Felix Warmer, Bsc.

Design and Implementation of a Fail-Operational Environmental Perception System

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institut für Technische Informatik

Graz, September 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present masters thesis.

Date

Signature

Kurzfassung

In der Automobilindustrie werden Systeme zur Umgebungswahrnehmung schon seit Jahren eingesetzt. Von vorerst einfachen Wahrnehmungssystemen, wie ein Abstandsmesser für das Einparken oder einer Rückfahrkamera, wurden mit der Zunahme von Automatisierung, immer mehr sicherheitsrelevante Systeme eingebaut, wie Spuren-Assistent, Auffahrwarnung mit Abbremsfunktion, Verkehrszeichen Assistent und so weiter. Da die Automobilindustrie derzeit in die Entwicklung echter selbstfahrender Autos (SAE Level 4 und 5) investiert, wird sich dieser Trend auch fortsetzen.

Aufgrund von Platz- und Kostenbeschränkungen für selbstfahrende Autos ist es unvermeidlich, dass ein einzelnes Wahrnehmungssystem vielseitig einsetzbar sein muss, damit es verschiedenste Aufgaben in unterschiedlichen Szenarien bewältigen kann. Ein solches Wahrnehmungssystem könnte dazu verwendet werden, die steigenden Anforderungen an die funktionale Sicherheit zu bewältigen, die mit einem selbstfahrenden Auto verbunden sind.

Diese Arbeit befasst sich mit dem Entwurf und der Implementierung eines Wahrnehmungssystems. Der Schwerpunkt liegt auf 'Fail-Operational' Konzepte die in Software implementiert werden können.

Abstract

Environmental perception systems have been used in the automotive industry for years now. From the first simple perception systems, such as a distance meter for parking or a reversing camera, have been built into more and more safety-relevant systems with the increasing automation, such as lane assistant, collision warning with braking function, traffic sign assistant and so on. This trend is continuing as the automotive industry is currently invested in creating true self-driving cars (SAE Level 4 and 5). Due to space and cost restrictions for self-driving cars, it will be inevitable that a single perception system will need to be versatile such that it can handle different tasks when confronted with diverse scenarios. Such a versatility perception system could be used to handle the increasing functional safety requirements that come with a self-driving car. This thesis deals with the design and implementation of a versatile perception system with the main focus on a fail-operational concepts that can be implemented in software.

Danksagung

Diese Masterarbeit wurde im (Studien)Jahr 2019/2020 als Teil des EU-Projekts PRYSTINE[1] am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

Zuallererst möchte ich meinem Betreuer Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger für seine hilfreiche Unterstützung beim Verfassen dieser Masterarbeit danken. Von Beginn an hatte ich alle Freiheiten die ich benötigte und wenn ich unentschlossen war oder Probleme hatte, half er mir stets die richtige Richtung zu finden um meine Arbeit fortzusetzen.

Einen besonderen Dank möchte ich an DDDipl.-Ing. Andreas Strasser, BSc MA und Dipl.-Ing. Philipp Stelzer, BSc richten, die mich in gerade kniffligen Abschnitten der Arbeit und bei detaillierten Fragen hervorragend unterstützt haben. Ihre Bürotür war immer offen, wenn ich Fragen hatte oder Feedback benötigte. Durch die konstruktiven Diskussionen mit Ihnen wurden Unklarheiten beseitigt und mit vielen neuen Ideen ersetzt.

Abschließend möchte ich mich bei meinen Eltern und meinem Bruder bedanken, die mich während des gesamten Studiums unterstützt und ermutigt haben. Sie zeigten stets Verständnis und ohne sie wäre es mir nicht möglich gewesen dieses Ziel zu erreichen.

Graz, September 2020 Felix Warmer

[1] Das PRYSTINE-Projekt wurde unter der Finanzhilfevereinbarung Nr. 783190 finanziert. PRYSTINE wird vom Bundesministerium für Verkehr, Innovation und Technologie (BMVIT) zwischen Mai 2018 und April 2021 im Rahmen des Programms IKT der Zukunft gefördert (Bewilligungsnummer 865310). Weitere Informationen: https://iktderzukunft.at/en/.

Acknowledgements

This master thesis was written during the year 2019/2020 within the EU project PRYSTINE[1] at the Institute for Technical Informatics at the Graz University of Technology.

First, I want to thank my Supervisor Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger for the extensive support he provided me with, when writing this thesis. From the start he made sure that I have the room and freedom to write this thesis and when I was indecisive or had problems, he would provide me with the right guidance so that I could continue my work.
I would like to give special thanks to DDDipl.-Ing. Andreas Strasser, BSc MA and Dipl.-Ing. Philipp Stelzer, BSc who supported me through the finicky parts of the thesis and helped me if there were more detailed questions. Their office was always open whenever I had questions or needed feedback and my uncertainties were replaced with new ideas, thanks to the constructive discussions I had with them.

Finally, I want to thank my parents and my brother for their unwavering support and encouragement throughout my studies. I would not have been able to accomplish this goal if it were not for their understanding and help.

Graz, September 2020 Felix Warmer

[1] The PRYSTINE project is funded under the grant agreement number 783190. PRYSTINE is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program ICT of the Future between May 2018 and April 2021 (grant number 865310). More information: https://iktderzukunft.at/en/.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Safety has many facets. Depending on the context, the meaning of the word safety can have vastly different implication like physical safety for people and things, data safety for privacy and intellectual property, financial safety for investors, or functional safety for processes. These different types of safety are normally not isolated, instead a failure of one safety measure usually has an effect on multiple areas. As safety is often seen as a problem that can be solved with technology, it's easy to forget that a safety measure in itself can introduce another safety risk. For example, the introduction of airbags in cars has drastically decreased the number of deaths and serious injuries but malfunctioning airbags also caused some deaths and serious injuries [MKK$^+$01]. This is why modern safety standards define some form of functional safety to reduce the risk of malfunctioning behavior that are harmful for a person. A typical behavior for systems that are designed with functional safety in mind, is to detect a malfunctioning component and put it into some kind of safe state. This safe state can be as simple as deactivating the malfunctioning component but naturally a safe state should not introduce more risks to a person's health. Furthermore, if the failing components is critical for the functionality of the system it cannot be simply deactivated. Instead there needs to be some kind of redundancy implemented so that component seamlessly continues working even in the case of failure.

In the automotive industry the functional safety of electrical and electronic systems (E/E systems) gained more significance over the decades as more E/E systems are introduced into a vehicle. This trend has no sign of stopping as manufactures at large are aiming for self-driving cars. The level of driving automation that a vehicle is capable of is usually described by the six SAE levels defined by SAE international, with SAE level 0 providing no driving automation and SAE level 5 providing full driving automation [Int18]. The automotive industry at large is currently adding more and more conditional automation driving systems (SAE Level 3) to the quite established partial automation driving systems (SAE Level 2). On top of that, high driving automation (SAE Level 4) is already tested and used in highly restricted scenarios (NAVYA building Level 4 shuttles, Alphabet's Waymo self-driving taxi, ...). Such development indicates that high driving automation systems (SAE Level 4) probably will be widely used in the near future. For SAE Level 1-3 the fail-safe responsibility lies by the driver. In a SAE Level 4-5 there might not be anything a passenger could use to trigger a fail-safe response. This is why vehicles with SAE Level 4 systems need to migrate from fail-safe systems to fail-operational systems. This trend to reach higher driving automation is highly dependent on the improvements in perception systems and computation power. Perception systems are already heavily used in vehicles and they will play a more and more important role as vehicles reach higher automation levels. Since their failure can have negative effects on all components that use them, it is important to implement perception systems in a fail-operational way.

1.2 Problem Definition

In a future of self-driving vehicles, perception systems are the vehicles eyes and ears. As with a human driver, a critical failure of any part of a perception system might have serious consequence for the safety of the passengers. A perception system has to overcome multiple challenges that might lead to single-point failures, some examples would be:

Weather: Fog and rain causes not only to reduced visibility for humans and conventional camera systems but also to a reduction of accuracy in LIDAR systems. Freezing cold weather can cause the lens to freeze over. Sand and snow storm can significantly block the view of any LIDAR system. [PGM17], [RSS11]

Heat: Strong heat changes the wave length of lasers which can accumulate and reduce the accuracy of an LIDAR system. Heat also accelerates electromigration what can cause micro-cracks over time. This effect is a concern especially for small and energy efficient systems that need to run constantly and reliable for years. Constant heating and cooling over years increase the likelihood of an early dead for ICs. [LS14]

Beside the single-point failures there are residual failures that aren't so obvious. For example, stuck bits in ram cells that cause soft or hard memory failures [HSS12]. Since this is a normal issue for servers, its standard to developed hardware capable of some error correcting codes (ECC) scheme to mitigate the effects memory errors have on a system. But compared to a server where every few years a ram block gets replaced and the replacement procedure is relatively simple. A perception system in an automated vehicle has soldered memory and would need to be replaced completely.

For the migration of perception systems from fail-safe systems to fail-operational systems the aforementioned issues need to be handled adequately.

1.3 Goal

The goal of this thesis is to design and implement parts of a Fail-Operational module between a ToF camera and the application in an Environmental Perception System.

The issues that this thesis wants to handle are:

Temperature: Handle CPU utilization, frequency, and voltage such the heat output of the CPU is reduced and that for driving situations where more headroom is needed the CPU not immediately thermal-throttles. This implies that depending on the driver situation we want to change the load on the CPU. A change in CPU load should be achieved by down-scaling the image resolution or reducing the frame rate of the camera. With a reduced CPU utilization, the frequency of the CPU can be stepped down to further decrease the heat output.

Memory Errors: Detect memory cells that are failing and correct them. Frequently failing memory cells should be dropped and not used anymore. The implementation should be done in software. Furthermore, if no memory is left, reduce the data flow to ensure functionality. Data-flow can be reduced by decreasing the image resolution.

1.4 Structure

This thesis is organized in six chapters, with **Chapter 1** being the current chapter. **Chapter 2** provides a background and references to related topics that are basis to this work. **Chapter 3** takes a conceptual look at simple perception system model. In the course of the chapter the model will be extended to a fail-operational system. The extended model will be used to explain the reasoning behind some decisions and it will be used as basis for further in-depth concept discussions.

Chapter 4 discusses the implementation of the perception system. The chapter focuses on how and what got implemented, what hardware was used and what problems accrued with them and how the implementation handles certain situations.

Testing and their Results will be discussed in **Chapter 5**. The main focus in this chapter lies in the explanation of the testing methodologies and the discussion of their Results.

At last we will discuss in **Chapter 6** the final conclusion of the thesis and take a look at possible further extensions and future projects build on this thesis.

Chapter 2

Related Work

2.1 Functional safety and fail-operational

The main focus in *functional safety* is to detect *faults*, prevent, and/or control the resulting *failures*. As shown in figure 2.1 it is important that the failing operations is brought to a *safe state* before a fault ends in a *hazardous event*.

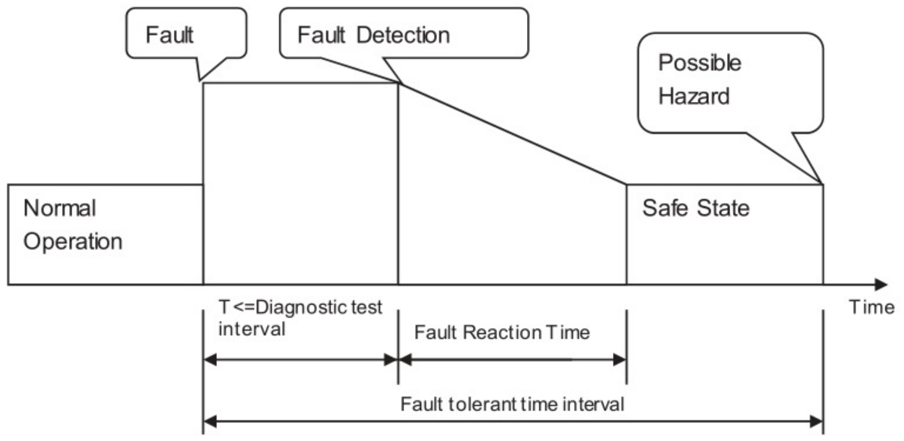


Figure 2.1: Fault tolerant time interval. [XSS+16]

The definitions for the terms used in the last few sentences can slightly differ depending on the source. Since this thesis is more connected to the automotive side of functional safety the definitions from Part 1 of the ISO Norm 26262:2018 "Road vehicles Functional safety" [ISO18a] are the ones used in this thesis. Some of the interesting definitions are [ISO18a]:

- **Functional safety:** "absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems"
- **Safe state:** "operating mode, in case of a failure, of an item without an unreasonable level of risk"

- **Hazard:** “potential source of harm caused by malfunctioning behavior of the item“
- **Fault tolerance:** “ability to deliver a specified functionality in the presence of one or more specified faults“
- **Fault:** “abnormal condition that can cause an element or an item to fail“
- **Error:** “discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition“
- **Failure:** “termination of an intended behavior of an element or an item due to a fault manifestation“

From the above definitions it is visible that fault and failure can be used recursively. If a fault occurs in a safety related product it will propagate through the system. The fault will lead to error(s) and those will lead to failure(s) which might again lead to further fault(s). The norm does not include the definition of *fail-operational* and the definition of *fault tolerance* was only recently added with the 2018 revision of the norm. With the relative new challenges of automotive driving, the automotive industry has to actively work towards industry-wide standards. This work towards industry-wide standards also applies to the safety aspects, as seen with the whitepaper *Safety First for Automated Driving*, or short SaFAD [Apt19]. The whitepaper was created by the twelve industry leaders of the automotive and component industry and summarizes widely known safety by design and verification and validation (V&V) methods for higher levels of automated driving. The definitions contained in it included the definition of *fail-operational* that was missing from the norm. The whitepaper defines [Apt19]:

- **Fail-operational:** “This refers to full & safe operations/service in the presence of hazardous events. The loss of safety-related functions or system components shall not lead to a hazard.“
- **Fail-degraded** “This means that the system is still able to operate safely when degraded“

From these definitions it is visible that fault tolerance is the basis of fail-degraded and fail-operational behavior. For example, if a fault tolerant system was designed that in the presence of a faulty component it would isolate or switch off the faulty component and continue operation with a degraded capability, the system would be a fail-degraded behaving system. This is normally known as graceful degradation. If the before mentioned fault tolerant system would replace the faulty component by a redundant backup component so that the system keeps its full and safe operation, then the system would be a fail-operational behaving system. To note is that a gracefully degrading system might first behave fail-operational and only after a certain number of failures enters a fail-degraded behavior. In such a case the component that is failing is overdesigned such that it has some headroom and is in itself redundant. So, the HW/SW-modules of a fail-operational behaving systems should continue to perform a certain set of functionalities to some predefined extent, performance, duration and for a certain amount of failures. As already mentioned, fail-operational systems often rely on redundancy to achieve this. According to Elena Dubrova book *Fault-Tolerant Design* [Dub13] redundancy methods can be categorized into two kinds of redundancies methods: *space redundancy* and *time redundant* methods. In her book *space redundancy* is described as an redundancy that “provides additional components, functions, or data items that are unnecessary for fault-free operation.“ and *time redundancy* as a redundancy where “the computation or data transmission is repeated and the result is compared to a stored copy of the previous result.“. Depending on the type of redundancy implemented in a system, space redundancy can be further classified into hardware, software, and information redundancy.

2.2 Redundancy and fault-tolerant behavior

2.2.1 Hardware redundancy

As previously mentioned, to achieve fail-operational behavior, systems rely on redundancy. To achieve hardware redundancy a system must contain at least two physical copies of the hardware component. Of course, this comes with a number of disadvantages like increased cost, size, weight, power consumption, time to design and testing effort. Depending on how a hardware redundant system handles a fault it can be grouped into *passive*, *active* and *hybrid* redundant system [Dub13].

Passive redundancy

A passive redundant system achieves fault tolerance by masking faults rather than explicit detecting them.

Two of the most common passive hardware redundancies are *Triple Modular Redundancy* (TMR) and *N-modular redundancy* (NMR). Figure 2.2 shows the basic configuration for TMR and NMR. As seen in figure 2.2 there are multiple components or modules that perform the same computation in parallel. Depending on the application, the modules can be processors, memories, network connections, power supplies and so on. A majority voting is used to determine the correct result. It should be visible form the figure that the voter is single point of failure in such systems. This is why the voter it is typically kept very simple compared to the redundant modules to reduce the probability of failure. Still there are schemes with redundant voters, for cases where a single point of failure is not acceptable. The main difference between TMR and NMR is that NMR uses N modules instead of three. N should be odd such that $N = 2k + 1$, where k is the number of failing modules that should be tolerated. This formula also shows that TMR can only mask

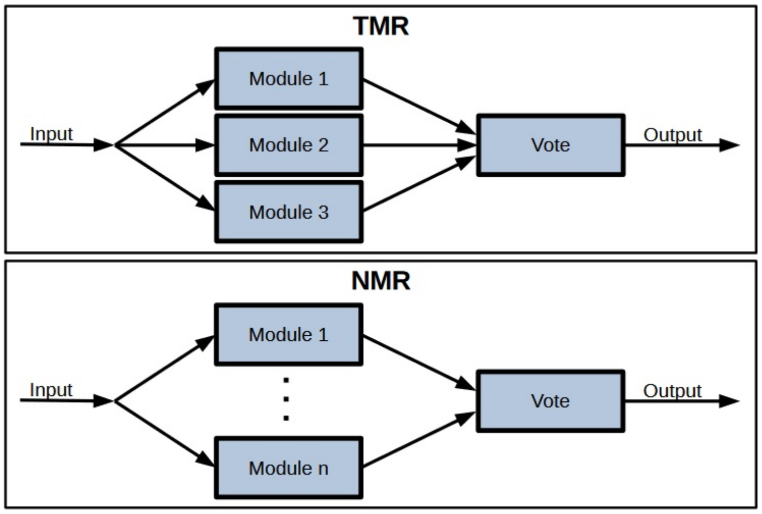


Figure 2.2: Common passive redundant architectures. [Dub13]

a single module fault. To note is that modules do not necessarily need to fail to cause an unreliable vote. It is very important that the input values arrive at the same time to avoid generating incorrect voting results. To achieve accurate timing a reliable synchronization service is needed for TMR and NMR system, which in itself needs to be fault-tolerant. Another problem for the voting system is that modules might create different outputs even in a fault-free case. This could happen in case of signal conversion. This could be solved for example, by using the median of the values that are in a certain span to each other. What this shows is that the voter handles the input should depend on the modules used [Dub13].

In Wang et al. work "*The Research of FPGA Reliability Based on Redundancy Methods*" [WYZ11], TRM is shown to be an effective method for dealing with single event effects in FPGA-based logic systems.

Active redundancy

An active redundant system achieves fault tolerance by detecting occurring faults and performing some action that returns the system back to a fault free state. Systems that utilize active redundancy are normally systems where occasional errors are allowed as long as the system returns back to a normal operation state in a certain amount of time. In some cases, it is even enough for a system to know that an output cannot be trusted. This principle follows the active redundancy method *duplication with comparison*. It simply compares the output of two identical modules that run in parallel and generates an error signal if the results of both modules are not equal. No further actions are taken. For cases where a simple detection of a fault is not enough the *standby redundancy* is a technique for active hardware redundancy that allows the return to a normal operation state on a module failure.

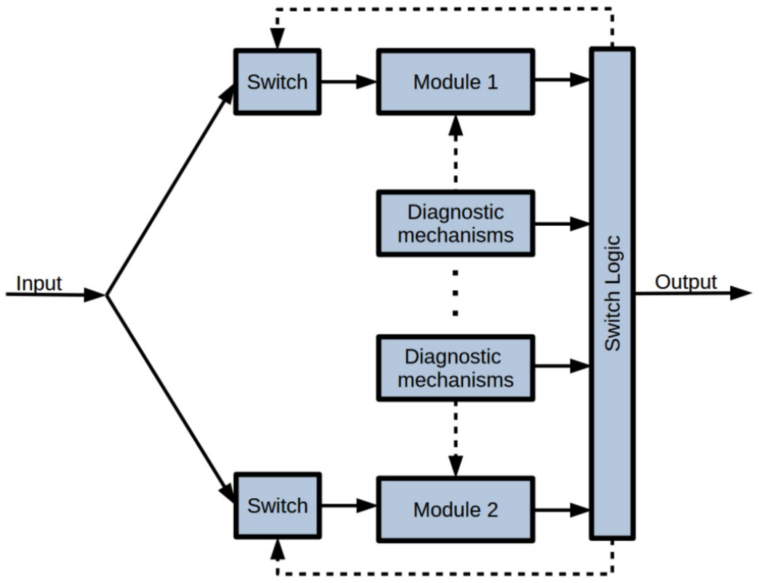


Figure 2.3: Example cold standby redundancy. [Dub13]

As figure 2.3 shows, the basic concept of a standby redundancy is that it contains N modules with some sort of fault detection or diagnostic mechanism. One of the modules is active and the other N-1 modules serve as backups. If an error is detected in a module, by the fault detection mechanism, a switch is triggered and one of the spare modules is used instead of the current active module. The standby redundancy can be grouped into two types: the hot standby and cold standby. In the hot standby, all modules are powered up. On a switch the spare can be used immediately after the current active module has failed. Such a system reduces downtime with the cost of power consumption. In the cold standby, only the current active module is powered on, all the other modules are in low power state or entirely turned off. This decreases power consumption but with the tradeoff of longer downtime if the module fails [Dub13].

An example for an active redundant system is shown in Gorelik et al. work "*Range Prediction and Extension for Automated Electric Vehicles with Fail-Operational Powertrain*" [GKO18]. There they present a torque distribution strategy for controlling a fail-operational powertrain topology consisting of two independent axles which are controlled separately. The shown system is a variant of the hot standby system where both modules have active roles but can substitute each other to a certain degree. Another example for an active redundant system can be found in Li and Eckstein work "*Fail-Operational Steer-By-Wire System for Autonomous Vehicles*" [LE19]. The Work-In-Progress fail-operational Steer-By-Wire system introduced in the work is similar to a hot standby redundant system containing two modules. The voter of the presented system contains the switch logic that decides which MCU to use. When a fault is detected in MCU1 by the internal diagnostic mechanism the voter will use MCU2 till MCU1 has recovered again.

Hybrid redundancy

As the name implies, the goal of hybrid redundancy is to combine the advantages of the passive and active approaches. In hybrid redundant system fault masking is used to prevent momentary erroneous results, while fault detection and recovery are used to reconfigure a system back to its normal operating state. Hybrid redundancy is normally used in safety-critical applications (medical equipment, weapons, aircrafts, ...). Figure 2.4 shows an example of a simple hybrid redundant architectures, the *self-purging* redundancy.

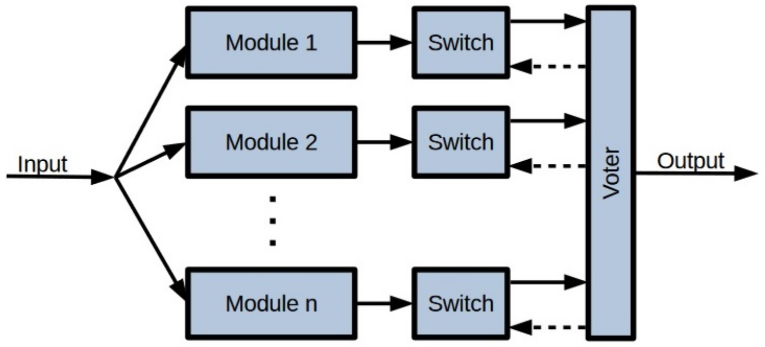


Figure 2.4: Example self-purging redundancy. [Dub13]

The self-purging redundancy can be seen as a mixture of the previously mentioned NMR redundancy and standby redundancy. It consists of N identical modules that actively participate in voting. In a feedback loop the output of the voter is compared with the vote of each individual module. If the output of the Voter is in disagreement with the result of the module, the switch opens and removes (purges) the faulty module. The system can mask N - *2* faults and is able to detect N - *1* [Dub13].

2.2.2 Software redundancy

Compared to hardware, where reliability evaluation is simplified by assuming that failures of components or modules are independent events, modules in software tend to have highly correlated failures, such that the error in one module affects the result of other modules. Software does not degrade over time. Instead software faults occur mostly due to design faults and bad specifications. Since a software fault will manifest itself as soon the relevant condition occurs, the manifestation of a software fault strongly depends on the input that the execution environment generates over time. A tragic example for this is the Ariane 5 rocket accident, where software that was safe for the Ariane 4 operation environment, caused a disaster when it was used in the Ariane 5 operation environment. Traditional hardware fault tolerance techniques like N-modular hardware redundancy are primarily designed to mitigate permanent module fault and transient faults that are caused by some external factor.

However, in software copying a module N times will just result in N faulty outputs when given the right input parameters. To achieve a useful behavior each redundant module would need to be implemented in a different way. Techniques to create fault tolerant software can be divided into *single version* techniques, *multi version* techniques [Dub13], and *process-level redundancy* [SBM$^+$09]:

- The principle of *single-version* techniques are to add components to a software that allow for detection of faults and the contain or recover the affected module. To detect faults, techniques like timing checks (for example watchdog timer), coding checks (for example CRC checksum), reasonableness checks (for example, check if value is in bounds), structural checks (for example, expect fixed number of bytes). Some techniques for fault containment in software are: modularization (split application into modules, avoid shared resources between modules, monitory communication between modules), system closure (functions and resources are only accessible if permission is granted) and atomic actions (components interact exclusively with each other). Some fault recovery techniques would be: exception handling (interrupt normal execution to handle an abnormal condition), checkpoint and restart (check correctness of a result, re-initialize to a previous state if fault is detected), process pairs (check correctness of a result, the second process takes over if first process fails, while second process is used for continuation of normal execution the first process tries to recover)

- The principle of *multi-version* techniques are use multiple versions of the same software component. The idea behind generating different versions of software executing the same tasks is to minimize the probability of a common fault between the versions. To generate different versions of software that fulfill the same design specifications it is common to use different teams, different coding languages, or different algorithms. To detect faults, techniques like *recovery blocks* (checkpoint and restart for multiple software versions), *N version programming* (software version of N-modular hardware redundancy) and *N self-checking programming* (combination of N version programming and recovery blocks).

