typically realized by multiplying the operation of software by using redundancy. This brings explicit challenges to the design of a safe architecture for AVs since the required software is resource heavy. The analysis of the research vehicles showed, that up to ten industrial computers are necessary, to operate the AD Software not considering redundancy (see previous chapter). That makes the usage of traditional patterns hard since vehicles are restricted in weight, dimensions, power consumption and cost.

Thus, the traditional signal-oriented architectures cannot be used for the purpose of AD systems. This makes up the first three research questions to which this dissertation will provide contribution:

RQ1: How does a safe E/E Architecture for autonomous vehicles look like?

RQ2: How is it possible to maximize the time in which the system performs without limitations, even in the event of a fault?

RQ3: How can restrictions with respect to weight, dimensions, power consumption and costs be considered in a safe E/E Architecture?

Beside the safety aspect, additional challenges must be mastered. To overcome the limitation of the low network and compute performance of traditional signal-oriented architectures, the automotive industry is tending towards service-oriented E/E Architectures (SOAs). SOAs are using automotive Ethernet as underlying network technology what supports a bandwidth up to 10 Gbit/s [16], thereby making it more suitable for future data intensive vehicles [17].

Furthermore, SOAs can be used in combination with higher performant microprocessors, which allow the operation of AD-Software but are not as safe as the mentioned traditional microcontrollers.

Both aspects lead to the next research question:

RQ4: How is it possible to realize such a safe E/E Architecture for production vehicles combining existing technologies and AV specific needs?

Lastly, the avionic industry is well known for high safety standards and is regularly mentioned as role model for AVs. This makes up the last research question of this dissertation:

RQ5: Is it possible to transfer elements from avionic architectures to enhance the safety of AVs?

2 Background

2.1 Development of Automotive Systems

Automotive engineering in its original form was a mechanical engineering discipline. The integration of software into a vehicle was an evolutionary process, replacing mechanical customer functions by electronic systems.

Definition 2.1: *A customer function is a desired functionality of the vehicle by the customer in order to meet her or his expectation.*

For autonomous driving, the highest customer function shall be defined as “drive autonomously from a known origin to a known destination without human interaction”. This will later represent the starting point for the prototypical implementation (see chapter 4.1).

In the beginning, automotive electric systems contained single ECUs, fulfilling one specific task. Since then, the number of features grew, leading to an increasing number of ECUs. Today, a top class vehicle contains up to 150 ECUs [18].

Efforts to optimize the increasing number of ECUs according to their physical allocation and their interaction led to first E/E Architectures.

Definition 2.2: *The E/E Architecture of an automobile is the systematical distribution of and the connection between E/E Components within the vehicle under the stipulation of the fulfillment of the desired customer functions [17].*

Definition 2.3: *An E/E Component is a physical device, piece of software or a function, necessary to realize the desired customer functions.*

The dominant E/E Architecture style within the last years was the signal-oriented E/E-Architecture, characterized by lower network bandwidth and static definition of communication (see Appendix 8.7). The future design will be the service-oriented E/E-Architecture [19] (see chapter 2.4.1) which is based on automotive Ethernet, providing higher data rates and middleware technologies, which can start, pause or shut down connections during runtime.

Furthermore, the increasing role of automotive electronic systems required a structured development process. A widely accepted reference process is the V-Shape model [20]. It describes the activities from the setting of customer functions until the final acceptance of the vehicle, containing three phases, namely the design-, implementation and the testing phase.

The design phase is represented by the left side of the V-Shape model. It starts with defining the relevant customer functions for the final vehicle followed by further specifying them according to their technical realization. After the design has been finalized, the implementation of the corresponding software and its realization on hardware takes place in the next phase. The last phase is characterized by testing activities, including the final acceptance. This process ensures, that the system in production works as specified in the design phase.

The prototypical implementation of this thesis will follow a lightweight V-Shape process (see chapter 2.1 and chapter 4.1.1).

2.2 ECUs and Operating Systems

2.2.1 Electronic Control Units

ECUs represent fundamental E/E Components within automotive E/E Architectures. It typically contains input/output (I/O) interfaces to sensors and actuators, a microcontroller unit (MCU), memory, bus interfaces and additional elements such as power supply [17] (see Figure 4).

Definition 2.4: An ECU is an E/E Component which contains physical elements such as a processing unit, memory and interfaces to realize a customer function within an E/E Architecture.

A core element of a traditional ECU is the MCU which can process software by receiving input data and creating output data.

Definition 2.5: A microcontroller unit (MCU) is an integrated circuit which is attached to input/output (I/O) interfaces, bus interfaces and memory, containing one or multiple cores.

Definition 2.6: A core represents the smallest element of a MCU to process software, thereby creating output data relative to input data.

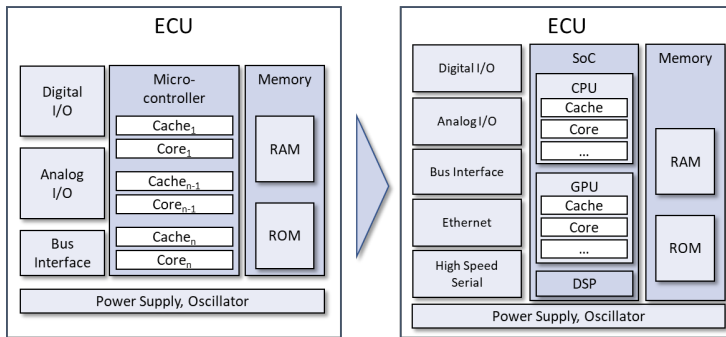


Figure 4: From microcontroller (left) to system on a chip (right)

The cache memory is accessed directly by each core and holds data as well as instructions. It is the fastest available memory, but at the same time also the smallest. The cache has access to the Random Access Memory (RAM) which is bigger but slower. The Read Only Memory (ROM) is the biggest memory but it cannot be written during runtime.

Modern ECUs are optimized towards specific mathematical operations required due to recent developments within automotive software. An example is the customer function “lane keeping assistance” which requires the processing of visual data with artificial intelligence (AI) [21]. For this purpose

specific integrated circuits are developed and compiled on a System on a Chip (SoC).

The graphical processing unit (GPU) is optimized to process the same mathematical operation on multiple cores in parallel. This parallelization is typically required for AI software. The general-purpose computing tasks are executed on the Central Processing Unit (CPU). The Digital Signal Processor (DSP) is used to process continuous digital signals such as audio or video data.

2.2.2 Processes, Scheduling and Real-Time

When an application is started on a SoC the instructions and the relevant data is loaded to the different memory areas.

Definition 2.7: *An application (app) is software, compiled against a desired CPU, MCU or GPU so that it can be loaded and processed.*

Definition 2.8: *An instance describes a loaded application which runs on one or multiple cores on a CPU or a GPU as a process.*

A core contains in minimum registers, a control unit, and an arithmetic-logic unit (ALU). While the registers are again a form of memory inside the core, needed to save interim calculation results between each clock cycle, the ALU is executing operations such as adding or subtraction of binary numbers. The control unit is responsible to push the right instruction set and the right related data to the ALU (see Figure 5).

When an application is loaded it spans out one or multiple processes on a processing unit. Each process can contain threads which hold an encapsulated sub-set of commands and data for further parallelization.

Definition 2.9: *A process is the loaded data and instructions of the application so that it can be processed by a processing unit.*

Definition 2.10: *A thread is a subset of the data of the process which can be processed in parallel and independently by a single core.*

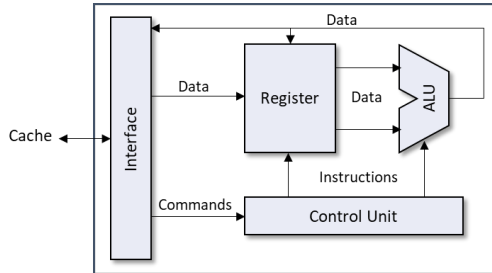


Figure 5: Schematic of a core based on Streichert and Traub [22]

The processing of a process is done iteratively, meaning a command is loaded to the ALU, together with the related data. It then is calculated by the ALU, and the result is saved again. The procedure is repeated in each clock cycle, leading to a varying status in the memory (including the registers) which is called the context.

Definition 2.11: *The context of a process is an immediate representation of the interim calculation results in the form of memory status.*

A processing unit operates in lockstep when a process is deployed on multiple cores in parallel and the register states are compared at each clock cycle. This ensures that the processing unit will not continue with a calculation result which is was not correct due to a random hardware fault.

Definition 2.12: *A processing unit (such as CPU or MCU) operates in lockstep when a process is deployed on multiple cores and the register states are compared before going into the next iteration.*

Multiple parallel running processes can share information with each other. This is typically realized by using shared memory.

Definition 2.13: *Shared memory is a dedicated memory block which two or more processes jointly use, to share information between each other.*

A modern system typically operates multiple processes at the same time, but they cannot be operated in parallel on the cores due to limitations on the processing unit. Thus, the processes compete towards the time they get for execution on the cores. This requires an execution order which is determined by the scheduler. Typical scheduling algorithms are round robin, least-laxity-first, absolute deadline first or earliest deadline first [17].

Definition 2.14: *A deadline represents the absolute duration until a process has to be completed.*

In contrast to round robin, least-laxity-first and earliest deadline first are real-time scheduling algorithms since both consider a deadline of a process which must be guaranteed to operate the system safely.

Definition 2.15: *Real time is the ability of a scheduler to finalize each process before its individual deadline. This deadline must be higher than the Worst Case Execution Time (WCET) of a process.*

However, based on these algorithms the scheduler decides which process to be changed in its state. A process with a higher priority displaces a process with a lower priority what is called preemption. When a process is changed from ready to running the commands and the data must be loaded into the cache of the specific core. This is called a context change.

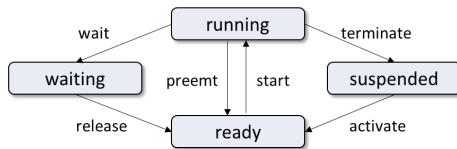


Figure 6: Scheduler process states based on Wolf [17]

A process can also be waiting, what means that it waits for an event before it continues, for instance for a keyboard input. Thus, a waiting process is not scheduled before this event takes place. Once a process is finalized it is terminated and suspended (see Figure 6).

A scheduler is typically part of an operating system (OS). Beside the scheduling, an OS also manages the system resources such as memory or the access to peripherals such as input-output devices. It represents an abstraction from the hardware towards the applications thereby enabling software engineers to develop applications without having to access the hardware directly.

Definition 2.16: *An operating system (OS) maintains the system resources such as access to the CPU, memory or peripherals.*

In the automotive industry the “Offene Systeme für die Elektronik im Kraftfahrzeug/Vehicle Distributed Executive (OSEK/VDX)” operating system is widely used, since it is embedded into the AUTOSAR classic standard.

However, recent ECUs have the performance to also operate generic OS such as Linux. They provide more features than the limited OSEK OS [23] and typically are based on the portable operating system interface (POSIX) standard. This standard provides multiple applications programming interfaces (APIs), allowing the programmer to use the functionalities of the OS.

The core of an OS is its kernel, which connects the hardware such as CPU, memory, or devices to the applications. The kernel is also holding the scheduler. There are real-time capable kernels and non real-time kernels. While the standard Linux kernel is not real-time capable, there is the PREEMT_RT patch available which is widely used to make it real-time capable [24].

2.3 Network Technologies

2.3.1 Topologies and Technologies

The described ECUs can be connected to a network technology so that they can share information, together forming a distributed system.

Definition 2.17: A distributed system is a set of applications which are distributed over multiple physical devices connected with a network technology in order to perform their task jointly by exchanging information thereby forming applications.

The most known network technologies are bus, ring or star topologies (see Figure 7).

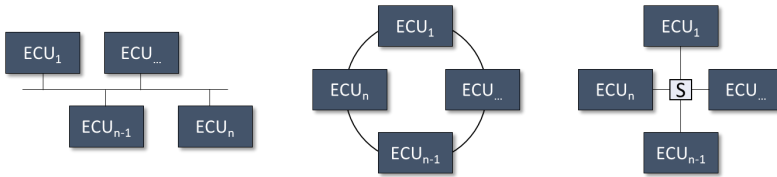


Figure 7: Bus (left), ring (middle) and star (right) topology schematics

A bus is a single cable to which all the ECUs are attached, including end resistors which absorb reflections of the signals at the end of the cable. Typical recent automotive technologies within signal-oriented architectures are using bus systems such as Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay or Multimedia Oriented System Transport (MOST). As depicted in Table 1 they do not exceed a data rate significantly higher than 1 Mbit/s which are not sufficient for AD-Systems.

	Name	Bit-Rate	Use-Case
Traditional	LIN	< 25 Kbit/s	Chassis
	CAN	1 Mbit/s	Chassis, Powertrain
	FlexRay	> 1 Mbit/s	X by Wire
	MOST	>10 Mbits/s	Multimedia
Future	Automotive Ethernet	< 1 Gbit/s	Autonomous Driving, Domain Controller

Table 1: Data rates of automotive bus systems based on Wolf [17]

Contrary a ring topology is characterized by an endless cable which connects the ECUs. Information from one ECU to the other can be sent in both directions of the ring, thereby increasing the probability that the information still arrives in case a corruption happens on one side of the ring.

Another topology is the star, containing one central element, typically a switch (s), which receives the information from a sending ECU and forwards it to the corresponding receiver [25]. Modern switches use the duplex communication patterns which allows to send and receive information simultaneously. The amount of information which can be transported in a second is called the data rate [25]. An advantage of the start topology compared to bus or ring is that the switch can dispatch the traffic only between the sender and receiver(s). While within a bus or a ring the whole traffic of all ECUs is on one wire, this must not be the case in a start topology. If configured properly this safes bandwidth and enables the system to move more data at the same time. Out of this reason, the star topology is selected for this dissertation.

Definition 2.18: *The data rate describes the amount of data transferred in a second between two or more E/E components.*

The data rate of automotive Ethernet is outreaching the data rate of the traditional networking technologies (see Table 1) what is one of the reasons why automotive manufacturers lean towards Ethernet as dominant networking technology.

A switch will use a specific mechanism to forward the information from a sender to a receiver. The most used methodologies are the store-and-forward or the cut-through technique.

In store-and-forward technology the switch will check the incoming data for corruption before forwarding it to the receiving ECU. This method ensures that the data in the network is valid and will not be discarded by the receiver. At the same time, it requires a buffer memory on the switch and makes the transportation slower since the validity check takes time. In case the buffer is full the switch will not accept new incoming data thereby causing a delay. Contrary, the cut-through methodology is not doing this integrity check. It just forwards the data from a sender to the receiver. The integrity check will be done at the receiver and eventually requires the resent of information

through the whole system [25]. Due to the higher integrity of the data, the store-and-forward technology is preferred for this thesis.

2.3.2 Data Transportation

In the 90th different computer manufacturers had proprietary solutions for the realization of the described network topologies. Each solution was not compatible to the other one what enforced the International Organization for Standardization (ISO) to create the Open Systems Interconnection Model (OSI) (see Figure 8).

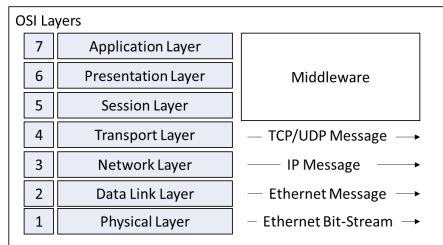


Figure 8: Based on the OSI-Model [128]

The physical layer is responsible to send data from a sender to a receiver by placing electric impulses on a physical wire. The structure of those impulses follow defined standards, the most famous one is Ethernet [25].

The data link layer handles errors by organizing multiple impulses in frames. These frames are acknowledged by the destination if they are received correctly. Otherwise, they are rejected. This layer also manages the available buffer memory at the destination, for example the switch as interim destination [25].

The network layer takes care of the routing of packets across a network. The internet protocol (IP) is currently widely used. By checking the IP address of a package a switch knows the destination and can forward this package accordingly [25].

The transport layer is building up an end-to-end connection between two computers or ECUs. It masks the characteristics of the underlying network thereby allowing logical connections between applications. The most used protocols are the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). While TCP checks if the transmission of data was successful UDP is not taking care of this, following the fire and forget principle [25].

The session layer builds up or shuts down one or multiple sessions between computers or ECUs [25].

The presentation layer formats the data with regard to compression or encryption. An example would be the encoding mechanism of characters, such as American Standard Code for Information Interchange (ASCII) [25].

Finally, the application layer contains multiple protocols which users can use to communicate over the network. A famous example is the hypertext transfer protocol (http) which is used by internet browsers [25].

2.4 Automotive E/E Architectures

2.4.1 Service-Oriented Architectures

The increasing amount of software in vehicles and the corresponding higher amount of data traffic led to the move from signal-oriented E/E architectures to modern SOAs. For the further explanation the widely established 4+1 model for software intensive products will be used, initially introduced by Kruchten in 1995. It was adjusted for the development of automotive architectures according the V-Shape model [16].

Functional View: The functional view represents the highest design level of a development process and the input to the V-Shape model since it is directly derived from the customer expectations. A customer function can be

decomposed into a sub-set of functions (see Figure 9). Thus, the realization of a customer function is the execution of a sequence of its sub-functions.



Figure 9: Decomposition of a customer function

Logical View: A function can be divided into multiple logical E/E components such as logical sensors, logical functions or logical actuators. The logical functions will be realized as software components. A core element of a SOA is the middleware (see Figure 10). It enables communication between the applications to share information.

Definition 2.19: A middleware is a software layer between the operating system and the applications, allowing them to interact locally or in a distributed system by using application programming interfaces (API).

Definition 2.20: A service provides a cyclical or request based output related to a specific input by using one or multiple applications.

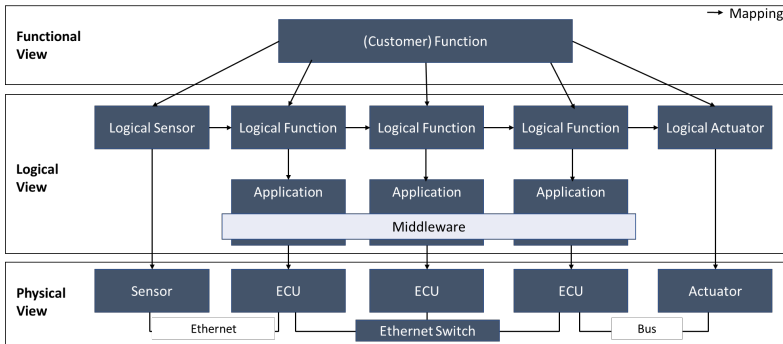


Figure 10: Schematic of a SOA

Physical View: Each application needs a hardware E/E Component as deployment device. This can be realized as point-to-point connection such as between a sensor and an ECU or by using a network device like a switch (see Figure 10). An Ethernet connection is also not only required by the

middleware, it also provides data rates, which are needed for autonomous driving applications (see Table 1). This is the reason why automotive manufacturers are leaning towards middleware based architectures in the future [27].

However, still legacy technologies are required. The traditional bus-systems are proven for safety critical applications and at the same time available at low cost, compared to automotive Ethernet [28].

2.4.2 Adaptive AUTOSAR and ROS2

There is a variety of middleware technologies existing, but only two of them fulfill automotive requirements, namely Adaptive AUTOSAR and ROS2.

Adaptive AUTOSAR: The need to reuse electronic systems and to make them compatible between multiple supplier proprietary solutions raised the demand for a standard, which was successfully addressed in 2003 with the AUTomotive Open System Architecture (AUTOSAR) standard.

Initially the AUTOSAR Classic standard was designed for traditional signal-oriented E/E Architectures. Its successor, Adaptive AUTOSAR, is now targeting towards the realization of service-oriented E/E-Architectures.

The Adaptive AUTOSAR standard is organized in three layers. The lowest level is the operating system interface, which is necessary to use services from the actual operating system, such as Linux or QNX (see chapter 2.2.2). This interface is used by the AUTOSAR Runtime for Adaptive Applications (ARA) which holds, in functional cluster aggregated, APIs to access AUTOSAR services. Above the ARA is the application layer, containing applications which are developed individually to realize a desired functionality holding specific algorithms such as an autonomous driving function.

For this thesis, relevant functional clusters are especially the execution management and the communication management.

Execution Management: The Execution Management cluster is responsible to initialize the platform itself and to start up and shut down adaptive applications [16]. It also monitors their behavior during runtime. In the event of a fault, the application can be restarted by using the health management functional cluster [29].

Communication and SOME/IP: The communication functional cluster holds services which allow applications to communicate with each other (see Figure 10). It contains the Scalable Middleware Over Ethernet (SOME/IP) protocol, Data Distribution Service (DDS), intra-process communication and signal-based communication.

SOME/IP was from its beginning an automotive middleware as part of AUTOSAR classic [30] before DDS was implemented [31]. It uses TCP and UDP as underlying transport protocol [18] and can be used for time or data triggered communication or as a remote procedure call [32].

Robot Operating System 2: The ROS2 project was started in 2013 in order to close the limitations with regard to safety of the initial ROS version [33]. ROS2 consists of multiple layers (see Figure 11).

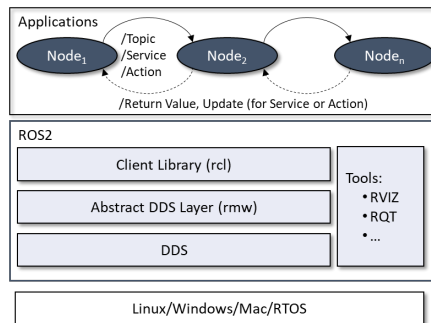


Figure 11: The ROS2 layers

The central element of ROS2 are the nodes, each containing specific algorithms [34]. Nodes communicate by passing messages which are either

published time or event driven as a *topic*. A topic follows the “fire and forget” principle, meaning that the publishing node is not expecting an answer.

Contrary, sometimes a direct response on a request is necessary. ROS2 enables this by its *service* functionality. A node can call a service of another node which then responds to the request. Both, the service call and the response take place again by using messages [34]. If the execution of a service requires some time it is implemented as an *action*. In contrast to the service, where the requesting node waits for the response, an action continues with the option to receive status updates. Once the result is ready it responds to the calling process [34].

Analogous to the explained AUTOSAR functional clusters, ROS2 contains similar clusters [35]:

Executor: The execution of ROS2 nodes can be separated into two portions. Firstly, the startup of nodes including the algorithms inside the nodes and secondly, the execution of the call-backs, necessary to realize the message passing between the nodes. It also provides extended features such as the managed nodes which allow to monitor the status of nodes and to restart or replace a node during runtime in the event of an error [29].

Definition 2.21: *A call-back is a function which is executed by the system based on an event, such as a system timer.*

Communication and DDS: The realization of the communication between nodes takes place within the DDS Layer (see Figure 11). Like SOME/IP, DDS is also a standard with regard to the definitions and APIs. Thus, multiple implementation variants exist (e.g. FastRTPS, RTI Connext, CycloneDDS). The ROS2 rmw layer provides an agnostic interface to each of those DDS layers, so that the differences are not visible to the applications developers using ROS2.

The actual transport of data between two participants takes place by using the Real-Time Publish/Subscribe Protocol. Depending on the specific implementation type of a vendor it either uses UDP, TCP or shared memory [36].

However, if two or more nodes run on the same CPU, DDS selects intra process communication over shared memory instead of UDP or TCP. While UDP and TCP starts at the transportation layer of the OSI model, thereby requiring the splitting of the data into packages through each layer, this has not to be done if using shared memory. This saves costly CPU resources and is also happening quicker.

Definition 2.22: *Shared memory is an option to realize the communication between local processes by defining a memory space to which multiple processes have access.*

Another aspect of DDS is the possibility to use Quality of Service Policies (QoS). QoS further specifies rules of communication between participants. Those rules can be related to different aspects such as the definition of publishing and subscribing intervals or confirmation about received packages by a subscriber to a publisher.

The important QoS for this work is HISTORY which defines the number of historic samples which are required for the individual use case. If HISTORY is activated, a publisher will save the defined number of samples so that it can republish them if a node joins a running system late.

2.5 Automotive Safety

Vehicles are operated mostly in public areas where a dense participation of other road users is typical. The interaction in a vulnerable environment requires special safety considerations.

Definition 2.23: *Safety is the absence of a risk for humans or subject matters. A risk is a state in which a damage could happen inevitably or randomly without the guarantee of appropriate countermeasures [37].*

Definition 2.24: *Safety-critical systems are those, where the loss of life or an environmental disaster must be avoided [38].*

Examples for safety-critical systems are the steering of nuclear powerplants, the automated or wired based steering of aircrafts and automotive systems which influence the lateral or longitudinal control of the vehicle [30].

2.5.1 Safety Parameters

There are two major sources of risk for automotive systems which are covered by ISO standards. First, the risk that a system does not fulfill the desired functionality, described in ISO21448 “Road Vehicles – Safety of the Intended Functionality”. An example for such a risk is an AD system which does not detect if it operates outside of its specified ODD. It would continue to operate even though it was not designed for this purpose. This could happen due to sensor limitations caused by rain, fog or sun. The second source of risk is handled by the ISO26262 “Road vehicles – Functional Safety” and treats the case of a failure in an automotive system. Such a failure can be caused by the wrong design of a system or by a random hardware fault. The focus of this dissertation will be on random hardware faults.

Definition 2.25: *A failure is a deviation of the system regarding its compliance to the system specifications [20, 38].*

Definition 2.26: *An error is an incorrect or inaccurate system state, which is responsible for a failure [20, 38].*

Definition 2.27: *A fault is the trigger of an error, due to a physical defect, imperfection or a flaw that occurs in a HW or SW component [20, 38].*

To avoid a risk, two approaches have been formed: The first approach attempts to find and eliminate all possible faults during the development phase. This fault-avoidance approach is aiming to maximize the safety parameter reliability of a safety critical system.

Definition 2.28: *The reliability of a system is the capability, to provide the specified function within a given time under reliable operating conditions [37].*

A widely used metric to define the reliability of a system is the mean time to failure (MTTF). It defines an average time span from a fault free system until a fault occurs. The Fault In Time (FIT) indicator is another important metric beside the MTTF. FITs represent the number of faults within one billion operational hours [38].

$$\text{FIT} = \frac{10^9}{\text{MTTF}}$$

Ideally, the FIT is zero which would mean, that there is no fault within a billion operating hours. However, this will not be the case since the hardware is exposed to physical effects and to environmental influences. This leads to material fatigue, which at some point will cause a fault.

This forms the need for the second approach which is the fault-tolerant design. A fault tolerant system can cope with faults, keeping up its functionality, thereby maximizing the second safety parameter, availability [30].

Definition 2.29: *The availability of a system is the ratio between the time within the interval $[0,t]$ in which the system works and the entire interval of operation [38].*

For the development of an AD system, it is crucial to maximize the availability additionally to the reliability. An AV on an interstate cannot just stop in lane in the event of a fault since this is a risk for the backwards traffic. It must be able to either reach the next workshop or at least drive from the next ramp

to get into a safe stopping position. A widely used approach to maximize the availability of the system is dynamic reconfiguration.

2.5.2 Dynamic Reconfiguration

Dynamic reconfiguration represents a key element to maximize the availability, thereby making the system fault tolerant.

Definition 2.30: *Dynamic reconfiguration is the process of changing E/E Components during the runtime of a system.*

A fault tolerant system intercepts the propagation of a fault to an error, keeping it working as specified, also in the case of a fault.

Definition 2.31: *Fault tolerance is the ability of a system to continue to perform its specified tasks after the occurrence of faults [39].*

Johnson states [39], that a fault tolerant system needs to fulfill three characteristics. First, it needs to detect faults. Second, it needs to localize and isolate the fault, thereby preventing it from further propagation through the system. Third, the fault recovery is executed to keep the system operational (see Figure 12).

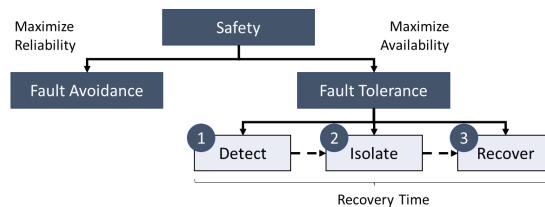


Figure 12: Summary of safety elements for automotive E/E Architectures

A widely accepted approach to design a fault tolerant system is the usage of redundancy which is distinguished between static and dynamic redundancy.

Definition 2.32: *Redundancy is the realization of a function on multiple E/E Components, which all have the same input and output [37].*

Definition 2.33: *Static redundancy is the presence of redundant E/E Components which in parallel to the primary E/E component executes the same function again so that the system could swap between both if necessary [30].*

Each E/E Component calculates results, which are compared by a voter (see Figure 13). If most of the results match, the system considers the result to be correct.

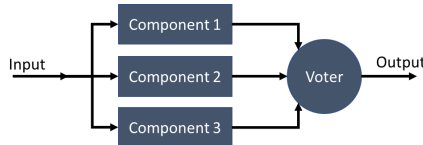


Figure 13: Triple modular redundancy as static redundancy pattern

For the case of three independent components this procedure is called Triple Modular Redundancy (TMR) which is typically realized on three cores of a MCU (see chapter 2.2.1). The ability of a system to continue operation in the event of a fault is also called fault masking [30].

A synonym is the terminology “2 out of 3” (2oo3) since within TMR two out of the three channels must provide the same output for the system to continue. Contrary, the 2oo2 pattern stipulates that both of two channels have to match. This set-up is able to detect faults but cannot identify which channel is the faulty one.

Definition 2.34: *Dynamic redundancy is the presence of redundant E/E Components, which are activated after the occurrence of a fault to keep up the supported process [30].*

In contrast to the static redundancy, dynamic redundancy is not actively contributing to the calculation of results. Dynamic redundancy can be reached by running a secondary component either in hot standby or cold standby. In hot

standby the component is running waiting for its assignment. In cold standby it first needs to be activated what requires more time.

Budget-, spacial- and weight restrictions typically enforce engineers to find compromises between the extend of fault tolerance and the number of redundant components [40]. Depending on this compromised set-up, a system can reach different levels of reconfiguration, namely being operational, fail-unsafe, fail-safe, fail degraded or fail-operational [41] (see Figure 14).

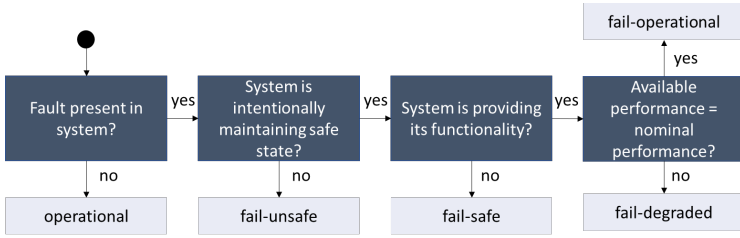


Figure 14: Scheme to distinguish fault tolerance regimes based on Stolte et al. [41]

Definition 2.35: *An operational system has no fault and, thus, can provide its specified functionality with at least nominal performance while maintaining a safe state [41].*

Definition 2.36: *A system is fail-unsafe in the presence of a fault combination if it is not able to maintain a safe state [41].*

Definition 2.37: *A system is fail-safe in the presence of a fault combination if it ceases its specified functionality and transitions to a well-defined condition to maintain a safe state [41].*

A fail-safe design is no solution for systems where no fail-safe state can be reached. An autonomous truck on a highway must be able to continue its mission even in the event of a fault, since a stop in lane as reaction as fault would still be a risk for other traffic participants. A fail-degraded or fail-operational system can overcome this limitation [40].

Definition 2.38: *A system is fail-degraded in the presence of a fault combination if it can provide its specified functionality with below nominal performance while maintaining a safe state [41].*

Definition 2.39: *A system is fail-operational in the presence of a fault combination if it can provide its specified functionality with at least nominal performance while maintaining a safe state.*

Dynamic reconfiguration will play a crucial role for this dissertation. A modern middleware (see previous chapter) allows to realize a novel reconfiguration pattern, which is neither the explained cold nor hot standby pattern. In contrast, the idea of this dissertation is to start up a redundant application but then to pause it, thereby not dispatching the related process to the CPU but having it already loaded into the memory. This is a hybrid redundancy pattern which combines the benefits of low resource consumption of the cold standby pattern with the fast reconfiguration time of the hot stand-by pattern. It will be investigated if this novel reconfiguration pattern still allows to realize a fail-operational architecture as required for safety critical Level 4 (L4) systems.

2.5.3 Functional Safety - ISO 26262

The evolution of automotive E/E systems from comfort functions to safety critical functions led to the definition of the ISO 26262: Road Vehicles - Functional Safety [42]. It addresses possible risk caused by safety-related E/E hardware failure leading into unintended system behavior [43]. It is a standard which contains multiple aspects such as development process definition, hazard and risk analysis methodologies and also mechanisms to treat depended failures. The central element for this dissertation is the treatment of random hardware faults, occurring at predictable rates but at unpredictable times due to material ageing or electromagnetic waves.

ASIL	Random Hardware Failure Target Values	Note
D	< 1E-08/hr (10 FIT)	Required
C	< 1E-07/hr (100 FIT)	Required
B	< 1E-07/hr (100 FIT)	Recommended
A	-	Not required

Table 2: Random Hardware Failure Target Values based on Das and Tylor [44]

Definition 2.40: *A random hardware fault is a fault that can occur unpredictably during the lifetime of a hardware element [42].*

The standard defines different safety levels, starting from Quality Management (QM) up to the highest safety level Automotive Safety Integrity Level (ASIL) D. For random hardware failures, these ASIL goals define thresholds of acceptable failures during a specific duration.

ASIL-B components are recommended to have a FIT not more often than 100 times during 1 billion operation hours (see Table 2). However, an ASIL-D component must not have more than 10 failures during 1 billion operation hours.

The target ASIL for an AD-System is ASIL-D (see dissertation of Sari [43]) because it potentially risks the life of passengers and other vulnerable road users. As conclusion, each SW and HW E/E component must be developed towards this target ASIL goal.

There are two ways to achieve a safety goal for an E/E component. It either could be developed directly towards the corresponding safety requirements or it could be decomposed into two redundant E/E components with a lower ASIL level which together achieve the higher ASIL Level. Possible combinations can be found in ISO 26262-Part 9 [45].

The decomposition principle will be used during this dissertation because the required HW E/E Components are not yet available on ASIL-D level but on ASIL-B.

Beside random hardware faults, the ISO 26262 also describes systematic faults which don't happen randomly but are caused by human made system design errors.

Definition 2.41: *A systematic fault is a fault whose failure is manifested in a deterministic way that can only be prevented by applying process or design measures [42].*

An example for a high risk of a systematic fault is a homogenous design. That means that the system contains multiple components, which are the same. For instance, if three identical ECUs are used, all of them could be exposed to the same systematic fault. If one ECU is different (e.g. with a different SoC), a potential fault could be identified by comparing the results of all ECUs.

A fault tolerant system can reach different levels of reconfiguration after the event of a fault (see Figure 14). ISO 26262 provides a definition about the time from a fault until the end of the reconfiguration.

Definition 2.42: *The Fault Handling Time Interval (FHTI) is the time-span from the detection of a fault to reaching a safe state or emergency operation [42].*

The FHTI contains two execution parts from the moment of the occurrence of a fault. First, the duration until the system detects the fault which is called “Time to Detect Fault”. Second the Fault Reaction Interval, which describes the reconfiguration time to an emergency operation (see Figure 15).

The definition of emergency operation shows, that ISO 26262 considers vehicles to have a safe state while AVs do not have a safe state [43, 46]. For instance, on a highway, an AV cannot just stop in lane in the event of a failure since it would represent a risk to the remaining traffic participants who have to maneuver around it. This shows that ISO 26262 is made for having humans as ultimate fallback solution what brought up the question by Schnellbach [47] if ISO 26262 has to be changed for AVs.

However, for the sake of this dissertation the safe state is a reconfigured system which continues to operate as in the nominal, defined state.

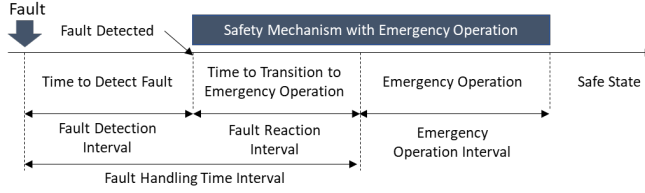


Figure 15: Time intervals of fault handling based on ISO26262 [42]

2.6 Other related Technologies

2.6.1 Convolutional Neural Networks

Deep Learning (DL) is a form of AI which represents a subset of Machine Learning (ML) technologies which led to many breakthroughs in computer vision and robotic applications [48].

ML approaches can be traced back to the single layer perceptron which is based on the work of McCulloch and Pitts in 1943. It was refined and implemented by Rosenblatt in 1960 [49]. The single layer perceptron represents a single brain cell as logical threshold element which fires an output once the input threshold is reached. The single layer perceptron was later extended to the multi-layer perceptron which forms the first simple neural network as ML-technology (see left side of Figure 16).

In each step the value of the input neuron I_n is multiplied with a weight factor v_{nm} and sent to the next neuron H_m . Within the neuron H_m all weighted input values are summed up to a total value H_m . All weight factors of one layer can be summarized in a weight vector.

$$H_m = \sum_n v_{nm} I_n$$

The summed-up weights are then used as input for an activation function, which represents the earlier mentioned threshold. A widely used example is the Sigmoid-Function $f(x)$ [50].

$$f(x) = \frac{1}{1 + e^{-x}} \in (0,1)$$

$f(x)$ then represents the input for the next hidden or output neuron layer. This calculation process is done for each neuron in each layer.

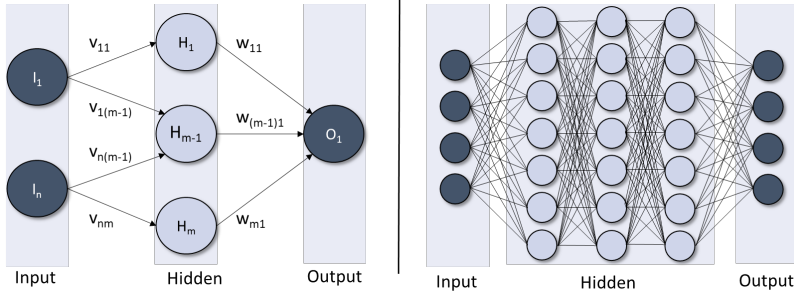


Figure 16: Simple Neural Network (left), Deep Neural Network (right) based on Wuttke [51]

While those mathematical operations can be executed on lower performant CPUs for the simple Neural Networks, this is not possible anymore for the DL - Neural Networks due to the significantly increased number of neurons [51]. However, the mathematical operations between each neuron are always the same but with different data. This opens the opportunity of heavy parallelization on the GPU or on a dedicated integrated circuit such as specific AI-Accelerators (see chapter 2.2.1).

Convolutional Neural Networks (CNN) are mainly used for processing spatial information, such as images. They are DNNs with additional convolution layers at the input. The convolution layer can be seen as filters which exploit local spatial correlations of image pixels to capture discriminant image features (e.g. lines, edges) [48]. Based on those image features, the DNN detects objects on the input image.

2.6.2 Hashing

The transfer of data has brought up multiple mechanisms to validate if it was transferred correctly. One option is the inefficient way of bitwise comparison which would require that the data is sent twice from a sender to a receiver to then compare it bit by bit [52].

To overcome this problem hashing is widely used. It takes input data of any length and maps it to a unique output of fixed bit length. This hash value is then sent to the receiver additionally to the original data [53]. The receiver again calculates the hash with the transferred data again and compares it to the transferred hash.

Definition 2.43: *Hashing represents a mechanism which maps an input of any bit length to an output of a fixed bit length.*

While multiple hashing algorithms exist, the Message-Digest Algorithm 5 (MD5) and the Secure Hash Algorithm (SHA) represent two of the most used [54]. MD5 maps an input of any length to a fixed size output of 128b. Due to the increased computational performance, it could happen that two input bit streams may have the same MD5 output, what is called a collision. This is the main reason why MD5 is no longer used for security reasons [54]. This security concern was overcome by the introduction of the SHA family. While multiple versions exist, one of the most commonly used is the SHA256 converting an input bit stream of any size to a fixed size output of 256 bits [53]. The increase-ment of the output string in size is minimizing the probability of a collision but at the same time requires more compute effort. Thus, the big advantage of MD5 is that it is faster to compute and creates a smaller output [52] what is beneficial to save precious bandwidth of the network technology.

2.6.3 Virtualization

Definition 2.44: *Virtualization allows multiple “virtual” systems to coexist on a single physical machine [55].*

Virtualization represents the abstraction of a physical host machine towards applications. Multiple abstracted host machines can operate on a physical machine. Virtualization provides benefits such as creation of hardware independence towards applications, isolation and increased scalability. Thus, it is also an enabling technology for reconfigurable E/E Architectures and can be realized with hypervisors or containers (see Figure 17).

Hypervisors: Currently hypervisors represent the most common form of virtualization within the automotive domain [55]. The abstracted machine is called virtual machine. Each virtual machine instance carries an own operating system including applications and related dependencies. In principle it is distinguished between two forms of hypervisors. Type 1 hypervisors are also called bare metal hypervisors and they are located directly on-top of the hosts hardware. Type 2 hypervisors, also called hosted hypervisors are allocated on-top of a host operating system.

Containers: Containers offer a more lightweight form of virtualization. They rather isolate processes on host operating system level than whole machines. Each container shares the underlying host operating system thereby creating less overhead. On the other side, that restricts the usage of multiple operating systems on one host machine.

A benefit of using hypervisors is the minimization of the shared resources between the virtual machines. This leads to a maximum of temporal and spacial isolation which is needed for fault tolerant systems. In contrast that also leads to the fact that each virtual machine holds its own operating system what creates unacceptable high overhead within resource constrained environments such as automotive E/E Architectures.

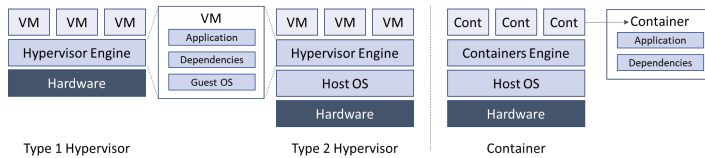


Figure 17: Comparison hypervisors vs. containers based on Morabio et al. [56]

2.6.4 Orchestration

Definition 2.45: *Orchestration represents the initial distribution and the re-configuration of applications within a distributed system.*

According to [57] a recent way to orchestrate functions is the combination with containers. As described above containers provide an encapsulated environment for functions which can be moved inside a distributed system by using an orchestrator (or Cluster Manager). The container can run in hot standby on the back-up machine, or it can be loaded to the back-up machine upon request. Basis again is the IP-based communication (see chapter 2.3.2) by providing each container an own IP-Address. Thereby it is reachable by any other container, regardless of if it is running on the same physical machine or elsewhere in the network.

Orchestration has several benefits. For instance, it allows to scale the functions of a system by adding new containers to the orchestrator. Additionally, containers can be reloaded during runtime or even be run in parallel. In the event of a fault, this mechanism provides isolation since each container leads to spacial and temporal isolation and additionally it allows to shut down and reload a container in the event of a fault.

3 State of Science and Technology

3.1 Autonomous Driving

3.1.1 Physical System Architecture

The following chapter is organized along the structure of Figure 2.

Sensing: A typical AD-System contains, cameras, lidar, radars, ultra-sonic sensors, a Global Navigation Satellite System (GNSS) sensor and an inertial measurement unit (IMU) [58, 59] (see research vehicles appendix 8.6.2, analysis of [15] for industrial research vehicles). In the following only camera and lidar will be further described (for remaining sensors see appendix 8.5) since they are most important for DL applications. The proposed system of this dissertation will be tested together with a DL application using cameras.

Camera: Marti et al. [15] and Liu et al. [58] characterize cameras as low-cost sensors with rich perception features. The camera image provides information which can be used for object classification and tracking [58]. If applied as stereo camera, it can also provide depth information and it can be used for vehicle localization purposes [60] with a sensing range up to 100 m [21]. Depending on the camera it produces 20 – 180 MB data per second [21, 61]. The update rate varies from 25 Hz to 50 Hz [61]. However, the camera image can be affected by low lightning, rapidly changing lightning or bad weather [58]. For this use case specific cameras for autonomous driving requirements have been developed, such as infrared cameras (IR) for effective pedestrian and animal detection or better night visibility. Another example, depicted by [15], are scenes with a high dynamic range (HDR) such as entering or exiting a tunnel, where dark and strongly illuminated areas are in the same frame. The amount of applied cameras varies between 0 - 10, the used network

technology is high speed serial connection, USB2, Firewire or Ethernet (see academic research vehicles Appendix 8.6, analysis of [15] for industrial research vehicles).

Lidar: Lidars have similarities to radars. They also provide distance information, calculated on the time of flight. In contrast to electromagnetic waves of a radar, a lidar uses laser beams. Beside measuring the angle and distance toward a target, those laser beams can also be used to create a three dimensional image of objects [58] or a three dimensional point cloud of the environment [15]. It is distinguished between mechanical lidars and solid-state lidars. While a mechanical lidar uses rotating lenses and can achieve a sensing angle up to 360°, a solid-state sensor has a constant sensing angle of up to 120°, thereby being more robust by the elimination of mechanical components [61]. The sensing range reaches up to 200 m and it typically produces between 10 – 70 MB data per second [58]. The update rate varies from a mechanical to a solid state lidar from 5 to 50 Hz [61]. According to [15], lidars have limitations regarding the detection of small objects, dark and specular objects since dark colors absorbs the emitted laser beam. Also, lidars are affected by weather conditions such as rain, fog and snow. In such kind of situation a lidar operates worse than a radar, but still better than a camera [15]. The amount of applied lidars varies between 0 - 13, connected with CAN, USB2 or Ethernet. (see analysis academic research vehicles Appendix 8.6, analysis of [15] for industrial research vehicles).

