Safety software testing implementation based on standard IEC 61508

Adrià Freixa Tió

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 06.2023

Thesis supervisor:

Dr. Attila Kovács

Thesis advisors:

Prof. Quan Zhou

Ch. Lead (Tech.) Toni Kuikka

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Adrià Freixa Tió

Title: Safety software testing implementation based on standard IEC 61508

Date: 06.2023 Language: English Number of pages: 061

Department Electrical Engineering and Autonomation

Professorship:

Supervisor: Dr. Attila Kovács

Advisors: Prof. Quan Zhou, Ch. Lead (Tech.) Toni Kuikka

In the services industry, the autonomy of a system is strictly related to the system's safety degree. Safety determines the capabilities and requirements that a system will have. Although safety limits the autonomy of systems, it ensures they are operational, reliable and usable. The thesis consists of an analysis of what functional safety should involve, how safety and testing are related and the application of the safety standard IEC 61508 on an autonomous system. The thesis is performed alongside an internship as a test engineer in KONE; in the critical safety software department.

Keywords: Safety, test planning, standard 61508, software testing, functional testing

Contents

Chapter 1

Introduction

1.1 Background

There is a need to establish safety-related activities to protect users and systems from potentially hazardous situations. No physical element presents zero risk, hence safety-related procedures should account as the basis for the development of any product. The paradigm developed for the protection of systems is known as functional safety. Functional safety regulates the safety-related characteristics that any system shall have by setting the bases for how safety product development can be managed - safety lifecycle and requirements - and how it should be assessed - testing principles [1]. Testing procedures respond to the need to verify and validate systems with respect to the constraints that functional safety presents. In that way, testing does not only involve functionality but also safety assessment, and it is required to have a detailed plan [2]. Therefore, functional safety is presented as the key safety-enforcement framework whose main assets are the definition of safety-related requirements and the planning of testing procedures.

During the development of the thesis, I had been granted a traineeship as a trainee test engineer for KONE in the lift Critical Software Safety department. This position shall give me a clear insight into the testing principles and safety implementations present in the development of electric/electronic/programable electronic systems.

1.2 Objectives and Scope

The purpose of the thesis is to present the basis of functional safety and to expose the safety standard IEC 61508 and its implementation. Objectives:

- Analyse functional safety.
- Analyse software testing procedures.
- Investigate the relation between safety guidelines and testing.
- Analysis of the safety standard IEC 61508.
- Application of the safety Standard IEC 61508 on an electric/electronic/programable electronic system.

The established objectives should be achieved by a literature study about functional safety and safety development lifecycle. After that, an analysis of the safety Standard IEC 61508 shall be executed with a highlight on the application of the standard on software safety implementation. With the recollected information, an implementation of the software-safety needs of the standard will be sketched. The object of study will be a mobile service robot known as TurtleBot3.

The scope of the thesis extends to the software aspect of functional safety, safety lifecycles, and testing procedures. The application of the standard focused on the application of the standard IEC 61508-3 to develop a testing plan for a study case. Furthermore, the application section is focused on the needs and will be developed until the theoretical procedures of verification and validation planning for the system but the project will not present any real implementation.

1.3 Structure

The thesis is divided into three main chapters. Chapter 2 introduces the conceptual elements and importance related to testing and functional safety. It also introduces the safety standard IEC 61508. Chapter 3 provides the needed guidelines to implement the standard IEC 61508 within the software development and safety development lifecycles. Chapter 4 presents a theoretical implementation of the IEC 61508 standard, centred on software aspects. The implementation is performed over a TurtleBot3 vehicle, a mobile service robot lent from the Aalto University autonomous-driving research group, that serves as the study case. The conclusions are presented in Chapter 5.

Chapter 2

Brief study on functional safety

This chapter presents a brief analysis of functional safety, the importance of software testing and showcases the safety standard IEC 61508 structure over safety-related activities.

Functional safety is the study of hazardous failures and the mitigation of the frequency at which those failures can happen. Its sole purpose is to protect users from harm. The hazard occurrence frequency or consequences can be mitigated by designing safety-related systems. Safety-related equipment is defined as the one involved in the potential appearance of a hazard. Hence, systems such as industrial process controls, process shutdown systems and automotive controls are considered safety-related [1].

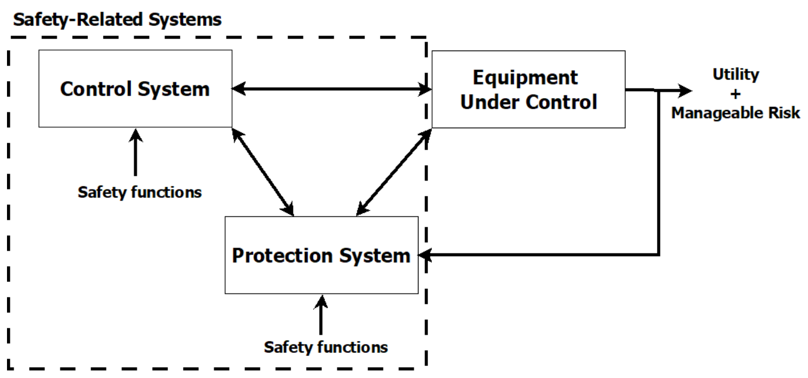


Figure 2.1: Simplistic safety functions role diagram

The goal of safety-related systems is to reduce the frequency, or probability, of a hazardous event and/or the consequences of the hazardous event [3]. In order to ensure the successful mitigation of risks, verification and validation (V&V) of the system, testing procedures have to be developed.

2.1 Software testing

Testing procedures of electric/electronic/programmable electronic-related applications account for almost the same, or more, resource-consuming activities and associated costs than the development of said applications. Furthermore, testing represents the deciding point over the correct functionality of a determined system. Hence, it is important to have a plan for the testing needs and procedures that need to be carried out.

Considering the software domain, testing probes software for faults and failures while it is being executed; referred to as a dynamic analysis of the product. Software testing can be defined as a process, or series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended [2]. Said in other words, testing confirms the existence of quality, not establishing it, while discovering faults that collide with the requirements of the system.

2.1.1 Software safety and development lifecycles

The software lifecycle is defined as the process of planning, writing, modifying, and maintaining software. Software lifecycle extents from the start of the software concievement to the decommissioning of the software. Analogously, the software safety lifecycle is the process of planning, writing, modifying, and maintaining safety-related software that implements the safety-related requirements of the system [4] [5].

The software development lifecycle has to include defences against systematic failures, as well as define the development stages of the system. Software safety-lifecycle models should be used to provide safety-related guidelines over the testing structure during the software development lifecycle [1]. To take into account functional safety within the software development, a primary software development lifecycle should be tailored to ensure the final overall development lifecycle covers the requirements of the safety-related systems [6]. An overview of the key components a safety lifecycle should present, according to the safety standard IEC 61508, is displayed in figure 2.2.

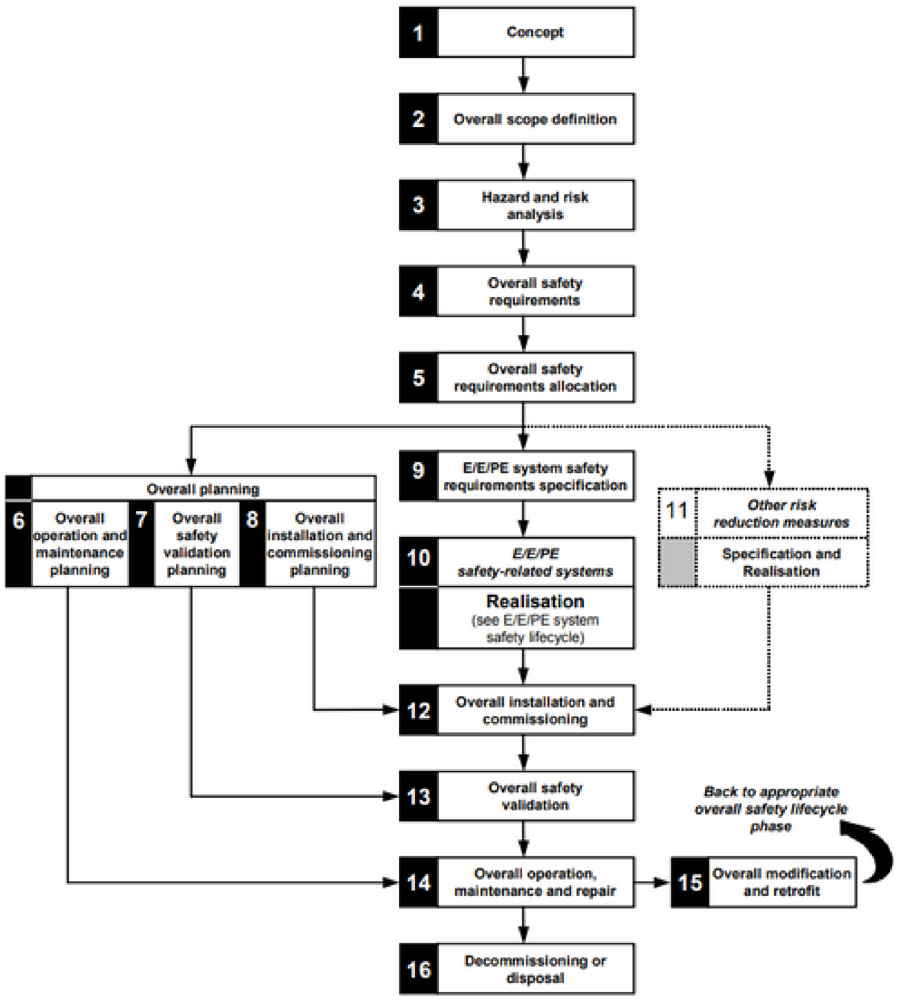

1 Concept
2 Overall scope definition
3 Hazard and risk analysis
4 Overall safety requirements
5 Overall safety requirements allocation
Overall planning
6 Overall operation and maintenance planning
7 Overall safety validation planning
8 Overall installation and commissioning planning
9 E/E/PE system safety requirements specification
10 E/E/PE safety-related systems
Realisation (see E/E/PE system safety lifecycle)
11 Other risk reduction measures
Specification and Realisation
12 Overall installation and commissioning
13 Overall safety validation
Back to appropriate overall safety lifecycle phase
14 Overall operation, maintenance and repair
15 Overall modification and retrofit
16 Decommissioning or disposal

Figure 2.2: Overall safety lifecycle [6].

Test structuring is needed in order to effectively reach the overall software safety lifecycle requirements. The allocation of different testing techniques is usually paired with specific phases of the software development lifecycle. The main software testing stages can be categorized as module testing and higher-order testing [2].

- Module testing: Process of testing the individual subprograms, subroutines, or procedures in a program. There is an understanding and visibility of the code (white-box oriented). The purpose of module tests is to find discrepancies between the program's modules and the interface specifications. Module testing presents two different integration approaches:

    - Nonincremental: Each module is independently tested and then combined with other independently tested modules.
    - Incremental: Combine a non-tested module with the rest of the tested modules to check its compliance.
- Higher-order testing: Testing procedures associated with the translation and communication of information.
    - Function testing: Finds discrepancies between the program and the external specifications; being external specifications, the behaviour from the point of view of the user. There is no direct contact with the code (black-box oriented).
    - System testing: Finds discrepancies between the product and its original objectives. It does not focus on testing the functions of the complete system (function testing), but tests if the product meets its objectives.
- Regression testing: Performed within each testing stage after making a functional imprevement or repair on the developing system. Determines whether the change has regressed other aspects of the program (error corrections are more error-prone than normal software development).
- Acceptance testing: Compares the software product to its initial requirements and the needs of the end-users. It is normally not considered the responsibility of the development organization and it is performed by the customer.
- Installation testing: It is not related to specific phases in the development lifecycle. Consists in the search for discrepancies between the installed system and the original objectives and requirements.

In the next chapter of the thesis 3 it is shown how the overall safety lifecycle can be catered alongside the V-model and showcases the required steps for the realisation phase, which describes the safety-related system necessities.