- The *process-level redundancy* is a technique that utilizes the multi-core architectures of modern processors and the process handling of operating systems to detect transient faults. It leverages the operating systems capability to schedule redundant processes efficiently with available hardware resources. This is done by adding a emulation layer between the operating system and processes which replicates the input and emulates system calls. The emulated layer is also used to detects transient faults by comparing the output, execution time, and error messages of the redundant processes.

2.2.3 Information redundancy

For safety critical devices it is important that faults in stored or transmitted information is detected and if possible corrected. Such fault-tolerant behavior can be achieved with *coding*. In coding *n* bit sized *data words* are encoded based on some *coding scheme* into n + k bit sized *code words*. The fault detection and correction capability of a coding scheme depends on the *code distance* the scheme creates. From the 2^{n+k} code words a coding scheme can create from n bit data words and k code bits, 2^n are valid. Faults cannot be detected if corrupted code word is valid. The code distance or minimum *hamming distance* is the distance between two valid code words (i.e. the number of bits that have to change to get from code word x to code word y). A code distance of C can detect C - 1 bit faults and correct $\frac{C-1}{2}$ bit faults. An important assumption of coding theory is that faults are much more likely to affect only a few bits than affecting many. Some common coding schemes are *Parity Code*, *CRC* and *AN-codes* [Dub13].

2.2.4 Time redundancy

The idea behind time redundancy is to repeat the computation or transmission of data and compare the results with the previously stored data. The comparison between the data can detect transient faults and in case of multiple computations/transmissions even correct them. In this case, systems similar to the voting techniques used for hardware redundancy, can be used. An issue with time redundancy is that in case of a fault the data or functionality required to repeat a computation is not affected by the fault. An interesting aspect of time redundancy is that it can detect and correct permanent faults when it is used in combination with some encoding scheme. Some techniques targeting permanent faults would be *alternating logic*, *recomputing with shifted or swapped operands*, and *recomputing with duplication with comparison* [Dub13].

2.3 Fail-operational aspects in Norms

The goal to reduce, control or prevent the occurrences of dangers and risks lead to the creations of norms and standards. Over the decades functional safety related norms and standards have not only been created but also adapted to fit the changing needs of the industries they were developed. Since different industries have different needs, their definitions for safety and risks often differ and with them the approaches they created to handle them. Table 2.1 is a shortened version of Table 2.1 from Ross book *"Funktionale Sicherheit im Automobil"* [Ros19] and gives a good overview about the different functional safety related norms from some industries.

	Automotive	Aeroplane	Railway
Risks and Dangers	ISO 26262	CFR 25.1309	EN 50126
Control of Risks	ISO 26262	DO 178C, DO 254 / ED-12C	EN 50126/28/29
Safety and Integrity	ISO 26262, ISO/PSA 21448	ARP 4754A, ARP 4761	Details IEC 61508

Table 2.1: Overview of safety related norms in different industries. [Ros19]

The following subsections will give a small overview of the Aeroplane Norms/Standards and a rather close look at the Automotive Norms.

2.3.1 Avionic Norms/Standards

The Aerospace Recommended Practice (ARP) 4754/A *"Guidelines For Development Of Civil Aircraft and Systems"* [Int10] from the SAE International, historically SAE stands for Society of Automotive Engineers, is a guideline for development of civil aircraft and systems with an emphasis on safety aspects. ARP 4754 goal is to harmonize the existing civil aviation regulations for transport category airplanes by the U.S. Federal Aviation Administration and the European Aviation Safety Agency. ARP 4754 defines following structural overview of the avionics safety standards [Int10]:

- *SAE ARP 4761 "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment"* covers safety assessment process, defining functions, failure and safety information of avionic systems, including specification and analysis techniques [Int96].
- *SAE ARP 4754/A "Guidelines For Development Of Civil Aircraft and Systems"* covers the development of aircraft systems taking into account the overall aircraft operating environment and functions [Int10].

- *RTCA DO-254 / EUROCAE ED-80 “Design Assurance Guidance for Airborne Electronic Hardware“* covers the aspect of electronic hardware development [fAEOfCAE00].
- *RTCA DO-178C / EUROCAE ED-12C “Software Considerations in Airborne Systems and Equipment Certification“* covers the aspect of software development [fAEOfCAE12].

Safety assessment and risk classification

The safety assessment processes for aviation systems and the methods for their application are defined in SAE ARP-4754 and SAE ARP-4761. Figure 2.5 shows the relation between the SAE specified System Assessment Process (SSA) and the development process for an aircraft. As seen in the figure the SAE safety assessment process runs in parallel to the development process, with a quite extensive common cause analysis. The objective is to prove independence where it is required by the system architecture, with special focus on common cause faults. Depending on the Functional Hazard Analysis (FHA) a Design Assurance Level (DAL) is assigned to the aircraft and its system components. As seen in Table 2.2 depending on the failure classification given by the FHA and the therefore resulting DAL a minimum failure probability per flight hour specified. Furthermore, depending on the DAL some safety goals can become mandatory. Though fail-operational behavior is not explicitly addressed in SAE ARP-4761, in practice to achieve the required failure risk resulted into redundancy systems like Duo-Duplex redundancy, implemented as part of Airbus flight control [TLS05] or a Triple-Triple redundancy as it is used in a Boeing 777 primarily flight computer [Yeh96].

Failure classification	DAL	Failure risk per flight hour	Frequency of occurrence
Catastrophic	Level A	$< 10^{-9}$	extremely improbable
Hazardous	Level B	$< 10^{-7}$	extremely remote
Major	Level C	$< 10^{-5}$	remote
Minor	Level D	$< 10^{-3}$	resonable remote
No effect	Level E	-	frequent

Table 2.2: Design Assurance Level (DAL). [Int96]

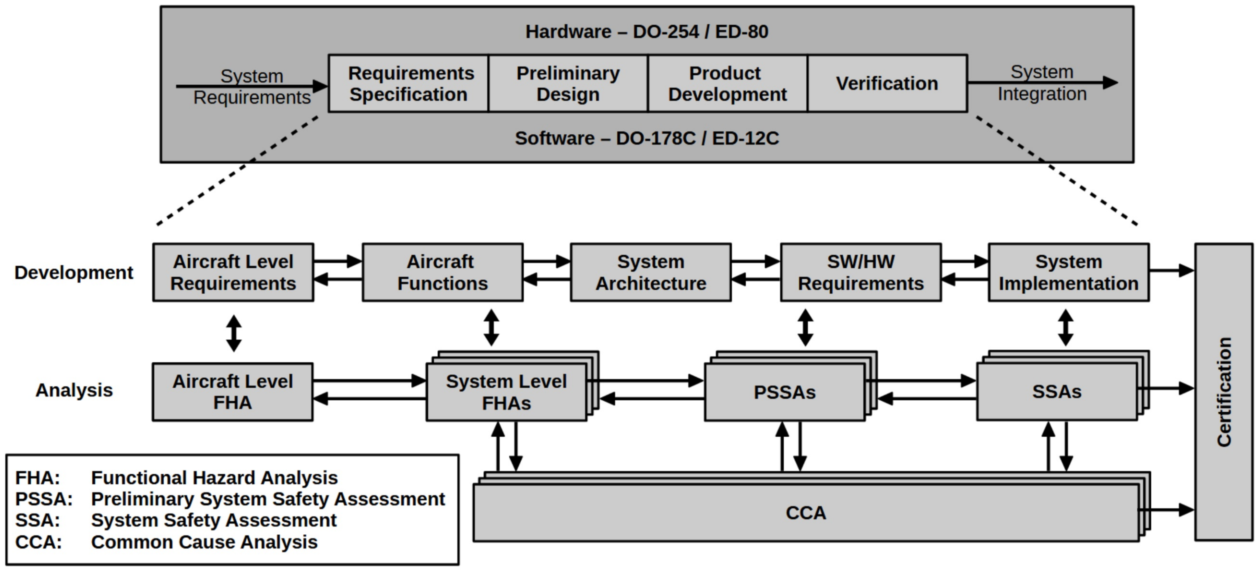


Figure 2.5: SAE System Safety Assessments process in relation to the development processes. [Int10]

2.3.2 Automotive Norms/Standards

The main focus of this thesis, in regards to functional safety, is on their application in automotive area. The development safe HW/SW systems is a completely different challenge to constructing more conventional mechanical and hydraulic system. Due to experience and necessarily the automotive industry used various functional safety methods even before they started to apply state-of-the-art functional safety standards. The first functional safety standard that found widespread use in the automotive industry was the international standard IEC 61508 *“Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems“* [ICE10] published by the International Electrotechnical Commission. The automotive industry recognized that the IEC 61508 was limited in its application to their specific needs. This led to the creation of the ISO 26262 *“Road vehicles Functional safety“* [ISO11] in 2011. One of the main challenges of the ISO 26262 was to have existing practices, methods, well established designs and architectures of existing techniques not be deemed as dangerous while ensuring that the planned safety of future technologies can be structurally argument. The ISO 26262 is heavily based on the IEC 61508 and in terms of functional safety its mainly focused on E/E-Systems. This focus on E/E-Systems is based on the assumption that the industry is capable of constructing safe non E/E-Systems. With the “second edition“ of the ISO 26262 [ISO18a], which was released at the end of 2018, the standard was revised and extended. Most changes in the revision are based on the extension of the norm to include commercial vehicles like busses, trucks, two-wheeler and three-wheeler, as well as some rational and informative changes. Even with the revisited publication, ISO 26262 is still a standard mainly for fail-safe systems. Figure 2.6 gives an overview of the ISO 26262:2018.

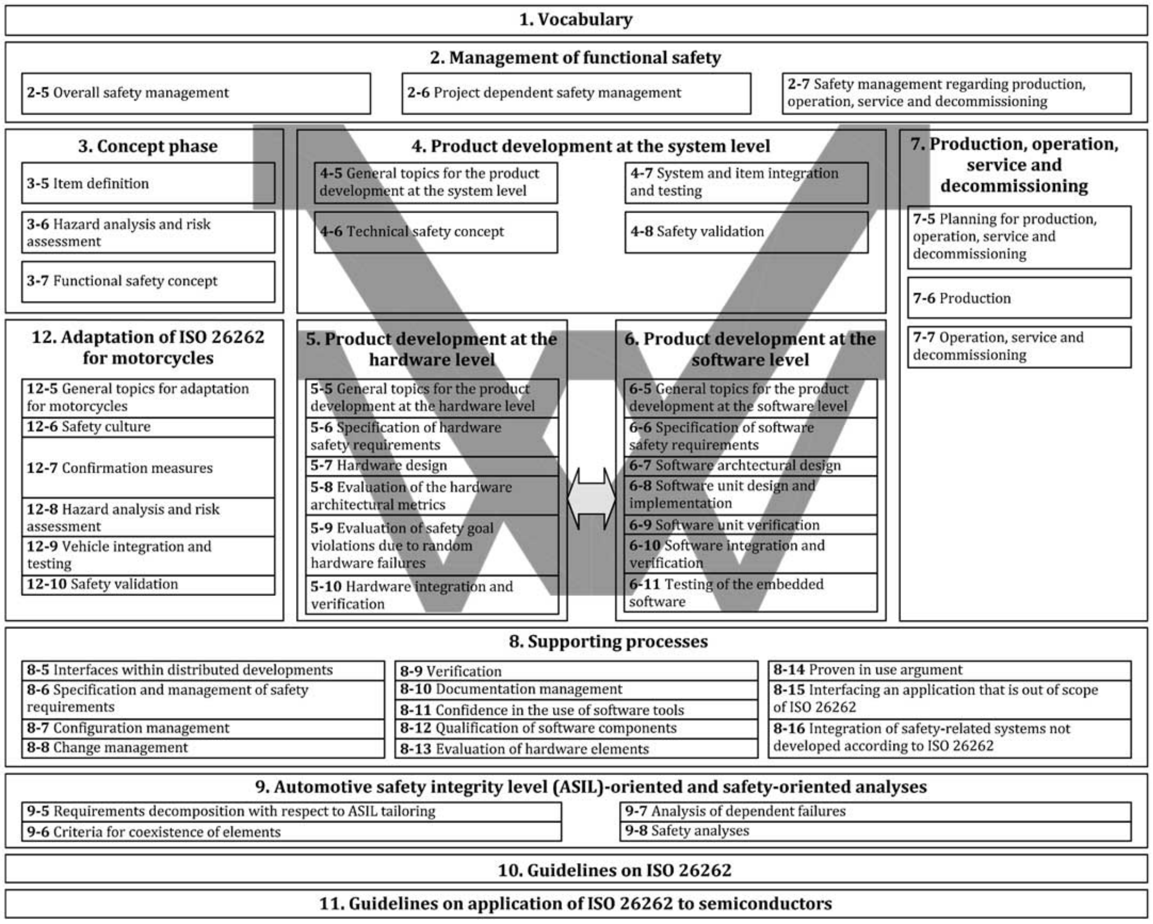
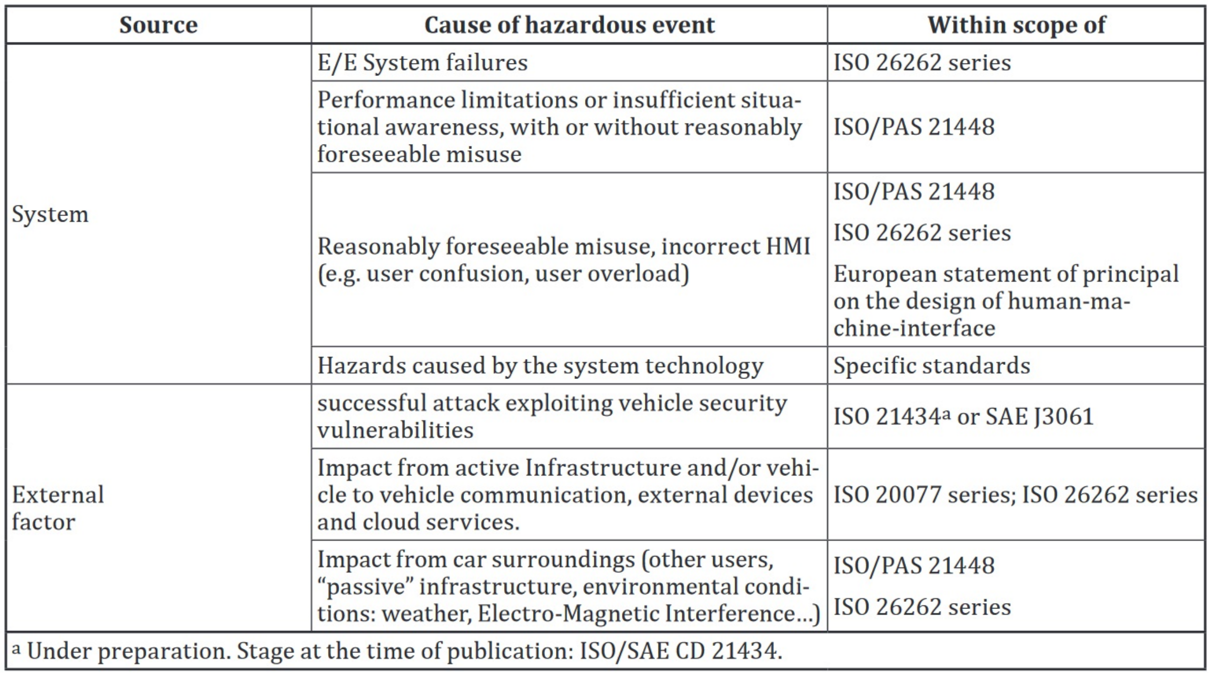

1. Vocabulary
2. Management of functional safety
2-5 Overall safety management
2-6 Project dependent safety management
2-7 Safety management regarding production, operation, service and decommissioning
3. Concept phase
3-5 Item definition
3-6 Hazard analysis and risk assessment
3-7 Functional safety concept
4. Product development at the system level
4-5 General topics for the product development at the system level
4-6 Technical safety concept
4-7 System and item integration and testing
4-8 Safety validation
7. Production, operation, service and decommissioning
7-5 Planning for production, operation, service and decommissioning
7-6 Production
7-7 Operation, service and decommissioning
12. Adaptation of ISO 26262 for motorcycles
12-5 General topics for adaptation for motorcycles
12-6 Safety culture
12-7 Confirmation measures
12-8 Hazard analysis and risk assessment
12-9 Vehicle integration and testing
12-10 Safety validation
5. Product development at the hardware level
5-5 General topics for the product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of safety goal violations due to random hardware failures
5-10 Hardware integration and verification
6. Product development at the software level
6-5 General topics for the product development at the software level
6-6 Specification of software safety requirements
6-7 Software archtectural design
6-8 Software unit design and implementation
6-9 Software unit verification
6-10 Software integration and verification
6-11 Testing of the embedded software
8. Supporting processes
8-5 Interfaces within distributed developments
8-6 Specification and management of safety requirements
8-7 Configuration management
8-8 Change management
8-9 Verification
8-10 Documentation management
8-11 Confidence in the use of software tools
8-12 Qualification of software components
8-13 Evaluation of hardware elements
8-14 Proven in use argument
8-15 Interfacing an application that is out of scope of ISO 26262
8-16 Integration of safety-related systems not developed according to ISO 26262
9. Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses
9-5 Requirements decomposition with respect to ASIL tailoring
9-6 Criteria for coexistence of elements
9-7 Analysis of dependent failures
9-8 Safety analyses
10. Guidelines on ISO 26262
11. Guidelines on application of ISO 26262 to semiconductors

Figure 2.6: Overview of the ISO 26262:2018 series of standards. [ISO18a]

With the rise of Smart Mobility, the demand for a standard with fail-operational system in mind is high. To answer this demand the standard ISO 21448 *"Safety of the intended functionality"* [ISO19], which is current in its PAS phase, was introduced. Still since ISO 21448 does not apply to faults covered by ISO 26262 it is necessary to consider both standards. [Sch16] takes a closer look at the fail-operational aspects for each part of the ISO 26262:2011 standard. Though ISO 26262:2011 is now replaced by ISO 26262:2018, since the changes made to the standard are evolutionary, [Sch16] is still a valid read for fail-operational automotive systems. ISO 26262:2018 now contains a fully new section on semiconductors and the scope of road vehicles got extended by application to commercial vehicles and motor cycles. From a safety aspect the interesting changes are the improved safety analysis methods for software, the more detailed requirements for semiconductors, security and the support safety case for ADAS, fail-operational and diversified re-

dundancy.
The safety aspect of the ISO 21448 applies to functionality that requires proper situational awareness. The ISO 21448 is concerned with guaranteeing safety of the intended functionality (short SOTIF). While ISO 26262:2018 covers functional safety in the event of system failures, ISO 21448 covers safety hazards that result without system failure. The ISO 21448 provides guidance on the applicable design, verification and validation measures needed to achieve the SOTIF. As mentioned before the ISO 21448 does not apply to fault s already covered by the ISO 26262:2018 series or to hazards directly caused by the system technology. For a better overview what norms are relevant for which hazards event the ISO 21448 contains the table seen in figure 2.7.



Source	Cause of hazardous event	Within scope of
System	E/E System failures	ISO 26262 series
	Performance limitations or insufficient situational awareness, with or without reasonably foreseeable misuse	ISO/PAS 21448
	Reasonably foreseeable misuse, incorrect HMI (e.g. user confusion, user overload)	ISO/PAS 21448 ISO 26262 series European statement of principal on the design of human-machine-interface
	Hazards caused by the system technology	Specific standards
External factor	successful attack exploiting vehicle security vulnerabilities	ISO 21434[a] or SAE J3061
	Impact from active Infrastructure and/or vehicle to vehicle communication, external devices and cloud services.	ISO 20077 series; ISO 26262 series
	Impact from car surroundings (other users, “passive” infrastructure, environmental conditions: weather, Electro-Magnetic Interference...)	ISO/PAS 21448 ISO 26262 series

[a] Under preparation. Stage at the time of publication: ISO/SAE CD 21434.

Figure 2.7: Overview of safety relevant topics addressed by different ISO standards. [ISO19]

As seen in table of figure 2.7, another important safety aspect is cyber-security. The SAE J3061 *“Cybersecurity Guidebook for Cyber-Physical Vehicle Systems“* [Int16] is used as a practical guide for information security of a product over its whole lifetime. The standard is one of the bases for the new ISO/SAE 21434 *“Road vehicles Cybersecurity engineering“* [ISO18b], which is currently in its Committee Draft phase. The focus of the ISO/SAE 21434 is on defining a common terminology and important aspects of Cybersecurity. It does not prescribe specific Cybersecurity technology, solutions or requirements to use certain methods or communication systems but instead sets some minimum criteria for vehicle Cybersecurity engineering and highlights key Cybersecurity challenges of the industry.

As mentioned in Hans-Leo Ross book *"Funktionale Sicherheit im Automobil"* [Ros19], there is another safety aspect that might not be voiced directly by the norms but is indirectly connected with their proper implementation, the investment safety. A lack of safety performance will lead to revenue erosion. Since a single component with lacking safety can create huge after production cost and in the worst cases a removal of the product from the market. Furthermore, a product with issues will inevitably damage the image of the company that produces the product.

Aspects of fail-operational in IEC 61508 [ICE10]

For the automotive industry the IEC 61508 can be seen as the "mother" for functional safety standards. In case of the ISO 26262 this is certainly the case since it is an adaptation of the ICE 61508. Interesting is that the ICE 61508 defines the term *fault tolerance* explicitly as an *functional unit* that needs to maintain a *required* function even in the presence of failures or errors. While the ISO 26262 only includes an explicit definition for *fault tolerance* in its latest 2018 edition as the *ability to deliver a specified functionality in the presence of one or more specified faults.*

The IEC 61508 standard defines following techniques for HW to achieve the safety standard [ICE10]:

- Monitored redundancy, as monitoring function for electrical components
- 16bit signature (or Cyclic Redundant Code (CRC)), as monitoring function non-volatile memory
- Block replication, as monitoring function for non-volatile memory
- Multi-channel parallel output, as a diagnostic measure for external interfaces
- Input comparison/voting, as a diagnostic measure for external interfaces and sensors
- HW redundancy, as a diagnostic measure for internal communication
- Transmission- and information redundancy, as a diagnostic measure for internal communication
- Switch to secondary power supply, as a reaction measure to power supply failures

For SW the standard defines following fault tolerant architectural measures [ICE10]:

- Diverse redundancy (different types with different degree of diversity)
- Recovery blocks (see section3.1)

- Regeneration by repetition
- Graceful degradation, meaning that the faulty parts of the SW can be deactivated
- Artificial intelligence - fault correction
- Dynamic reconfiguration, meaning that in case of a HW fault, SW functions executed by this faulty unit are allocated to another, correctly functioning HW resource

To note here is that the IEC 61508 does not define the HW diagnostic methods as measure for fail-operational behavior or fault tolerance. Furthermore, some of the fault tolerance architectural measures for SW are techniques that are not highly recommended safety environment (e.g. artificial intelligence - fault correction). Adam Schnellbach doctoral thesis *"Fail-operational automotive systems"* [Sch16] further investigates how the techniques addressed above can be implemented and are of interest for a system that should behave fail-operational.

Aspects of fail-operational in ISO 26262 [ISO18a]

As shortly mentioned in the previous section, the ISO 26262 a standard mainly for fail-safe systems. This is further reassured with the addition of the definition of the term *fault tolerance* in the 2018 revision of the ISO 26262. *Fault tolerance* is defined as the ability to deliver a specified functionality in the presence of one or more specified faults. Interesting is that the definition from the ISO 26262 is somewhat weaker than the definition from the ICE 61508, since the ISO 26262 uses the term *"... in the presence **specified** faults"* while the ICE 61508 just is more general and uses *"... in the presence of failures or errors"*. Of course, the ISO 26262 defines a concept that helps with specifying the faults.

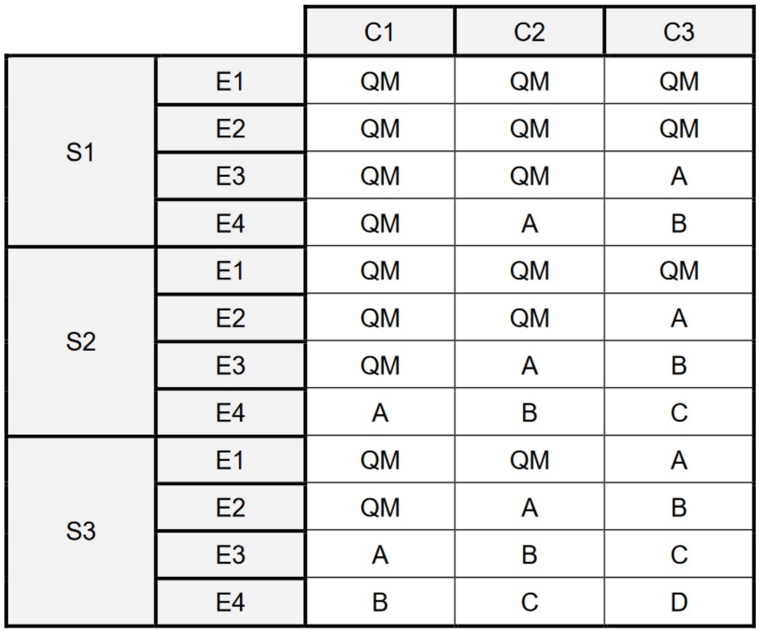
Similar to the SAE System Safety Assessments process for aircraft, the ISO 26262 specifies a Hazard Analysis and Risk Assessment (HARA). In the HARA hazards get identified and get an Automotive Safety Integrity Level (ASIL) from ASIL "A" to ASIL "D" assigned. The primary output of the HARA are the defined safety goals with respective ASIL-s. Functional safety concepts are defined to fulfil the safety goals. The following list summarizes the steps in a HARA [ISO18a]:

- **Situation analysis and hazard identification:** This step is to determine potential hazards and evaluate their consequences for each operation mode an "item" might have.
- **Hazard Classification:** The identified potential hazards are classified based on the estimation of probability of:
  - *Exposure*: Defines the likelihood of an operational situation which may lead to a hazardous event in case of a failure. The exposure ranges from E0 extremely unusual situation to E4 highly likely situation.

