Master's thesis

NTNU
Norwegian University of Science and Technology

Helene Maria Festø Pisani

Simulation-based testing of an all-electric subsea valve for safety demonstration

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Ludvig Björklund
June 2023

NTNU
Norwegian University of
Science and Technology

Helene Maria Festø Pisani

Simulation-based testing of an all-electric subsea valve for safety demonstration

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Ludvig Björklund
June 2023

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Preface

This thesis, titled ”Simulation-based testing of an all-electric subsea valve for safety demonstration,” is written as part of the TTK4900 course, Master’s Thesis in Engineering Cybernetics, at the Norwegian University of Science and Technology. The course carries a weight of 30 credits. It builds upon the completion of the TTK4551 course, Specialization Project in Engineering Cybernetics, during the autumn semester of 2022.

Acknowledgement

I would like to express my gratitude to my supervisor, Mary Ann Lundteigen, for her invaluable support and encouragement throughout the duration of my master's thesis. Her guidance and feedback have been immensely helpful in shaping the outcome of my research. I appreciate her expertise in the field and her commitment to my academic growth.

I would also like to thank my co-supervisor, Ludvig Björklund, for his valuable insights, technical expertise, and collaborative efforts. His guidance and feedback have been immensely helpful.

Furthermore, I would like to extend my thanks to my fellow co-students in the master's program for their feedback and support.

Lastly, I want to acknowledge the support of my family and friends. Their encouragement and understanding have been crucial throughout my five years of study. I am grateful for their belief in me and their constant support.

Summary

The Research Center for Subsea Production and Processing has an ongoing project to develop a digital twin for the safety demonstration of an all-electric actuation system for a subsea valve. The research is conducted by Ph.D. candidate Ludvig Björklund. All-electric valves are not new to the industry but are new for safety-critical functions in Norway. Before the all-electric valve can be used for this, it should be provided documentation that the valve meets the safety requirements.

Digitization and automation continue to drive the emergence of complex systems. In this context, novel techniques such as simulation-guided approaches are emerging for automatic testing and verification of complex systems. These approaches utilize a digital model and simulations to test different scenarios.

To address these challenges, this thesis investigates two types of simulation-guided testing approaches where one is implemented on a digital twin. The all-electric valve's digital twin comprises a lithium-ion battery and a battery management system, among other things. For this master thesis, a simple digital twin has been implemented by Ludvig Björklund. As part of this thesis work, the digital twin has been exported to a functional mock-up unit and implemented in Python to provide a versatile testing environment.

In Python, optimization-guided falsification using signal temporal logic and Bayesian optimization have been implemented and applied to several fault scenarios on the digital twin. The battery management system is responsible for observing and handling faults in the battery. The objective was to use the optimization-guided falsified to search for bad inputs to inject faults into the battery and observe the management of the battery management system. One test case has been implemented in a graphical user interface to showcase the testing method to test engineers.

The method successfully identified falsifying examples and injected faults the battery management system could not resolve.

Sammendrag

Forskningsenteret for Subsea Produksjon og Prosessering har et pågående prosjekt med formål å utvikle en digital tvilling for sikkerhetsdemonstrasjon av et hel-elektrisk aktuatorsystem for en subsea ventil. Doktorgradskandidat Ludvig Björklund står for forskningen. Selv om hel-elektriske ventiler ikke er nye i bransjen, er de nye innenfor sikkerhetskritiske funksjoner i Norge. Før den hel-elektriske ventilen kan brukes til dette formålet, kreves det dokumentasjon som viser at den oppfyller sikkerhetskravene.

Digitalisering og automatisering fortsetter å drive fremkomsten av komplekse systemer. I denne sammenhengen har det dukket opp nye teknikker, som simuleringstyrte metoder, for automatisert testing og verifisering av komplekse systemer. Disse metodene bruker simuleringer av en digital modell for å teste ulike scenarioer.

For å takle disse utfordringene undersøker denne avhandlingen to typer simuleringstyrt testingmetoder, hvorav den ene er implementert på en digital tvilling. Den digitale tvillingen av den hel-elektriske ventilen inkluderer blant annet et lithium-ion-batteri og et batteristyringssystem. I parallell med denne masteroppgaven har Ludvig Björklund implementert en enkel digital tvilling. Forfatteren av denne avhandlingen har eksportert og implementert den digitale tvillingen i Python for å skape et allsidig testmiljø.

I Python har det blitt implementert optimeringsstyrt falsifisering ved hjelp av signaltemporal logikk og bayesisk optimering. Dette har blitt anvendt på flere feilscenarier på den digitale tvillingen. Batteristyringssystemet har ansvaret for å observere og håndtere feil i batteriet. Målet var å bruke optimeringsstyrt falsifisering for å finne dårlige parametere som kan injisere feil i batteriet og observere håndteringen av batteristyringssystemet. Én test-case er implementert i et grafisk brukergrensesnitt for å demonstrere testmetoden for testingeniører.

Metoden klarte å identifisere eksempler på falsifisering og injiserte feil som batteristyringssystemet ikke klarte å håndtere.

Table of Contents

List of Tables

List of Figures

List of Code Listings

Abbreviations

Abbreviation	Description
APS	Abandon Platform Shutdown
ASC	Actuation System Control
BMS	Battery Management System
CAD	Computer-Aided Design
DCV	Directional Control Valve
DHSV	Down Hole Safety Valve
DT	Digital Twin
DTI	Digital Twin Instance
DTP	Digital Twin Prototype
E/E/PE	Electrical/Electronic/Programmable Electronic
EPU	Electric Power Unit
ESD	Emergency Shutdown
EUC	Equipment Under Control
FMU	Functional Mock-up Unit
FMEA	Failure Modes and Effects Analysis
FMI	Functional Mock-up Interface
GUI	Graphical User Interface
HDL	Hardware Description Language
HSE	Health, Safety and Environment
IEC	International Electrotechnical Commission
IoT	Internet of Things
ISC	Internal Short Circuit
KISS	Keep It Stupid Simple
Li-ion	Lithium-ion
LP	Low Pressure
LTL	Liner Temporal Logic
MTL	Metric Temporal Logic
Ni-Cd	Nickel-Cadmium
PMW	Production Master Valve

PFD	Probability of Failure on Demand
PSA	Petroleum Safety Authority
PSD	Process Shutdown
PST	Partial Stroke Test
SIL	Safety Integrity Level
SIF	Safety Instrumented Function
SOF	State of Function
SOH	State of Health
SOC	State of Charge
SOV	Solenoid-Operated Valve
STL	Signal Temporal Logic
UCB	Upper Confidence Bound

1

Introduction

1.1 Background

SUBPRO is a research-based innovation center that focuses on subsea production and processing. It is funded by the Research Council of Norway, nine industrial partners, and NTNU [15]. Established in 2015, the center has a planned duration of eight years. As stated in the SUBPRO annual report [15], their objectives include reducing the cost and complexity of subsea field developments, facilitating the development of new and challenging oil and gas fields, increasing production, and extending the life of existing fields, minimizing the environmental impact of subsea field developments, and ensuring safety levels are maintained.

SUBPRO has several ongoing research projects, including developing a digital twin (DT) of an all-electric safety valve. The goal is to use the DT to test and demonstrate that the high software-dependent all-electric valve can reach a safe state in various scenarios and are reliable enough according to governing standards compared to the state-of-the-art hydraulic valve design [15]. The research is in collaboration with Aalen University. The Ph.D. candidate, Ludvig Björklund, is working on developing the DT. He is also the co-supervisor of this thesis.

Improved sensor technology and increasing computational power have opened up for the use of more digital solutions offshore [16]. However, together with this, the scale and complexity of systems have grown. Reaching the safe state is therefore reliant on the execution of software-embedded functionality, increasing the complexity of demonstrating

the safety capabilities of the system. Testing and validating these complex systems present challenges as the composition of parameters can become infinite. The traditional approach of manually selecting test cases is time-consuming and may not provide comprehensive coverage for these new systems.

Today, automatic simulation-based methods are emerging [9]. These methods use simulations of a DT of the system as test coverage to demonstrate reliable behavior [7]. Various approaches to simulation-based testing and verification are available. This thesis will look into two of them.

1.2 Problem description

The objective of this master thesis is to implement fault scenarios into a DT for simulation-based testing of the control system as part of the safety demonstration process and to automate the process of finding issues in the control system. This master focuses on the battery and battery management system (BMS) part of the DT.