	Camera	Lidar
Range	<= 100 m	<= 200 m
Data Rate	10 – 180 MB/s	10 – 70 MB/s
Update Rate	25 -50 Hz	5-50 Hz
Bad weather	Poor	Fair
Low Lightning	Fair	Good
Number	0-10	0-13

Table 3: Comparison sensor characteristics adjusted from [58]

Processing (Computation Hardware): The analysis of the research vehicles showed, that up to 10 personal computers (PC) are used for AVs, connected

with Ethernet (see analysis academic research vehicles Appendix 8.2). Those PCs are used to run the AD Software Stack. However, this number is not considering redundancy for safety purposes. This will lead to the redundant operation of L4 applications what requires additional compute demand (see chapter 2.5.2).

Both, standard Ethernet and standard PCs are no automotive grade solutions. Several automotive suppliers developed first automotive grade ECUs, containing dedicated SoCs which are powerful enough to execute AD-Software functions. Additionally, such an ECU contains a safety MCU, allowing the monitoring of software functions on the SoC and the hardware of SoC itself. This dual chip approach is important to realize safe architectures according to ISO 26262 [62].

Currently, one of the most powerful automotive grade SoCs is the NVIDIA Orin. It is certified for the CPU and GPU up to the level ASIL B. It contains 12 ARM A78 Hercules CPU cores operating up to 2.2GHz, a GPU and two dedicated Deep Learning accelerators (see chapter 2.2.1) which together are capable to operate 250 Trillion INT8 Operations¹ (TOP) per second, consuming 200W of power. It has 32GB RAM and 256 Flash Memory. It has a dedicated input for 16 cameras using the Gigabit Multimedia Serial Link (GSML) connector [63].

The Orin is extended by a supervision MCU which is the Infineon Aurix TC397, together forming a typical automotive ECU (see Figure 18). The Aurix is certified to ASIL D due to its 6 lockstep capable cores which operate at 300Mhz. The Aurix has 6.9MB RAM and 16MB Flash Memory. It has standard interfaces such as Ethernet, FlexRay, CAN and LIN.

While the Aurix has a higher safety level compared to the Orin it does not provide sufficient resources for the needs of AD-Software. The missing GPU,

¹ TOPs is a metric to evaluate the performance of a GPU or HW accelerator. INT8 represents the datatype for the weights of a DNN (see chapter 2.6.1)

the low memory and the low clock speed are significant limitations not making it suitable to be used for processing the in the previous chapter defined AD-Software. Out of this reason the idea of this dissertation is to combine two Orins by using the ASIL decomposition principle so that together they achieve the same safety level as the Aurix but still have sufficient performance to operate the AD-Software.

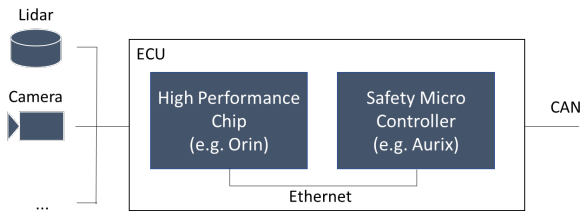


Figure 18: Schematic of an AD ECU

Acting: The result of the processing layer is information such as vehicle commands, which is forwarded as direct steering commands to the responsible ECUs.

Example: An AV is travelling on a straight lane on an interstate. The allowed speed is 80 km/h and the vehicle is traveling in the middle of the lane, with a yaw angle of -1° . The steering command to the vehicle in this case is Speed = 80km/h, steering angle = $+1^\circ$.

Typically, those ECUs reside in existing domains such as the powertrain (e.g. acceleration) or chassis domain (e.g. steering, braking). Those ECUs are traditional signal based E/E Components (see appendix 8.7), thus using CAN as communication technology [58] (see research vehicles 8.6.2).

3.1.2 AD Software Stack

The highest level of AD Software Stack abstraction was done by Paden [64], who differentiates between the perception module and the decision module.

While the authors didn't further refine the perception part, they decomposed the decision module into four sub-modules: Route Planning, Behavior Planning, Motion Planning and Local Feedback Control. Local Feedback Control executes the commands of the previous modules and provides feedback regarding the execution status.

The second high level abstraction model is the Observe, Orient, Decide, Act (OODA) model. It was initially developed for military usage [65] and later represented the basis for most of the vehicles in the DARPA urban challenge [66]. This approach can be aligned with [64] by allocating Observe and Orient to Perception and Decide and Act to Decision.

Badue et al. [60] follows the segmentation of Paden et al. [64] and further decomposes the perception module into localization, moving obstacles detection, mapping and tracking as well as traffic signalization detection. This represents a mixture between modules (localization) and applications (moving obstacles detection, traffic signalization detection).

Another further refinement of the perception module can be found in the work of Behere et al. [65]. The authors decomposed it into the sub modules sensing, sensor fusion, localization, semantic understanding and the world model.

Liu et al. [58] state that a typical AD-Software architecture is modular. In contrast to the other mentioned authors, they consider object or lane detection, localization and mapping, prediction, planning and vehicle control to be applications of the AD software stack.

Another variation can be found in the Autoware.Auto autonomous driving software stack, which is an open-source project and hence represents the opinion and understanding of voluntarily working software engineers for autonomous driving. It differentiates between sensing, perception, decision, planning and actuation [67]. The perception module consists out of localization and obstacle detection, as well as prediction of future sceneries. The

information from this module will be transferred to the decision module which brings the vehicle into a certain driving state such as “follow the lane” or “unprotected left turn”. The planning module is divided into the mission and motion module. The mission module plans the meta route of the vehicle, whereas the calculation of the precise motion trajectory is executed in the motion module.

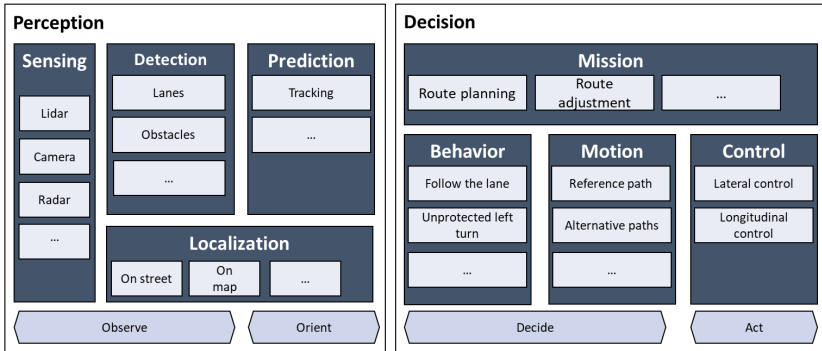


Figure 19: Modules of a typical AD software stack

The findings from the above mentioned delimitation of an AD-Software are combined in Figure 19, underscoring the modular design pattern. Each module contains multiple L4 applications which produce information which is relevant for one or multiple other modules.

3.1.3 Dataflow between the Modules

The input into the AD Software stack is represented by the sensing module. Each sensor has a certain update rate (see chapter 3.1.1) thereby representing a cyclic input (in form of an image, lidar point cloud, radar point cloud, ...) into the sensing module.

Starting from this input, the data is forwarded through the modules of the AD Software stack and consumed by L4 applications. Typically, in the next step it is used for localization or detection purposes. For instance, the behavior of

the vehicle is derived from the objects around the vehicle. The detection module detects those objects, using the input from the sensing module and forwards the object list to the behavior module (see Figure 20).

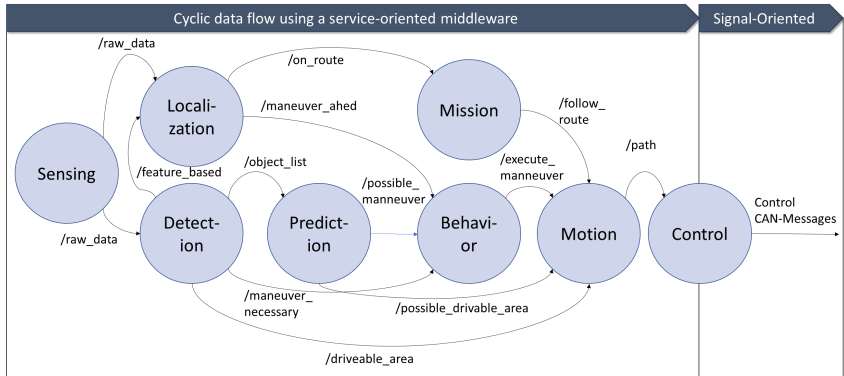


Figure 20: Exemplary data flow between the modules of the AD software stack

Each module contains multiple L4 applications. Exemplarily, the localization of the vehicle typically takes place using multiple technologies. For instance, it can localize itself by using data from the GNSS device. In parallel it also localizes itself on a lane by using the lane markings coming from the detection module.

This distributed communication pattern requires an efficient technique to realize the information flow between the L4 applications inside the modules. Promising solutions are recent middleware technologies [58], which abstract the AD-Software from the underlying hardware. This approach allows to run and reallocate the applications on multiple devices within a distributed system (see Figure 21).

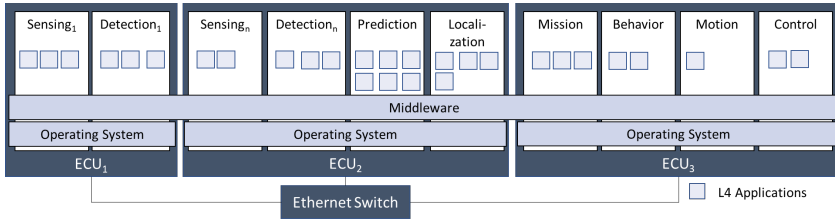


Figure 21: Exemplary mapping of applications to modules to ECUs

Examples for middleware solutions which have been used for AVs are KogMo-RTDB [68], Lightweight Communication and Marshalling [69], the VL-Bus [70], ZeroCM [71], OpenDaVINCI [72] or the Robot Operating System (ROS).

Since the above-mentioned solutions suffer from limitations with regard to safety, the most promising frameworks towards production ready AVs are the Scalable Service-Oriented Middleware over IP (SOME/IP) in combination with Adaptive AUTOSAR or (commercial versions) of the Robot Operating System 2 with a Data Distribution Service (DDS) Middleware (see chapter 2.4.2).

3.1.4 Applications inside the Detection Module

While a typical AD Software Stack contains multiple software modules (see chapter 3.1.2, appendix 8.2) for this dissertation the focus will be on the detection module using camera data as input. The progress within the research of artificial intelligence lead to separate between traditional and DL based detection methods [73].

Traditional computer vision algorithms are feature related. Based on similarities, such as colors, shapes or textures, features are extracted from an image [74]. Typical algorithms for feature extraction are Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), Features from Accelerated Segment Test (FAST), Hough Transforms and Geometric hashing [75]. The peak of performance for traditional algorithms was the Deformable Part-based Model (DPM) algorithm, which won the Pascal Visual Object Classes

Challenges (VOC) in the years 2007, 2008 and 2009 with an object detection accuracy of 21% in the 2008 challenge [73].

The traditional detection algorithms are more and more replaced by DL-based algorithms (see chapter 2.6.1 and appendix 8.5). One of the first algorithms was the Regions with CNN features (RCNN) algorithm in the year 2014. It was applied to the VOC data set of 2008 in order to compare its accuracy to DPM. With an accuracy of 58,5% (compared to 21% DPM) it clearly can be seen that DL based approaches by far outperform traditional computer vision algorithms [73]. Since then, the technology continuously has been further refined. Multiple algorithms have been developed such as Fast and Faster RCNN, Alex Net, You Only Look Once (YOLO), Retina-Net and Refine-Net [73] which are popular also for object detection within autonomous driving [48].

For DL based approaches, beside 2D images, also 3D data as point cloud, either directly from a lidar or derived from a stereo-camera, can be taken as input for DL-algorithms. PointNet and Voxel Net are two examples of algorithms which detect objects and additionally provide depth information based on 3D data up to a precision of 3 cm when using a lidar. However, since 3D data alone does not contain rich visual information for object classification it is typically used in combination with above mentioned 2D visual classification methods to receive both, a high accuracy on detected objects as well as a high accuracy on the corresponding depth information [48] (see sensor fusion chapter 8.2).

Regardless of whether camera or lidar data is used for object detection, both would require a GPU to be utilized instead of a CPU. Thus, traditional lockstep microprocessors are not appropriate to operate those applications (see chapter 2.2.1). A possible solution is the novel approach of this dissertation which uses modern SoCs to operate the DL applications in an ASIL D out of B decomposition set-up.

3.2 Related Safety Research

3.2.1 Traditional Safety Architectures

S. Bak et al “The System-Level Simplex Architecture for Improved Real-Time Embedded Systems Safety” [76]

A fundamental work to realize safety critical systems is the Simplex-Architecture. The Simplex Architecture contains a subsystem which is performing the actual nominal functionality of the system called the complex subsystem. A decision logic detects faults within the complex subsystem and changes to the safety subsystem in the event of a fault. Bak et al. [76] ported this basic layout to multiple processors with different safety levels in order to realize a fail-operational behavior. In particular, the safety critical decision logic and the safety subsystems are operated on a dedicated Field Programable Gate Array (FPGA) with a degraded functionality. The authors proved their novel approach with a cardiac pacemaker.

The experiment contains a fault detection mechanism which is not related to a SOA. It is further not explicitly described how the fault detection is executed. Since it reacts with the degradation of the system (see chapter 2.5.2) it is rather a fail-degraded than fail-operational.

F.K. Bapp et al “Towards Fail-Operational Systems on Controller Level Using Heterogeneous Multicore SoC Architectures and Hardware Support” [77]

The authors see a significantly increasing demand of safety critical functions within automotive applications with the upcoming L4 and L5 AVs [77]. They presented an extension of the simplex architecture towards a fail-operational system by porting it onto a heterogeneous multicore platform. The control and the fallback system was placed on a triple-redundant MicroBlaze processor. The inertial system (similar to the Complex System) is placed on a real-time processing unit and the state transfer takes place in a dedicated programmable logic.

The authors showed that the system was able to detect and react on faults by switching to the fallback system. However, again here, the fault detection mechanism is not explicitly described, and it is not related to a SOA. The authors showed that the system could change back to the complex system due to a not described recovery mechanisms. However, in the event of a fault, the system first continues to work with degraded functionality before it recovers to the complex sub-system. Out of this reason it is rather fail-degraded than fail-operational (see chapter 2.5.2).

D. Niedballa "Concepts of functional safety in E/E architectures of highly automated and autonomous vehicles" [78]

As part of the UNICARagil project, sponsored by the German Ministry of Education and Science with a duration from 2018-2023, Niedballa et al [78] are proposing a safe architecture for L4 AV. He is following the dual duplex approach (see chapter 3.3) by using two Zynq Ultrascale+ SoCs. The architecture of the UNICARagile project distinguishes between the Cerberum, which contains the autonomous driving algorithms such as behavior or trajectory planning, the brainstem which is closer to the actuators and tracks the trajectory execution and finally the spinal cord providing the commands to the vehicle such as steering angles [79]. The proposed safety architecture of this work focuses on the spinal cord part, where over the MCU cores of the Zynq SoC potential hardware faults are detected. In this case the whole SoC would be muted and swapped over to the other SoC.

The experiment set-up guarantees that the trajectory which is sent to the actuators is correct. However, it does not avoid that the incoming trajectory of the service-oriented cerebrum may contain faults. In this case a fault would propagate from the cerebrum to the brainstem. Furthermore, by using the dual-duplex approach and without a recovery mechanism the set-up can manage one single fault. Even though the authors claim that the dual-duplex approach is using a minimal number of redundant components this is not the case since the dual-duplex always executes an operation on four cores which is one core more than with the TMR model.

A. Schnellbach “Fail-operational automotive systems” [47]

In his dissertation Schnellbach [47] analyzed fail-operational architectures from other domains such as avionics, agriculture or railway and compared it to the automotive domain. A key-finding of the author is, that a redundancy concept must be developed carefully and only for spots in the architecture where fault tolerance is necessary and reasonable. He states that a full redundancy, as within TMR, is not always necessary. Additionally, he analyzed the safety standard ISO 26262 (see chapter 2.5.3) according to its validity regarding fail-operation. Equal to Sari [43] he came to the conclusion that the standard only provides little guidance towards fail-operational systems. The author states that ISO 26262 rather treats fail-safe architectures as content than fail-operational architectures. Additionally, the author analyzed existing architectures regarding their fail-operation capabilities by using mathematical models.

The work provides a good orientation to what extend existing automotive safety standards apply to fail-operational architectures. It also provides a good insight into the required redundancy level and potential high level fault detection mechanisms for different use-cases within the vehicle, claiming that full fail-operational is not necessary in most cases. However, the work is not focusing on a SOA and it does not contain a prototypical implementation. The minimization of redundant components is only described in a way that full redundancy is not necessary for all vehicle functions.

S. Orlov et al. “Automatically reconfigurable actuator control for reliable autonomous driving functions”, AutoKonf Project [80]

An attempt to minimize the number of redundant components can be found in the research project “Automatically reconfigurable actuator control for fail-safe automated driving functions” (AutoKonf), sponsored by the German Ministry of Education and Science with a duration from 2016 to 2019. The main idea was, to avoid the duplication of all ECUs and thereby reduce the cost of

ECUs by developing a fallback ECU which is able to realize functions of two other ECUs, namely steering and braking [80].

The project contributed to the problem of the multiplication of components required by a fail-operational architectures. However, it does not contain any fault detection mechanisms rather than listening on the CAN if the primary ECUs are still active. Additionally, this takes place on the signal-oriented layer and does not contribute to the question of a safe architecture on the service-oriented level.

3.2.2 Service-oriented Architectures

F. Oszwald "Dynamic Reconfiguration Method for reliable real-time capable embedded Systems in Automotive" [30]

In his dissertation Oszwald [30] presented a holistic approach for the development of dynamically reconfigurable E/E Architectures. Its core is a library which contains software-, hardware and methodology elements to develop dynamically reconfigurable systems. In his implementation example the author further refined the previously described extended simplex architecture of [77] by encapsulating the sending of CAN messages into a service, running on the high performance chip of the platform. He attached a second development board to run the application redundantly, communicating with SOME/IP and he chose the steering of an AV as demo example. The extended simplex can detect whether the service is alive or not. In the event of a fault, it will inform the application layer which then changes the service provider to the second high-performance ECU, thereby striving to be fail-operational.

With his implementation example the author embedded the generation of CAN signals into an application which runs within a SOA. However, the fault detection covers lockstep errors within the signal-oriented extended Simplex CAN-Provider which runs on a newly developed proprietary programmable logic. It can detect if a message conversation within the signal layer was faulty

and triggers the upper service layer to use the fallback “Extended Simplex CAN-Provider” service. Accordingly, the fault detection mechanism is described but, again here, it takes place on the non-service-oriented part of the architecture. The author implemented a service-oriented E/E-Architecture and proved that dynamic reconfiguration is possible with SOME/IP. However, after the configuration the system is in a fail-unsafe state (see chapter 2.5.2) due to the missing continuous redundancy.

T. Bijlsma et al “A Distributed Safety Mechanism using Middleware and Hypervisors for Autonomous Vehicles” [82]

The E-Gas safety concept² was used and extended by Bijlsma [82] by transferring it in a way that the functions of the system are realized in a publish/subscribe pattern. The primary function (L1) thereby published a topic what is monitored by a supervising function (L2), which published on a diagnostic topic to a safety MCU. This supervising function (L2) is again supervised by a function (L3) which is localized on the MCU. The setup was able to detect multiple plausibility errors by verifying the output of the L1 function by the L2 monitor. For instance, it detects if the acceleration is higher than 100% what would be implausible. For the implementation the authors chose DDS and the lightweight version DDS-XRCE which can also run on the safety controllers.

The authors showed that the system was able to detect multiple faults and could react with different countermeasures by using a service-oriented middleware. For instance, it can restart the whole ECU or, in the worst case, the safety controller would take over the steering task by executing an emergency stop. However, the error detection just covers plausibility checks whether an indicator is within a specified range. It does not contain fault detection regarding reliability. An indicator (such as acceleration) can be within the

² The E-Gas concept was developed by a working group of major automotive manufacturers to ensure that no unintended acceleration of the vehicle happens in combination with electric combustion engine gasoline injection systems [81].

specified range, but it does not have to be correct. Due to the missing redundancy the system is rather fail-degraded than fail-operational. It can be argued that the system even is fail-unsafe since a stop in lane is not a save maneuverer for vehicles travelling on high-ways.

Y. Fu et al "A Formally Verified Fail-Operational Safety Concept with Degraded Modes for Automated Driving" [46]

An attempt to close this limitation of the distributed safety mechanism was done by Fu et al. [46]. They added redundant sensors (safety sensors) to the system and a redundant network to make it fail-operational. Depending on the impacts of the failure, the vehicle would either execute a detour, a comfort stop, a safe stop or an emergency stop.

With their work the authors showed that the distributed safety mechanism is capable to be fail-operational in the event of a failure of a sensor. A failure within a L1 function itself is not covered due to the missing redundancy of the ECU. Analogous to Bijlsma et al. [82], still in that case the distributed safety mechanism would pass over the steering of the vehicle to the safety controller which executes an emergency stop, resulting in a fail-degraded (or unsafe) behavior.

M. Li et al "Fail-Operational Steer-By-Wire System for Autonomous Vehicles", UNICARagil Project [83]

Another attempt for a fail-operational architecture was done during the UNICARagil project by Li et al [83]. Similarly to Bijlsma et al [82] and Fu et al. [46] the authors also chose the layered concept of the E-Gas monitoring concept. To minimize the redundant components, they implemented a steering control algorithm on two multicore processors (see chapter 2.4.1) where on the first processor the control algorithm was operated on core two while core one and core three contained the e-gas monitoring functions. In the event of a detected fault the second processor took over. The arbitration took place with a multiplexer.

The experiment provided contribution to the fault detection and to the fail-operational execution of a motor control algorithm on two safety MCUs using a multiplexer as arbitration logic. With this set-up the authors tried to reduce the number of redundant components for a fail-operational operation. However, their set-up needs five cores on two MCUs. The TMR would only need three cores. Additionally, the used MCU has limited resources and is not appropriate for functions which have to be placed on the high-performance chip (see chapter 3.1.1). Accordingly, the set-up can be used for non-compute intensive algorithms but not for all L4 applications.

T. Stolte “Towards Safety Concepts for Automated Vehicles by the Example of the Project UNICARagil” [84]

Another work as part of the UNICARagil project is the proposal of a safe architecture where in Stolte et al. [84] proposed an architecture which is focused on behavioral safety, dealing with the question of safe vehicle behaviors and only barely touching functional safety.

S. Kugele et al [1] “Data-Centric Communication and Containerization for Future Automotive Software Architectures” [85]

The authors of the work “Data-Centric Communication and Containerization for Future Automotive Software Architectures” [85] executed an experiment where they implemented and tested a publish/subscribe architecture using DDS. They containerized services using Docker and distributed them on four Raspberry Pis, which were connected with a 1 Gbit/s Ethernet. They tested the OWNERSHIP_STRENGTH quality of service (QoS) of DDS which allows publishers to simultaneously publish information on the same topic. The consumer only consumes the information from the publisher with the highest OWNERSHIP_STRENGTH. Thus, once this service breaks down, the consumer will consume the information from the provider with the lower OWNERSHIP_STRENGTH. In the experiment, small information packages were published in an interval of 10ms from two different devices. The authors randomly shut down the publisher and measured the time, until the consumer

processed the back-up publisher. The experiment was executed 1.000 times with an average recovering time of 16.8ms and a maximum measured recovery time of 41.5ms. The standard deviation was 10.4ms.

The author strived to realize a service-oriented E/E-Architecture style, targeting to be fail-operational. In the event of a fault the set-up was able to continue to work without limitation. However, their experiment solely increased the availability. They didn't include any fault detection mechanism as basis for reconfiguration. Thus, his architecture is rather highly available than fail-operational. Additionally, it is not addressing the problem of the high amount for required redundant components.

S. Kugele et al [2] "Elastic Service Provision for Intelligent Vehicle Functions" [86]

The experiment of Kugele et al. [85] was later extended again by Kugele et al. [86], deploying a local service provider on a Raspberry Pi and the fallback of the service in the cloud. The consumer was also deployed locally on another Raspberry Pi. The service published small data packets in an interval of 50ms. Analogue to his previous work [85] the primary service was shut down randomly, measuring the time until the consumer receives packets from the fallback service from the cloud.

Within 100 test runs, the average takeover time was 131.2ms, the fastest take over time was 103.2ms and the slowest one took 181ms. The authors state, that the take overtime is heavily impacted by the definition of the LIVELINESS QoS which regulates the duration, until the consumer considers a service to be dead. The results are hard to interpret for the usage of autonomous driving since the authors left open how the primary-system was connected to the cloud. No information with regard to the internet-connection speed, the usage of a hard wire or wireless network or the physical location of the cloud-server was mentioned. Furthermore, the authors used a middleware for their experiment, thereby using a fundamental element of a service-oriented E/E-Architecture. However, they did not explicitly bring their experiment in the

context of a SOA. Additionally, similarly to his previous work [85], the experiment didn't contain any fault detection mechanisms, thereby again representing rather a high available architecture, not a fail-operational.

In this work the authors increased the availability of the system by running software in the cloud. In general, the experiment was realized in a SOA style. However, again here, it was only touching the aspect of availability. Furthermore, the results show, that the reconfiguration time is unfavorable long for autonomous driving.

F. Oszwald et al "Model-Based Design of Service-oriented E/E-Architectures for Reliable Dynamic Reconfiguration" [87]

Oszwald et al. [87] presented a model-based architecture for reliable dynamic reconfiguration. The experimental set-up was similar to Kugele et al in [85] and [86] where one ECU provides the nominal service and another ECU, connected via Ethernet, provides the fallback service. In this case it was a trajectory planning. A third ECU was connected via Ethernet holding the client application which again was connected to a fourth ECU with CAN, thereby representing the service-signal interface. In contrast to Kugele et al. [85], all three service-oriented ECUs were connected with SOME/IP as middleware. The authors simulated a breakdown of service 6.000 times and defined 150ms as maximal boarder for reconfiguration, starting from the service provision of trajectory data, until the transmission to the brake actuator. The worst execution time of the experiment was 147,3ms.

Same as with Kugele et al [85] [86] this experiment only contains parts of a fail-operational architecture, namely the increment of the availability. since it does not contain contribution to the question about the fault-detection on service level.

B. Liu “Towards an On-Demand Redundancy Concept for Autonomous Vehicles Functions using Microservice Architecture” [88]

Similar research has been done by Liu et al. [88] where the authors investigated the usability of Docker containers in combination with Kubernetes for reconfiguration. The authors created a heterogeneous network of 4 devices, two Raspberry Pi 3B, one Raspberry Pi Zero and a Coral 2. The application together with the container was deployed on one of the Raspberry Pis 3Bs. The orchestrator and the back-up container on the other. The Coral 2 and the Pi Zero were used for other experiments, not relevant for this thesis. The benchmark covered two scenarios. In the first scenario, the back-up container is in cold standby, in the second the container is in hot standby (see chapter 2.5.2).

It was shown, that after the shutdown of a machine, it took the orchestrator an average of 14 seconds to detect it. Another 30 seconds were necessary, to confirm that the machine was shut down. That makes 45 seconds recovery time for the scenario where the backup application is in hot standby. For cold standby, the container image needs to be pulled, created and started what took additional 25 seconds.

The experiment showed that the orchestration of containers is helpful to improve the overall availability of the system, especially in a heterogeneous environment. However again here, no fault detection mechanism was implemented. Even though the author put his experiment into relation to a SOA he leaves open how containerization contributes to the realization of a SOA. He is mentioning QoS settings which potentially refer to a middleware but does not specify if he used one.

Schindewolf et al “Toward a Resilient Automotive Service-Oriented Architecture by using Dynamic Orchestration” [89]

As part of the project “InnovationsCampus Mobilität der Zukunft” of the federal state of Baden Württemberg, Germany, Schindewolf et al. [89] proposed

a resilient architecture which is keeping a system operational in the event of a fault. The concept is based virtualization technologies using Kubernetes in combination with ROS2.

The proposed concept touches the topic of fault masking, however in a resource constrained environment such as an automotive embedded vehicle this approach is too resource intensive. Furthermore, the work does not explicitly describe how a potential fault can be detected.

H. Stoll “The (re)configurable Vehicle Architecture” [29]

In his dissertation Stoll [29] worked out a SOA which allows the exchange of hardware and software components of a vehicle during runtime or in standby of the system. As implementation use-cases the author extended an existing E/E Architecture, replaced existing hardware components, extended existing functions and allocated services in the cloud. Even though those use-cases didn’t touch safety features, the author indicated that hardware and software components can be operated redundantly, thereby providing the basis for safety critical applications. For his implementation, the author chose ROS2 and therewith DDS as middleware. Services were embedded into containers, which were managed by the orchestrating tool Kubernetes.

With his work, the author demonstrated that service-oriented E/E-Architectures can be reconfigured, also if they are combined with traditional signal-based system subnets. However, as already stated, his research did not touch safety aspects beside mentioning the redundancy aspect and the option to use containers to recover applications.

SoftDCar:

Another publicly funded project is the Software Defined Car (SoftDCar) project. With its four workstreams Data-Loop, Re-Deployment, Digital Twin and Deonstrator it touches the aspects of a SOA but is not further drilling into functional safety.

3.2.3 CPU Lockstep Architectures

B. Sari “Fail-operational Safety Architecture for ADAS/AD Systems and a Model-driven Approach for Dependent Failure Analysis” [43]

In his dissertation the focus of Sari [43] was on the development of a model-based fail operational architecture for L4 and L5 AVs. The author investigated the ASILs of the autonomous driving functions and came to the conclusion, that all of them have the highest safety level which is ASIL-D. He proposed to run the functions redundantly on multiple separated high-performance cores, comparing the results on a safety MCU (see chapter 3.1).

The author provided a concept, how to realize a fail-operational architecture on multicore level. However, his proposal is focused on the distribution of redundant components on multiple high-performance cores which results are compared on a MCU. By doing so random faults on the high-performance chips are detected and can be mitigated. To what extent this approach can be integrated into a service-oriented E/E-Architecture was not touched. Additionally, this approach does not minimize the number of redundant components since a fully TMR approach was proposed. Furthermore, the author didn't touch the aspect of software modularization and also not the problem of applications requiring a GPU.

F. Kempf “An Adaptive Lockstep Architecture for Mixed-Criticality Systems” [90]

A similar approach as Sari is proposed by Kempf et al. [90] where the authors could change the configuration of a CPU from single core operation to lockstep clusters during runtime, depending on the criticality of the application. This adaptive lockstep mechanism created between 1.46% to 5.28% overhead compared to an operation without lockstep.

The experiment was applied to simple applications such as sorting algorithms and is not demonstrated to work with autonomous driving applications.

Additionally, it focuses on the lockstep of an CPU which is not ensuring the integrity of the GPU.

3.3 Avionic System Design Patterns

H. Fluehr “Avionic and Air Traffic Control Technology” [91]

In his textbook Fluehr [91] categorized mainly three architecture styles for avionics. Namely duo-duplex, Quadruplex and Triple-Triple Redundancy.

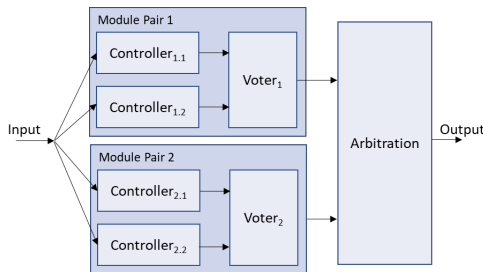


Figure 22: Duo-Duplex Architecture based on Schnellbach [47]

The Duo-Duplex architecture (see Figure 22), which was used by Airbus [47], is characterized by two module pairs, each holding two individual controllers. Both controllers are connected to a voter which checks if the module pair is working correctly. In the event the voter detects an error, the arbitration mechanism switches over to the remaining module pair to keep the system working nominal. This concept allows one fault, still keeping the system operational. This is equal to the already introduced TMR (see chapter 2.5.2).

The quadruplex architecture (see Figure 23 on the left side), has been used by the NASA space shuttle [92] and is based on the 3oo4 concept. It contains four individual channels, each having a controller and a voter. The channels are connected with four bus systems. Each controller sends its result through its own bus. Each voter reads the output of the own channel, but also from the other channels on the respective buses. In case a voter detects that the own

message does not match to the message of the other channels it will shut itself down.

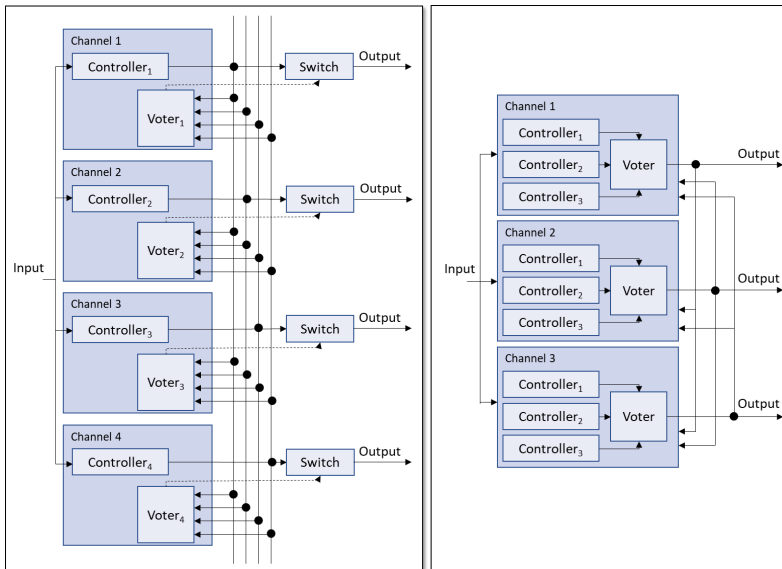


Figure 23: Quadruplex (left) and Triplex-Triplex (right) based on Schnellbach [47]

The third redundancy concept which will be introduced is the triplex-triplex model which is used in Boing 777 [93]. It is characterized by three parallel channels, each of them containing three different controllers. One channel is operational, the remaining two channels are used for validation purposes. Similar to the Quadruplex architecture each channel puts its result on a bus, thereby making it available for the other channels. Each voter compares the results of its own channels and also compares its local result with the results from the other channels. In the event of a detected fault, the voter shuts down the effected channel, thereby making one of the remaining channels operational. In this concept even two channels can be defect and the system stays operational [91] (see Figure 23).

3.4 Critical Appraisal

The analysis of the state of science and technology identified mainly three areas. First, the research area of SOAs (A1), second the area of automotive safety architectures (A2) and third other safety research for automotive applications (A3) (see Table 4). Inside those areas mainly three dominant topics have been identified, namely the initial set-up of SOAs, the fault detection and the fault masking. However, ASIL levels to achieve safety (contribution to RQ1), system recovery mechanisms to maximize the availability of the system (contribution to RQ2) and minimization of redundant components to consider restrictions of weight, power consumption or cost (contribution to RQ3) have only been very barely touched.

The research of SOAs (A1) was heavily focused on the value which a service-oriented middleware provides. This value mainly has two aspects. First, it was showcased that a middleware is the key component for the design of SOAs. This included aspects such as required bandwidth or updatability for future vehicles. Second, aspects of SOAs being fault tolerant have been analyzed. In particular, the novel reconfiguration features of different middleware technologies or related technologies such as containers in combination with orchestrators have been investigated. The results indicate, that SOAs are the architecture pattern for future vehicles and thereby is selected to contribute to RQ4. However, the following two points regarding SOAs are unanswered and require further research.

First, the investigated mechanisms of fault masking focused mostly on the reconfiguration part of a fault tolerant system by restarting middleware applications, containers or virtual machines. However, before a component can be reconfigured, a fault must be detected. The SOA fault detection mechanisms have been only plausibility checks. In fact there is no work focussing on the detection of ISO 26262 random hardware faults for AV applications. Second, the proposed concepts are either using cold standby patterns, which are slow, or they use hot standby patterns, which are

resource intensive. Accordingly another way must be found to design a fail-operational system with low recovery times and a small number of redundant components.

Contrary to A1, fault detection was part of the research for safe automotive architectures (A2). It was typically realized on multicore MCUs where the cores are operated in lockstep and each application runs on multiple cores redundantly.

Those MCU based concepts can not be used for AVs since their applications require more CPU power than current MCUs can deliver. Additionally, those MCUs don't have a GPU which is necessary for L4 applications to run DL applications (see chapter 2.2.1, chapter 2.6.1, chapter 3.1.4, appendix 8.4). Lastly, the parallel operation on multiple cores again represents the too resource intensive hot standby pattern which can't be used due to restrictions in power consumption, space and costs within AVs. As result the existing solutions and technologies for random fault detection are not suitable for L4 systems.

A3 is neither referring to traditional fault tolerant systems nor to SOAs. It is mainly transferring the concept of multicore usage as used in the MCUs to higher performant CPUs. Again here, the permanent operation of software in a fail-operational set up is a resource intensive 2oo2 hot standby pattern. If looking on the Orin (see chapter 3.1.1) this would mean that from initially 12 CPU cores only 4 could be used in parallel since the rest is busy with redundancy. In addition to the argument of the resource intensive hot standby, it can be doubted that 4 cores are sufficient for LD Applications. Also, it does not solve the problem of random hardware faults which may happen in the GPU.

In summary, there is yet no concept existing, which can detect ISO26262 random hardware faults while processing a resource intensive L4 application. Furthermore, there is yet no fault masking technology existing, which makes

the system fail-operational but at the same time does not triple the total amount of components.

	SOA	Fault Detection	Fault Masking	System Recovery	Min. Components	Safety Goals (ASILs)	
<i>This Dissertation</i>							
<i>S. Kugele [1] et al</i>							A1
<i>S. Kugele [2] et al</i>							
<i>F. Oszwald et al</i>							
<i>H. Stoll</i>							
<i>F. Oszwald</i>							
<i>T. Bijlsema et al</i>							
<i>Y. Fu et al</i>							
<i>B. Liu et al</i>							
<i>T. Stolte et al</i>							
<i>M. Schindewolf et al</i>							
<i>SoftDCar</i>							A2
<i>D. Niedballa et al</i>							
<i>S. Orlov et al</i>							
<i>S. Bak et al</i>							
<i>F.K. Bapp et al</i>							
<i>M. Li et al</i>							
<i>A. Schnellbach</i>							A3
<i>B. Sari</i>							
<i>F. Kempf et al</i>							

Table 4: Summary of the state of science

4 The On-Demand TMR System

4.1 Idea and Contribution

4.1.1 Function Design and Requirements

This dissertation will contribute to the identified gaps within research and science by proposing a novel concept for a save and highly available E/E architecture. This concept will be developed along a lightweight V-Shape process (see chapter 2.1).

To achieve this, the defined customer function (see chapter 2.1) will be summarized as “drive autonomously” and represents the input to the lightweight V-Shape model. A system must be found which realizes this customer function and at the same time contributes to the defined research questions (see chapter 1.4).

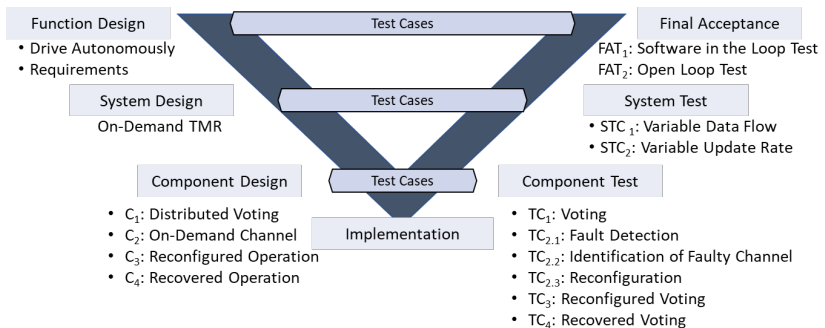


Figure 24: Lightweight V-Shape model for on-demand TMR system development

The requirements are clustered into the section's architecture requirements (contributing to RQ1 and RQ4), safety requirements (contributing to RQ1, RQ2 and RQ5) and AV specific requirements (contributing to RQ3 and RQ4).

Architecture Requirements: The purpose of the architecture requirements (AR) is to ensure, that the proposed system will combine the needs from an AV and also considers the limitations (such as embedded controllers).

AR1: The system must be designed in a pattern which fulfills the needs of future software intensive vehicles.

AR2: The system must use components which reflect automotive grade limitations.

AR3: The system must contain a minimal number of redundant components.

Safety Requirements: The purpose of the safety requirements is to ensure that the system can detect, isolate and recover faults (see Figure 12) to realize a fail-over process as required by ISO 26262 (see Figure 15). The realization of those requirements will be inspired by using elements from avionic architectures.

SR1: The system must be able to detect random hardware faults according to ISO 26262.

SR2: The system must operate in real-time for fault detection.

SR3: The system must isolate a detected fault.

SR4: The system must be able to continue to operate in the event of a fault.

SR5: The system must have an acceptable Fault Handling Time Interval (FHTI).

SR6: The system must be able to mask more than one consecutive random hardware fault.

AV Specific Requirements: The AV specific requirements will ensure, that the specific restrictions coming from the AV technology are considered.

AVR1: The system must be able to transport the amount of data which is required by an AD-System.

AVR2: The system must be able to process the data together with recent AD algorithms.

AVR3: The current structure of the autonomous driving software shall not be changed.

4.1.2 System Design Alternatives

The next step is to define a system architecture. To achieve this, the identified automotive and avionic architectures are analyzed to what extend they fulfill the above-mentioned requirements (see summary in Table 5).

The typical way in automotive for realizing a safe architecture is using a MCU, such as the Infineon Aurix. For fault tolerant systems this MCU is operated in lockstep with three cores to comply to the traditional TMR (see chapter 2.5.2). The TMR requires three individual channels and thereby is not minimizing the required number of redundant components, what violates the architecture requirement AR3 (minimal redundant components). Furthermore, the usage of MCUs is not possible for L4 systems since they suffer from low clock speed, low memory and no GPU, thereby violating the architecture requirement AR1 (needs of future software intensive vehicles) and the AV specific requirement AVR2 (data processing capabilities). Additionally, they typically use AUTOSAR classic which is not allowing an easy integration of POSIX (see chapter 2.2.2) based L4 applications, which violates the AV specific requirement AVR3 (no change of AV Software structure). Thus, the MCUs are not suitable for future vehicles, they are not able to process the algorithms as required by L4 applications and since they use the traditional TMR concept a

triplication of the compute resources is necessary to create a fault tolerant system.

Transferring the TMR to CPU cores is more promising since a CPU can host a POSIX operating system such as Linux. However, a hardware lockstep such as in MCUs for CPU is still part of the research and yet only works for non-resource intensive applications (see chapter 3.2.3) and is not suitable for L4 applications. This leads to an only partially fulfillment of the architecture requirement AR2 (automotive grade components) and AV specific requirement AVR2 (data processing capabilities). Furthermore, CPU lockstep also triples the number of resources required, thereby violating the architecture requirement AR3 (minimal number of redundant components) and it can only detect CPU random hardware faults, not faults which may occur on the GPU. Consequently, safety requirement SR1 (random fault detection) and architecture requirement AR1 (needs of future software intensive vehicles) are only partially fulfilled.

Another alternative is to monitor L4 applications using an external monitoring device such as an MCU (see work of Bijlsma et al. [82] in chapter 3.2.1.). In this system pattern a L4 application could run on a POSIX system in a service-oriented fashion, thereby meeting architecture requirement AR1 (needs of future software intensive vehicles). It could also run on automotive grade ECUs, thereby meeting architecture requirement AR2 (automotive grade components) with a minimal number of redundant components. This is the case since only one instance is running and the output is supervised externally using plausibility checks. Those plausibility checks may not detect random hardware faults on bit level, thereby only partially fulfilling safety requirement SR1 (random fault detection). Furthermore, since the MCU is not powerful enough to operate a full L4 application, a detected fault would bring the system into a degraded mode, thereby not masking the fault in a fail-operational manner what violates safety requirement SR4 (continue operation without degradation). A recovery process to bring the system for the fail-degraded operation back to a nominal operation could be developed but would

require a hand-over from the AUTOSAR classic based Safety MCU to the POSIX based SoC. Since L4 systems typically are realized on POSIX systems (see appendix 8.6.2) this synchronization mechanism would differ due to the required AUTOSAR classic part what only partially fulfills safety requirement SR6 (mask more than one fault) and AV specific requirement AVR3 (no change of AV Software structure).

Furthermore, options which modify avionic architectures to meet automotive requirements are conceivable. A Duo-Duplex, Quadruplex or Triplex-Triplex architecture could be realized on automotive ECUs by connecting them with automotive Ethernet. However, duo-duplex requires the redundant operation of an application on two cores and Triplex-Triplex even on three cores, thereby basically representing a CPU TMR on each ECU (see Figure 22). Same as for CPU TMR this is yet not existing on automotive grade, thereby only partially fulfilling architecture requirement AR2 (automotive grade components) for Duo-Duplex and Triplex Triplex.