 - *Severity*: Estimates the harm to person(s) that are at risk to receive an injury from an occurring hazardous event that was not timely controlled. The severity ranges from S0 no injuries to S3 fatal injuries.
 - *Controllability*: Characterizes the ability the person(s) that are at risk to react to a hazardous event in a timely fashion. The controllability ranges from C0 simple controllable to C3 uncontrollable.

 The hazard classification strongly depends on the use case. For example, a perception system that is used as parking assistance won't cause fatal injuries on failure, where a perception system for a break assistant might will.

- **ASIL Determination:** Figure 2.8 shows how the before defined hazardous event parameters S, E and C are used to define ASIL. Four ASILs are defined, where ASIL "A" is the lowest safety integrity level and ASIL "D" is the highest one. While a "quality managed" (QM) rating signifies that the safety goal is not severe enough to require specific regulations through the standard.



		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 2.8: ASIL Determination [ISO18a].

- **Safety goal formulation:** Safety goals (SG) are determined for each evaluated event from the hazard analysis. Functional safety concepts are defined to fulfil the safety goals.

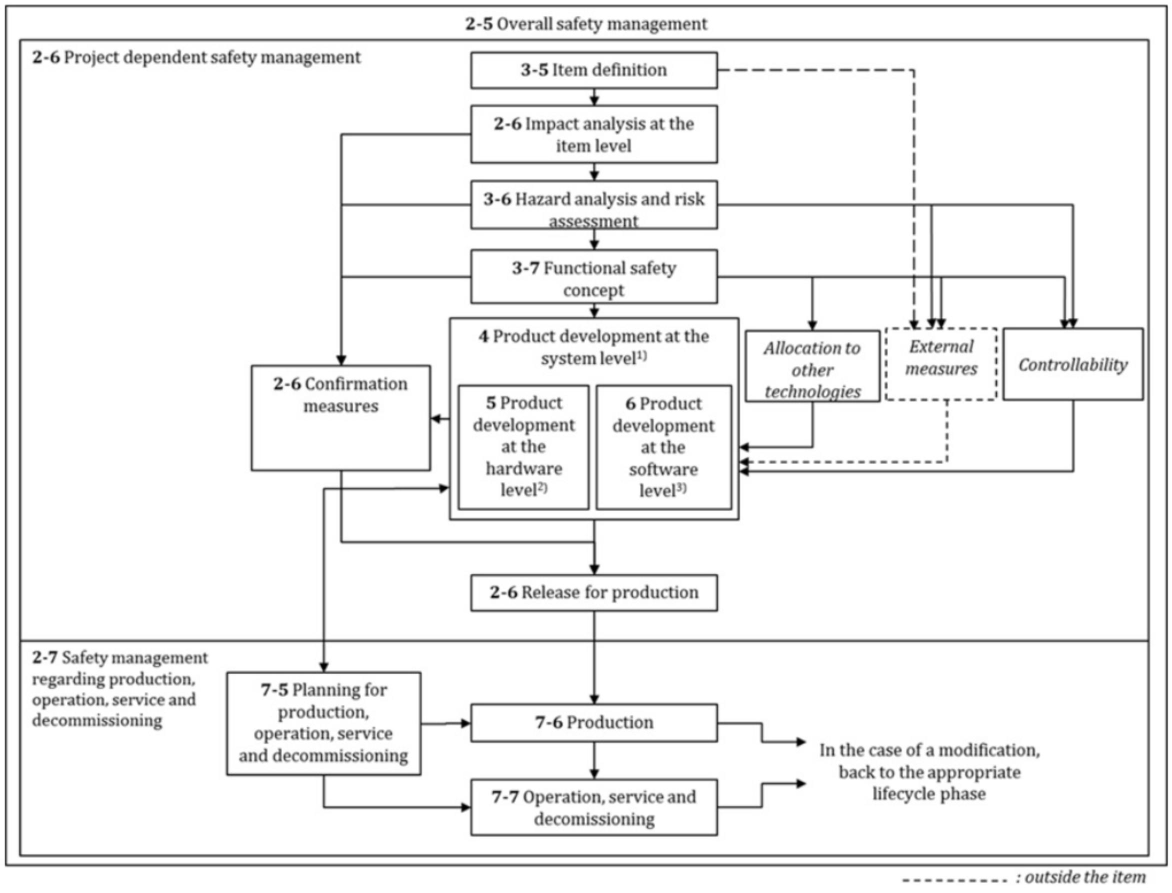

2-5 Overall safety management
2-6 Project dependent safety management
3-5 Item definition
2-6 Impact analysis at the item level
3-6 Hazard analysis and risk assessment
3-7 Functional safety concept
4 Product development at the system level1)
5 Product development at the hardware level2)
6 Product development at the software level3)
2-6 Confirmation measures
Allocation to other technologies
External measures
Controllability
2-6 Release for production
2-7 Safety management regarding production, operation, service and decommissioning
7-5 Planning for production, operation, service and decommissioning
7-6 Production
7-7 Operation, service and decomissioning
In the case of a modification, back to the appropriate lifecycle phase
: outside the item

Figure 2.9: Overall safety management according to ISO 26262:2018. [ISO18a]

The ISO 26262:2018 overall safety management cycle shown figure 2.9 gives an overview of the most important safety activities with references to the specific chapter of the ISO 26262:2018. In general, the ISO 26262:2018 gives guidance to project independent safety management through establishing a corporate safety culture and a project specific safety management guidance by using a safety plan, containing the safety relevant process steps.

As seen in the figure 2.10 the project safety management part can be split into three phases:

- Concept phase
- Production development phase
- After release for production phase

The concept phase is the first phase in the lifecycle of the safety management. This phase contains previously mention HARA, the definition of the safety goals and the functional safety concept. The HARA with its resulting ASIL can identify if a fault tolerant behavior is required for a mechanism. Furthermore, with an

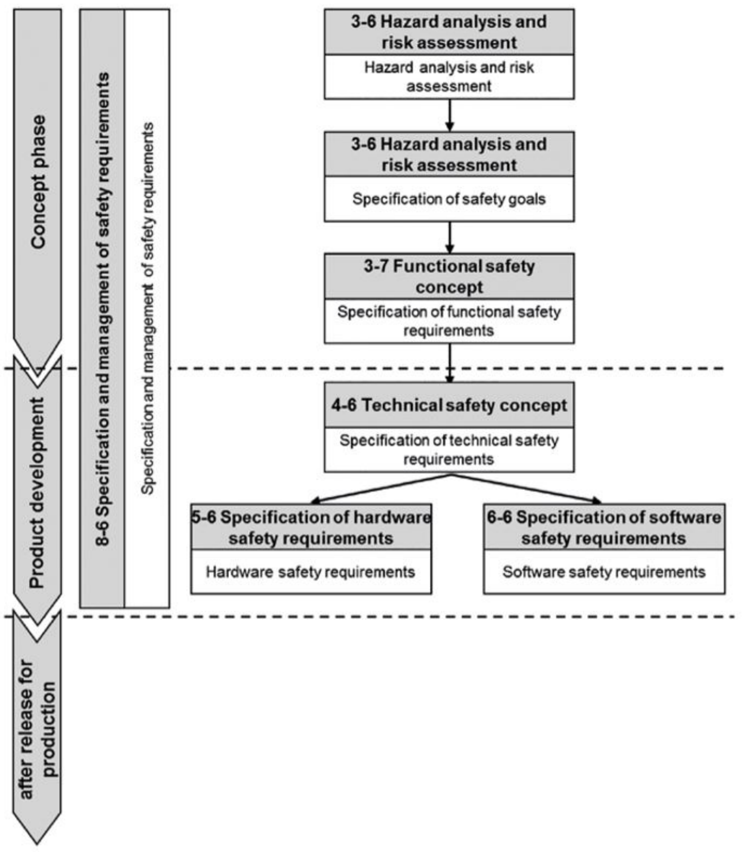

Concept phase
Product development
after release for production
8-6 Specification and management of safety requirements
Specification and management of safety requirements
3-6 Hazard analysis and risk assessment
Hazard analysis and risk assessment
3-6 Hazard analysis and risk assessment
Specification of safety goals
3-7 Functional safety concept
Specification of functional safety requirements
4-6 Technical safety concept
Specification of technical safety requirements
5-6 Specification of hardware safety requirements
Hardware safety requirements
6-6 Specification of software safety requirements
Software safety requirements

Figure 2.10: Structure of requirements 26262:2018. [ISO18a]

ASIL $\geq$ A a fail-operational behavior is required from the mechanism. The primary output of the concept phase are the safety goals. They are always related to a “loos of function“ hazard. The safety concept on the vehicle architectural level are then defined from the safety goals by determines the necessity of a fail-operational behavior and the appropriated safe state for each safety goal.

The next phase in the lifecycle of the safety management is the product development phase. As seen in figure 2.9 and figure 2.10 it can be split into three parts depending on what level the technical safety concept development currently is. Comparing the two previous mentioned Figures, the figure 2.9 has the better illustration on how the technical safety concept definition, defined on the System level envelops the production and safety requirements for HW and SW. Important to note is that the defined technical safety concept can define safety mechanisms that require an implementation on both, HW and SW. The system level also focuses on testing the correct implementation and performances of the safety mechanism to provide evidence of compliance with the safety goals. On the Hardware level the technical safety requirements specification for the HW further refines the input requirements, influencing the architectural and detailed design phase of the HW. Also, on the Hardware level is the architectural analyze and testing of the HW. Similar as with the Hardware level, on the Software level the technical safety requirements specification for the SW is further refined and implemented into the SW requirement specification, which acts as a reference for the SW architecture and detailed design. Also, on the Software level is the architectural analyze and testing of the SW.

The last phase in the lifecycle of the safety management is the after release for production phase. An important aspect that the ISO 26262:2018 contains that is missing from the ICE 61508 is that it is not sufficient to develop a safe system but it also necessary to produce it as a safe system. Furthermore, it is necessary that it can be operation and maintained safely, as well as decomposed safely after it reached its end of lifetime.

2.4 Reliability and Thermal Stress

The reliability of semiconductor devices can be represented by a failure rate curve which is commonly known as the “bathtub curve“. As mentioned in Martin L. Shooman book *“Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design“* [Sho02], the bathtub curve is a hazard curve that has been known for decades now and is valid for many types of equipment’s. Looking in figure 2.11 the curve can be divided into three regions [Sho02]:

- **Early Failures:** Young devices often have a high failure rate due to manufacturing imperfections or performance marginality. This early high failure rate tends to decrease over time. To find and eliminate early failures, manufactures normally put the semiconductors devices under stresses such as temperature and voltage.
- **Random Failures:** After the latent defects have been found and removed, the device enters its stable operating period. Failures that occur during this period are usual due to some unexpected excessive stress (power surge, software error, ...) or early failures that were not detected.
- **Wear-Out Failures:** When a device reaches the end of its inherent lifetime it enters the wear-out period. Depending on the usage conditions the failure rate tends to increase rapidly.

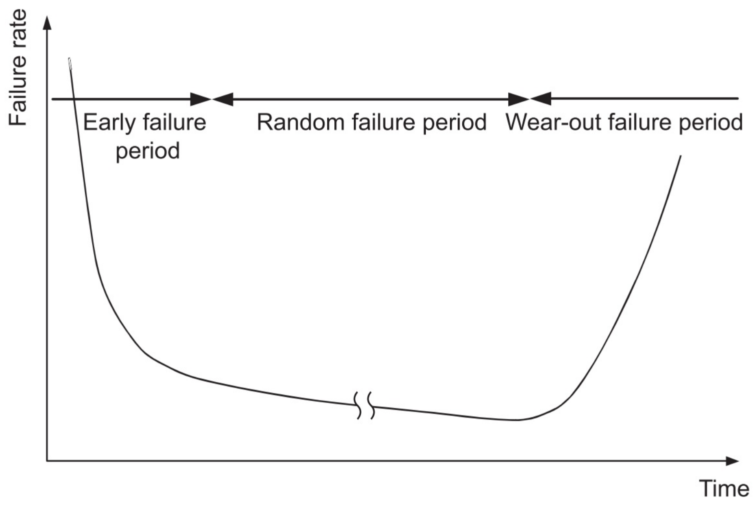


Figure 2.11: Failure Rate Curve (Bathtub Curve). [Cor17]

2.4.1 Memory Reliability

For all computing systems the memory systems play major role in determining their power consumption, reliability and performance. Operating system, file system structures, program binaries, program variables and so on are constantly stored and loaded from the main memory of a modern system. Dynamic random-access memory (DRAM) is used by nearly any modern system and programmers rely on that that a byte written to an address can reliably be read until it's overwritten or the module has been powered off. The International Technology Roadmap for Semiconductors (ITRS) suggest a nominal DRAM cell lifetime of $3 * 10^{16}$ write cycles before failure [ITR12]. For DDR3 with a row change latency of ~50ns, according to the JEDEC DDR3 SDRAM standard JESD79-3C [Ass08], this would result in about 47.5 years of constant data accessing before a failure accrues.

DRAM Device Failures

Though theoretically DRAM should be able to run reliable for yeas there are various device-level failures mechanisms in DRAM.

- **Retention Failures:** Since the charge of a DRAM cell leaks over time it needs to be refreshed periodically to prevent the DRAM cell from losing its data. The time a cell can store data varies between each cell, the manufacturing processes and the size of the cells. With smaller sized cells being more prone to failure (see [FDN+01], [LJK+13], and [MWKM15]). Furthermore Patel et al. in *"Enabling the mitigation of dram retention failures via profiling at aggressive conditions"* [PKM17] and Liu et al. in *"An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms"* [LJK+13] show that a DRAM cell's retention time decreases exponentially as ambient temperature rises.

- **Disturbance Failures:** Beside the loos of data due to charge leaks, the data in a DRAM cell can altered due to external events. For example, alteration due to electrical interference between cells (see [LJK+13], [MS14], and [PKM17]). A software example that exploits this effect is for example the RowHammer as shown in Kim et al. work *"Flipping bits in memory without accessing them: An experimental study of dram disturbance errors"* [KDK+14]. Another example of disturbance failure from external events is when charged particle transfers a part of its charge to the capacitor. The charge of a charged particle needs to invert the state of the capacitor to change the data in a DRAM cell. This mostly happens at higher altitudes where cosmic rays can cause such an effect on DRAM capacitors due to reduced atmospheric protection (see [SSD+13] or [HSS12]).

- **Endurance Failures:** As mentioned above theoretically DRAM cells should be able to endure for decades before they wear out. But the work of Wang et al. in *"What can we learn from four years of data center hardware failures?"* [WZX17] and Meza et al. in *"Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field."* [MWKM15] observed signs of DRAM endurance failures over the course of several years. While Sridharan et al. in *"Feng shui of supercomputer memory positional effects in dram and sram faults."* [SSR+17] shows no sign of wear out failures after a operational lifetime of five years and Chia et al in *"New dram hci qualification method emphasizing on repeated memory access"* [CWB10] shows that endurance failures can be forced when DRAM cells operate in and by extreme conditions.

DRAM Failures in the wild

For this section the dissected studies will be of large-scale studies on DRAM failures fin the field, since for the functional safety aspect of this thesis the large-scale studies of DRAM failures in the field are more helpful in regards of failure rate and severity. Schroeder et al. in their work *"Dram errors in the wild: A large-scale field study."* [SPW09] performed a study on Google servers over a period of 2.5 years, and shows a relative high memory error rate across Google servers. Their study also observes a correlation between temperature and correctable error rate. But as seen in figure 2.12, they also observe that the effect of temperature is relatively small when compared to the effect memory utilization has on the correctable error rate. Furthermore, they observe that the rate of uncorrectable errors is strongly influenced by age. [SPW09]

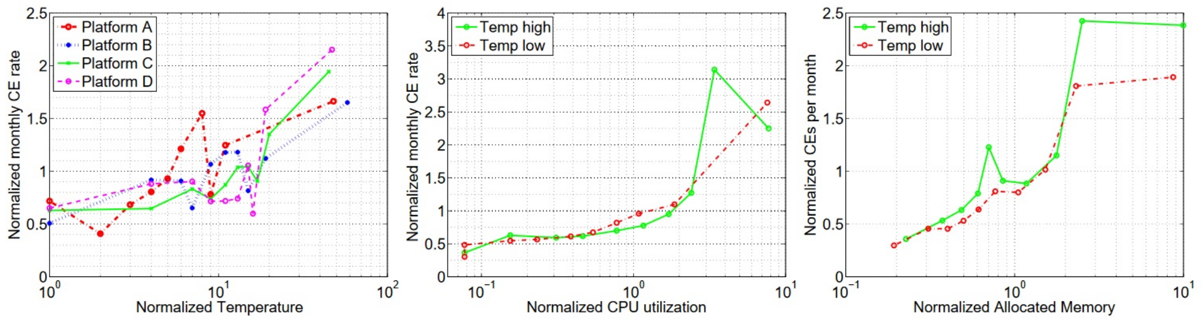


Figure 2.12: The left graph shows the normalized monthly rate of experiencing a correctable error as a function of the monthly average temperature, in deciles. The middle and right graph show the monthly rate of experiencing a correctable error as a function of memory usage and CPU utilization, respectively, depending on whether the temperature was high (above median temperature) or low (below median temperature). [SPW09]

Meza et al. observed and analyzed in their work *"Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field."* [MWKM15] for fourteen months the memory errors in the "entire fleet of servers" at Facebook. The study shows, as seen in figure 2.13, that the majority of errors are caused by the memory controller and memory channels but also that these errors occur on only a small fraction of servers. Furthermore, their work shows that

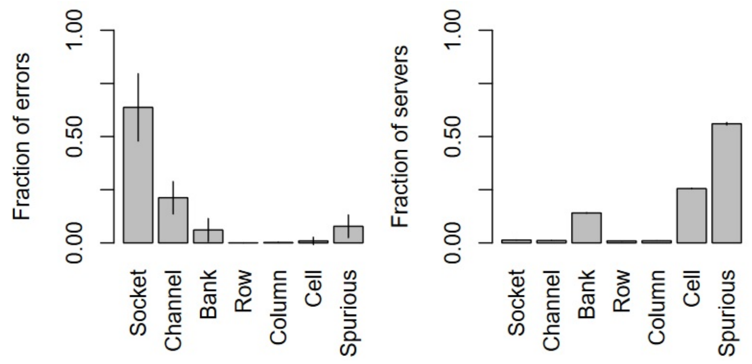


Figure 2.13: Left hand side shows the fraction of logged errors each month that are attributed to different types of failures. Right hand side shows the fraction of servers hit with correctable errors each month per type. [MWKM15]

modern DRAM cell fabrication technologies have higher failure rates and that the DIMM architecture also effects memory reliability, where DIMMs with fewer chips and lower transfer rate have the lowest error rates. Interesting is that Meza et al.

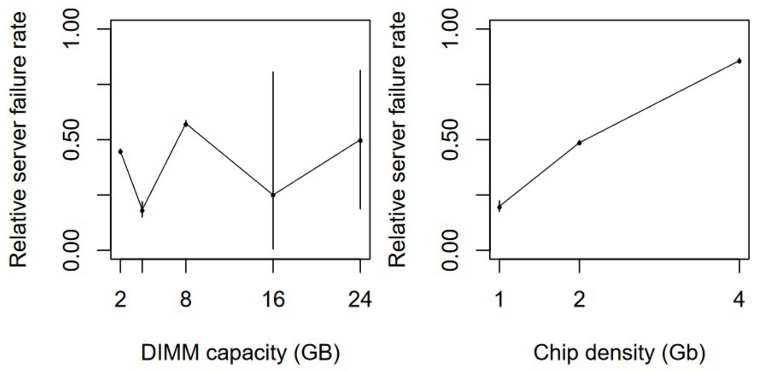


Figure 2.14: Left hand side relative failure rate for severs with different DIMM capacities. Right hand side relative failure rate for severs with different chip density. [MWKM15]

[MWKM15] work compared to Schroeder et al. work [SPW09] sees no clear trends between failure rates and CPU and memory utilization but rather a correlation of workload types and failure rates.

The previous mentioned work of Sridharan et al. *"Feng shui of supercomputer memory positional effects in dram and sram faults."* [SSD+13] examines the impact of aging on DRAM in the early lifespan of two supercomputers over multiple months. Because both supercomputers in their work include hardware scrubbers in DRAM, L2 and L3 caches the authors define a permanent fault when the device generates error messages in multiple scrub intervals, and else transient fault.

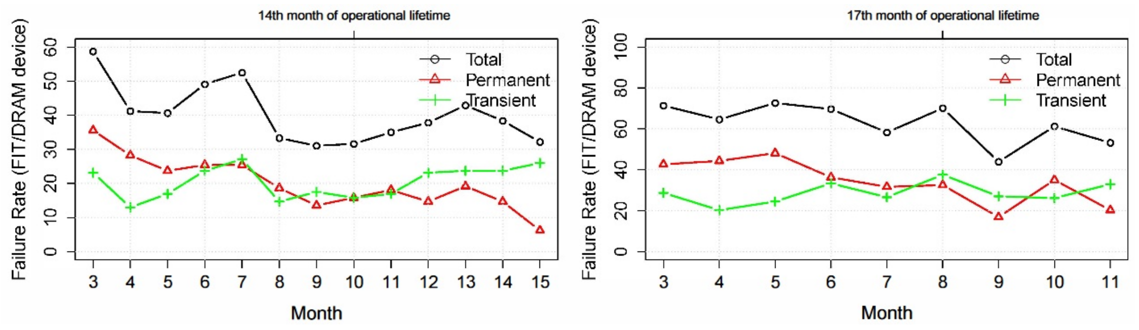


Figure 2.15: Left hand side Cielo DDR3 device fault rates per month; 23 billion DRAM hours total. Right hand side Jaguar DDR2 DRAM device fault rates per month; 17.1 billion DRAM hours total. [SSD+13]

As figure 2.15 shows, both supercomputers experienced a constant rate of transient faults but a declining rate of permanent faults over the observed time span.
They also took a look at the fault rate experienced by each vendor. As figure 2.16 shows, there is a substantial difference among vendors, with Vendor A having a 3.9x higher total fault rate than Vendor C. It also shows a strong variation between transient and permanent fault rate between each vendor.

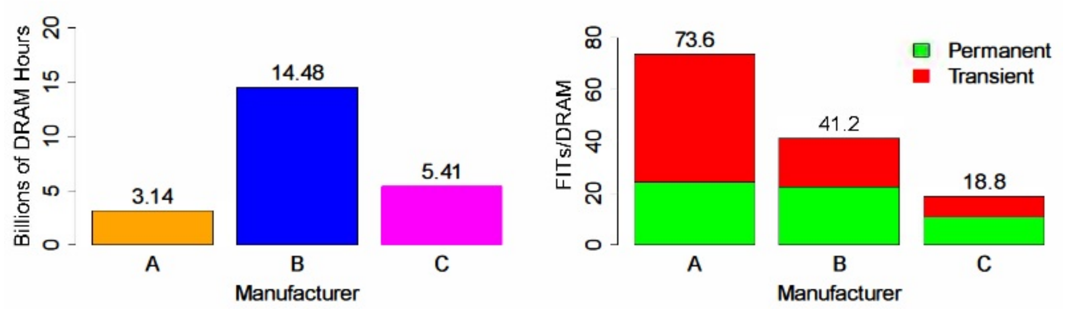


Figure 2.16: Left hand side are the operations hours per vendor. Right hand side are the fault rates per vendor. [SSD+13]

2.4.2 Temperature based Failures

- **Thermal fatigue:** Semiconductor devices are normally subjected to of repeated cycles of heating and cooling, be it through a changing computational workload or a constant changing environment. Such temperature cycles cause mechanical stress on a semiconductor device since different materials have different thermal expansion coefficients. Over time this stress causes material fatigue which can result in a magnitude of different failures like: hairline crack, solder fatigue, bond-wire fracture, cracking of the die, etc. [LS14]

- **Electro-migration:** It describes phenomenon of material transportation caused by the gradual movement of Ions in a current-carrying conductor. Electrons hat are flowing as electric current collide with the Ions and cause them to migrate. This leads to voids in the conductor which again increases the resistance of the conductor due to loss of metal and ultimately will lead to disconnection. Higher temperature will increase the rate of electromigration and with it reduces the Mean Time To Failure (MTTF). [LS14]

- **Frequency, current, voltage drift:** In a Semiconductor device, properties such as frequency, leakage current, threshold voltage, device characteristics, etc. drift due to temperature changes. Such drifts can cause malfunctions, instabilities and general inconsistency in device functionality. Nowadays circuits are designed with temperature stabilization techniques that consider the temperature fluctuations a circuit encounters during their expected life time. For example, as mentioned in the previous chapter, the retention time of DRAM decreases exponentially as ambient temperature increases. This can be countered by adapting the periodically refresh cycle accordingly. [LS14]

- **Circuit protection:** The Thermally Accelerated Age Factor (TAAF) usually rises roughly exponentially with temperature. So, it's no wonder that some modern circuits like CPUs are designed to reduce the thermal stress they inflict on themselves to prolong their lifetime. Such countermeasures can be voltage and frequency scaling as well as thermal throttling. The issue is that if the effects of such countermeasures are not considered in the design of a device they can lead to reduced functionality and reliability in certain situations. [LS14]

2.4.3 Temperature effects on light based Time of Flight (ToF) devices

Even though the basic working principle of a ToF devices is quite simple, they still can be very accurate and precise devices with the right configuration. Depending on the use case, ToF devices come with very different configurations, range, energy consumption, accuracy and price points. For example, depending on what frequency and wavelength a ToF device light pulse emitters is set to, it can have different properties in regards to accuracy, range, energy consumption, eye safety, etc.. It usually depends on the use case for the ToF device which wavelength it uses in the end.