The task of this master thesis is described by the bullet points below:

- Identify important goals and requirements for a safety demonstration process of a control logic of a safety-critical system.
- Familiarize with current state-of-the-art methods on simulation-based testing with a focus on STL and Lyapunov stability analysis.
- Implement the STL for fault injection on a simple model related to the all-electric valve system and operation.
- Develop a simple user interface to run and visualize the results of the experiments.
- Discuss the results, including the pros and cons of STL in achieving goals of safety demonstration. Indicate possible opportunities with Lyapunov.

1.3 Approach

An overview of this master thesis approach is shown in Figure 1.1. The work began with a literature review on safety demonstrations for safety-critical systems. This was followed by exploring existing and emerging simulation-based testing and verification techniques, specifically focusing on STL and Lyapunov. Additionally, related work was examined, and suitable software for implementing these techniques in Python was identified.

During the search for information on Lyapunov, contact was made with Professor James Kapinsky via email. Although a response was received, the conversation did not progress further as the decision was made not to pursue the topic of Lyapunov.

Furthermore, an investigation was conducted on Li-ion batteries and BMS to gain insights into common faults and how the BMS can effectively handle them.

Research was conducted to determine the appropriate tool or framework to implement the graphical user interface (GUI).

The subsequent phase involved the implementation process, which was carried out iteratively in collaboration with the co-supervisor, Björklund. Weekly meetings were held with Björklund, who developed the DT. The workflow typically involved Björklund creating the DT in Matlab, followed by my implementation and testing in Python, and then jointly evaluating the results and discussing test cases. Based on the evaluation, Björklund would modify the DT as necessary for enhancing the results of the testing method.

Lastly, the testing method was implemented in a GUI.

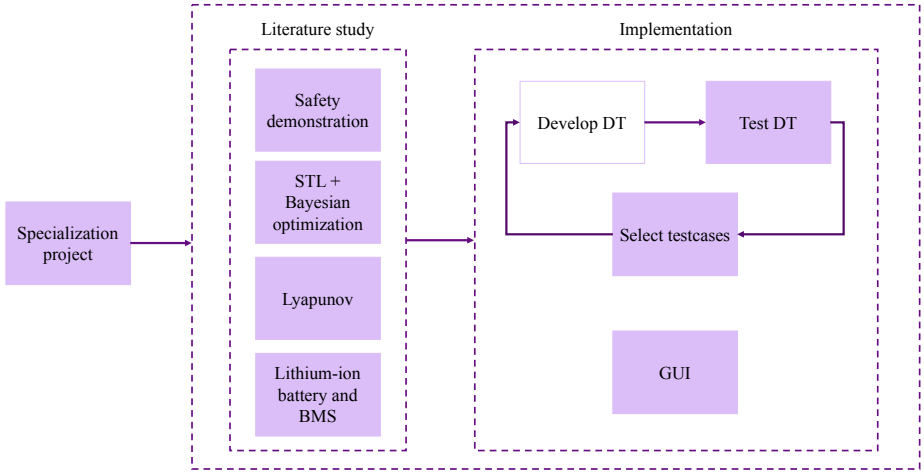


Figure 1.1: Overview of the approach of this master thesis

In addition, the completion of the specialization project and the corresponding specialization courses during the previous semester provided a solid foundation. The project's main objective was to gain knowledge about all-electric valves and their differentiation from conventional hydraulic valves, as well as to understand DT and the concepts of safety

demonstration.

The subsea safety valves, DT, and safety demonstration chapters in this master's thesis have been derived from the specialization project, incorporating modifications.

1.4 Assumptions and delimitations

This thesis is specifically focused on examining the battery and BMS of the DT. However, it is important to note that the ultimate goal is to include all components of the all-electric valve in the DT, which may generate additional relevant test cases, such as evaluating the behavior while simultaneously interacting with several different controller objectives.

It is worth highlighting that the DT was developed in collaboration and in parallel with this master's thesis and was not pre-existing. The DT was continuously adjusted and modified throughout the process, which may introduce certain limitations.

As stated in the problem description, this thesis concentrates on testing and verification methods using STL and Lyapunov analysis, although many other methods exist.

1.5 Structure of the report

The structure of this report is based on a template provided by NTNU [17].

The introduction begins with the thesis background and research questions. It also outlines the objectives and limitations of the thesis.

In chapter 2, an overview of the all-electric actuation system and the current state-of-the-art in hydraulic design is provided. It also covers the relevant guidelines and standards necessary for valve development.

The next section, chapter 3, delves into the concept of Industry 4.0 and the development of DTs within this context. It also explores safety demonstrations for novel technologies and highlights the application of DTs in this process.

Moving on to chapter 4, various emerging simulation-based testing and verification techniques are examined. This includes a detailed exploration of optimization-guided falsification and simulation-guided Lyapunov analysis.

In chapter 5, a deeper investigation into lithium-ion batteries and the BMS is provided. It includes a discussion of Li-ion battery failures and offers insights into the Matlab model used and the implemented failure modes.

The implementation details are outlined in chapter 6.

The obtained results are presented in chapter 7.

Finally, the report concludes with a comprehensive discussion and conclusion section.

2

Safety valves in subsea Xmas trees

This chapter introduces the all-electric valve. The content primarily originates from the specialization project, with the addition of information regarding previously utilized all-electric valves.

A subsea xmas tree is a crucial component in offshore oil and gas production systems. It is an assembly of valves, control systems, and other equipment installed on the seabed to control the flow of hydrocarbons from the ground to the platform. The valves are called downhole safety valve (DHSV), production master valve (PMV), and production wing valve (PWV). The valves are normally open. Their main function is to maintain safety on the platform by closing and stopping the flow of hydrocarbons during emergencies. Safety is defined as freedom from risk that is not tolerable [18]. Today the electro-hydraulic Xmas trees are the standard architecture worldwide [2]. Due to the increased digitalization and electrification of subsea equipment, there is a proposal to make the valves all-electric. The new design will reduce cost, be more environmentally friendly and be safer [1]. The DHSV will still be hydraulic, only PMV and PWV are proposed to be changed to electric.

2.1 Electro-hydraulic Xmas trees

Figure 2.1 shows a simplified design of an electro-hydraulic Xmas tree. The figure is taken from the book published by DNV [2]. A Xmas tree can have other designs and consist of multiple types of valves, but they are not included in this figure because we want to address the safety valves: DHSV, PMV, and PVW.

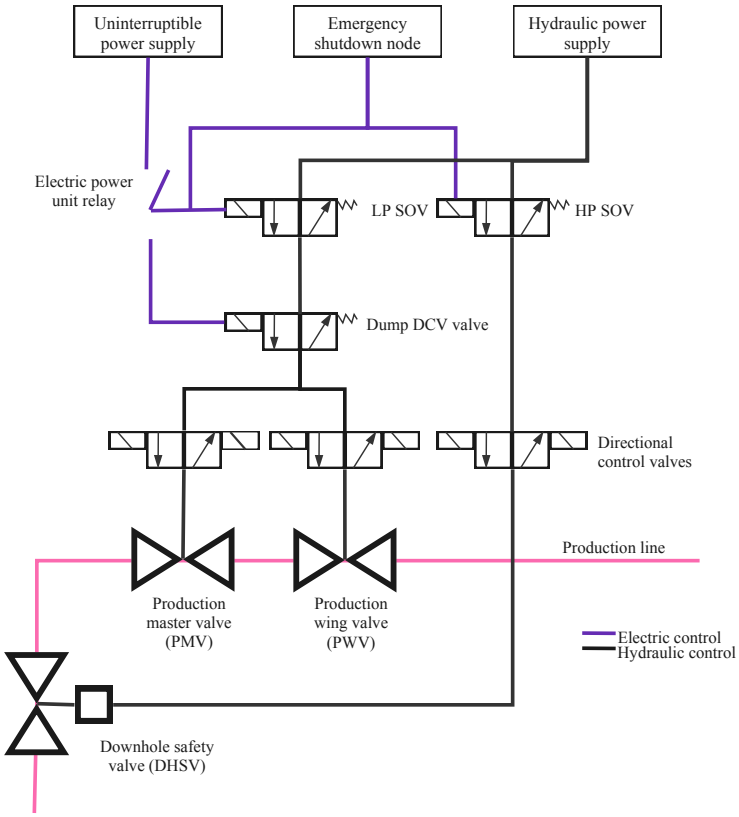


Figure 2.1: Architecture of electro-hydraulic Xmas tree. Adapted from [2]

The electro-hydraulic system described in the book published by DNV [2] works as follows: From below, the hydrocarbons first pass the DHSV, then the PMV and the PWV. The DHSV, PMV, and PWV are hydraulic spring-return valves. They open and stay open by the pressure from the hydraulic fluid. The valves close when the hydraulic pressure is withdrawn. From the topside, the hydraulic pressure goes through the solenoid-operated valves(SOV), through the umbilical and the directional control valves(DCV).