Contrary Quadruplex operates the application redundantly on four ECUs and not on multiple cores within an ECU. Thus, the mechanism which ensures the lockstep on core level is not necessary for the Quadruplex architecture. The basic idea of distributing the software on multiple ECUs and checking the integrity over the network is conceivable to also work in combination with a SOA. However, again here, this has not been done yet, and accordingly architecture requirement AR2 (automotive grade components) is only partially fulfilled.

A downside of all avionic architectures is the mentioned high number of required redundant components, thereby violating architecture requirement AR3 (minimal number of redundant components). Due to its setup, Duo-Duplex cannot handle a consecutive fault, thereby violating safety requirement SR6 (mask more than one fault). Contrary, Quadruplex could manage at least two faults until it is fail unsafe, what only partially fulfills safety requirement SR6 (mask more than one fault). Triplex Triplex can at least manage five faults thereby having the highest availability of the avionic architectures.

	MCU TMR	CPU TMR	Monitoring (E-Gas)	Avionics: Duo-Duplex	Avionics: Triplex Triplex	Avionics: Quadruplex	On-Demand TMR
	Automotive			Avionics			
AR1: Future Vehicles							
AR2: Automotive Grade							
AR3: Minimal Components							
SR1: Fault Detection							
SR2: Real-Time							
SR3: Fault Isolation							
SR4: Fault Masking							
SR5: Fault Handling Time							
SR6: Multiple Faults							
AVR1: Data Transportation							
AVR2: Data Processing							
AVR3: AV Structure							

Table 5: Architecture realization alternatives

However, the management of more faults for Quadruplex and Triplex Triplex is possible, depending on if any recovery mechanism is implemented.

Another downside which comes along with the high number of redundant components of the avionic architectures is the increasing amount of traffic on the network technology. This is explicitly the case for Quadruplex where four and for Triplex Triplex where three ECUs require the same input. In both architectures the data must be transported to multiple ECUs thereby consuming costly bandwidth. Out of this reason Quadruplex and Triplex Triplex only partially fulfill AV specific requirement AVR1 (data transportation).

Lastly, the distributed voting mechanism for avionic architectures requires a synchronization between the redundant operating applications. In fact, the output of each application must be validated before its consumed by the next application (see Figure 20). With increasing number of redundant operations of the application (on redundant ECUs) this synchronization effort gets bigger. Thus, for two instances it is easier to realize than for three or four instances. Out of this reason, Quadruplex and Triplex only partially fulfill AV specific requirement AR3 (minimal number of redundant components).

4.1.3 The novel On-Demand TMR Concept

As explained in the previous chapters, it yet is not possible to directly achieve ASIL-D for L4 applications with traditional lockstep technologies. This fact is even intensified by missing ASIL-D GPUs, which are required for DL based L4 applications. The idea of this dissertation is to overcome this limitation by utilizing the ISO 26262 decomposition principle in combination with an adaption of the avionic Quadruplex architecture.

The core of the decomposition principle is to combine two ASIL-B elements to jointly achieve ASIL-D (see chapter 2.5.3). As mentioned, the SoCs are not available natively on ASIL-D, but they are available on ASIL-B. This applies for the CPU as well as for the GPU (see chapter 2.2.1) and represents the starting point of the following concept. The Quadruplex architecture showcases, that redundant operation of applications on multiple controllers is possible using distributed voters. Thus, those avionic controllers will be replaced by automotive ECUs. The ASIL decomposition principle will be realized by distributed voting, same as done in the Quadruplex architecture.

However, the proposed idea with Quadruplex as baseline comes along with a high number of redundant ECUs, thereby violating AR3 (minimal number of redundant components) (see previous chapter). This problem is intensified by the fact, that the AD Software Stack is not suiting on one single ECU (see Figure 2). The analysis of research vehicles showed that up to 10 computes

were necessary to deploy a whole AD software stack, not even considering redundancy (see appendix 8.6). If operated in the Quadruplex pattern, 40 computes would be necessary. If operated in TMR still 30 computes would be necessary.

This dissertation proposes a novel mechanism to overcome the challenge of the high number of necessary computes due to redundancy. It utilizes a new functionality coming from modern middleware technologies which allows to start, pause or shutdown applications during runtime. Thus, the idea is to run two instances of a L4 application in parallel on two ECUs rather than on four as in Quadruplex or three as in TMR (see L4 App and L4 App' on Figure 25).

This redundant operation of the L4 applications allows to detect random hardware faults, by comparing the output of both instances of a L4 application on each ECU (see L4 App and L4 App' on Figure 25). In the event of a fault, another instance of the exact same L4 application will be activated on a generic back-up ECU. This activation builds out the third channel as required for the identification of the faulty ECU. Thereby it follows the traditional TMR pattern with the difference, that the third channel is not permanently operated in hot standby but instead on-demand activated in case necessary.

The approach allows to map all L4 applications of the whole AD-Software stack on one single back-up ECU for redundancy purposes (see L4 App''... - L4 App''_m on Figure 25). This is possible since the L4 applications are not running but instead are started and loaded without being dispatched to the CPU or GPU. Out of this reason the introduced system is neither operating in hot standby, nor in cold standby but as hybrid in between.

To maximize the availability of the system the last element is a recovery mechanism. In the event an instance of the L4 application on either channel₁ or channel₂ is faulty, the channel will be recovered by dynamically restarting only the instance or the whole ECU. This will allow the system to manage multiple consecutive faults, thereby promising to be on a similar availability level as the introduced avionic architectures but with less redundant components.

In fact, the proposed on-demand TMR concept should be able to enhance the safety and availability of the mentioned AD system of 10 computes by utilizing 21 ECUs due to redundancy instead of 30 ECUs as required in the traditional TMR, thereby realizing AR3 by minimizing the required components.

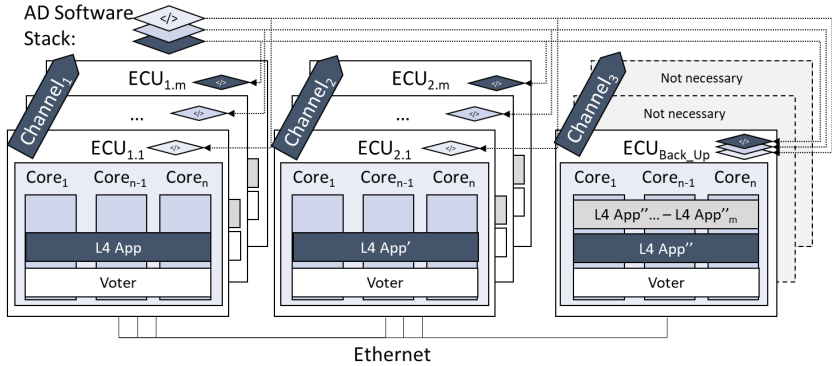


Figure 25: The on-demand TMR system

In summary, the concept contains the three major elements, namely the distributed voters, on-demand redundancy and auto recovery (see Figure 25).

Distributed Voters: As previously mentioned, avionic architectures are highly reliable and highly available. This is achieved by using voters on multiple channels connected with a networking technology (see chapter 3.3). Taking this concept as orientation, the idea is to realize SR1 (fault detection) by detecting ISO 26262 related random hardware faults by moving the voters from a local multicore lockstep processor to multiple SoCs on multiple ECUs as “Distributed Voters”. Beside the voters, also the L4 applications are distributed across multiple ECUs. For each L4 application multiple instances are launched for redundant operation (see L4 App and L4 App’ on Figure 25). Those instances use the voters to check whether their calculation results are correct after each iteration. As addition to the redundant operation of L4 applications, the distributed voting comparison mechanism is the core to achieve the ASIL decomposition principle as required in the ISO 26262 (see chapter 2.5.3).

On-demand Redundancy: The second novelty of this work is, that, contrary to the existing TMR, the third channel is not running continuously in parallel. Instead, in nominal mode the system runs in the 2oo2 pattern (see channel₁ and channel₂ on Figure 25). In fact, two instances of an L4 application run in parallel on two ECUs, using two distributed voters to validate their calculation results. Only in case those voters detect a fault the system is lifted to a 2oo3 level by calling channel₃. The purpose of this on-demand channel₃ is to figure out if the instance of channel₁ or the instance of channel₂ is faulty. A third instance of the L4 application is executed one more time on channel₃ what creates the required third output. This third output is then used to identify which channel is correct or faulty so that the third voter can shut down the faulty instance and instead reconfigures the instance on ECU_{back_up} (see L4 App'' on Figure 25) together with the correctly working channel.

Accordingly, on channel₃ each L4 application of the whole AD software stack (see Figure 19) is loaded as instance and waits for the need to be dispatched to the CPU or GPU. Depending on which L4 application is faulty, the respective instance can be on-demand activated, thereby realizing SR4 by keeping the system operational with minimal components.

Auto-Recovery: The last element of the concept is the recovery mechanism of the faulty channel. While the system continues to operate by utilizing the correct channel in combination with the on-demand channel₃, the faulty channel will be recovered. This happens by restarting the faulty instance of the L4 application and executing a phase over with channel₃ once the restart is completed. This brings the system back into the nominal mode, allowing to mask following consecutive faults with a minimal number of redundant components. Compared to existing solutions the major benefit is, that the proposed architecture is promising to manage an unlimited number of faults, if they occur during nominal phase and thereby realizes SR6 (multiple faults).

4.2 Architecture and Components

The on-demand TMR system requires an interaction between the three described elements (see previous chapter). While it is in nominal operation, it must be able to detect faults. If a fault is detected, it transitions into the on-demand TMR state, which executes the detection of the faulty channel, initiates the reconfiguration, shuts down the faulty channel and initiate its recovery. Until the faulty channel is recovered, the system acts in the reconfigured mode (see Figure 26).

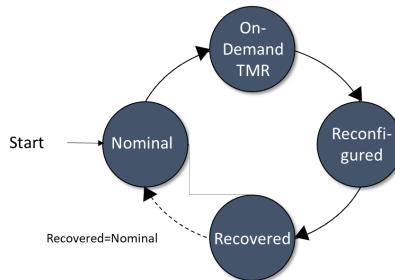


Figure 26: On-demand TMR as State Machine

The recovered state and the nominal state contain the exact same software deployment, so that the system can mask potential consecutive faults. However, from a result analysis perspective both states are distinguished in the following chapters.

For simplification purposes the elements of the on-demand TMR will be explained for one L4 application. However, the fully populated on-demand TRM system should be able to run all required L4 applications of the AD Software stack in parallel following the same procedure.

4.2.1 Distributed Voters

One of the core elements of the concept are the three voters as counterparts to the three redundant instances of the L4 applications. The main purpose of the voters is to detect random hardware faults as required by a fault tolerant system (see Figure 12).

As mentioned, the system operates two instances of a L4 application redundantly in hot stand-by, namely L4 App and L4 App', to detect random hardware faults (see Figure 27). Additionally, it loads but pauses another instance of the L4 application on the ECU_{back_up} (see L4 App'' on Figure 27). Each instance of the L4 application is connected to an instance of the distributed voters. The access point for L4 App and L4 App' to the voters is a well-defined interface which can receive data to be verified (see dashed green arrow from L4 App to V_1 and L4 App' to V_2 on Figure 27). This interface must fulfill two characteristics. First, after it was called by the respective instance of the L4 application, it must wait for the result of the voting process before it publishes its result to the following L4 application (indicated as blue arrow from L4 App to L4 App_n on Figure 27). Second, the distributed voters must provide a return value to L4 App and L4 App' if the calculation result is correct (see dashed green arrow from V_1 to L4 App and V_2 to L4 App₂ on Figure 27).

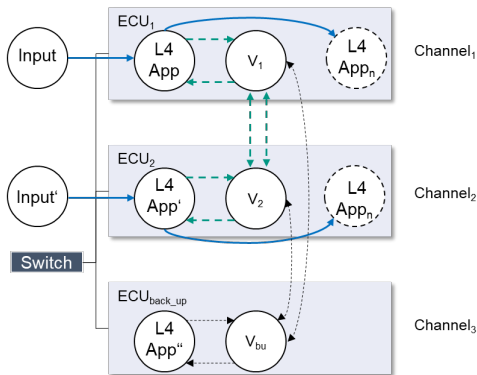


Figure 27: Distributed voting

Once a voter received the input data from L4 App or L4 App' to be validated it must check whether the other L4 App created the same input data. To keep the traffic between both voters minimal, the input data from the L4 Apps will be compressed by using hashing (see chapter 2.6.2). Accordingly, both voters cross-share the compressed input data of L4 App and L4 App' as a hash.

Since the L4 App, L4 App' and the voters are individual processes with own schedulers on each ECU (see chapter 2.2.2), it cannot be estimated, which hash arrives at which voter first. In fact, it can be the case that in one calculation iteration L4 App finishes before L4 App'. Vice versa in the next calculation iteration L4 App' finishes before L4 App. To improve this, a synchronization mechanism for those parallel tasks on the distributed ECUs is necessary which holds on the voting of the first hash receiving voter until the hash of the other voter is also available.

To ensure the integrity of the system, not only the L4 applications have to be decomposed according to ISO 26262 but also the voters (see chapter 2.5.3). It is not sufficient if only one voter will compare the input data from L4 App and L4 App'. This problem is solved by a redundant comparison mechanism, where each voter first compares both compressed input data and second returns the result to the other voter. Thus, after each iteration each voter has two information available: First its own comparison result and second the result of the voting procedure of the other voter in form of true or false. Before confirming towards L4 App or L4 App', the voter compares both, its own two hashes and the result of this comparison with the result of the other voter. Only if both are true, the voter will confirm towards L4 App and L4 App' that it can publish its result (see upper part of Figure 28 outside if-loop).

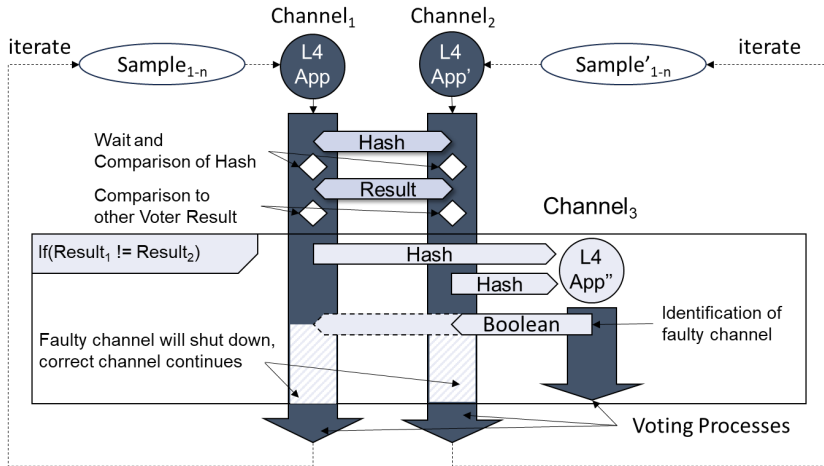


Figure 28: On-Demand TMR Process View

The comparison mechanism is cyclically repeated, depending on the update rate of the L4 applications and represents the nominal state of the system. Furthermore, the voting procedure must have happened within an acceptable duration to comply to the real-time requirement (SR2).

4.2.2 On-Demand Channel

A random hardware fault would lead to a change in the bitstream of the result of L4 App or L4 App' what ends up in deviating hashes as input to each voter (see chapter 2.6.2) (see red lighting in Figure 29 between V_1 and V_2). In case such a mismatch is detected channel₃ is activated to on-demand lift the system from a 2oo2 pattern to a 2oo3 pattern, thereby bringing it from a pure fault detection to a fail-operational level (see chapter 2.5.2). Both voters V_1 and V_2 send each hash to V_{bu} which then will create a third hash to identify if channel₁ or channel₂ is faulty.

To create the third hash V_{bu} activates L4 App'', which then will do the exact same calculation iteration as L4 App and L4 App' did one more time. This

process requires the exact same input to L4 App'' as it was given to L4 App and L4 App' for the initial calculation iteration.

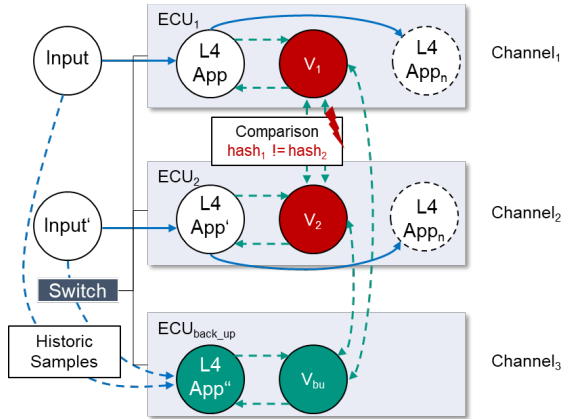


Figure 29: On-demand TMR activation

This brings an explicit challenge to the system setup since the required data input was already published and consumed by L4 App and L4 App', thereby being historic. Therefore, a mechanism which enables L4 App'' to pull those historic samples from the inputs is required, marked as dashed blue arrow between Input and L4 App as well as Input' and L4 App'. L4 App'' then will loop through the historic samples until it has reached the one which was initially processed. It then will be processed one more time by L4 App'' to create the required third hash for V_{bu}.

The third hash represents the input for V_{bu} to identify if channel₁ or channel₂ is right or wrong by comparing it with the hash from V₁ and from V₂. Once the faulty channel was identified, V_{bu} will initiate the reconfiguration process which is characterized by two steps.

The first step is to isolate the fault (see Figure 12) by shutting down the faulty instance of the L4 application to avoid fault propagation through the system (see chapter 2.5.3 and yellow L4 App on Figure 30). Since the instance of the L4 application itself is faulty, it must be shut down by an external mechanism

because it eventually cannot shut itself down anymore. This avoids the fault propagation through the system, thereby realizing SR3 and utilizes the corresponding voter, which is either V_1 or V_2 .

The second step is to mask the fault (see Figure 12) by activating the third instance of the L4 Application as L4 App'' on the ECU_{back_up} to get the system back into a 2oo2 operation pattern. It will on-demand subscribe to the exact same input data, and it will create the exact same output data as previously L4 App did.

The duration from the detection of a fault until the L4 application is reconfigured is the Fault Handling Time Interval (FHTI) as described in ISO 26262 (see Figure 15). Same as with the real-time, also the faulty channel detection and the reconfiguration must have happened within an acceptable duration to comply to the FHTI requirement (SR5).

4.2.3 Reconfigured Operation

While the system continues in reconfigured operation, the faulty channel is recovered. Such a recovery mechanism can be the restart of the L4 App itself or even the whole ECU, depending on the severity of the occurred fault. The corresponding voter (see V_1 on Figure 30) will initiate the recovery processes while the faulty L4 application is reconfigured and running on ECU_{back_up}.

The mentioned approach is realized by L4 App'' subscribing to the exact same input as L4 App previously did, depicted as solid blue arrow from Input to L4 App'' (see Figure 30). It also creates the same output as initially L4 App did, indicated with the solid blue arrow from L4 App'' to L4 App_n. Furthermore, it also will use the same voter as L4 App did with one difference:

While initially L4 App and V_1 were running on the same machine, they now run on different machines. The hash from L4 App'' is now send to V_1 via a networking technology rather than shared on the same SoC between the

process of the instance L4 App and the processes of the voter using shared memory.

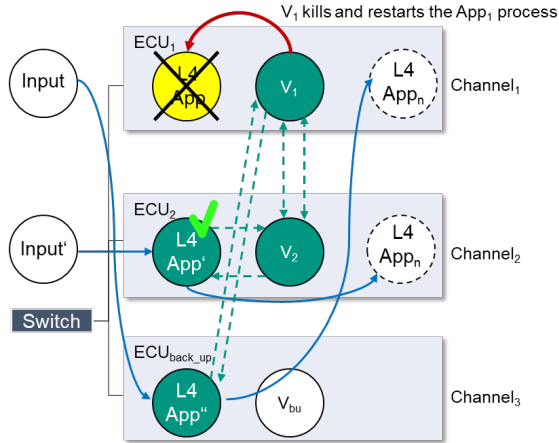


Figure 30: Reconfiguration of the system

During the reconfigured state the system is not capable to mask another consecutive fault since all ECUs including the back-up ECU are in use. Since the reconfigured state is active for only a short duration of a couple of seconds, the probability of a consecutive fault during this duration is fairly low (see fault probability according to ASIL levels in chapter 2.5.3). However, if another fault happens the vehicle still could react with a stop in lane maneuver as ratio ultimo. To do so it could utilize the safety MCU which is still operational in reconfigured state due to its ASIL-D classification.

To keep the likelihood of another consecutive fault during reconfigured operation low, the duration in the reconfigured state of the system must be minimal. ECU₁ must be recovered quickly so that the ECU_{back_up} can go back to its initial on-demand state again.

4.2.4 Recovery Mechanism

Once the faulty L4 application was shut down, the voter at the same time will start a recovery mechanism such as either the restart of the whole ECU or only the restart of the faulty instance of the L4 application.

Thus, while the recovery is happening for L4 App the system still is in the re-configured state as described in the previous chapter. Once the recovery mechanism is done, the original channel will take over again.

For this purpose a carefully designed synchronization mechanism, which notifies L4 App'' that L4 App is now ready to take over again, must be designed. Upon this notification, L4 App'' stops operating and transmits to L4 App the last processed iteration. The recovered L4 App then continues to operate from there (see solid black arrow in Figure 31).

It also indicates a slight difference between L4 App and L4 App''. While L4 App'' needs to provide a service which allows L4 App to synchronize back into operation including its shut down, this service is not necessary for L4 App. Instead L4 App gets shut down externally by the corresponding voter in the event of a fault, regardless the state in which it is.

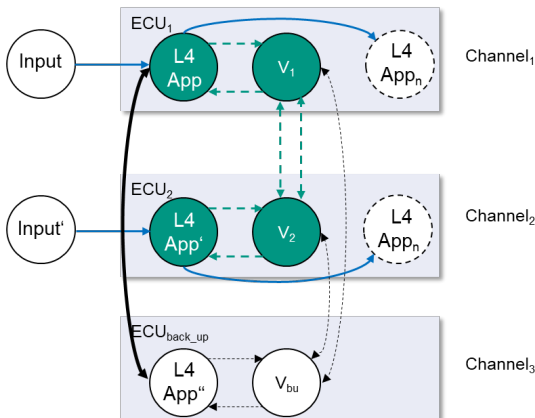


Figure 31: Recovered operation of the on-demand TMR system

Starting from there, L4 App on ECU₁ will take over the processing again while L4 App'' will stop working. This releases the resources on ECU_{back_up} for an eventually happening consecutive fault, thereby indicating that, contrary to all identified existing solutions in automotive and avionics, the system would be able to manage an unlimited number of faults if they are not occurring during the reconfigured state.

4.3 Implementation Alternatives

4.3.1 SOA Technologies

Today there are two architecture styles for automotive vehicles existing, namely the signal-oriented and the SOA (see chapter 2.4.1). The research of the state of the art showed, that the future architecture style is tending towards the service-orientation with its core of Ethernet as networking technology. The second core element of a SOA is the middleware for which a variety of implantation technologies exist (see list in 3.1.3). However, only two of them are automotive certified and could be used to comply with AR1 (future vehicles) and AR2 (automotive grade). This is namely Adaptive AUTOSAR, with SOME/IP as core, and ROS2 with DDS as core.

Both middleware technologies can start, pause or shutdown L4 applications during runtime. This function is of special importance since the on-demand channel₃ needs to be activated during runtime. Furthermore, both technologies are Ethernet based with TCP or UDP as underlying protocol (see chapter 2.3.2) allowing the transportation of a large amount of data as required for modern sensors for autonomous driving (see chapter 3.1.1). Additionally, it has been shown that both technologies can be used to build up automotive SOAs, even though ROS2 still suffers from limitations regarding the available certified features compared to AUTOSAR [94].

The main difference between both technologies resides within two aspects. First, in contrast to Adaptive AUTOSAR, ROS2 is open source and can be accessed without any restrictions. The second aspect, which is even more important is, that ROS2 got more and more popular for robotic systems and is now a widely used solution for autonomous driving. During the last years, it developed towards the de-facto standard for L4 applications [58], thereby fulfilling AVR3. Thus, ROS2 also will be used for this dissertation (see summary in Table 6).

	ROS2	Adaptive AUTOSAR
Automotive Certified	+	++
Ethernet based	+	+
Start/Shutdown during runtime	+	+
Standard for Autonomous Driving	++	+
Accessibility	+	-
		

Table 6: Comparison ROS2 vs. Adaptive AUTOSAR

As result, the proposed system will be realized with ROS2 as SOA, namely the Galactic release using FastRTPS as middleware. Each L4 application as well as the voters are designed as ROS2 nodes. The communication between the L4 application nodes are designed as ROS2 topics (see solid arrows arrow in Figure 27, Figure 29, Figure 30 and Figure 31). This SOA based approach realizes AR1 (future vehicles) and AR2 (automotive grade) and utilizes Ethernet which is powerful enough to realize AVR1 (data transportation) by transporting the amount of data needed for AVs.

Contrary, the interfaces to the voters and also the data sharing between the voters (see dashed green arrow in Figure 27, Figure 29, Figure 30 and Figure 31) are designed as ROS2 service since in contrast to a ROS2 topic, a ROS2 service provides a return value. In the case of the voter this value is either true or false depending on the voting result. However, the cross sharing of data between the voters using ROS2 services produces traffic on the Ethernet which bandwidth is limited.

One option to overcome this problem is hashing which reduces a large amount of input data to a small unique hash value (see chapter 2.6.2). Even though the MD5 hashing algorithm suffers from security concerns it still is used to check unintentional corruption of data. The advantage of MD5 over other alternatives (e.g. the successor SHA) is the faster computation which is crucial within resource constrained automotive systems. Accordingly, to minimize the traffic on the Ethernet, the results of the L4 applications after each iteration are hashed into MD5, thereby representing the input to the voters (see summary in Table 7).

	Bitwise	MD5	SHA256
Compression	-	++	+
Data Integrity	++	+	++
Compression Speed	-	++	+
			

Table 7: Data integrity checking technologies

4.3.2 Reconfiguration Technologies

For the temporary reconfiguration of channel₃ as reaction on a fault, as well as for the recovery of the faulty channel itself, existing reconfiguration technologies could be used. They detect if a process is down and automatically initiate a restart mechanism. These are namely QoS settings of ROS2, the ROS2 Managed Nodes concept, the equivalent for SOME/IP and orchestration of containers or virtual machines.

A closer look on the QoS setting for ROS2, namely the OWNERSHIP_STRENGTH, shows, that this mechanism runs the publishing process on two devices in parallel. Thus, technically this mechanism is no option to detect if a process was shut down. It rather detects whether messages from a publisher arrive at a subscriber. If not, it swaps to a back-up publisher with the lower OWNERSHIP_STRENGTH. Accordingly, this mechanism is a hot standby pattern which is not made to restart a process but to swap to a running back-up process.

The other option is the ROS2 managed node concept which can detect if a node is down to automatically restart it. Accordingly, as part of this dissertation, a closely supervised master thesis was executed to investigate the re-configuration capabilities of the ROS2 Managed Nodes concept for safety critical L4 applications. A Managed Node was implemented into the open source Autoware.Auto stack (see chapter 3.1.2) and executed on a remote machine connected with 1Gb/s Ethernet. This node was arbitrarily shut down to see how much time is required to restart it. As result, the average reconfiguration time was 1.6 seconds [95].

Even though the ROS2 Managed Nodes concept looks promising regarding the recovery time of a faulty process, it still suffers from the limitation that the restart of the node also is triggered by a ROS2 process. If a fault in an L4 application happens the whole ROS2 process must be shut down to avoid fault propagation. However, following the ROS2 managed node concept there still must be a ROS2 instance running on the affected machine to restart a node.

The orchestration of containers to recover processes were used by Liu et al and Stoll [29, 88]. In contrast to QoS and ROS2 managed nodes, the idea of container orchestration is to not recover a single process but a whole container (see chapter 2.6.3). The advantage compared to the ROS2 Managed Nodes concept is that containers provide a higher isolation (see chapter 2.5.3) to avoid the propagation of a fault to other elements of the system. The research of Liu et al. [88] showed, that in the best case the recover mechanism of orchestrated containers took 16,8 seconds. This is roughly 8 times longer than the ROS2 managed nodes concept and is no adequate solution for this dissertation.

Thus, a new approach will be implemented were the voters (V_1 and V_2) will be extended by a service, which shuts down and restarts the process of the faulty L4 application using Linux system calls. This approach requires only little resources since it uses already existing Linux system commands. It has an

acceptable level of isolation since it is outside of the process of the affected L4 application and has a high recovery speed (see summary in Table 8.)

	ROS2 Ownership Strength	Container with Orchestrator	ROS2 Managed Nodes	Restart through ser- vice
Resource Demand	--	-	+	+
Recovery Speed	++	--	+	++
Isolation	-	+	-	+
				

Table 8: Comparison of reconfiguration technologies

4.4 Test Cases and Metric

4.4.1 Component Test Cases

The lightweight V-Shape development process which this dissertation follows (see chapter 2.1 and chapter 4.1.1) requires that the defined customer function, requirements, the derived architecture and the implemented components must be tested for verification and validation.

To comply to this process, the requirements have been the input to create the on-demand TRM architecture with its four components (see chapter 4.2). To allow a conclusive statement about the usability of the on-demand TMR system (see chapter 7.1), the compliance towards the requirements must be demonstrated. This is achieved by defining six component test cases (see Figure 24), which are allocated to the components.

Additionally, to each component test, in the next step the interaction between all of them must be showcased. This is realized by two system test cases. The last element required for a conclusive statement is the validation of the customer expectation, namely the customer function (see chapter 4.1.1) (see Figure 32).

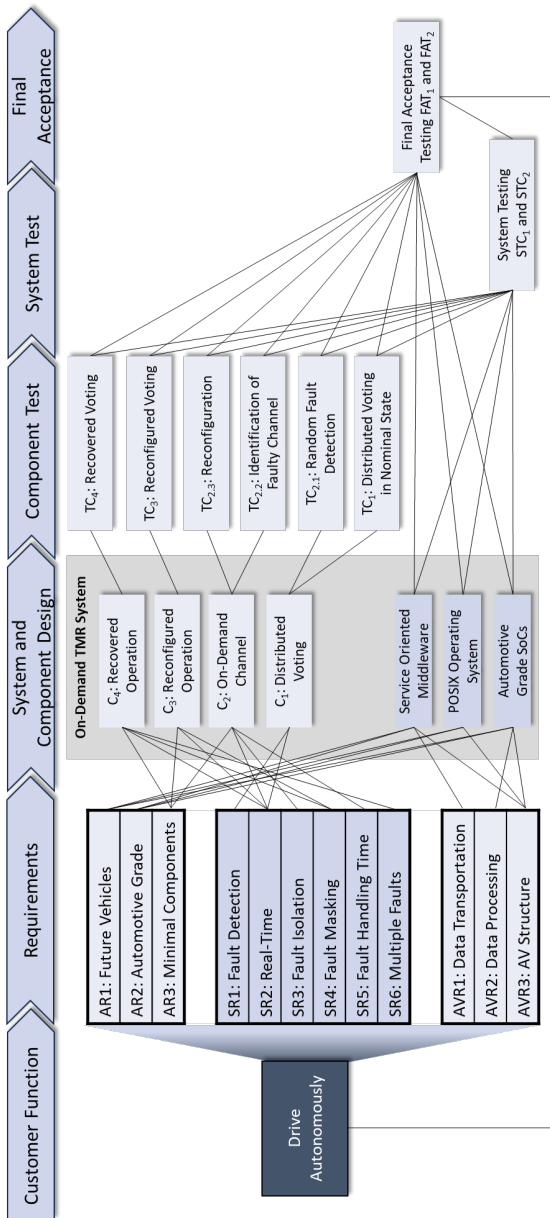


Figure 32: Structured approach for test case generation

TC₁ Distributed Voting in Nominal State: The purpose of this test case is to analyze if and how efficient the distributed voting mechanism works.

Metric: To judge on the efficiency of the distributed voting, the latency (see t_{add_ch1} and t_{add_ch2} in Figure 33), will be analyzed for channel₁ and channel₂. The purpose is to see, how much overhead in terms of additional necessary processing time the voters add to the calculation duration of the instances of the L4 application in each iteration (A_1 , A'_1). In fact, t_{add_ch1} and t_{add_ch2} covers:

- The duration required to send the hash of L4 App' from V_2 to V_1 (1).
- The waiting duration of V_1 for its own hash (2).
- The duration required to send the hash of L4 App from V_1 to V_2 (3).
- The comparison of both hashes by V_2 (4).
- The comparison of both hashes and returning of the comparison result to V_2 by V_1 (5).
- The comparison of both hashes and returning of the comparison result to V_1 by V_2 (6).
- The confirmation of V_1 to L4 App (7) and the confirmation of V_2 to L4 App' (8) to publish their calculation results (P) for the corresponding calculation iteration (A_1 or A'_1).

Each described calculation and voting iteration is executed in combination with a corresponding sample as input (S_1 or S'_1).

For simplification purposes, this voting process, summarized as t_{add_ch1} or t_{add_ch2} depending on if the overhead for L4 App or L4 App' will be analyzed, will be summarized in one box (V) in each following figure. It represents the voting duration which contains the voting time (t_{voting}) and network time ($t_{network}$). While the network time is the duration required to transport the hash from the instances of the L4 application to

the voters, the voting time is the duration which is needed for the comparison of the hashes only.

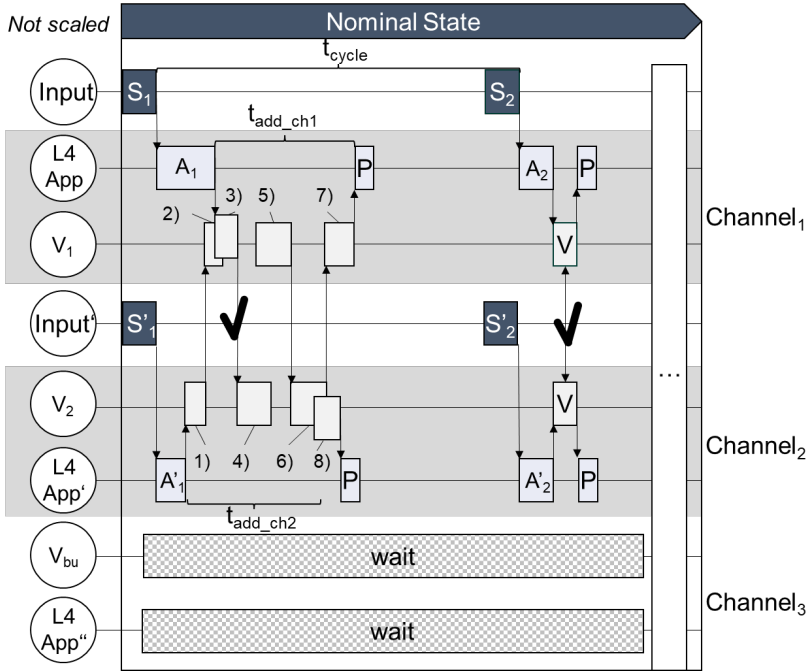


Figure 33: Process diagram of nominal state

TC_{2.1} Random Fault Detection: Within this case it will be tested if the fault detection mechanism of the distributed voters works. The hash of L4 App as input to V₁ will arbitrarily be changed, thereby simulating a random hardware fault (see iteration A₅₁ in Figure 34). This will lead to a mismatch in the voting processes (see lightning in Figure 34).

Metric: The metric of this test case is binary. It will be tested if the system will detect the fault or not.

TC_{2.2} Identification of Faulty Channel: After a detected fault, the faulty channel must be identified so that it can be shut down to avoid further propagation (see black circle in the lane of L4 App on Figure 34). This test case will showcase, that after a detected fault the on-demand TMR system is able to identify the faulty channel.

Metric: The key performance metric of this TC is to measure the time which is required (t_{add_51}) from the moment a faulty hash was transmitted to one of the voters until the faulty channel was identified by V_{bu} . This duration is defined as faulty channel detection duration. It is further broken down into each required step the system executes during the faulty channel detection process which is namely:

- t_{v_bu1} : The duration until V_{bu} activates L4 App'' including the duration V_{bu} needs to respond to V_1 and V_2 .
- $t_{context}$: The duration required until L4 App'' is dispatched to the CPU.
- t_{call_back} : The duration required until the first historic sample was received by L4 App''.
- t_{loop} : The duration to loop through the historic samples.
- t_{calc} : The duration to calculate the third result.
- t_{hash} : The duration required to create the third hash.
- t_{third} : The duration to compare $hash_1$ and $hash_2$ with $hash_3$ to identify the faulty channel.

V_{bu} will then reply to V_1 to kill the faulty L4 App (K) and confirms (C) to V_2 that this channel is correct and can be published (P).

TC_{2.3} Reconfiguration: Based on the identified wrong channel V_{bu} will then start the faulty instance of the L4 application on ECU_{back_up} so that the system stays operational (see A''_{52} in Figure 34). Thus, the purpose of this test case is to show, that the faulty instance of the L4 application can be temporarily reconfigured within an acceptable fault handling time interval (see Figure 15).

Metric: The reconfiguration of the system happens in three steps (see Figure 34) which will be in sum defined as reconfiguration duration:

- t_{activate} : The duration to activate L4 App'' as recovery for the faulty instance of L4 App.
- $t_{\text{loop_start}}$: Again here, L4 App'' will loop through the historic samples. It will loop for one more iteration as within $\text{TC}_{2.2}$, namely until the sample after the faulty sample (see S_{52} on Figure 34). The reason is, that the faulty iteration of $\text{TC}_{2.2}$ detected the faulty channel and $\text{TC}_{2.3}$ reconfigures for each iteration after the faulty iteration.
- $t_{\text{calc_r}}$: The duration which is required to calculate the first result within reconfigured operation.

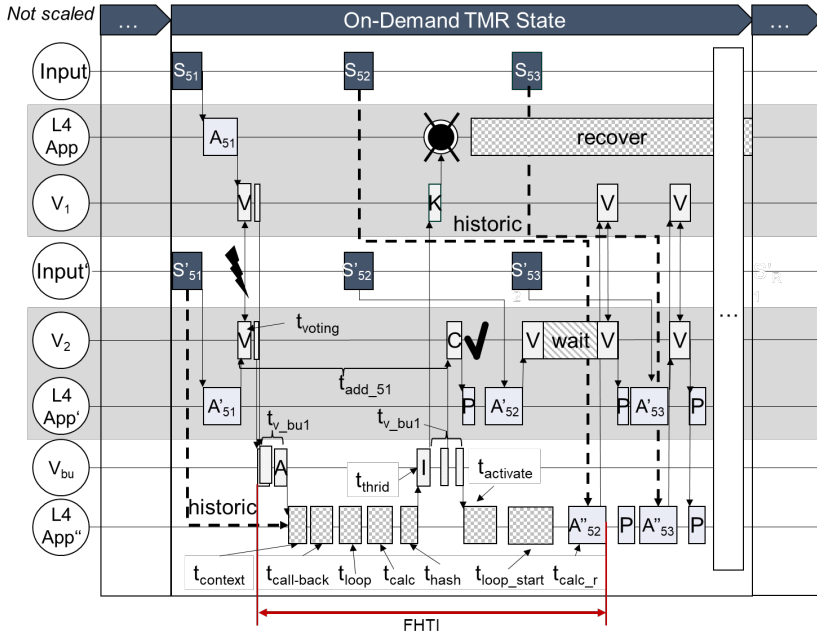


Figure 34: Process diagram on on-demand TMR state

The faulty channel detection duration and the reconfiguration duration together form the FHTI as required in the ISO26262 (see Figure 15).

TC₃ Reconfigured Voting: To stay operational after the faulty iteration the system must continue to execute the fault detection.

Metric: Same as in TC₁, again here, the additional time the voters require is in the center of interest. It will be analyzed how much additional calculation time the on-demand TMR adds while the system is in reconfigured state (see t_{add_ch1} and t_{add_ch2} on Figure 35).

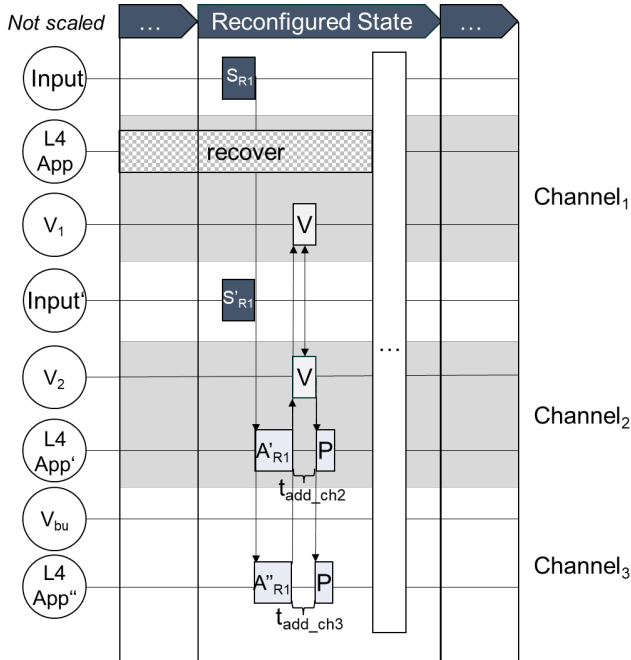


Figure 35: Process diagram on reconfigured state

TC₄: Recovered Voting: The temporarily occupied resources of ECU_{back_up} must be released again for eventually happening consecutive faults. To achieve this, L4 App must be restarted so that L4 App'' can be shut down. The restart process (see recover bar in Figure 35) needs to be

synchronized (S) into the program flow again by asking L4 App'' for its last processed sample (1), which then replies the last processed sample to L4 App (2), finishes its last iteration and then stops operating (see circle on Figure 36).

Metric: This test case will deliver two results. The first metric is binary in a sense that it shows that the phase in of L4 App is possible. The second metric will analyze again, how the voters behave in the recovered mode. Again here, it will be analyzed how much overhead t_{add_ch1} and t_{add_ch2} they will create in this system state.

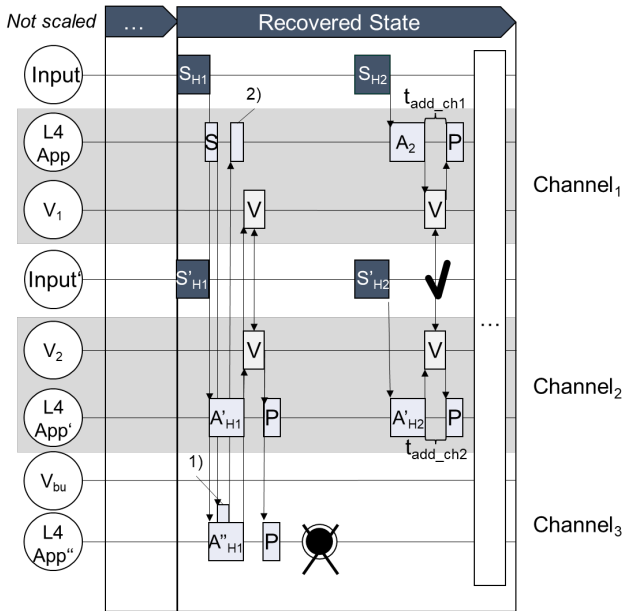


Figure 36: Process diagram on recovered state

4.4.2 System Test Cases and Final Acceptance Testing

As next step, the interaction between the designed components will be tested. For this purpose, system test cases (STC) are defined, which analyze the behavior of the TCs in context.

The major input for DL based AD-Algorithms comes from the camera and the lidar (see chapter 2.6.1, chapter 3.1.1), forming the boundaries of the STCs with their update and data rates. While the update rates of a lidar varies between 5Hz to 50Hz, creating between 10MB/s to 70MB/s of raw data, a camera typically has an update rate of between 25Hz to 50Hz and creates between 10MB/s to 180MB/s of raw data (see Table 3).

Since not only the sensing module of the AD software stack but also other modules such as behavior, planning or control may be uplifted to a higher safety level using the on-demand TMR, also smaller data rates as 10MB/s will be considered (see Figure 20).

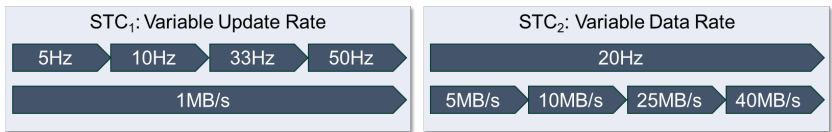


Figure 37: System test case overview

The application for STC₁ and STC₂ is the calculation of the number Pi for the precision of one million digits. This calculation is repeated in each iteration. To avoid that the result is always the same, the number Pi in each iteration is divided by the sample number leading to a unique result in each iteration.

STC₁ Variable Update Rate: STC₁ starts with a low update rate of 5Hz and is stepwise increased to 50Hz as maximum. The data rate will stay constant at 1MB/s what means that the sample size of each sample will decrease with increasing update rate. During this test all system states are executed, thereby embedding the component test cases (TC₁₋₄) into the system testing.

Metric: The main metric is binary in a sense that 1000 iterations have been processed for each update rate (see Figure 28). The system is either able to process them or not.

STC₂ Variable data Rate: Contrary to STC₁ for STC₂ the update rate will stay constant at 20Hz. The data rate will start with 5MB/s and is stepwise increased to 40MB/s. Again here, STC₂ will contain TC₁₋₄ as described in the previous chapter.

Metric: Same as with STC₁ the metric is binary in a sense that 1000 samples have been processed for each data rate. The system is either able to process them or not.