Some common wavelength used in the automotive industry are around 850nm, 905nm, 950nm, 1550nm, with wavelength around 905nm and 1550nm being currently the most discussed wavelength for LIDAR devices in the automotive industry. Another part that can change with the use case is the light pulse emitter. If the use case allows it some cost-efficient devices might use LEDs instead of Lasers Diodes as light pulse emitters.

Though this diversity makes it hard to quantify the measurement errors caused by temperature there are some universal points for that Manufacture should watch out for:

- **Warm-Up induced drift** [HK91]: As with any electric device, ToF device experience a Warm-Up phase where the temperature of the device changes from to environment temperature to its working temperature. The time it takes for a device to thermal stabilize and the amount of drift can vary from device to device. For example, Hebert and Krotkov see in their work "*3-D measurements from imaging laser radars: how good are they?*" [HK91] experienced a range drive of around 40cm over 30 minutes as their amplitude-modulated continuous-wave laser radar pointing at a target six meters heated up form 21C to 45C. Interesting here is that this measurement drift is not necessary due to a frequency shift of the light pulse emitters but can also happen due to the other components in the ToF device. In modern ToF devices manufactures usually embed some temperature compensation algorithms in their products to counteract such measurement drift over time.

- **Frequency shift** [YWP15]: As mentioned in the previous subsection, a change in temperature causes a frequency shift. The severity of shift depend on factors like wavelength and what light pulse emitters is used. But there are also some other considerations to make. Yulianto et al. investigating in their work "*Temperature Effect towards DFB Laser Wavelength on Microwave Generation Based on Two Optical Wave Mixing*" [YWP15] the temperature effect when mixing two single mode lasers with equal properties and output. When mixing two lasers it is important to keep the temperature of the laser in order so that the wavelength difference between the two lasers is stable. Because of

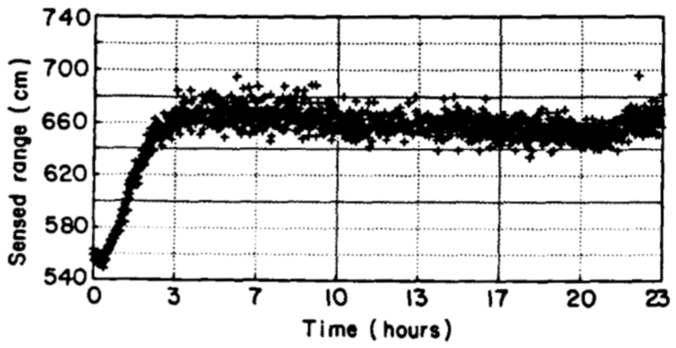


Figure 2.17: Range measurements vary over time. The target is a sheet of cardboard 6 m in front of the scanner [HK91]

this, it requires an automatic control method that changes the temperature of one laser so that it matches the temperature of the other laser. Another effect discussed in the paper is that the increase in temperature results in decreasing intensity of the laser output power which also needs to be controlled by varying the current injected into the laser.

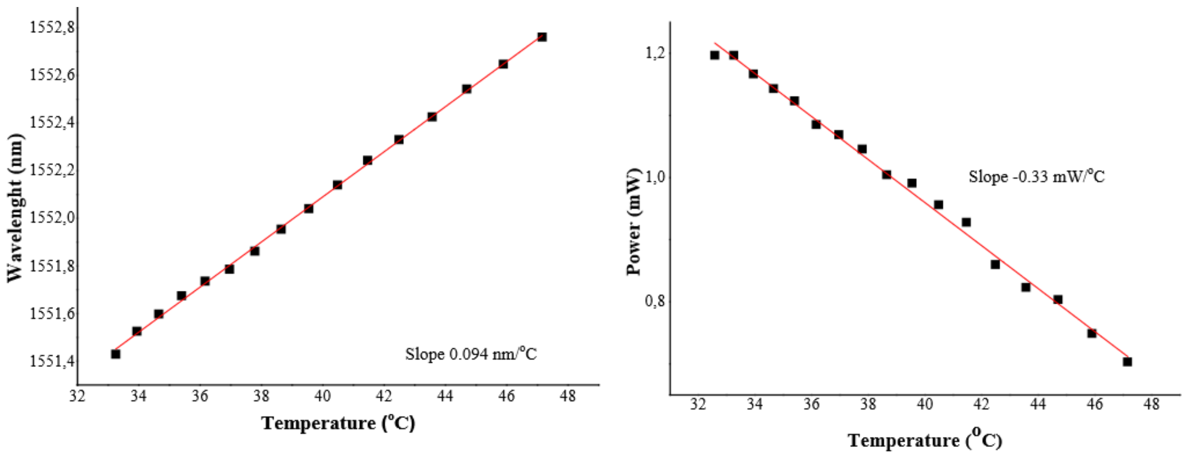


Figure 2.18: Left hand side shows the wavelength variation versus laser temperature . Right hand side shows the optical power laser output with respect to increasing the laser temperature [YWP15]

- **Mode-Hopping** [AVG15]: As shown on the left-hand side of figure 2.19, lasers are affected by an effect called mode-hopping where the laser oscillate between different lasing modes or wavelengths. The effect itself is temperature independent but as Alhashimi et al. show in their work "*Joint Temperature-Lasing Mode Compensation for Time-of-Flight LiDAR Sensors*" [AVG15] the position modes vary with changing temperature. From the right-hand side of figure 2.19, they concluded that for their device the manufacture did not compensate for Mode-Hopping.

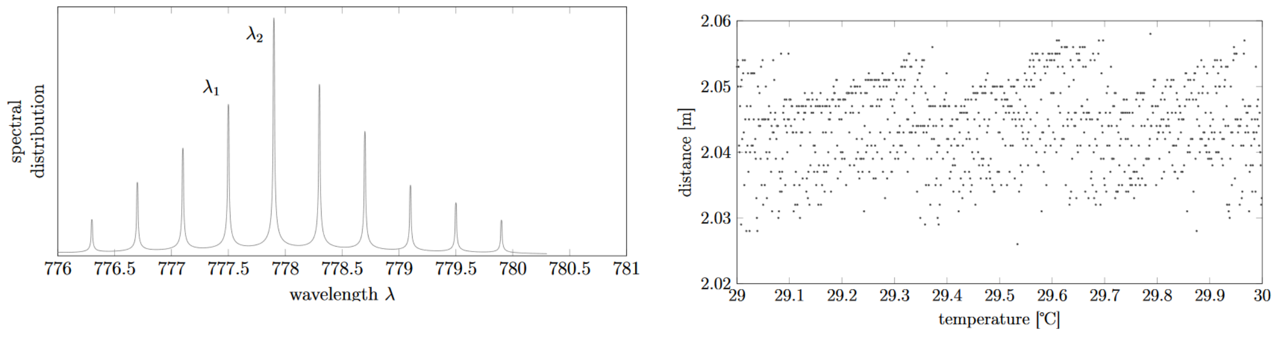


Figure 2.19: Left hand side shows the average spectral distribution for GaAsP lasers at the nominal temperature of 21C. Right hand side shows the Dependency of the distance measurements on the device temperature for a SICK 200 device pointing to a xed object in a controlled environment. The device does compensate for temperature change but not for the Mode-Hopping effect [AVG15]

Chapter 3

Design

3.1 Overview

The goal for this thesis is the conceptually design and implementation of a software-based Fail-Operational Environmental Perception System. In this case Software based means that the software should detect some potential hazardous events and react in a fail-operational way to them. The design of a simple environmental perception system would look roughly as seen in figure 3.1.

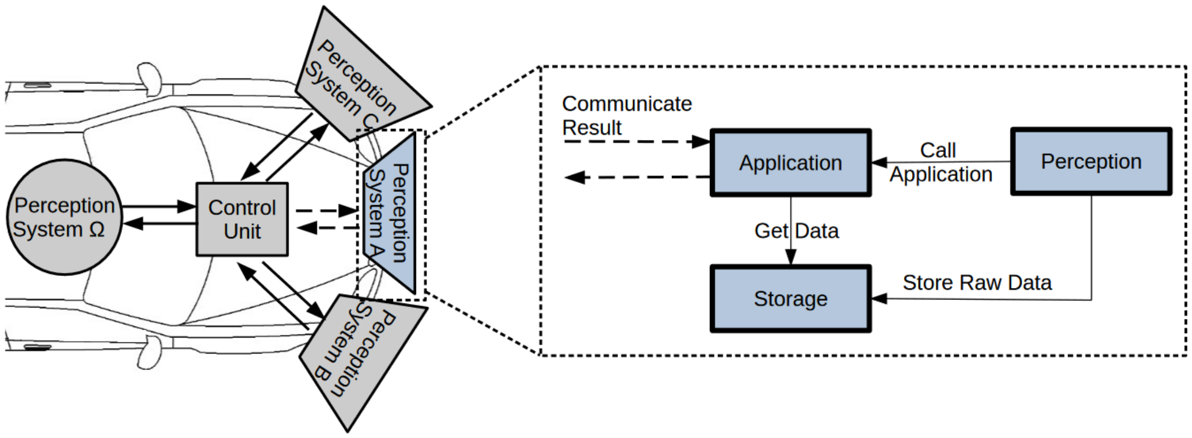


Figure 3.1: Rough sketch of a basic environmental perception system.

As visualized in figure 3.1, the perception device would simply store the image to the memory and calls the application. The application would fulfill some perception task like the detection of lanes, traffic lights/signs and so on. Accordance to [ISO18a] part 3, potential hazards are identified following an analysis of the operational situations of the "item". The potential hazards need to be categorized into severity, probability of exposure and controllability. This categorization is used to determine an Automotive Safety Integrity Level (ASIL) for the potential hazard. The ASIL is assigned to the formulated safety goal(s) and the safety requirements are derived from the safety goals. The following sections will go through each of these steps and explain what the thought process behind them was.

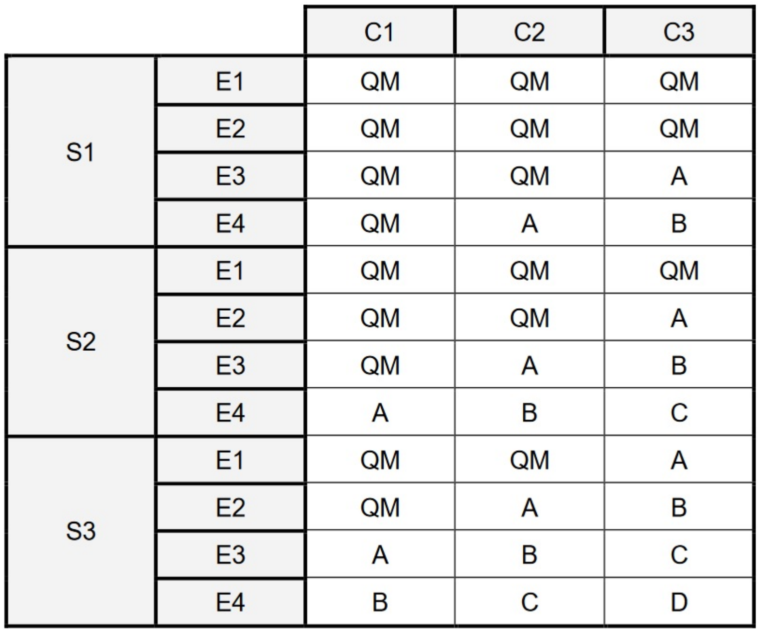
3.2 Hazard Analysis and Risk Assessment (HARA)

As mentioned before this thesis concentrates on software-based solution so the analysis will be software level. As seen in figure 3.1, the basic model is quite abstract and as such the analysis needs to be extended depending on the actually used perception device, storage and application. The following list gives a simplified HARA with using the steps previously discussed in Chapter 2.3.2, to give a better insight of the model:

- **Situation analysis and hazard identification:** Hazards that comes to mind when looking at figure 3.1 are:
  - Complete failure of one or all of the components (perception device, storage, or application). This might happen due to production or design errors in each component. The consequence is that the perception system cannot fulfill its purposed functionality.
  - Memory corruption. This means the data in the storage might get corrupted or some memory cells fail for one reason or another. The consequence for such a hazard is that the application might not create an accurate enough output or in the worst case might cause the application to crash.
  - Heat is a hazard for any electronic component. The consequences of heat range from accelerate the aging process of electronic components and with it causing an early end of life cycle, to frequency shifting causing inaccurate measurements of a perception system, to unexpected loss of efficiency and computing power (e.g. some kind of thermal throttling for self-protection of electronic components).
  - Overload due to unexpected workloads. Do to issues with another component the computational overhead of the perceptions system might be used for another and currently more important task. This could cause the application and the extra workload to create delayed output.
- **Hazard Classification:** Since this model does not define an explicit use case the hazard classification unavoidably has to be more general. So, for all hazard the classification will be:
  - **S3**: In worst case fatal injuries.
  - **C2**: Normally controllable, since an issue is detected there is normally a cluster of other perceptive systems or a redundant system that can take over some functionality of the failing system.

  - **E2**: Low probability.

- **ASIL Determination:** When looking at figure 3.2 the ASIL for the above Hazard Classification is ASIL A.



		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 3.2: ASIL Determination [ISO18a].

- **Safety goal formulation:** Safety goals (SG) are determined for each evaluated event from the hazard analysis.
  - SG1: Detect memory corruption in the storage component and react fail-operational manner if they occur.
  - SG2: Avoid heat exposure and heat generation due to excessive usage.
  - SG3: Avoid high utilization due to unexpected workload and react in a fail-operational manner.
  - SG4: Avoid complete failure of components and react in a fail-operational manner if component failure occurs.

3.3 Safety Concept

This System seen in figure 3.1 would depend on that the memory will never fail and the processing unit would be cool enough that it never thermal throttles and that even over years phenomenon like electromigration wont cause any issues to the processing unit or other components. In case of Smart Mobility, the assumption is that each vehicle needs to run reliable for years under ever chaining thermal and kinetic conditions. When thinking about the safety goals SG1 to SG3 there are two simple aspects with that we could improve the concept of figure 3.1. First would be to regulate the load on the CPU depending on its current temperature. This can improve the longevity of the CPU and its surrounding components as well as prevent thermal throttling in critical situation in that the CPU has to compute a high load. The Second would be to integrate a memory module that not only corrects memory errors but also reacts to complete memory failure of a block by shifting the data accordingly and if no space is left reduce the image quality to let the camera image fit to the memory. Now let us look at the reworked concept in figure 3.3.

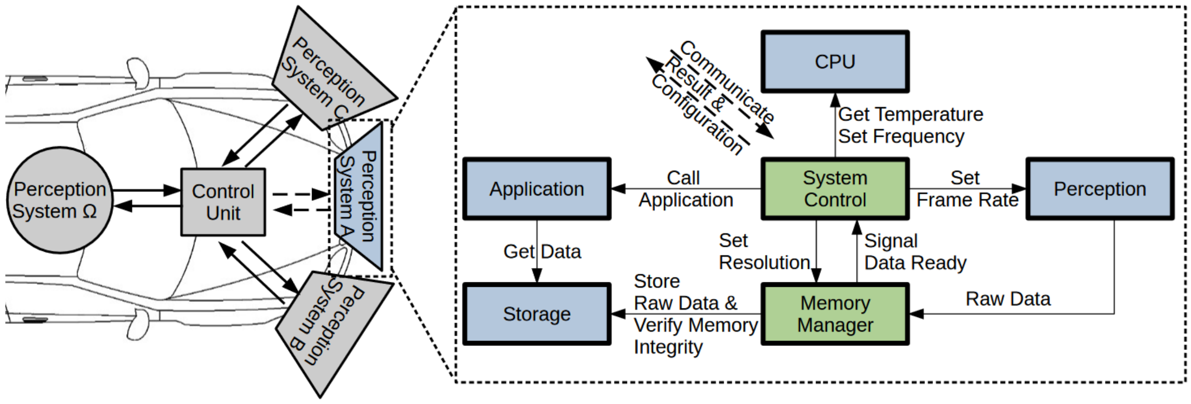


Figure 3.3: Reworked basic environmental perception system.

Beside the CPU we see that following two blocks added to figure 3.3:

Memory Manager receives the raw data from the camera and stores it to the memory. It should be able to verify the integrity of the memory and handle memory errors accordingly (fulfill SG1). On a successful storage it should signal the System Control part. Furthermore, it should be able to reduce the resolution of the received image to a desired fidelity (first part to fulfills SG2 and SG3).

System Control main task is to configure the CPU, Camera, and Memory Manager in regards to Frequency, Frame Rate and Resolution depending on the current CPU temperature and the configured options (second part to fulfills SG2 and SG3). The desired configurations are assumed to come from outside the model and set depending on the current situation the desired target frame rate, resolution and system temperature.

The safety goal SG4 cannot be controlled by the perceptions system itself. It needs to be handled by the above lying “item“, with redundancy or workload distribution to other systems.

3.3.1 Software Concept

With the safety goals SG1-SG4 and the reworked basic system sketch seen in figure 3.3 the next step is to define the basic software architecture to achieve the desired functionalities.

Logical View

Here we take a simple look at what each part of the software system should provide as terms of services. From the basic system sketch we can see that the software part of the prototype can roughly segmented into four different parts. As seen in figure 3.4 the logical segmented software services are:

- *System Control*: The main software component of the prototype. Its goal is to connect the software parts with each other, as well as situational configure them. Depending on the current settings and CPU temperature the System Control component should be able to configure the other components accordingly.

- *Communication*: The communication component of the prototype should be able the to connect to the Control Unit. When connected it should send the results from the application to the control unit and receive setting changes from it. In case of a setting change, the communication component should notify the system control component about the changes.

- *Memory Manager*: The memory manager component should process the data it receives from the camera. Processing in this case means to store the data depending on the provided settings and handling of memory failures.

- *Application Example*: The application component should contain a simple example on how the data can be processed. It is used for testing purpose and should be simple to replace.

Procedural interworking

Here we take into account some non-functional requirements and show the workflow of the system. The activity diagram seen in figure 3.5 gives a rough overview of the internal workings of each class and how they interact with each other and the environment. As seen in figure 3.5 when the prototype is initialized it reads its desired default configuration form a configuration file and initializes the classes accordingly.

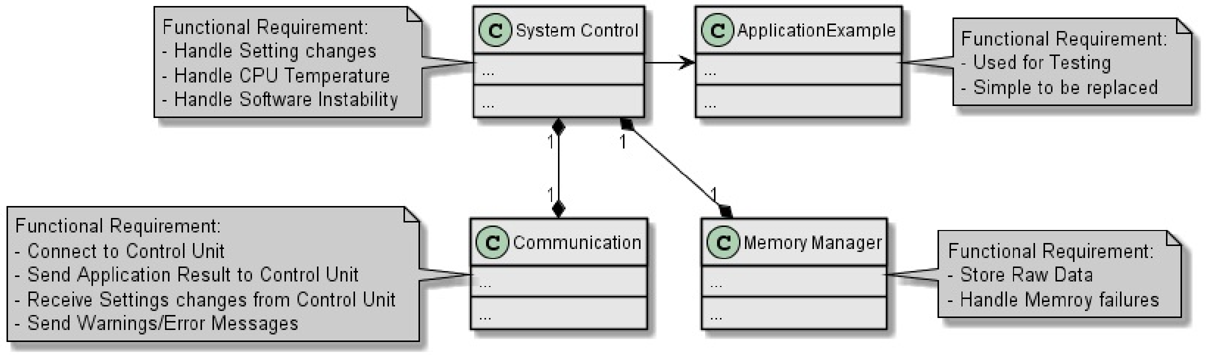


Figure 3.4: Logic View Software Architecture.

After the initialization there are following five asynchronous parts of the prototype running simultaneously:

- *System Control classes*: As the name might suggest the System Control classes are built to handle the System Control requirements established in the logic view. The classes contain all configuration related data and a thread initialized on startup periodically checks the current system status (Current Settings, Temperature, CPU Frequency, CPU Utilization, etc.) and applies changes depending to the System depending on the current status and configuration. To note is that since the checks are performed periodically, configuration send by the control unit are not applied immediately.

- *Camera and Memory Manger Class*: This part of the prototype manages each frame the camera records. After the camera is started it sends the pointer to the raw data to the function that was registered by the startup configuration. In this case it is the "onNewData" function of the Memory Manger class. The function stores the raw data into a different buffer while checking the integrity of the stored data and handles errors according to the current configuration. Depending on the set configuration from the system control part of the prototype, the function might also decrease the frame size. After the frame is ready the application part of the prototype is signaled that a new frame is ready. To note here is depending on the image size and current configuration this part of the prototype can be quite computationally intensive.

- *Application Class*: The application class is an easy to replace class that currently is creating a gray Image, depth Image and an angled view onto the 3D Point Cloud received form the Memory Manger Class. The goal is to have a class that is easily adapted to different use cases. The current flow of the application class is that the main thread waits for a signal from the Memory manager class that a new image frame is ready and calls the processing function of the application. After processing the frame, the communication class is called to send the results to the control unit

- *Communication Class*: The communication class is the bridge between the prototype and the control unit. It sends the current frames, temperature, CPU information and memory information to the control unit and receives the desired configuration from it. The receiving part of the communication is its own thread that waits for connection requests and data from the control unit. Configuration changes received by the control unit are set in the system control classes and will be applied with the next periodical check. The sending part is a function called by the application. To note is that currently only one connection can be active at a time and it should be taken care that the time needed to send the data does not exceed the frame time.

- *Control Unit*: In the prototype example the control unit is the UI that visualizes the data. In a working case scenario, the control unit would be a device that is connected and receives data from multiple camera applications and processes the data according to the devices purpose.

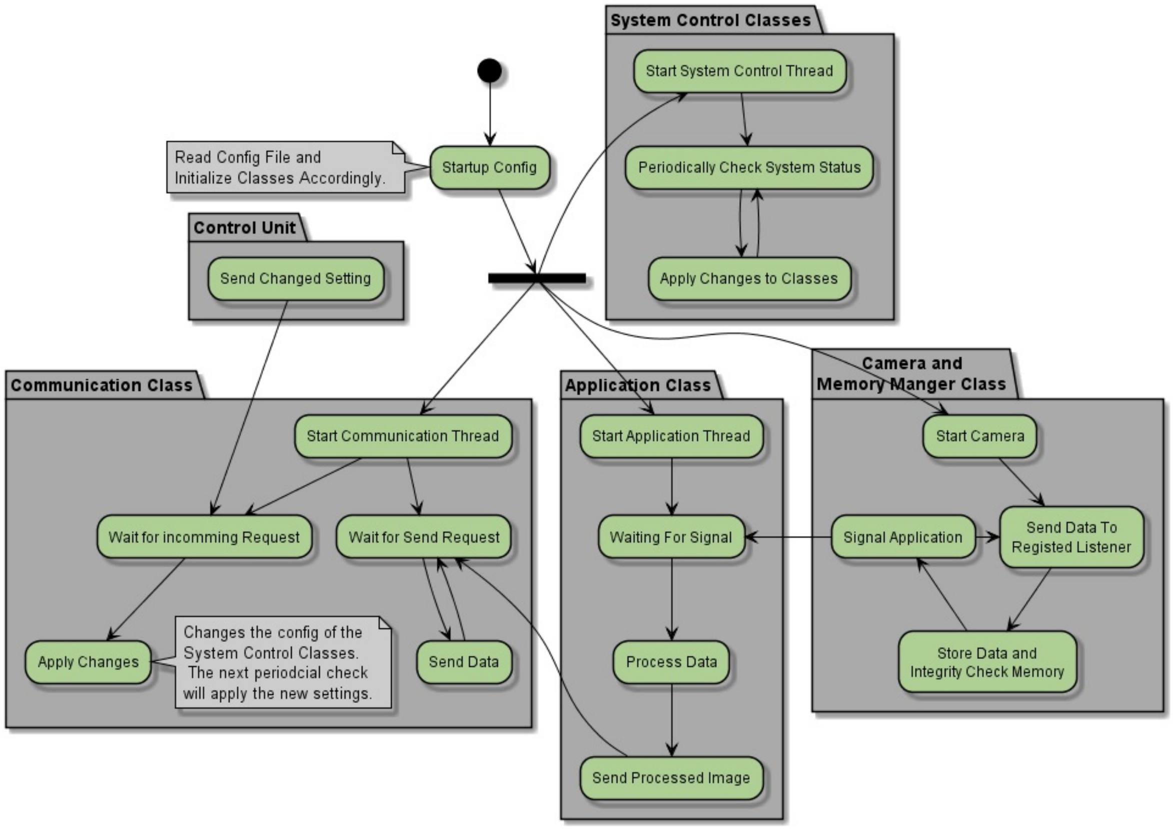

System Control Classes
Start System Control Thread
Periodically Check System Status
Apply Changes to Classes
Read Config File and Initialize Classes Accordingly.
Startup Config
Control Unit
Send Changed Setting
Communication Class
Start Communication Thread
Wait for incomming Request
Wait for Send Request
Apply Changes
Changes the config of the System Control Classes. The next periodcial check will apply the new settings.
Send Data
Application Class
Start Application Thread
Waiting For Signal
Process Data
Send Processed Image
Camera and Memory Manger Class
Start Camera
Signal Application
Send Data To Registed Listener
Store Data and Integrity Check Memory

Figure 3.5: Process View Software Architecture.