2.2 Standard IEC 61508

The risk uncertainties regarding the development of safety-related systems prompted the need for regulations since the beginning of the 1980s. The early 1990s produced the basis for functional safety standards, with IEC 1508, which would evolve into the current most used functional safety standard, the IEC 61508. The first instance of the standard was published in 1998 and has been revised multiple times. The current version of the standard refers to 2010. The standard originated from the need to regulate software engineering techniques as control mechanisms in the context of the functional safety of programmable electronic systems. The safety standard IEC 61508 is created and regulated by the International Electrotechnical Commission (IEC) and accepted in the industrial sector as the main regulation document over functional safety [7].

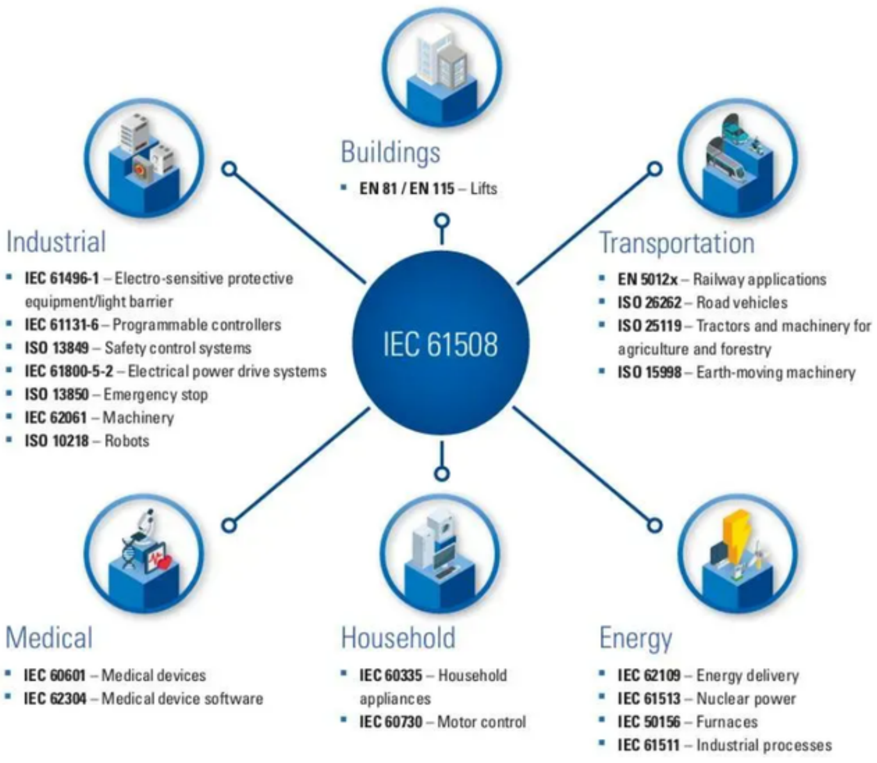

IEC 61508
Buildings
EN 81 / EN 115 – Lifts
Industrial
IEC 61496-1 – Electro-sensitive protective equipment/light barrier
IEC 61131-6 – Programmable controllers
ISO 13849 – Safety control systems
IEC 61800-5-2 – Electrical power drive systems
ISO 13850 – Emergency stop
IEC 62061 – Machinery
ISO 10218 – Robots
Transportation
EN 5012x – Railway applications
ISO 26262 – Road vehicles
ISO 25119 – Tractors and machinery for agriculture and forestry
ISO 15998 – Earth-moving machinery
Medical
IEC 60601 – Medical devices
IEC 62304 – Medical device software
Household
IEC 60335 – Household appliances
IEC 60730 – Motor control
Energy
IEC 62109 – Energy delivery
IEC 61513 – Nuclear power
IEC 50156 – Furnaces
IEC 61511 – Industrial processes

Figure 2.3: Industry connections with the IEC 61508 standard [8].

The standard is constructed based on the identification, assessment and dealing of the possible risks a system presents. It is intended to satisfy the needs of all the industry sectors and can be used to perform industry-specific or application-specific functional safety evaluation and planning. Despite being formally focused on hardware and software aspects of Electrical/Electronic/Programmable Electronic (E/E/PE) systems, its principles form a framework for managing all aspects of safety [7]. Different norms and standards have been developed from IEC 61508 depending on the industrial application. Figure 2.3 shows how different industries have adapted the standard for their purposes.

In this project, only the IEC 61508 is taken into account. The standard is structured in seven parts (plus one overview document).

- Part 0: Functional safety and IEC 61508. - Overview of the standard
- Part 1: General Requirements - Defines safety lifecycle steps and safety assessment
- Part 2: Requirements for Electrical/Electronic/Programmable Electronic (E/E/PE) Safety-Related Systems - Interpretation of the part 1 requirements in the context of hardware.
- Part 3: Software Requirements - Interpretation of the part 1 requirements in the context of software.
- Part 4: Definitions and Abbreviations
- Part 5: Examples of Methods for the Determination of Safety Integrity Levels (SIL) - Risk analysis examples and demonstration of SIL allocation.
- Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3 - Guidance over the application of the standard for hardware and software necessities respectively.
- Part 7: Overview of Techniques and Measures - Description of techniques used in safety and software engineering.

2.3 Risk assessment and safety integrity level (SIL)

"There is no such thing as zero risk. This is because no physical item has zero failure rate, no human being makes zero errors and no piece of software design can foresee every operational possibility." [1]

The first step in functional safety development is the risk assessment of the studied system. Risk can be defined as the probability and consequence of a specified hazardous event occurring[3]. As presented earlier, the IEC 61805 is a risk-based standard, hence, it is important to evaluate the system risks when performing a functional safety assessment. The study about functional safety, based on risks, can be summarized by: establishing the tolerable risk criteria, assessing the risks associated with the equipment under control and determining the risk reduction needed to meet the risk acceptance criteria [9].

2.3.1 Risks

Risk tolerabilities depend on environmental, cultural and socio-economic circumstances. The selection criteria can be qualitative or quantitative. Qualitative criteria use time-frequency adverbs to describe the likelihood of occurrence, such as frequent, occasional, etc; and situational status to describe the consequences, such as critical, catastrophic, etc. On the other hand, quantitative criteria use numbers, such as frequency occurrence boundaries and impact percentages to describe the likelihood and consequences. In both criteria, there has to be an agreement over the boundaries represented by each term of occurrence and status and should estimate the same kind of results when implemented. It is worth noticing that both criteria need one another to ensure a suitable implementation as qualitative methods need some numbers to define boundaries and quantitative ones need some words to ensure a consistent interpretation[9].

The safety-related system's main purpose is to reduce risks in order to meet a tolerable frequency of hazardous events. Furthermore, a safety-related system can be defined as a system that:

- implements the required safety functions to achieve achieve, or maintain, the equipment under control, by usage of safe states;
- ensures the necessary safety integrity for the required safety functions.

Risk assessment regarding the tolerable risk and the necessary risk reduction is used to determine the safety integrity requirements of the E/E/PE safety-related system[3].

2.3.2 Safety integrity

Safety integrity applies solely to the safety-related systems and other risk reduction measures and determines the likelihood that those systems achieve the necessary risk reduction under all the stated conditions within a stated period of time. It is important to discern the concept of

safety integrity with risk [7].
Safety integrity can be divided into two components, hardware safety integrity and systematic safety integrity. Hardware safety integrity refers to random hardware failures while systematic safety integrity refers systematic failures. The distribution of systematic failures presents a higher uncertainty than hardware failures, which makes systematic failures harder to predict.

Safety integrity levels are designed to distribute different levels of safety integrity satisfaction in order to organize the development of safety integrity requirements specifications. Furthermore, the requirement specifications shall specify the safety integrity levels for the E/E/PE safety systems[3]. The standard IEC 61508 presents four levels of safety integrity defined by the occurrence of dangerous failure on demand of the safety function. Depending on the demand of failure occurrence that the safety system has to manage, there are three different modes of operation that SIL should contemplate:

- low demand mode: frequency of demands over operation on the safety function is no greater than one per year.
- high demand mode: frequency of demands over operation on the safety function is greater than one per year.
- continous mode: demand for operation of the safety function is continuous.

The aforementioned modes shall take into account the existence of a EUC and a control system, associated human factors issues, and safety protection features that comprise a safety-related system operating at the specified demand mode (low, high, continuous) and other risk reduction measures. Furthermore, the safety-related demands that the different modes handle can come from [3]:

- General demands from the EUC.
- Demands arising from failures in the EUC control system.
- Demands arising from human failures.

Table 2.1 illustrate the SIL division depending on the mode of operation.

Safety integrity level (SIL)	**Average probability of a dangerous failure on demand of the safety function (PFD_{avg})**	**Average frequency of a dangerous failure on demand of the safety function [h^{-1}] (PFH)**
4	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-4}$ to $< 10^{-3}$	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-3}$ to $< 10^{-2}$	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-2}$ to $< 10^{-1}$	$\geq 10^{-6}$ to $< 10^{-5}$

Table 2.1: Safety integrity levels - target failure measures for a safety function operating in low (PFD_{avg}) and high/continuous (PFH) demand mode of operation

A SIL 4 represents the highest safety integrity level and SIL 1 the lowest. SIL 1 and 2 are relatively easy to achieve through a well-defined design; the latter will require more review and testing hence, more cost. SIL 3 involves a significantly more substantial increase in effort and competence. SIL 4 considers state-of-the-art practices such as "formal methods" in design. The competence required is scarce and costs will be much higher than the other levels [1].

Different quantitative and qualitative methodologies have been developed in order to establish the SILs for safety-related systems. Multiple assessments of risk are taken into account and are contrasted with each other when selecting appropriate SILs. Numerical Assessment, severity matrices, LOPA and Risk Graphs are some examples of these methodologies, which are based on determining a failure rate [7]. Furthermore, the techniques should not only ensure that a specific quantitative failure rate has been met but aim for a failure rate "as low as reasonably practicable". This paradigm is known as ALARP . It is worth noting that quantitative and qualitative conformances with SIL have to be equally met and that ALARP can be applied independently on the applied criteria [1].

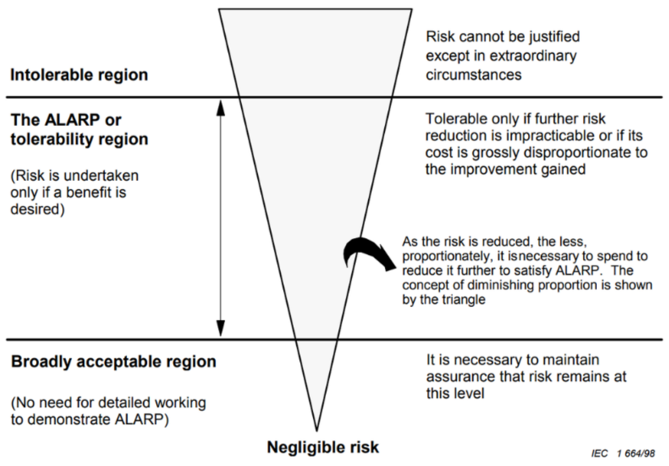

Intolerable region
Risk cannot be justified except in extraordinary circumstances
The ALARP or tolerability region
(Risk is undertaken only if a benefit is desired)
Tolerable only if further risk reduction is impracticable or if its cost is grossly disproportionate to the improvement gained
As the risk is reduced, the less, proportionately, it isnecessary to spend to reduce it further to satisfy ALARP. The concept of diminishing proportion is shown by the triangle
Broadly acceptable region
(No need for detailed working to demonstrate ALARP)
It is necessary to maintain assurance that risk remains at this level
Negligible risk
IEC 1 664/98

Figure 2.4: Tolerable risk and ALARP

2.4 Safety Requirements

Safety requirements are defined to specify the procedures and basis needed to apply to a system in order to mitigate risks. The definitions shall incorporate the need for the reduction of risks as well as the detailed processes needed to make it effective. The safety requirements shall be implemented via the use of safety functions. Note that safety requirements may need multiple safety functions to be properly fulfilled [7].

The bases to create safety integrity requirements for the safety functions should be the software's systematic capabilities so that the safety-related software will be able to handle the needs of the safety functions. The allocation of safety requirements to E/E/PE safety-related systems should depend on the necessary risk reduction, the measures to achieve the risk reduction and the correspondent SIL level of the requirement. Figure 2.5 exemplifies the safety requirement allocation diagram [10].