The minimal V-Shape model (see Figure 24) requires a final acceptance testing (FAT) of the system for which a real L4 application is selected to realize the on-demand TRM system.

FAT₁ Robust Lane Detection (RoLand) as Software in the Loop on Test Bench:

While the STCs indicate the general functionality of the on-demand TMR system, the target of FAT₁ is to show that it also can be used for real L4 applications. To achieve this, a robust lane detection algorithm is selected which has a CNN as basis (see chapter 2.6.1, chapter 3.1.4). The input to RoLand is a raw image, the output is a state vector which contains the position of the center of the vehicle relative to the left and to the right lane markings.

Metric: Same as with STC₁ and STC₂ the metric is binary in a sense that 1000 samples have been processed for each data rate. The system is either able to process them or not.

FAT₂ Robust Lane Detection (RoLand) as Real World in Class 8 Long Haul Truck:

While the FAT₁ testing answers the usability of the on-demand TMR system for a L4 application on a test bench, the target of FAT₂ is to demonstrate that the proposed system also works in a real on road test.

Metric: Same as with FAT_1 the metric is binary in a sense that 1000 samples have been processed for each data rate. The system is either able to process them or not.

4.4.3 Acceptance Criteria

The goal of the TCs is to give insight into the behavior of the system in each state. However, the acceptance criteria of each TC deviates, if tested in the context of STC or FAT.

STC₁ and STC₂:

The purpose of the STCs is to showcase the general functionality of the on-demand TMR system, meaning the integration and interaction of all TCs with each other. The focus of the STCs is rather the general functionality of the system, than the performance. In fact, the STCs must demonstrate that the architecture requirements (AR1 to AR3), the AV Specific Requirements (AVR1 to AVR3) and the Safety Requirements (SR1, SR3, SR4 and SR6) are fulfilled (see chapter 4.1.1).

The following acceptance criteria will be defined for each TC:

TC₁: *“The distributed voting mechanism must validate all correct iterations during the nominal state by using the voters.”*

TC_{2.1}: *“The voters must identify two different hashes and initiate the identification process of the faulty channel by activating V_{bu} ”.*

TC_{2.2}: *“ V_{bu} must identify the faulty channel. The faulty channel must stop working while the correct channel continues.”*

TC_{2.3}: *“The system must continue to operate with the correct channel together with the reconfigured channel₃.”*

TC₃: *“The system must continue in the reconfigured deployment. During this duration, the distributed voting mechanism must validate all correct iterations.”*

TC₄: *“The system must restart the faulty application. After the restart, the temporarily running application from the reconfigured state must hand over to the restarted application. After this hand over the system must continue to validate all correct iterations.”*

STC Integration: *“1000 samples have been processed by the system iteratively. The transitions between the system states nominal, on-demand TMR, reconfigured and recovered must have happened without error.”*

FAT₁ and FAT₂:

The purpose of the FAT is to analyze the usability of the proposed on-demand TMR system for L4 applications. Thus, only the fact that the components of the system work together is not sufficient. The FAT testing rather gives an insight if the performance of the on-demand TMR is sufficient to be usable for L4 Applications. Thus, the target of the FATs is to give insight into the level of achievement of the requirements SR2: Real-Time and SR5 Fault handling time (see chapter 4.1.1).

The following acceptance criteria will be defined for each TC:

TC₁: *“The average voting duration t_{add} should not exceed an acceptable overhead. The WCET execution time should be acceptable. If t_{add} or the WCET is not acceptable, countermeasures must be identified.”*

The terminology *acceptable* will be further elaborated by comparing the TC results to references (see chapter 6).

For a conclusion about performance, TC_{2.1-2.3} can be summarized in the context of the FATs.

TC_{2.1-2.3}: *“The FHTI should be acceptable long. If it is not, countermeasures must be defined.”*

Again, since the focus of TC₁, TC₃ and TC₄ is to analyze the overhead and WCET, also the acceptance criteria of TC₃ and TC₄ is the same as TC₁.

FAT Integration: *“1000 samples have been processed by the system iteratively. The transitions between the system states nominal, on-demand TMR, reconfigured and recovered must have happened without error.”*

5 Prototypical Realization

5.1 Overview

The on-demand TRM system is realized in a three-step approach. In the first step the general behavior shall be understood. Therefore, a Hardware in the Loop (HiL) test bench is realized in a minimal set-up, which is the connection of three automotive ECUs and three industrial computers using Ethernet. Also, the application which calculates the number P_i requires only a CPU and contains only little program data.

The purpose of the second step is, to get insight into the behavior of the on-demand TMR system using a real L4 robust lane detection (RoLaND) application. While the HiL test bench remains the exact same, RoLaND requires more resources by utilizing the GPU for inference of a neural network. Additionally, it contains a neural network as program data.

Finally, in the third step RoLaND remains the same, but the system is scaled by applying it into a real heavy-duty truck and adding two cameras. The purpose is to understand how the on-demand TMR system behaves if the target system is as close as possible to the reality.

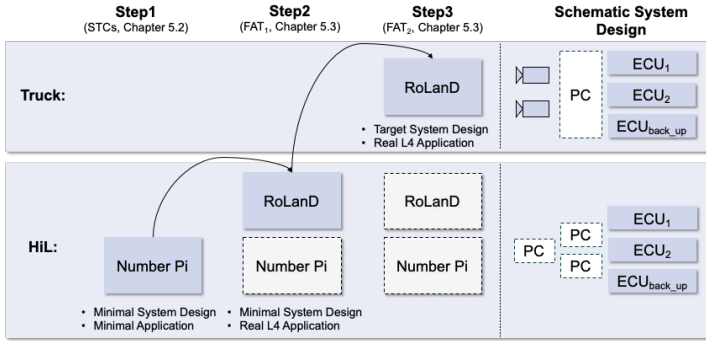


Figure 38: Stepwise System Realization

5.2 System Testing

5.2.1 System Set-Up

Both STCs start with a source node which is deployed on a personal computer (PC₁). It cyclically publishes samples to the Input node on PC₂ and the Input' node on PC₃. This sample only is the trigger for the input nodes to instantly publish their own samples to the L4 application (which calculates the number Pi in this testing). Together with each voter, the primary instance of the application (Pi) is located on ECU₁, the first redundant instance on ECU₂ (Pi') and the on-demand instance on ECU₃ (Pi'') (see Figure 39).

The reason for creating the redundant Input nodes is the ISO 26262 requirement of fully independent channels. While the source node still represents a single point of failure, the rest of the logical dataflow is fully redundant, thereby representing the sphere of replication. The source node will be eliminated in FAT₂ testing in which the Inputs are replaced by cameras. Thus, the chosen set-up of redundant input nodes is closer to a production ready target system as it would be if the source node is directly connected to the L4 applications.

The samples which are sent from the inputs to P_i and P_i' contain a fixed size array with 64bit unsigned integers and another separate additional unsigned integer. While the array represents the payload, creating the desired traffic on the Ethernet, the unsigned integer acts as counter which is incremented with each iteration. It thereby represents the sample number. Thus, STC_1 is controlled over the number of messages per second of the trigger node and STC_2 is controlled by changing the size of the 64bit unsigned integer array.

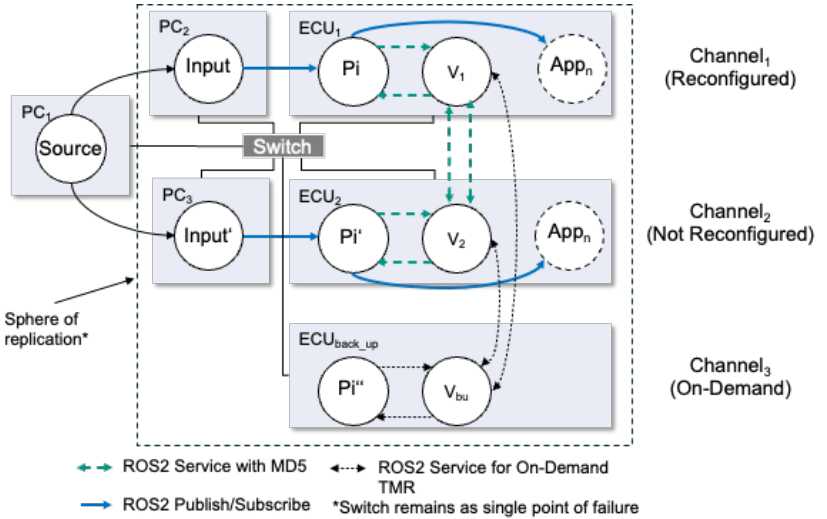


Figure 39: Deployment schematic for system testing

As described for both STCs, P_i and P_i' will calculate the number P_i and hash the result into an MD5 string (see chapter 2.6.2). This hash will be arbitrarily changed in iteration₅₁ (see Figure 28) for P_i , thereby simulating a fault on ECU_1 . This initiates the faulty channel detection and reconfiguration mechanism of the system. The time from the moment the faulty channel was detected ($TC_{2.2}$) until the effected instance of the L4 application will be restarted on the faulty ECU_1 (transition to TC_4) was set to 3.5s to simulate a potential restart duration of the faulty ECU. The number of historic samples was set to 10 since the edge case of 40MB/s required 8 historic samples. Two more were added as buffer.

The remaining test attributes such as update rates or data rates remain the same as in the integration test definition (see chapter 4.4.2).

To get the test set-up as close as possible to a real automotive set-up all three instances of Pi as well as the voters have been deployed on three identical NVIDIA AGX drive boards (see chapter 3.1.1) flashed with a Linux PREEMT_RT Image (see chapter 2.2.2) what represents a state of the art automotive ECU.

In contrast, the source node and the input nodes are operated on industrial computers (PC₁, PC₂, PC₃), each equipped with an Intel Xeon processor with 22 cores operating at a clock speed of 3.2 GHz, 97GB of RAM. This is accepted because they are just acting as input to the system without performance impact (see Figure 40).

The PCs and ECUs are connected with a 1Gb/s Ethernet star topology connection, namely with a Netgear GS108v4 unmanaged 8 port switch.

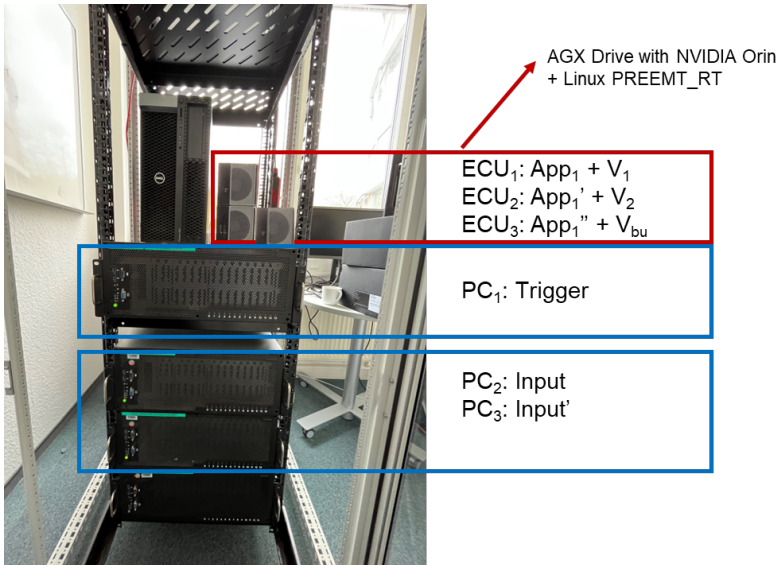


Figure 40: Test bench for Software in the Loop testing (STCs and FAT₁)

5.2.2 Results of not Reconfigured Channel

To operate on an ASIL-D level, the system must use two independent channels (see chapter 2.5.3). In contrast to the faulty channel₁, which was shut down and reconfigured on channel₃, the correct channel₂ continued to operate without reconfiguration. However, the reconfiguration of the faulty channel also had impact on the behavior of the not reconfigured channel₂.

STC₁: The average voting durations t_{add} were around 2ms for all four tested update rates. During the faulty channel detection of TC_{2.2} and for the reconfiguration in TC_{2.3} the voting durations increased significantly up to 114.93ms. During the reconfigured state TC₃ the voting durations were below 4ms and then went back to 2ms in recovered state TC₄ (see detailed numbers in appendix 8.2.1). Thus, the voting durations during nominal state and recovered state have been equal (see red bar on Figure 41).

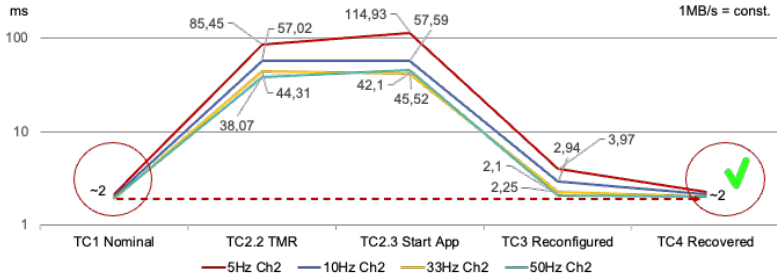


Figure 41: Average voting durations for V_2 for *STC₁*

This was expected since in nominal state as well as in recovered state the deployment configuration of the system was the same. In fact, in nominal state TC₁, as well as in recovered state TC₄, Pi and V₁ operated on ECU₁ and Pi' and V₂ on ECU₂ (see Figure 27 and Figure 31).

STC₂: Same as in *STC₁* also in *STC₂* the voting durations started at around 2ms in nominal state TC₁. An exception was the 40MB/s case which started at 31.75ms. Overall, the system behavior was similar as in *STC₁* where the faulty channel detection duration for TC_{2.2} and the reconfiguration duration TC_{2.3}

increased significantly. However, in contrast to STC_1 the voting durations in recovered state TC_4 have not been the same as in TC_1 nominal state, even though also in STC_2 the system configuration was the same (see red bar and lightning on Figure 42).

The reason for this behavior has two root causes, namely limitations in the network and in ROS2.

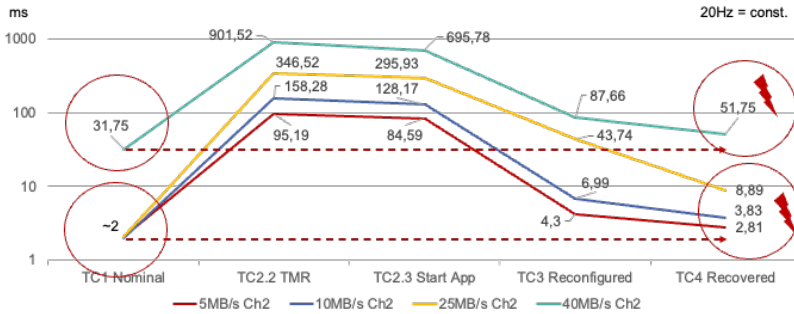


Figure 42: Average voting durations for V_2 for STC_2

5.2.3 Influence of Network and ROS2

To investigate the mentioned phenomenon of longer voting durations in TC_4 compared to TC_1 within STC_2 further, the 25MB/s test was repeated in a more advanced setup. While the logical configuration and the test bench did not change (see Figure 39), four changes have been undertaken. First, the reconfiguration time of TC_3 was increased to 7s to have a longer observation period of the reconfigured state. Second, the faulty sample was injected at S_{200} to have a higher number of samples for TC_1 for statistic relevance. Third, the computers ($PC_1 - PC_3$) and ECUs (ECU_1 , ECU_2 and ECU_{back_up}) got time synchronized using the ptp4l [124] protocol. Fourth, all TCP and UDP packages within the local network of the switch have been recorded for PC_2 , ECU_1 , ECU_2 and ECU_{back_up} using the tool tcpdump [125].

The advanced experiment showed an average t_{add} of 3.89ms within TC₁, t_{add_200} of 344.94ms for channel₂ in TC_{2.2}, a t_{add_201} of 294ms for the reconfiguration of TC_{2.3}, an average t_{add} of 50.33ms in TC₃ and an average t_{add} 31.33ms in TC₄. In general, the curves between the initial 25MB/s test case and the advance test case look similar with some differences (see Figure 43).

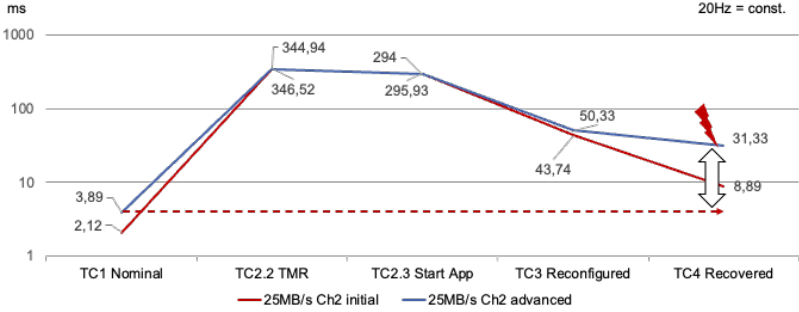


Figure 43: Voting durations in advanced testing for V₂ in 25MB/s

First, the voting durations in TC₁ and TC₃ deviate. The reason is, that in the advanced testing, the CPU was occupied with additional tasks, such as the tcpdump tool to record the Ethernet traffic, as well as the p2p tool to synchronize the system clocks. This led to a higher CPU utilization, creating the higher average values for the advanced testing (see TC₁ and TC₃ on Figure 43). Second, for the advanced set-up the nominal state contained 199 samples, while in the initial testing it has been only 49 samples. Third, the difference between the average t_{add} of TC₁ nominal state and TC₄ recovered state is even more visible for the advanced testing.

Analyzing each system state deeper, two observations have been made. First, the voting behavior of TC₄ was very similar to the reconfigured state TC₃ over a significant duration of the testing. Instead, a similar behavior between TC₄ and TC₁ was expected since the deployment is the same (see gray box on Figure 44). It took roughly 20s until the voting behavior of TC₄ changed again and turned into the expected behavior as in TC₁. The second observation has

been significant peaks of up to almost 2s during the reconfigured operation and at the beginning of the recovered state TC₄.

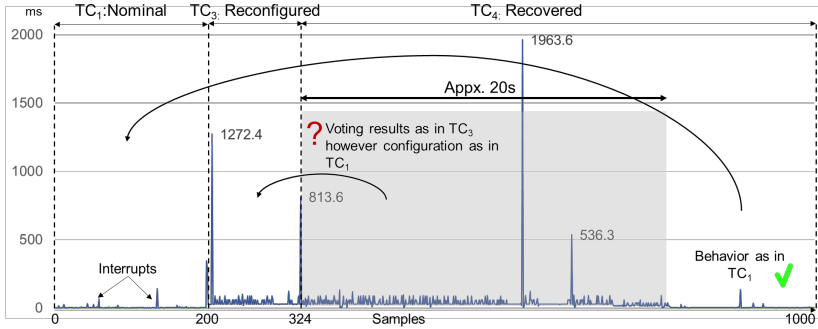


Figure 44: Voting durations of V₂ in advanced testing for Sample 1 - 1000

To further investigate the root cause of both observations the recorded Ethernet traffic was analyzed using the Wireshark tool [126]. It showed, that during TC₁ nominal state the traffic on the Ethernet in average was within the desired range of 25MB/s (see /Input on Figure 45) representing the data flow between Input and Pi on Figure 27). After the fault was detected in iteration 200, Pi'' was activated on ECU_{back_up} and Input started to publish its samples to Pi'' (see /Input_reconfigured on Figure 45 representing the data flow between Input and Pi'' on Figure 30). Pi'' then had to cross share its hash to V₁ using Ethernet instead of shared memory in TC₁ (see /Voter_cross_share on Figure 45). At the same time, Pi was shut down by V₁ since it was identified to be faulty. However, Input did not stop publishing its samples to Pi even though it Pi was already shut down and thus not subscribing anymore at that point in time (see /Input on Figure 45 during TC₃).

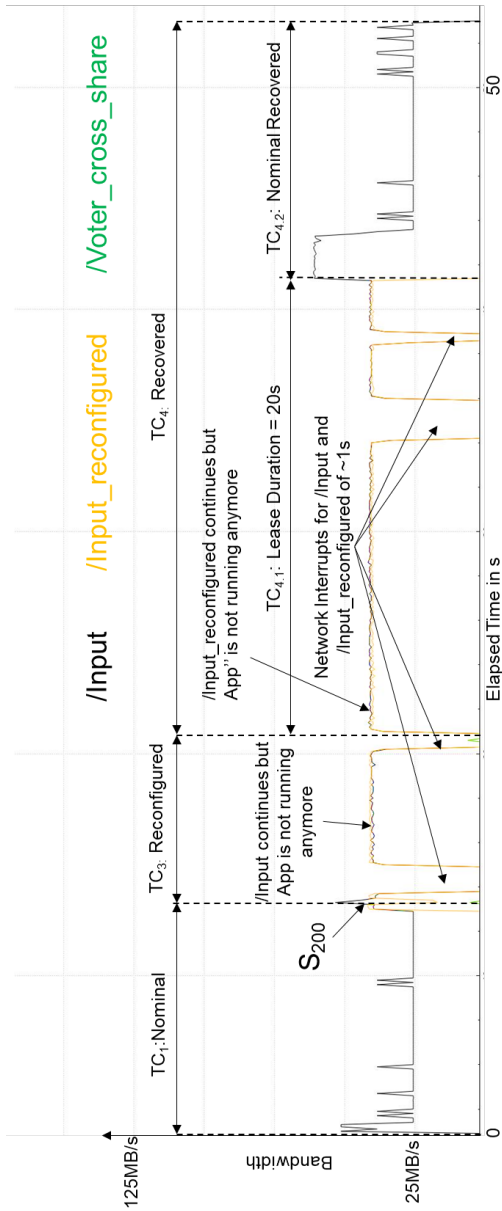


Figure 45: Network traffic in advanced testing based on Wireshark recording

Similarly, after the transition to TC₄ from TC₃, Pi was activated again after the recovery process and Pi'' was stopped. However, Input still published its messages to Pi'' (see /Input_reconfigured on Figure 45 during TC₄). In fact, Input continued to publish for 20 more seconds, even though Pi'' was already shut down and not subscribing anymore. The root cause for this behavior is a FastRTPS DDS configuration mechanism which is called lease duration. It is per default configured at 20s [96] and defines the duration until ROS2 considers a not answering subscriber to be dead. During this time a publisher continues to publish, even though the subscriber is shut down already.

The second aspect is the limited bandwidth of the Ethernet. After the faulty sample identification of TC_{2.1} for S₂₀₀, 10 historic samples have been pulled. To achieve this, FastRTPS was trying to push as much data through the Ethernet as possible. This increased the Ethernet traffic for those 10 samples to over 110MB/s what equals 0.9Gbit/s (see /Total_ethernet_traffic on Figure 46). This amount of data brings the switch to its limit, thereby dropping frames and enforcing Input to stop sending Ethernet frames for appx. one second (see interrupts on Figure 45 and Figure 46). This pause of sending Ethernet frames led to the peak values of the voting durations of almost 2s, thereby increasing the average voting durations (see Figure 44).

However, even though the Ethernet frames are not sent out during this one second on the data link and physical layer, Input continues to publish on the session layer (see OSI Model in Figure 8), thereby filling its publishing queue. Thus, the ~1s Ethernet interrupt fills the queue with appx. 25MB again, which should have been sent but have not been sent physically.

Thus, same as with the historic samples, FastRTPS tries to publish them as quickly as possible which created a peak traffic of almost up to 125MB/s again (see upper gray box on Figure 46), what equals 1Gbit/s. Again here the switch stops the receiving of frames what brings the system into a stall again. This led to the next filling up of the FastRTPS publishing queue, thereby initiating the next interrupt.

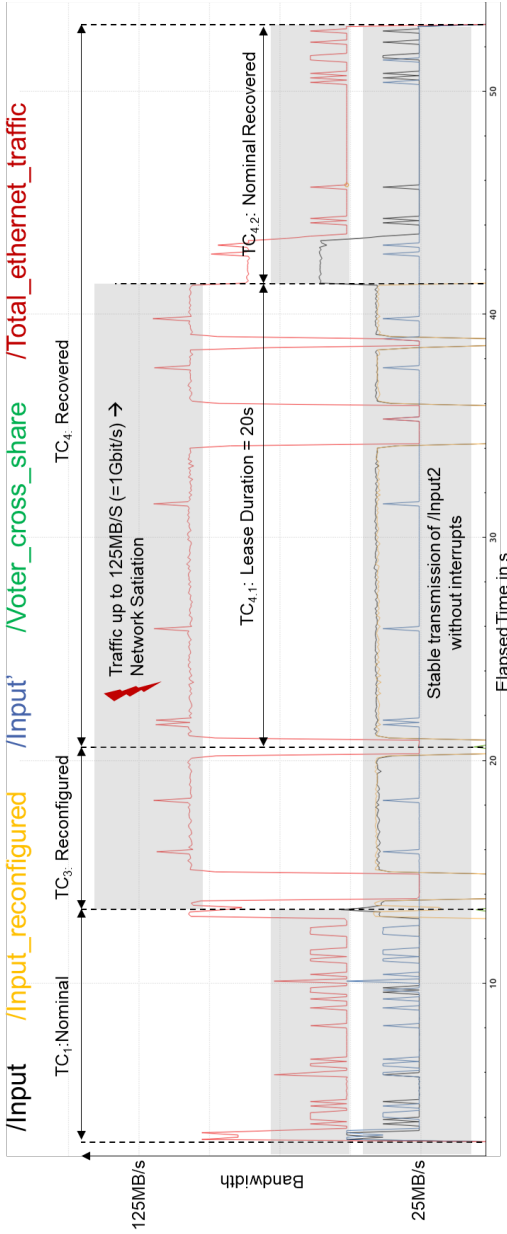


Figure 46: Triplication of Ethernet traffic leading to network saturation

The described bottleneck of Ethernet is intensified by the fact, that during the lease duration the 25MB/s data rate is published three times instead of two times (/Input, /input_reconfigured and /Input' on Figure 46). Out of this reason TC_4 can be divided into the $TC_{4.1}$: *Lease duration* and the $TC_{4.2}$: *Nominal recovered state*.

The advanced testing showed average voting durations of 3.89ms for channel₂ during the TC_1 nominal state compared to 31.33ms during TC_4 recovered state. Splitting up TC_4 shows, that the average voting duration in $TC_{4.1}$ is 42.13ms and for $TC_{4.2}$ it is 3.5ms. Thus, the delta between the average voting duration of TC_1 and $TC_{4.2}$ was 0.39ms, thereby being in the same range again. Also, the standard deviation of 11.42ms in TC_1 for channel₂ was close to the 10.47ms of $TC_{4.2}$ again (see Table 9). Similar voting durations in TC_1 and TC_4 have been expected since the logical configuration in both cases was the same again.

1MB/s const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
TC_4	t_{Voting}	1.40	28.60	7.73	79.05	0.21	0.20	135.42	1917.43
	$t_{Network}$	1.72	2.73	0.97	7.73	0.88	1.13	26.04	136.49
	t_{Add}	3.12	31.33	7.80	80.92	1.10	1.10	137.14	137.14
$TC_{4.1}$	t_{Voting}	1.35	39.53	6.43	90.83	0.21	0.24	74.90	1917.43
	$t_{Network}$	1.62	2.60	0.24	6.45	0.88	1.13	2.12	76.42
	t_{Add}	2.97	42.13	6.47	92.92	1.10	1.10	76.91	76.91
$TC_{4.2}$	t_{Voting}	1.52	0.41	10.37	1.79	0.22	0.20	135.42	24.82
	$t_{Network}$	1.98	3.08	1.77	10.33	1.24	1.20	26.04	136.49
	t_{Add}	3.50	3.50	10.48	10.47	1.49	1.49	137.14	137.14
TC_1	t_{Voting}	1.23	1.25	5.62	10.05	0.19	0.19	65.24	139.68
	$t_{Network}$	2.72	2.64	10.05	5.61	1.12	1.06	141.25	66.74
	t_{Add}	3.95	3.89	11.45	11.42	1.33	1.27	141.47	141.20

Table 9: Split of voting durations for 25MB/s advanced testing in ms

Three elements for further improvement of the systems can be identified. First the bandwidth of the Ethernet itself since a higher bandwidth allows to push more samples through the switch in the same period. Second, to avoid the network stall, a flow control limit for Fast RTPS could be set which is below the critical value of the Ethernet switch. In that case the network would never go into a stall because the ROS2 publisher would control itself to never exceed

the critical limit. This would avoid the interrupts and thereby the high waiting times of the voter. The third element is to manually configure the lease duration. By configuring it to zero the redundant publishing of Input to Pi and Pi'' would be avoided, thereby saving costly Ethernet bandwidth.

5.2.4 Results of Reconfigured Channel

The faulty channel (which is channel₁ in this testing) is reconfigured on channel₃ to keep the system operational.

STC₁ and STC₂: Contrary to the behavior of the not reconfigured channel₂, the voting durations of the reconfigured channel₁ are low in TC₃ and TC₄. The average duration of V₁ is around 2ms during the nominal state in TC₁ until the Sample₅₀. However, the faulty channel detection duration in TC_{2.2} and the re-configuration duration of the faulty instance on ECU_{back_up} within TC_{2.3} increased up to 114.93ms in maximum, followed by a drop down to ~2ms again in reconfigured state TC₃ (see Figure 47) (see detailed numbers in appendix 8.2.1). This voting duration in TC₃ is already almost the same as during the nominal state TC₁, even though the system configuration is different. Furthermore, it stays constant also in TC₄ with around 2ms compared to TC₃.

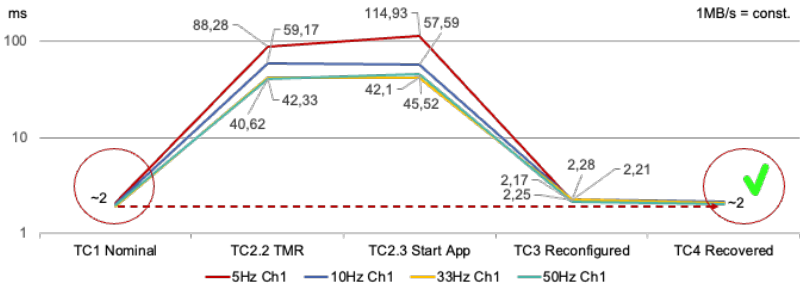


Figure 47: Average voting duration for V₁ for STC₁

The root cause for this behavior resides in short calculation durations of the applications with around 2ms. During the nominal phase, Pi and Pi' receive their samples almost at the same time, thereby start their calculation at the

same time and finish the calculation almost at the same time. P_i calls V_1 very close to the moment P_i' calls V_2 . This is the case for both, STC_1 (see Figure 47) and STC_2 (see Figure 48).

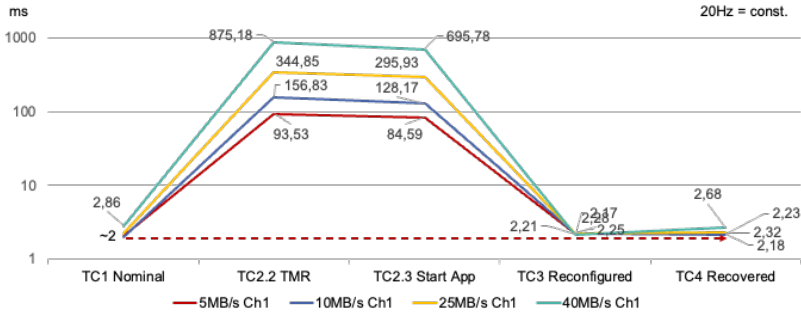


Figure 48: Average voting duration for V_2 for STC_2

There are two reasons why for STC_1 and STC_2 the voting durations are low for the reconfigured channel₁ compared to not reconfigured channel₂ (see chapter 5.2.2).

First, P_i'' and V_1 are not on the same ECU_1 anymore. Instead, P_i'' (on ECU_{back_up}) must send its hash through Ethernet to V_1 (on ECU_1) which takes longer than P_i' (on ECU_2) communicating with V_2 (also on ECU_2) using shared memory. While shared memory minimizes the latency, the hash for P_i'' must be generated from the session layer down to the data link layer of the OSI reference model (see Figure 8) since it is using Ethernet. Thus, V_2 has its hash always earlier available than V_1 what leads to V_2 waiting until V_1 received its hash from P_i'' . This extends the voting duration of V_2 , thereby actually being waiting time rather than voting duration. The other way around that also means, that V_1 never has to wait for V_2 what the reason is why the reconfigured channel₁ (with V_1 as voter) has low voting durations compared to the not reconfigured channel₂.

The second and even stronger reason which leads to the waiting times of V_2 for V_1 are the network effects during the runtime of the system. The reconfiguration did not change the communication of channel₂ between Input' to

Pi'. This is different for the reconfigured channel₁ where in TC₁ nominal state Input published to Pi (on ECU₁) and then changed, publishing to Pi'' (on ECU_{back_up}) during reconfigured operation in TC₃. Even though Pi was shut down, Input continued to publish to both, Pi and Pi''. The root cause resides within the mentioned FastRTPS DDS Lease Duration as part of ROS2 which waits for 20s until it considers a not answering subscriber as dead [96] (see previous chapter).

During reconfigured state TC₃ and during the beginning of the recovered state TC₄, Input has two subscribers, namely Pi and Pi''. This doubles the Ethernet traffic coming from the Input node which goes through the same physical port on the Ethernet switch for PC₂. In contrast, Input' is communicating with Pi' on another physical port of the Ethernet switch. Thus, the transmission of the samples from Input to Pi and Pi'' takes longer than from Input' to Pi'. That also means, that Pi'' starts its calculation later than Pi'. Thus, Pi'' creates its hash and sends it to V₁ later than Pi' to V₂ since it is using Ethernet instead of shared memory.

Both together, Ethernet vs. shared memory and the Lease Duration are the reasons why V₂ waits for V₁ in all iterations of TC₃ and significant parts of TC₄. Thus, this is also the reason why V₁ never has to wait, leading to low voting durations of the reconfigured channel₁.

5.2.5 Behavior of the on-demand TMR State

The analysis of the behavior of the reconfigured and not reconfigured channel showed, that the transition from nominal state (TC₁) to the reconfigured state (TC₃) took the longest duration. This transition is done by the on-demand TMR state as captured in TC_{2.1}, TC_{2.2}, TC_{2.3}.

STC₁: The testing showed that the biggest influence of the faulty channel detection duration and the reconfiguration duration of the FHTI is determined by the looping times through the historic samples, taking 42.03ms for t_{loop} and

72.79ms for $t_{\text{loop_start}}$ for STC_1 (see left side of Figure 49 based on Table 14 and Table 15). The reason for the different loop times relative to the update frequencies resides in the sample sizes. A constant data flow of 1MB/s for the case of 5Hz means a size for each sample of 200KB and a sample contains 20KB when published at 50Hz. Accordingly, the sample sizes of the lower rates are higher than those for the higher update rates (see Table 10).

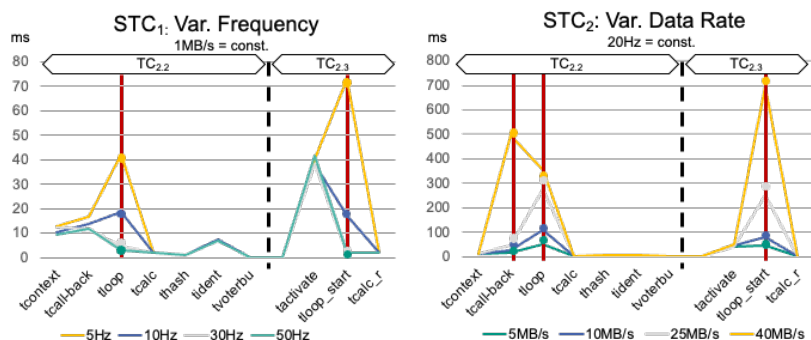


Figure 49: On-demand durations for STC_1 and STC_2 as in $\text{TC}_{2.2}$ and $\text{TC}_{2.3}$

While the number of ten historic samples require to pull 2000KB for the 5Hz STC (since a sample contains 200KB) it is only 200KB for the 50Hz STC. The higher amount of data for the lower data rates leads to the situation that the looping duration for the lower data rates is higher than for the higher data rates.

	Sample Size	10 Historic Samples	
5Hz	200KB	2000KB (2MB)	1MB/s = const.
10Hz	100kB	1000KB (1MB)	
33Hz	30KB	300KB (0.3MB)	
50Hz	20KB	200KB (0.2MB)	
5MB/s	250KB	2500KB (2.5MB)	20Hz = const.
10MB/s	500KB	5000KB (5MB)	
25MB/s	1250KB	12500KB (12.5MB)	
40MB/s	2000KB	20000KB (20MB)	

Table 10: Sample sizes for test cases

This situation is even more visible for STC_2 (see right side of Figure 49). While each sample for the 5MB/s case at 20Hz has a size of 250KB it varies up to 2MB per sample for the 40MB/s case (see Table 10). The impact can be seen in t_{call_back} which represents the time until the first historic sample was pulled. With increasing sample sizes also the loop time through the historic samples increases from 50.94ms for 5MB/s to 351.88ms for 40MB/s (see Table 14).

The same behavior of t_{loop} is also observable for t_{loop_start} . However, both numbers are not directly comparable. While for t_{loop} the system is looping through the historic samples to identify the wrong channel, t_{loop_start} is done by the reconfigured Pi'' to synchronize into the system state. At the point in time t_{loop_start} begins more time is elapsed than when t_{loop} started. While having the same number of historic samples than t_{loop} only a subset will be used for t_{loop_start} , since in the meantime new samples have been published.

As conclusion, another element to improve the system behavior is the optimization of the number of historic samples.

5.2.6 Influence of Historic Samples

The number of historic samples directly influences t_{loop} and t_{loop_start} with the consequence that a high number of historic samples leads to a longer loop time of Pi'' in iteration₅₁ (A_{51}) and iteration₅₂ (A_{52}). Exemplary TC_1 at 5Hz with the faulty iteration₅₁ (with S_{51} as input) will be analyzed. S_{51} sample was processed by Pi in iteration A_{51} which sent its result to voter V_1 . V_1 detects the fault and forwards its hash to V_{bu} . V_{bu} then started the faulty channel detection mechanism. During the faulty channel detection duration, Pi was blocked and waiting for the result if its channel was faulty or not. As the results of $TC_{2.2}$ showed, Pi'' was able to receive the first historic sample after 29.46 seconds (sum of $t_{context}$ and $t_{call-back}$ in Table 14) and then pulled the ten historic samples. However, at this point in time no new sample was published since the cycle time of 5Hz is 200ms what made Pi'' pulling and iterating through nine

historic samples what took 37.5ms and was not required since in the meantime no new sample was published.

The same applies for the activation of the reconfigured instance Pi'' in $TC_{2.3}$ during the reconfiguration duration to continue with A_{52} . After 40.02ms (see Table 15) it pulled its first historic sample and then iterated until S_{52} which was not published at that point in time. Thus, no historic sample was needed for this case because the identification process ($TC_{2.2}$) and the reconfiguration process ($TC_{2.3}$) in total required less time than one cycle of the input nodes (see Figure 50).

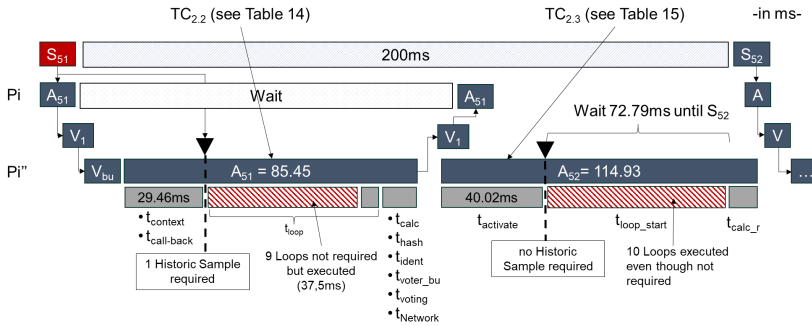


Figure 50: Activation behavior for STC1 at 5Hz

Another illustration of the optimization potential is visible in the STC_2 for 10MB/s (see Figure 51). Contrary to the 5Hz example, in this case a new sample is published every 50ms. Since Pi'' pulled the first historic sample after 33.7ms (see Table 16) only one historic sample was required at that point in time. Again here, nine historic samples are not necessary which were wasting 95.8ms. However, the system continued to publish samples during $TC_{2.2}$. Accordingly, when Pi'' was reconfigured, it pulled the first historic sample after 45.62ms. That means, that in the meantime, four new samples have been published, namely S_{52} , S_{53} , S_{54} and S_{55} . Thus, those samples are needed as historic samples, but not the six samples before (S_{46} - S_{51}).

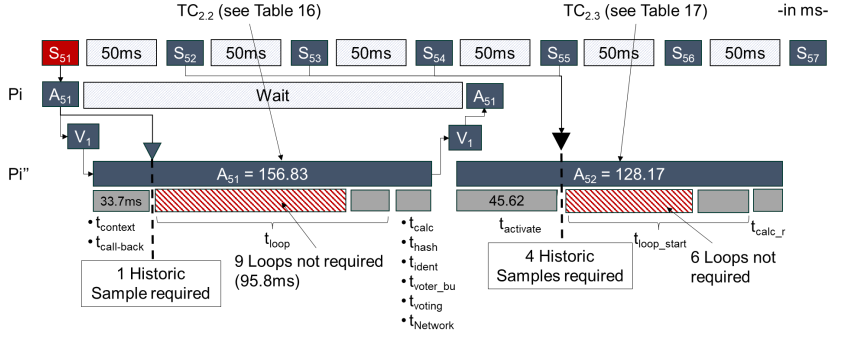


Figure 51: Activation behavior for STC₂ at 10MB/s

The optimization of the faulty channel detection duration $TC_{2.2}$ and reconfiguration duration $TC_{2.3}$ cannot be done isolated. If $TC_{2.2}$ is optimized, this also means that a smaller number of samples has been published when $TC_{2.3}$ starts. Thus, the reduction of the historic samples for $TC_{2.2}$ leads to an earlier start of the reconfiguration $TC_{2.3}$. This reduces the time between $TC_{2.2}$ and $TC_{2.3}$ what leads to a smaller number of new samples in between. (see Figure 52).

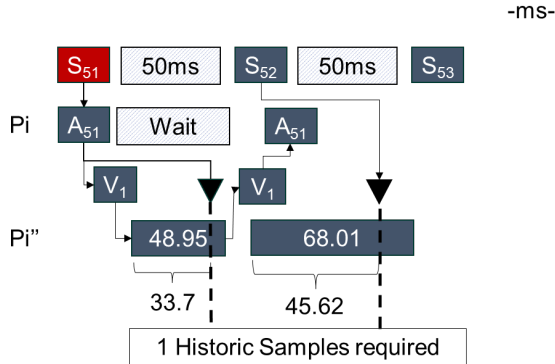


Figure 52: Optimized activation behavior for STC₂ at 10MB/s

5.2.7 Interim Summary

The successful processing of all 1000 samples within STC_1 and STC_2 indicates that the on-demand TMR system can execute the defined four system states: Nominal state, on-demand TMR state, reconfigured state and recovered state. Thus, the general functionality of fault detection, faulty channel detection, reconfiguration to stay operational and its recovery to get back into the nominal state was demonstrated.

In summary, the test results indicate, that the following general conclusive statements can be made:

1) The minimal average voting duration approximates towards 2ms:

The STCs showed average voting durations constant around 2ms during the nominal state (TC_1) for all test cases except of 40MB/s (see Figure 53 and details in Table 12, Table 13). During reconfigured state TC_3 , the average voting durations also were stable around 2ms for all STC_1 and the 5MB/s test case of STC_2 . A slight increase to around 4ms was visible for the 10MB/s test case and a significant increase for the 25MB/s and 40MB/s test case (see Figure 53 and details in Table 18, Table 19). Those results indicate the approximation towards 2ms regardless how small the samples are (see gray box on Figure 53).

2) The average voting duration is stable until a certain threshold. The almost constant voting duration until 25MB/s indicates that within this range the real utilization on the Ethernet has no significant impact on the average voting durations. However, once a certain Ethernet utilization threshold is achieved, the average voting durations increase significantly. In fact, for the implemented 1G Ethernet star topology (see chapter 5.2.1) this threshold was 25MB/s in the nominal test case TC_1 (see blue line on Figure 53). The threshold of 10MB/s in TC_3 is related to the effect of the lease

duration and can be moved to 25MB/s by applying the identified counter measures (see chapter 5.2.3).

- 3) **The sample size has a direct influence on the Fault Handling Time Interval:** The results indicate that with increasing sample sizes the duration of the FHTI also increased accordingly (see dashed black line on Figure 53 or details in Table 14, Table 15, Table 16, Table 17). This behavior can be traced back to the 10 historic samples which require to pull 200KB in the 50Hz test case and 20.000KB in the 40MB/s test case. Using a 10Gbit/s Ethernet or PCIe instead of 1Gbit/s would significantly increase the faulty channel detection times. It would also allow a stable use of the on-demand TRM for the 25MB/s and 40MB/s test cases.
- 4) **The number of historic samples directly influence the FHTI:** Similarly to the sample sizes a higher number of historic samples also means a higher amount of data which must be pulled through the Ethernet during TC_2 . The system behavior can be optimized by carefully selecting the number of historic samples.