3.3.2 Asynchronous Parts

From figure 3.3 in section 3.1 we see that there are some parts of the model that can (and should) run parallel to each other. Ass seen in figure 3.6, apart from the obvious asynchronous parts Camera, CPU and Storage access, it shows that the System Control and Communication run mostly independent from each. The figure shows that the dependent parts of the concept is the Application call from the System Control and the communication from the Camera to the Memory Manager. The Camera should not wait for the Memory Manager and the Memory Manger should only store data into the Storage according to the configuration and signal System Control. System Control should run independent since it needs time for a configuration change to have a measurable impact on the CPU temperature.

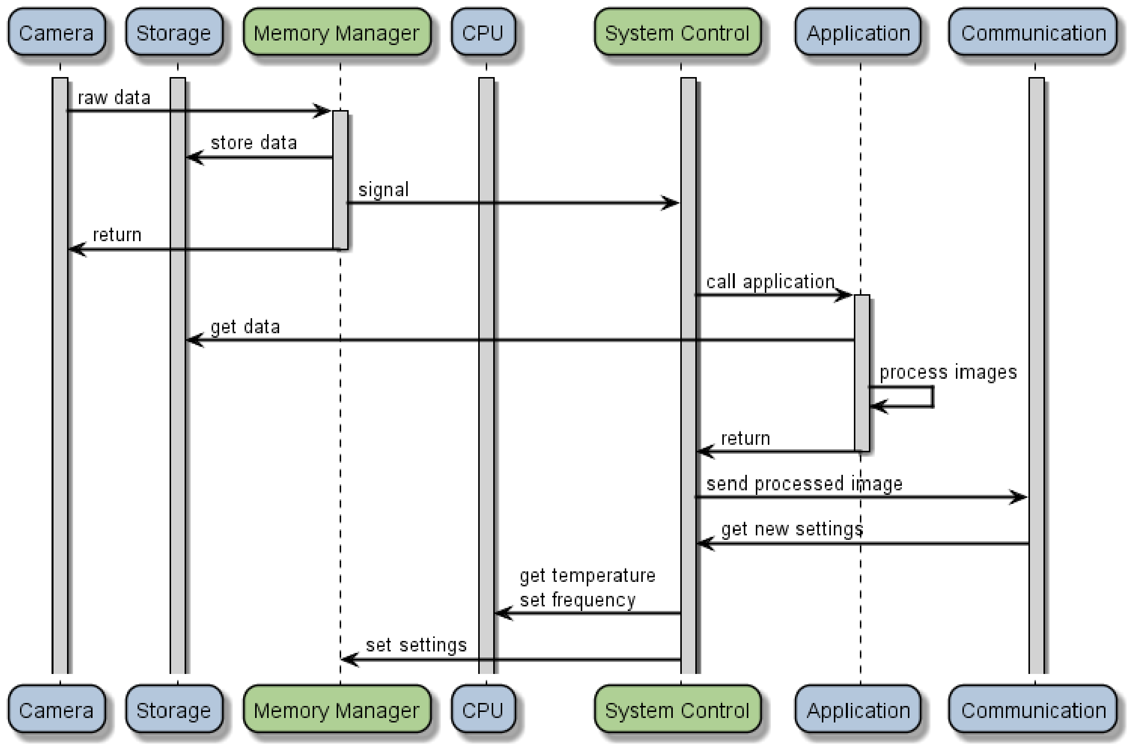


Figure 3.6: Parallel running threads of the environmental perception system.

3.3.3 Memory Handling

As already mentioned, the Memory Manager module should be able to check the memory integrity and reduce the image size if needed. With the first part, the checking of the memory integrity, is not the protection of malicious software or physical attacks on the memory meant but rather a solution similar to ECC memory. The assumption here is that the Memory Manger will get for each single pixel a complete set of data (e.g. gray value, color value, distance, position, certainty of measurement, ...) with a check sum at the end. When copying the data to the memory a new checksum is calculated and compared to the received checksum. Depending on the configuration on a failing check the memory section should be dropped or the write process should be tried again. On a dropped memory section, the data is simple written to the next free section. It's assumed that a dropping memory sections is a very rare event. To keep track on which memory section is in use, an array is needed that stores the indices of the used memory sections. Figure 3.7 shows an example of a memory block before and after a copy processes with multiple failures and some dropped memory sections.

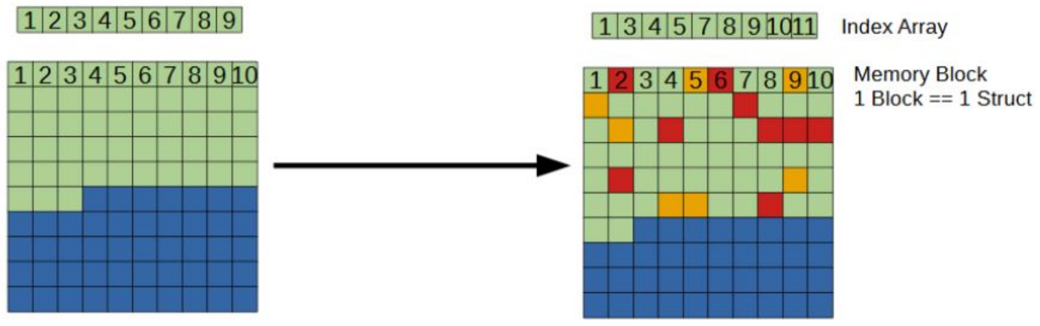


Figure 3.7: Data shifted in memory block due to memory integrity failures.

Another task of the Memory Manger is to reduce the image size if needed. A simple method is shown in figure 3.8, where a reduction factor dictates which n-th pixel gets actually stored into the memory. Of course, this method can easily be improved by using a scaling filters instead of simply taking a single point. This is especially interesting when we consider that a part of the received data are 3D-Points from a point cloud.

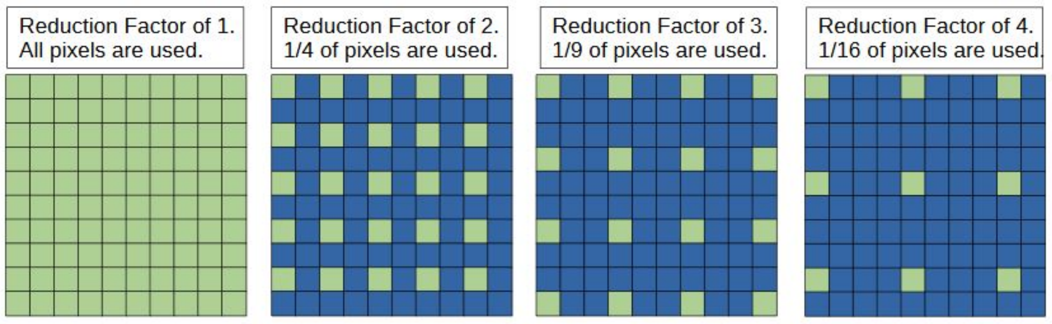


Figure 3.8: Image size reduction based on a reduction factor.

Chapter 4

Implementation

4.1 Development Environment

Figure 4.1 gives an overview about the composition of the development environment as well as the relation of the hardware and software parts used in the development of the prototype. The following subsections will explain the hardware and software parts that are seen in figure 4.1.

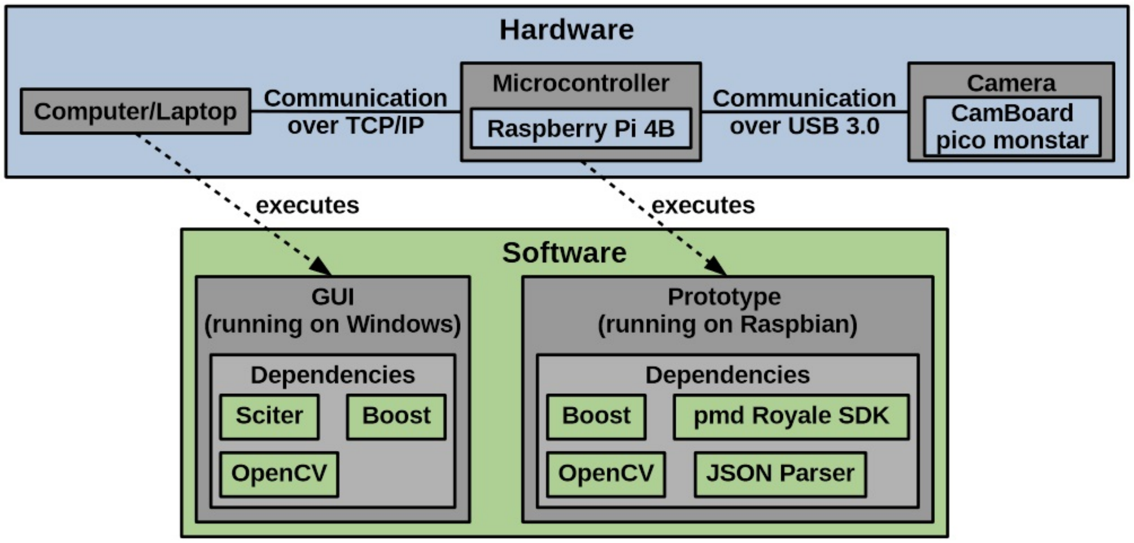


Figure 4.1: Hadware & Software relation.

4.1.1 Hardware

- **Raspberry Pi** [Fou19]: is one of the most well-known single-board computers. It comes in different configurations, sizes and versions and officially supports a Debian Linux based operating system called Raspbian. As seen in figure 4.1, the model used as an execution device for the application and the extended modules is the Raspberry Pi 4B. A reason why the Raspberry Pi 4B was chosen for this thesis was because it supports USB 3.0 which is needed for the camera device. Another reason why the Raspberry Pi 4B was chosen is that even though it has good performance for its price, it also has some thermal issues in it current state. The SoC on the Raspberry Pi 4B is a Broadcom BCM2711 is a quad-core cortex-A72 (ARM v8) 64-bit SoC and starts thermal throttling at 80°C, which can be reached by extended heavy usage. This model

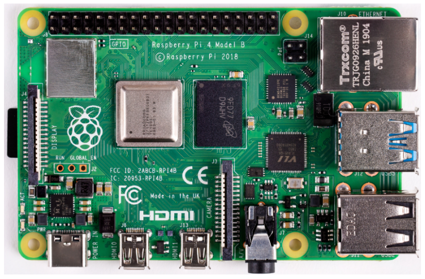


Figure 4.2: Raspberry Pi 4B.

has following core specifications:

- Broadcom BCM2711, ARM Cortex-A72 64-bit SoC @ x 1.50GHz
- 4GB LPDDR4-2400 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11a/b/g/n/ac WLAN, Bluetooth 5.0
- 1 Gb LAN
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header
- 5V/3A DC power input

- **CamBoard pico monstar** [pmd18]: Is a USB powered, high-end 3D camera development kit from pmd and is used as the camera for this thesis. The CamBoard pico monstar is a Flash Time-of-Flight (ToF) camera and with 352 x 287 pixels it has a relative high resolution for at ToF camera. Furthermore, the camera has a configurable frame rate from 2 FPS to 60 FPS. This flexibility allows for a good use case selection. Another advantage of the camera is that pmd provides a powerful software suit called *Royale* for the camera. Royale is a cross-platform SDK containing all the logic to operate the 3D camera. As seen in figure 4.1 the pico monstar is connected to the Raspberry Pi 4B via USB 3.0. For the camera to work correctly the appropriate driver, which are provided by the Royale SDK, need to be installed on the system.

Figure 4.3: CamBoard pico monstar.

This model has following core specifications:

  - IRS1125C Infineon REAL3 3D Image Sensor IC based on pmd intelligence
  - 0.5m 6m measurement range (Framerate dependent)
  - Up to 60 fps (3D frames) Framerate
  - 352 x 287 (100k) px Resolution
  - 4x VCSEL, 850 nm, Laser Class 1
  - 4.5W max for IRS chip, illumination and USB3.0

- Laptop: Is a simple of the shelf Lenovo Thinkpad E495. The GUI running on the laptop is used to simulate the communication between perception system and overlying system, as well as to gather data about the current status of the Raspberry Pi 4. As seen in figure 4.1 the Raspberry Pi 4B communicates with the laptop with TCP/IP over an Ethernet connection. The laptop can be replaced with any other device capable of running the GUI.

4.1.2 Programming Languages

The primary programming language for the implementation is C++. C++ was selected simply because a preexisting C++ example of the Royale SDK already had the form as in figure 3.1 and the decision was made to use this example as basis. Secondary programming language is TIScript (an extension of JavaScript) with some HTML/CSS elements to realize the GUI.

4.1.3 Libraries and SDKs

The application makes use of following libraries:

- **pmd Royale SDK** [pmd19] provides the logic to operate pmd-based ToF cameras. The Royale SDK has cross-platform compatible and runs on Windows, Linux for ARM based systems, Ubuntu Linux, macOS and Android. This allowed us to develop and test the prototype in Visual Studio under a Windows operating system and deploy the solution onto the Raspberry PI 4B which runs a Linux operating system for ARM based systems. The solution created with the Royale SDK is to communicate with the camera and configure it if needed.

- **Boost** [lib19] is a powerful collection of free peer-reviewed portable C++ source libraries. The prototype uses the *Asio library* [Koh19] contained in the Boost collection which can be used to created simple servers and clients that are able to handle TCP/IP requests. For the thesis, the Asio library is used for the TCP/IP communication between Raspberry Pi 4B and the GUI, which consists of configuration data, system status and captured frames.

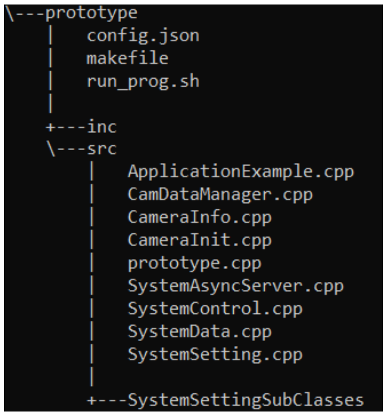
- **JSON.hpp** [Loh19] is a simple to use json file parser for C++ written by Niels Lohmann. It was chosen because its implementation is uncomplicated, since it is one header file contains all functionality and because it provides a clean way to parse json files. In the prototype it is used to read the startup configuration file “config.json“, which contains the configuration for the camera and the application running on the Raspberry Pi 4B

- **Sciter** [TIS19] is a embeddable HTML/CSS/script engine for modern UI development. It is used to create the GUI for this thesis. The advantage of the Sciter engine is that a GUI can be defined with HTML and CSS which allows a quite uncomplicated and fast GUI development. Furthermore, for computationally intensive task, it is capable to of callbacks between the GUI and native code. Sciter allows for easy deployment by using a single compact DLL which is placed right next to the executable of the application. Because of this the Sciter engine seems to be a quite elegant solution to use for the GUI prototype especially when comparing it to other solutions that allow GUI definition with HTML and CSS like for example electron, which runs the Chromium browser engine in the background.

- **OpenCV** [IC19] (Open Source Computer Vision Library) is an open source computer vision and machine learning software library aimed at real-time applications. Since OpenCV is a cross-platform it is a good fit for this thesis as it was used for the Application part of the prototype and in the GUI. In the prototype OpenCV is used in the example Application part to create the depth image, gray image and a rotated point cloud. In the GUI OpenCV is used to handle the incoming images form the prototype so they can be displayed in the GUI.

4.1.4 Toolchain Overview

The prototype and the GUI where both develop using *Microsoft Visual Studio Community 2019* edition which was running naively on a *Microsoft Windows 10* operating system. For future projects that might use parts of this code, it is not necessary to use Visual Studio to change and compile the code but it provides a good starting point since the Visual Studio project files already contain the right structure. One thing to take care of when using the Visual Studio project files is that the path to the external libraries is updated accordingly. Thanks to the cross-platform capabilities of all the libraries and SDKs used in the development, it is possible to write and debug the code for the prototype and the GUI on a Windows machine and later compile the code with a simple makefile on the Raspberry Pi for further testing. In the actually development the basic interaction between the prototype and the GUI was tested on the same machine before deploring the code to the Raspberry Pi for a more in depth testing. In general, any environment can be used for development and debugging the prototype and the GUI so long as one is careful to link the right versions of the shared libraries when developing and knows that the GUI can look significantly different when running it on different operating systems.



```
\---prototype
    |   config.json
    |   makefile
    |   run_prog.sh
    |
    +---inc
    \---src
        |   ApplicationExample.cpp
        |   CamDataManager.cpp
        |   CameraInfo.cpp
        |   CameraInit.cpp
        |   prototype.cpp
        |   SystemAsyncServer.cpp
        |   SystemControl.cpp
        |   SystemData.cpp
        |   SystemSetting.cpp
        |
        +---SystemSettingSubClasses
```

Figure 4.4: Project File Tree.

Figure 4.4 shows a shortened file tree of the prototype folder. As seen in the figure the naming scheme of the files does not completely correlate with software concept discussed in Chapter 3. The example application is stored in the *ApplicationExample.cpp*. The *CamDataManager.cpp* handles the incoming data from the camera and contains the memory manager. The *CameraInfo.cpp* and *CameraInit.cpp* are provided by the pmd Royale SDK and are used to print information and initialize the camera. The *prototype.cpp* contains the main and start up procedure of the program. The *SystemAsyncServer.cpp* contains the communication part of to the gui. The *SystemControl.cpp*, *SystemData.cpp* and *SystemSetting* contain the system control part of the program for general control functionality, data handling and settings handling.

4.2 Class Design

The first set of classes where derived from the concept seen in figure 3.3 and further refined as the development of prototype advanced. Figure 4.5 shows a simplified class diagram without methods or variables. As seen in figure 4.5 the classes of the prototype can be summarised into following sections:

- **System Control section:**
  - **SystemControl:** The logic part of the prototype is handled by the SystemControl class. It contains the decision making part of the prototype. Its main task is to periodically check the current status of the system regarding temperature, utilized frame rate and CPU utilization. Depending on the user defined settings configuration it changes the image resolution, set frame rate or CPU frequency. As seen in figure 4.5 the SystemControl class connects the Memory Manager and the Communication part of the prototype.
  - **SystemSetting:** In the setting part of the prototype are all settings stored that are defined by the user on startup or through the GUI. The idea of the SystemSetting class and its parent classes was to split off the handling of user defined data and application defined data. The SystemSetting classes mostly consist of getter and setters.
  - **SystemData:** The SystemData part of the prototype handles the data that the prototype creates and sets. Furthermore, compared to the SystemSetting it contains some algorithm to update existing data or generate new data from the received data.
- **Communication:**
  - **SystemTcpServer:** Creates the communicator for the prototype. Its task is to accept incoming connections and create a new SystemTcpConnection to handle incoming requests from the GUI and send an appropriate answer.
  - **SystemTcpConnection:** The communication part of the prototype. Its task is to communicate with the GUI. It handles incoming requests from the GUI and send an appropriate answer.
- **Memory Manager:**
  - **CamDataManger:** This class handles the incoming data from the camera and adapts them according to the set settings. Depending on the settings set by the SystemControl class the resolution of the received image is reduced before it gets stored to the memory. Furthermore, the user has the option to enable a memory control code that checks the integrity of the stored data.

- **Application:**
  - **ApplicationExample:** This class contains a simple example where a gray, depth and point cloud image are created from the data. To note is that only the gray and depth image are visualized in the GUI.

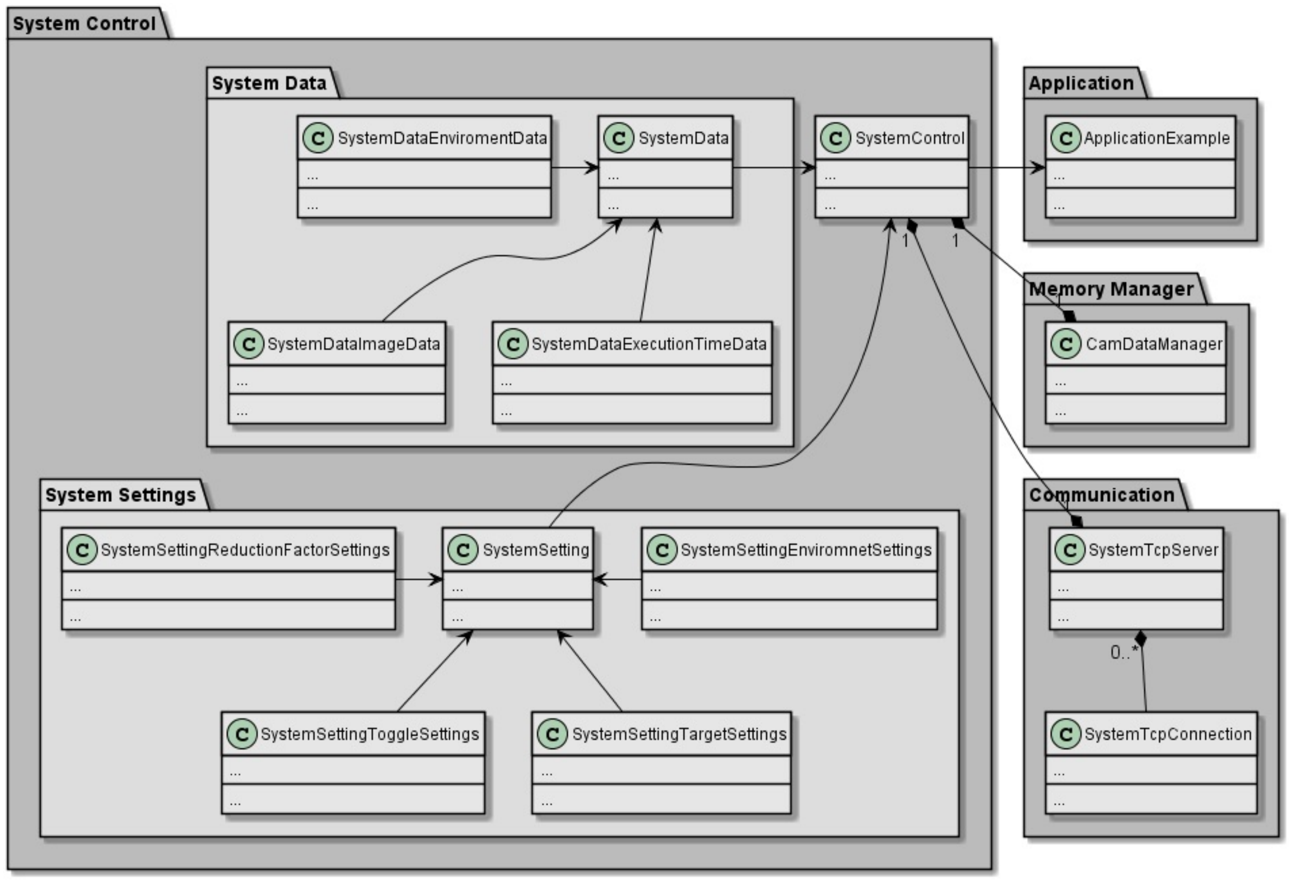


Figure 4.5: Class Diagram (Simple).

4.3 Program Flow

The following subsection takes a closer look at the program flow for certain parts of the prototype.

4.3.1 Startup Program Flow

Figure 4.6 visualizes the sequence on Startup. First System Control, Camera and Application are initialized. After that the configurations read form config.json file are set for System Control and the Camera Memory Manager is initialized accordingly. Default configurations are used if the config.json file does not exist. In the next step the camera parameters (frame rate, resolution, lens parameters) are read and the Memory Manger is set as data listener for the camera. After that the threads for communication, periodical device checks and signal wait are initialized and started. Finally, the capturing is started and the and the program flow changes as described in the next section.