During normal operation, hydraulic pressure is available to the subsea Xmas tree, so the valves stay open. The hydro-electro valve uses the de-energize-to-close principle, which means it does not need energy to stay closed. Normally and during a low-level emergency shutdown (ESD), the valves can be closed by using the DCVs.

During a high-level ESD and abandon platform shutdown(APS), the ESD node is used to de-energize the low-pressure (LP) SOVs and the electric power unit (EPU) relay. Fol-

lowing, the hydraulic pressure supply and the electric power supply disconnect. This de-energizes the Dump DCV valve, and the hydraulic fluid is drawn back, so the valves close. Lastly, the high-pressure (HP) SOV is de-energized. With this method, the process goes slower. Doing so makes the DHSV close last.

2.2 The all-electric Xmas tree

In 1999, TechnipFMC began exploring an alternative to the hydraulic-electric valve design due to environmental concerns and cost considerations. In some parts of the world at that time, hydraulic-fluid costs accounted for as much as 4-8 percent of OPEX expenditure [19]. In addition, direct hydraulic technology yields slow response time, demands many lines, and is not scalable [19]. The concept of an all-electric valve was introduced, but the main challenge was to provide sufficient power to the valve. In contrast to the hydraulic valve, the all-electric operates on an energize-to-close principle. Meaning that if power is lost, the valve will not go to a safe state. As a result, maintaining a continuous power supply becomes crucial for ensuring overall operational safety. One approach involved a valve that derived power from the topside. However, due to long distances, a battery was ultimately chosen as the solution. Over the years, various battery technologies have been developed, ranging from lead-acid batteries in the early days to today's lithium-ion (Li-ion) batteries.

Since 2006 TechnipFMC has been utilizing electric actuators for all non-safety-critical production valves and chokes [19]. The next step was to evolve the technology into safety-critical systems, which posed a more challenging task as the valves had to meet the governing requirements. In 2005, the first actuators featuring a fail-safe close functionality were deployed as a pilot system in the Statoil Norne field, but this system was not certified [19]. Nonetheless, it served as the foundation for the design of later certified solutions.

Figure 2.2 shows a generic architecture of an all-electric Xmas tree. The figure is a simplified version of the figure in Mahler et al. [3]. The original contains more information than is necessary for the safety perspective and is therefore not relevant to this thesis. Only the PMV and the PWV are made electric. The DHSV will still be hydraulic and is, therefore, not in the figure. The box safety logic, battery and BMS, and switch module are for safety's sake [3].

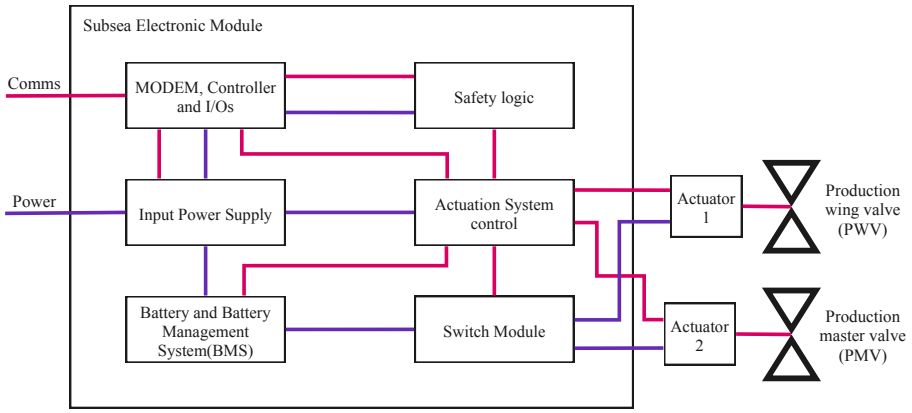


Figure 2.2: Architecture of all-electric Xmas tree. Adapted from [3]

In the all-electric Xmas tree, the power from the topside comes through the input power supply. In addition, a battery is needed for an uninterruptible power supply. If the power from the topside is lost, then the battery will ensure power to the valves at all times [1]. Along with the battery, a BMS is needed. The BMS monitors and controls the condition of the battery. The main function of the BMS is to detect and prevent failures of the battery. The battery and BMS will be further explored in chapter 5

The Switch Module on the Xmas tree is there to handle power to the actuators. To connect or disconnect each actuator separately [1]. The switch unit is important as it prevents a common cause failure, i.e., ”a failure that is the result of one or more events, causes concurrent failures of two or more separate channels in a multiple channel system, leading to system failure” [20]. E.g. a short circuit between the actuator and the battery. In this case, the working actuator must be closed [21].

The safety logic block observes the system status and commands the valves to their fail-safe position when necessary, e.g. in the event of power loss [1].

The Actuation System Control (ASC) communicates with the topside controller via a safe communication link. The ASC also provides communications to the actuators, and it monitors all of the safety-relevant system components.

Potential failure modes

A failure mode is a description of how a function can fail. Failure modes can divide into *safe* and *dangerous* and then into *detected* or *undetected*, shown in Figure 2.3. The strength of the red indicates how critical the failure is. A dangerous failure prevents the system from executing the safety function when required [20]. Safe failures are failures that do not prevent the safety function of a system from being executed and are, therefore, not considered critical. Safe failures can still cause complications and inconvenience but do not directly threaten the safety of users and surroundings. An undetected failure is a failure that we cannot detect with traditional diagnostics. Detected dangerous failures are less critical than undetected dangerous. This is because other measures can be taken to maintain safety when the failure is known.

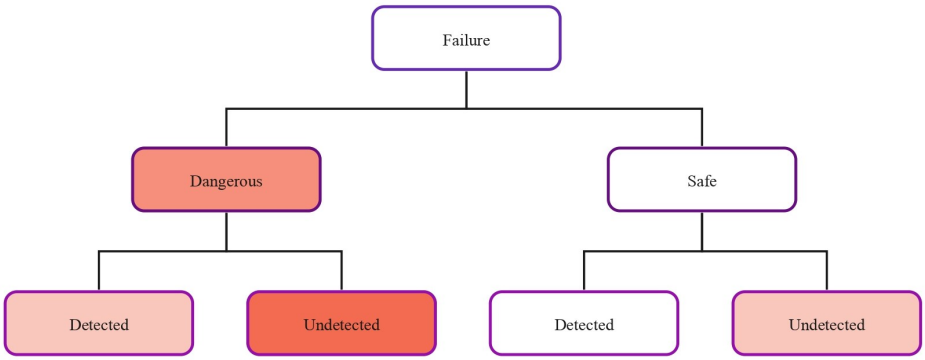


Figure 2.3: Failure mode categories

The paper by Mahler et al. [1] has performed a failure modes and effects analysis (FMEA) of the all-electric Xmas tree. An FMEA is used to find the causes of potential failures and evaluate their consequences. As a result, four main failure modes of the valve for isolating the well are presented:

Failure mode	Failure type
Fail to close	Dangerous
Fail to open	Safe
Internal leakage	Dangerous
External leakage	Dangerous

Table 2.1: Failure modes of valve for isolating the well. Adapted from [1].

In the all-electric Xmas tree, failure of the individual components can lead to the failure mode of the safety valves. A few examples are:

- The motor provides a force to the actuator to either close or open the valve. If the torque is too big, the valve shaft can be damaged, which then can lead to leakage.
- A short circuit in an actuator can lead to failure mode *Fail to close* or *Fail to open*. The risk of this is mitigated using redundancy. If one actuator fails, the switch module disconnects the failed actuator and provides power to the other remaining one working.
- Topside and subsea communicate with a digital communication link. However, it is important to note that this digital communication link is susceptible to freezing, which in turn can result in system failures [2].
- If the battery fails, the control of the valves is lost. The BMS is included to prevent failures of the battery, but if the BMS fails, the safety function of the system will fail. Failures of the battery will be introduced more thoroughly in chapter 5.

2.3 Governing standards and guidelines

For developing novel technology to be used in subsea oil and gas settings, the following standards are useful, IEC 6108, NOG 070, and NORSOK S-001. This section is adapted from the specialization project.

IEC 61508: Ensuring safety and reliability in industrial systems

The International Electrotechnical Commission (IEC) is an international organization that develops and publishes standards for electrical, electronic, and related technologies. It is a leading global organization in this field. Their objective is to promote international cooperation concerning standardization in electrical and electronic fields [20]. IEC 61508 is a standard for functional safety achieved by safety-related systems that are primarily implemented in electrical and/or electronic and/or programmable electronic (E/E/PE) technologies [20].