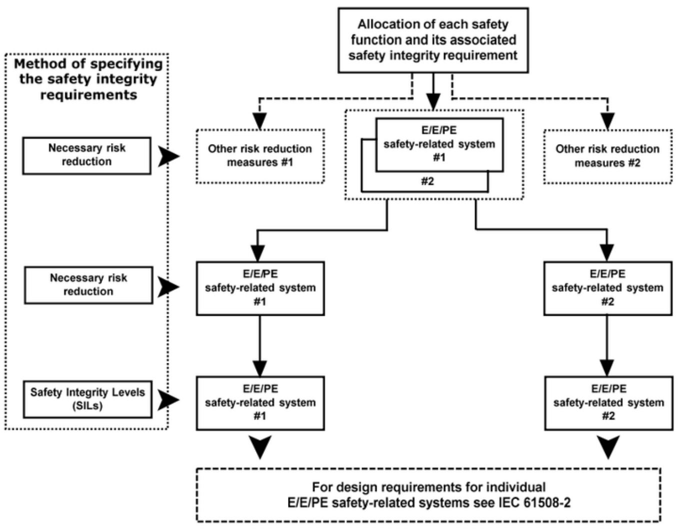

Allocation of each safety function and its associated safety integrity requirement
Method of specifying the safety integrity requirements
Necessary risk reduction
Necessary risk reduction
Safety Integrity Levels (SILs)
Other risk reduction measures #1
E/E/PE safety-related system #1
#2
Other risk reduction measures #2
E/E/PE safety-related system #1
E/E/PE safety-related system #2
E/E/PE safety-related system #1
E/E/PE safety-related system #2
For design requirements for individual E/E/PE safety-related systems see IEC 61508-2

Figure 2.5: Allocation of safety requirements to the E/E/PE safety-related systems and other risk reduction measures

The description of safety requirements should follow the key points described in the third part of the IEC 61508 standard [6]. Requirement specifications need to consider:

- Safety and non-safety functions. Detection about when software is (not) performing safety functions.
- Configuration/architecture of the system.
- Hardware safety integrity requirements.
- Software systematic capability requirements.
- Capacity and response times.
- Equipment and operator interfaces, including foreseeable misuse.

Accounting for the capabilities of the system architecture design, software safety requirements specifications shall also consider:

- Software self-monitoring.
- Monitoring of the programable electronics hardware, sensors, and actuators.
- Periodic testing of safety functions while the system is running.
- Enabling safety functions to be testable when the EUC is operational.
- Software functions to execute proof tests and diagnostic tests to check the SIL requirement of the safety-related system.

Chapter 3

Safety standard IEC 61508 application guidelines

This chapter focuses on the software safety-related measures of the implementation of the standard IEC 61508. It expands on software development and the software safety lifecycle.

3.1 Software lifecycle and testing

The traditional waterfall model can be adjusted becoming the widely used software development V-model, which in turn is tailored to match the overall safety lifecycle necessities. The V-model will be used to showcase the integration of the software development lifecycle with testing.

As figure 3.1 shows, the phases of software development are divided into two branches. One side of the model (left-hand side) shows a typical software development lifecycle proper of a waterfall model; it is known as the verification branch. The other side (right-hand side) shows the testing-related activities corresponding to the appropriate software development stage; it is known as the validation branch. At each stage of the verification branch, the testing plan for the appropriate testing module should also be developed. The course of action starts at the requirement definition step and ends in the validation testing step [11]. A more in-depth analysis about the requirements of the verification and validation software activities regarding the safety lifecycle can be found in subsections 3.2.6 and 3.2.7.

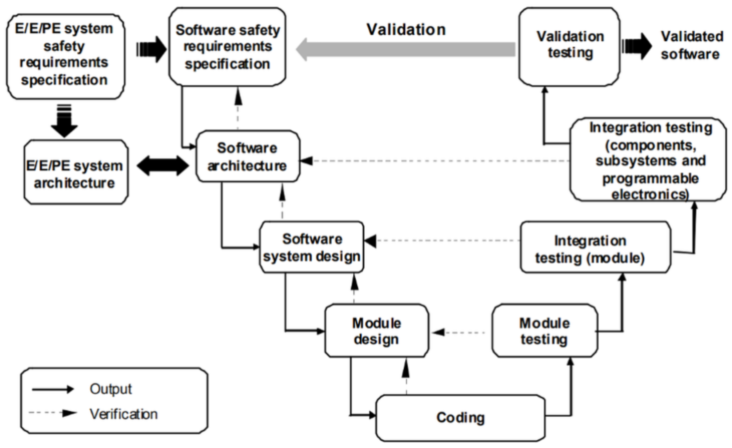

E/E/PE system safety requirements specification
Software safety requirements specification
Validation
Validation testing
Validated software
E/E/PE system architecture
Software architecture
Integration testing (components, subsystems and programmable electronics)
Software system design
Integration testing (module)
Module design
Module testing
Coding
Output
Verification

Figure 3.1: Software development lifecycle V-model [6].

The main testing stages of the presented V-model, following the testing definitions established in section 2.1, are the following:

- Module (Unit) testing: Finds bugs in the internal processing logic and data structures in individual modules. The creator of the test has an understanding and visibility of the code [12].
- Integration testing (module testing): Validate whether the components of the application are working and communicating correctly. Checks interfacing faults that may occur in the module integrating phase. The integration strategy, normally incremental (bottom-up, top-down), has to be taken into account [12].
- Integration testing (Function and system testing): New delivered applications are integrated with the whole system, so the application is tested on the integrated environment [12].
- Validation testing (Acceptance testing): Normally executed by the costumer. Validation if the delivery of the system meets the requirements of the customer [12].

The test completion criteria determine when testing activities shall end. The most common criteria are [2]:

- Stop when the scheduled time for testing expires
- Stop when all the test cases execute without detecting errors.

A more in-depth analysis about the requirements of the testing necessities regarding module and integration testing can be found in subsections 3.2.3 and 3.2.4.

3.2 Software safety lifecycle (realisation phase)

This section expands on the framework applied to define software safety lifecycles according to the standard IEC 61508. Furthermore, the block of interest will be the "realisation" phase of the overall safety lifecycle 2.2. The study case software safety-related plan requirements will be decided from the definitions given by the aforementioned phase in the context of software. Note that all the material that can be found in this section belongs to the IEC 61508-3 [6].

The realisation phase of the software safety lifecycle sets the bases and definitions for the application of the software requirements (IEC 61805-3). The structure of the software safety lifecycle realisation phase can be seen in figure 3.2.

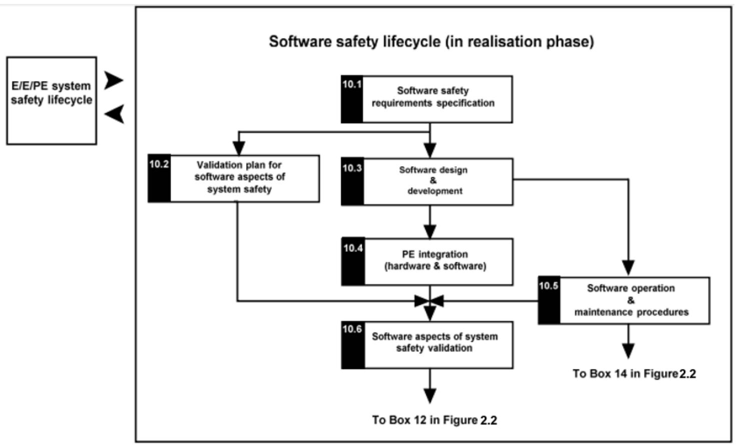

Software safety lifecycle (in realisation phase)
E/E/PE system safety lifecycle
10.1
Software safety requirements specification
10.2
Validation plan for software aspects of system safety
10.3
Software design & development
10.4
PE integration (hardware & software)
10.5
Software operation & maintenance procedures
10.6
Software aspects of system safety validation
To Box 14 in Figure2.2
To Box 12 in Figure 2.2

Figure 3.2: Software safety lifecycle (realisation phase) [6].

3.2.1 Software safety requirements specification

Box 10.1 in figure 3.2.

Objectives: Establish the requirements for safety-related software considering the software safety functions and capability requirements. Definition of software safety functions and software systematic capabilities for each E/E/PE safety-related system.

Requirements: The safety requirements will be created for each E/E/PE safety-related system in order to ensure that they comply with the E/E/PE capability and safety integrity level specified for each safety function. These requirements need to follow the description and allocation presented in section 2.4.

Software safety functions needed to be taken into consideration:

- Functions that enable the EUC to achieve or maintain a safe state.
- Functions related to the detection, annunciation and management of faults in the programmable electronics hardware.
- Functions related to the detection, annunciation and management of sensor and actuators faults.
- Functions related to the detection, annunciation and management of faults in the software itself (software self-monitoring).
- Functions related to the periodic testing of safety functions online (self-checks).
- Functions related to the periodic testing of safety functions offline.
- Functions to allow the PE system to be safely modified.
- Interfaces to non-safety-related functions.
- Capacity and response time performance.
- Interfaces between the software and the PE system.
- Safety-related communications.

Software systematic capability requirements need to define:

- Safety integrity level(s) for each of the safety functions above.
- Independence requirements between functions

3.2.2 Validation plan for software aspects of system safety

Box 10.2 in figure 3.2.
Objective: Develop a plan for validating the safety-related software aspects of system safety.

Requirements: Planning shall specify the steps used to enforce the safety requirements. The validation plan shall include:

- Details of when the validation shall take place.
- Details of who should carry out the validation.
- Identification of the relevant modes of the EUC operation including:
  - Preparation for use, setting and adjustment.
  - Operation modes.
  - Shut-down/resetting/maintenance modes.
  - Foreseeable abnormal conditions and reasonably foreseeable operator misuse.
- Identification of the safety-related software which needs to be validated for each mode.
- Technical strategy for validation.
- Measures (techniques) and procedures that shall be used.
- Required environment in which validation is to take place.
- Pass/fail criteria:
  - Required input signals with their sequences and their values.
  - Anticipated output signals with their sequences and their values.
  - Other acceptance criteria (memory usage, timing, value tolerances).
- Policies and procedures for evaluating results, particularly failures.

3.2.3 Software design and development

Box 10.3 in figure 3.2.
Objectives: Create a software architecture that fulfils the safety-related

software requirements with the required SIL. Evaluate requirements on the software by the hardware architecture. Select an adequate set of tools for the required SIL, over the safety lifecycle. Design and implement software that fulfils requirements for safety-related software with respect to SILs. Verify that requirements for safety-related software are achieved.

<ins>General requirements:</ins> The design method chosen shall possess features that facilitate:

- Abstraction, modularity and other features which control complexity
- The expression of:
  - Functionality
  - Information flow between elements
  - Sequencing and time related information
  - Timing constants
  - Concurrency and synchronized access to shared resources
  - Data structures and their properties
  - Design assumptions and their dependencies
  - Exception handling
  - Design assumptions
  - Comments
- Ability to represent several views of the design inluding structural and behavioural views.
- Comprehension by developers
- Verification and Validation.

The design method has to allow the execution of safety-related practices such as testability, software modification and integration capabilities. It shall also allow to verify safety integrity levels and to attain at the regulations established by them.

<ins>Software architecture design:</ins> Defines the major elements and subsystems of the software, how they are interconnected, and how safety integrity is achieved.

- Select and justify an integrated set of techniques and measures necessary to fulfil the safety requirements and SILs. These techniques and measures shall include software design strategies for fault tolerance and fault avoidance.
- Partition subsystems and elements according to:
  - Whether the elements/subsystems have been previously verified; and their verification conditions
  - Whether each element/subsystem is safety-related or not
  - Regarding the software systematic capability of the subsystem/element
- Determine all software/hardware interactions and their significance.
- Note if the architecture unambiguously defined or restricted to defined features.
- Select the design features to be used for maintaining the safety integrity of all data.
- Specify appropriate software architecture integration tests to ensure the safety requirement is satisfied at the required SIL.

<u>Support tools requirements, including programming languages</u>: Support online tools (can directly influence safety-related systems during run time) and offline tools (cannot directly influence safety-related systems during run time), including programming languages, have to be taken into account. Online support tools shall be taken into account as a software element of the software-related system. Offline support tools shall be assessed and validated.