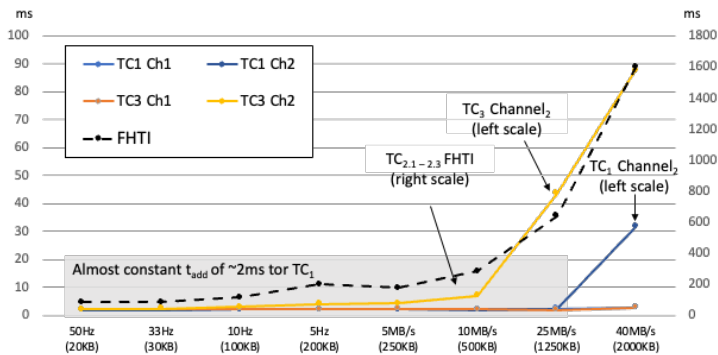


Figure 53: Summary of average voting durations and FHTI per test case

Additionally to the generalized statements the following additional observations have been visible.

It was identified that a significant portion of the increasement of the average voting durations within the reconfigured state TC_3 can be traced back towards the FastRTPS lease duration. It continues to publish to a subscriber even though the subscriber is not active anymore. That means, that during the re-configured state (and also at the beginning of the recovered state TC_4) the input nodes continue to publish for 20s to all three nodes (P_i , P_i' and P_i'') instead to only two (either P_i and P_i' or P_i' and P_i''). This brought unnecessary additional traffic to the Ethernet, leading to the longer voting durations in TC_3 and the first 20s of TC_4 . This fact is even sharper for the 25MB/s and 40MB/s test case, where the triplication of the Ethernet traffic was exceeding the maximal bandwidth of 1Gb/s, thereby bringing the network into a stall.

The findings will be further analyzed by applying the on-demand TMR system to a real level 4 lane detection algorithm as used in an autonomous long-haul truck.

5.3 L4 Lane Detection Testing

5.3.1 System Set-Up

FAT₁ Lane detection as Software in the Loop:

The selected robust lane detection algorithm (RoLanD) is an algorithm using a CNN for lane detection purposes. The output of RoLanD is the offset of the center of the vehicle to the left and to the right lane marking. It was trained for usage on interstates in southern states of the USA.

In contrast to STC_1 and STC_2 where only the CPU was used, RoLanD is also using the GPU for inference of the CNN (see chapter 2.2.1). Since momentarily there is no ASIL-D ready GPU available, this experiment will show if the system can be brought to an ASIL-D level with the ASIL decomposition principle by running two instances of RoLanD on two different GPUs, each having ASIL-B (see chapter 2.5.3).

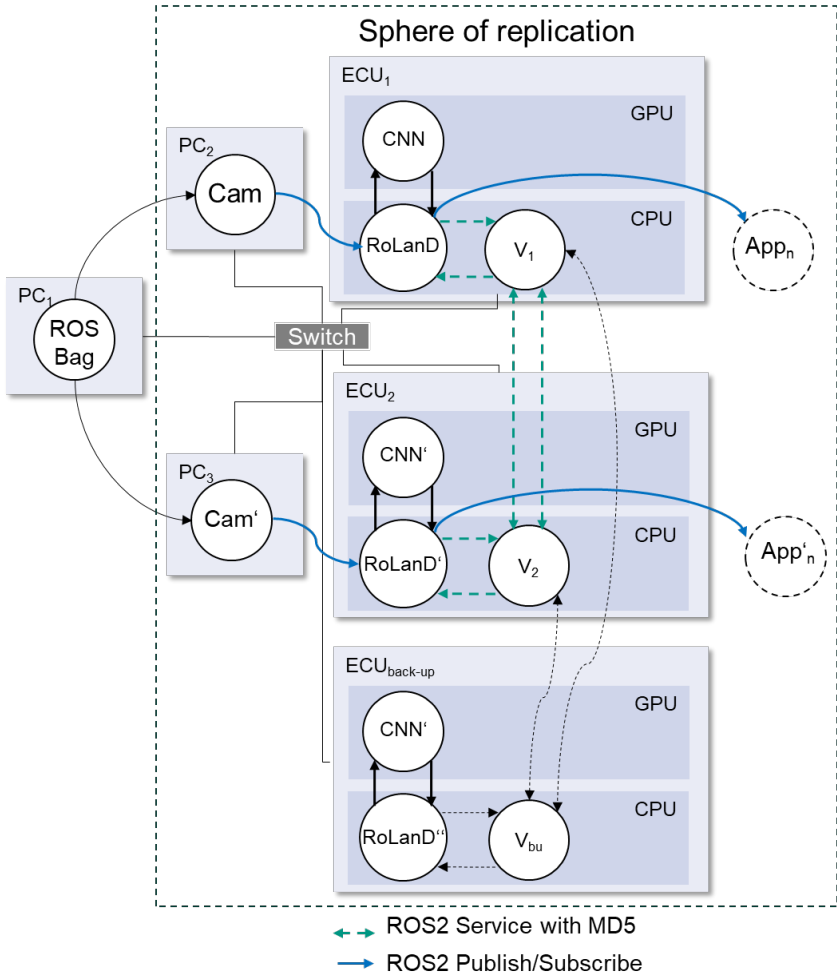


Figure 54: Deployment schematic for FAT₁ testing

Similarly to the test-bench in the previously described system testing approach, also here the trigger is allocated on PC₁. However, it is not only a trigger but a recorded camera stream in the form of a ROS2 bag. Initially it was recorded as ROS-Bag which was manually converted to ROS2. It cyclically publishes the raw images to the redundant camera nodes which forwards the

images directly to RoLanD without any further processing. The reason for this set-up is same as for the STCs (see chapter 5.2.1). The ROS bag still is a single point of failure and again here the target of the redundant camera nodes is to simulate the fully redundant channels.

After the redundant instances of RoLanD received images, each instance loads its image into the GPU for inference. As return it receives the detected lanes (see Figure 55). The distance to the left and right lane marking from the center of the vehicle is saved in a state vector, which then is hashed into MD5 (see chapter 2.6.2). The hash represents the input to the voters for the on-demand TMR system (see Figure 54).

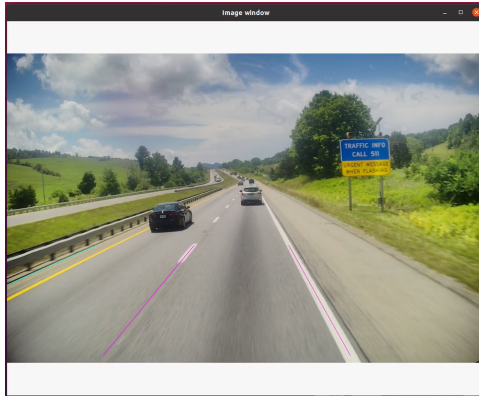


Figure 55: Lane markings as output of RoLanD

The ROS bag published compressed images with an average update rate of 3.7Hz (every 265ms). However, it is not possible to define a fixed update rate when using the ROS2 bag replay. Instead, the replay speed can be reduced as a percentage of the recording speed. However, still the ROS2 replay is not exactly replaying the bag as it was recorded. It is rather exposed to system interrupts what led to a standard deviation of 102ms for publishing each image. Thus, the update rate changed during runtime and was not stable.

One image had a size of 1.6MB. This leads to an average data rate of 5.9MB/s what is inside the boundaries of the STCs (with 5MB/s test case as closest).

The numbers of historic samples has been arbitrarily set to 15 since the re-configuration took longer than in the STCs. Out of this reason, the initially selected number of 10 historic samples was not sufficient for the FAT testing.

In FAT₁ RoLanD was executed on a test bench which was the exact same as for the system testing (see chapter 5.2.1).

FAT₂ Lane detection as real world testing

To further analyze the system behavior, it was extended and implemented into a long-haul Freightliner Cascadia truck. This vehicle integration has been done as part of a closely supervised master thesis by Gupta [97].

While the previously introduced system set-up as in STC₁, STC₂ and FAT₁ realized redundancy for the applications and the voters, this was not the case for the source of data, namely the trigger node in STC₁ and STC₂ or the ROS Bag in FAT₁ (see chapter 5.2.1 and previous paragraph). Accordingly, the ASIL decomposition principle has not been valid for the input of the system.



Figure 56: Position of two Leopard cameras above windshield on the truck

This limitation was closed by placing two cameras above the windshield side by side on the center of the cabin. Their physical distance was 2.5cm (see Figure 56).

Each camera has an own field of view (FoV). Due to the physical displacement, the resulting images are different. However, especially since the cameras are located side by side, both images have a wide overlapping area which is the region of interest (RoI) and should be very similar (see Figure 57).

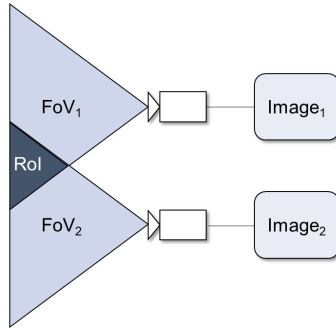


Figure 57: Overlapping areas of cameras [97]

The proposed dual cropping algorithm identifies the RoI by detecting features on each image. It uses existing feature detection algorithms (e.g. shapes, illuminations) on both images. The area on each image which had the same features is considered to be the RoI. To only process the RoI, the remaining part is cropped out on each image (see Figure 58).



Figure 58: Features and cropping area [97]

The physical realization was done by using two Leopard cameras of type LI-IMX728-9295-120H (see Figure 56) which have been attached to a Solectric Frame Grabber PCIe card on PC₁. PC₁ and the ECUs have been connected with the same 1Gbit Ethernet switch as in FAT₁ and a star topology (see chapter 2.3.1).

The program flow started with the images which have been captured by both cameras with an update rate of 25Hz. To convert the ROS message from the camera driver to a ROS2 message for the cropping nodes, both images have been published to an own instance of the ROS bridge, since the camera driver itself was developed in ROS1. The ROS bridge then forwarded the image to the respective instance of the cropping nodes on the two distinct ECUs (see Crop₁ and Crop₂ on Figure 59).

The cropping nodes have been deployed on the exact same NVIDIA Drive AGX ECUs as in FAT₁. Thus, they operate on the same SoC as RoLanD and the Voters. This deployment led to a change within the communication between the data producer (ROS Bag in FAT₁ and cropping node in FAT₂ to RoLanD). While in FAT₁ the ROS bag communicated with both instance of RoLanD using Ethernet (see Figure 54), the cropping nodes in FAT₂ communicate with RoLanD using shared memory (see red arrow on Figure 59) (see chapter 2.2.2).

Since the update rate of 25Hz was too fast for RoLanD it was reduced to 3Hz. This was realized by implementing a timer which waited for 333ms before cropping and forwarding an image (see red clock on Figure 59). The images in between have been dropped unused.

The reason why the reduction of the update rate was realized inside two cropping nodes and not in the camera driver directly was to get the system setup as close to reality as possible. In a production system, the step over a PC with a frame grabber card is not necessary. The cameras would rather be attached to the ECUs directly what means, that the trigger for each camera would also be on each ECU, simulated by the timers and not on one PC which contains a frame grabber card.

Once the cropping of the images has happened, they are forwarded into both instances of RoLanD and continue from there in the same manner as in FAT₁ testing.

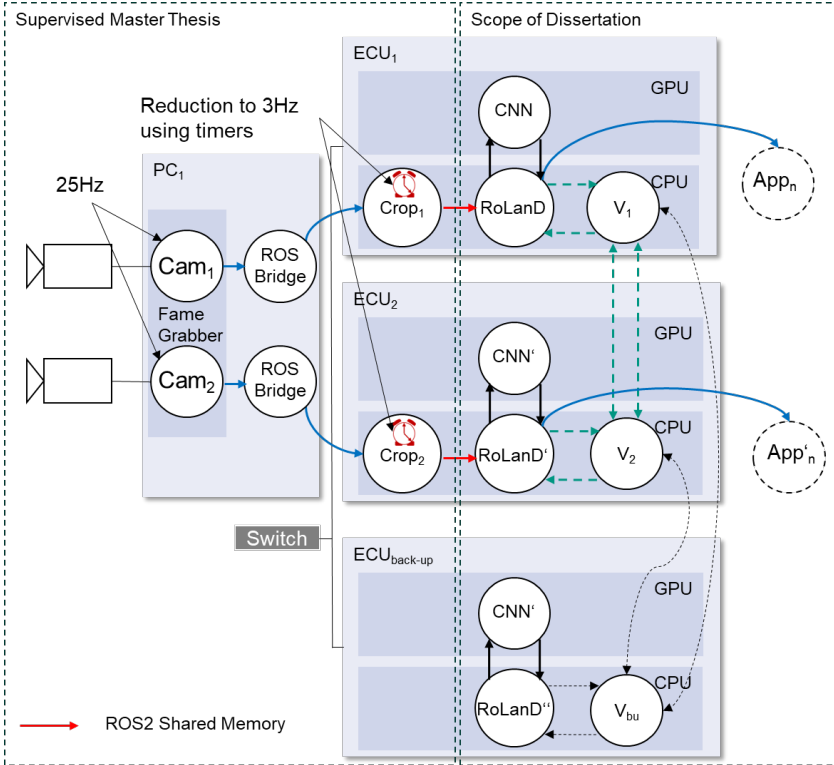


Figure 59: Simplified schematic of enhanced system set-up for FAT₂ testing

5.3.2 Results and Comparison to System Testing

The general shape of the system behavior for FAT₁ and FAT₂ was the same as for the system testing (see Figure 60 and Figure 61) (see detailed numbers in appendix 8.2.2). However, there are differences observable between the FAT and the STC testing, as well as between FAT₁ and FAT₂.

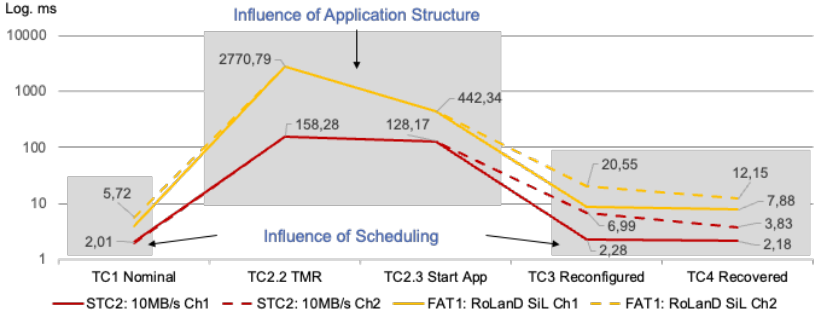


Figure 60: Voting durations for FAT₁ compared to 10MB/s STC₂

When comparing the results from the STC testing to the results from FAT testing it was observable that the voting duration in each TC was higher for FAT than for STC. In fact, the voting duration for the 10MB/s STC₂ case was in average 2.01ms in nominal state TC₁, while FAT₁ had an average voting duration of 5.72ms in the same test case. The same behavior was visible in TC₃ and TC₄. The reason for this is the longer processing time of RoLanD compared to the calculation of the number Pi. This leads to a higher exposure of RoLanD against system interrupts, summarized as “influence of scheduling” (see Figure 60).

The second factor which creates a difference between FAT and STC is the size of the application and the way it is allocating memory during runtime, summarized as “influence of application structure”. RoLanD needs a trained neural network loaded to the GPU while the calculation of the number Pi is a simple function which is executed only on the CPU. The combination between more data, necessary for the inference of the neural network and the fact that it must be loaded from the CPU to the GPU makes the on-demand durations of TC_{2.2} and TC_{2.3} longer for the FATs compared to the STCs.

There are also two areas where FAT₂ (real world) deviates from FAT₁ (SiL). First, the voting durations of FAT₂ with up to 14.01ms were significantly higher during nominal phase compared to FAT₁ with only 5.72ms. The root cause resides in the implemented time synchronization mechanism which

was used to reduce the update rate of the camera in FAT_2 and is summarized as “influence of time synchronization”.

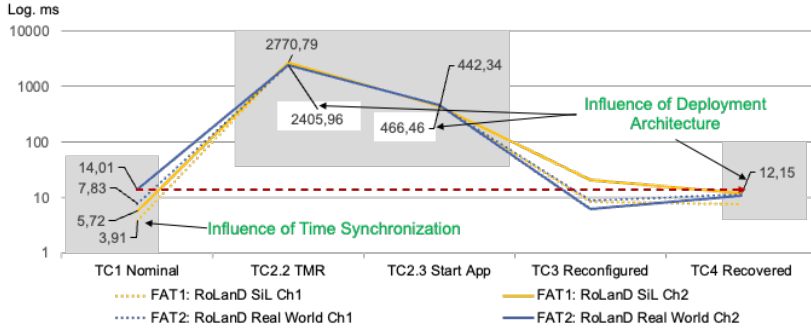


Figure 61: Comparison voting durations FAT_1 to FAT_2

The second difference between FAT_1 and FAT_2 is related to the different allocation of the nodes and the way they communicate, summarized as “influence of deployment architecture” (see Figure 61). The lower on-demand state durations in $TC_{2.2}$ and $TC_{2.3}$ of FAT_2 compared to FAT_1 are related to the fact that in FAT_1 the historic samples were pulled over Ethernet, while in FAT_2 they were pulled using shared memory due to the added cropping nodes. Additionally, the effect of lease duration (see chapter 5.2.3) was visible within FAT_1 but not within FAT_2 (see red bar in Figure 61). Again here, the root cause is in the different deployment architecture of FAT_1 compared to FAT_2 . The reduction of the update rate of the camera frames inside the cropping node leads to the situation, that the effect of lease duration is still there, but not visible.

5.3.3 Influence of Scheduling and Application Structure

Influence of Process Scheduling: As mentioned, the FAT_1 testing showed higher average voting durations than STC_1 and STC_2 (see Figure 60). This also applies for TC_3 and TC_4 . The reason is the difference in the calculation duration of the instances of the L4 Applications. For instance, it took 1.88ms in average at 10Hz in STC_1 to calculate the number Pi with a standard deviation of 0.03ms

on ECU₁. As comparison, it took in average 89.91ms with a standard deviation of 9.18ms for FAT₁ on ECU₁. Since the calculation duration for FAT₁ is significantly longer than in STC₁ and STC₂ its exposure to system interrupts is higher. Even though a real-time Linux was used, there are still processes which overrule the application calculation process, thereby pausing it for a while. However, those system interrupts are not happening at the exact same time on both ECUs, since each has an own OS with an own scheduler. This leads to the situation that always one L4 application is finished before the other, thereby making the other voter waiting. Accordingly, the higher voting durations are rather waiting times from one voter to the other. An example is Iteration₁₉₁, where RoLanD' finished 23.01ms earlier than RoLanD', making V₂ wait before the voting (v) can be finished to publish (p) the result (see Figure 62).

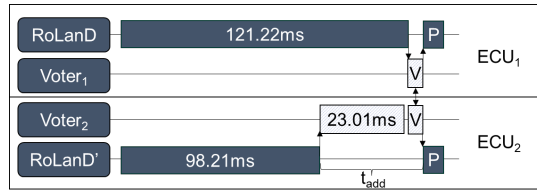


Figure 62: Exemplary differing calculation times of RoLanD in iteration₁₉₁

The described varying calculation time happens not only for a single iteration but for each through the whole operation of the system. Therefore, waiting times for the other voter occur in each iteration and thus increase the voting duration t_{add} (see Figure 63).

Previous work as part of this dissertation showed that the usage of a Linux PREEMT_RT was already significantly reducing the process interrupts and along with this the voting durations of the on-demand TMR system [98]. However, studies demonstrated that the Linux Kernel, also when used together with the PREEMT_RT patch, is not hard real-time capable [24]. Still the latencies cannot be predicted reliably to guaranty a WCET. This is the reason why a Linux PREEMT_RT can be used for academic purposes but is not recommended for safety critical systems [24].

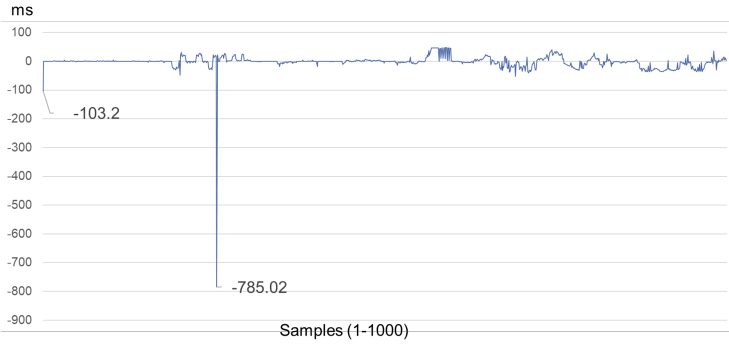


Figure 63: Calculation duration deltas RoLanD' vs. RoLanD('') for FAT₁

A way to overcome this limitation is by using commercial operating systems, such as QNX. Those commercial OS use micro kernel architectures which perform as a hard real-time system and guarantee fixed calculation times, thereby making the OS predictable. The calculation durations of the on-demand TMR system could be minimized and synchronized against each other. This would eliminate the waiting times for the voters, thereby minimizing the average voting durations and standard deviation for compute heavy applications such as RoLanD in FAT₁.

A peak is visible at iteration₁ where RoLanD needed 103.2ms longer than RoLanD' and iteration₂₅₅ where RoLanD needed 785.02ms longer than RoLanD'. The deeper analysis of the results showed that always the first iteration of RoLanD happening at A₁ (in TC₁), A₂₀₀ (in TC_{2.2}), A₂₀₁ (in TC_{2.3}) and A₂₅₅ (in TC₄) took in the minimal 811.74ms and in maximum 918.85ms to be processed while for each following sample it was only between 90ms to 121ms (see Table 11).

The longer calculation of the first inference cycle of RoLanD after its initialization can be traced back to the memory allocation which happens in the first iteration. Once after its startup RoLanD received the first image as input, RoLanD will start to process it and allocates the necessary memory to the GPU. After this memory was allocated, each following iteration will use the memory

space without the need to allocate it again. Out of this reason the first iteration takes longer than each following iteration. A solution of this would be to change the memory allocation mechanism. Accordingly, it could already happen during initialization time of RoLanD rather than after receiving the first image for inference.

For A_{200} an even a longer processing time of in total 2041.03ms was observable which can be divided into 1229.29ms needed to dispatch the process to the CPU after activation by V_{bu} and 811.74ms needed for the mentioned first inference. The reason for the dispatch time is that the process was for a longer time on pause, thereby being deprioritized by the scheduler of the OS. Thus, it pushed the data to a memory area which is not CPU close, thereby not consuming costly L1, L2 or L3 cache (see chapter 2.2.1).

	RoLanD	RoLanD'	RoLanD''		Note
	Inference	Inference	Inference	Activation	
A_1	918.85	815.68	-	-	TC_1
$A_2 - A_{199}$	89.91	88.97	-	-	
A_{200}	118.04	121.14	811.74	1229.29	$TC_{2,1}, TC_{2,2}$
A_{201}	-	125.26	173.32	-	$TC_{2,3}$
$A_{202} - A_{254}$	-	111.16	105.82	-	TC_3
A_{255}	899.41	114.39	-	-	TC_4
$A_{256} - A_{1000}$	107.9	106.06	-	-	

Table 11: Calculation durations for FAT_1 in ms, first iterations of an instance in red

Influence of Application Size: In contrast to RoLanD, the system testing had a calculation duration for the faulty iteration in STC_1 and STC_2 of 2.11ms – 2.4ms (see Table 14 and Table 16) which is not significantly deviating for all other calculation durations until iteration₁₀₀₀. The difference between the STCs and the FATs using RoLanD'' is the size of data which is required to run the execution. In fact, for RoLanD a neural network must be dispatched to the GPU memory after RoLanD was activated by V_{bu} . This is not necessary in the system testing. This loading of the neural network requires time, thereby having a significant contribution to the 1229.29ms activation time.

Since the caches are limited not all processes can be loaded jointly into them what makes a prioritization for usage necessary. An option is to increase the

size of the caches to allow more data to be stored. However, since this impacts the hardware design of the SoC this measurement is rather hard to realized. The second option to optimize the behavior is to combine the independent steps of faulty channel detection ($TC_{2.2}$) and temporary reconfiguration ($TC_{2.3}$) since they use the same data, namely the CNN. The current design cleans the memory area after finishing the faulty channel detection ($TC_{2.2}$) and then pushes data into the same memory area again ($TC_{2.3}$) which is not the exact same but has big overlaps such as the neural network itself. By doing so the 173.32ms of $TC_{2.3}$ could be reduced closer to the average execution time of around 90ms.

5.3.4 Influence of Time Synchronization and Deployment

Influence of Time Synchronization: While the higher voting durations of up to 5.72ms in TC_1 in FAT_1 compared to the ~ 2 ms in the STCs are related to the longer calculation times of RoLanD (see previous chapter), there is also a difference visible between FAT_1 and FAT_2 . In contrast to FAT_1 where $channel_1$ had an average voting duration of 3.91ms and $channel_2$ of 5.72ms, the spread in FAT_2 between $channel_1$ with 7.8ms and 14.01ms for $channel_2$ is significantly higher (see Figure 61). The reason of the higher spread is the extension of the architecture during the supervised master thesis. While in FAT_1 the data source was a single ROS bag which contained one camera image (see Figure 54) this limitation of a single point of failure was closed by adding two cameras instead (see Figure 59).

Both cameras had an initial update frequency of 25Hz and they were deployed on the same PC₁. However, even though they have been attached the same PC the trigger to both cameras to capture an image was not always at the exact same point in time. This led to an offset between the capture time of the images and in consequence also to an offset in forwarding the image to the cropping node (see $offset_1$ on Figure 64).

Additionally, the 25Hz update rate was too fast for the inference of RoLanD and out of this reason it was reduced to 3Hz. This reduction of the update rate was realized inside both cropping nodes on ECU₁ and ECU₂ by setting a timer of 333ms (see clocks on Figure 59). During this duration, all incoming samples were dropped. The most current sample after the timer elapsed was forwarded to RoLanD.

However, this set-up requires a perfectly synchronized timing between ECU₁ and ECU₂ what was not the case within the experiment. Since Crop₁ and Crop₂ have not been started at the exact same time there was always a slight offset between the timer on ECU₁ and the timer on ECU₂. In fact, Crop₂ was started slightly after Crop₁. Out of this reason, Crop₂ forwarded its sample a little bit later to RoLanD', than Crop₁ to RoLanD (see offset₂ on Figure 64). Since both instances of RoLanD directly forward their samples to V₁ or V₂, again here V₁ always had to wait for V₂ since the result of RoLanD was continuously available at V₁ prior to the result of RoLanD' at V₂. As result, V₁ always had to wait for V₂ what causes the spread between the average voting durations between channel₁ and channel₂.

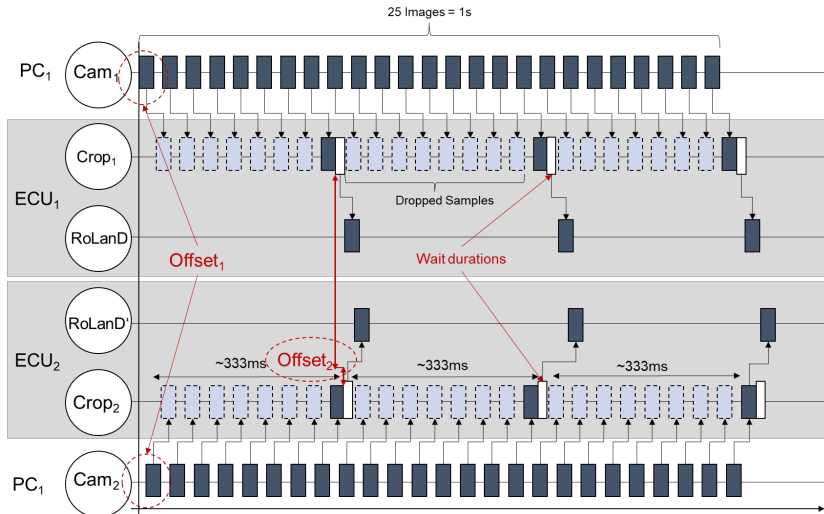


Figure 64: Schematic of time synchronization

Influence of Architecture: The first aspect where a difference in behavior of FAT_2 compared to FAT_1 as well as STC_1 and STC_2 was visible is the voting behavior in TC_4 . While FAT_1 shows the same behavior as STC_1 and STC_2 where the voting durations in TC_4 have been higher than in TC_1 during the 20s lease duration (see chapter 5.2.3), this was not the case for FAT_2 (see red dashed bar in Figure 61).

Again here, the cause for this difference is in the different deployment. As explained, the camera is publishing with an update rate of 25Hz, which is reduced to 3Hz in both cropping nodes. That means that the cropping nodes drop the not necessary camera images and only publish the latest one after a timer of 333ms has elapsed. However, the images arrived prior to the point in time when the timer elapsed. That means that the image was stored for a short period of time and waited before it was cropped and forwarded to RoLanD (see wait durations on Figure 64). Thus, the increasing voting durations due to the lease duration as in FAT_1 are filtered by the wait durations created by the timers of the cropping node. This waiting time was longer than the increasing transmission times from the ROS-Bridge to the cropping node because of the effect of lease duration. As result, for RoLanD and the voters, the effect of the lease duration is not visible anymore in FAT_2 .

The second aspect inside the influence of architecture is the difference in the faulty channel detection duration $t_{\text{third_hash}}$ of $TC_{2.2}$ between FAT_1 compared to FAT_2 . While it took 2770.79ms for FAT_1 to create the third hash for the faulty channel detection, it took 2405.96ms for FAT_2 (see Figure 61 or Table 24). Drilling deeper into the test results shows, that the main difference comes from the looping time through the historic samples t_{loop} and $t_{\text{loop_start}}$ where FAT_2 was significantly quicker than FAT_1 (see Figure 65).

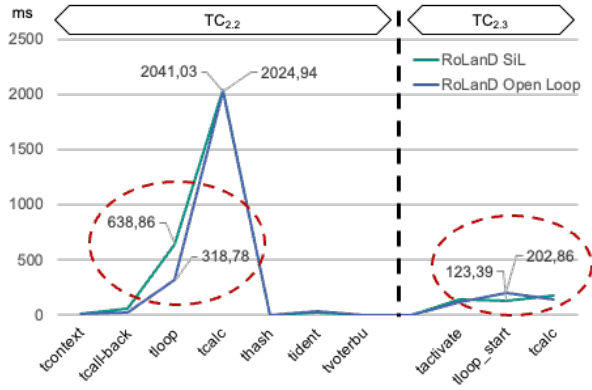


Figure 65: On-demand durations for FAT_1 and FAT_2 as in $TC_{2.2}$ and $TC_{2.3}$

The reason is the different deployment of software components in FAT_2 compared to FAT_1 . The image cropping nodes were deployed on the same ECUs as RoLand and the voters what means that the Ethernet connection in FAT_2 is between the ROS-Bridge on PC_1 and the cropping nodes on ECU_1 and ECU_2 and not between the Cam nodes and RoLand anymore. Thus, in FAT_2 the cropping node and RoLand communicate with shared memory and not with Ethernet as in FAT_1 (see solid red bar on Figure 59 vs. Figure 54). Out of this reason, the historic samples are not pulled through Ethernet but through shared memory in FAT_2 . This leads to the faster faulty channel detection duration compared to FAT_1 .

5.3.5 Interim Summary

The execution and analysis of FAT_1 and FAT_2 indicates, that the on-demand TMR system is promising for usage for real L4 autonomous driving applications. In FAT_1 and FAT_2 the system again was able to process all 1000 samples by detecting the faulty iteration, followed by the reconfiguration and the recovery.

While STC_1 and STC_2 demonstrated relatively stable voting durations around 2ms until 10MB/s (see chapter 5.2.7), those voting durations increased up to 5.7ms for FAT_1 and ~14ms in FAT_2 . This deviating behavior between the $FATs$ and $STCs$ can be traced back to the following two root causes:

- 1) **Processing duration:** The longer one calculation iteration takes, the more it is exposed to system interrupts. In fact, the processing of RoLanD was in average about 90ms compared to about 2ms in the STC testing. Each instance of RoLanD was interrupted by the system but not at the same time since they have been deployed on different ECUs. This made the voter of the not interrupted instance waiting for the voter of the interrupted instance (see differing system interrupts on Figure 66).
- 2) **Time Synchronization:** The timers of the cropping nodes within FAT_2 (representing the triggers for the cameras) have not been perfectly synchronized. The different starting times of the timer in $crop_1$ and $crop_2$ led to deviating forwarding times of the cropped images to RoLanD and RoLanD'. Thus, the instance of RoLanD which received the image first forwarded its calculation result earlier to its voter. This led to waiting times for the other voter (see Time Synchronization of Cameras on Figure 66).

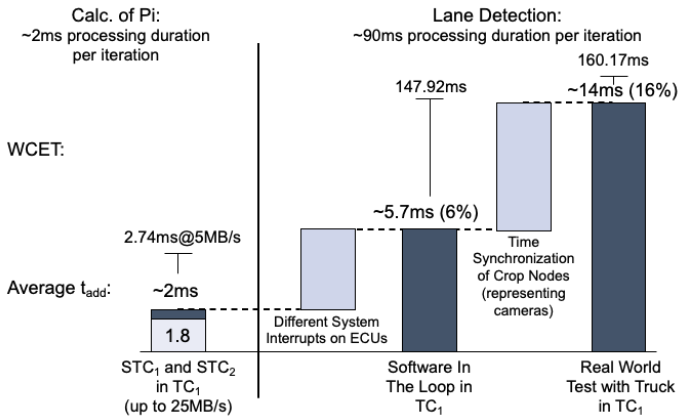


Figure 66: Comparison of voting duration between STC and FAT

Beside the difference of the voting durations, there were also differences in the FHTI visible. For the comparison the 5MB/s TC is selected, since its data rate is the closest the FAT₁ testing with 5.9MB/s (see chapter 5.3.1).

- 1) **Context Change:** The test results indicate, that the higher the size of the data required for a L4 application is, the longer it takes to reconfigure it. While for the STCs a simple program was used, this was different for FAT₁ and FAT₂. Here, a trained neural network was used which required more time to be loaded into the registers of the CPU and into the GPU after the process state was changed from pause to running (see “application process context change” on Figure 67).
- 2) **Buffer allocation:** RoLanD is designed in a way that the required buffer allocation for inference happens in the first iteration when a sample is received. This explicitly affects the faulty channel detection duration in TC_{2.2} and the reconfiguration duration in TC_{2.3}. Due to the on-demand activation, RoLanD'' needs significant more inference time than RoLanD and RoLanD'. Contrary to RoLanD'' those both are already running and don't require the buffer allocation of the first inference anymore. This leads to waiting times between the voters of RoLanD and RoLanD' for to RoLanD'' what was not the case within the STCs (see “GPU Buffer Allocation after first iteration” on Figure 67).
- 3) **Number of historic samples:** The longer context change duration and the buffer allocation duration of the FATs compared to the STCs required to increase the number of historic samples from 10 to 15. This led to longer loop durations for the FATs compared to the STCs (see “15 instead of 10 historic samples” on Figure 67).
- 4) **Shared Memory Communication:** The FHTI dropped from FAT₁ to FAT₂ by roughly 400ms (see Table 25). The reason for this drop is the implemented architecture changes. In fact, it showcases that the historic samples are pulled faster, when the effected nodes communicate with shared memory rather than Ethernet. (see “Architecture change: Shared memory instead of Ethernet for historic samples” on Figure 67).

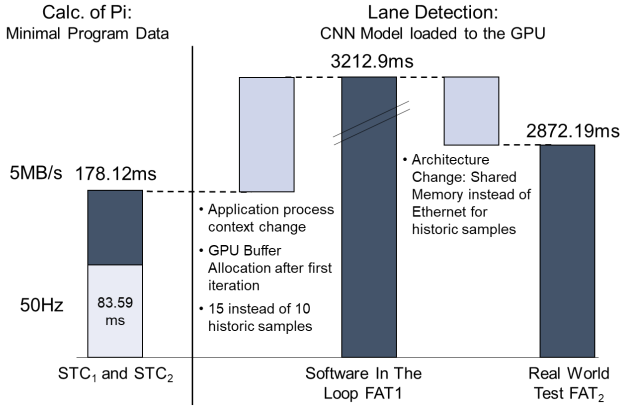


Figure 67: Comparison of FHTI between STC and FAT

For all above listed aspects optimization measurements have been identified. This is namely the usage of a hard real-time OS to cover longer calculation durations, the memory allocation during start-up of an instance of a L4 application rather than during the first sample iteration and finally the optimization of the time synchronization. However, the fact that the on-demand TMR system was able to realize FAT₁ and FAT₂ in addition to the STCs for all 1000 samples, it yet does not allow a statement regarding its performance compared to other technologies and for its usability in the real world.

6 Performance of FHTI and the Voting Duration

6.1 Performance Related to Comparable References

The analysis showed, that the FHTI (see chapter 2.5.3) was the part of the on-demand TMR which took the longest within both, the STCs and the FATs. To judge on the performance, references from other use-cases must be identified. Therefore, three comparable scenarios have been identified, namely, the sleep detection of a human driver in L2 and L3 systems, the reaction time of a human driver and the safety distance.

6.1.1 Sleep Detection

The first performance metric for the reconfiguration time is a comparison to existing attention tracking systems. The absence of a human driver is only allowed for L4 vehicles or higher according to the SAE J3016 standard. For all levels lower, the human still is the fallback of the system to get into a safe state in case of a malfunction.

The first L3 system was released by Mercedes-Benz in the year 2022 [99], allowing the driver to not pay attention to the driving scene anymore under the caveat that he can take back control within several seconds. Accordingly, the system will supervise if the driver is in the position to take over the driving task again. An unallowed behavior is to fall asleep or even only to close the eyes. If this happens, the system will warn the driver within the duration of 10 seconds [100].

This fact is even stronger for L2. Recent L2 driver assistant systems allow the drivers to permanently drive hands free, but they still are responsible in case

of a malfunction or a critical traffic situation. An example for such a L2 system is the Highway Assistant by BMW and the Bluecruise System of Ford, both released in 2023 allowing their customers hands free driving [101]. Again here, the attention of the driver is supervised and the system gives a notification in case the driver closes her or his eyes for a certain time. While no information was found for the Highway Assistant of BMW, the Bluecruise systems gives a notification after 5 seconds. After additional 5 seconds, the system will react with a light braking but it will never execute a minimal risk maneuver such as a stop in lane [102].

In the case of Ford with a released product, the manufacturer and the lawmaker (by approval of the system as a product) accepts a duration of ten seconds in which its system acts in unspecified conditions since per definition the driver always and immediately must take over the driving task. This duration is roughly three times as long as the total reconfiguration time of the on-demand TMR system, which had a maximal FHTI of 3212.9ms for the whole reconfiguration (see Table 25).

6.1.2 Reaction of a Human Driver

A driver must react to a variety of unforeseen events. In their analysis Zoellner et al. [103] showed, that the basic reaction time from the moment a driver spotted an obstacle to the moment he pushed the brake is lower than 1.03s for 98% of the tested humans. Additionally, the driver needs time to detect the obstacle. This time varies between 0.55s if the obstacle is in maximum 5° outside of the viewing direction and 0.7s if it is more than 5° outside of the viewing direction. As conclusion the human reaction time is about 1.73s during which the vehicle motion is not influenced by the driver.

In contrast, FAT₁ has the worst case FHTI of 3212.9ms (see Table 25) which is appx. double as high as the human reaction time. Without optimization measurements the human reaction time is superior to the on-demand TMR system.

However, a change of the deployment architecture can optimize the on-demand TMR system to close this limitation (see next chapter).

6.1.3 Safety Distance

A critical scenario, even for manually driven vehicles, is the event, that a preceding vehicle (Vehicle₁) executes an emergency brake until standstill, having a certain safety distance (Δs_{init}) to the rear vehicle (Vehicle₂). The rear vehicle must react properly so that both vehicles still have a gap between each other (Δs) once they are in standstill, thereby avoiding a crash (see Figure 68).

The braking process of the preceding is described by the following formula where (s) is the traveled distance, (a) the deceleration speed, (t) the elapsed time and (v_0) the initial speed at the moment the braking starts. It comes to a stop after t_1 seconds with a distance traveled of S_{v1} (see Figure 68).

$$(1) \quad s = \frac{1}{2}at^2 + v_0 * t$$

The movement of the rear vehicle is characterized by two phases. Until the driver reacts at t_r , the Vehicle₂ has traveled a distance of s_r . The distance-time relationship in this case is linear, represented by the following formula where (v) is the initial speed and (t) the elapsed time.

$$(2) \quad s = v * t$$

Once the driver hits the brake pedal at t_r , the movement is similar to the preceding vehicle until t_2 , so that vehicle₂ comes to a standstill at t_2 seconds with a distance of S_{v2} traveled (see Figure 68).

Consequently, the reaction maneuver of the rear vehicle was successful, when $\Delta s > 0$, meaning it stopped earlier than the preceding vehicle.

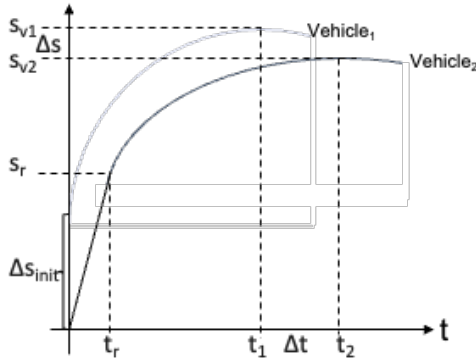


Figure 68: Distance-Time Diagram for decelerated motion

There are two factors which define the criticality of such a scenario. First, the safety distance towards the preceding vehicle (Δs_{init}). Second, the maximal possible brake deceleration which depends on three aspects, namely the weight force of the vehicle F_G , the friction coefficient μ_H and the mass of the vehicle m .

$$(3) \quad a = \frac{\mu_H F_G}{m}$$

The friction coefficient describes the relationship between the friction force and the normal force of two objects and is significantly influenced by the material types. Also, moisture or temperature makes an impact what is the reason why the friction coefficient on a dry road is higher than on a wet road.

For the example of an autonomous truck, the reaction time t_r represents the worst case FHTI of FAT_1 with 3212.9ms (see Table 25). It would make a commercial vehicle on a highway at 80km/h (22.2m/s) not responsive for a s_r of 71.3m (using formula (2)). If traveling on a high way, per law a safety distance (Δs_{init}) of in minimum 50m is required between a commercial vehicle and its preceding vehicle [105]. The General German Automotive Association (ADAC) [104] showed in an experiment, that a passenger car stops within 22m after a full brake and a commercial vehicle with a payload of 40t stops after 36.2m

from an initial speed of 80km/h. The safety distance of 50m (Δs_{init}) plus the braking distance bring the car to a stop at s_{v1} of 72m. Contrary the autonomous truck is reconfiguring until s_r of 71.03m and then would start to brake for 72m. Consequently, with a total s_{v2} of 143.03m a crash would be unavoidable since $\Delta s < 0$.

This scenario can be improved by initiating a comfort braking of the commercial AV with -3m/s^2 (vs. -6.8m/s^2 full brake in the experiment of ADAC [104]) in the moment the reconfiguration starts during the whole reconfiguration duration. Considering this braking the blind travel distance s_r can be reduced to 55.8m (using formula (1) where v_0 is the initial speed of 22.2m/s) at a remaining speed of 45.2km/h (12.6m/s) (using formula (4)).

$$(4) \quad v = a * t + v_0$$

Consequently, between the stopped car and the truck are 16.2m left, once the truck is operational again. If the truck then executes a full brake, it will come to a stop after 11.6m, making up a total s_{v2} of 67,4m (using formula (1) in (4) where $v=0\text{m/s}$ since the truck shall be in standstill and with an initial speed of $v_0=12.6\text{m/s}$).

$$(1) \text{ in } (4) \quad s = \frac{1}{2a} (v^2 - v_0^2)$$

Thus, a remaining distance Δs of 4.6m between the car and the truck is existing, a crash would not have happened (see Figure 69).

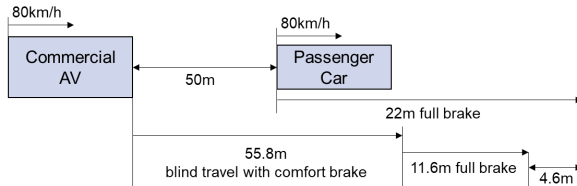


Figure 69: Full brake scenario of a passenger car

However, a fully blind travel time of 55.8 is still critical. For instance, a curve could come up to which the commercial AV would not react. Same as for the human reaction time, this limitation can be overcome by a change in the software mapping.

6.2 System Enhancing by Software Mapping

While the comparison of the on-demand TMR to recent sleep detection systems and the analysis of the braking distance in the previous chapter indicated a general feasibility of the on-demand TMR, the fact that the vehicle still travels blindly during the 3212.9ms worst case reconfiguration time is still critical. For instance, the vehicle would not change its steering angle or its speed outside of the comfort brake what potentially could lead to a risk.

To mitigate this risk, the MCU of the ECU can be utilized. While it cannot execute compute intensive calculations such as RoLanD itself, it still can execute simple safety critical tasks which are not compute intensive due to its higher ASIL-D rating. Accordingly, to cover the blind distance during reconfiguration, the sending of the real actuator commands using CAN should be moved from the high-performance SoC (which was the Orin) to the safety MCU (for instance the Aurix). Therefore, the AD software stack, (see Figure 70) can always forward a valid trajectory in form of a command queue to the safety MCU covering the time required for reconfiguration. This approach could be even further optimized by attaching a sensor like a radar to the Aurix. It would solve the corner case that a preceding vehicle does a hard brake during the reconfiguration of the AV.