IEC 61508-0 describes functional safety as part of the overall safety relating to the equipment under control (EUC) and the EUC control system that depends on the correct functioning of the E/E/PE safety-related systems and other risk reduction measures [20].

The functional safety is executed by a safety instrumented function (SIF). The SIF acts as a safety barrier, designed to intervene when a process or system exceeds certain safety

limits or when a hazardous condition is detected [20]. The IEC 61508 gives advice on how to maintain functional safety. It is a crucial tool to utilize when dealing with functional safety and especially when introducing novel technology.

In the context of safety demonstration, IEC 61508 [20] plays an important role by providing a set of requirements and guidelines that systems must meet to be considered safe and reliable. The standard defines the different safety integrity levels (SIL) that systems must achieve. SIL is a quantified measure of the risk reduction level achieved or intended by a SIF [20].

NOG070: Application of IEC 61508 and IEC 61511 in the Norwegian petroleum industry

The NOG070 is a guideline on the use of IEC 61508 and IEC 61511 for the sake of the Norwegian petroleum industry, where IEC 61511 is a standard on the functional safety of safety instrumented systems for the process industry sector. The main purpose of the guideline is to standardize and simplify the use of IEC 61508 and IEC 61511 [22]. While the IEC 61508 and IEC 61511 gives a risk-based approach to identify the performance requirements, NOG070 proposes a set of predefined performance requirements for functions that are already identified as necessary by national and international standards used by the Norwegian petroleum sector [22].

The guideline provides the probability of failure on demand (PFD) and, based on this, the SIL of different SIFs. The PFD is the probability of the system not working when we want it to execute the safety function [20]. The PFD is calculated in NOG070 by using data from experience. The SIL tells us how much we can rely on the system to execute its safety function when needed [20]. Table A.13.9 in NOG070 summarises the SIL requirements for isolation of the subsea well [22]. The PMV and PWV make the secondary isolation barrier of the production line and are given SIL 2.

NORSOK S-001: Technical safety

NORSOK S-001 is a national standard for technical safety [4]. The purpose of the standard is to reduce the time and cost of the development and operation of petroleum installations on the Norwegian continental shelf.

Included in this standard are hazard identification, risk analysis and evaluation, risk treatment, and performance requirements. The standard describes the execution of ESD on the platforms and when the different safety valves will close and are therefore helpful in this article. The purpose of the ESD system is to prevent the escalation of abnormal conditions

into a major hazardous event and to limit the extent and duration of any such events that do occur [4]. The process safety system is part of the ESD system, and it is here the safety valves are involved.

ESD Hierarchy
The ESD system is needed in multiple scenarios, from shutting down only the production system to shutting down the whole platform. The shutdowns are triggered by either push of a manual button or the detection of a dangerous event, such as a fire in a hazardous area.

The ESD functions are put in a hierarchy shown in Figure 2.4 [4]. The ESD levels are divided into three where the APS is the highest, ESD 1 is in the middle, and lastly, ESD 2. The process shutdown (PSD) is initiated by ESD 2. The hierarchy works so that a higher ESD level should initiate a lower one, and a signal on a certain level should never initiate shutdowns or actions on higher levels [22].

An example of such an action is shutting safety valves. ESD valves shall isolate and sectionalize the process segments in a fast and reliable manner to reduce the number of released hydrocarbons in the event of a leak.

The safety valves are closed by the action ”activation of the ESDV incl. MV and WV”, which is highlighted in the ESD 2 box. The highlighted signals trigger the action:

- Manual push button
- Confirmed gas in a non-hazardous area
- Knock out drum LAHH (ESD)
- Confirmed gas in hazardous area
- Manual depressurization
- Confirmed fire in hazardous area

The action is also initiated in the event of APS or ESD 1.

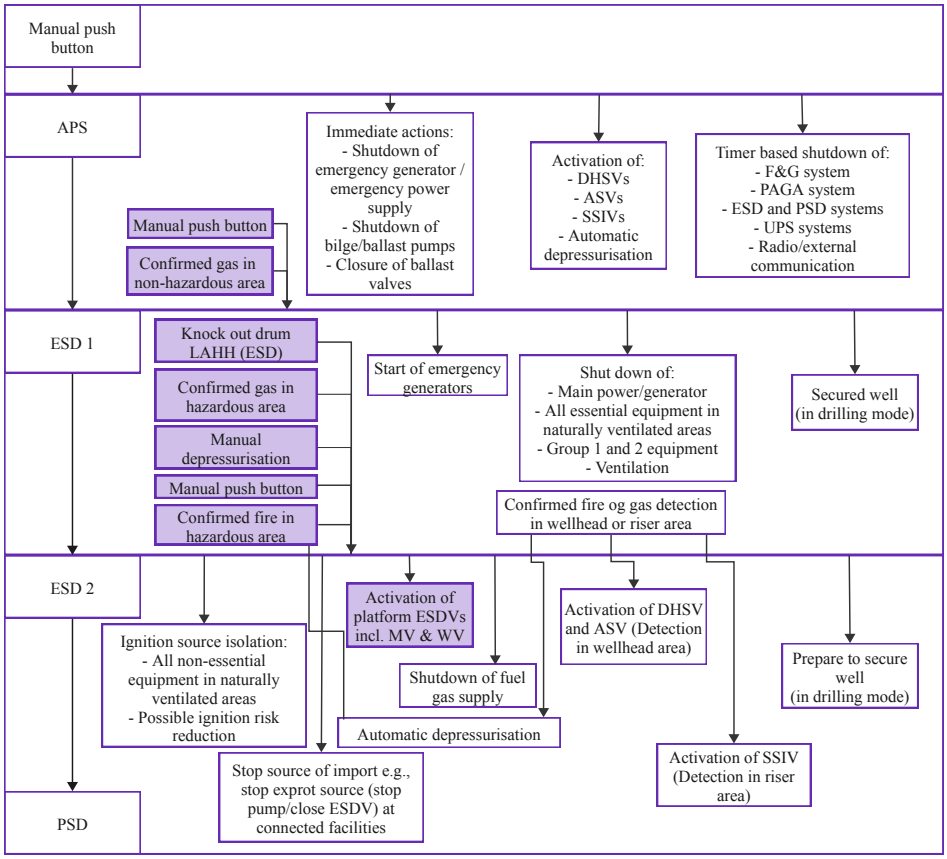


Figure 2.4: ESD hierarchy. Adapted from [4]

3

Digital twin and safety demonstration

This chapter primarily builds upon the chapter about DTs from the specialization project. However, the section on safety demonstration has been extensively revised.

3.1 Digital twin and industry 4.0

The fourth industrial revolution, known as Industry 4.0, refers to the ongoing trend of automation and data exchange in manufacturing technologies, including the use of advanced technologies such as artificial intelligence, the Internet of Things (IoT), and machine learning [16]. This trend is leading to the development of more complex, software-dependent systems that are capable of greater levels of automation and intelligence.

As part of the digitalization and the industry 4.0 initiatives, the development of DTs has increased. DTs can be used to simulate and analyze the performance of a manufacturing process or product in real time, allowing manufacturers to identify and address potential issues before they arise. This can help improve efficiency, reduce downtime, and, ultimately, enhance the overall performance of the manufacturing process.

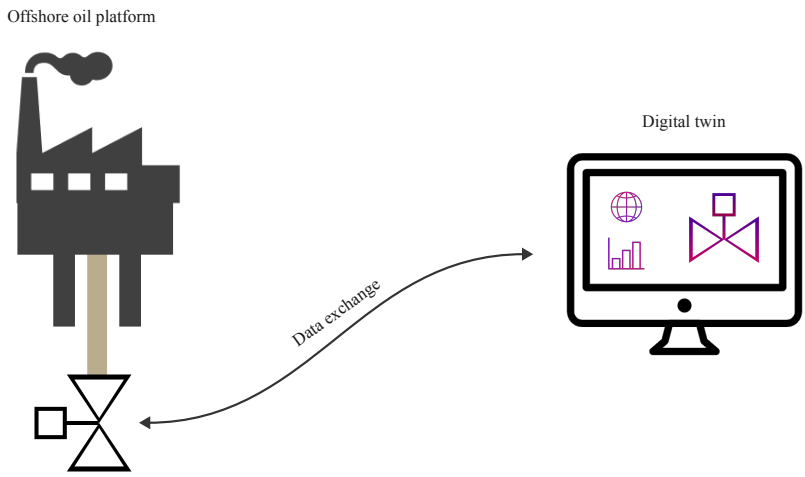


Figure 3.1: The DT is on a computer while the physical valve is placed on the platform. They are linked through data exchange.