<u>Software system design requirements:</u> The software design (partitioning of major elements of the architecture into a system of software modules) shall be planned before the specification requirements of the safety-related system, the software architecture design and the validation plan for software aspects of system safety. The software shall achieve modularity, testability and the capability for safe modification. Software module testing specifications shall originate from the specification of the software system design. Integration testing shall be specified to comply with the required safety integrity level.

Requirements for code implementation: Code implementation shall take into account its readability, clarity, testability and satisfy the specified requirements of software module design, coding standards and safety planning. Each software code module shall be reviewed accordingly.

Requirements for software module testing: Software modules shall be verified (module testing) as presented in the software module test specification developed during software system design. Module testing shall show whether or not each software module performs its intended function and does not perform unintended functions. The status of the test (pass/fail) shall be specified and documented.

Requirements for software integration testing: Shall be specified during software system design phase. Software system integration tests shall state:

- Division of the software into manageable integration sets.
- Test cases and test data
- Types of tests to be performed
- Test environment, tools, configuration and programs
- Test criteria on which the completion of the test will be judged.
- Procedures for corrective action over failure of test.

Integration tests shall show that all software modules and software elements interact correctly to perform their intended function and do not perform unintended functions.

3.2.4 Programmable electronics integration

Box 10.4 in figure 3.2.
Objectives: Integrate the software onto the target programmable electronic hardware. Combine hardware and software in safety-related programmable electronics.

Requirements: The hardware and software compatibility in the safety-related PE shall be verified by integration tests. These integration tests shall be specified during the design and development phase. The software/PE integration test specification shall state:

- The split of the system into integration levels.
- Test cases and test data.
- Types of tests to be performed.
- Test environment including tools. support software and configuration description
- test criteria over the completion of the tests

In addition, the software/PE integration test specification shall distinguish between:

- The activities performed in the developer's premises and the ones in the user's site.
- Merging of the software system on to the target PE hardware.
- Adding interfaces such as sensors and actuators; E/E/PE integration
- Applying the E/E/PE safety-related system to the EUC

Any change in the integrated system during the integration testing of safety-related software/electronics shall be subject to impact analysis. The impact analysis shall determine all software modules impacted and forward actions.

3.2.5 Software modification

Box 10.5 in figure 3.2.
Objectives: Define the maintenance needs and actions for the functional safety of the E/E/PE safety-related systems during operation and modification. Guide corrections, enhancements or adaptations to a validated software, the software's systematic capability has to be sustained.

Requirements: Software modifications procedures shall be make available. Modifications shall only be initiated if there is a software request that is accepted, hence was accounted for, by the safety planning procedures. The software modification request shall include the hazards which may be affected, the proposed modification and the reasons for the modification. Impact analysis is required for the proposed software modification and it shall be documented. Impact analyses shall determine the existence of a hazard, perform a risk analysis and determine the software safety lifecycle phases that are affected. The safety planning of the

modifications shall:

- Detail the required actions in the specified software lifecycle phases.
- Identify the staff and specification of the required competence.
- Detail specification for the modification.
- Perform verification planning.
- Define the scope of revalidation and testing to the extent required by the SIL.

Modifications shall be executed as stated in the plan. Modification details shall be documented. Documentation shall include:

- Modification request
- Results of the impact analysis and the justified taken decisions.
- Software configuration management history.
- Deviation from normal operations and conditions.
- Information affected by the modification activity.
- Re-verification and re-validation of data and results

3.2.6 Software system safety validation

Box 10.6 in figure 3.2.
Objective: Ensure that the integrated system follows the software safety requirement specification imposed by the SIL.

Requirements: The validation activities shall be specified in the validation plan. If the fact that the integrated system complies with the safety requirement specification has already been verified in the safety validation planning, then the validation does not need to be repeated. Software safety validation shall document:

- The division of responsibility for conformance (if needed).

- The results of validating the software aspects.
- For each safety function:
  - Chronological record of the validation activities.
  - The version of the validation plan
  - The safety function being validated.
  - Tools, equipment and calibration data
  - The results of the validation activity
  - Discrepancies between expected and actual results.
- Analysis of discrepancies and the decision to continue the validation or return to an earlier part of the development lifecycle, if discrepancies are found.

The validation of safety-related software aspects of system safety shall take into account:

- Testing shall be the main validation method for software. In order to supplement the validation activities, analysis, animation and modelling may be used.
- The software shall be evaluated by means of simulating:
  - input signals that would be present in normal operation
  - anticipated occurrences
  - undesired conditions that require the system to act
- The documented results of the validation and the documentation that enables the developer product to meet the requirements of IEC 61508-1 and IEC 61508-2.
- The results of the validation shall meet the following requirements:
  - The tests shall show compliance with the specified requirements for the safety-related software. Hence, the software performs the intended functions and does not perform unintended functions.
  - Test cases and test results shall be documented for subsequent analysis and assessment, as required by SIL.
  - The documented results of the validation process shall state either:
    - ∗ The software has passed the validation
    - ∗ The reasons for not passing validation

3.2.7 Software verification

Objective: Test and evaluate the outcomes from a safety lifecycle phase to ensure consistency and compliance with the inputs of the lifecycle phase. The consistency and correctness of the relation between inputs and outputs are given by desired safety integrity level.

Requirements: The software verification and development shall be planned, and documented, concurrently for each phase of the software lifecycle. The software verification shall be done as planned. The planning shall address:

- The evaluation of safety integrity requirements
- the selection and documentation of verification strategies, activities and techniques
- The selection and usage of verification tools
- The evaluation of verification results
- The corrective actions to be taken

Verification documentation shall be performed after the satisfactory verification of each phase. It shall include:

- Identification of items to be verified
- Identification of the information from which it has been done the verification
- non-conformances, such as software aspects poorly adapted to the problem.

The output information from each phase shall include:

- Adequacy of the specification, design, or code for:
  - Functionality
  - Safety integrity, performance and other safety planning requirements
  - Readability, by the development team
  - Testability
  - Safe modification
- Adequacy of the validation planning for describing the design of the present phase.

- Check incompatibilities between:
  - the tests described in the current phase and the ones described in the previous one.
  - the outputs within the current phase

The following verification activities shall be executed for the chosen software development lifecycle:

- Software safety requirements verification
  - Check whether software safety requirements specification fulfils the E/E/PE system safety requirement specification for functionality, safety integrity, performance, and other safety planning requirements.
  - Check whether the validation plan fulfils the software safety requirement specification
  - Check for incompatibilities between the software safety requirements specification and:
    - the E/E/PE system safety requirements specification
    - the validation plan
- Software architecture verification
  - Check whether the software architecture design fulfils the software safety requirements specifications
  - Check the adequacy of the integration tests specified in the software architecture design
  - Check the adequacy of major elements with respect to:
    - feasibility of the safety performance required
    - testability for further verification
    - readability by the development and verification team
    - safe modification to permit further evolution
  - Check for incompatibilities between:
    - the software architecture design and the software safety requirements specification
    - the software architecture design and its integration tests
    - the software architecture integration tests and the validation plan for software aspects of system safety

- Software system design verification
  - Check whether the software system design fulfils the software architecture design
  - Check whether the specified tests of the software system integration fulfil the software system design
  - Check the adequacy of the attributes of each major element of the software system design specification with respect to:
    - feasibility of the safety performance required
    - testability for further verification
    - readability by the development and verification team
    - safe modification to permit further evolution
  - Check for incompatibilities between:
    - the software system design specification and the software safety architecture design
    - the software system design specification and the software system integration test specification
    - the tests required by the software system integration test specification and the software architecture integration test specification
- Software module design verification
  - Check whether the software module design specification fulfils the software system design specification
  - Check the adequacy of the software module test specification with the software module design specification
  - Check the adequacy of the attributes of each software module with reference to:
    - feasibility of the safety performance required
    - testability for further verification
    - readability by the development and verification team
    - safe modification to permit further evolution
  - check the incompatibilities between:
    - the software module design specification and the software system design specification
    - the software module design specification and the software

module test specification (for each software module)
    * the software module test specification and the software system integration test specification

- Code verification
  - The source code shall be verified by static methods. it shall achieve conformance to the software module design specification, the required coding standards, and the validation plan.
- Data verification
  - Data structure shall be verified
  - The application data shall be verified for consistency with data structures, completeness against the application requirements, compatibility with the underlying system software, and correctness of the data values
  - Operational parameters shall be verified against the application requirements
  - Interfaces and associated software (sensors, actuators and off-line interfaces) shall be verified for detection of anticipated interface failures and tolerance to anticipated interface failures
  - Communication interfaces and associated software shall be verified for an acceptable level of failure detection, protection against corruption and data validation.
- Timing performance verification
  - Verification of the predictability of behaviour in the time domain
- Software module testing (3.2.3)
- Software integration testing (3.2.3)
- PE integration testing (3.2.4)
- Software aspects of system safety validation (3.2.6)

3.3 Application of IEC 61508-3

This section of the thesis focuses on the implementation of IEC 61508-3. For that, there exists the sixth part of the standard that walks the user through the steps needed to develop a testing plan; IEC 61508-6 [13]. It

is worth noting that parts two and three of the standard do not instruct on the selection of an appropriate SIL for a given tolerable risk. However, the aforementioned parts do give guidance over:

- Application measures and techniques corresponding to safety integrity levels (Annex B provides a table with the measures and techniques associated to each SIL level for software-related safety systems).
- Control of systematic failures and random failures by feature design.

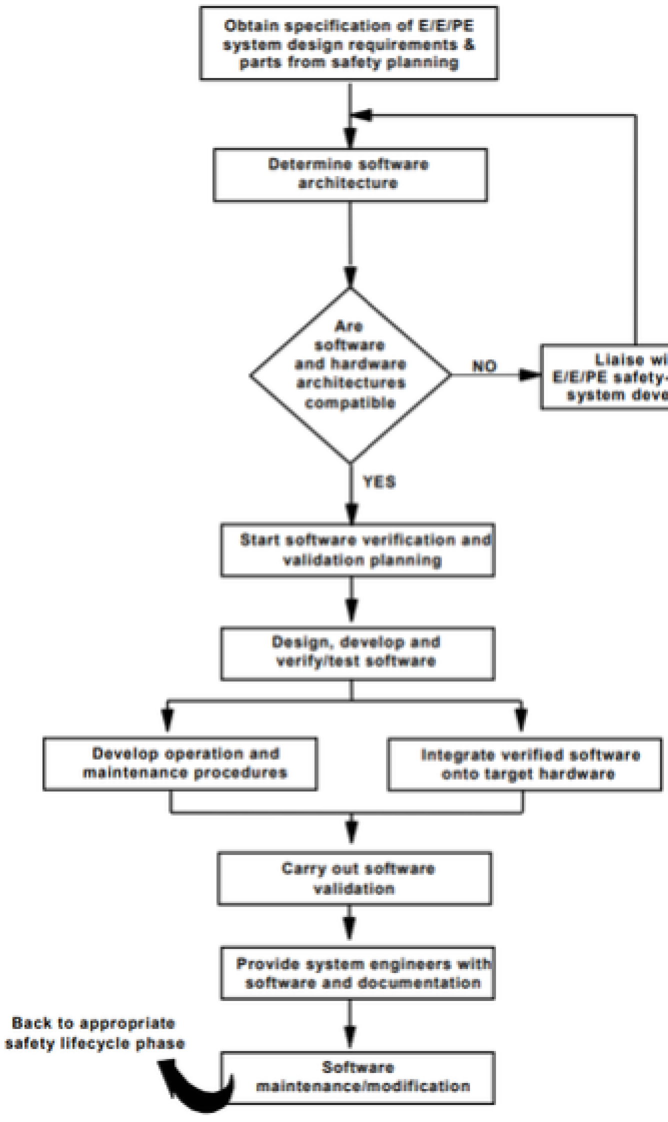

Obtain specification of E/E/PE system design requirements & parts from safety planning
Determine software architecture
Are software and hardware architectures compatible
NO
Liaise with E/E/PE safety-related system developer
YES
Start software verification and validation planning
Design, develop and verify/test software
Develop operation and maintenance procedures
Integrate verified software onto target hardware
Carry out software validation
Provide system engineers with software and documentation
Back to appropriate safety lifecycle phase
Software maintenance/modification

Figure 3.3: Diagram of the application of IEC 61508-3 [13].