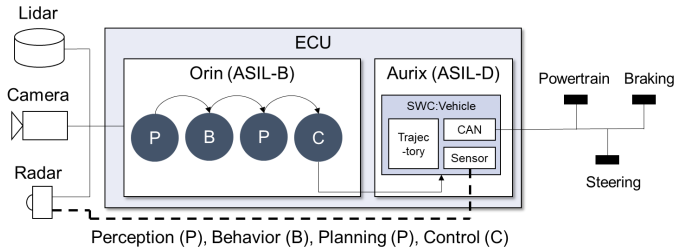


Figure 70: Optimized mapping of AD modules to ECU SoC and safety MCU

Accordingly, in the event of a fault the system would still be able to execute the last known correct trajectory and it would also be able to react in a minimal fashion to external events such as a preceding vehicle doing a full brake.

6.3 Performance Related to the State of the Art Technologies

The references in the previous chapter gave indications, that the on-demand TMR can be used on public roads. However, those comparisons did not provide insight into the performance of the on-demand TMR compared to existing technologies, as identified in the state of the art (see chapter 3.2).

Fault Detection (TC_1 , TC_3 , TC_4): While the average voting duration in nominal mode of the TMR was around 2ms for STC_1 and for STC_2 within 5MB/s and 10MB/s it was around 6ms for FAT_1 and 14ms for FAT_2 . If comparing the voting duration of STC_1 and STC_2 with calculation durations of around 2ms the overhead caused by the distributed voters is roughly 100% for the STCs. However, if the calculation duration is around 90ms such as in FAT_1 , the overhead is reduced to appx. 6% for FAT_1 and 15% for FAT_2 . Since the higher voting durations of FAT_2 are caused by an unoptimized deployment architecture (see chapter 5.3.4), the more reasonable value is FAT_1 . The 6% overhead could be reduced by further optimizing the system e.g. by using a hard real-time operating system to minimize the influence of scheduling (see chapter 5.3.3) or

using PCIe instead of Ethernet to minimize the influence of the network (see chapter 5.2.3).

In his work for adaptive CPU lockstep Kempf et al. [90] was able to execute fault detection with a lockstep mechanism creating an overhead between 1.46% to 5.28%, depending on the application (see chapter 3.2.3). Relatively to those numbers the on-demand TMR is competitive, even for low latency usage if the optimization mechanisms are applied. Furthermore, the CPU lockstep technology is not able to detect faults on the GPU which is an unique advantage of the proposed on-demand TMR system.

Identification of faulty channel ($TC_{2.1}$): The on-demand population of a third channel for fault tolerance purposes is a novelty of this work. Out of this reason no direct comparison work was found.

Reconfiguration ($TC_{2.3}$): The longest reconfiguration duration of the technologies in the analysis of research and science was visible for the orchestrated container. Here not only an instance of an application is restarted but also a whole OS (see chapter 2.6.3). This overhead leads to up to 70s reconfiguration time [88] what is too long for safety critical applications (see chapter 3.2.2).

The shortest reconfiguration times were possible with the DDS Ownership Strength experiment. However, this is a hot standby pattern which has per-se the lowest reconfiguration time. Even being in hot standby, still the on-demand TMR has comparable reconfiguration times for a subset of the cases, namely STC_1 for 50Hz with 45.52ms. This is ~ 2.5 times higher than the average reconfiguration time of DDS with 16.8ms and only slightly higher than its max value with 41.5ms in the experiment of Kugele et al. [85] (see chapter 3.2.2).

Contrary to DDS-Ownership strength, the SOME/IP experiment of Oszwald et al. [87] was not operating in hot standby. The application was most likely a little more resource consuming than the calculation of the number PI as in the on-demand TMR STC testing. With 147ms, still the reconfiguration

performance of SOME/IP was lower compared to all STC₁ and for 5MB/s (84.59ms) and 10MB/s (128.17ms) in STC₂ (see chapter 3.2.2).

The reconfiguration duration of ROS2 Managed Node in the supervised master thesis of Mathew [95] was closely to the reconfiguration duration of FAT₁. However, again here, the chosen application, which was a ROS2 node only subscribing to a small set of data and publishing it to the next node without further interaction, is not as resource intensive as RoLand.

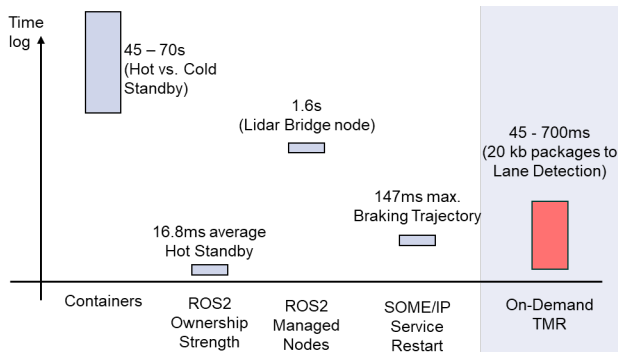


Figure 71: On-demand TMR performance comparison to the state of the art

The indicated comparison provides a high-level orientation for the performance of the on-demand TMR (see Table 15, Table 17 and Table 25) compared to other reconfiguration technologies (see Figure 71). However, for a precise comparison also the different system set-ups in terms of used ECU or network technology should be considered.

7 Conclusion and Outlook

7.1 Test Case and Requirement Compliance

The analysis of the system testing data in conjunction with the performance evaluation allow a judgement regarding the level of fulfillment of the defined test cases (see chapter 4.4).

7.1.1 System Test Acceptance

TC₁: *“The distributed voting mechanism must validate all correct iterations during the nominal state by using the voters.”*

The system was able to validate all correct iterations during the nominal state for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂.

TC_{2.1}: *“The voters must identify two different hashes and initiate the identification process of the faulty channel by activating V_{bu} ”.*

The system was able to detect the faulty sample for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂.

TC_{2.2}: *“ V_{bu} must identify the faulty channel. The faulty channel must stop working while the correct channel continues.”*

The system was able to identify the faulty channel for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂.

TC_{2.3}: *“The system must continue to operate with the correct channel together with the reconfigured channel₃.”*

The system was able to activate the faulty instance on channel₃ by reconfiguring the system for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂.

TC₃: *“The system must continue in the reconfigured deployment. During this duration, the distributed voting mechanism must validate all correct iterations.”*

The system was able to validate all correct iterations during the reconfigured state for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂.

TC₄: *“The system must restart the faulty application. After the restart, the temporarily running application from the reconfigured state must hand over to the restarted application. After this hand over the system must continue to validate all correct iterations.”*

The system was able to phase out the reconfigured instance of the application Pi”, to phase in the restarted instance of the application Pi and to validate all correct iterations during the recovered state for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂.

STC Integration: *“1000 samples have been processed by the system iteratively. The transitions between the system states nominal, on-demand TMR, reconfigured and recovered must have happened without error.”*

During the STCs 1000 samples have been processed for all update rates (5Hz to 50Hz) in STC₁ and all data rates (5MB/s to 40MB/s) in STC₂. The transition between the system states happened without error.

As conclusion, all TCs and their integration into the STCs have been achieved, demonstrating the general feasibility of the system with the STCs.

At the same time, the results also showed, that the critical factor for performance is the ROS2 configuration *lease duration*, the number of historic samples for the FHTI and the overall traffic on the Ethernet.

7.1.2 L4 Lane Detection Test Acceptance

TC₁, TC₃, TC₄: *“The average voting duration t_{add} should not exceed an acceptable overhead. The WCET execution time should be acceptable. If t_{add} or the WCET is not acceptable, countermeasures must be identified.”*

The average voting durations t_{add} within TC₁ created an overhead of 6% for the FAT₁ testing compared to an alternative CPU lockstep technology which creates between 1.46% to 5.28% (see previous chapter). Considering that, contrary to the CPU lockstep technology, the L4 application of the on-demand TMR system utilizes the GPU and that the upper range of the CPU lockstep is close to the overhead of the on-demand TMR system, the created overhead of the on-demand TMR can be seen as acceptable high during nominal state.

The fact that the overhead increased up to 16% for the reconfigured state of both STCs and FATs puts this statement into a different perspective (see chapter 5.3.4). This also applies to the WCET which was measured with 234.7ms in TC₄ of FAT₂. The increasing overhead and the outliers are not acceptable for a production ready system.

However, the reasons for the increasing average voting durations within TC₃ and TC₄ as well as for the WCET with 234.7ms have been identified. This was namely the ROS2 lease duration which triples the Ethernet traffic during reconfigured and parts of the recovered state (see chapter 5.2.3) and the quality of the time synchronization (see chapter 5.3.4). If those countermeasures are applied, the voting durations and WCET would be the same range as in TC₁.

Furthermore, the average voting durations of the FAT testing could be also reduced to the values of the STCs by reducing the waiting times of the voters. They are caused due to system interrupts on either ECU due to the relatively long calculation duration of 90ms in average for RoLandD (see chapter 5.3.3). If a hard real time operating system is used, those interrupts would be avoidable, bringing down the average voting durations close to the STCs of 2ms, thereby reducing the overhead close to 2%.

As result this acceptance criteria is only partially fulfilled since the overhead is only acceptable for TC_1 and FAT_1 . Countermeasures for all other states have been identified.

TC_2 : *“The FHTI should be acceptable long. If it is not, countermeasures must be defined.”*

The worst case FHTI was 3212.9ms in FAT_2 . The FHTI interval references showed, that the on-demand TMR is within the range of existing driver attention detection systems as applied in L2 systems which allow hands-free driving. At the same time, it also showed, that the human reaction time with about 1.6s is still superior to the system. Also, a blind travel duration of 55.8m during reconfiguration is still critical due to curvature or unexpected situations on the road (see chapter 6.1 and chapter 6.2).

Again here, countermeasures for optimization have been identified. The usage of either a 10Gbit/s Ethernet or PCIe would already drastically improve the situation since it would give significantly more bandwidth. This would allow to pull the historic samples way quicker (see chapter 5.2.3). Second, the number of historic samples should be optimized to avoid that unnecessary historic samples are pulled through the limited bandwidth of the networking technology (see chapter 5.2.6). Third, the structure of the L4 application also matters. In fact, in the chosen example the memory allocation happened during runtime when the first sample is received. This could be improved by allocating the memory directly after the start-up of the application before it receives the first sample (see chapter 5.3.3).

Lastly, the optimization of the deployment architecture allows the system to still react on changes in the environment while reconfiguring (see chapter 6.2).

Again here, the acceptance criteria is only partially fulfilled since the FHTI is within the range of the driver attention reaction time but still needs improvement for a safe operation.

FAT Integration: *“1000 samples have been processed by the system iteratively. The transitions between the system states nominal, on-demand TMR, reconfigured and recovered must have happened without error.”*

All 1000 samples have been processed. The transition between the states was possible without error.

As conclusion, all TCs within both FATs have been successfully executed. However, the performance of the system requires further improvement if intended to be used in a production system.

7.1.3 Compliance to Requirements

In summary, the architecture requirements have been realized by using technologies as required in future vehicles. This is namely the realization of a SOA using Ethernet and a POSIX Operating System. For the realization, automotive grade components have been used, such as the state-of-the-art NVIDIA AGX ECU with the Orin as SoC, 1G Ethernet, ROS2 and an Ubuntu Linux RT.

It has been showcased, that the system can detect faults. However, this fault detection mechanism creates overhead and has outliers regarding its voting durations, only partially fulfilling SR2 (real-time). It was demonstrated that a fault can be isolated by the system, since the faulty instance of the application was shut down externally by the voter, rather than shutting itself down. The reconfiguration mechanism allowed to mask a fault.

However, again here, the FHTI was only partially fulfilled. With a duration of about 3212.9ms it would lead to a blind travel distance of roughly 52m if the vehicle brakes while reconfiguring (see chapter 6.1.3). Still those 52m are critical but could be realized if additional countermeasures are applied (see chapter 6.2). The transition into the recovered state allows the system to be able to cover another consecutive fault.

The general feasibility of the data transportation requirement was fulfilled. However, it was shown that the bandwidth is a critical element and could be improved to 10Gbit/s or PCIe with even higher data rates. Furthermore, the system was able to process data as required in modern L4 applications. This affects namely the usage of a GPU for which no lockstep processor is existing yet. Lastly, it was shown that the on-demand TMR can be embedded into a L4 Software stack since it was realized as ROS2 module. This ROS2 module can be easily included into the AD Software stack, which is also realized as a set of ROS2 modules (see chapter 3.1.3).

Areas of Requirements	Requirements	Comment
Architecture Requirements	AR1: Future Vehicles	SOA, Ethernet, POSIX OS
	AR2: Automotive Grade	NVIDIA AGX, 1G Ethernet, ROS2, Linux RT
	AR3: Minimal Components	Generic Back-Up ECU enables minimal 3 rd channel
Safety Requirements	SR1: Fault Detection	Shown in system, bench and truck testing
	SR2: Real-Time	For GPU L4 Applications partially acceptable due to voting duration outliers
	SR3: Fault Isolation	Shown by shut down of application by voter
	SR4: Fault Masking	Shown by reconfiguration in system, bench and truck testing
	SR5: Fault Handling Time	3.2s reconfiguration leads to 52m blind travel
	SR6: Multiple Faults	Shown by recovery mechanism of faulty channel
AV Specific Requirements	AVR1: Data Transportation	Shown by 1G Ethernet, however, represents limitation
	AVR2: Data Processing	Shown by application which utilizes the GPU of a SoC
	AVR3: AV Structure	ROS2 and POSIX allow to use AV Software stacks with minimal adjustments

Figure 72: Compliance to requirements

7.2 Summary and Limitations

At the beginning of this dissertation five research questions have been asked (see chapter 1.4). Those research questions represented the input to the requirements of the on-demand TMR system (see chapter 4.1.1). Based on those requirements the system design was executed as well as the component design and the implementation. To conclude about the capabilities of the proposed system, component test cases were defined and executed together with system test cases and the final acceptance tests (see chapter 4.4).

The first research question asked how a safe architecture for an AV may look like. SOAs have been introduced as design pattern for future software driven vehicles (see chapter 2.4). Furthermore, it was analyzed how currently safe E/E Architectures are realized, in automotive as well as in avionics (see chapter 3.2). The limitations of current safe architectures have been identified what is in summary the low performance of safety MCUs for the CPU and the complete absence of a GPU (see chapter 3.4). The research question was answered by proposing a SOA in combination with a save architecture pattern from avionic (see chapter 4.1). The safety of the on-demand TMR system was achieved by redundantly operating the L4 applications on multiple ECUs following the ISO 26262 decomposition principle (see chapter 2.5.3). The output of the ECUs was validated by voters (see chapter 4.2), which also have been distributed across multiple ECUs, thereby achieving a similar safety level for random hardware faults as in traditional safety architectures. The usage of automotive SoCs which have a higher CPU performance and also contain a GPU allows the on-demand TMR system to safely operate recent L4 applications.

The second research question addressed the availability of the system. Since modern automotive systems still have the human driver as fallback in place the system availability yet does not play a significant role. Contrary, the most quoted example for highly available systems are airplanes. Accordingly, avionic architectures have been analyzed (see chapter 3.3) and – as described in

the previous paragraph - used as orientation for the design of the safe architecture. In fact, the adaptation of the Quadruplex architecture was proposed since it provides the baseline of high availability and could be adjusted to reduce the required number of redundant components (see chapter 4.1.3).

This minimization of required components was part of the third research question, generally aiming towards the limitations of automotive vehicles which are namely restrictions within weight, dimensions, power consumption and cost. Avionic architectures, also the selected Quadruplex architecture, suffer from a high number of redundant components. The sole realization of the Quadruplex architecture would exceed the mentioned cost targets, power budget and spacial limitations of automotive vehicles (see chapter 3.4). Out of this reason it cannot just be taken over but required changes. To overcome this limitation the possibility to start, pause or shut down applications of a modern middleware as core element of a SOA was utilized. Therefore, the Quadruplex architecture was adjusted and down stripped to the safety pattern of the TMR. However, contrary to the traditional TMR, which is using static redundancy, only two channels were acting continuously together (see chapter 4.2.1). The third channel was only activated in case the other both channels detected a fault. Therefore, it was on-demand made available and could be used for any L4 application in the whole system (see chapter 4.2.3). Consequently, it was possible to realize a highly available architecture but with less components than existing solutions such as the traditional TMR.

The fourth research question was about the challenge to realize such an architecture by combining existing technologies with AV specific needs. This question was mainly addressed by two instances: First, the usage of a modern middleware which can easily be used to realize the software architecture of AVs (see chapter 3.1.2). Second, the need of high performance SoCs also containing a GPU by L4 applications was solved by the redundant operation of the apps using the ISO 26262 decomposition principle (see chapter 2.5.3), thereby overcoming the problem that MCUs are not powerful enough and

even worse, don't have specific hardware accelerators. The research question was answered by selecting and combining components, which are already available on automotive certified level, such as ECUs, Ethernet or POSIX based operating systems (see chapter 5.2.1 and chapter 5.3.1).

Since avionic architectures are regularly mentioned as role model for AVs, the last research question asked if elements can be taken over. As mentioned, the on-demand TMR system was realized as modified avionic Quadruplex architecture (see chapter 4.1.3). The prototypical realization and its performance evaluation showed, that the on-demand TMR system is promising to enhance the safety for automotive architectures (see chapter 6). These promising results answered the last research question by indicating the general feasibility of the adjusted Quadruplex architecture for automotive purposes.

It can be concluded that all raised research questions have been addressed. Also, it was indicated by the prototypical realization that the proposed system is a promising option as safe architecture for future AVs.

However, the system is facing several limitations which require a critical discussion.

Limited consideration of ISO 26262:

In this work, the random hardware fault detection of ISO 26262 was considered, also following a minimal V-Shape process. However, the scope of the ISO 26262 is wider. One of the missing elements which has not been considered as part of this dissertation is the treatment of systematic faults. The standard does not per-se forbid the use of the exact same software and exact same hardware for the decomposition as applied in the proposed system. However, a fault analysis would have to be done which was not part of this work. Additionally, several requirements towards the software are addressed in the standard, such as the usage of a proven subset of a coding language. An example would be the Motor Industry Software Reliability Association

(MISRA) subset of C++. The usage of such a standard would reduce the hurdle to apply the system to real automotive use.

Hardware single point of failure:

The redundancy concept of the proposed system only works when all required components are redundant. Accordingly, this also must be the case for the switch. The switch as single point of failure could be overcome by the usage of a 10G ring topology where UDP or TCP packages could travel in both directions to a destination. In case there is a fault in one direction (e.g. a defect cable), the packages still would arrive at the destination by travelling into the other direction.

Data corruption:

A random hardware fault could not only occur during the calculation process on an ECU. Another fault source is the data transportation over the network technology. In the case of this experiment corrupted data would be detected since the data is published separately to both redundant apps, thereby causing a fault in the voters. However, this duplication of the data also consumes the double amount of bandwidth, which is one of the main limiting aspects of the system. An alternative to the redundant sending of data would be the usage of checksums. By doing so corruption could be detected by both applications via a checksum thereby not requiring two independent data streams.

No consideration of stateful applications:

The introduced system yet only works for stateless applications. However, some applications such as probabilistic localization algorithms (see appendix 8.4.3) or recurrent neural networks (see appendix 8.5) are stateful applications. A stateful application requires its own output of a calculation iteration as input for the next calculation iteration (see Figure 28). In other words, the output of an of a stateful application depends on previous, historic input. In contrast, when an L4 application is on-demand started, it has not executed

those previous iterations, thereby not being in the same state as the other L4 apps in channel₁ and channel₂. The extension of the system to stateful applications would make it suitable for even more L4 applications.

Statistic relevance:

The low number of 1000 samples per testing is not allowing a hard statistic relevant statement. This is even intensified by the fact, that the testings of each test case have been executed only once. Additionally, the system was integrated into only one truck with one test execution. To get more reliable data, the tests must be repeated several times on several trucks over a longer duration. However, the data set still allows an early indication about the usage of the on-demand TMR system.

7.3 Need for Research

As mentioned in the limitations, the current system proposal is realized on a fully homogenous design using the state-of-the-art NVIDIA Orin. Those homogenous designs are vulnerable towards systematic faults. In fact, three different SoCs should be selected instead of only one. Also, the operating system itself as well as the ROS2 implementation should be heterogenous, requiring further research how the on-demand TMR system behaves in a heterogenous set up.

Furthermore, the proposed on-demand TMR was applied with a simple CPU only application which calculated the number Pi. Additionally, it was applied to a lane detection algorithm using a CNN as DL technology. Both applications don't require a context. In fact, the CNN is only processing the data forward through the network. There are no recursion loops done, where the output of one layer is the input for the next layer. Similar recursion loops are also required for non-DL based applications such as a Kalman filter.

Both, Kalman filters and context requiring DL applications are already used in L4 AD software stacks. However, the trend within DL technologies is even intensifying this fact. An example is the area of transformer-based AI which is utilized heavily within the modern research area of generative AI. As an example, generative AI will be used in future behavior modules of an AD software stack since it has the capability to generate potential future driving scenes. Thereby it improves the capability of future prediction modules.

The current realization of the on-demand TMR is not able to cope with context requiring applications. Future research could close this limitation by approaching it from two sides. First, the third instance of the application on the ECU_{back_up} could always stay in context by transferring the context after each iteration from both operating instances to the on-demand instance without executing the application itself. The second possible solution could be to use the historic samples to loop the third application on-demand into the context of the other both instances. Both proposals or other solutions need to be investigated to extend the usability of the on-demand TMR system.

The other wider research area which was identified during this dissertation is the usage of Ethernet itself as base technology for SOAs. The analysis of the result showed that Ethernet is creating overhead since it requires to pass all layers from the OSI reference model to build out connections between the ROS2 nodes. Additionally, for each published item still multiple layers from the OSI reference model must be passed since the underlying technology is UDP, TCP and IP.

In contrast, PCIe as network technology is based on shared memory, thereby only affecting the lower layers of the OSI reference model. This creates less overhead compared to Ethernet. Additionally, while the usage of automotive Ethernet is limited to 10Gb/s (appx 1.25 GB/s), a generation 6 PCIe connection using 4 lanes can achieve up to 32GB/s [106]. This significant higher bandwidth would close the limitation within the on-demand TMR for high data rates such as the 25MB/s and the 40MB/s case.

Those findings allow the question why only Ethernet is mentioned in the context of modern SOAs in current research and science. PCIe is outperforming Ethernet in terms of bandwidth and overhead, thereby promoting it as capable alternative. This opens two potential additional research areas. First, the general question about the usability of PCIe for modern SOAs. Second, the proposed on-demand TMR should be analyzed regarding its behavior using PCIe instead of Ethernet.

8 Appendix

8.1 Publications

8.1.1 Journals and Conferences

[P1] Henle, J., Stoffel, M., Schindewolf, M. Naegele, A., Sax, E., „Architecture platforms for future vehicles: a comparison of ROS2 and Adaptive AUTOSAR”, 2022, “25th IEEE International Conference on Intelligent Transportation Systems (ITSC)”, Macao, 08 – 12th October 2022

[P2] Stoffel, M., Sax, E. „Distributed Voters for Automotive Applications”, IEEE Intelligent Vehicles Symposium (IV), Anchorage, Alaska, 04-07th June 2023

[P3] Stoffel, M.; Schindewolf, M., Sax, E. „On-Demand Triple Modular Redundancy for Automotive Applications“, IEEE International Systems Conference (SysCon), Montreal, Canada, 15-18th April 2024

[P4] Gupta, S., Stoffel, M., Agh, H., Sax, E., Wagner, S. „Dual Image Cropping Algorithm for Enabling Redundant ASIL-D Safe Lane Detection for Autonomous Vehicles” submitted to IEEE 27th International Conference on Intelligent Transportation Systems, 24-27th September 2024, Edmonton, Canada

8.1.2 Patents

[PT1] Stoffel, M. „Redundante Prozessorarchitektur für sicherheitskritische Anwendungen“, DE: 10 2022 129 939.2, Daimler Truck AG

[PT2] Stoffel, M., Gupta, S. „Safety decomposition using redundant field of view of multiple sensors”, Torc Robotics Inc.

8.2 Test Results

8.2.1 System Test

Nominal State:

STC₁ Varying update rate: All 50 iterations have been validated correctly by the voters with an average voting duration t_{add} between 1.92ms for 33Hz and 2.12ms for 5Hz and with a standard deviation of 0.14ms for 10Hz to 0.25ms for 10Hz. The highest voting duration was observed at 10Hz with 2.75ms, the lowest at 1.16ms at 50Hz.

The voting duration t_{add} (see chapter 4.4.1) is divided into the voting time and the network time which is needed to move the hashes from one voter to the other using Ethernet. While the average for the voting time varies from 0.21ms for 33Hz to 0.24ms for 5Hz the network transmission time varies from 1.7ms to 1.88ms (see summary in Table 12).

1MB/s const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
5 Hz	t_{Voting}	0.24	0.24	0.03	0.03	0.2	0.2	0.39	0.4
	t_{Network}	1.87	1.88	0.15	0.15	1.13	1.35	2.18	2.33
	t_{Add}	2.11	2.12	0.18	0.19	1.33	1.54	2.57	2.73
10 Hz	t_{Voting}	0.25	0.22	0.08	0.04	0.18	0.19	0.56	0.4
	t_{Network}	1.79	1.8	0.17	0.1	1.34	1.54	2.19	2.16
	t_{Add}	2.04	2.02	0.25	0.14	1.52	1.72	2.75	2.55
33 Hz	t_{Voting}	0.21	0.22	0.02	0.04	0.18	0.18	0.3	0.39
	t_{Network}	1.73	1.7	0.14	0.19	1.14	1.28	2.14	2.07
	t_{Add}	1.93	1.92	0.17	0.22	1.45	1.46	2.43	2.46
50 Hz	t_{Voting}	0.21	0.22	0.02	0.04	0.19	0.18	0.27	0.44
	t_{Network}	1.77	1.75	0.18	0.15	0.98	1.37	2.11	2.25
	t_{Add}	1.98	1.98	0.2	0.19	1.16	1.55	2.38	2.69

Table 12: Voting durations for STC₁ for TC₁ in ms

STC₂ Varying data rate: All 50 iterations have been validated correctly. The average voting duration t_{add} varied between 2.01ms at 10MB/s to 31.75ms for 40MB/s with a standard deviation of 0.17ms at 5MB/s and 10MB/s to 26.86ms for 40MB/s. The highest voting duration was observed with at 40MB/s with 101.57ms, the lowest at 1ms also at 40MB/s.

The average time for the pure voting varied from 0.24ms at 10MB/s to 30.19ms at 40MB/s with a standard deviation from 0.03ms at 10MB/s to 26.3ms at 40MB/s. The average time to transmit a hash varied from 1.39ms at 40MB/s to 1.88ms at 25MB/s (for summary results see Table 13).

20hz const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
5 mb	t_{voting}	0.26	0.26	0.05	0.06	0.21	0.21	0.44	0.54
	$t_{Network}$	1.86	1.86	0.12	0.12	1.39	1.45	2.20	2.20
	t_{Add}	2.12	2.12	0.17	0.18	1.59	1.66	2.64	2.74
10 mb	t_{voting}	0.24	0.24	0.04	0.03	0.21	0.21	0.38	0.35
	$t_{Network}$	1.79	1.77	0.13	0.16	1.43	1.20	2.00	2.01
	t_{Add}	2.03	2.01	0.17	0.19	1.64	1.41	2.38	2.36
25 mb	t_{voting}	0.39	0.27	0.37	0.07	0.22	0.21	1.82	0.52
	$t_{Network}$	1.88	1.85	0.11	0.25	1.52	1.24	2.24	2.17
	t_{Add}	2.27	2.12	0.48	0.32	1.74	1.44	4.07	2.69
40 mb	t_{voting}	1.47	30.19	2.15	26.30	0.21	0.21	8.71	96.72
	$t_{Network}$	1.39	1.56	0.29	0.56	0.80	1.11	2.14	4.85
	t_{Add}	2.86	31.75	2.45	26.86	1.00	1.32	10.85	101.57

Table 13: Voting durations for STC₂ for TC₁ in ms

On-demand TMR:

STC₁ and STC₂ for TC_{2.1}: For both system test cases the fault leading to varying hashes was detected in iteration A_{51} . This detected fault first initiated the mechanism to identify if channel₁ or channel₂ was faulty (TC_{2.2}). After that, channel₃ reconfigured to mimic the faulty channel, thereby keeping the system operational (TC_{2.3}).

STC₁ Varying Update Rate for TC_{2.2}: The detection of the faulty channel required sequential steps (see Figure 34). The duration to activate the back-up Pi'' varied from 9.31ms at 50Hz to 12.95ms at 5Hz. The duration until Pi'' received the first historic sample after its activation varied from 11.64ms at 33Hz to 16.51ms at 5Hz. The loop through the 10 historic samples required a minimum of 2.8ms at 50Hz to 42.07ms at 5Hz. The execution of Pi'' required 2.11ms at 10Hz in a minimum to 2.19ms in a maximum. The creation of the MD5 hash varied from 0.99ms at 50Hz to 1.11ms at 10Hz. The return of the MD5 hash to the back-up voter and the identification of the faulty channel required 7.04ms at 33Hz in a minimum to 7.29ms at 5Hz in a maximum. The remaining parts of the voter such as receiving the MD5 hash and answering the result to the calling voter V₁ and V₂ varied from 0.24ms at 33Hz to 0.28ms at 10Hz (see summary in Table 14).

1MB/s Const	5Hz		10Hz		33Hz		50Hz	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
$t_{context}$	Wait	12.95	Wait	10.14	12.25	Wait	Wait	9.31
$t_{call-back}$		16.51		13.64	11.64			11.84
t_{loop}		42.07		18.76	4.40			2.80
t_{calc}		2.19		2.11	2.17			2.17
t_{hash}		1.06		1.11	1.04			0.99
t_{third}		7.29		7.21	7.04			7.15
t_{voter_bu}		0.25		0.28	0.24			0.27
t_{third_hash}		84.59		53.25	38.78	41.22		34.53
t_{voting}	0.24	0.31	0.23	0.2	0.22	0.28	0.21	0.24
$t_{Network}$	3.46	2.81	3.42	3.57	3.32	2.81	3.63	3.3
t_{add_51}	88.28	85.45	59.17	57.02	42.33	44.31	40.62	38.07

Table 14: Durations for identification of faulty channel for STC₁ for TC_{2.2} in ms

In consequence, the time until the third hash was created, and the wrong channel was identified varied from 82.33ms at 5Hz to 34.53ms at 50Hz. The voting time to identify that A₅₁ was wrong took 0.2ms at 10Hz in minimum to 0.31ms at 5Hz in maximum. The network time to transport the hash varied from 2.81ms at 5Hz and at 33Hz to 3.63ms at 50Hz. Those steps made up a total faulty channel detection duration t_{add_51} between 38.07ms for 50Hz to 85.45ms at 5Hz to identify that Pi contains the random hardware fault. This included the command from V_{bu} to V₁ to shut down the process of Pi.

STC₁ Varying Update Rate for TC_{2,3}: Since V_{bu} identified that P_i contained the random hardware fault, it then initiated the reconfiguration of P_i on ECU_{back_up} to proceed with the next iteration after the faulty iteration. In the case of this experiment this was A_{52} (see $t_{activate}$ and t_{loop_start} in Figure 34).

Equal to the on-demand activation process of A_{51} also for A_{52} the faulty P_i must be activated first, representing a context switch on the CPU. This process took between 38.11ms in the case of 33Hz up to 41.64ms in the case of 50Hz. After its activation P_i'' looped through the ten historic samples (see t_{loop_start} on Figure 34) until S_{52} . This process required 72.79ms for 5Hz to 1.88ms for 50Hz. The calculation time for A''_{52} was between 1.97ms for 10Hz to 2.12ms at 5Hz so that in total the reconfiguration duration t_{add_52} varied between 42.1ms for 33Hz to 114.93ms at 5Hz.

The faulty channel detection duration and the reconfiguration duration together make up the FHTI. For STC₁ this varied between 83.59ms to 200.38ms (for summary see Table 15).

	5Hz	10Hz	33Hz	50Hz
$t_{activate}$	40.02	38.4	38.11	41.64
t_{loop_start}	72.79	17.22	1.97	1.88
t_{calc_r}	2.12	1.97	2.02	2.0
t_{add_52}	114.93	57.59	42.1	45.52
FHTI	200.38	114.61	84.43	83.59

Table 15: Durations for reconfiguration of App'' for STC₁ for TC_{2,3} in ms

STC₂: Varying Data-Rate for TC_{2,2}: The time to activate the waiting P_i'' was within a range of 9.27ms at 25MB/s to 10.26ms for 40MB/s. After P_i'' was activated and dispatched to the CPU, it required additional 17.87ms in a minimum for 5MB/s and in maximum 351.88ms for 40MB/s until the first historic sample was received. The time to loop through the ten historic samples varied from 50.94ms at 5MB/s to 351.88ms at 40MB/s.

It required between 2.23ms at 40MB/s to 2.4ms at 5MB/s to execute P_i'' as iteration A''_{51} with the historic sample S_{51} as input. The third hash was generated within 1.05ms at 5MB/s to 3.58ms at 40MB/s. The return of the third

hash to the back-up voter as well as the identification of the faulty channel required between 6.98ms at 5MB/s to 7.61ms at 40MB/s. The time which V_{bu} required to receive the hashes from V_1 and V_2 as well as the time to answer the voting result to V_1 and V_2 varied between 0.26ms at 25MB/s to 0.35ms at 10MB/s.

In total, the time required to generate the third hash and to respond the V_1 and V_2 if they are faulty or not varied from 89.41ms at 5MB/s to 872.05ms at 40MB/s. Adding the initial voting time which was needed to identify that both hashes are not identical of 0.21ms at 10MB/s to 29.21ms for 40MB/s as well as a network time to transfer the hash of between 2.24ms at 40MB/s to 4.06ms at 25MB/s the total faulty channel detection duration for Sample₅₁ varied between 93.53ms at 5MB/s to 901.52ms at 40MB/s.

20Hz const	5MB/s		10MB/s		25MB/s		40MB/s	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
$t_{context}$	9.89	Wait	9.78	Wait	9.27	Wait	Wait	10.26
$t_{call-back}$	17.87		23.92		43.47			496.49
t_{loop}	50.94		107.89		276.44			351.88
t_{calc}	2.40		2.25		2.30			2.23
t_{hash}	1.05		1.36		1.54			3.58
t_{ident}	6.98		7.25		7.20			7.61
t_{voter_bu}	0.29		0.35		0.26			0.26
t_{third_hash}	89.41	91.87	152.8	155.22	340.49	342.91	874.95	872.05
t_{voting}	0.31	0.28	0.22	0.21	0.3	0.28	0.23	29.21
$t_{network}$	3.81	3.04	3.81	2.84	4.06	3.34	2.24	2.85
t_{add_51}	93.53	95.19	156.83	158.28	344.85	346.52	875.18	901.52

Table 16: Durations for identification of faulty channel for STC₂ for TC_{2.2} in ms

STC₂: Varying Data-Rate for TC_{2.3}: Again here, after the detection of the faulty channel at Sample₅₁ the faulty Pi was activated on the back-up ECU to take over the task of the faulty application from Sample₅₂ on for the following. The duration until the Pi'' was dispatched to the CPU and received the first call-back for the historic samples varied between 39.35ms at 5MB/s to 49.58ms at 40MB/s (see $t_{activate}$ on Figure 34). It then looped through the historic samples until it reached S_{52} what took 43.07ms for 5MB/s up to 642.9ms for 40MB/s (see t_{loop_start} on Figure 34). The calculation time to process S_{52} varied

from 2.17ms at 5MB/s to 3.31ms at 40MB/s so that the total reconfiguration duration was between 84.59ms at 5MB/s up to 695.78ms at 40MB/s.

The FHTI for STC₂ varied between 178.12ms to 1600.15ms (for summary see Table 17).

	5MB/s	10MB/s	25MB/s	40MB/s
t_{activate}	39.35	45.62	42.37	49.58
$t_{\text{loop_start}}$	43.07	80.19	250.63	642.9
$t_{\text{calc_r}}$	2.17	2.35	2.92	3.31
$t_{\text{add_52}}$	84.59	128.17	295.93	695.78
FHTI	178.12	285	640.78	1600.15

Table 17: Durations for reconfiguration of App'' for STC₂ for TC_{2,3} in ms

Reconfigured State:

After the faulty channel was identified and reconfigured, the system continued to operate in reconfigured state, using ECU₂ and ECU_{back_up} while Pi on ECU₁ was being recovered.

STC₁ Varying Update Rate: The average additional time t_{add} for voting during the temporary execution of Pi'' on ECU_{back_up} varied between 2.17ms at 50Hz to 3.97ms for 5Hz with a standard deviation from 0.27ms at 5Hz to 0.58ms at 10Hz. The maximal value was observable at 5Hz with 4.45ms.

1MB/s const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
5 Hz	t_{Voting}	0.38	2.04	0.10	0.18	0.22	1.74	0.53	2.33
	t_{Network}	1.83	1.93	0.17	0.12	1.31	1.56	2.02	2.13
	t_{Add}	2.21	3.97	0.27	0.30	1.53	3.30	2.56	4.45
10 Hz	t_{Voting}	0.46	0.96	0.11	0.39	0.22	0.22	0.66	1.72
	t_{Network}	1.82	1.97	0.20	0.19	1.42	1.62	2.15	2.30
	t_{Add}	2.28	2.94	0.31	0.58	1.64	1.84	2.81	4.02
33 Hz	t_{Voting}	0.45	0.35	0.10	0.16	0.19	0.20	0.72	1.25
	t_{Network}	1.80	1.90	0.22	0.16	1.20	1.49	2.24	2.33
	t_{Add}	2.25	2.25	0.32	0.32	1.38	1.69	2.96	3.58
50 Hz	t_{Voting}	0.44	0.29	0.11	0.17	0.20	0.19	0.68	1.60
	t_{Network}	1.73	1.82	0.22	0.19	0.82	1.11	2.08	2.19
	t_{Add}	2.17	2.10	0.33	0.36	1.01	1.31	2.75	3.79

Table 18: Voting durations for STC₁ for TC₃ in ms

Again here, t_{add} is divided into the time required for the pure voting t_{voting} and the time required to transport the hash through Ethernet $t_{network}$. While the average voting time t_{voting} varied from 2.04ms at 5Hz to 0.29ms at 50Hz, the time to transport the hash over the Ethernet remained relatively constant with 1.82ms at 10Hz and 50Hz to 1.97ms at 10Hz (see summary in Table 18).

STC₂ Varying Data Rate: The average additional time required for voting t_{add} varied between 87.66ms for 40MB/s on channel₁ to 1.95ms at 25MB/s. The average standard deviation was between 204.69ms for 40MB/s to 0.22ms for 5MB. The maximum t_{add} was observable with 1306.41ms for 40MB/s.

While the average time for the pure voting t_{voting} varied from 0.39ms at 10MB/s to 85.13ms at 40MB/s, again here, the time to transport the hash over the Ethernet $t_{network}$ remained stable with 1.54ms at 25MB/s to 2.53ms at 40MB/s (for summary see Table 19).

20hz const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
5 MB	t_{voting}	0.44	2.38	0.09	0.30	0.24	1.54	0.63	3.26
	$t_{Network}$	1.88	1.92	0.13	0.15	1.42	1.43	2.08	2.17
	t_{Add}	2.32	4.30	0.22	0.45	1.66	2.97	2.71	5.42
10 MB	t_{voting}	0.39	5.09	0.07	3.46	0.24	1.43	0.52	31.28
	$t_{Network}$	1.80	1.89	0.20	0.24	1.07	1.15	2.17	2.41
	t_{Add}	2.19	6.99	0.27	3.70	1.31	2.58	2.69	33.69
25 MB	t_{voting}	0.41	42.13	0.10	35.22	0.23	24.09	0.66	241.72
	$t_{Network}$	1.54	1.62	0.28	0.25	0.90	1.06	2.29	2.17
	t_{Add}	1.95	43.74	0.38	35.47	1.13	25.15	2.94	243.89
40 MB	t_{voting}	1.29	85.13	2.87	201.78	0.23	37.64	15.94	1289.02
	$t_{Network}$	1.55	2.53	0.24	2.91	1.08	1.17	2.04	17.40
	t_{Add}	2.84	87.66	3.11	204.69	1.31	38.81	17.98	1306.41

Table 19: Voting durations for STC₂ for TC₃ in ms

Recovered State:

After Pi was restarted on ECU₁, what was arbitrarily set to 3.5 seconds, Pi phased in, while Pi'' phased out. After this phase in/out has happened, the logical configuration of the system again was as in nominal state.

STC₁ Varying Update Rate: All remaining iterations until A_{1000} have been successfully validated by the distributed voting mechanism. The additional voting duration t_{add} varied from 2.02ms at 50Hz to 2.26ms at 5Hz with a standard deviation from 0.19ms at 5Hz for channel₁ and 0.47ms at 5Hz for channel₂.

While the average time for the pure voting was between 0.23ms at 33Hz and 50Hz to 0.37ms at 5Hz the time to transport the hash over the Ethernet required between 1.79ms at 50Hz to 1.89ms at 5Hz (for summary see Table 20).

1MB/s const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
5 Hz	t_{Voting}	0.27	0.37	0.07	0.36	0.20	0.20	0.80	2.22
	$t_{Network}$	1.88	1.89	0.12	0.12	1.01	1.33	2.30	2.37
	t_{Add}	2.16	2.26	0.19	0.47	1.21	1.53	3.10	4.59
10 Hz	t_{Voting}	0.30	0.29	0.12	0.16	0.19	0.19	0.91	1.43
	$t_{Network}$	1.82	1.88	0.17	0.15	0.94	1.18	2.28	2.40
	t_{Add}	2.12	2.17	0.29	0.31	1.13	1.37	3.20	3.83
33 Hz	t_{Voting}	0.23	0.23	0.04	0.04	0.18	0.19	0.61	0.65
	$t_{Network}$	1.83	1.80	0.11	0.14	1.10	0.91	2.25	2.00
	t_{Add}	2.06	2.03	0.16	0.18	1.28	1.10	2.87	2.65
50 Hz	t_{Voting}	0.23	0.23	0.04	0.03	0.17	0.03	0.54	0.54
	$t_{Network}$	1.82	1.79	0.15	0.15	1.05	1.12	2.03	1.99
	t_{Add}	2.05	2.02	0.19	0.19	1.22	1.15	2.57	2.54

Table 20: Voting durations for STC₁ for TC₄ in ms

STC₂ Varying Data Rate: The average voting duration varied between 51.75ms for 40MB/s to 2.18ms at 10MB/s. The standard deviation varied from 0.23ms at 5MB/s to 60.74ms at 40MB/s. The maximal additional voting duration was observable with 1586.3ms at 40MB/s, the minimum value was 1.08ms at 10MB/s.

The pure average voting time varied between 0.33ms at 5MB/s and at 10MB/s up to 49.4ms at 40MB/s. Again here, the network time remained relatively stable between 1.48ms at 40MB/s to 1.84ms at 10MB/s (for summary see Table 21).

20hz const.		Average		St. Dev.		Min.		Max.	
		Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
5 MB	t_{Voting}	0.33	0.86	0.10	1.41	0.21	0.21	0.75	38.18
	t_{Network}	1.89	1.95	0.12	0.11	1.09	1.13	2.13	2.37
	t_{Add}	2.23	2.81	0.23	1.53	1.30	1.35	2.88	40.55
10 MB	t_{Voting}	0.33	1.91	0.08	1.81	0.21	0.21	0.63	5.85
	t_{Network}	1.84	1.91	0.17	0.20	0.87	1.20	2.17	2.34
	t_{Add}	2.18	3.83	0.26	2.00	1.08	1.42	2.80	8.19
25 MB	t_{Voting}	0.36	6.94	0.14	10.76	0.20	0.21	2.31	88.41
	t_{Network}	1.96	1.95	0.17	0.18	1.02	1.05	2.25	2.41
	t_{Add}	2.32	8.89	0.30	10.94	1.22	1.26	4.56	90.82
40 MB	t_{Voting}	1.20	49.40	6.98	53.75	0.21	0.24	168.45	1416.4
	t_{Network}	1.48	2.36	0.25	6.99	0.90	0.95	2.21	169.89
	t_{Add}	2.68	51.75	7.23	60.74	1.11	1.19	170.65	1586.3

Table 21: Voting durations for STC₂ for TC₄ in ms

8.2.2 Final Acceptance Test

TC₁ Nominal State:

FAT₁ Software in the Loop Testing: For RoLanD in SiL testing the average observed voting duration t_{add} for channel₁ was 3.91ms with a standard deviation of 11.23ms and an average t_{add} of 5.72ms for channel₂ with a standard deviation from 16.09ms during the nominal phase for the first 199 samples. The maximum voting duration was 147.92ms, the minimal was 1.43ms.

The average voting time was 1.5ms for channel₁ and 2.96ms for channel₂ during nominal state. The network time is 2.41ms for channel₁ and 2.76ms for channel₂ (see Table 22).

It was observed that the first sample has a significant impact on the maximal values. The reason for this was, that the execution time for the first sample was 918.85ms on ECU₁ and 815.6ms on ECU₂ while for the remaining 198 samples until the faulty S_{200} the execution time had an average of only 89.9ms for channel₁ and 88.96ms for channel₂. This significant longer calculation duration for the first samples leads to waiting times for the voter with the

shorter calculation duration, representing the reason for the maximal values (detailed explanation will follow in chapter 5.2.6).