Multiple definitions of DTs exist. A DT, defined by DNV [23]: ”is a dynamic virtual representation of a physical object or system across its lifecycle, using real-time data to enable understanding, learning, and reasoning.” Figure 3.1 shows how the DT is on the computer while the real system is offshore.

The book by Vickers et al. [6] addresses how we earlier only could have any knowledge about a system or object by being right next to it and looking at it. Over the years, engineers have found ways to have more and more knowledge about systems, both about their design and functions. Firstly making national standards for products made it easier to mass produce and repair. In the last half of the twentieth century, the transition toward digital information became prevalent. Some sparse computer-aided design (CAD) illustrations were the beginning of the DT [6]. These illustrations were just a static representation of the object. Today we can make detailed simulations about the system, not only before it is set up but also during its operation time.

3.2 Level of integration

Kritzinger et al. [5] distinguishes the DT into three forms based on the level of data integration between the physical asset and the DT. Figure 3.2 visualizes the data flow between the three DTs and the physical asset. The figure is inspired by the figures in the

article by Kritzinger.

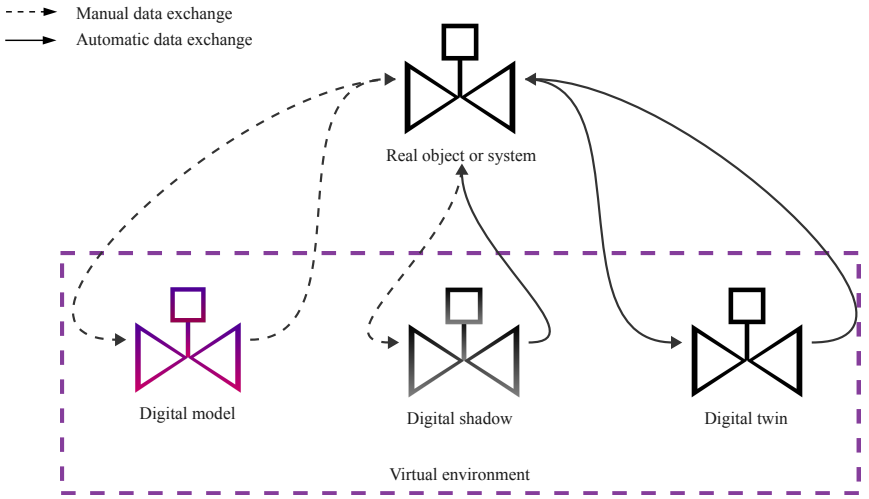


Figure 3.2: Different types of DTs based on level of integration. Adapted from [5]

The three forms of DTs described in Kritzinger et al. are:

The first form of DT is a simple digital replica of a physical asset. It is called a *digital model* and typically only includes basic information about the asset, such as its shape, size, and location. It may not be connected to any real-time data or information about the asset's performance or condition.

The second form of DT is called a *digital shadow*. It involves some level of data integration between the physical asset and the DT. In this case, the DT may be connected to sensors or other sources of real-time data about the asset's performance or condition. This allows the DT to provide more detailed information about the asset and its behavior.

The third form of DT is a fully integrated *digital twin*, in which the DT is closely connected to the physical asset and is updated in real-time with data from sensors and other sources. This type of DT can provide a highly detailed and accurate representation of the asset and its behavior and can be used for a variety of purposes, including predictive maintenance, performance optimization, and scenario planning.

Vickers et al. [6] propose a different way of classifying DTs, using the terms *digital twin prototype* (DTP) and *digital twin instance* (DTI). In their approach, a DTP is a virtual model representing a physical object and the requirements to produce a physical version

that duplicates the virtual version. In other words, a DTP is a virtual representation of an object used to design and develop the physical version of that object. On the other hand, a DTI is a DT that is linked to a specific physical product. This means that the DTI represents a specific instance of a physical product, and it remains linked to that product throughout the life of the product. The DTI provides real-time data and information about the physical product. It can be used for various purposes, such as predictive maintenance, performance optimization, and scenario planning. Overall, both approaches to classifying DTs are useful in different contexts. Depending on the specific use case and the goals of the DT, one approach may be more useful than the other.

3.3 Digital twin through the product lifecycle

Systems are not always perfect on the first try and often require trial and error to develop and improve [6]. This can be costly, inefficient, and even dangerous if not managed properly. Additionally, systems are not static and tend to evolve over time as they go through their product life cycle. The product lifecycle presented by Vickers et al. [6] contains four main stages presented in Figure 3.3.

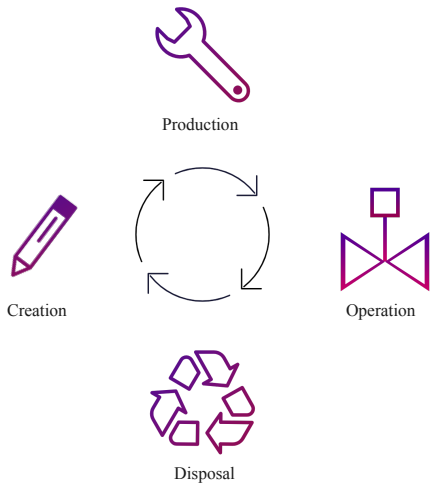


Figure 3.3: Product lifecycle. Adapted from [6]

During the creation and design phase, the engineers identify the concept and the overall scope of the system. To support this process, standards and guidelines, such as IEC 61508 and NOG070, provide frameworks and guidance for designing and developing robust and

reliable systems. These standards and guidelines help ensure that the system meets specific performance and safety criteria and can make designing and developing more robust systems easier. The next step is to create a DTP to represent the physical asset.

Using a DTP to simulate the behavior of an asset can provide valuable insights and information about the asset [24]. Testing the all-electric valve in the real world is more costly because it requires physical equipment and may be limited to testing only certain parameters, such as a single valve size. In contrast, a DTP allows for testing a wider range of parameters and settings, providing greater flexibility and insight.

In addition to testing correct functionality, injecting parameters to initiate failure modes in the system's DT can give valuable insight. This process involves simulating the system's behavior under various conditions and inputs and applying specific parameters known to cause the system to fail. By simulating these failure modes and observing the system's behavior, developers can evaluate the system's reliability and identify potential weaknesses or vulnerabilities.

The next phase is the production phase. In this phase, the DTP is used to produce a physical version of the asset, which is then linked to the DTI. The design of a system is not always physically possible. Therefore the design usually changes during the production phase, and because of these changes, undesirable system behaviors can arise [6]. By using a DTI during production, changes can first be implemented and tested on the DTI before implementing it on the physical system.

The third phase is the operation phase. In this phase, the DTI is used to provide real-time data and information about the physical asset and to support various operations and activities related to the asset. It is first here it is called a DT by Kritzinger et al. [5] definition. Changes during the operation phase can first be implemented and tested on the DTI before being implemented on the physical asset.

The last phase is the disposal phase. In this phase, the physical asset reaches the end of its useful life and is decommissioned or retired. The DTI is also retired in this phase. Data from the DTI can be stored and analyzed to make better systems in the future systems.

3.4 Safety demonstration

Safety demonstration is defined as documentation, based on evidence and structured reasoning, that adequate safety criteria are specified and met [2]. This is an essential step in developing and deploying systems, as it helps to ensure that the systems are safe and reliable.

Demonstrating safety provides evidence that the system will not pose a risk to people, property, or the environment. It is an important step when developing novel technology, especially for safety-critical systems such as the all-electric Xmas tree.

3.4.1 Safety 4.0

Due to the trend of increased advanced technologies in manufacturing, such as artificial intelligence and the IoT, more complex and software-dependent systems are developed.

The phrase ”Keep it stupid simple” (KISS) is often used in engineering and safety to emphasize the importance of simplicity in design. The idea behind KISS is that simple, straightforward designs are often the most effective and the easiest to understand, maintain, and operate [2]. The emergence of software-dependent systems brings potential in terms of cost-efficiency, safety, and environmental friendliness. However, the inherent complexity of these systems poses a challenge that can comprise these advantages. The increasing complexity sparked the inception of the Safety 4.0 project.

Safety 4.0 is a three-and-a-half-year project funded by the Research Council of Norway and the project participants. The project has developed a safety demonstration framework to enable and accelerate the safe adoption of new subsea solutions [2]. The framework includes how to deal with diverse types of failures, increasing complexity, and uncertain assumptions. Like the Norwegian regulations, Safety 4.0 refers to international safety standards, e.g. IEC 61508.