The functional steps shown in the figure 3.3 above are defined using the safety lifecycle requirements in the following way:

Obtain requirements for the E/E/PE safety-related system: Present the relevant parts of the safety planning described in IEC 61508-2.

Software architecture: Determine the software architecture for all safety functions allocated to software. Implement software architecture subsection within 3.2.3.

Review the E/E/PE safety-related system: Review with the system's supplier the software and hardware trade-off safety implications and their architectures. Iterate until hardware and software compatibility is reached.

Start the planning for software safety verification and validation: The developing requirements of the planning for software safety verification and validation are presented in subsections 3.2.3 and 3.2.2.

Design, develop and verify/test the software: The software testing shall take into account the software safety planning, the software safety integrity level and the software safety lifecycle.

Operation and maintenance procedures: Develop the software aspects of the maintenance and operation procedures that shall be followed when operating the system. Shall be done in parallel with integration of the verified software step. Subsection 3.2.5 shows how and what aspects the procedures may define.

Integration of verified software into hardware: Integrate the verified software onto the target device. Shall be done in parallel with the definition of operation and maintenance procedures. Subsection 3.2.4 presents the requirements that have to be made for the integration purposes.

Carry out software validation: Validate the software in the integrated E/E/PE safety-related systems. The specifications about this validation process can be found in subsection 3.2.6.

Provide system engineers with software and documentation: Deliver the software safety validation results to system engineers.

Chapter 4

Study case

In this section, the study case and the test planning guidelines are defined. The E/E/PE system used is the burger model of a TurtleBot3. The standard implementation for the system may seem superfluous as the hazardous capabilities of the studied case are almost negligible. However, it serves as a suitable showcase of the implementation guidelines of the standard IEC 61508. The implementation will account for the generation of theoretical guidelines regarding how the application of the IEC 61508-3 shall be executed for the study case.

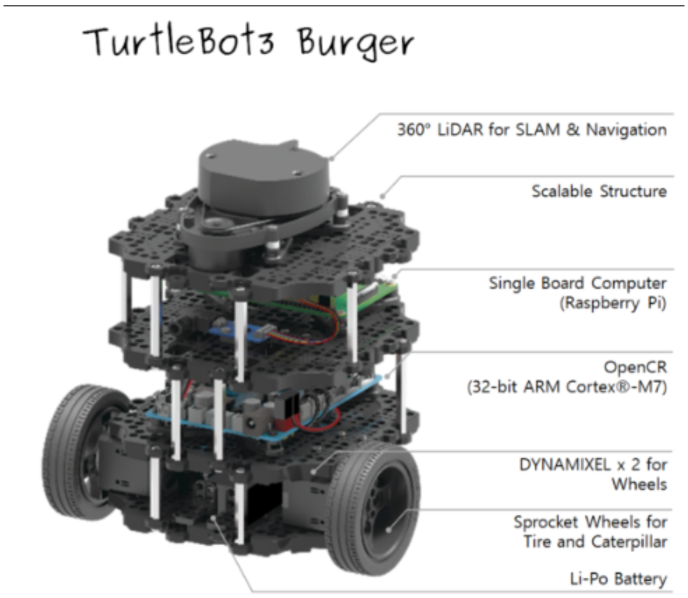

TurtleBot3 Burger
360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer (Raspberry Pi)
OpenCR (32-bit ARM Cortex®-M7)
DYNAMIXEL x 2 for Wheels
Sprocket Wheels for Tire and Caterpillar
Li-Po Battery

Figure 4.1: TurtleBot3 burger component architecture

4.1 System description

In this section, the different specifications of the E/E/EP study case system will be presented. The used study system is the burger model of a TurtleBot3. TurtleBot is a small programmable, ROS-based mobile robot designed as a teaching tool for the ROS platform. It possesses key mobile robot's technologies such as SLAM, navigation and manipulation utilities, which allow the system to be suitable for home service robots [14].

4.1.1 Hardware

The 138 x 178 x 192 mm and 1 kg of mass robot consist of a four-layered structure with two driving wheels, a motion control unit (MCU) containing an inertial measurement unit (IMU) , a single board computer (SBC) , and a laser distance sensor. All the components of the robot are powered up by a 11.1 V DC, 1800mAh Lipo battery.

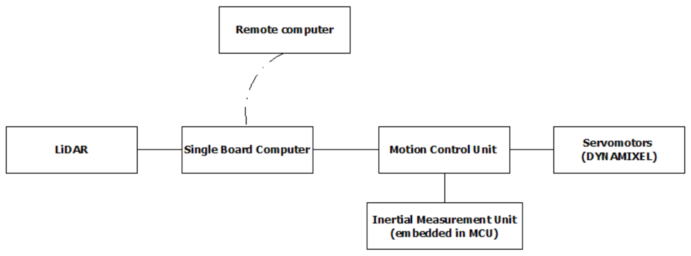


Figure 4.2: Diagram of the hardware components connectivity

Single Board Computer (SBC)

The device capable of regulating and managing the information throughout the TurtleBot3 is a single-board computer, most specifically, a Raspberry Pi B3+. It is responsible for data handling, decision-making, action planning and governing the rest of the components of the vehicle. Furthermore, it is also capable of connecting to a remote computer (laptop) and receiving and sending commands, as well as sharing any of the available data.

Actuator system

Two DYNAMIXEL servomotors (XL430-W250) are used as the main driving mechanism for the robot. They use a PID control and present a stall torque of 1.4 N.m at 11.1 V. A dummy wheel is also present to ensure stability.

The MCU is the main actuator controller. The IMU of the system is embedded in the MCU board The MCU is under the regulation of the SBC and executes the commands received regarding movement and odometry data acquisition.

The robot has a maximum payload of 15 kg, a maximum translational velocity of 0.22 m/s and a maximum rotational velocity of 2.84 rad/s.

Sensoric system

The main sensors used by the robot are a 360 Laser Distance Sensor (LiDAR) and an inertial measurement unit (IMU). Despite not being very relevant in the following functionalities, the servomotors are able to withdraw velocity, angular positions, load, and trajectory, which can be used as another sensor device. More sensors can be attached to the robot to increase its functionalities, such as ultrasonic sensors, contact switches or cameras. However, in this project, it is not considered the addition of other components. The LiDAR, 360 Laser Distance Sensor

LDS-01, is a 2D 360 degrees scanner used to collect data to perform SLAM (Simultaneous Localization and Mapping) and Navigation. It operates at 5V and has a detection distance of 120 mm   3500 mm. The accuracy and precision depend on the distance range as shown in the following table 4.1. The LASER presents a safety of Class 1 from IEC60825.

Item	Specifications
Distance range	120 $\sim$ 3500 mm
Distance accuracy (120 $\sim$ 499 mm)	$\pm$15 mm
Distance accuracy (500 $\sim$ 3500 mm)	$\pm$5.0%
Distance precision (120 $\sim$ 499 mm)	$\pm$10 mm
Distance precision (500 $\sim$ 3500 mm)	$\pm$3.5%
Angular range	360°
Angular resolution	1°
Scan rate	300 $\pm$ 10 rpm

Table 4.1: 360 Laser Distance Sensor, LDS-01, specifications

The IMU consists of a 3-axis gyroscope and a 3-axis accelerometer. It is the main reliable source of information for odometry purposes, as it enables the robot to locate itself as well as monitor its movement. It is embedded in the MCU board. In this project, the information regarding displacement in the z-direction (vertical displacement) and rotation over x and y (pitch and roll) will not be considered. Only 2D planar mapping is required.

4.1.2 Software

The TurtleBot3 operates with ROS (Robot Operating System). ROS is an open-source software development kit that provides the building blocks, such as libraries and software tools, to develop robot applications[15]. The ROS platform is able to control the main functionalities, data management, action planning and connectivity procedures. There are other software tools in use, such as the DYNAMIXEL SDK and Arduino IDE, which are used for the lower control of the servomotors and t he MCU driver. However, ROS is used to govern and manage the actions that the MCU executes.

To control and monitor the robot, a remote computer can be linked to the robot's single-board computer (Raspberry Pi). ROS is used to establish a connection between the remote computer and the single board computer and publish data indistinctively in both machines.

The main instructions used for the project are already built-in commands to set up the ROS dependencies and interactions needed, as well as to activate certain functionalities of the robot. Functionalities can be added to the vehicle by programming with ROS-supported languages, such as C++ or Python. The vehicle, as well as a remote computer, shall be able to have ROS compatibility and contain the necessary up-to-date packages The main ones used to verify the main functionalities of the device are the following, extracted from the documentation guides of the TurtleBot3 and compiled by the DigiTally team from Aalto University's Autonomous Driving research group [16].

- Initiate ROS environment [comm1]

```
    $ roscore
```

- Access TurtleBot3 from computer via ssh [comm2]

```
    $ ssh ubuntu@<IP address of TurtleBot>
```

- Definition of TurtleBot type [comm3]

```
    $ export TURTLEBOT3_MODEL=$burger
```

- Bring up basic packages (make the robot operational) [comm4]

```
    $ roslaunch turtlebot3_bringup
      turtlebot3_robot.launch
```

- Read ROS topics [comm5]

```
    $ rostopic list
    $ rostopic info /<topic>
```

- Activate teleoperation with keyboard [comm6]

```
    $ roslaunch turtlebot3_teleop
      turtlebot3_teleop_key.launch
```

- Activate SLAM [comm7]

```
    $ roslaunch turtlebot3_slam turtlebot3_slam.
      launch
```

- Activate Navigation [comm8]

```
    $ roslaunch turtlebot3_navigation
      turtlebot3_navigation.launch
```

The ROS environment also includes visualization tools to help interpret the data gathered. R-viz is a ROS visualization software tool used to visualize the robot's behaviour and show display the required data. It can be considered to show what the robot senses and thinks it is happening. It is a ROS package that is also able to communicate with the connected devices. Most of the navigation directives used in this project are handled by r-viz. The study E/E/EP system has been operated with

ROS Kinematic Kane version and contains the main TurtleBot3 packages. Note that the single board computer and the external computer use Ubuntu operating system on Linux.

4.1.3 Functionality

For the purpose of designing a test plan for the robot, some of the main states and functionalities of the vehicle have been sorted into possible modes of operation. Note that the robot's capabilities extend beyond the selected functionalities and that the selected functionalities have been tailored for the purposes of the project, meaning that the capabilities of the functionality may be greater than the ones presented.

These modes can be divided into standby states, where the robot remains still and is awaiting instructions or readiness, and functional states, where the robot executes some action.

Standby states

The standby modes may present a sequential increase in the usage readiness of the robot. The robot should not be allowed to move and the main purpose of these states is to prepare/set up the robot.

<u>Disconnected:</u> The robot is powered up; components have access to the power supply but no orders shall be given. Communication with the remote computer is not present. No movements allowed

Idle: The robot is powered up. There is a connection between the robot and the remote computer via ssh. The robot is ready to be initialized. Connectivity self-check between single board computer and remote computer should be done. No movements allowed.

Ready: The robot is powered up. There is a connection between the robot and the remote computer. Roscore nodes are initialized; the robot is ready for launch commands. Components and systems should be fully operative. Self-check of components should be executed and reported using ROS connectivity to the remote computer. LiDAR movement allowed. Robot movement is allowed if self-checks are okay and order is given.

A successful response from the following command [comm4] shall be a key point to enter this state.

Functional modes

n order to make use of the functional modes, the E/E/EP system has to be, at least, in the "Ready" state defined previously.

Teleoperation: Nodes that allow teleoperation are opened. The robot can be teleoperated by the user from a remote computer with the keyword; WASDX keys. Longitudinal, transversal and rotational movements over the vertical axis shall be inputted; movement shall not be backward, over the vertical axis or rotation over the longitudinal (roll) and transverse(pitch) axis. Real-time communication with the robot. Collisions may be permitted. Safety function shall be defined if severe collision occurs. Low level of autonomy and safety.

A successful response from the command [comm7] shall be a key point to enter this state.

SLAM: Nodes that allow SLAM are opened. R-viz is also operational. This mode of operation shall be combined with the teleoperation functionalities to move the robot and create the mapping. The map generated shall be a two-dimensional occupancy grid map (OGM). Teleoperation functionality is the same as described above. Real-time communication with the robot; mapping updates in r-viz shall also be up to date with the robot's progress. As the movement is controlled by the user, the safety functionality and autonomy level are the same as in the teleoperated mode.