	Average		St. Dev.		Min.		Max.	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
t_{Voting}	1.50	2.96	5.57	10.48	0.36	0.36	43.64	102.57
t_{Network}	2.41	2.76	5.66	5.61	1.07	1.15	74.78	45.35
t_{Add}	3.91	5.72	11.23	16.09	1.43	1.50	118.42	147.92

Table 22: Voting durations for FAT₁ for TC₁ in ms

FAT₂: Real World Testing: For the open world testing the average voting durations for channel₁ have been in average 7.83ms with a standard deviation of 19.81ms. The average voting duration t_{add} for channel₂ was 14.01ms during nominal state with a standard deviation of 28.1ms. The minimal t_{add} in min was 1.36ms the maximum was 160.17ms.

The average voting time for was 5.91ms for channel₁ with a standard deviation of 18.58ms and 11.75ms for channel₂ with a standard deviation 24.8ms.

	Average		St. Dev.		Min.		Max.	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
t_{Voting}	5.91	11.75	18.58	24.80	0.34	0.32	98.73	113.77
t_{Network}	1.92	2.26	1.24	3.35	1.11	1.04	14.97	46.41
t_{Add}	7.83	14.01	19.81	28.15	1.45	1.36	113.70	160.17

Table 23: Voting durations for FAT₂ for TC₁ in ms

TC₂: On-Demand TMR

FAT₁ Software in the Loop Testing: $TC_{2.1}$: Same as in the STCs, also in the FAT₁ a fault was injected into channel₁. This fault was detected what initiated the on-demand TMR process.

$TC_{2.2}$: The time until the context was changed and RoLand'' was activated took t_{context} of 13.11ms and additional $t_{\text{call_back}}$ of 57.97ms were necessary until the first historic sample was received. It took 638.86ms to loop through the 15 historic samples and another 2041.03ms to calculate the third hash.

The serialization and the creation of the third hash took t_{hash} 2.1ms, the identification time t_{ident} of the faulty channel took V_{bu} 17.05ms. The remaining time of V_{bu} took 0.23ms and was necessary to reply the results to V_1 and V_2 .

The initial voting time with the result that hash_1 and hash_2 are not matching took 0.41ms for channel_2 , the overall network transportation time for both, the hashes for the initial voting, as well as the forwarding to the V_{bu} took 2.87ms for channel_2 (see summary in Table 24).

	FAT ₁		FAT ₂	
	Ch1	Ch2	Ch1	Ch2
t_{context}	Wait	13.11	8.02	Wait
$t_{\text{call-back}}$		57.97	21.51	
t_{loop}		638.86	318.78	
t_{calc}		2041.03	2024.94	
t_{hash}		2.1	1.09	
t_{ident}		17.05	31.4	
$t_{\text{voter_bu}}$		0.23	0.23	
$t_{\text{third_hash}}$	2771.43	2770.79	2405.96	2406.97
t_{Voting}	0.34	0.42	0.26	108.26
t_{Network}	4.72	2.87	4.03	3.66
$t_{\text{add_200}}$	2776.49	2774.08	2406.22	2518.89

Table 24: Durations for identification of faulty channel for FAT₁ and FAT₂ in ms

$TC_{2,3}$: Same as for the system testing, after the faulty instance of RoLand was identified, it was reconfigured on the $\text{ECU}_{\text{back_up}}$. Accordingly, again here, a ROS2 node holding the RoLand'' was activated what took t_{activate} of 145.46ms. After 123.39ms S_{201} was reached by looping through the historic samples and it took RoLand'' t_{calc} of 173.50ms to process S_{201} . After 442.34ms the system was in reconfigured state with the configuration of RoLand' running on ECU_2 and RoLand'' on $\text{ECU}_{\text{back_up}}$.

Same as with STC₁ and STC₂ the FHTI contains both, the faulty channel detection duration and the reconfiguration duration which together is 3212.9ms (see summary values in Table 25).

	FAT ₁ Ch3	FAT ₂ Ch3
t_{activate}	145.46	118.84
$t_{\text{loop_start}}$	123.39	202.86
$t_{\text{calc_r}}$	173.50	144.76
$t_{\text{add_52}}$	442.34	466.46
FHTI	3212.9	2872.19

Table 25: Durations for reconfiguration of App'' for FAT₁ and FAT₂ in ms

FAT₂: Real World Testing: TC_{2.1}: Again here, the injected fault at iteration 200 was detected during real world testing.

TC_{2.2}: The time to change the context was 8.02ms followed by 21.51ms until the call-back for the third instance of RoLand'' was activated. The loop through the historic samples took 318.78ms and the inference to create the third hash took 2024.94ms. It took 1.09ms to create the third hash and 31.4ms to identify the faulty channel. The pure voting time of the back-up voter was 0.23ms so that it took in total 2405.96ms to create the third hash.

Adding the initial voting duration for the identification that the hashes are not matching took 0.26ms and a total network time to transport the hashes was 4.03ms. In total the whole $t_{\text{add_200}}$ was 2406.22ms (see summary in Table 24).

TC_{2.3}: After the faulty channel was identified it took 118.84ms to activate RoLand'' to reconfigure RoLand. The loop through the historic samples took 202.86ms followed by 144.76ms to execute iteration 201. This leads in total to a FHTI of 2872.19ms.

TC₃: Reconfigured State

FAT₁: Software in the Loop Testing: Same as in the system testing again here RoLand'' runs on ECU_{back_up} until it is recovered on ECU₁. The duration for the temporary operation of RoLand'' was set arbitrarily to 10 seconds since it took longer to load RoLand compared to the simple application Pi of STC₁ and STC₂. These 10 seconds mimic the restart of an ECU and the loading of the application.

The average required voting duration t_{add} was 8.8ms for channel₁ and 14.84ms for channel₂ with a standard deviation of 14.53ms for channel₁ and 16.47ms for channel₂. The maximal voting duration was 46.65ms for channel₂. The pure voting took an average of 5.14ms for channel₁ and 12.37ms for channel₂ while the time to transport the hash through Ethernet took in average 3.65ms for channel₁ and 2.47ms for channel₂ (see summary of in Table 26).

	Average		St. Dev.		Min.		Max.	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
t_{Voting}	5.14	12.37	7.55	14.00	0.42	0.48	30.42	46.65
$t_{Network}$	3.65	2.47	6.98	2.48	1.20	1.30	42.52	18.73
t_{Add}	8.80	14.84	14.53	16.47	1.62	1.78	72.94	65.38

Table 26: Voting durations for FAT₁ for TC₃ in ms

FAT₂: Real World Testing: Also, for FAT₂ the system was able to continue in the reconfigured state. The average voting duration t_{add} for those samples was 9.1ms with a standard deviation of 23.51ms for channel₁. Channel₂ showed an average voting duration t_{add} of 6.38ms with a standard deviation of 4.56ms.

While the pure voting duration for channel₁ was 7.18ms with a network transportation time of 1.92ms it was 4.37ms for channel₂ with a network transportation time of 2.01ms.

	Average		St. Dev.		Min.		Max.	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
t_{Voting}	7.18	4.37	23.51	4.26	0.38	0.42	94.42	16.78
$t_{Network}$	1.92	2.01	0.90	0.30	1.08	1.34	7.25	3.41
t_{Add}	9.10	6.38	24.41	4.56	1.46	1.76	101.67	20.19

Table 27: Voting durations for FAT₂ for TC₃ in ms

TC₄: Recovered State

FAT₁: Software in the Loop Testing: After the reconfiguration RoLand was started on ECU₁ again and the remaining samples up to S_{1000} have successfully been processed.

The voting required t_{add} for channel₁ in average of 7.88ms and for channel₂ an average of 12.15ms with the maximal value of 907.95ms. This maximum value was at iteration 255 as first sample of the recovered state. Same as in TC₁ it is related to the heavy difference in the RoLaND calculation time. While it took RoLaND' 114.38ms to be processed for sample 255, at the same time it took RoLaND 899.41ms (explanation will follow in chapter 5.3.3). The standard deviation varied from 15.67ms for channel₁ to 39.01ms for channel₂.

The pure voting time was in average 5.57ms for channel₁ and 8.79ms for channel₂ while the time to transport the hash through Ethernet varied from 2.31ms for channel₁ to 3.36ms for channel₂ (see summary of values in Table 28).

	Average		St. Dev.		Min.		Max.	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
t_{Voting}	5.57	8.79	12.18	30.99	0.36	0.37	103.80	65.85
$t_{Network}$	2.31	3.36	3.49	8.02	1.02	1.10	67.49	105.55
t_{Add}	7.88	12.15	15.67	39.01	1.38	1.47	171.28	171.4

Table 28: Voting durations for FAT₁ for TC₄ in ms

FAT₂: Real World Testing: Again here, the system was able to process the remaining samples during the recovered state.

The average voting duration t_{add} for channel₁ was 11.59ms and for channel₂ 10.92ms with a standard deviation of 31.51ms for channel₁ and 39.02ms for channel₂. The maximal value was 798.14ms, again here this value was observable at the first sample after the reconfiguration happened at iteration 243.

The voting time for channel₁ was in average 9.37ms and for channel₂ 8.6ms. The time to transport the hash over the Ethernet was in average 2.22ms for channel₁ and 2.32ms for channel₂.

	Average		St. Dev.		Min.		Max.	
	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2	Ch1	Ch2
t_{Voting}	9.37	8.60	26.52	35.52	0.35	0.33	115.46	117.75
$t_{Network}$	2.22	2.32	5.00	3.50	0.98	0.82	119.28	54.21
t_{Add}	11.59	10.92	31.51	39.02	1.33	1.15	234.74	171.96

Table 29: Voting durations for FAT₂ for TC₄ in ms

8.3 Autonomous Driving Sensors

Beside the camera as used in this work, there are typically more sensors used in a L4 systems:

Radar: A radar measures the time of flight of a high frequency electromagnetic wave and thereby calculates the distance and speed of objects, which reflect the wave. According to Liu et al. [58] a radar has a higher price than a camera and is less affected by weather and low lightning environment and thereby provides valuable distance and speed information. Its sensing range is up to 200 m and it produces 10 – 100KB of data per second at an update rate of 24 GHz or 77 GHz [21]. However, radar signals depend on the reflectivity of the objects in terms of size and material. Big metal objects can be easily detected, whereas small objects or objects not containing metal (e.g. wood) are hard to detect. The current research on radar is mainly focused on the resolution. A higher resolution provides more accurate data which could be used for object classification [15]. The number of applied Radars varies between 0 - 10. The used network technology is CAN, however an information was only available for 2 academic research vehicles (see academic research vehicles Appendix 8.6, analysis of [15] for industrial research vehicles).

Ultrasonic Sensor: The ultrasonic sensor is based on ultra-sound and calculates the distance to an object based on the time of flight. It has a good performance also in low light and bad weather conditions but works only in short distances, with less than 20 m. It produces between 10 – 100KB of data per second at an update rate of 20kHz [21]. The amount of applied ultra sonic sensors varies between 0 - 13. Information about the used network technology was not found (see analysis academic research vehicles Appendix 8.6, analysis of Marti et al. [15] for industrial research vehicles).

Global Navigation Satellite System: A GNSS is used to receive information about the position of the vehicle. There are several such systems existing, such as GPS, Galileo or BeiDu Navigation Satellite System. The accuracy of GPS varies between few meters to 20 meters. Real-time kinematic solutions use

dual-frequency GPS receivers to estimate the relative position of a vehicle with respect to the position of a base station with a known position. This brings the accuracy down to a few centimeters. Another improvement is expected by the European Galileo GNSS, which promises to provide centimeter precise information. However, despite the accuracy, the availability remains a problem, especially in inner urban environments or with respect to weather conditions [107]. Additionally, the update rate of GPS is 10 Hz [59], what would lead to a positioning error of appx. 3 m for a vehicle traveling at 100 km/h speed.

Inertial Measurement Unit: The IMU consists of gyroscopes and accelerometers. With a gyroscope it is possible to measure the angular speed of axes (yaw, pitch, roll). The accelerometer measures the three axes linear speed. [58]. A modern IMU has an update frequency of up to 200 Hz [59]. The typical setup within the research vehicles was a combined solution, where the IMU also contained a GNSS sensor (see analysis academic research vehicles Appendix 8.6, Sensor Fusion chapter 8.2).

8.4 AD-Software and Algorithms

In this work, a lane detection application was selected, which belongs to the localization module. However, an AD-Software stack contains more modules with a variety of used algorithms.

8.4.1 Sensing

Definition 8. 1: *Sensing is the process of gathering raw data with the sensors*

The sensing module provides access to the data, created by the physical sensors. Technically it represents the hardware driver layer. It gathers the data and pre-processes it in a manner, that it can be made available for other applications. Bapp et al. [77] state that “Sensing is easy, perception is difficult”.

Raw-data itself is not sufficient for the vehicle to take decisions. The raw-data is further processed in the following modules, deriving semantics from the scenery.

Sensors are related to two restrictions, namely the quality of data with regard to external conditions such as weather (see Table 3) and the type of information which can be derived from their data. Table 30 allocates the capabilities of typical AD-Sensors to the required type of information.

Example: A mono camera can classify objects. However, its information can only barely be used for localization purposes.

Information	Spatial			Identification	Kinematics	Semantics					Context	
	Location	Size	Shape			Traffic Signs	Road Marks	Gestures (human)	Clothes (human)	Vehicle	Weather	Driving Situation
Sensor												
Camera Mono	-	+	+	+	-	+	+	+	+	+	+	+/-
Camera 3D	+/-	+	+	+	+/-	+	+	+	+	+	+	+/-
Radar	+	+/-	-	+/-	+	Not possible						
Lidar 2D	+	+	+/-	-	+	Not possible						
Lidar 3D	+	+	+	+	+	-	+/-	+/-	Not possible			+/-
Ultra Sonic	+/-	Not possible			+/-	Not possible						
IMU	+/-	Not Possible			+	Not possible						
GNSS	+/-	Not Possible			+/-	Not possible						

+ Well suitable, +/- Moderately suitable, - Low suitable

Table 30: Sensor adequacy for relevant types of information according to [50]

Both aspects, the limitation with regard to weather and to the type of information of a sensor lead to the problem that an AD-System can not rely on information of only one sensor to realize the driving task.

This limitation is overcome by using sensor fusion, which is an essential aspect of autonomous driving [61].

Definition 8.2: *Sensor fusion is the process of integration of acquired data from multiple sensing sourced to reduce the number of detection uncertainties [61].*

Example: While the localization error of GPS is too high for automated driving purposes (multiple meters and update rate of appx 10 Hz, see chapter 3.1), the fusion with IMU data (relative position and update rates of 200 Hz) reduces the error to appx. 10 cm.

A well-established fusion algorithm to realize the above mentioned example is the Extended Kalman Filter (EKF) which is typically used in an environment, where the sensors suffer from noise and the estimated states of the system can be represented with a Gaussian Normal Distribution.

8.4.2 Detection

Definition 8.3: *Detection is the process of converting raw data into semantic information by identifying objects such as obstacles, pedestrians, other road users, lanes, landmarks, traffic-lights or traffic signs [108].*

The progress within the research of artificial intelligence lead to separate between traditional and DL based detection methods [73].

Traditional computer vision algorithms are feature related. Based on similarities, such as colors, shapes or textures, features are extracted from an image [74]. Typical algorithms for feature extraction are Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), Features from Accelerated Segment Test (FAST), Hough Transforms and Geometric hashing [75]. The peak of performance for traditional algorithms was the Deformable Part-based Model (DPM) algorithm, which won the Pascal Visual Object Classes Challenges in the years 2007, 2008 and 2009 with an object detection accuracy of 21% in the 2008 challenge [73].

Deep Learning Based Methods:

One of the first DL based algorithms was the Regions with CNN features (RCNN) algorithm in the year 2014. It was applied to the VOC data set of 2008 in order to compare its accuracy to DPM. With an accuracy of 58,5%

(compared to 21% DPM) it clearly can be seen that DL based approaches by far outperform traditional computer vision algorithms [73]. Since then, the technology continuously has been further refined. Multiple algorithms have been developed such as Fast and Faster RCNN, Alex Net, Single Stage (Yolo), You Only Look Once (YOLO), Retina-Net and Refine-Net [73] which are popular also for object detection within autonomous driving [48].

For DL based approaches, beside 2D images, also 3D data as point cloud, either directly from a lidar or derived from a stereo-camera, can be taken as input for DL-algorithms. PointNet and Voxel Net are two examples of algorithms which detect objects and additionally provide depth information based on 3D data up to a precision of 3 cm when using a lidar. However, since 3D data alone does not contain rich visual information for object classification it is typically used in combination with above mentioned 2D visual classification methods to receive both, a high accuracy on detected objects as well as a high accuracy on the corresponding depth information [48] (see sensor fusion chapter 8.2).

8.4.3 Localization

Definition 8.4: *Localization is the process of positioning the vehicle, a pedestrian or other road furniture, with respect to a reference map [107].*

The task to determine the pose of an AV is called the localization problem.

Definition 8.5: *The pose of a vehicle is its position and its orientation.*

For localization purposes, two types of maps exist. Planar High Definition (HD) Maps and Point-Cloud Maps. HD-Maps contain information about infrastructure such as roads, lanes, signs or landmarks while point-cloud maps represent the environment by data-sets [107].

The localization problem is solved with different algorithms. The Iterative Closest Point (IPC) algorithm is a numeric method¹ which compares an existing point-cloud map with the instant measured point-cloud of the vehicle [58]. The localization error of this method is < 20 cm [109]. Widely used alternative solutions are probabilistic algorithms², namely the Extended Kalman Filter (EKF), Particle Filters (PF) or Markov Localization (ML). The EKF provides a position on a map by fusing noisy sensor data such as GPS, IMU or visual odometry³ which localizes on centimeter level but suffers from weather and signal availability limitations (see chapter 8.2). Particle Filters randomly place samples (particles) on a map and calculate the probability that a sample is the real position by comparing sensor data to map data [110]. The localization accuracy is < 1.5 m [111]. ML requires significant higher compute and memory demand and at the same time provides less accuracy, thereby being inferior compared to MCL [112].

In a comparison between multiple localization methods done by Kuuti et al. [107], the best results were achieved by Kim et al [113]. The authors reduced the localization error to below 10 cm by applying a mixture between point cloud based localization in combination with a PF.

Deep Learning Based Methods:

A promising example of deep learning for localization can be found in the work of Barsan et al. [109]. A trained CNN was able to localized the vehicle by using a instantly captured point cloud map. This approach provides a

¹ Numeric methods solve mathematical problems by approximation with an algorithm.

² Probabilistic algorithms try to achieve an approximately correct result by selecting random interim results.

³ Visual Odometry is using camera data and detects the relative movements of the pixels on each image, thereby deriving odometry data of the vehicle.

localization accuracy of below 10 cm and at the same time does not require costly CPU performance but could be executed on the GPU.

In contrast to the above mentioned, an AV could also act in an unknown environment. In that case no map is existing which increases the difficulty of the localization problem since the map has to be created first. This problem is called Simultaneously Localization and Mapping (SLAM) [114]. However, for L4 AVs the ODD and its environment is known.

8.4.4 Prediction

Definition 8.6: *Prediction is the process of understanding the surrounding human driver intentions [115].*

To take driving decisions, it is important to have a model available of how the surrounding may develop. The Prediction module takes the detected (dynamic) objects from the Detection module and forecast their behavior.

Example: A pedestrians who is walking parallel beside the road of the AV may suddenly enter the driving corridor. However, the probability of this event is lower, than if the pedestrian would directly walk towards the drive corridor.

Lefevre et al. [116] distinguishes between three layers, namely physics-based, maneuver-based and interaction-aware motion modelling.

The simplest case is the physics-based approach which predicts the future behavior based on the law of physics by designing dynamic and kinematic models. While dynamic models typically describe motions based on Lagrange's quotations, for kinematic models multiple methods such as Constant Velocity, Constant Acceleration, Constant Turn Rate and Velocity and Constant Turn Rate and Acceleration have been evolved. Kinematic models are more popular in practice since they are simpler [116].

Deep Learning Based Methods:

In contrast to physics-based approaches, maneuver-based approaches early detect the maneuvers, that other drivers intend to perform, thereby assuming his or her future motion. From those scenarios a finite amount of prototype trajectories is generated which has been learned during a learning phase based on previously observed maneuvers. Typical used algorithms are the Multi-Layer Perceptron, Logistic Regression, Relevance Vector Machines, Support Vector Machines or Hidden Markov Model [116].

Several attempts to predict the behavior of vehicles using DL methods have been executed. One group is using recurrent neural networks (RNN)s, anticipating the future target position of an object to be the result of the previous sequences. Another group is using CNNs in combination with the bird view⁴ perspective, provided by the vehicle cameras. CNNs provide good spacial information such as future occupancy of areas on a map. However, they are not able to model data as a series, thereby ignoring the states of the objects over time. To close this limitations the last group combines RNNs with CNNs. For instance several images from the birds eye perspective are fed into a RNN, thereby enriching the spacial dimension of a CNN with temporal dimensions of a RNN [117].

Interaction-aware motion models represent vehicles as maneuvering entities which interact with each other. Thus, the motion of a vehicle is influenced by the motion of the other vehicles. Most interacting-aware motion models are based on Dynamic Bayesian Networks in combination with Coupled Hidden Markov Models [116].

⁴ Bird view is a 360° image of the surrounding of the vehicle using cameras.

8.4.5 Mission

Definition 8.7: *Mission planning is the process to determine a possible route from an origin to a desired destination of the AV [115].*

Global navigation is a well-studied subject for almost all modern cars. GPS in combination with an offline map is used to plan a global route [115]. Multiple algorithms have evolved which [118] however the A* algorithm evolved to be standard since 50 years [115]. It was derived from the goal-directed Dijkstra Algorithm, which scans in a graph network all vertices with distances to figure out the shortest distance from an origin to a destination.

The mission planner provides a high-level path as waypoints for the vehicle to follow. While following this path, the vehicle incrementally takes decisions regarding its behavior according to stipulated traffic rules, signs, or individual traffic situations.

8.4.6 Behavior

Definition 8.8: The behavior modules realizes the mission and reacts to the environment by determining the appropriate driving behavior.

Example: The vehicle is approaching a traffic light. Because of this, the behavior module changes from “follow the lane” to “traffic light handling” which will then check the color of the traffic light and react accordingly.

The dominant guiding criteria for the behavior selector are safety and road traffic rules. The vehicle must always behave in a manner that it operates with maximum safety for itself and its environment, taking into account all stipulated traffic rules [108]. This overarching guideline is the reason, why the behavior module represents the most critical part in the AD-Stack with respect to ethical decisions [60]. In principle it can be distinguished between logical rule based decision making modules and deep learning based decision making modules [119].

A logical rule based decision module is typically designed as finite state machine [60], where each state represents a specific driving maneuver.

Definition 8.9: *Driving maneuvers are closed-loop control algorithms, each capable of maneuvering the AV in a specific traffic situation [120].*

A driving maneuver is activated once specific discrete entry events are fulfilled. Therefore, [120] formulates the decision problem as following:

- As set $M_{all} = \{m_1, m_2, \dots, m_n\}$, $n \in \mathbb{N}$, of all available driving maneuvers which can be performed by the AV,
- A k -tuple $(w_1, w_2, \dots, w_k) \in W_{events}$ of events, $w_l \in \{0,1\}$, $l = 1,2,\dots,k$,
- A direction indication $d_i \in D_{route} = \{\text{forward straight, forward right, forward left, turn around}\}$.

The general problem of decision-making is, to identify the most appropriate driving maneuver $m_{most_appr.} \in M_{all}$.

This decision-making process requires a mixture from a priori information, such as the route, coordinates of intersections or information about the road infrastructure (e.g. lanes). Additionally, information from the detection module, such as obstacles, traffic lights etc. are considered. Based on this information, a two-step decision process is activated: First, all safe driving maneuvers are evaluated which allow the vehicle to follow the route. Based on this, in the second stage the driving maneuver which provides the most comfort and efficiency is selected [120] and represented as lane level path.

On one side, the finite character of the state machine and the driving maneuvers allows to use them as use-case input for the V-Shape model, including the definition of testing cases and thereby realizing an appropriate safety approach. On the other side, the AV is limited to the number of its specific maneuvers, what may lead to an unexpected behavior in a situation which was not explicitly accounted [108].

Deep Learning Based Methods:

An alternative to a rule based state machine are recent deep learning approaches. One option is Imitation Learning (IL) where the vehicle learns how a human would behave in a equivalent situation. Technically this refers to the Deep Reinforcement Learning (DRL) algorithm where the goal is to learn the reward function by observing a human driver [48].

8.4.7 Motion

Definition 8.10: *Motion planning is the process of computing a drivable trajectory which follows the desired lane level route, given by the behavior module [60].*

The motion planner is responsible to compute a drivable trajectory, also called path, taking into account the requested maneuver from the behavior module, the cars dynamic and the drivable area [108]. In literature four types of motion planning techniques have evolved: Graph Search Based Techniques, Sampling Based Techniques and Interpolating Curve Based Techniques [121]. They all have in common, to transform the continues space into a discrete model [108].

Graph Search Based Planners have similarities to the global route planning algorithms in the Mission Module (chapter 8.4.5). Typically, a defined area in front of the vehicle, the state space, is represented as an occupancy grid map which is a map divided into multiple cells on which the objects in the given environment are placed. Along the not-occupied grid cells, a graph search takes place, typically by using the Dijkstra, A* or State Lattice algorithm [121]. Graph Search Based Planners suffer from exponentially increasing computational cost when applied to problems in higher dimensions [122].

Possible solutions are Sampling Based Techniques, which, as already described, randomly place samples within the state space and generate connections between them. Rapidly-Exploring Random Tree is a widely used

algorithm which builds a search tree from the cars current position to a goal by randomly placing samples in the drivable area [60]. This approach provides fast results, however at the same time they don't represent optimal solutions [121].

Interpolating Curve Planners take higher level waypoints such as a given lane level path from the behavior module and further refine it. Therefore, it generates a new set of data in benefit of the trajectory continuity, the vehicle and environment constraints. In case of an obstacle it generates a new path to overcome the obstacle and the re-entry the previously planned path [121]. The interpolation process takes place by using Lines and Circles, Clothoid Curves, Polynomial Curves or Bezier Curves. A disadvantage is, that it needs waypoints as input and re-calculation in case of an obstacle requires additional compute resources [121].

Deep Learning Based Methods:

DRL for path-planning is an approach where a vehicle would learn driving trajectories in a simulator. Similar to IL the attempt is to learn and maximize a reward function with regard to comfort, safety, overtake opportunity and others [48]. Even though the main purpose of this approach is path planning, it also contains elements of behavior such as realization of overtake opportunities.

8.4.8 Control

Definition 8.11: *The controller module realizes the trajectory from the motion module by translating it into commands to the actuators such as steering angle, throttle or brakes [60].*

The control module is responsible to convert the trajectory from the motion module into real driving commands. Those driving commands are realized by sending them to the actuators as signals which then convert them into physical actions [60]. The result of the motion planner can be a sequence of

commands, such as a desired velocity or steering angle at a specific point in time with the target to reach a specified goal while avoiding collisions with obstacles. There are several methods existing to realize the control module. Proportional Integral Derivate Control is based on a specific hardware input and an error measurement which gives feedback of how far the output is away from the input [60]. Stanley Control is utilized to follow the reference path by minimizing the heading angle [58].

Deep Learning Based Methods:

Recently learning controlling techniques such as the Iterative Learning Control or the Model Predictive Control have been evolved. Iterative Learning Control (ILC) is a method for controlling systems which typically execute tasks repetitively, taking into account learnings from the previous iteration. Model Predictive Control (MPC) typically solves an optimization problem based on minimizing an underlying cost function over a short time horizon, while considering input-output constraints and the system's dynamics given by a process model. Those traditional MPC techniques can be conjunct with Convolutional Neural Networks [48].

Another approach to control the vehicle are the End2End learning methods. However, those methods reflect the whole pipe from perception to control, they directly deriving control commands from the detection of objects and scenes. A list of algorithms with explanations can be found in [48].

8.5 Other Artificial Intelligence Technologies

In this work, a CNN was used. However, a L4 software stack typically contains more AI technologies.

A: Recurrent Neural Networks:

RNNs are similar to FFNs but contain backwards oriented edges. That means that the information is not always propagated towards the output but can be traced back to a previous layer of the neural net. This feedback loop is also called memory cell and is made for applications where sequential data is processed such as speech recognition [50]. In the context of autonomous driving an example could be the scene prediction. RNNs use a specific variant of the FFN learning algorithm.

B: Deep Reinforcement Learning:

Reinforcement Learning itself is a form of artificial intelligence which is not related to deep learning. Contrary to the supervised learning approach, reinforcement learning is not supervised. Figure 73 shows an agent, which receives state information from its environment. Based on this state it takes decisions in form of actions which influence the environment. The learning is realized by rewards for its actions thereby providing feedback what is a desired action and what not.

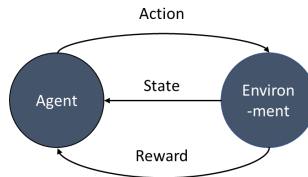


Figure 73: Reinforcement learning agent model

The Markov Decision Process (MDP) is a model which tries to maximize the accumulated reward of an agent towards a sequence of decisions. Deep Reinforcement Learning (DLR) is building on top of MDP by adding neural nets to this decision process, thereby learning which sequence of decisions lead to a maximum reward [123].

8.6 Research Vehicles

8.6.1 Approach

The following research is based on the IEEEExplore Database, using a search string which is composed by two parts. It starts with “Autonomous AND Middleware” followed by another AND connection to the XOR attributes “safety”, “scalable”, “flexible” and “security”. The search string is depicted in Table 31.

The result was a total of 221 articles which all have been analyzed. The majority was related to autonomous robotic, marine and aerial systems. Thus 9 articles have been considered to be important for this thesis. Starting from these 9 articles, a forward, backward approach was executed.

Autonomous AND Middleware AND (Safety XOR Scalable XOR Flexible XOR Security) = • Autonomous AND Middleware AND Safety • Autonomous AND Middleware AND Scalable • Autonomous AND Middleware AND Flexible • Autonomous AND Middleware AND Security
--

Table 31: Search String

The analyzed articles contained references to research vehicles. However, further delimitation was necessary.

A wide field of research refers to miniature vehicles. Those vehicles don’t act in public areas and will not be further considered. Additionally, the terminology “autonomous” was used with different understanding. Some authors consider SAE Level 3 or even ADAS-vehicles to be autonomous. Since the focus of this thesis is on SAE Level 4 (L4), those research vehicles also have been excluded.

The remaining of the identified vehicles was analyzed according to available information. Some of them were described on a high level, which was not

sufficient in order to get a deep insight into their system and software architecture. This typically applies for industrial research vehicles as companies want to show that they work on autonomous but don't explicitly want to show how they work on it.

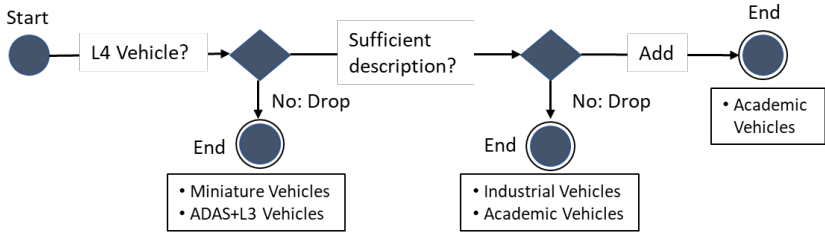


Figure 74: Decision Process

8.6.2 Vehicles

Boss (Tartan Racing, Carnegie Mellon University, 2007)	
System:	
Camera:	• 2 x High dynamic range (HDR) camera (Point Grey Firefly)
Lidar:	• 1 x 360°/90° with 70 m max. range (Velodyne HDL-64) • 6 x 180°/90° with 80 m max range (SICK LMS 291-S05/S14) • 2 x 12°/3.2° with 150 m max range (Continental ISF 172) • 2 x 240°/3.2° with 300 m max range (IBEO Alasca XT)
Radar:	• 5 x 60° or 17°/3.2° with 60 m or 200m range (Continental ARS 300)
Ultra-Sonic:	None
IMU:	Tightly coupled IMU with GPS (Applanix POS-LV 220/420)
GNSS:	
Connectivity:	Not described
Compute:	• CompactPCI chassis with 10 2.16-GHz Core2 Duo processors • Each 2 GB of memory, a pair of Gigabit Ethernet ports and 4-GB flash drive • Two machines with 500 GB hard drive for data logging • Time synchronized through custom pulse-per-second adaptor
Power System:	12V DC for vehicle • 24V DC battery converted to 120V AC for AD-System
Other:	• Active sensing data more valuable than visual data • Custom drive-by-wire interface • Computer 1 mainly to run sensor and bus system processes, as well as perception process • Computer 2 mainly for planning, motion controlling, health monitoring etc. • Presumably CAN since can process on ECU
Software:	
Localization:	• Capable to estimate road geometry or localize itself relative to known geometry • Map incl. markings derived from aerial image, but not sufficient due to disrupted signals

	• Lane detection (using SICK Lasers) for roads with markings implemented • Road shape estimator for areas, where geometry is not known a priori
Detection:	• Moving object detection: Classified into states (moving, not moving, observed moving or not observed moving), observed moving objects behavior will be predicted • Static obstacle detection: Static obstacle map is created
Prediction:	• Prediction based on logical constraints for objects located on the road • Whenever a driver can make a choice (e.g. intersection) multiple options are predicted
Mission:	• Planning with route network definition file (graph based search with cost function) • Update of graph in case newly observed information such as road blockage
Behavior:	• Responsible to execute the policy from the mission planner • Highest level separation of states: Lane driving, intersection handling, zone pose (parking)
Motion:	• Part of behavior within specific state
Control:	• Not explicitly explained
Middle Layer:	
• Abstracts around basic inter process communication over UNIX sockets, TCP/IP or UDP • Anonymous publish/subscribe principle for easy interchange of components	

AnnieWay (Team AnnieWay, TH Karlsruhe, 2007)	
System:	
Base Vehicle:	VW Passat
Camera:	None
Lidar:	• 1 360°/26.5° on roof top, max. range 100 m • 2 lidars at front and back of vehicle for parking
Radar:	None
Ultra-Sonic:	None
IMU:	GPS—aided inertial navigation system
GNSS:	
Connectivity:	Not described
Compute:	• 1 quad-core computer

	• 1 ECU for low – level control • Communication via Ethernet
Power System	Not described
Other:	• CAN communication to drive-by-wire system • Odometry measured directly from wheels • Remote E-Stop existing
Software:	
Localization:	• A occupancy grid map is created by using multiple sensors • RNDF file given, converted to graph-based representation
Detection:	• Pipeline from raw sensor data to a list of dynamic obstacles, incl. estimated location, sizes and relative velocities • Lane and curb detection using the lidar sensor • Detection of blocked areas
Prediction:	• State vector for each moving object using a Kalman filter
Mission:	• Graph based search with regard to minimal travel time
Behavior:	• Driving behavior based on a hierarchical state machine where each state represents a high-level driving behavior (e.g. drive, zone or intersection) • Sub-states represent refine high level states with precise behaviors (e.g. drive on lane, drive stop, ...)
Motion:	• Not described (respective included into control)
Control:	• Combination of path planning and determining the free section of the path • Lateral and longitudinal control
Middle Layer:	
• Central communication framework: Real-time database for cognitive automobiles communication KogMo-RTDB	

Caroline (Team Carolo, TU Braunschweig, 2007)	
System:	
Base Vehicle:	VW Passat
Camera:	- Stereo camera behind front window - Universal mono camera on the front roof 4 Point Grey Flea2 (HDR)
Lidar:	2 lidars (in front and back under license plate), (Hella IDIS), max. range 200 m 2 lidars (left- and right) 240°, 60m max range (Alasca XT) 1 180°/3.2° at rear side (Ibeo ML) 2 120° on front roof, max range 20 m (Sick LMS-291)
Radar:	4 radars (front, rear and both sides)
Ultra-Sonic:	None
IMU:	GPS—aided inertial navigation system
GNSS:	
Connectivity:	Not described
Compute:	Multiple industrial PCs
Power System	Not described
Other:	- CAN communication to drive-by-wire system - Remote E-Stop existing 4 Cameras use IEEE 1394 interface - Lidar + radar with CAN - PCs communicate via Ethernet - Camera on rooftop uses USB2
Software:	
Localization:	Grid-Map creation
Detection:	- Object detection based on a pipes-and-filters pattern using sensor fusion (lidar and radar) - Lane detection using cameras - Detection of drivable/non drivable areas with cameras
Prediction:	6D state vector for moving objects
Mission:	- Usage of mission definition file and route network definition files - Search algorithm not further specified
Behavior:	Artificial intelligence decides whether to follow way-points from RNDF or to stay in lane

	• On each point, an interrupt can take place, taking into account situations such as intersection, queueing, parking etc.
Motion:	• Not clearly described (evt. part of behavior in the course of trajectory builder)
Control:	• Lateral and longitudinal controlling
Middle Layer:	
• Not clearly described • Linear communication between the modules	
Other Remarks:	
• Description of AI for components • Safety concept using a watchdog indicated	

Leonie (TU Braunschweig, 2010)	
System:	
Base Vehicle:	VW Passat
Camera:	2 cameras 1 at front and 1 rear, only for data recording purposes
Lidar:	Hella IDIS2 lidar with 160° below the front bumper
Radar:	None
Ultra-Sonic:	None
IMU:	iMAR iTrace GPS/INS unit with improved accuracy through real time kinematic correction data obtained via 3G network
GNSS:	
Connectivity:	3G network
Compute:	3 Industrial PC - Control task PC - Visualization PC (co-driver seat) - Recorder PC (Blackbox recorder)
Power System	Supplied with power from secondary battery which is charged by additional generator
Other:	- CAN communication to vehicle actuators via vehicle gateway - CAN communication with sensors - PCs are connected with gigabit Ethernet
Software:	
Localization:	Localization on given map with GNSS and IMU
Detection:	Lane detection with lidar
Prediction:	Not described
Mission:	Not described
Behavior:	Artificial Intelligence to choose follow the lane or traffic light (traffic light status via HMI)
Motion:	Not described
Control:	Responsible to keep the trajectory (No explanation where trajectory was calculated)
Middle Layer:	
Real-time communication framework RTI DDS	
Other Remarks:	
Safety aspects indicated (front and rear camera to record situations)	

CaRINA II (INCT-SEC, 2012)	
System:	
Base Vehicle:	Fiat Palio Adventure
Camera:	• 1 x PointGrey Bumblebee2 stereo • 1 x LadyBug2 spherical
Lidar:	• Velodyne HDL32 (Roof top) • SICK LMS 291 (Front bumper)
Radar:	None
Ultra-Sonic:	None
IMU:	Integrated GPS/IMU Xsens MTI-G
GNSS:	
Connectivity:	• Teleoperation possible
Compute:	• 3 Industrial PC ◦ Vehicle interface (low-level steering and speed control) ◦ Sensorial perception ◦ User interface and data logging
Power System	Not described
Other:	• No description about usage of CAN • Local network of PCs indicate usage of Ethernet
Software:	
Localization:	• Mapping using lidars • Given topological map • World model including drivable areas and obstacles
Detection:	• Obstacle, lane, curb and lane detection
Prediction:	• Not described
Mission:	• Given GPS waypoints
Behavior:	• State machine • Artificial neural network decides the states for each approached node (from topological map)
Motion:	• “Reactive Control” for each state (e.g. keep distance to curb) • Calculation of drivable path, taking into account kinematics of vehicle and detected obstacles
Control:	• Control of steering, brake and throttle
Middle Layer:	
• ROS as middle layer	
Other Remarks:	
• Safety aspects such as raw data measurement concept	

Bertha (KIT in cooperation with Daimler AG, 2012)	
System:	
Base Vehicle:	Mercedes-Benz S 500 INTELLIGENT DRIVE
Camera:	• 1 StereoCamera • 1 wide-angle monocular color camera for pedestrians and traffic lights) • 1 wide-angle monocular color camera looking backwards for self-localization
Lidar:	• None
Radar:	• 4 short-range radars • 3 long-range radars
Ultra-Sonic:	• Standard S-500 Ultra-Sonic sensors
IMU:	Not described
GNSS:	• Standard S-500 GPS Sensor
Connectivity:	Not described
Compute:	Not described
Power System	Not described
Other:	• No description about usage of CAN
Software:	
Localization:	• Feature-based localization (Landmarks on a map) • Lane-marking-based localization
Detection:	• Explicitly for vehicles, pedestrians and traffic lights • Sixtel Vision to detect drivable areas with camera • Radar Processing for detection of scenery within intersection
Prediction:	• Possible movements of pedestrians and vehicles within a short time
Mission:	• Fixed route (no recalculation)
Behavior:	• Hierarchical state machine with 4 states and sub states
Motion:	• Driving corridor including obstacles and other traffic participants calculated
Control:	• Trajectory planner/control for lateral and longitudinal control
Middle Layer:	
• No description	
Other Remarks:	
• Reactive layer (PRE-SAFE brake functions, standard Level2)	

BRAiVE (University of Parma, 2013)	
System:	
Base Vehicle:	Hyundai Sonata
Camera:	• 10 firewire A Cameras (PointGrey FireFlyMV, PointGrey Dragonfly2)
Lidar:	• 5 lidars ◦ 1 Hokuyo UTM-30LX (rear bumper) ◦ 2 Hokuyo UTM-30LX-EW ◦ 1 IBEO Lux ◦ 1 Hella IDIS
Radar:	• 1 Universal medium range radar at front bumper
Ultra-Sonic:	None
IMU:	• GPS assisted IMU (RaceLogic VBOX-2 + RaceLogic IMU)
GNSS:	
Connectivity:	None
Compute:	• 3 PCs with same setup and devoted to perception: Intel Core 2 Duo, 2.6 GHz and Mini-ITX industrial motherboard ◦ Frontal obstacle and lane detection ◦ Lateral obstacle detection ◦ Rear and approaching obstacles detection plus road markings and traffic lights • dSpace Micro Autobox to connect vehicle CAN
Power System	• Additional 100 Ah battery for AD-System
Other:	• Cameras connected via Firewire • Radar is connected via CAN • Lidar is connected via USB 2.0 • Sensors connected with Firewire, USB, CAN and Ethernet • Additional yaw rate sensor • All vehicle actuators are controlled via CAN • PCs are connected with Ethernet
Software:	
Localization:	• World Perception Server creates a Global Perception Map by sensor input • Localization on lane (keep the lane) • Offline map used
Detection:	• Explicitly traffic light, traffic signs, obstacles, vehicles, lane detection, parking lot, tunnel and pedestrian detection • Classification done by the world perception server
Prediction:	• Not clearly described but statement, that classified objects behavior can be better predicted

Mission:	• Mission planed through offline map • GPS waypoint following
Behavior:	• Set of different maneuvers existing (e.g. lane keeping, vehicle following)
Motion:	• Path planner generates drivable trajectory, based on input from the world perception server and the obstacle avoider module
Control:	• Trajectory is transferred to low level controller which creates the actual steering commands
Middle Layer:	
• VisLab Bus as message passing service • Reference to LCM: Lightweight communications and marshalling, which is based on publish/subscribe pattern	
Other Remarks:	
• Brakes and gas can be overridden by using pedals, also steering wheel • Emergency baking if obstacle appears	

IARA (UFES, 2017)	
System:	
Base Vehicle:	Ford Escape Hybrid
Camera:	• 2 x Bumblebee XB3 • 1 x ZED
Lidar:	• Velodyne HDL-32E • SICK LD_MRS
Radar:	None
Ultra-Sonic:	None
IMU:	• Xsens MTi
GNSS:	• RTKGps (based on Trimble BD982 receiver)
Connectivity:	None
Compute:	• Dell Precision R5500 with 2 Xeon X5690 six-core 3.4 GHz and NVIDIA GeForce GTX-1030 • Indication that multiple computers exist, but not clearly described
Power System	• Usage of 330 Volts battery of electric powertrain for AD-System (converted)
Other:	None
Software:	
Localization:	• Localization on offline map using point-clouds, GNSS data, roll, pitch and yaw data
Detection:	• Moving obstacles, lane and traffic light detection
Prediction:	• Not described
Mission:	• Not described
Behavior:	• Plans the next goal and suggests a path to it, provided by the path planner • Path planner provides a path to the goal, provided by behavior
Motion:	• Motion planner creates a trajectory to the goal, taking obstacles into account • Obstacle avoider evaluates the path, created by the motion planner with regards to obstacles
Control:	• FordEscape module takes care of actuation in order to realize the path from the obstacle avoider module
Middle Layer:	
• Not mentioned	
Other Remarks:	
• none	

TiEV (Tongji University, 2017)	
System:	
Base Vehicle:	SAIC Motors, Rowe E50
Camera:	• 7 Cameras ◦ 2 x in front configured as stereo vision ◦ 1 x in front for detection ◦ 4 x fisheye for top-down panorama view
Lidar:	• Velodyne HDL-64 • SICK LMS 511 to complement the blind area • IBEO Lux4 to complement the blind area
Radar:	None
Ultra-Sonic:	None
IMU:	precision system combining Novatel simpak6 GPS and Oxts RT2000
GNSS:	
Connectivity:	None
Compute:	• 2 x Advanced Industrial PCs • 1 Industrial PC for Velodyne lidar - NVIDIA Jetson Tx2 for deep learning and connection to motor with CAN
Power System	Not explained
Other:	• Motor can be controlled via CAN • Lidars connected with Ethernet • Cameras connected with USB and Ethernet • Actuation via CAN • PCs and Jetson connected via Ethernet and switch
Software:	
Localization:	• Offline map and creation of real-time map
Detection:	• Explicitly cars, bicycles, pedestrians and traffic signs
Prediction:	• Prediction of movements of cars, bicycles and pedestrians
Mission:	• Not described
Behavior:	• Not clearly described
Motion:	• 1st planning provides lane level path, based on HD maps coming from open-source spatial database • 2nd planning will further refine and follow
Control:	• Not described
Middle Layer:	
• ZeroCM/LCM Middleware	
Other Remarks:	
• First fail-op measures by implementing heart beat	

ZMP Robocar HV 2015	
System:	
Base Vehicle:	Toyota Prius
Camera:	- Multiple Point Grey Ladybug 5 - Multiple Grasshopper 3
Lidar:	- Multiple Velodyne Lidars - Multiple Ibeo Lidars - Multiple Hokuyo Lidars
Radar:	n/a
Ultra-Sonic:	n/a
IMU:	Javad RTK
GNSS:	
Connectivity:	n/a
Compute:	Multiple Industrial Computers
Power System	n/a
Other:	Software Stack Autoware
Software:	
Localization:	See content Autoware https://www.autoware.org/
Detection:	
Prediction:	
Mission:	
Behavior:	
Motion:	
Control:	
Middle Layer:	
ROS in combination with Autoware	
Other Remarks:	

Deeva 2015	
System:	
Base Vehicle:	Audi A4 2.0T FWD
Camera:	13 IDS-imaging Stereo cameras (type UI-5242LE9
Lidar:	- One eight-layer Lux8L in front bumper - Two four-layer Lux HD each side of the front bumper - LuxHD middle of rear bumper - Connected with CAN and Ethernet

Radar:	• n/a
Ultra-Sonic:	• n/a
IMU:	Oxford Technical solutions RT-3040
GNSS:	
Connectivity:	Novatel GNSS+L-Band high performance antenna
Compute:	• 17 Computers (Commel LS-576a, i7-3740QM, 2,760 Ghz, 6MB cache and 8GB RAM)
Power System	• n/a
Other:	• System health sensors with CAN amperometers to monitor the current on diverse positions • Safety feature: Stop the vehicle remotely
Software:	
Localization:	• n/a
Detection:	• n/a
Prediction:	• n/a
Mission:	• n/a
Behavior:	• n/a
Motion:	• n/a
Control:	• n/a
Middle Layer:	
• n/a	
Other Remarks:	
• Very little to no description about the autonomous software	

GISA 2015	
System:	
Base Vehicle:	
Camera:	One AVT Stingray F-033C (60fps)
Lidar:	- One ibeo LUX - One Velodyne HDL 32E
Radar:	n/a
Ultra-Sonic:	n/a
IMU:	SbgIG-500N IMU
GNSS:	Septentrio AsteRx2eH
Connectivity:	n/a
Compute:	
Power System	
Other:	
Localization:	GPS Based Localization
Detection:	Road, Obstacles and Pedestrian Detection
Prediction:	n/a
Mission:	n/a
Behavior:	n/a
Motion:	n/a
Control:	n/a
Middle Layer:	
Robots Operating System ROS	
Other Remarks:	
Vehicle was not capable to drive autonomously. It was made to test the above mentioned AD-Algorithms	

A1 2012	
System:	
Base Vehicle:	Hunday Tucson ix
Camera:	- Multiple Mono Color Cameras - One Color Camera
Lidar:	Eight Laser Scanners
Radar:	n/a
Ultra-Sonic:	n/a
IMU:	One IMU installed
GNSS:	RTK-GPS and DGPS
Connectivity:	n/a
Compute:	n/a
Power System	n/a
Other:	n/a
Localization:	GPS Based Localization with RTK-GPS
Detection:	Passengers, traffic lights, parking location signs, crosswalks, moving objects.
Prediction:	Moving Object prediction using the integrated probabilistic data association filter
Mission:	Route was given by the competition organizers
Behavior:	n/a
Motion:	Proprietary path planning algorithms
Control:	- Two controllers, speed and steering - Input from Motion Planner
Middle Layer:	
Other Remarks:	

8.7 Signal-oriented E/E Architecture

While for this work a SOA was selected, still signal oriented E/E architectures are dominant in the automotive domain. Same as for the SOA the signal oriented E/E Architectures will be explained along the 5+1 Model of Kruchten.