The project utilizes the all-electric Xmas tree in the development of the framework with the help of pre-approved standards and guidelines recommended by Norwegian Regulations.

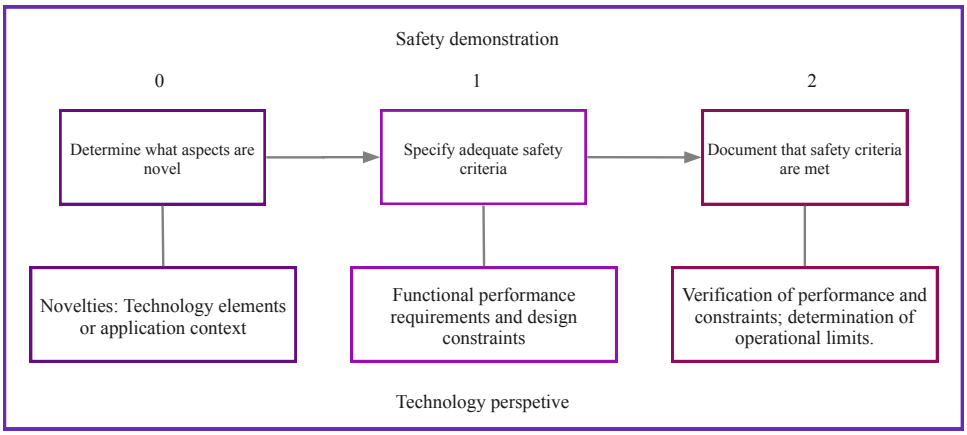


Figure 3.4: Safety demonstration in technology perspective. Adapted from [2]

Safety demonstration according to safety 4.0 [2] is based on three steps which are shown in Figure 3.4:

0. Determine novel aspects of a solution.
1. Specify adequate safety criteria.
2. Provide arguments and evidence that these are met.

Combined with the three steps, the safety demonstration process divides into three subprocesses, each with a different focus and perspective. Dividing makes it easier to identify where the novelty occurs and where to put in the effort. The subprocesses are the *activity* perspective, the *strategy* perspective, and the *technology* perspective. The technology perspective is most relevant for the battery and BMS part of the all-electric actuation system.

The technology perspective asks, ”Is the technology safe?”. It focuses on the technical aspects of the system, including the hardware and software components used. The technical equipment must pass the performance requirements and operational constraints needed for the activity to be safe and reliable [2].

DNV-RP-A203 is a recommended practice for technology qualification [2]. According to DNV [25], technology qualification is the systematic process of obtaining evidence that demonstrates the functionality of a technology within predefined operational limits while

maintaining an acceptable level of confidence. The qualification process involves three main steps:

- Establishing the qualification basis
- Assessing technology risk
- Planning and executing qualification activities to verify performance.

The first is about establishing a set of criteria that will guide all qualification activities and decisions. Here the standards NORSOK S-001 and IEC 61508 can be of relevant use.

Subsequently, the technology threat assessment needs to be conducted to identify significant uncertainties associated with the innovative technology and identify potential hazards associated with the technology. It is noteworthy that DNV-RP-A203 highlights the importance of establishing the probability of failure concerning individual failure modes. Nevertheless, not all failure modes can be accurately characterized by failure probabilities. E.g. systematic failures that occur under specific conditions.

Finally, the qualification activities should be planned and executed. This step includes the qualification plan, the plan's execution, and the performance assessment. The goal of this step is to provide a convincing argument that the technology is safe and dependable. To achieve this goal, it is essential to develop a detailed plan that encompasses the safety requirements, the hardware and software components of the system, and the system's design choices.

The qualification activities involve testing. A technology must demonstrate safe performance across a wide range of conditions based on multiple parameters. However, it is not feasible to test every possible combination of parameters. Therefore, the intelligent selection of test scenarios becomes essential in obtaining reliable safety conclusions while minimizing resource costs.

DNV-RP-A203 also mentions virtual testing. Virtual testing, conducted through computer simulations, presents an alternative approach that enables the evaluation of scenarios that may be hazardous, time-consuming, challenging, or expensive to replicate in real-life settings. An example of this is the utilization of a digital twin. Further exploration of emerging techniques for more efficient testing and verification can be found in chapter 4.

3.4.2 Digital twin for safety demonstration

The SUBPRO project aims to develop and implement a DT of an all-electric Xmas tree and to use this DT in the safety demonstration of the system. There are many benefits of

this.

DTs provide a safe and controlled environment for testing and evaluating the safety of a system [26]. By simulating the system's behavior on a computer, engineers can test and evaluate the system's safety without exposing it to real-world hazards. DTs allow testing to be conducted in diverse ways to assess the performance and resilience of a system. One approach involves intentionally injecting failures to evaluate the system's ability to withstand and recover from such faults. While demonstrating correct behavior is crucial for safety, it is equally important to test how the system handles incorrect behavior or unexpected events. This approach would not be as ideal on a physical system because of cost and hazards.

It is efficient and cost-effective to use DTs as they provide the flexibility to alter system parameters and settings. This allows for comprehensive testing without the need for multiple physical prototypes. This can save time and money and allow engineers to test and evaluate the system's safety more quickly and efficiently.

Another benefit is that a DT can simulate how the system will be after many years of operation. Over time, certain components may degrade or become faulty. However, the rest of the system can still function sufficiently.

DTs allow for real-time monitoring and evaluation of the system's safety. Because DTs are digital models, they can be updated and modified in real time based on new information and data.

A challenge of using DTs for safety demonstration is ensuring that the DT accurately represents the behavior of the physical system. This can be difficult, as not all test scenarios may be realistic enough on a computer, and the DT may not capture all the nuances and complexities of the physical system. DNV has published a recommended practice for qualification and assurance of DTs [23]. In order to ensure that the DTs behavior is realistic, the DT should be verified and validated.

Verification is the process of checking whether the simulation model is correctly implemented and accurately represents the conceptual model it is intended to simulate [27]. This involves testing the model against a set of known solutions or benchmarks to ensure that the model produces the expected output. Verification aims to ensure that the simulation model is free from coding errors, algorithmic mistakes, and numerical instabilities.

Validation, on the other hand, is the process of determining whether the simulation model accurately represents the real-world system it is intended to simulate [27]. Validation involves comparing the simulation output with empirical data or other sources of informa-

tion to assess the degree to which the simulation model captures the essential features of the real system. Validation aims to ensure that the simulation model is fit for its intended purpose and can be used with confidence to make predictions or inform decision-making.

4

Simulation-based testing and verification

This chapter focuses on emerging techniques for testing and verifying cyber-physical systems. Initially, various methods will be presented, followed by the specific methods chosen in this thesis: optimization-guided falsification and simulation-based Lyapunov analysis.

4.1 Existing and emerging methods

Testing and verification are important parts of developing novel technology. This part helps the developers increase their confidence that the safety and performance requirements are met. Testing and verification can be costly and lengthy, occupying a significant portion of the developmental process. Due to the increase in functionalities being reliant on software, the sheer scale of the embedded software today makes formal verification an intractable issue. To overcome this, new techniques are emerging. Figure 4.1 displays some testing and verification techniques for cyber-physical systems today. The figure is taken from [7], which is inspired by the figure in [9]. The techniques are informally categorized based on scalability and exhaustiveness. Their scalability refers to their ability to handle large or complex systems, while their exhaustiveness refers to how thoroughly an approach considers all feasible behaviors of a model.

On the bottom right are what can be categorized as formal methods. They have high exhaustiveness but little scalability due to their high computational cost. Model checking and

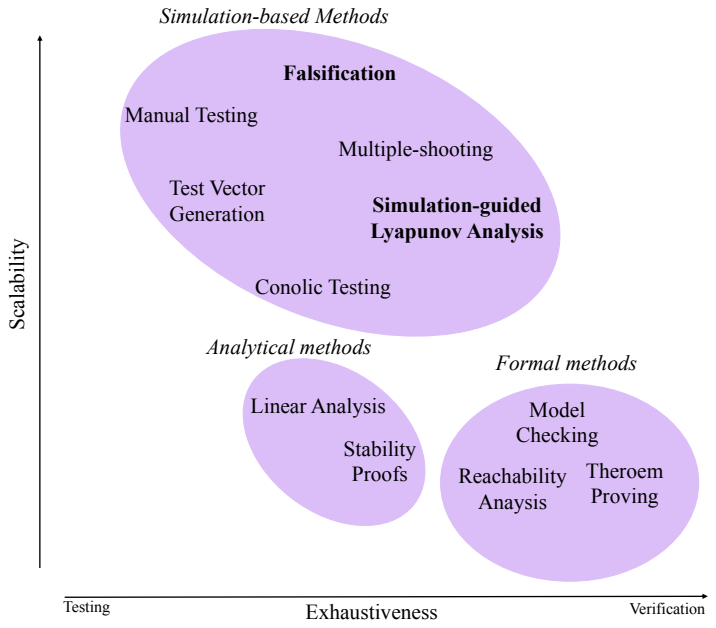


Figure 4.1: Overview of testing and verification methods. Adapted from [7]

theorem provers are two well-established verification methods for hardware and software development [28]. The technique of model checking involves constructing a finite model of a system and verifying whether a particular property is satisfied within that model [29]. Model checking is an efficient and automatic process that determines whether the system is correct or it will output a counterexample which can be useful in debugging.