A successful response from the command [comm8], SLAM, and command [comm7], keyword teleoperation, shall be a key point to enter this state.

Navigation: Nodes that allow Navigation are opened. A map (OGM) file is needed. R-viz tools are used to perform navigation. The estimation tool in r-viz shall be used to initialize the robot orientation and position into the loaded map. The goal tool in r-viz shall be used to set goal points and orientation in the loaded map. Once a goal is set, the robot shall move towards it avoiding collisions. The robot shall also update the mapping while following the goal and shall be able to detect and avoid dynamic obstacles. The robot shall not be able to recognize if obstacles belong to the same or different entities but shall avoid them. The minimum path shall be calculated when targeting the goal point. Safety functions shall be defined when the target cannot be reached or imminent collision occurs. Communication between the remote computer and robot shall be up to date. Safety mechanisms are needed and the level of autonomy is medium-high.

A successful response from the command [comm9] shall be a key point to enter this state.

4.2 Standard application

The application of the standard is not fully covered in this section. Due to time constraints, the implementation of the standard will only cover the first three steps of the guidelines provided by the standard. Hence, a rough characterization of the application of the standard will be developed for the study case described above. The implementation has been guided by section 3.3.

The software development lifecycle will be based on the V model presented in figure 3.1. It complies with the requirements specified in subsection 3.2.7.

4.2.1 Obtain requirements for the E/E/PE safety-related system

Safety functions are used to mitigate the risk of creating accounted hazardous situations. A key functionality of the safety functions is to bring the system into a safe state in which the risk situation can be assessed and controlled appropriately. The safety functions are defined taking into account the safety integrity level (SIL) that is wanted to achieve. Specific implementations and architectural measures of the E/E/PE safety-related systems depend on what the SIL wanted to achieve.

Safety-related application

The following block related to safety functions and states has been developed to exemplify the application of the standard IEC61508 and is not an intrinsic behaviour of the marketed robot. Furthermore, the implementation and feasibility of these functions are out of the project scope.

It should also be taken into consideration that this project expands on the software measures that can be applied to perform an E/E/PE safety-related system that mitigates certain risks. Meaning that, although mechanical constraints and elements can also be used as SIL-specific safety-related systems, this analysis only accounts for software procedures.

Note that the states and functions defined tend to be as simplistic as possible as they aim to showcase the utilities of the standard. Specifically, the safety functions will be defined by focusing on:

- Mitigation of the risk of bumping into objects, and;
- Regulation of the vehicle speed so that any collision shall not inflict any injury.

SIL target

Taking into account the displayed definitions of the characteristics and main functionalities of the studied system (Turtlebot3 burger model), there would practically not be needed risk reduction mechanisms as the system is hardly hazardous on its own. However, for the purpose of this thesis, a higher-than-needed SIL of 3 target has been selected for the E/E/PE safety-related system. A SIL level of 3 is normally attributed to safety-related systems that perform a high-level risk reduction, typically used to mitigate the risk of possible life-threatening situations.

Safe state

The main considered safety state shall be "Safe" state. When a diagnostic indicates that a safety action must be taken, the system shall redirect itself into the "Safe" state.
Definition: The robot is powered up. There is a connection between the robot and the remote computer. Roscore nodes are initialized; the robot is not ready for launch commands. Components and systems should indicate their operativity status. Self-check of components should be executed and reported using ROS connectivity to the remote computer. Fault detection should indicate the reason for the safe state activation. LiDAR movement is not allowed. Robot movement is not allowed. The safe state can only be recovered when no faults are active and manual intervention allows for its recovery.

The aforementioned safe state shall be achieved using software safety-related activities that comply with the SIL target specified. A list of measures and techniques the software safety system shall take into account can be found in the annexe A of the IEC 61508-3 standard [6]. The main critical sectors, focused for the purposes of this thesis, that need supervision will be the LiDAR, specifically the capabilities of detecting the proximity of objects, and the IMU, specifically the capability to detect translational and angular speeds and accelerations. These elements altogether with the appropriate software will constitute the safety-related systems.

Safety detection tool

There shall exist a safety detection software tool that monitors the sensors' data and decide whether the system shall be put into a safe state. The tool may be referred to as "Diagnostics" and shall be responsible for detecting if the system's normal-mean functionality data is properly handled, recurrently checking critical data and expressing faults. The requirements of the detection tool shall comply with the support tools specifications of subsection 3.2.3.

Safety functions

Safety functions are designed to address the two main desired mitigations (stated in 4.2.1).

Safety distance: Addresses the mitigation of bumping into objects. This safety function will be operating in high-demand mode as the software safety systems responsible for braking may be triggered very often. To consider the targetted SIL and according to table 2.1, the safety distance function shall present a frequency of less than 10^{-7} dangerous failures per hour.

The safety function shall monitor the proximity of objects when the system is moving in some of the functional states (4.1.3). When the proximity value of an object is under a specified threshold (safety distance), the safety function shall bring the system to its safe state. The information about object proximity shall be gathered from the LiDAR of the system. The acceptable boundaries from which to choose a suitable safe distance shall take into account the sensibility and range of the used LiDAR (table 4.1). Note that the Turtlebot3 burger presents a body radius of 105 mm.

- Proposed safety distance boundaries: 120 $\sim$ 3500 mm
- Proposed default value: 150 mm

The safety function shall be allocated in a safety-related system composed of the LiDAR and the MCU. This safety-related system will be referred to as the safety distance system.

Safety speed: Addresses the mitigation of collision severity. This safety function will be operating in low-demand mode as the software safety systems responsible to account for breaches in the maximum allowed operating speed will not be prompted very regularly. To consider the targetted SIL and according to table 2.1, the safety distance function shall present an average probability of less than 10^{-3} dangerous failures.

The safety function shall monitor the velocity of the vehicle when the system is moving in some of the functional states (4.1.3). When the system's speed value is over a specified threshold (maximum allowed speed), the safety function shall bring the system to its safe state. The information about translational and angular speeds shall be gathered from the IMU of the system. The acceptable boundaries from which to choose a suitable safe distance shall take into account the sensibility and range of the used servomotors.

- Proposed translational velocity boundaries: 0.1 $\sim$ 0.22 m/s
- Proposed angular velocity boundaries: 0.1 $\sim$ 2.84 rad/s
- Proposed default maximum translational velocity: 0.22 m/s
- Proposed default maximum angular velocity: 2.84 rad/s

The safety function shall be allocated in a safety-related system composed of the IMU and the MCU. This safety-related system will be referred to as the safety speed system.

DISCLAMER: In a real-life situation, more safety functions shall be presented, covering all the targeted SIL necessities. Safety systems would have to be defined properly as stated in IEC 61508-2.

4.2.2 Software architecture

Techniques and measures: To fulfil the SIL 3 target the software architecture shall present a defined status of the following techniques and measures [6]:

- Fault detection
- Error detection codes
- Failure assertion programming
- Diverse monitor techniques
- Functionally diverse redundancy
- Stateless software design
- Graceful degradation
- Modular approach
- Use of trusted software elements
- Forward traceability between the software safety requirement specification and software architecture
- Backward traceability between software safety requirements specification and software architecture
- Structured diagrammatic methods
- Semi-formal methods
- Formal design and refinement methods
- Automatic software generation
- Computer-aided specification and design tools
- Cyclic behaviour, with guaranteed maximum cycle time
- Time-triggered architecture
- Event-driven, with guaranteed maximum response time
- Static resource allocation
- Static synchronisation of access to shared resources

The safety distance function safety software will be present in the SBC, the LiDAR and the MCU. The safety speed function safety software will be present in the SBC, the MCU. Note that the MCU is also responsible for the data acquisition of the IMU. The LiDAR and the IMU are considered acquisition elements, the SBC is the decision-making element, the MCU is the execution controller element and the servomotors are the actuators. Overall, acquisition elements report to decision-making elements. Decision-making elements demand actions from the execution controller. The execution controller formalizes the requests into directives that the actuators can execute.

The LiDAR shall give information regarding the distance of its surroundings continuously. If the SBC detects the LiDAR is operational and does not give conclusive data about the violation of the safe distance parameter, the system shall be allowed to perform its normal operations. Otherwise, the system shall perform the safety-related actions defined in 4.2.1. The appropriate actions shall be assessed by the SBC, which shall give the according commands to the MCU. The SBC shall also display accordingly the fault situation and status. The MCU shall be aware of the safety situation and govern the actuators to perform the detailed actions needed.

The software implementation and architecture presented above allow for software modularity and integration testing. An example of modular tests would be the ones developed for the detection of different sizes and position objects by LiDAR, speed detection by IMU, or speed correction by servomotors. Integration tests could be developed to verify the integration of different modules such as the object detection module and the speed manipulation module, for example, to test the braking of the robot when an object is close.

ROS can be considered the main software tool of the project. It complies with the latest tool requirements.

4.2.3 Planning for software safety verification and validation

The validation plan shall be developed using the guidelines from 3.2.2. And the verification plan shall be developed using the guidelines from 3.2.7.

The rest of the steps regarding the application of the standard require a real implementation of the safety-related software, as well as the execution of testing procedures. As agreed with the owners of the robot system and due to severe time constraints, these implementations shall not be accounted for and are out of the scope of this thesis. This last section was meant for showcasing an exemplification of how the standard should be applied in a real E/E/PE system.

Chapter 5

Conclusion

Functional safety is one of the most important frameworks in product development. Its usability is highly spread within the industrial sector but it is always treated with high discretion as it involves company-specific procedures and sensitive information. For that reason, the availability of comprehensive studies is reduced to a few books and the international standard. This thesis has tried to compile the already existent information into a comprehensible short document that presents the basis of functional safety.

The study of functional safety and its applications paired with a traineeship in a software safety team has been very useful for my learning process. The ability to work as a test engineer has given me a lot of insights into why and how functional safety is important and relevant in any software-related production system, as well as to get a better understanding of the standard from which the safety functional paradigm is regulated. Furthermore, the professional insight of software developers and testers has provided a more realistic overview of how safety is built, caressed and implemented in a real industrial environment.

The study case has been used as a support to showcase the different stages of software safety procedures and to provide a more hands-down experience in the project. However, due to time and system availability constraints, the complete process of functional safety has not been applied to the system. Nevertheless, the opportunity to develop a safety study on a real system has given me a broader, more accurate, perspective on the necessities and time required for safety implementations.

The most important learnings from this thesis have been:

- Functional safety as a paradigm and how it is applied to all industrial environments
- Testing procedures and their place in the software development lifecycle
- How safety is categorized; risks and safety integrity levels.
- The IEC 61508 application

The scope and focus of the thesis have changed over the course of its writing in favour of a more concrete and standard-oriented result. However, there are some topics and actions that should give a more complete status to the thesis and that can be assessed in the future. These descarded topics include:

- Analysis of software development (different models and their advantage over safety implementations)
- Analysis over software and hardware integration (would lay some of the basis about the functional safety paradigm importance)
- In-depth analysis of risk assessment techniques (LOPA, risk graphs, severity matrices, ALARP, ...)
- Software testing metrics and analysis, statistical reasoning for testing techniques
- Implementation and analysis of a real safety-related system
- Functional safety assessment of an E/E/PE system
- Integration validation of safety-related requirements between software and hardware

Acknowledgements

I would like to thank my university supervisors Attila Kovács and Quan Zhou for their disponibility, guidance, and trust and freedom deposited in me.

I would like to warmly thank Toni Kuikka, my industrial supervisor for his presence, advice, trust and his encore, which allowed me to focus on the important steps when developing the thesis. Furthermore, I would also like to thank the entire critical safety team for their professional support and patience.