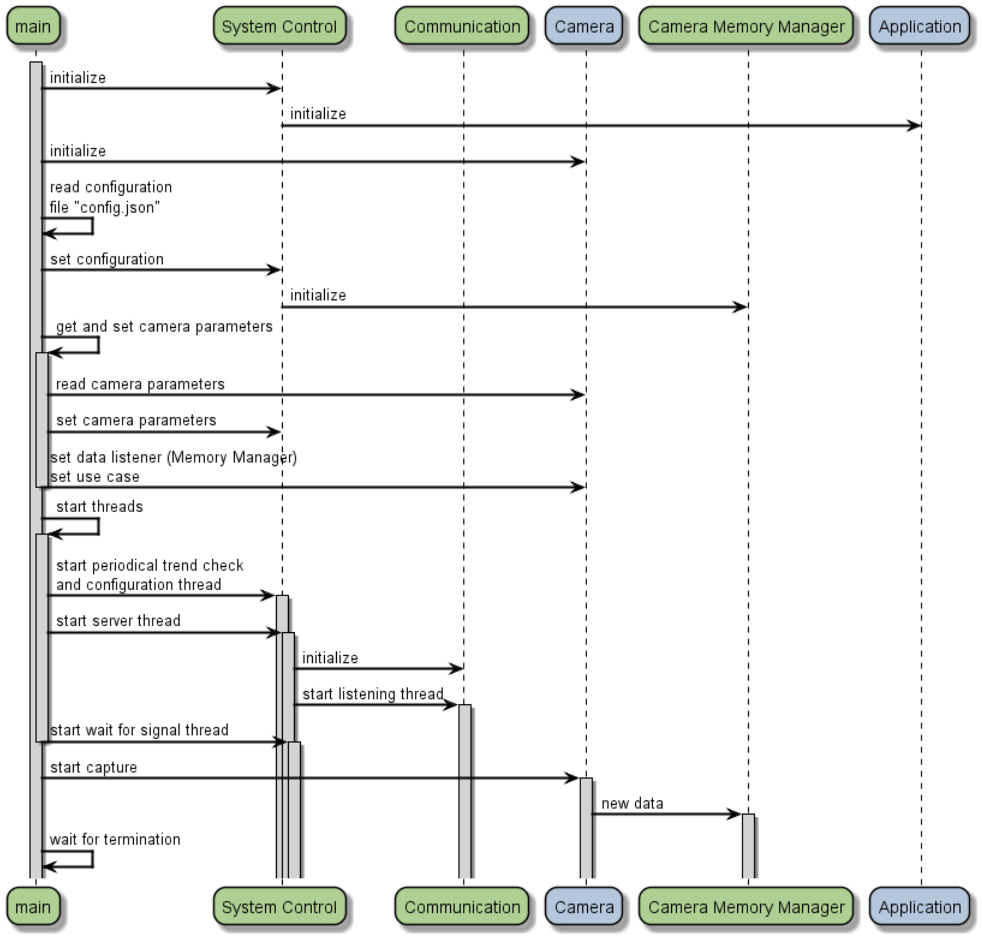


Figure 4.6: Start Up Execution Sequence.

4.3.2 Single Frame Program Flow

Figure 4.7 visualizes the sequence for a single frame. The camera calls the *onNew-Data()* function with a pointer to the raw data after each captured frame. The Memory Manager locks the access to the memory and starts an execution timer. After that the Memory Manager copy the copies and checks the integrity of the memory. According to what the current configuration is the Memory Manager also changes the image resolution at this point. When the image is successfully stored in the memory the Memory Manager wakes up the System Control signal thread and unlocks the memory lock. After being woken up by the Memory Manger the System Control locks the memory and calls the Application to process the image. When the Application is finished processing the image the System control sends the finished image to the Control Unit (in this case the GUI). Finally, the System Control stops the timer started by the Memory Manger, unlocks the memory and goes back to sleep, waiting for the next signal from the Memory Manager.

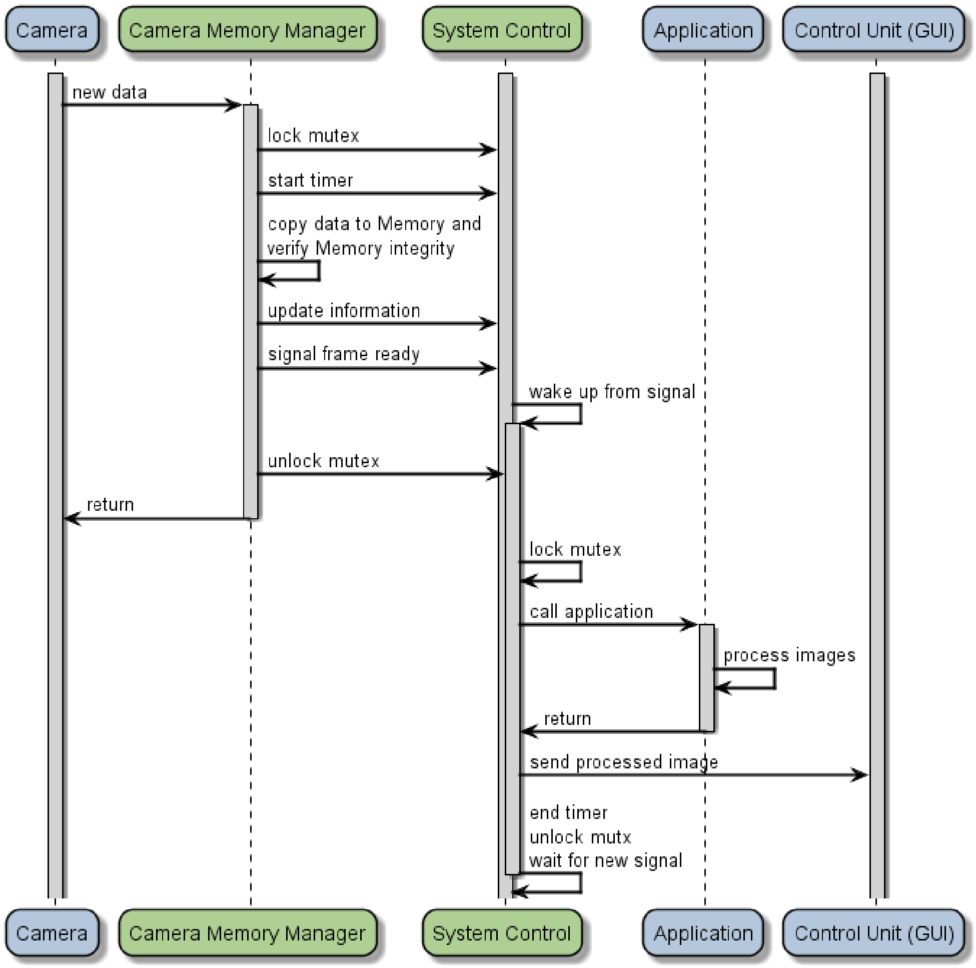


Figure 4.7: Single Frame Execution Sequence.

4.3.3 Periodically System Status Check Program Flow

Figure 4.8 visualizes the sequence for a periodical system status check. When the thread in System Control wakes up it reads the current CPU temperature and calculates the temperature trend with respect to previous readings. Furthermore, it calculates the actually utilized frame rate and might change the configured frame rate of the camera. Thereafter the thread processes if changes are needed for configured Frame Rate and CPU frequency depending on the current configured option from the GUI, temperature trend, utilized frame rate and CPU utilization. Depending on result the thread configures the Camera, CPU and Memory Manager accordingly. Finally, the thread goes back to sleep for a predefined amount of time.

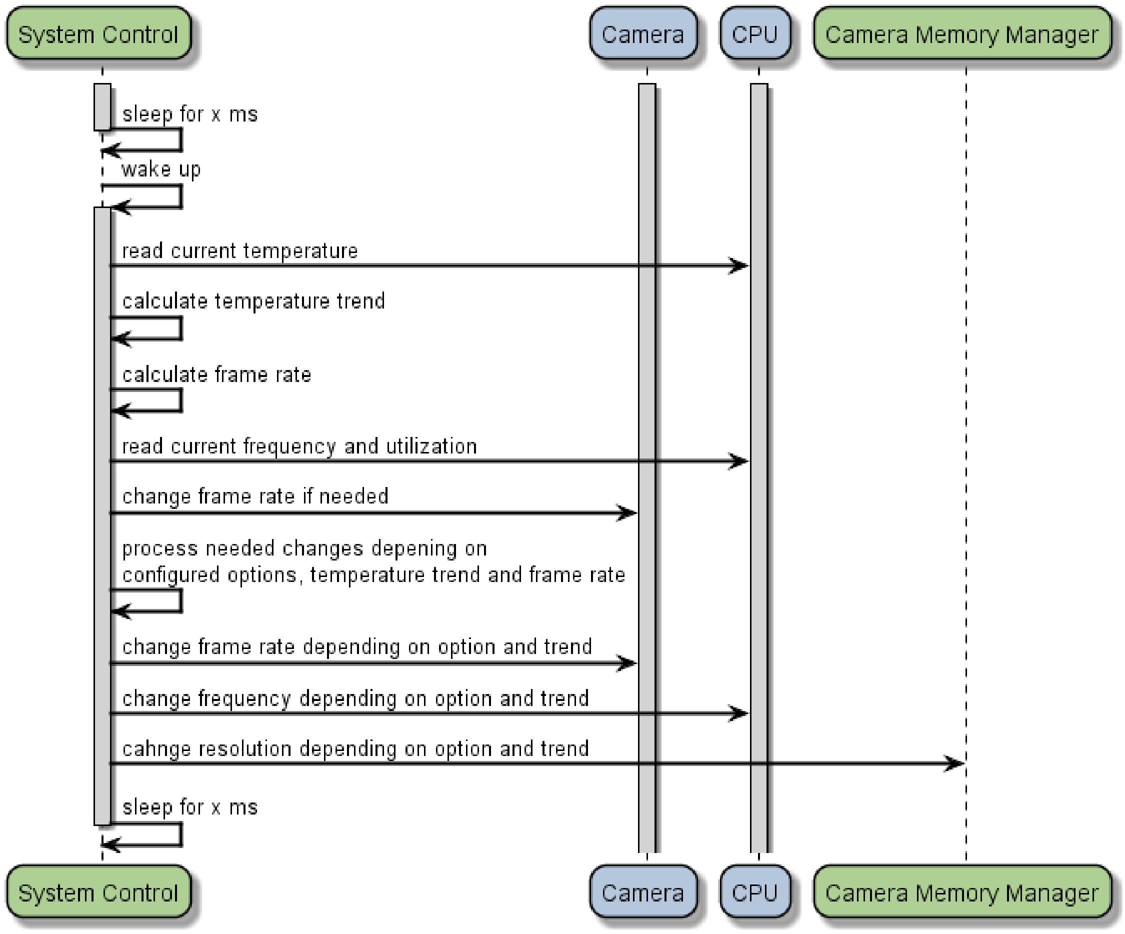

System Control
Camera
CPU
Camera Memory Manager
sleep for x ms
wake up
read current temperature
calculate temperature trend
calculate frame rate
read current frequency and utilization
change frame rate if needed
process needed changes depening on
configured options, temperature trend and frame rate
change frame rate depending on option and trend
change frequency depending on option and trend
cahnge resolution depending on option and trend
sleep for x ms
System Control
Camera
CPU
Camera Memory Manager

Figure 4.8: Periodically System Status Check Execution Sequence.

4.3.4 Server Receives Message Program Flow

Figure 4.9 visualizes the sequence when the prototype receives a message form the GUI. The fixed sized incoming message from the GUI gets buffered and processes in the communication part of the prototype. The message contains the current desired configuration and a request flag on what data the GUI wants from the prototype. Depending on the request, data gets poled from the System Control. Finally, the communication part sends back the desired data to the GUI.

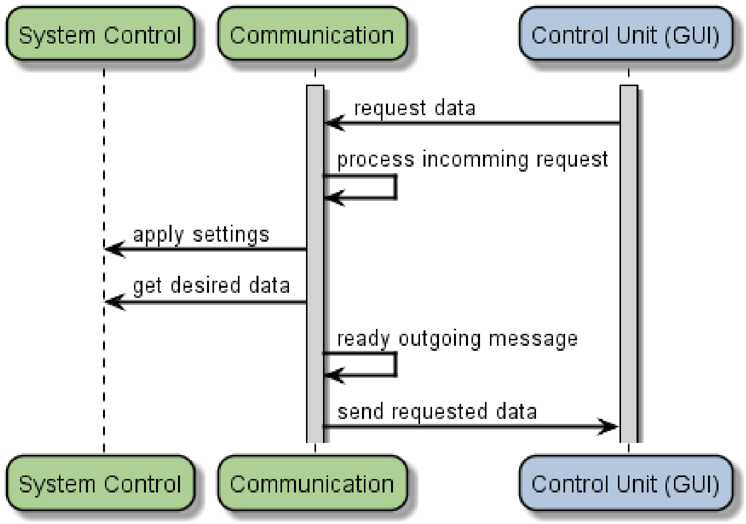
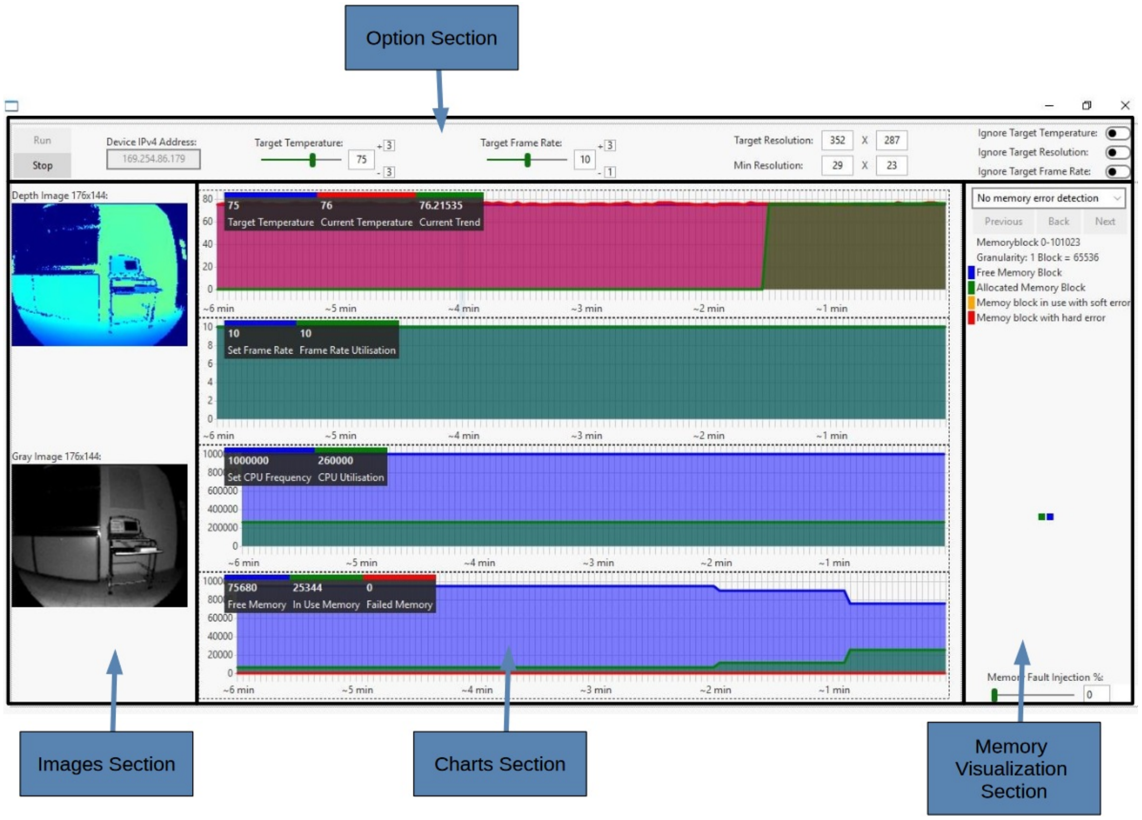


Figure 4.9: Server Receives Message Execution Sequence.

4.4 Graphical User Interface

The GUI was originally developed for debugging purpose and to better visualize the status of CPU temperature and set Frame rate. It later on evolved to the main interface to configure the application. The GUI was realized with the help of Sciter engine API. As already mentioned in the *Libraries and SDKs* section, Sciter allows GUI definition with HTML and CSS. The computational part of the GUI is handled through JavaScript and simple callbacks to the C++ that handles the server communication and readies the incoming data for visualization.



Device IPv4 Address: 169.254.86.179

Target Temperature: 75

Target Frame Rate: 10

Target Resolution: 352 X 287

Min Resolution: 29 X 23

Ignore Target Temperature:

Ignore Target Resolution:

Ignore Target Frame Rate:

No memory error detection

Memoryblock 0-101023

Granularity: 1 Block = 65536

Memory Fault Injection %: 0

Figure 4.10: Sections of the GUI.

As seen in figure 4.10, the GUI is divided into following four sections:

Option Section: The option part of the GUI allows to configure desired temperature target, the desired frame rate, a minimum and maximum resolution as well as three options for further granularity on the behavior of the prototype. Though these settings can be changed any time, it depends on the update cycle how fast the prototype applies the changes. The effects of the tree Boolean options, *Ignore Target Temperature*, *Ignore Target Resolution* and *Ignore Target Frame Rate* can be seen in figure 4.1. Also, in the option section is the *Device IP Address* field where the IP address of the device running the prototype is entered. The run and stop buttons then connect/disconnect to the device. After a connection to the prototype is established the GUI periodically sends request data for the charts and memory view and the prototype sends after each frame the resulting image. The testing version of the GUI also contains a start recording and start test run button which generate a CSV file containing the data visible in the GUI with time stamp.

Images Section: Displays the gray and depth images that the application creates in their current resolution. The text above the image displays what the current resolution of the images is. Compared to the charts and memory section of the GUI, the image section receives the images without sending a request for it after each frame. Meaning that the image section is a live feed of the processed images.

Charts Section: Displays four charts that show the last 120 data points received by the application in the last 720 seconds (6 minutes). So, the GUI sends every three seconds a request to the prototype for a data update. The four charts see in figure 4.10 are sectioned for:

- Temperature information: Displays the *desired temperature* set by the GUI in blue, the *current temperature* of the CPU in red and the *temperature trend* in green. The temperature trend line is generated from the last 30 temperature readings and projected onto the last 10 data points. The goal of the line is to visualize how the temperature behaved in average over the last 30 seconds. This is interesting when the device is exposed to high temperature fluctuations.
- Frame rate: Displays the current configured *frame rate* of the camera in blue as well as the *frame rate utilization* in green. The Frame rate utilization is the number of frames that where actually processed by the application on average over the last 3 seconds.
- CPU frequency: Displays the current *set CPU frequency* of the device in blue and the current *CPU utilization* in green.
- Memory usage: Displays the *free memory* in blue, the *in use memory* in green and the *failed memory* in red. The free memory is the memory allocated for image storage that is currently not used. While the in-use memory is the memory that is currently used (which means that the image is stored on it). The failed memory is the memory of the allocated image storage that won't be used by the prototype anymore.

Memory Visualization Section: This section visualizes the memory section that is allocated for the images of each frame. Depending on the granularity each block can represent a single structure that contains all information of a pixel or a cluster of said structures. It is possible to step into a cluster of structures by clicking on the corresponding block. The blocks are individual color coded to visualize their status. Blue means the block is free/unused, green means the block is used and there is no issue with it, yellow means that the block is in use but there were some issues and red means that the block is considered as have being failed and that it is not used any more. Also, in this section is the option enable/disable the software ECC and the percentage of memory failure for the test run.

Ignore Target Temperature	**Ignore Target Resolution**	**Ignore Target Frame Rate**	
False	**False**	**False**	Reduce resolution and frame rate till CPU temperature reaches equilibrium.
False	**False**	**True**	Reduce frame rate till CPU temperature is below the target temperature.
False	**True**	**False**	Reduce resolution till CPU temperature is below the target temperature.
False	**True**	**True**	Reduce frame rate and resolution till CPU temperature is below the target temperature.
True	**False**	**False**	Set resolution and frame rate to their defined maximum. The image resolution gets reduced if frame rate is below 50 percent of the set target frame rate.
True	**False**	**True**	Set resolution to it's defined maximum. Then try to level out CPU temperature.
True	**True**	**False**	Set frame rate to it's defined maximum. Then try to level out CPU temperature.
True	**True**	**True**	Reduce resolution and frame rate till CPU temperature reaches equilibrium

Table 4.1: The effect of the Boolean options in the GUI.

Chapter 5

Experimental Results

The focus of this chapter is the description of the different test cases that have been executed and the discussion of their individual result(s). The test cases described in this chapter will be a mix of testing the functionality of the prototype and tests that shall show the fulfillment of the safety goals defined in Chapter 3.2. Figure 5.1 shows the test environment and hardware used for testing. The description of the hardware can be found in the Section 4.1. As seen in figure 5.1 the Raspberry PI 4B was tested with a 3D printed plastic enclosure on. The enclosure has some ventilation holes but no active cooling was used for testing.

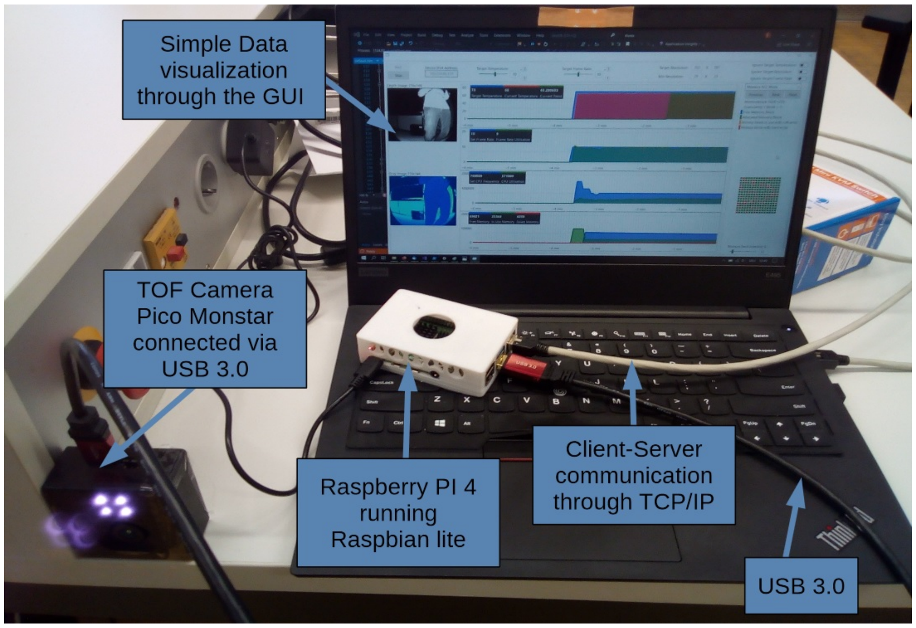

Simple Data visualization through the GUI
TOF Camera Pico Monstar connected via USB 3.0
Raspberry PI 4 running Raspbian lite
Client-Server communication through TCP/IP
USB 3.0

Figure 5.1: Prototype Setup.

5.1 Test Cases

The list below describes the executed test cases in regards of test scenario, test execution and the expected result. The test names for the settings of the GUI Boolean options, as seen in Table 4.1, are shorten versions of their configuration in the form of GUI_{T/F}{T/F}{T/F}. For example, GUI_TFT would means that the option *Ignore Target Temperature* is set to True, *Ignore Target Resolution* is set to False and *Ignore Target Frame Rate* is set to True. Following test cases have been executed:

- **GUI_FFF & GUI_TTT:**
  - Scenario: Reduce resolution and frame rate till CPU temperature reaches equilibrium.
  - Execution: Start with temperature around target temperature. Set all three Boolean options to false. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes resolution and frame rate of the camera such that the temperature of the device doesn't exceeds the target temperature with defined hysteresis.
- **GUI_FFT:**
  - Scenario: Reduce frame rate till CPU temperature is below the target temperature.
  - Execution: Start with temperature around target temperature. Set *Ignore Target Frame Rate* to true and the other options to false. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes only the frame rate of the camera such that the temperature of the device doesn't exceeds the target temperature with defined hysteresis.
- **GUI_FTF:**
  - Scenario: Reduce resolution till CPU temperature is below the target temperature.
  - Execution: Start with temperature around target temperature. Set *Ignore Target Resolution* to true and the other options to false. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes only the resolution of the camera such that the temperature of the device doesn't exceeds the target temperature with defined hysteresis.

- **GUI_FTT:**
  - Scenario: Reduce frame rate and resolution till CPU temperature is below the target temperature.
  - Execution: Start with temperature around target temperature. Set *Ignore Target Resolution* and *Ignore Target Frame Rate* to true and the other options to false. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes resolution and frame rate of the camera such that the temperature of the device false below the target temperature.
- **GUI_TFF:**
  - Scenario: Set resolution and frame rate to their defined maximum. The image resolution gets reduced if frame rate is below 50 percent of the set target frame rate.
  - Execution: Start with temperature around target temperature. Set *Ignore Target Temperature* to true and the other options to false. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes resolution and frame rate of the camera to their defined maximum. The temperature gets ignored. If the device temperature reaches 80°C the effect of thermal throttling should be visible in the utilized frame rate.
- **GUI_TFT:**
  - Scenario: Set resolution to its defined maximum. Then try to level out CPU temperature by changing the frame rate.
  - Execution: Start with temperature around target temperature. Set *Ignore Target Resolution* to false and the other options to true. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes resolution of the camera to its defined maximum. The frame rate of the camera is changed such that the temperature of the device doesn't exceeds the target temperature with defined hysteresis.
- **GUI_TTF:**
  - Scenario: Set frame rate to its defined maximum. Then try to level out CPU temperature by changing the resolution.

  - Execution: Start with temperature around target temperature. Set *Ignore Target Frame Rate* to false and the other options to true. Around 20 minutes test Execution time. Target Temperature set to 70°C. Target frame rate set to 10 FPS.
  - Expected Result(s): The prototype changes the frame rate of the camera to its defined maximum. The resolution of the camera is changed such that the temperature of the device doesn't exceeds the target temperature with defined hysteresis.

- **Memory Failure:**
  - Scenario: The memory cells of the device fail one after another.
  - Execution: Set *Ignore Target Temperature* to true and the other options to false. Define ECC memory error detection and set memory failure rate to 100 percent in the GUI. The device temperature has no importance in this test. Test ends if resolution is below the configured minimum.
  - Expected Result(s): The prototype changes the resolution of the camera to its defined maximum. Thereafter one memory error after another gets detected and the image resolution starts to degrade.

- **Cool down CPU Temperature:**
  - Scenario: The device current temperature is above the target temperature, including the hysteresis. See how long it takes for the device to reach the target temperature.
  - Execution: Start with temperature at least 5°C above target temperature with hysteresis. Set all options to false. Let test run for around 20 minutes.
  - Expected Result(s): The prototype changes the resolution and the frame rate to its minimum till it comes close to the target temperature. Then the prototype will step wise increase the frame rate and resolution till the device temperature reaches equilibrium.