Theorem proving is a method that involves representing both the system and its desired properties as formulas within a mathematical logic [29]. Theorem proving construct mathematical proofs to demonstrate that a model satisfies a given property. Theorem proving can be an automated process or an interactive process that requires inputs from a user.

Reachability analysis is a numerical technique that provides a conservative estimation of the potential behaviors that a closed-loop system model can demonstrate [30]. It approximates the set of possible behaviors that a system can exhibit and can be utilized to ensure that a set of unsafe behaviors is never reached.

One disadvantage of model checkers and reachability analysis is their susceptibility to the state-explosion problem, where the number of states increases exponentially with the number of input variables [31]. Advancements in techniques and the availability of powerful computers have enabled these methods to verify complex verification challenges.

However, they are still placed at the bottom of the figure because the type and complexity of systems they can verify are more limited than the ones simulation-based methods can verify.

The analytic methods are less exhaustive but more scalable than the formal ones. Linear analysis refers to the process of linearising the system and then using Lyapunov's indirect method to prove stability [9]. The lineralization can be done both numerically and symbolically. This method is placed where it is on the figure because it can only give local proof and therefore is not fully exhaustive. Stability Proofs determine stability or invariant sets of a system using Lyapunov's direct method [9]. Unlike linear analysis, this approach can verify global stability, which is why it is located further to the right in the Figure.

Simulation-based methods refer to creating a simulation model of a system together with its operative environment to perform the testing on the simulation model instead of the actual system in the real world [7]. The simulation refers to manually or randomly selecting inputs to the simulations and observing the system behavior. This method is very scalable as it can be done on any system, but not exhaustive as the number of input variations can be infinite. Simulation is, in general, only a testing method because a single simulation only assesses behavior for a single test case. Simulation, in combination with valid processes for test case selection methods, can provide extensive coverage and could potentially be considered to provide verification in the sense that it is defined in [9].

Test-vector generation is an automated process used to generate system inputs that meet specific coverage criteria [9]. Test-vector generation is located to the right of linear analysis in the Figure, as it is expected to explore a wide range of system behaviors by utilizing a finite collection of simulation traces. While Test-vector generation techniques may be applied to any system that supports simulation, they may not achieve the desired level of coverage for large models, placing them in the middle of the scalability dimension.

Concolic testing is a method that combines concrete and symbolic execution of the code under test to generate new test inputs for better test coverage [32]. Concolic testing finds counterexamples automatically for many systems but is limited due to computational cost. However, there are various ways to apply concolic testing, so arguments could be made to relocate its location on the Figure.

Multiple-shooting attempts to discover a counterexample by executing many partial simulations instead of complete simulations and then joining the outcomes together to identify a counterexample [7]. This method is suitable for testing hybrid systems because hybrid systems consist of both discrete and continuous behavior and can therefore be challenging to verify using other approaches [33].

Falsification methods use a parameterization of test cases and apply optimization techniques to search for cases with low performance [34]. Instead of searching for correct behaviors, falsification methods search for counterexamples to prove that the system does not hold for the requirements. These requirements are often given by STL semantics. This method is the one that is going to be more explored in this thesis.

Simulation-guided Lyapunov analysis uses the direct method to search for a Lyapunov candidate that proposes the system is either stable or to find an invariant set [10]. This method is exhaustive, hence a verification method. This method will also be further examined in this thesis.

Kapinski et al. [9] deliver a more in-depth understanding of the concepts briefly explained above. In this master thesis, the main focus will be instead on optimization-guided falsification with STL and simulation-guided Lyapunov analysis.

4.2 Preliminaries

A model of the system is given by M [9]. The system has a set of parameters P and a set of inputs U. A particular behavior of a system is given by $\Psi(M, p, u)$, where $p \in P$ and $u \in U$. During testing and verification, the system is checked against a property ϕ.

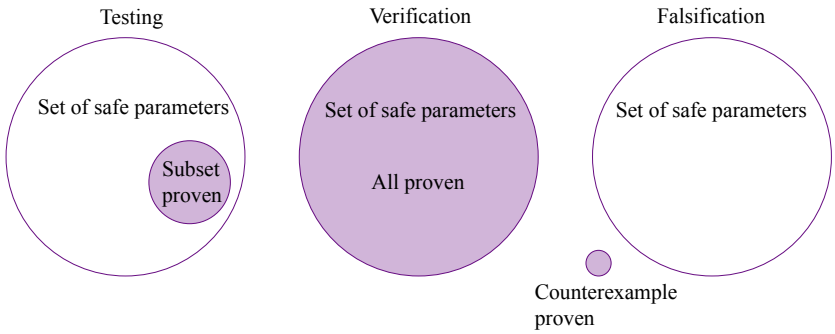


Figure 4.2: Figure to illustrate the difference between testing, verification, and falsification given a safety requirement.

In addition to the definitions below, Figure 4.2 illustrates the differences between testing, verification, and falsification.

Definition 4.2.1 (Testing)**.** *Testing is the task of determining whether a subset of P and a subset of U in M holds for ϕ. i.e. whether $\psi(M, \hat{P}, \hat{U}) \models \phi$, where $\hat{P} \subseteq P$ and $\hat{U} \subseteq U$ [9]*

Definition 4.2.2 (Verification)**.** *Verification is the task of determining whether all sets of P and U in M hold for ϕ. i.e. whether $\psi(M, P, U) \not\models \phi$, for a given P and U [9].*

Definition 4.2.3 (Falsification)**.** *Falsification is the task of determining a specific parameter of P and input of U in M which do not hold for ϕ. i.e. $\psi(M, p, u) \not\models \phi$ where $p \in P$ and $u \in U$ [9].*

For example, in the context of this thesis, the testing of a BMS aims to demonstrate that the BMS behaves as intended within the defined subset of foreseeable scenarios. Verification involves confirming that the BMS behaves correctly for all reasonable scenarios and parameters. Falsification refers to the process of identifying a set of parameters that indicate incorrect behavior of the BMS.

Each of the three options has its own advantages and disadvantages. Testing is simple, but verification is more comprehensive and provides a stronger proof of correctness than testing [9]. If a model is verified, it cannot be falsified. Falsification, on the other hand, can be useful for debugging the model by providing valuable information to the user.

4.3 Signal temporal logic

In 1977 Amir Pnueli introduced linear temporal logic (LTL) for program verification in computer science to formally specify desirable and acceptable behaviors of reactive systems [35]. That is, systems that continuously interact with the environment. The fundamental concept of the method is the time dependence of events. Pnueli introduced the concept of evaluating whether a system always or eventually holds for a condition. The method became a turning point in formal verification, and Pnueli received the Turing Award for his work.

The verification framework developed over the years. LTL operates on boolean signals in discrete time. Different types of temporal logic have evolved from the original. Metric temporal logic (MTL) was introduced to operate on boolean signals in continuous time. Later Signal temporal logic (STL) came as an addition to MTL to operate on real-valued signals in continuous time [7]. The motivation was to produce a verification method suitable for continuous and hybrid systems [36]. Moreover, STL is equipped with quantitative

semantics that allows for the determination of a specification's robustness value for a given signal.

Before introducing the formal syntax of STL, an informal description of each operator is presented as done in [7]:

- *Conjunction*: $\phi_1 \wedge \phi_2$ is true if both ϕ_1 and ϕ_2 are true.
- *Disjunction*: $\phi_1 \vee \phi_2$ is true if either ϕ_1 or ϕ_2 are true.
- *Negation*: $\neg\phi$ is true if ϕ is false.
- *Implication*: $\phi_1 \implies \phi_2$ is true if ϕ_2 follows from ϕ_1, that is, $\phi_1 \implies \phi_2$ is false if and only if ϕ_1 is true and ϕ_2 is false.
- *Eventually*: $\Diamond\phi$ is true if ϕ is true at some time.
- *Always*: $\Box\phi$ is true if ϕ is true at all times.
- *Next*: $\bigcirc\phi$ is true if ϕ is true at the next discrete time step.
- *Until*: $\phi_1 U \phi_2$ is true if ϕ_1 is true until ϕ_2 first becomes true.
- *Release*: $\phi_1 R \phi_2$ is true if ϕ_2 is true until ϕ_1 first becomes true.