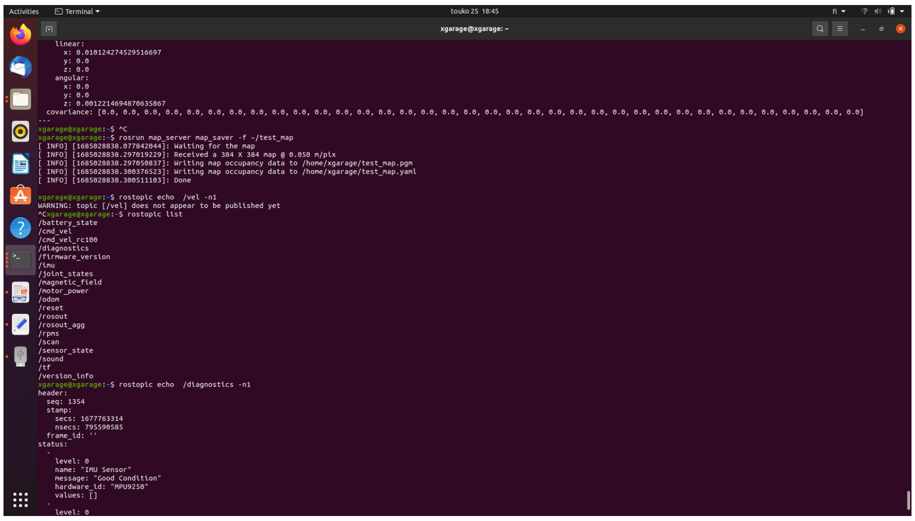
Finally, I would like to thank the Autonomous Driving research group at Aalto, especially the DigiTally team, for allowing me to work with a TurtleBot3 and for their technical support over its functionality and documentation.

Appendix A

TurtleBot3 operations and functionalities - Screenshots

A.1 Rostopic

List of topics that the robot has access to

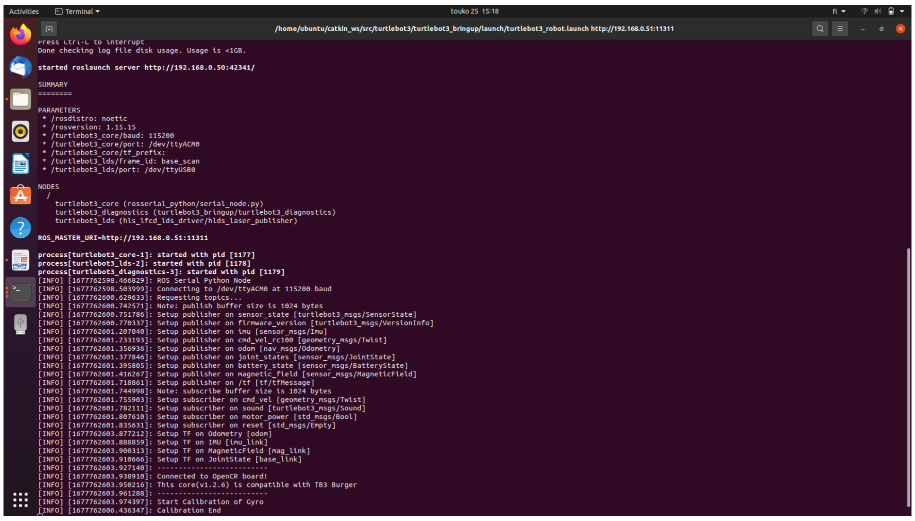


```
    linear:
      x: 0.010124274529516697
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0012214694870635867
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
xgarage@xgarage:~$ ^C
xgarage@xgarage:~$ rosrun map_server map_saver -f ~/test_map
[ INFO] [1685028838.077842044]: Waiting for the map
[ INFO] [1685028838.297019229]: Received a 384 X 384 map @ 0.050 m/pix
[ INFO] [1685028838.297050837]: Writing map occupancy data to /home/xgarage/test_map.pgm
[ INFO] [1685028838.300376523]: Writing map occupancy data to /home/xgarage/test_map.yaml
[ INFO] [1685028838.300511103]: Done

xgarage@xgarage:~$ rostopic echo  /vel -n1
WARNING: topic [/vel] does not appear to be published yet
^Cxgarage@xgarage:~$ rostopic list
/battery_state
/cmd_vel
/cmd_vel_rc100
/diagnostics
/firmware_version
/imu
/joint_states
/magnetic_field
/motor_power
/odom
/reset
/rosout
/rosout_agg
/rpms
/scan
/sensor_state
/sound
/tf
/version_info
xgarage@xgarage:~$ rostopic echo  /diagnostics -n1
header:
  seq: 1354
  stamp:
    secs: 1677763314
    nsecs: 795590585
  frame_id: ''
status:
  -
    level: 0
    name: "IMU Sensor"
    message: "Good Condition"
    hardware_id: "MPU9250"
    values: []
  -
    level: 0
```

Figure A.1: Rostopic list

A.2 Robot initialisation - bring up system

Turtlebot3 self-assesses itself when the system is operative. Showcase of how data is displayed.

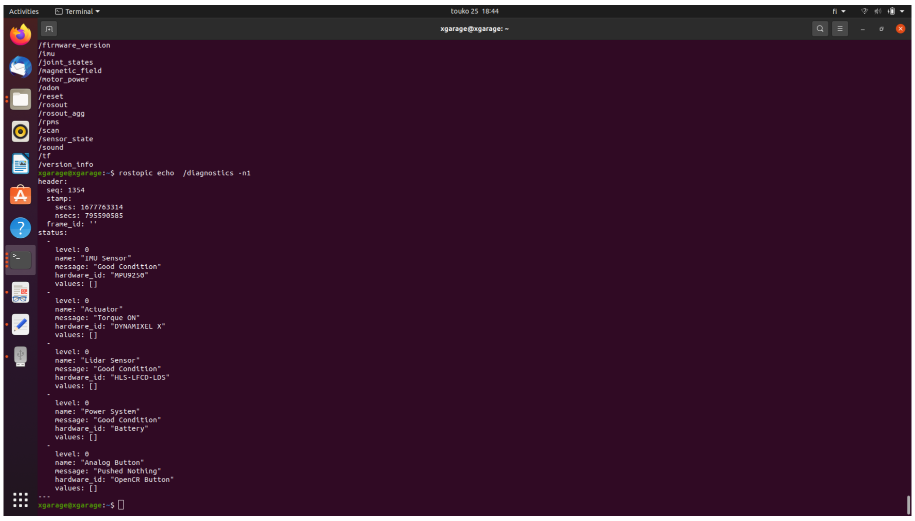


```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.50:42341/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.15
 * /turtlebot3_core/baud: 115200
 * /turtlebot3_core/port: /dev/ttyACM0
 * /turtlebot3_core/tf_prefix:
 * /turtlebot3_lds/frame_id: base_scan
 * /turtlebot3_lds/port: /dev/ttyUSB0

NODES
  /
    turtlebot3_core (rosserial_python/serial_node.py)
    turtlebot3_diagnostics (turtlebot3_bringup/turtlebot3_diagnostics)
    turtlebot3_lds (hls_lfcd_lds_driver/hlds_laser_publisher)

ROS_MASTER_URI=http://192.168.0.51:11311

process[turtlebot3_core-1]: started with pid [1177]
process[turtlebot3_lds-2]: started with pid [1178]
process[turtlebot3_diagnostics-3]: started with pid [1179]
[INFO] [1677762598.466829]: ROS Serial Python Node
[INFO] [1677762598.503999]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1677762600.629633]: Requesting topics...
[INFO] [1677762600.742571]: Note: publish buffer size is 1024 bytes
[INFO] [1677762600.751786]: Setup publisher on sensor_state [turtlebot3_msgs/SensorState]
[INFO] [1677762600.770337]: Setup publisher on firmware_version [turtlebot3_msgs/VersionInfo]
[INFO] [1677762601.207040]: Setup publisher on imu [sensor_msgs/Imu]
[INFO] [1677762601.233193]: Setup publisher on cmd_vel_rc100 [geometry_msgs/Twist]
[INFO] [1677762601.356936]: Setup publisher on odom [nav_msgs/Odometry]
[INFO] [1677762601.377846]: Setup publisher on joint_states [sensor_msgs/JointState]
[INFO] [1677762601.395805]: Setup publisher on battery_state [sensor_msgs/BatteryState]
[INFO] [1677762601.416267]: Setup publisher on magnetic_field [sensor_msgs/MagneticField]
[INFO] [1677762601.718861]: Setup publisher on /tf [tf/tfMessage]
[INFO] [1677762601.744998]: Note: subscribe buffer size is 1024 bytes
[INFO] [1677762601.755903]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
[INFO] [1677762601.782111]: Setup subscriber on sound [turtlebot3_msgs/Sound]
[INFO] [1677762601.807610]: Setup subscriber on motor_power [std_msgs/Bool]
[INFO] [1677762601.835631]: Setup subscriber on reset [std_msgs/Empty]
[INFO] [1677762603.877212]: Setup TF on Odometry [odom]
[INFO] [1677762603.888859]: Setup TF on IMU [imu_link]
[INFO] [1677762603.900313]: Setup TF on MagneticField [mag_link]
[INFO] [1677762603.910666]: Setup TF on JointState [base_link]
[INFO] [1677762603.927140]: -------------------------
[INFO] [1677762603.938910]: Connected to OpenCR board!
[INFO] [1677762603.950216]: This core(v1.2.6) is compatible with TB3 Burger
[INFO] [1677762603.961288]: -------------------------
[INFO] [1677762603.974397]: Start Calibration of Gyro
[INFO] [1677762606.436347]: Calibration End
```

Figure A.2: ROS bring up command and assessment

A.3 ROS diagnostics

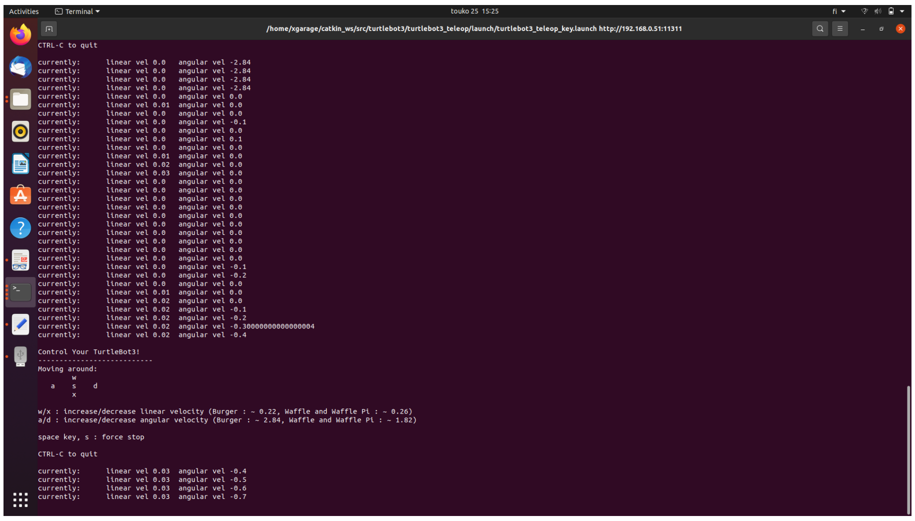
Showcase of diagnostics tool of the robot. Fault detection and safety information display shall be based on this.



```
/firmware_version
/imu
/joint_states
/magnetic_field
/motor_power
/odom
/reset
/rosout
/rosout_agg
/rpms
/scan
/sensor_state
/sound
/tf
/version_info
xgarage@xgarage:~$ rostopic echo  /diagnostics -n1
header:
  seq: 1354
  stamp:
    secs: 1677763314
    nsecs: 795590585
  frame_id: ''
status:
  -
    level: 0
    name: "IMU Sensor"
    message: "Good Condition"
    hardware_id: "MPU9250"
    values: []
  -
    level: 0
    name: "Actuator"
    message: "Torque ON"
    hardware_id: "DYNAMIXEL X"
    values: []
  -
    level: 0
    name: "Lidar Sensor"
    message: "Good Condition"
    hardware_id: "HLS-LFCD-LDS"
    values: []
  -
    level: 0
    name: "Power System"
    message: "Good Condition"
    hardware_id: "Battery"
    values: []
  -
    level: 0
    name: "Analog Button"
    message: "Pushed Nothing"
    hardware_id: "OpenCR Button"
    values: []
---
xgarage@xgarage:~$ 
```