- **Simulate Drive at Test Track:**
  - Scenario: Simulate a test drive for a predefined test track on a city street, highway and country road.
  - Execution: Create a CSV file to simulate the test drive. The file contains the settings (target temperature, frame rate, resolution, options) for each read type (city street, highway and country road, ...) and a list of roads with road type and distance to drive on them.
  - Expected Result(s): See the behavior of the prototype when it changes between different configuration depending on the road that is currently simulated.

5.2 Test Procedure

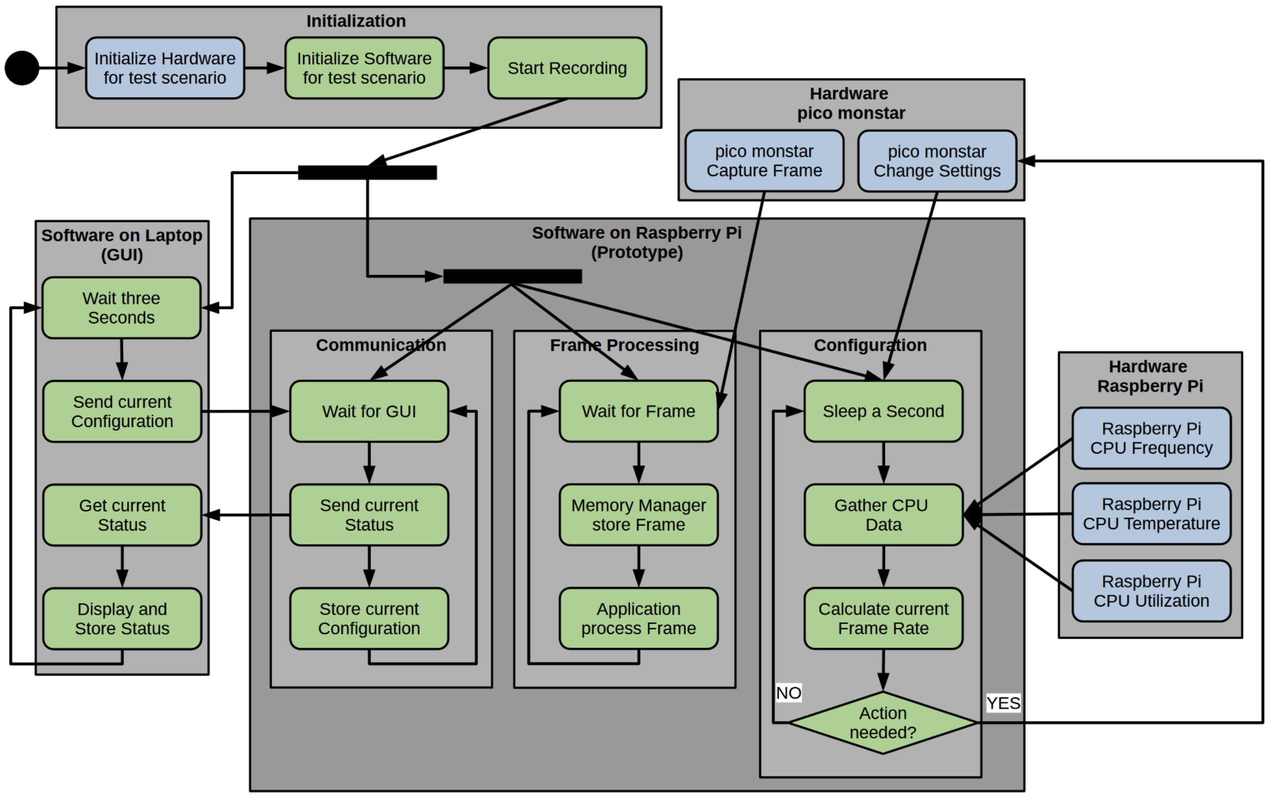

Initialization
Initialize Hardware for test scenario
Initialize Software for test scenario
Start Recording
Hardware pico monstar
pico monstar Capture Frame
pico monstar Change Settings
Software on Laptop (GUI)
Wait three Seconds
Send current Configuration
Get current Status
Display and Store Status
Software on Raspberry Pi (Prototype)
Communication
Wait for GUI
Send current Status
Store current Configuration
Frame Processing
Wait for Frame
Memory Manager store Frame
Application process Frame
Configuration
Sleep a Second
Gather CPU Data
Calculate current Frame Rate
Action needed?
NO
YES
Hardware Raspberry Pi
Raspberry Pi CPU Frequency
Raspberry Pi CPU Temperature
Raspberry Pi CPU Utilization

Figure 5.2: Test Procedure Sketch.

Figure 5.2 gives a general overview on how the test results seen in the Section 5.3 Results where produced. The following points give a more detailed description on the parts seen in figure 5.2:

- The *Initialize Hardware* step contains task like, connect the pico monstar camera to the Raspberry Pi via USB, connect the Raspberry Pi to the Laptop via Ethernet and making sure that current temperature of the Raspberry Pi corresponds to the current test case and if not let it heat up or cool down.
- The *Initialize Software* step contains task like, start the prototype on the Raspberry Pi, start the GUI of the on the Laptop and connect to the GUI prototype running on the Raspberry Pi. Finally set the configuration in the GUI corresponding to the current test case.
- As its name states, the *Start Recording* step starts the recording of the test. Of course, the recording should only be started after hardware and software are in their corresponding testing state. The recording task itself is handled by the GUI.

- The *GUI* part for the testing is quite simple. Every three second the GUI sends it current configuration to the prototype while requesting the current status of the CPU and Frame Rate as well as the last processed frame itself. Since the GUI handles the recording task of the test the received data are not only displayed but also stored into a csv file for further use.
- The *Prototype* part of the testing can be split into three general parts:
  - The *Communication* part handles the communication with the GUI. It stores the settings it received from the GUI, so that the Configuration part can use them. It sends back the CPU and Frame Rate data collected by the Configuration part as well as the last frame created by the Frame Processing part.
  - The *Configuration* part wakes up every second to gather data like temperature, utilization and frequency form the CPU as well as the current frame rate by calculating how many frames got processed since the last time it woke up. Depending on the collected data and the current configuration the Configuration part might change some settings of the pico monstar camera.
  - The *Frame Processing* part simply waits till the pico monstar camera captures a new frame and sends it to the Memory Manager. After the Memory Manger stores the frame according to its current configuration it calls the application to further process the frame.

5.3 Results

This section shows and discuss the results of the previous defined test cases. The figures seen in this section are marked with points of interest (POI) and region of interest (ROI) which will be discussed in more detail.

5.3.1 GUI_FFF & GUI_TTT:

Figure 5.3 shows a 25-minute run, containing around 500 data points. As seen in figure 5.3 the Prototype was able to keep the temperature between the target temperature and the top hysteresis of 3°C (dashed blue lines). *POI 1* marks the point where the temperature went above the target temperature with hysteresis. As seen in the *frame rate chart* and *image size chart* of the figure a single overstepping of the boundary is not enough to trigger a frame rate and image size reduction. Instead *ROI 1* marks the region where a steady increase of temperature is visible and after it passed a certain threshold the frame rate and the image size have been decreased. The *ROI 2* marks a region where the temperature strongly fluctuated and with it the frame rate and image size. The test run ended with a set frame rate of 9 FPS and a resolution of 25 percent (176x144 pixels) of the original image size.

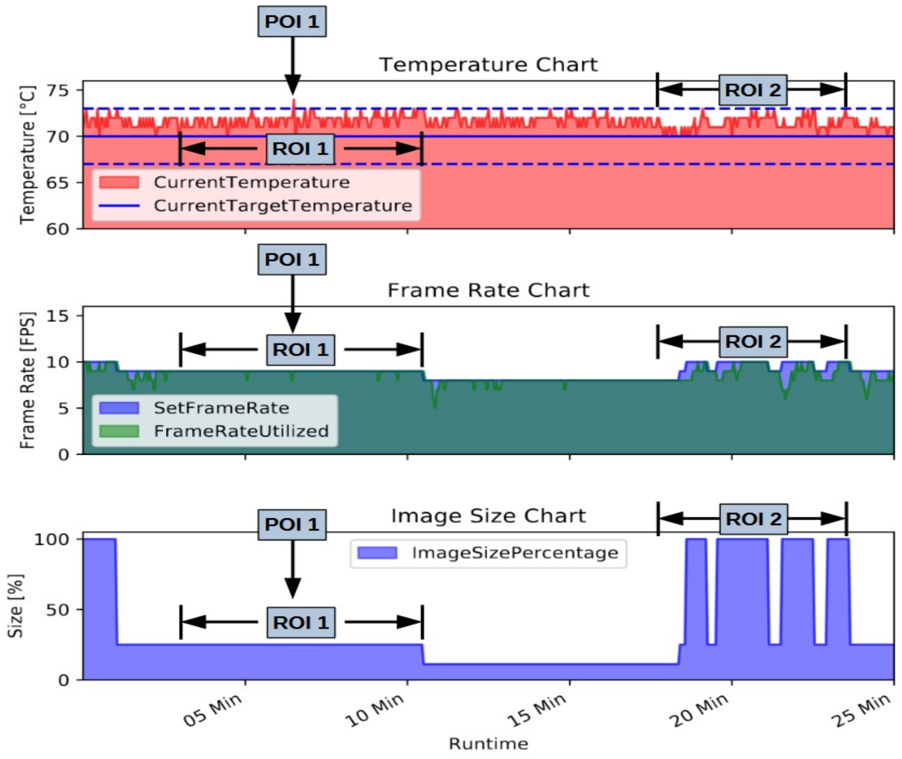


Figure 5.3: Result Chart test case GUI_FFT and GUI_TTT.

5.3.2 GUI_FFT:

Figure 5.4 shows a 25-minute run, containing around 500 data points. As seen in figure 5.4 the Prototype was able to keep the temperature well below the target temperature. Since this option ignores the configured frame rate, the frame rate is set to its minimum of 2 FPS and ignores the target frame rate of 10 FPS. As seen in *ROI 1* this causes a first steep decline below the hysteresis of the target temperature and with this decline the frame rate is increased twice to keep the temperature between target temperature and hysteresis. The test run ended with a set frame rate of 4 FPS and resolution of 100 percent (352x287 pixels) of the original image size.

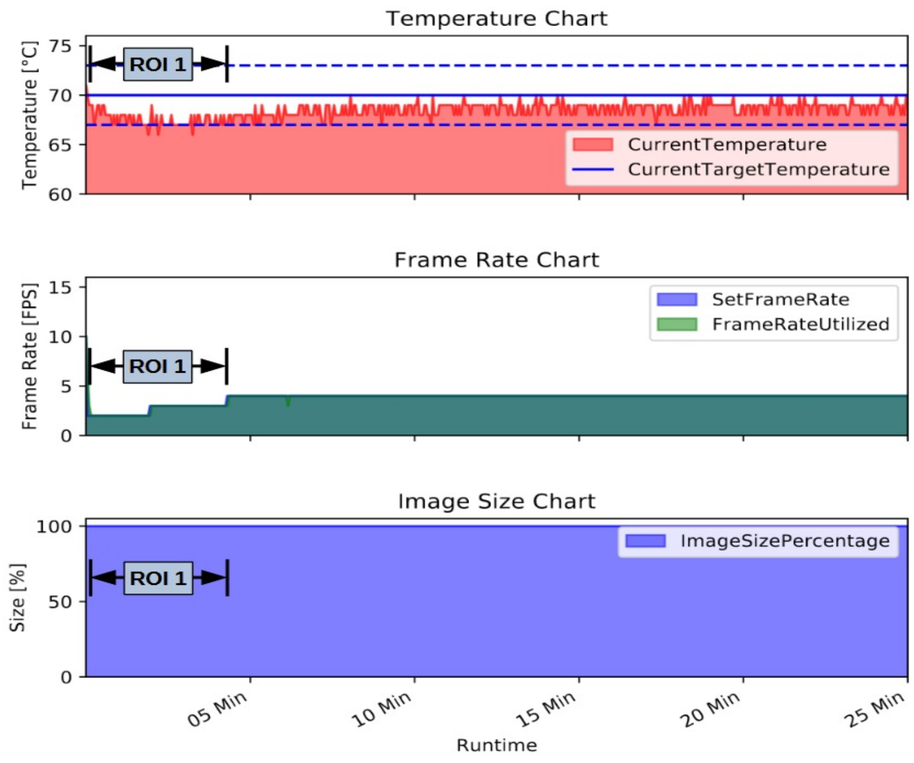


Figure 5.4: Result Chart test case GUI_FFT.

5.3.3 GUI_FTF:

Figure 5.5 shows a 25-minute run, containing around 500 data points. As seen in figure 5.5 the Prototype was able to keep the temperature between the hysteresis of 3° and the target temperature. Comparing the result to the one of the GUI_FFT test it is clearly visible that the frame rate has a higher impact one operating temperature than image size. But this impact might change depending on how computationally intensive the image processing part of the application is. Also compared with the GUI_FFT result where the frame rate is increased twice, the image size is never increased since the temperature never falls low enough. The test run ended with a set frame rate of 10 FPS and a resolution around one percent (30x24 pixels) of the original image size.

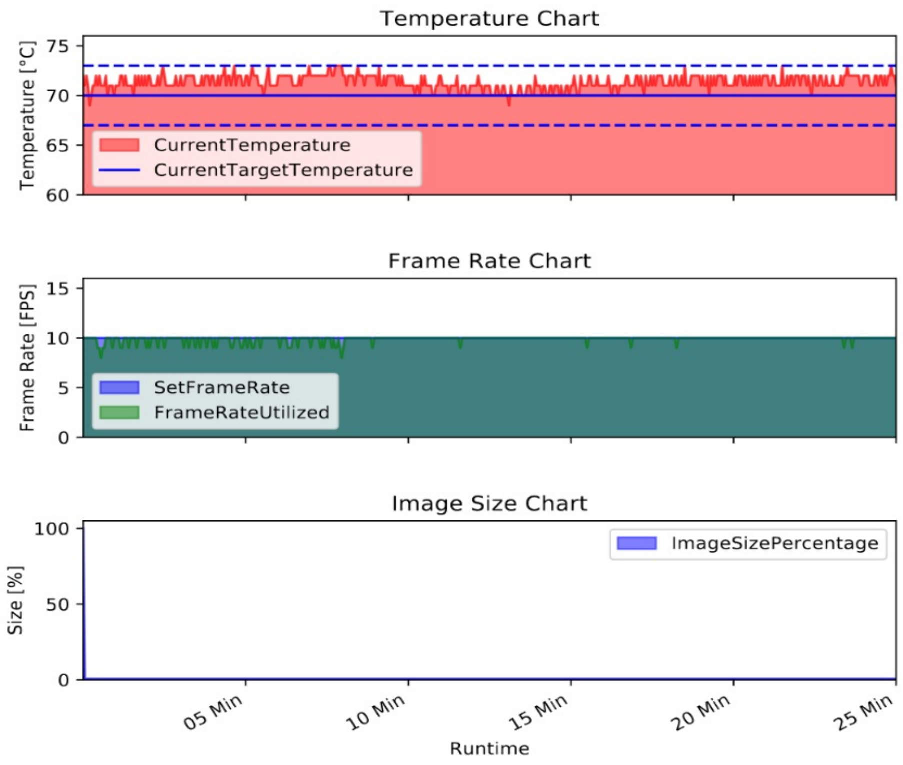


Figure 5.5: Result Chart test case GUI_FTF.

5.3.4 GUI_FTT:

Figure 5.6 shows a 25-minute run, containing around 500 data points. As seen in figure 5.6 the Prototype was able to keep the temperature well below the target temperature. The frame rate and resolution were kept at minimum the entire run. The note here is that even with a very low computational load the device takes a significant time to cool down after reaching a certain disparity between ambient temperature and device temperature. Which will be further visualized in the test case *Cool down CPU Temperature*.

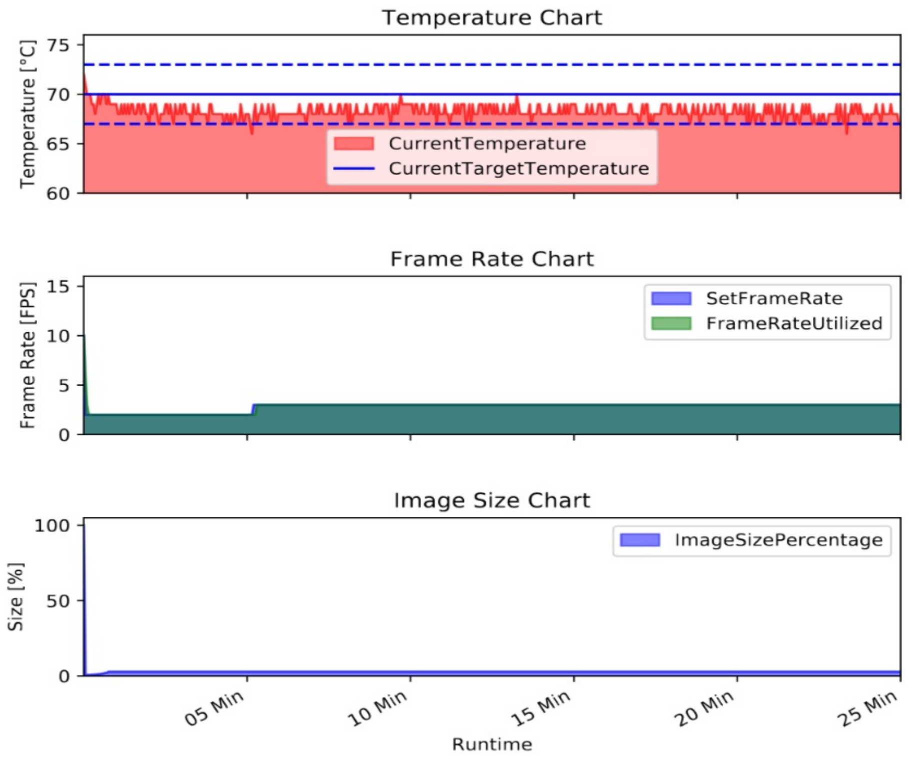


Figure 5.6: Result Chart test case GUI_FTT.

5.3.5 GUI_TFF:

Figure 5.7 shows a 25-minute run, containing around 500 data points. As seen in figure 5.7 the frame rate for this test was set to 15 FPS compared to the 10 FPS in the previous tests. The figure visualizes how the *utilized frame rate* in the *frame rate chart* drops as the device reaches and rises above the 80°C mark. Also visible in the figure is that even when the device thermal throttles he temperature still continues to trend upwards. The test run ended with a set frame rate of 11 FPS and a resolution 100 percent (352x287 pixels) of the original image size.

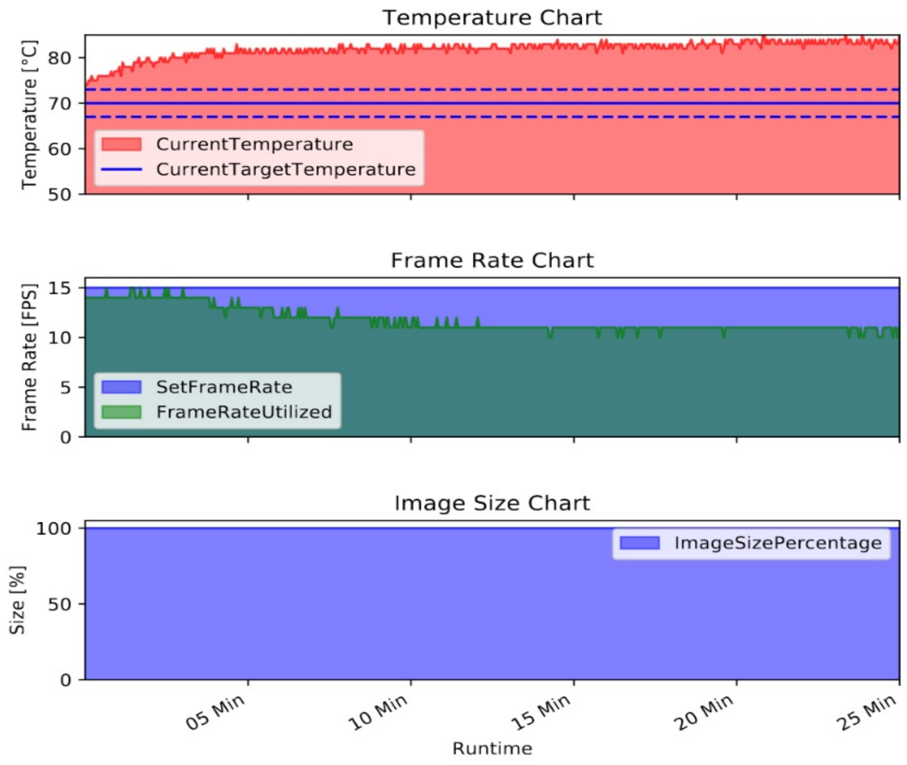


Figure 5.7: Result Chart test case GUI_TFF.

5.3.6 GUI_TFT:

Figure 5.8 shows a 25-minute run, containing around 500 data points. The behavior seen in figure 5.8 is very similar to what the test run for GUI_FFT showed. This is not very surprising since in the test run for GUI_FFT it was already visible that by reducing the frame rate it's possible to keep the temperature below the target temperature. Since this option ignores the configured frame rate, the frame rate is set to its minimum of 2 FPS and ignores the target frame rate of 10 FPS. As such a change causes the temperature to drop rather quickly the frame rate gets increased again. The test run ended with a set frame rate of 5 FPS and a resolution 100 percent (352x287 pixels) of the original image size.

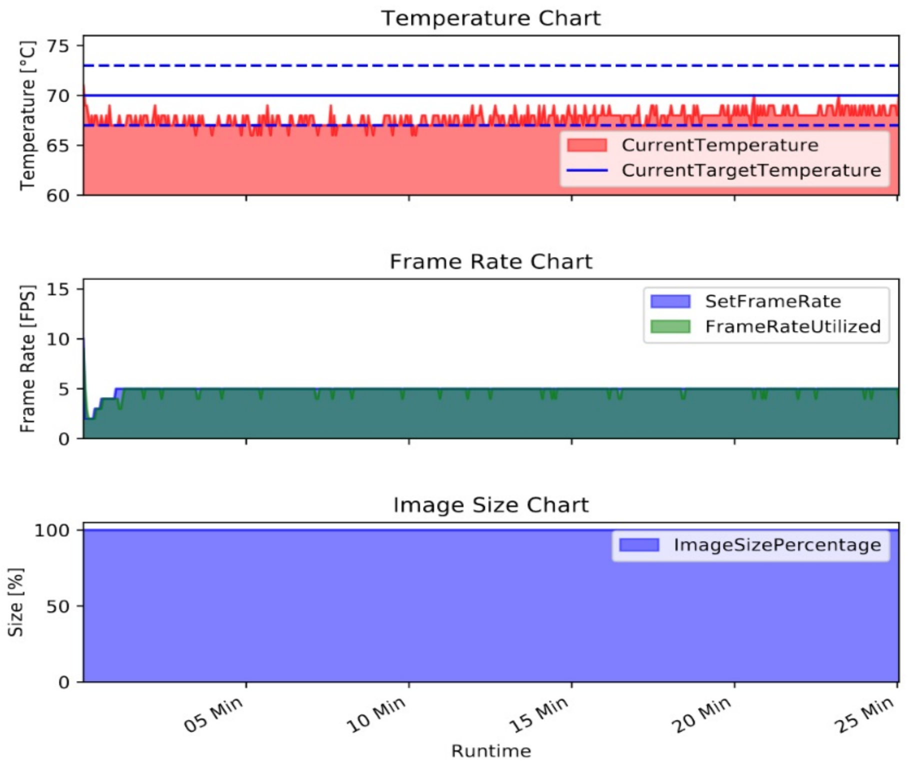


Figure 5.8: Result Chart test case GUI_TFT.

5.3.7 GUI_TTF:

Figure 5.9 shows a 25-minute run, containing around 500 data points. As seen in figure 5.9 the temperature is ignored and increases above the target temperature with hysteresis. The slope of the temperature increase is not as steep as in the GUI_TFF test case but the temperature is still steadily increasing. At the set frame rate the temperature did not level out, even with the image resolution set to its minimum. Still at the end of the test run the temperature was below the 80°C mark. The test run ended with a set frame rate of 15 FPS and a resolution around one percent (30x24 pixels) of the original image size.

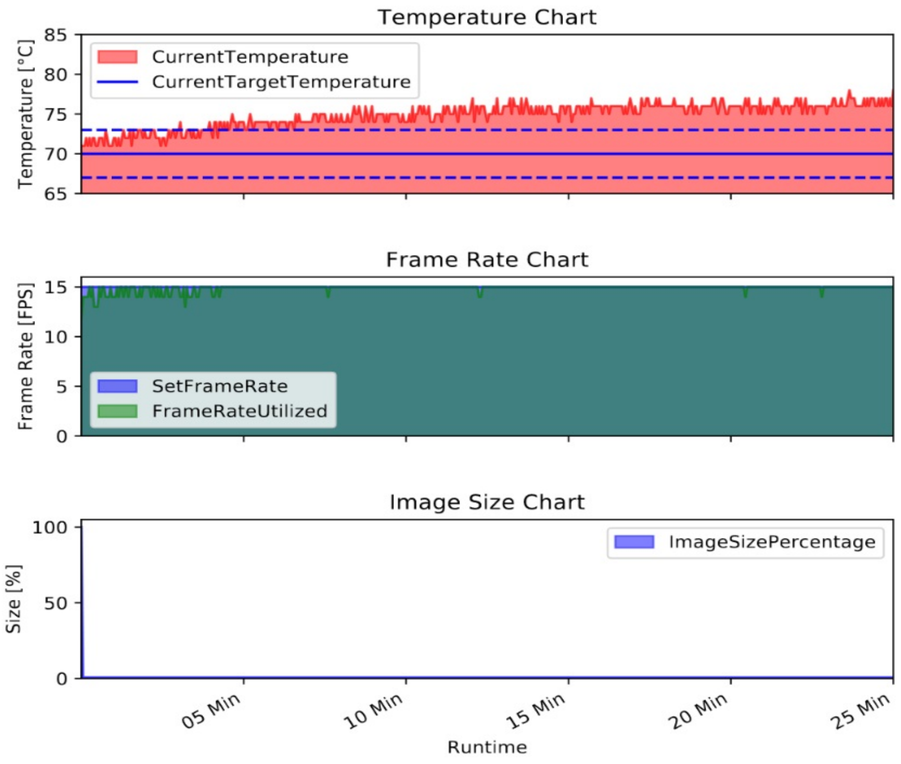


Figure 5.9: Result Chart test case GUI_TTF.