Logical View:

Definition 8.12: *The logical view describes the realization of functions with logical E/E Components including the information flow in between.*

The logical view contains logical E/E Components, which typically are logical sensors, logical functions and logical actuators. A logical sensor gathers data and forwards it to a logical function. A logical function processes the data and may send the result to a logical actuator. The actuator translates the information into a physical actions.

Definition 8.13: *Mapping describes the process of bringing an E/E Component from one view in relation to an E/E Component of another view.*

The logical components are derived from the functional view by mapping customer (sub-) functions to logical components.

Example: A logical sensor in the automatic distance keeping system example of chapter 0 can be a radar, continuously measuring the distance to the object in front of the vehicle. A logical function can be the keep distance function which sends signals to the logical actuators such as brakes or the engine.

The realization of logical functions within a signal-oriented E/E-Architecture takes place by mapping them to software E/E Components (in the following SW-Components).

Definition 8.14: *A SW-Component represents one or multiple functions as set of data and instructions which can be processed by a MCU.*

An example is depicted in Figure 75. Even though the information flow is part of the logical view, the realization within a signal-oriented E/E-Architecture takes place in the physical view.

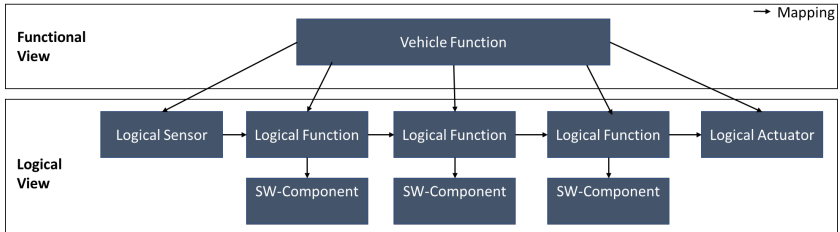


Figure 75: Logical view of signal-oriented E/E-Architecture

Physical View:

In the physical layer, the SW-Components are mapped to ECUs. The core of a signal-based architecture is the communication pattern. Therefore, the information which is shared between SW-Components are defined as signals. Depending on the size and structure of the information, one or multiple signals are used. In the next step they are aggregated to messages which are tagged with dedicated identifiers. Messages are then sent from one ECU to others via a traditional bus. A receiving ECU can identify a relevant message by the identifier and then extract the relevant signals for further processing in its local SW-Components.

	Message	Signal	Description	ECU 1	ECU 2	...
ECU 1 (ABS)	ID 100	1,2	Rotation Speed Rear Left	S	R	...
	ID 101	3,4	Rotation Speed Rear Right	S	R	...
ECU 2 (Engine)	ID 200	1	Rotation Speed Engine	R	S	...
	ID 201	2	Throttle		S	...
...	...	..	...	...	...	S

S = Sender R = Receiver

Figure 76: Indication of communication matrix

The above-mentioned communication relationships are aggregated in the communication matrix (c-matrix), indicatively depicted in Figure 76. It describes the relationship between the messages (and its signals) of each sending ECU to the corresponding receiving ECU.

The c-matrix is specified in the system design phase of the V-Shape model [17]. It is deeply integrated into the ECU, more precisely in its firmware [28]

what is indicatively shown in Figure 77. This deep integration is making it relevant for the whole lifecycle of the vehicle, thereby setting its static character.

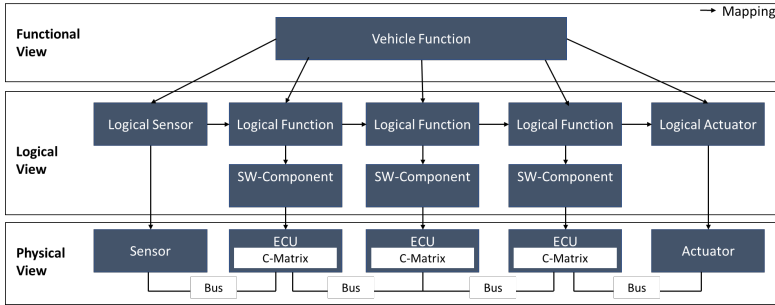


Figure 77: Physical view of a signal-oriented E/E Architecture

The increasing numbers of E/E Components brought up the question, how to connect them in order to efficiently use the restricted bandwidth of the bus systems. Figure 78 depicts the domain layout which has turned out to be an efficient solution [22].

Definition 8.15: *A domain represents a set of ECUs allocated to the same sub net and containing similar function.*

Herein the E/E Components are allocated to domains with similar functions such as powertrain, body or chassis [124] and they are connected with one central gateway. The communication within a domain does not affect the bandwidth of another domain.

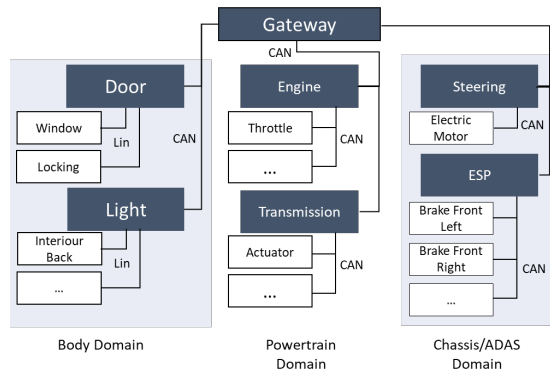


Figure 78: Domain layout

Signal-based architectures suffer from two limitations. First, the static nature by the deep integration of communication relationships into the firmware makes it hard to adjust or scale functions in the field as required for software products in the course of agile development. Second it was made for traditional bus systems which suffer from insufficient bandwidth with respect to modern applications [29].

8.8 List of Figures

Figure 1: Academic (upper row) and industrial (lower row) research vehicles.....	3
Figure 2: Schematic of an AD-System.....	6
Figure 3: Autonomous driving closed loop.....	7
Figure 4: From microcontroller (left) to system on a chip (right).....	11
Figure 5: Schematic of a core based on Streichert and Traub [22]	13
Figure 6: Scheduler process states based on Wolf [17].....	14
Figure 7: Bus (left), ring (middle) and star (right) topology schematics.....	16
Figure 8: Based on the OSI-Model [128]	18
Figure 9: Decomposition of a customer function	20
Figure 10: Schematic of a SOA.....	20
Figure 11: The ROS2 layers	22
Figure 12: Summary of safety elements for automotive E/E Architectures	27
Figure 13: Triple modular redundancy as static redundancy pattern	28
Figure 14: Scheme to distinguish fault tolerance regimes based on Stolte et al. [41]	29
Figure 15: Time intervals of fault handling based on ISO26262 [42].....	33
Figure 16: Simple Neural Network (left), Deep Neural Network (right) based on Wuttke [51].....	34
Figure 17: Comparison hypervisors vs. containers based on Morabio et al. [56].....	37
Figure 18: Schematic of an AD ECU	42
Figure 19: Modules of a typical AD software stack	44
Figure 20: Exemplary data flow between the modules of the AD software stack	45

Figure 21: Exemplary mapping of applications to modules to ECUs	46
Figure 22: Duo-Duplex Architecture based on Schnellbach [47]	60
Figure 23: Quadruplex (left) and Triplex-Triplex (right) based on Schnellbach [47]	61
Figure 24: Lightweight V-Shape model for on-demand TMR system development	65
Figure 25: The on-demand TMR system.....	73
Figure 26: On-demand TMR as State Machine.....	75
Figure 27: Distributed voting.....	76
Figure 28: On-Demand TMR Process View	78
Figure 29: On-demand TMR activation	79
Figure 30: Reconfiguration of the system	81
Figure 31: Recovered operation of the on-demand TMR system	82
Figure 32: Structured approach for test case generation	88
Figure 33: Process diagram of nominal state	90
Figure 34: Process diagram on on-demand TMR state	92
Figure 35: Process diagram on reconfigured state.....	93
Figure 36: Process diagram on recovered state	94
Figure 37: System test case overview	95
Figure 38: Stepwise System Realization	102
Figure 39: Deployment schematic for system testing	103
Figure 40: Test bench for Software in the Loop testing (STCs and FAT ₁)	104
Figure 41: Average voting durations for V_2 for STC ₁	105
Figure 42: Average voting durations for V_2 for STC ₂	106
Figure 43: Voting durations in advanced testing for V_2 in 25MB/s	107
Figure 44: Voting durations of V_2 in advanced testing for Sample 1 - 1000.....	108

Figure 45: Network traffic in advanced testing based on Wireshark recording	109
Figure 46: Triplication of Ethernet traffic leading to network saturation	111
Figure 47: Average voting duration for V_1 for STC_1	113
Figure 48: Average voting duration for V_2 for STC_2	114
Figure 49: On-demand durations for STC_1 and STC_2 as in $TC_{2.2}$ and $TC_{2.3}$	116
Figure 50: Activation behavior for STC_1 at 5Hz	118
Figure 51: Activation behavior for STC_2 at 10MB/s	119
Figure 52: Optimized activation behavior for STC_2 at 10MB/s	119
Figure 53: Summary of average voting durations and FHTI per test case	121
Figure 54: Deployment schematic for FAT_1 testing	123
Figure 55: Lane markings as output of RoLand	124
Figure 56: Position of two Leopard cameras above windshield on the truck	125
Figure 57: Overlapping areas of cameras [97]	126
Figure 58: Features and cropping area [97]	126
Figure 59: Simplified schematic of enhanced system set-up for FAT_2 testing	128
Figure 60: Voting durations for FAT_1 compared to 10MB/s STC_2	129
Figure 61: Comparison voting durations FAT_1 to FAT_2	130
Figure 62: Exemplary differing calculation times of RoLand in iteration ₁₉₁	131
Figure 63: Calculation duration deltas RoLand' vs. RoLand(') for FAT_1	132
Figure 64: Schematic of time synchronization	135

Figure 65: On-demand durations for FAT_1 and FAT_2 as in $TC_{2.2}$ and $TC_{2.3}$	137
Figure 66: Comparison of voting duration between STC and FAT.....	138
Figure 67: Comparison of FHTI between STC and FAT	140
Figure 68: Distance-Time Diagram for decelerated motion	144
Figure 69: Full brake scenario of a passenger car	145
Figure 70: Optimized mapping of AD modules to ECU SoC and safety MCU	147
Figure 71: On-demand TMR performance comparison to the state of the art	149
Figure 72: Compliance to requirements.....	156
Figure 73: Reinforcement learning agent model.....	193
Figure 74: Decision Process	195
Figure 75: Logical view of signal-oriented E/E-Architecture	213
Figure 76: Indication of communication matrix	213
Figure 77: Physical view of a signal-oriented E/E Architecture	214
Figure 78: Domain layout	215

8.9 List of Tables

Table 1: Data rates of automotive bus systems based on Wolf [17].....	16
Table 2: Random Hardware Failure Target Values based on Das and Tylor [44]	31
Table 3: Comparison sensor characteristics adjusted from [58]	40
Table 4: Summary of the state of science	64
Table 5: Architecture realization alternatives	70
Table 6: Comparison ROS2 vs. Adaptive AUTOSAR	84
Table 7: Data integrity checking technologies.....	85
Table 8: Comparison of reconfiguration technologies	87
Table 9: Split of voting durations for 25MB/s advanced testing in ms	112
Table 10: Sample sizes for test cases.....	116
Table 11: Calculation durations for FAT_1 in ms, first iterations of an instance in red	133
Table 12: Voting durations for STC_1 for TC_1 in ms	166
Table 13: Voting durations for STC_2 for TC_1 in ms	167
Table 14: Durations for identification of faulty channel for STC_1 for $TC_{2.2}$ in ms.....	168
Table 15: Durations for reconfiguration of App'' for STC_1 for $TC_{2.3}$ in ms	169
Table 16: Durations for identification of faulty channel for STC_2 for $TC_{2.2}$ in ms.....	170
Table 17: Durations for reconfiguration of App'' for STC_2 for $TC_{2.3}$ in ms	171
Table 18: Voting durations for STC_1 for TC_3 in ms.....	171
Table 19: Voting durations for STC_2 for TC_3 in ms.....	172
Table 20: Voting durations for STC_1 for TC_4 in ms	173

Table 21: Voting durations for STC_2 for TC_4 in ms.....	174
Table 22: Voting durations for FAT_1 for TC_1 in ms	175
Table 23: Voting durations for FAT_2 for TC_1 in ms	175
Table 24: Durations for identification of faulty channel for FAT_1 and FAT_2 in ms	176
Table 25: Durations for reconfiguration of App'' for FAT_1 and FAT_2 in ms.....	177
Table 26: Voting durations for FAT_1 for TC_3 in ms	178
Table 27: Voting durations for FAT_2 for TC_3 in ms	178
Table 28: Voting durations for FAT_1 for TC_4 in ms	179
Table 29: Voting durations for FAT_2 for TC_4 in ms	179
Table 30: Sensor adequacy for relevant types of information according to [50]	182
Table 31: Search String.....	194

8.10 References

- [1] C. Urmson *et al.* "Autonomous driving in urban environments: Boss and the Urban Challenge" in *Experience from the DARPA Urban Challenge*, C. Rouff and M. Hinchey, Eds., London: Springer London, 2012, pp. 425–466.
- [2] E. Shi, T. M. Gasser, A. Seeck, and R. Auerswald "The Principles of Operation Framework: A Comprehensive Classification Concept for Automated Driving Functions" *SAE Intl. J CAV*, vol. 3, no. 1, 2020, doi: 10.4271/12-03-01-0003.
- [3] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, On-Road Automated Driving (ORAD) committee, 400 Commonwealth Drive, Warrendale, PA, United States.
- [4] Manager Magazin, *Daimler holt sich Waymo-Roboter ans Lkw-Steuer*. [Online]. Available: <https://www.manager-magazin.de/unternehmen/tech/daimler-trucks-kooperation-mit-waymo-fuer-entwicklung-autonomer-lkw-a-35eb1ae6-60c6-4f01-ae6b-9f2a68525155> (accessed: Jan. 5 2024).
- [5] Aurora LLC, "Aurora Investor Presentation - March 2022" [Online]. Available: https://d1io3yog0oux5.cloudfront.net/_c010bb4c608bc7c43b88775aec9fcaaa/aurora/db/856/7486/pdf/Aurora+Investor+Presentation_March+2022_.pdf.
- [6] C. Berger and M. Dukaczewski, "Comparison of Architectural Design Decisions for Resource-Constrained Self-Driving Cars – A Multiple Case-Study" in *Informatik 2014: Big Data - Komplexität meistern*; 22. - 26. September 2014 in Stuttgart, Deutschland, 2014, pp. 2157–2168.
- [7] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and a. Q. A. Chen, "A comprehensive study of autonomous vehicle bugs" in *2020 ACM/IEEE 42nd International Conference on Software Engineering: ICSE 2020 : 27 June-19 July 2020, Seoul, South Korea : proceedings*, Seoul South Korea, 2020, pp. 385–396. Accessed: Jan. 18 2022.

-
- [8] K. A. Olsen, "The Case Against Autonomous Cars," *IEEE Potentials*, vol. 39, no. 1, pp. 25–28, 2020, doi: 10.1109/MPOT.2017.2744678.
 - [9] R. Riess, *Uber self-driving car test driver pleads guilty to endangerment in pedestrian death case*. [Online]. Available: <https://edition.cnn.com/2023/07/29/business/uber-self-driving-car-death-guilty/index.html>
 - [10] A. Roy, *How GM's Cruise robotaxi tech failures led it to drag pedestrian 20 feet*. [Online]. Available: <https://www.reuters.com/business/autos-transportation/how-gms-cruise-robotaxi-tech-failures-led-it-drag-pedestrian-20-feet-2024-01-26/>.
 - [11] Korosec K. "Cruise recalls entire fleet after robotaxi ran over, dragged pedestrian." [Online]. Available: <https://techcrunch.com/2023/11/08/cruise-recalls-entire-fleet-after-robotaxi-ran-over-dragged-pedestrian/>
 - [12] H. Jin and A. Roy, "Waymo arson in San Francisco sparks new debate on self-driving cars." [Online]. Available: <https://www.reuters.com/business/autos-transportation/san-francisco-waymo-arson-sparks-fresh-debate-self-driving-cars-2024-02-13/>
 - [13] Deutscher Verkehrssicherheitsrat, "Vision Zero: Grundlagen & Strategien" *Schriftenreihe-Verkehrssicherheit-16*, 2012.
 - [14] R. Johansson, "Vision Zero – Implementing a policy for traffic safety," *Safety Science*, vol. 47, no. 6, pp. 826–831, 2009, doi: 10.1016/j.ssci.2008.10.023.
 - [15] E. Marti, M. A. de Miguel, F. Garcia, and J. Perez "A Review of Sensor Technologies for Perception in Automated Driving" *IEEE Intell. Transport. Syst. Mag.*, vol. 11, no. 4, pp. 94–108, 2019, doi: 10.1109/MITS.2019.2907630.
 - [16] M. Staron, *Automotive Software Architectures: An Introduction*: Springer, 2021.
 - [17] Wolf, *Fahrzeuginformatik*, Wiesbaden: Springer Fachmedien Wiesbaden, 2018.
 - [18] A. Vetter, P. Obergfell, H. Guissouma, D. Grimm, M. Rumez, and E. Sax, "Development Processes in Automotive Service-oriented Architectures" in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, Jun. 2020, pp. 1–7.

- [19] P. Obergfell, S. Kugele, and E. Sax "Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Munich, Germany, Sep. 2019 - Sep. 2019, pp. 128–138.
- [20] M. Hillenbrand, *Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen*. Zugl.: Karlsruhe, KIT, Diss., 2011. Karlsruhe: KIT Scientific Publishing, 2012. Accessed: Oct. 22, 2021.
- [21] W. Shi and L. Liu, *Computing Systems for Autonomous Driving*. Cham: Springer International Publishing AG, 2021. Accessed: Feb. 18, 2022.
- [22] T. Streichert and M. Traub, *Elektrik/Elektronik-Architekturen im Kraftfahrzeug*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
- [23] M. Pohnl, A. Tamisier, and T. Blass "A Middleware Journey from Microcontrollers to Microprocessors " in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2022, pp. 282–286.
- [24] F. Reghenzani, G. Massari, and W. Fornaciari, "The Real-Time Linux Kernel " *ACM Comput. Surv.*, vol. 52, no. 1, pp. 1–36, 2020, doi: 10.1145/3297714.
- [25] J. Cowley "Communications and Networking", London: Springer, 2013.
- [26] P. Obergfell "Entwurfsmethodik für hybride Software- und Systemarchitektur", Karlsruher Institut für Technologie (KIT).
- [27] P. Obergfell, S. Kugele, C. Segler, A. Knoll, and E. Sax "Continuous Software Engineering of Innovative Automotive Functions: An Industrial Perspective" in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany, Mar. 2019, pp. 127–128.
- [28] S. Frohn and F. Ree "Vom Signal zum Service: Herausforderungen für das Entwickeln von AUTOSAR-Adaptive-Anwendungen" 2018. [Online]. Available: https://assets.vector.com/cms/content/know-how/_technical-articles/Ethernet_AUTOSAR_Adaptive_Elektronik_Automotive_201803_PressArticle_DE.pdf.

-
- [29] H. F. Stoll "Die (re-)konfigurierbare Fahrzeugarchitektur," Dissertation, 2021.
 - [30] F. Oszwald, "Dynamische Rekonfigurationsmethodik für zuverlässige, echtzeitfähige Eingebettete Systeme in Automotive", Dissertation, 2021.
 - [31] AUTOSAR "Specification on SOME/IP Transport Protocol: Document Identification No: 809".
 - [32] AUTOSAR "Specification of Communication Management: Document Identification No: 717".
 - [33] J. Becker, M. Sagar, and D. Pangercic "A Safety-Certified Vehicle OS to Enable Software-Defined Vehicles" in *Automatisiertes Fahren 2021: Vom assistierten zum autonomen Fahren*, T. Bertram, Ed., Wiesbaden: Springer Fachmedien Wiesbaden, 2021.
 - [34] M. Giordano "Transparent access to local,edge,cloud services - the autonomous driving use case", Masterthesis, POLITECNICO DI TORINO, 2021.
 - [35] M. Schindewolf et al. "A Comparison of Architecture Paradigms for Dynamic Reconfigurable Automotive Networks" in *2022 International Conference on Connected Vehicle and Expo (ICCVE)*, Lakeland, FL, USA, 372022, pp. 1–7.
 - [36] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proceedings of the 13th International Conference on Embedded Software*, Pittsburgh Pennsylvania, 01.10.2016- 07.10.2016, pp. 1–10.
 - [37] K. Ehtle "Fehlertoleranzverfahren", Berlin: Springer, 1990.
 - [38] E. Dubrova "Fault-Tolerant Design", New York: Springer, 2013.
 - [39] B. Johnson "Fault-Tolerant Microprocessor-Based Systems" *IEEE Micro*, vol. 4, no. 6, pp. 6–21, 1984, doi: 10.1109/MM.1984.291277.
 - [40] R. Isermann "Mechatronische Systeme: Grundlagen", 2nd ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

- [41] T. Stolte *et al.* "A Taxonomy to Unify Fault Tolerance Regimes for Automotive Systems: Defining Fail-Operational, Fail-Degraded, and Fail-Safe", *IEEE Transactions on Intelligent Vehicles*, p. 1, 2021, doi: 10.1109/TIV.2021.3129933.
- [42] ISO "ISO 26262-1:2018"
- [43] B. Sari "Fail-operational Safety Architecture for ADAS/AD Systems and a Model-driven Approach for Dependent Failure Analysis", Dissertation, Wiesbaden.
- [44] N. Das and W. Taylor "Quantified fault tree techniques for calculating hardware fault metrics according to ISO 26262" in *2016 IEEE Symposium on Product Compliance Engineering (ISPC)*, Anaheim, CA, USA, 2016, pp. 1–8.
- [45] ISO "ISO 26262-9:2018"
- [46] Y. Fu, A. Terechko, J. F. Groote, and A. K. Saberi "A Formally Verified Fail-Operational Safety Concept for Automated Driving" *SAE Intl. J CAV*, vol. 5, no. 1, pp. 7–21, 2022, doi: 10.4271/12-05-01-0002.
- [47] A. Schnellbach "Fail-operational automotive systems", Dissertation, Graz University of Technology.
- [48] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu "A survey of deep learning techniques for autonomous driving" *J. Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020, doi: 10.1002/rob.21918.
- [49] T. Kaffka "*Neuronale Netze - Grundlagen: Mit Beispielprogrammen in Java*", 1st ed. Frechen: mitp, 2017.
- [50] S. Dörn "*Programmieren für Ingenieure und Naturwissenschaftler*" Berlin, Heidelberg: Springer, 2018.
- [51] L. Wuttke "*Deep Learning: Definition, Beispiele & Frameworks*", [Online]. Available: <https://datasolut.com/was-ist-deep-learning/#deep-learning-Einfuehrung> (accessed: Dec. 22 2023).
- [52] G. Reddy, A. Narayana, P. Keerthan, B. Vineetha, and P. Honnavalli "Multiple Hashing Using SHA-256 and MD5" in *Lecture Notes in Electrical Engineering 735, Advances in computing and network communications: Proceedings of CoCoNet 2020*, Singapore: Springer, 2021.

- [53] D. Wätjen "Kryptographie", Wiesbaden: Springer Fachmedien Wiesbaden, 2018.
- [54] S. Ghoshal, P. Bandyopadhyay, S. Roy, and M. Baneree "A Journey from MD5 to SHA-3" in vol. 99, *Trends in Communication, Cloud, and Big Data*, Singapore: Springer, 2020, pp. 107–112.
- [55] N. Ayres, L. Deka, and B. Passow "Virtualisation as a Means for Dynamic Software Update within the Automotive E/E Architecture" in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, Leicester, United Kingdom, 82019, pp. 154–157.
- [56] R. Morabito, J. Kjallman, and M. Komu "Hypervisors vs. Lightweight Virtualization: A Performance Comparison" in *2015 IEEE International Conference on Cloud Engineering*, Tempe, AZ, USA, 32015, pp. 386–393.
- [57] D. Bernstein "Containers and Cloud: From LXC to Docker to Kubernetes" *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, 2014, doi: 10.1109/MCC.2014.51.
- [58] L. Liu *et al.* "Computing Systems for Autonomous Driving: State of the Art and Challenges" *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6469–6486, 2021, doi: 10.1109/JIOT.2020.3043716.
- [59] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot "Computer Architectures for Autonomous Driving", *Computer*, vol. 50, no. 8, pp. 18–25, 2017, doi: 10.1109/MC.2017.3001256.
- [60] C. Badue *et al.* "Self-Driving Cars: A Survey", 2019.
- [61] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh "Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review" *Sensors (Basel, Switzerland)*, vol. 21, no. 6, 2021, doi: 10.3390/s21062140.
- [62] R. Tyagi "Sicheres automatisiertes Fahren mit AURIX™ Mikrocontrollern von Infineon und DRIVE PX Computing-Plattform von NVIDIA: 2017" [Online]. Available: <https://www.infineon.com/cms/de/about-infineon/press/market-news/2017/INFATV201710-002.html>

- [63] NVIDIA Developer, *NVIDIA DRIVE AGX Developer Kit*. [Online]. Available: <https://developer.download.nvidia.com/driveworks/secure/docs/nvidia-drive-platform-for-developers.pdf?xe6NjavxWhqtB-Klr5GLE6pmsrUekGNX8IXoCjj1dbOxMM4O-LyR5JTLb2Jre6BiMnXkDN7cJzW53X1ddqHkzv9Pztpkm7Xo-chAdB6S7T05xBttihW8CvI52l-qSyoyu7SRXi5J4yiOeXFQ98jld-sik0Xueqb6zXpL7zI4Ri2&t=eyJscyl6ImdzZW8iLCJsc2QiOiJodH-RwcZovL3d3dy5nb29nbGUuZGUvIn0=> (accessed: 12.2023).
- [64] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles" *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016, doi: 10.1109/TIV.2016.2578706.
- [65] S. Behere and M. Törngren "A functional reference architecture for autonomous driving" in *Information and Software Technology*, vol. 73, pp. 136–150, 2016, doi: 10.1016/j.infsof.2015.12.008.
- [66] Y. Dajsuren and M. van den Brand "Automotive Systems and Software Engineering", Springer International Publishing, 2019.
- [67] V. M. Raju, V. Gupta, and S. Lomate "Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo" in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, Bombay, India, Mar. 2019 - Mar. 2019, pp. 1–5.
- [68] M. Goebl and G. Farber "A Real-Time-capable Hard-and Software Architecture for Joint Image and Knowledge Processing in Cognitive Automobiles" in *Proceedings of 2007 IEEE Intelligent Vehicles Symposium*, Jun. 2007, pp. 734–740.
- [69] A. S. Huang, E. Olson, and D. C. Moore "LCM: Lightweight Communications and Marshalling" in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, 2010, pp. 4057–4062.
- [70] A. Broggi et al. "PROUD—Public Road Urban Driverless-Car Test" *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3508–3519, 2015, doi: 10.1109/TITS.2015.2477556.

- [71] J. Zhao *et al.* "TiEV: The Tongji Intelligent Electric Vehicle in the Intelligent Vehicle Future Challenge of China" in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, USA, Nov. 2018, pp. 1303–1309.
- [72] F. Giaimo, C. Berger, and C. Kirchner "Considerations About Continuous Experimentation for Resource-Constrained Platforms in Self-driving Vehicles" in *11th European Conference on Software Architecture (ECSA 2017)*, Canterbury, United Kingdom, 2017, pp. 84–91.
- [73] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye "Object Detection in 20 Years: A Survey", 3. May. 2019. [Online]. Available: <http://arxiv.org/pdf/1905.05055v2>
- [74] N. O. Mahony *et al.* "Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics" in *Proceedings of the 2019 Computer Vision Conference (CVC)*, Las Vegas, USA, 2019. Accessed: Mar. 1 2022.
- [75] K. Arai and S. Kapoor "Advances in Computer Vision" vol. 943, 2020, doi: 10.1007/978-3-030-17795-9.
- [76] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety" in *15th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2009*, San Francisco, CA, USA, 2009, pp. 99–107. Accessed: Mar. 28 2022.
- [77] F. K. Bapp, T. Dörr, T. Sandmann, F. Schade, and J. Becker "Towards Fail-Operational Systems on Controller Level Using Heterogeneous Multicore SoC Architectures and Hardware Support" in *SAE Technical Paper Series*, 2018. Accessed: Mar. 28 2022.
- [78] D. Niedballa and H.-C. Reus "Concepts of functional safety in E/E-architectures of highly automated and autonomous vehicles" in *20. Internationales Stuttgarter Symposium: Automobil- und Motorentechnik*, Stuttgart, Germany, 2020.
- [79] T. Wooten *et al.* "UNICARagil - Disruptive modulare Architektur für agile, automatisierte Fahrzeugkonzepte" in *27th Aachen Colloquium Automobile and Engine Technology 2018*, Aachen, Germany, 2018.

- [80] S. Orlov, M. Korte, F. Oszwald, and P. Vollmer "Automatically reconfigurable actuator control for reliable autonomous driving functions (AutoKonf)" in *10th International Munich Chassis Symposium 2019*, Wiesbaden: Springer Fachmedien Wiesbaden, 2020.
- [81] E-Gas Working Group "Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units: Version 6.0," 2015.
- [82] T. Bijlsma *et al.* "A Distributed Safety Mechanism using Middleware and Hypervisors for Autonomous Vehicles" in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 32020, pp. 1175–1180.
- [83] M. Li and L. Eckstein "Fail-Operational Steer-By-Wire System for Autonomous Vehicles" in *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Cairo, Egypt, 2019. Accessed: May 11 2022.
- [84] T. Stolte *et al.* "Towards Safety Concepts for Automated Vehicles by the Example of the Project UNICARagil" in *29th Aachen Colloquium Sustainable Mobility 2020*, Aachen, Germany, 2020.
- [85] S. Kugele, D. Hettler, and J. Peter "Data-Centric Communication and Containerization for Future Automotive Software Architectures" in *2018 IEEE International Conference on Software Architecture (ICSA)*, Seattle, WA, Apr. 2018 - May. 2018, pp. 65–6509.
- [86] S. Kugele, D. Hettler, and S. Shafaei "Elastic Service Provision for Intelligent Vehicle Functions" in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, USA, 2018, pp. 3183–3190.
- [87] F. Oszwald, P. Obergfell, B. Liu, V. Pazmino Betancourt, and J. Becker "Model-Based Design of Service-Oriented Architectures for Reliable Dynamic Reconfiguration" in *SAE Technical Paper Series*, 2020.
- [88] B. Liu, V. P. Betancourt, Y. Zhu, and J. Becker "Towards an On-Demand Redundancy Concept for Autonomous Vehicle Functions using Microservice Architecture" in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, 2020.

- [89] M. Schindewolf, D. Grimm, C. Lingor, and E. Sax "Toward a Resilient Automotive Service-Oriented Architecture by using Dynamic Orchestration" in *2022 IEEE 1st International Conference on Cognitive Mobility (CogMob)*, Budapest, Hungary, 2022, pp. 147–154.
- [90] F. Kempf, T. Hartmann, S. Baehr, and J. Becker "An Adaptive Lockstep Architecture for Mixed-Criticality Systems" in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Tampa, FL, USA, 2021, pp. 7–12.
- [91] H. Flühr "Avionik und Flugsicherungstechnik", Springer Berlin Heidelberg, 2012.
- [92] H. Blair-Smith "Space shuttle fault tolerance: Analog and digital teamwork" in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, Orlando, FL, USA, 2009, 6.B.1-1-6.B.1-11.
- [93] Y. C. Yeh "Triple-triple redundant 777 primary flight computer" in *1996 IEEE Aerospace Applications Conference*, Aspen, CO, USA, 1996, pp. 293–307. Accessed: Mar. 29 2022.
- [94] J. Henle, M. Stoffel, M. Schindewolf, A.-T. Nagele, and E. Sax "Architecture platforms for future vehicles: a comparison of ROS2 and Adaptive AUTOSAR" in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, Macau, China, 2022, pp. 3095–3102.
- [95] D. Mathew "Dynamic Reconfiguration Capabilities of ROS2 Managed Nodes in a Distributed System" Master Thesis, Dept. of Computer Science, Chair of Computer Engineering, Chemnitz Institute of Technology, Germany, 2023.
- [96] eprosima "10.3. DomainParticipant profiles", [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/xml_configuration/domainparticipant.html
- [97] S. Gupta "Enhancing Safety in Autonomous Vehicles: Realtime Dual Image Cropping Algorithm for Redundant Lane Detection" Master Thesis, University of Stuttgart, Germany.
- [98] M. Stoffel and E. Sax "Distributed Voters for Automotive Applications" in *2023 IEEE Intelligent Vehicles Symposium (IV)*, Anchorage, AK, USA, 2023.

- [99] G. Hebermehl, *Drive Pilot kostet wenigstens 5.950 Euro: MERCEDES MIT LEVEL-3-ZULASSUNG*. [Online]. Available: <https://www.auto-motor-sport.de/tech-zukunft/mercedes-autonom-level-3-drive-pilot-haftung-unfall/> (accessed: Dec. 11 2023).
- [100] M. Hohensee *"Mit der autonomen S-Klasse unterwegs in Los Angeles"* [Online]. Available: <https://amp2.wiwo.de/technologie/mobilitaet/mercedes-benz-mit-der-autonomen-s-klasse-unterwegs-in-los-angeles/28183356.html> (accessed: Dec. 11 2024).
- [101] S. Grundhoff *"Wir testen den Autobahn-Assistenten von BMW und der kann leider nicht viel"* [Online]. Available: https://www.focus.de/auto/ratgeber/unterwegs/haende-in-den-schoss-technik-bmw-i5-mit-autobahn-assistent-im-selbstversuch_id_209252588.html (accessed: Dec. 11 2023).
- [102] F. Greis *"Die Hand wird frei - der Blick noch nicht: Nach BMW schafft auch Ford die Freihanderkennung beim automatisierten Fahren auf der Autobahn ab. Der Komfortgewinn ist beträchtlich."* [Online]. Available: <https://www.golem.de/news/probefahrt-mit-ford-bluecruise-die-hand-wird-frei-der-blick-noch-nicht-2308-177076.html> (accessed: Dec. 11 2023).
- [103] Zöllner, H., Hugemann, W. *"Zur Problematik der Bremsreaktionszeit im Straßenverkehr"* [Online]. Available: https://www.unfallrekonstruktion.de/pdf/bdp_1998_german.pdf
- [104] ADAC *"Bremswege im Vergleich"* [Online]. Available: <https://www.adac.de/rund-ums-fahrzeug/autokatalog/autotest/bremswege-vergleich/> (accessed: Dec. 11 2024).
- [105] Bußgeldkatalog 2023 *"Abstand und Bastandsvergehen bei LKW."* [Online]. Available: <https://www.bussgeldkatalog.org/lkw-abstand/#:~:text=Lastkraftwagen%20mit%20einem%20zul%C3%A4ssigen%20Gesamtgewicht,Mindestabstand%20von%2050%20Meter%20einhalten.> (accessed: Dec. 11 2023).

-
- [106] Elektronik Kompendium "PCIe - PCI Express: Versionen 1.1 / 2.0 / 3.0 / 4.0 / 5.0 / 6.0 / 7.0." [Online]. Available: <https://www.elektronik-kompendium.de/sites/com/0904051.htm>.
- [107] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis "A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications" *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 829–846, 2018, doi: 10.1109/IIOT.2018.2812300.
- [108] S. Pendleton *et al.* "Perception, Planning, Control, and Coordination for Autonomous Vehicles" *Machines*, vol. 5, no. 1, p. 6, 2017, doi: 10.3390/machines5010006.
- [109] I. A. Barsan, S. Wang, A. Pokrovsky, and R. Urtasun "Learning to Localize Using a LiDAR Intensity Map" in *2nd Conference on Robot Learning*, Zurich, Switzerland, 2018.
- [110] J. Li, W. Zhan, Y. Hu, and M. Tomizuka "Generic Tracking and Probabilistic Prediction Framework and Its Application in Autonomous Driving" *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3634–3649, 2020, doi: 10.1109/TITS.2019.2930310.
- [111] J. K. Suhr, J. Jang, D. Min, and H. G. Jung "Sensor Fusion-Based Low-Cost Vehicle Localization System for Complex Urban Environments" *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1078–1086, 2017, doi: 10.1109/TITS.2016.2595618.
- [112] F. Dellaert, D. Fox, W. Burgard, and S. Thrun "Monte Carlo localization for mobile robots" in *Proceedings 1999 IEEE International Conference on Robotics and Automation*, 1999, pp. 1322–1328.
- [113] H. Kim, B. Liu, C. Y. Goh, S. Lee, and H. Myung "Robust Vehicle Localization Using Entropy-Weighted Particle Filter-based Data Fusion of Vertical and Road Intensity Information for a Large Scale Urban Area" *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1518–1524, 2017, doi: 10.1109/LRA.2017.2673868.
- [114] S. Thrun, W. Burgard, and D. Fox "*Probabilistic robotics*" Cambridge, London: MIT Press, 2006.

- [115] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda "A Survey of Autonomous Driving: Common Practices and Emerging Technologies" *IEEE Access*, vol. 8, pp. 58443–58469, 2020, doi: 10.1109/ACCESS.2020.2983149.
- [116] S. Lefèvre, D. Vasquez, and C. Laugier "A survey on motion prediction and risk assessment for intelligent vehicles" *ROBOMECH Journal*, vol. 1, no. 1, 2014, doi: 10.1186/s40648-014-0001-z.
- [117] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis "Deep Learning-Based Vehicle Behavior Prediction for Autonomous Driving Applications: A Review" *IEEE Trans. Intell. Transport. Syst.*, vol. 23, no. 1, pp. 33–47, 2022, doi: 10.1109/TITS.2020.3012034.
- [118] L. Kliemann and P. Sanders "Algorithm Engineering" *Algorithm Engineering*, vol. 9220, 2016, doi: 10.1007/978-3-319-49487-6.
- [119] P. Wang, S. Gao, L. Li, S. Cheng, and H. Zhao "Research on driving behavior decision making system of autonomous driving vehicle based on benefit evaluation model" *AoT*, vol. 53, no. 1, pp. 21–36, 2020, doi: 10.5604/01.3001.0014.1740.
- [120] A. Furda and Vlacic L. "Real-Time Decision Making for Autonomous City Vehicles" *Journal of Robotics and Mechatronics*, vol. 2010. [Online]. Available: https://eprints.qut.edu.au/217844/1/Furda_Vlacic_RTDecisionMaking_JournalRoboticsMechatronics2010.pdf.
- [121] D. Gonzalez, J. Perez, V. Milanés, and F. Nashashibi "A Review of Motion Planning Techniques for Automated Vehicles" *IEEE Trans. Intell. Transport. Syst.*, vol. 17, no. 4, pp. 1135–1145, 2016, doi: 10.1109/TITS.2015.2498841.
- [122] E. Huang, M. Mukadam, Z. Liu, and B. Boots "Motion Planning with Graph-Based Trajectories and Gaussian Process Inference" 2017. [Online]. Available: <https://homes.cs.washington.edu/~bboots/files/GPMP-GRAPH.pdf>.
- [123] A. E. L. Sallab, M. Abdou, E. Perot, and S. Yogamani "Deep Reinforcement Learning framework for Autonomous Driving" *Electronic Imaging*, vol. 29, no. 19, pp. 70–76, 2017, doi: 10.2352/ISSN.2470-1173.2017.19.AVM-023.

- [124] L. Braun, F. Gauterin, and E. Sax "Experteninterview zur Anforderungsanalyse heutiger und zukünftiger E/E Architekturen im Kraftfahrzeug", Abschlussbericht, 27. April 2016," 2016. Accessed: Oct. 7, 2021.
- [125] Time-Sensitive Networking (TSN) Documentation project for Linux, [Online] Available: <https://tsn.readthedocs.io/timesync.html#introduction>
- [126] TCPDUMP & LIBCAP, [Online] Available: <https://www.tcpdump.org/>
- [127] The Wireshark Foundation, [Online] Available: <https://www.wireshark.org/>
- [128] ISO/IEC 7498-1:1994 - Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model

One recent focus topic in automotive development is the move from manually driven vehicles to autonomous vehicles (AV) which no longer require human intervention. Instead, computers take over the driving task by maneuvering the AV in public areas what brings exceptional high safety requirements to the system. Additionally, AVs require the usage of machine learning algorithms which need specialized Systems on a Chips (SoCs) as processing units inside automotive Electronic Control Units (ECU).

While those SoCs are already existing, they are limited in their safety level regarding randomly occurring hardware faults. That makes them not suitable for safety critical applications in standalone operation. However, the functional safety standard ISO 26262 allows to decompose a desired safety level by combining two lower rated SoCs to jointly achieve a higher safety level.

This work proposes a novel concept which utilizes the decomposition principle by operating AV software applications redundantly on two connected ECUs. Taking avionic architectures as example, the required fault detection is realized by comparing the output of both applications using distributed voters. In case of a detected fault the system on-demand executes the faulty software application on a third generic back-up ECU again. This allows the identification of the faulty ECU and the reconfiguration of the system to stay operational.

Contrary to existing solutions the proposed approach minimizes the number of required redundant components since it does not need a fully populated third channel as in the traditional triple modular redundancy pattern. The system was tested on a test bench and in a class 8 long haul truck. It showed promising worst case fault handling durations of 3.2s, thereby contributing to the research for safe architectures for AVs.