Figure A.3: Diagnostics showcase capabilities

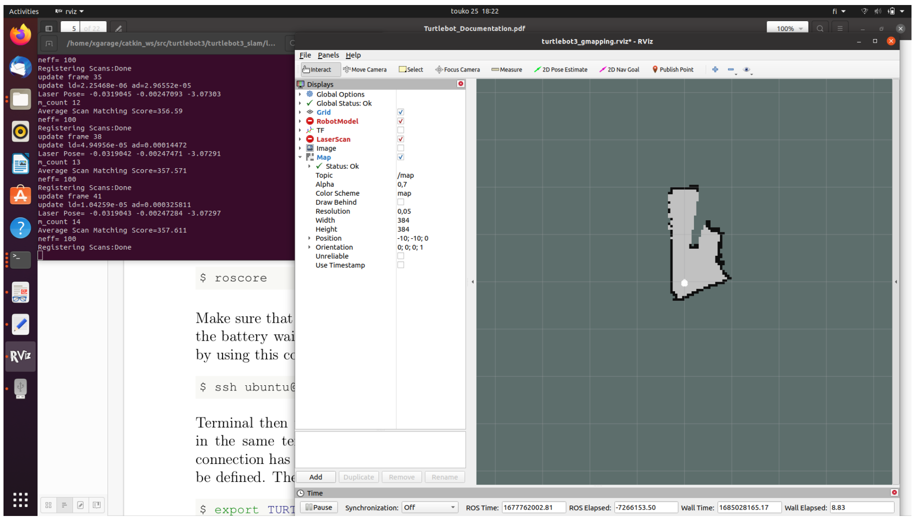
A.4 TurtleBot3 teleoperation


Activities
Terminal
touko 25 15:25
/home/xgarage/catkin_ws/src/turtlebot3/turtlebot3_teleop/launch/turtlebot3_teleop_key.launch http://192.168.0.51:11311

```
CTRL-C to quit

currently:      linear vel 0.0   angular vel -2.84
currently:      linear vel 0.0   angular vel -2.84
currently:      linear vel 0.0   angular vel -2.84
currently:      linear vel 0.0   angular vel -2.84
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel -0.1
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.1
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.02  angular vel 0.0
currently:      linear vel 0.03  angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel -0.1
currently:      linear vel 0.0   angular vel -0.2
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.02  angular vel 0.0
currently:      linear vel 0.02  angular vel -0.1
currently:      linear vel 0.02  angular vel -0.2
currently:      linear vel 0.02  angular vel -0.30000000000000004
currently:      linear vel 0.02  angular vel -0.4

Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear vel 0.03  angular vel -0.4
currently:      linear vel 0.03  angular vel -0.5
currently:      linear vel 0.03  angular vel -0.6
currently:      linear vel 0.03  angular vel -0.7
```

Figure A.4: Teleoperation command

A.5 TurtleBot3 SLAM



```
neff= 100
Registering Scans:Done
update frame 35
update ld=2.25468e-06 ad=2.96552e-05
Laser Pose= -0.0319045 -0.00247093 -3.07303
m_count 12
Average Scan Matching Score=356.59
neff= 100
Registering Scans:Done
update frame 38
update ld=4.94956e-05 ad=0.00014472
Laser Pose= -0.0319042 -0.00247471 -3.07291
m_count 13
Average Scan Matching Score=357.571
neff= 100
Registering Scans:Done
update frame 41
update ld=1.04259e-05 ad=0.000325811
Laser Pose= -0.0319043 -0.00247284 -3.07297
m_count 14
Average Scan Matching Score=357.611
neff= 100
Registering Scans:Done
```

```
$ roscore
```

Make sure that
the battery wai
by using this co

```
$ ssh ubuntu@
```

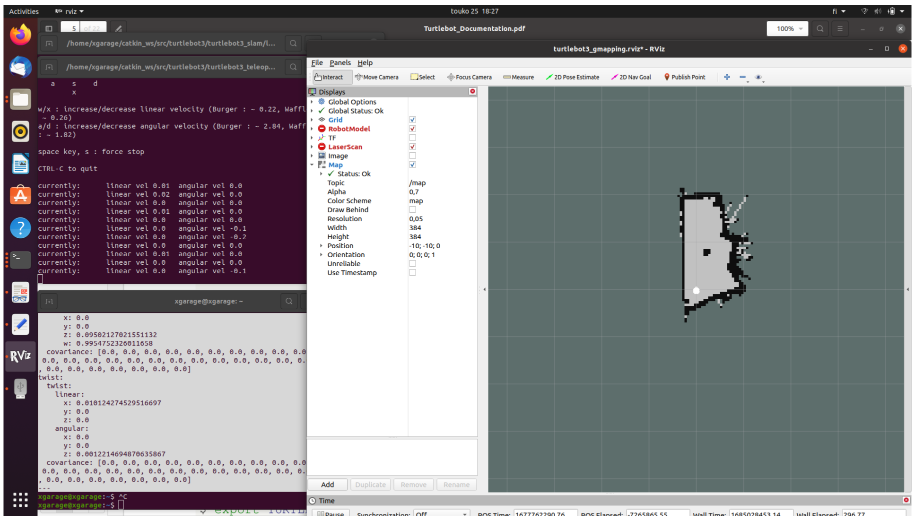
Terminal then
in the same te
connection has
be defined. The

```
$ export TURT
```

Global Options	
✓ Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Image	☐
Map	✓
✓ Status: Ok	
Topic	/map
Alpha	0,7
Color Scheme	map
Draw Behind	☐
Resolution	0,05
Width	384
Height	384
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐

Synchronization: Off ROS Time: 1677762002.81 ROS Elapsed: -7266153.50 Wall Time: 1685028165.17 Wall Elapsed: 8.83

Figure A.5: Initialization of SLAM directives. Showcase of R-viz


Activities
rviz
touko 25 18:27
Turtlebot_Documentation.pdf
/home/xgarage/catkin_ws/src/turtlebot3/turtlebot3_slam/l...
/home/xgarage/catkin_ws/src/turtlebot3/turtlebot3_teleop...
xgarage@xgarage: ~
turtlebot3_gmapping.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Add
Duplicate
Remove
Rename
Time

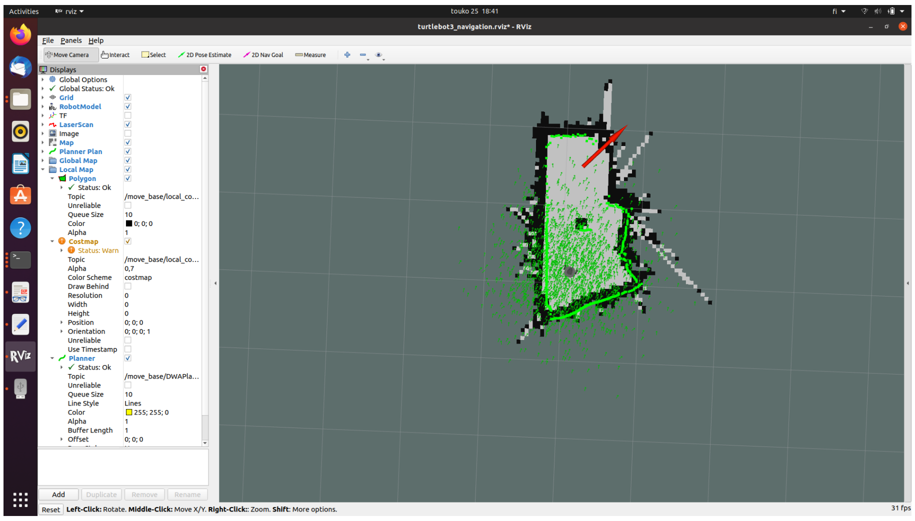
```
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffl
 ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waff
: ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.02  angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel -0.1
currently:      linear vel 0.0   angular vel -0.2
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel -0.1
```

```
      x: 0.0
      y: 0.0
      z: 0.09502127021551132
      w: 0.9954752326011658
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
twist:
  twist:
    linear:
      x: 0.010124274529516697
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0012214694870635867
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
xgarage@xgarage:~$ ^C
xgarage@xgarage:~$
```

Displays	
Global Options	
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Image	☐
Map	✓
Status: Ok	
Topic	/map
Alpha	0,7
Color Scheme	map
Draw Behind	☐
Resolution	0,05
Width	384
Height	384
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐

Figure A.6: SLAM functionalities. Showcase of R-viz

A.6 TurtleBot3 Navigation



Displays	
Global Options	
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Image	☐
Map	✓
Planner Plan	✓
Global Map	✓
Local Map	✓
Polygon	✓
Status: Ok	
Topic	/move_base/local_co…
Unreliable	☐
Queue Size	10
Color	■ 0; 0; 0
Alpha	1
Costmap	✓
Status: Warn	
Topic	/move_base/local_co…
Alpha	0,7
Color Scheme	costmap
Draw Behind	☐
Resolution	0
Width	0
Height	0
Position	0; 0; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐
Planner	✓
Status: Ok	
Topic	/move_base/DWAPla…
Unreliable	☐
Queue Size	10
Line Style	Lines
Color	🟨 255; 255; 0
Alpha	1
Buffer Length	1
Offset	0; 0; 0

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options.

Figure A.7: Navigation functionality of the TurtleBot3. R-viz showcase

Bibliography

[1] Simpson Kenneth G. L. Smith David J., ed. *Safety critical systems handbook a straightforward guide to functional safety: IEC 61508 (2010 edition) and related standards, including process IEC 61511, machinery IEC 62061 and ISO 13849*. 3rd ed. Amsterdam: Butterworht-Heinemann/Elsevier, 2011. ISBN: 1-283-02031-9.

[2] Glenford J. Myers. *The Art of Software Testing*. 2nd ed. John Wiley Sons, 2004. ISBN: 0-471-46912-2.

[3] *IEC 61508-5:2010 en. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part5: Examples of methods for the determination of safety integrity levels*. Standard. Netherlands Standardization Institute, 2010.

[4] *IEC 61508-4:2010 en. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part4: Definitions and abbreviations*. Standard. Netherlands Standardization Institute, 2010.

[5] *exida* Loren Stewart. *Back to Basis 07-Safety Lifecycle - IEC 61508*. 2019. URL: `https://www.exida.com/Blog/back-to-the-basics-07-safety-lifecycle-iec-61508`.

[6] *IEC 61508-3:2010 en. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part3: Software requirements*. Standard. Netherlands Standardization Institute, 2010.

[7] MTL Instruments Group. "An introduction to Functional Safety and IEC 61508". In: (Mar. 2002). URL: `http://www.mtl-inst.com/images/uploads/datasheets/App_Notes/AN9025.pdf`.

[8] TÜV SÜD. *Functional Safety Overview, Protect users from technology and technology from users.* 2023. URL: `https://www.tuvsud.com/en-us/services/functional-safety/about`.

[9] Simon Dean. "IEC 61508 - Understanding functional safety assessment". In: *Measurement + Control,* 32 (7 Sept. 1999), pp. 201–204. URL: `http://doi.org/10.1177/002029409903200703`.

[10] *IEC 61508-1:2010 en. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part1: General requirements.* Standard. Netherlands Standardization Institute, 2010.

[11] *Airbrake* Frances Banks. "V-Model: What Is It And How Do You Use It?" In: (2016). URL: `https://blog.airbrake.io/blog/sdlc/v-model`.

[12] Steven Rakitin. *Software Verification and Validation for Practitioners and Managers.* 2nd ed. Artech, 2001. URL: `http://ieeexplore.ieee.org/document/9100784`.

[13] *IEC 61508-6:2010 en. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part6: Guidelines on the application of IEC 61508-2 and IEC 61508-3.* Standard. Netherlands Standardization Institute, 2010.

[14] ROBOTIS. *TurtleBot3 e-manual.* 2023. URL: `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#overview`.

[15] Open Robotics. *ROS-Robot Operating System.* 2023. URL: `https://www.ros.org/`.

[16] DigiTally Project Team Autonomous Driving research group (Aalto University). *TurtleBot3 Documentation.* 2023.

List of Figures

List of Tables

List of Symbols

ALARP	As Low As Reasonably Practicable
E/E/PE	Electrical/Electronic/Programmable Electronic
EUC	Equipment Under Control
HW	Hardware
IMU	Inertial Measurement Unit
LOPA	Layers of Protection Analysis
MCU	Motion Control Unit
PE	Porgrammable Electronic
ROS	Robot Operating System
SBC	Single Board Computer
SDLC	Software Development Life Cycle
SIL	Safety Integrity Levels
SW	Software