5.3.8 Memory Failure:

Figure 5.10 shows a 1 minute and 30 seconds run, containing around 30 data points. The figure 5.10 shows how the image size decreases step by step as the portion of failed memory blocks increases. The images resulting from the image size reduction can be seen in figure 5.11.

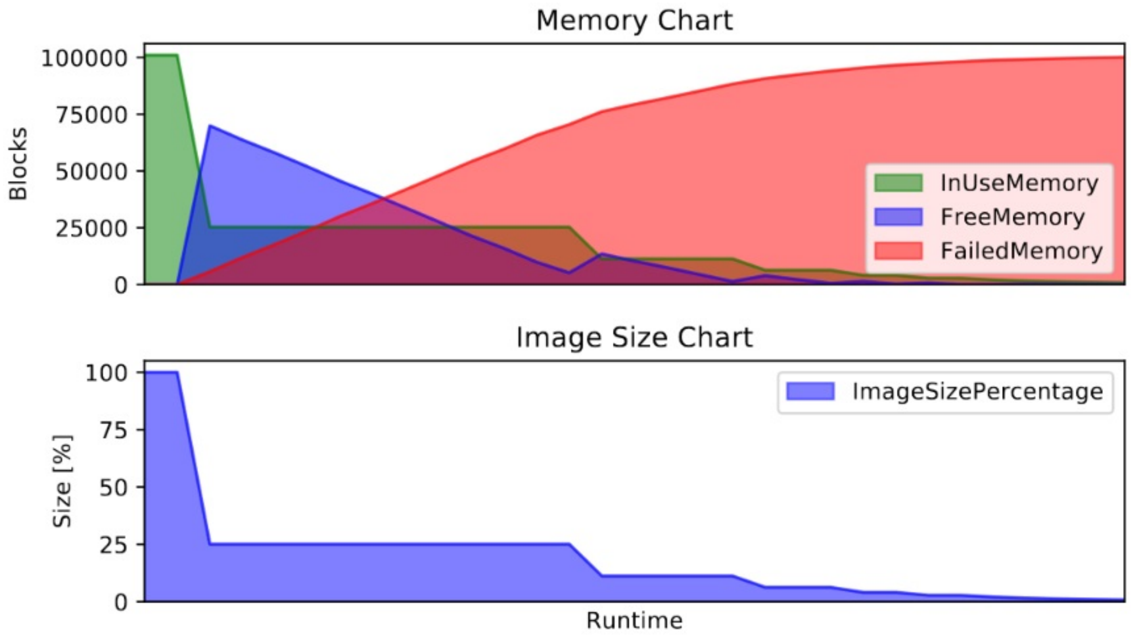


Figure 5.10: Result Chart test case memory failure.

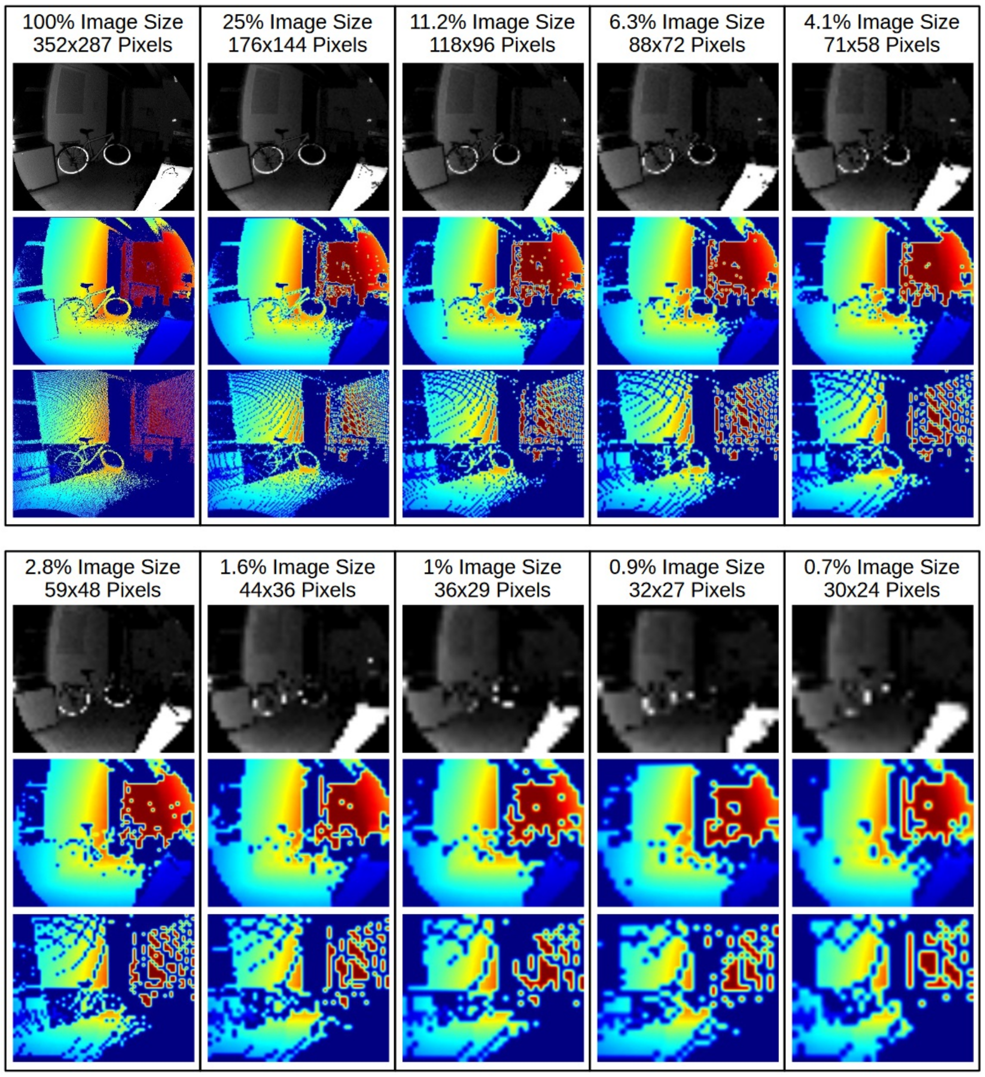


Figure 5.11: Step by Step image size reduction. From top to bottom: Gray image, depth color image and point cloud

5.3.9 Cool down CPU Temperature:

Figure 5.12 shows a 15-minute run, containing around 325 data points. As seen in figure 5.12 with an aggressively reduction of frame rate and image resolution the device manages to cool from around 80°C to below 73°C in about 3 minutes. Also seen in the figure is that after the temperature starts to drop below the target temperature the prototype increases the frame rate and image resolution again. The test run ended with a set frame rate of 7 FPS and a resolution of around two percent (51x41 pixels) of the original image size.

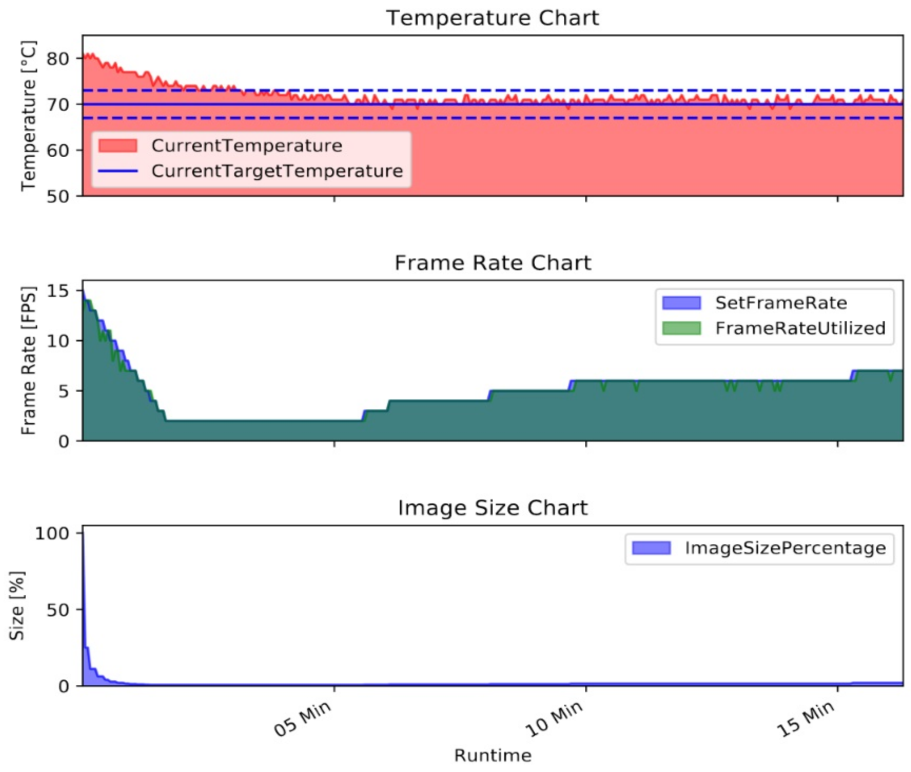


Figure 5.12: Result Chart test case cooling down.

5.3.10 Simulate Drive at Test Track:

Figure 5.14 shows a 45 minute run, containing around 850 data points. The figure 5.13 is a map snipped containing the test track.

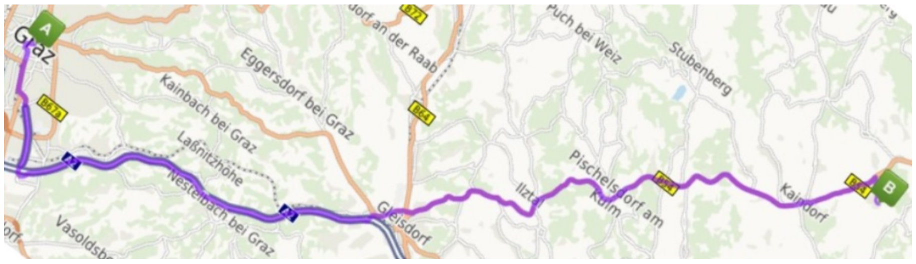


Figure 5.13: Road map of the simulated drive.

As seen in figure 5.13 and figure 5.14 the run can be roughly segmented into three parts:

- City: The simulated drive starts on a city road. In a city environment the driving speed is relatively low (around 50km/h) but traffic scenarios can be very complex. This is why it was decided to priorities image resolution over frame rate in this case, so that the complex scene could be captured in full detail. In the end this resulted in a setting with a frame rate of 5 FPS and a 100 percent resolution (352x287 pixels).

- Highway: The next section of the simulated drive on a highway. In the highway environment the driving speed is very high (up to 120km/h) but the traffic scenarios are quite simple. This is why it was decided to priorities frame rate over resolution in this case, so that it is possible to react faster. In the end this resulted in a setting with a frame rate of 15 FPS and a two percent resolution (51x41 pixels).

- Country Road and Localities: The last part of the simulated drive is on a country road that passes through different localities. On the country road the driving speed can differ vastly depending on the localities (from 50km/h up to 100km/h). This is also true for the complexity of the traffic scenarios. This is why it was decided to use two sets of configurations for the country road, one when the vehicle drives between localities and one when the vehicle is in a locality. The configuration used when driving between localities is a mixture form the city and highway configuration, with a frame rate of 10 FPS and a resolution of 25 percent (176x144 pixels). The configuration for the localities is the same as for the city part of the simulated drive.

Each of these parts has a different configuration in regards to frame rate and image size while the target temperature has not been changed between each part. The chart shows that except for the highway part the temperature was between the target temperature of 70°C and hysteresis. For the highway part the chart shows that the prototype regulates the frame rate to keep the temperature in check. To note is that this simulation was executed with the same setup as seen at the start of the chapter in figure 5.1 and that on the highway with highway speeds the airstream might have some positive effects on the temperature.

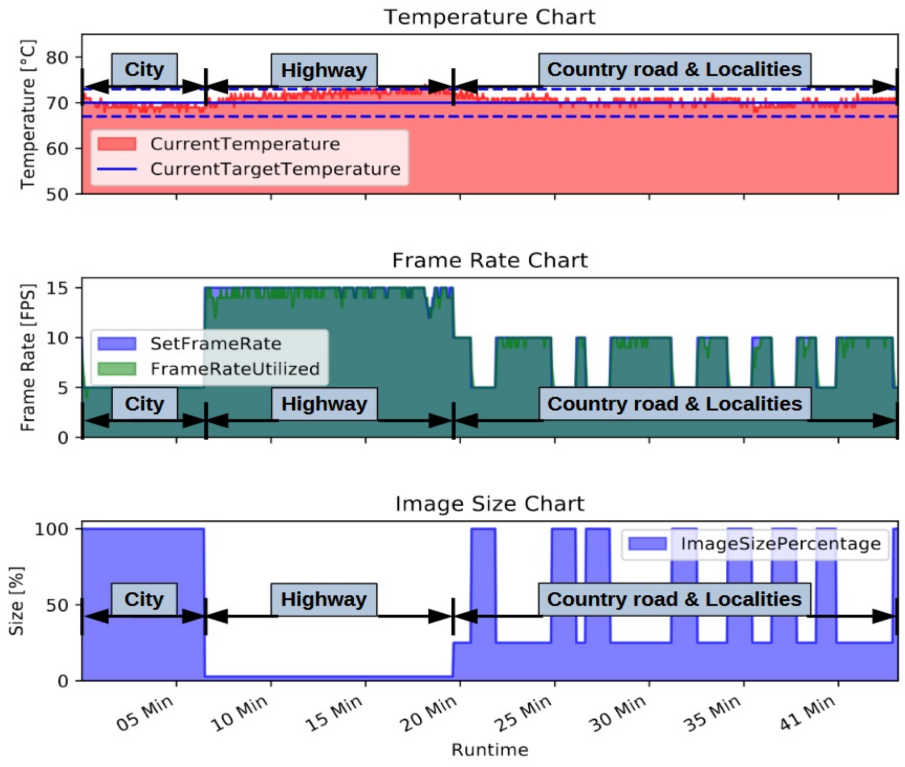


Figure 5.14: Result Chart test case simulated drive.

Chapter 6

Conclusion and Future Work

This thesis introduced a system architecture design that enables the creation of a partially fail-operational environmental perception system for a variety of applications. The design and its basic concept was introduced in Chapter 3. With Section 3.1 introducing the basic perception system which was reworked in Section 3.3 with the defined safety goals that resulted from a the simplified HARA from Section 3.2 and the goals defined in the Section 1.3. The Chapter 4 describes how the design of Chapter 3 was implemented. With the Section 4.1 describing the hardware and software used for the implementation, Section 4.2 describing the class design, Section 4.3 describing the program flow of important software parts and Section 4.4 describing the UI that was used for testing and logging. In the end the Chapter 5 shows how the implementation described in Chapter 4 was tested and what results where produced. When comparing the results in Section 5.3 with the safety goals defined in Section 3.2, it shows that a software solution can help to achieve functional safety goals but also that they have their limits. The safety goal SG1 for memory is handled through graceful degradation. It depends on the memory overhead how long it takes before the image does not fit anymore in the memory and needs to be reduced. Furthermore, a degradation scheme won't protect against complete hardware failure. The safety goal SG2 for overheating is handled mainly by decreasing the load on the CPU by reducing the image resolution or frame rate. This only works if an image resolution or frame rate reduction can be safely executed in the current driving scenario. The safety goal SG3 for avoiding overload is handled the same way as SG2 and therefore suffers from the same limitation. As described in Section 3.3, the safety goal SG4 cannot be controlled by the perceptions system itself.

One of the biggest challenge when designing a functional safe system is to find the balance between safety, efficiency and cost. Solutions that require a higher degree of functional safety normally rely heavily on hardware redundancy, which is efficient but usually also introduces a lot of extra costs. The environmental perception system introduced by this thesis showed that a software solution can help find a balance between safety and cost.

6.1 Future Work

Future work might consist of investigating on how to encode the internal data transmission inside the device such that the integrity of configuration data and variables are ensured. Or an extension to the memory manager that distributes the data in such a way that no memory cell is utilized proportionally more to other memory cells. This extension might also contain a monitor to track the wear of the memory. Another future work might be the design of a fail-operational system that makes use of multiple instances of the system described in this thesis. This design might look similar as seen in figure 3.3 in Chapter 3.

However, no matter what future works might include one thing is clear, as cars evolve towards self-driving cars and cars-as-a-service, so must the safety designs and concepts used in cars evolve from fail-safe to fail-operational.

Bibliography

[Apt19] Aptiv, Audi, Baidu, BMW, Continental, Fiat Chrysler Automobiles, HERE, Infineon, Intel and Volkswagen. *Safety First for Automated Driving*, 2019.

[Ass08] JEDEC Solid State Technology Association. *JEDEC standard: DDR3 SDRAM, JESD79-3C.*, 2008.

[AVG15] Anas Alhashimi, Damiano Varagnolo, and Thomas Gustafsson. Joint Temperature-Lasing Mode Compensation for Time-of-Flight LiDAR Sensors. *Sensors*, 15:31205–31223, 12 2015.

[Cor17] Renesas Electronics Corporation. *Semiconductor Reliability Handbook*. Renesas Electronics Corporation, 2017.

[CWB10] P. C. Chia, S. Wen, and S. H. Baeg. New DRAM HCI qualification method emphasizing on repeated memory access. In *2010 IEEE International Integrated Reliability Workshop Final Report*, pages 142–144, Oct 2010.

[Dub13] Elena Dubrova. *Fault-Tolerant Design*. Springer Science & Business Media, 2013.

[fAEOfCAE00] Radio Technical Commission for Aeronautics & European Organisation for Civil Aviation Equipment. *RTCA DO-254 / EUROCAE ED-80: Design Assurance Guidance for Airborne Electronic Hardware*, 2000.

[fAEOfCAE12] Radio Technical Commission for Aeronautics & European Organisation for Civil Aviation Equipment. *RTCA DO-254 / EUROCAE ED-12C: Software Considerations in Airborne Systems and Equipment Certification*, 2012.

[FDN$^+$01] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and Hon-Sum Philip Wong. Device scaling limits of Si MOSFETs and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288, March 2001.

[Fou19] Raspberry Pi Foundation. *Raspberry Pi 4B*, 2019. 4GB version.

[GKO18] K. Gorelik, A. Kilic, and R. Obermaisser. Range prediction and extension for automated electric vehicles with fail-operational powertrain: Optimal and safety based torque distribution for multiple traction motors. In *2018 Annual IEEE International Systems Conference (SysCon)*, pages 1–7, April 2018.

[HK91] M. Hebert and E. Krotkov. 3-D measurements from imaging laser radars: how good are they? In *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, pages 359–364 vol.1, 1991.

[HSS12] Andy A. Hwang, Ioan A. Stefanovici, and Bianca Schroeder. Cosmic Rays Don'T Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 111–122, New York, NY, USA, 2012. ACM.

[IC19] Itseez Intel Corporation, Willow Garage. *OpenCV*, 2019. C++ package version 4.1.0.

[ICE10] ICE. *Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2010.

[Int96] SAE International. *SAE ARP4761: GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT PROCESS ON CIVIL AIRBORNE SYSTEMS AND EQUIPMENT.*, 1996.

[Int10] SAE International. *SAE ARP4754A: Guidelines for Development of Civil Aircraft and Systems.*, 2010.

[Int16] SAE International. *SAE J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle.*, 2016.

[Int18] SAE International. *SAE J3016: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.*, 2018.

[ISO11] ISO. *ISO26262: Road vehicles - Functional Safety.*, 2011.

[ISO18a] ISO. *ISO26262: Road vehicles - Functional Safety.*, 2018.

[ISO18b] SAE ISO. *ISO/SAE CD 21434: Road Vehicles Cybersecurity engineering.*, 2018.

[ISO19] ISO. *ISO/PAS 21448:2019 Road vehicles Safety of the intended functionality.*, 2019.

[ITR12] ITRS. *Process Integration, Devices, and Structures.*, 2012.

[KDK+14] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, June 2014.

[Koh19] Christopher M. Kohlhoff. *Asio C++ Library*, 2019. Version 1.14.1.

[LE19] M. Li and L. Eckstein. Fail-Operational Steer-By-Wire System for Autonomous Vehicles. In *2019 IEEE International Conference of Vehicular Electronics and Safety (ICVES)*, pages 1–6, Sep. 2019.

[lib19] Boost (C++ libraries). *Boost*, 2019. Version 1.71.0.

[LJK+13] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. 06 2013.

[Loh19] Niels Lohmann. *Json.hpp*, 2019. Version 3.6.1.

[LS14] V. Lakshminarayanan and N. Sriraam. The effect of temperature on the reliability of electronic components. pages 1–6, 01 2014.

[MKK+01] Jason Mouzakes, Peter J. Koltai, Siobhan Kuhar, Dan S. Bernstein, Paul Wing, and Edward Salsberg. The Impact of Airbags and Seat Belts on the Incidence and Severity of Maxillofacial Injuries in Automobile Accidents in New York State. *Archives of Otolaryngology-Head & Neck Surgery*, 127(10):1189–1193, 10 2001.

[MS14] Onur Mutlu and Lavanya Subramanian. Research Problems and Opportunities in Memory Systems. *Supercomput. Front. Innov.: Int. J.*, 1(3):19–55, October 2014.

[MWKM15] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 415–426, June 2015.

[Nen07] Philipp Nenninger. *Vernetzung verteilter sicherheitsrelevanter Systeme im Kraftfahrzeug.* PhD thesis, KIT, 2007.

[PGM17] Tyson Phillips, Nicky Guenther, and Peter Mcaree. When the Dust Settles: The Four Behaviors of LiDAR in the Presence of Fine Airborne Particulates. *Journal of Field Robotics*, 34, 02 2017.

[PKM17] M. Patel, J. S. Kim, and O. Mutlu. The reach profiler (REAPER): Enabling the mitigation of DRAM retention failures via profiling at aggressive conditions. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 255–268, June 2017.

[pmd18] pmd. *CamBoard pico monstar*, 2018.

[pmd19] pmd. *pmd Royale SDK*, 2019. C++ package version 3.23.0.86.

[Ros19] H.L. Ross. *Funktionale Sicherheit im Automobil: Die Herausforderung für Elektromobilität und automatisiertes Fahren.* Carl Hanser Verlag GmbH & Company KG, 2019.

[RSS11] R. Rasshofer, M. Spies, and H. Spies. Influences of weather phenomena on automotive laser radar systems. *Advances in Radio Science*, 9, 07 2011.

[SBM$^+$09] Alex Shye, Joseph Blomstedt, Tipp Moseley, Vijay Janapa Reddi, and Daniel Connors. PLR: A Software Approach to Transient Fault Tolerance for Multicore Architectures. *IEEE Trans. Dependable Sec. Comput.*, 6:135–148, 04 2009.

[Sch16] Adam Schnellbach. *Fail-operational automotive systems.* PhD thesis, TUG, 2016.

[Sho02] Martin L. Shooman. *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[SPW09] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *SIGMETRICS*, 2009.

[SSD$^+$13] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. Feng Shui of supercomputer memory positional effects in DRAM and SRAM faults. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Nov 2013.

[SSR$^+$17] T. Siddiqua, V. Sridharan, S. E. Raasch, N. DeBardeleben, K. B. Ferreira, S. Levy, E. Baseman, and Q. Guan. Lifetime memory reliability data from the field. In *2017 IEEE International Symposium* *on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, Oct 2017.

[TIS19] Inc Terra Informatica Software. *Sciter*, 2019. Sciter DLL version 4.2.8.3.

[TLS05] Pascal Traverse, Isabelle Lacaze, and Jean Souyris. A Process Toward Total Dependability Airbus Fly-by-Wire Paradigm. volume 7, page 1, 04 2005.

[WYZ11] W. Wang, Yin Jun, and Zhang Mei-jie. The research of FPGA reliability based on redundancy methods. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 3, pages 1608–1611, Dec 2011.

[WZX17] G. Wang, L. Zhang, and W. Xu. What Can We Learn from Four Years of Data Center Hardware Failures? In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 25–36, June 2017.

[XSS+16] You Xiang, Lei Sun, Li Sui, Jun Kang, and Yong Jiang. Interactive Safety Analysis Framework of Autonomous Intelligent Vehicles. *MATEC Web of Conferences*, 44:01029, 01 2016.

[Yeh96] Y. C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, Feb 1996.

[YWP15] Nursidik Yulianto, Bambang Widiyatmoko, and Purnomo Priambodo. Temperature Effect towards DFB Laser Wavelength on Microwave Generation Based on Two Optical Wave Mixing. *International Journal of Optoelectronic Engineering*, 2015:21–27, 09 2015.