Definition 4.3.1 (STL syntax)**.** *The formal syntax of an STL formula ϕ is given as follows:*

$$\phi ::= T \mid \pi_c \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc_I \phi \mid \phi_1 U \phi_2 \tag{4.1}$$

where π_c is a predicate on the form $\pi(x) > 0$, T is the True constant and I is an interval. A predicate π is a differentiable function that maps the state x to a scalar value [37]. Predicates are the building blocks in STL formulas.

Various articles present different definitions for the formal syntax of STL. However, in this thesis, the syntax definitions proposed in [7] have been adapted due to their clarity and comprehensiveness. It is worth noting that despite the different definitions, most logical and temporal operators in STL can be derived from each other, which ultimately results in

similar syntax definitions, e.g.:

$$\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2) \tag{4.2}$$

$$\phi_1 \implies \phi_2 \equiv \neg\phi_1 \vee \phi_2 \tag{4.3}$$

$$\Diamond_I \phi \equiv TU_I\phi \tag{4.4}$$

$$\Box_I \phi \equiv \neg\Diamond_I\neg\phi \tag{4.5}$$

$$\phi_1 R \phi_2 \equiv \neg(\neg\phi_1 U_I \neg\phi_2) \tag{4.6}$$

One notable attribute of STL is its ability to provide a robustness score. This distinguishes STL from LTL, where users only receive a binary value indicating whether a formula is satisfied or not. In contrast, the robustness semantics of STL provides a continuous real-valued measure of the extent to which a signal satisfies or violates an STL formula [37]. Specifically, if the robustness score is less than zero, it indicates a violation of the formula, while scores above zero indicate satisfaction. The lower the score is below zero, the greater the degree of dissatisfaction.

Definition 4.3.2 (Robustness semantics)**.** *[37]*

$$\rho(s_t, T) = \rho_{max} \quad where \quad \rho_{max} > 0 \tag{4.7}$$

$$\rho(s_t, \pi_c) = \pi(x_t) - c \tag{4.8}$$

$$\rho(s_t, \neg\phi) = -\rho(s_t, \phi) \tag{4.9}$$

$$\rho(s_t, \phi_1 \wedge \phi_2) = min(\rho(s_t, \phi_1), \rho(s_t, \phi_2) \tag{4.10}$$

$$\rho(s_t, \phi_1 \vee \phi_2) = max(\rho(s_t, \phi_1), \rho(s_t, \phi_2) \tag{4.11}$$

$$\rho(s_t, \phi_1 \implies \phi_2) = \max_{t' \in I}(-\rho(s_t, \phi_1), \rho(s_t, \phi_2) \tag{4.12}$$

$$\rho(s_t, \Diamond_I \phi) = \max_{t' \in I} \rho(s_{t'}, \phi) \tag{4.13}$$

$$\rho(s_t, \phi_1 U_I\}phi_2) = \max_{t' \in I} min(\rho(s_{t'}, \phi_1), \min_{\tau \in [t,t']} \rho(s_\tau, \phi_2)) \tag{4.14}$$

In Figure 4.3, a simulation trace is presented together with its robustness. The STL formula specifies that the simulation trace must always remain within 20% and 80%, depicted by the red area on the plot. Any values outside of this range indicate a violation of the STL. The robustness trace drops below zero whenever the simulation trace deviates from the STL. The magnitude of the negative robustness signifies the degree to which the simulation trace deviates from the STL, indicating the extent of the violation.

An alternative approach to evaluating STL formulas called VBools is proposed in [38].

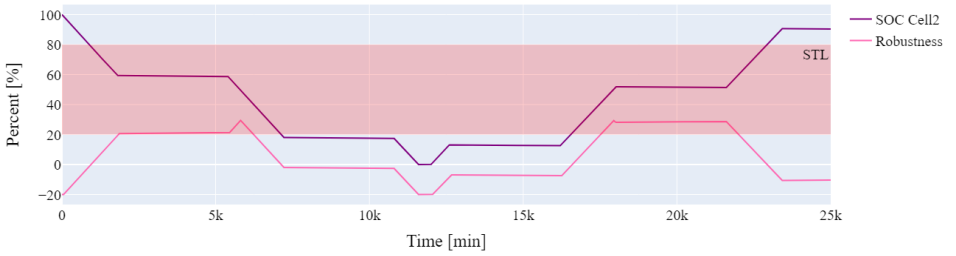


Figure 4.3: Example showing the robustness of a simulation trace. The STL formula states the value of SOC Cell2 should always stay between 20% and 80%.

The method is very similar to robustness, but the difference is that VBools allows the tester to control how robustness is measured for each property, while robust semantics imposes the same robustness measure on all properties. Then STL is used in falsification to find counterexamples. VBools can be a better alternative. The interpretation of the severity of a counterexample depends on the physical context of the property being evaluated, implying that no single semantics approach can always be universally applied as the most appropriate.

The article claims they have a strong suspicion that using VBools in optimization-based falsification may actually improve bug-finding effectiveness. However, they do not have enough experimental evidence yet to support this claim. As robustness is more widely adapted, it is the approach employed in this master's thesis.

4.4 Optimization

STL is used in falsification together with an optimization algorithm. Before diving into optimization-guided falsification, an explanation of optimization will be done.

Optimization refers to the process of finding the best solution to a problem within a given set of constraints [39]. Mathematically, optimization refers to the process of minimizing or maximizing a function while considering constraints on its variables. The optimization problem is stated in [39] as:

$$\begin{aligned} &\min_{R^n} && f(x) \\ &\text{subject to} && c_i = 0, i \in \mathcal{E} \\ &&& c_i \geq 0, i \in \mathcal{I} \end{aligned} \tag{4.15}$$

where

- x is the vector of variables, also called unknowns or parameters.
- f is the objective function, a function of x that we want to maximize or minimize.
- c_i are constraint functions, which are scalar functions of x that define certain equations and inequalities that the unknown vector x must satisfy.

The optimization problem is solved using optimization algorithms. These algorithms work by iteratively refining a candidate solution until a satisfactory solution is found [39]. The optimization algorithm typically starts with an initial guess for the solution and then makes incremental changes to the variables in order to improve the objective function value. At each iteration, the algorithm evaluates the objective function and uses this information to determine the direction of the search for the next iteration.

The choice of the search direction and how to arrive at the optimum is determined by the specific algorithm being used. For example, gradient descent algorithms use the gradient of the objective function to determine the direction of the search [39]. These algorithms depend on the objective function to be well-known and easy to evaluate and compute.

However, when evaluating cyber-physical systems, the objective function may be non-differentiable, discontinuous, or noisy [34]. Computation of the gradient can be either expensive or not even possible. Therefore gradient-free optimization algorithms are needed. Derivative-free optimization algorithms are a broad category of algorithms that do not use gradient information to search for an optimal solution [39]. Instead, they rely on other methods to explore the search space and refine the candidate solutions iteratively. Relevant optimization algorithms in this thesis are linear programming, Nelder-Mead, and Bayesian optimization.

In linear programming, the objective function and constraints are linear [39]. A linear program can be solved in different ways. A common one is the simplex algorithm. It is conceptually simple, computationally efficient for many practical problems, and it has a sound theoretical foundation. The method starts with an initial feasible solution and then iteratively improves it until the optimal solution is found. The method is primarily derivative-free, but the gradient can be added for extra support. The algorithm is based on a geometric interpretation of the problem, where the feasible region of the linear programming problem is represented as a convex polytope. The algorithm moves from one vertex of the polytope to another along the edges, and the objective function is improved at each iteration. The fact that the feasible region is a convex polytope ensures that the algorithm will always converge to an optimal solution.

The Nelder-Mead algorithm is a heuristic optimization method for finding the minimum

of a nonlinear objective function in multiple dimensions [39]. It belongs to the family of direct search algorithms, which do not require the calculation of derivatives, and can handle non-smooth or non-convex functions. The algorithm starts with an initial set of points, called the simplex, which defines a region of the search space. The simplex consists of n+1 points, where n is the number of dimensions of the search space. In each iteration, the algorithm generates a new point by reflecting the worst point about the simplex's centroid. Next, it generates a new point by extrapolating from the reflected point. If neither of the two first steps results in a lower function value, the algorithm generates a new point between the best and worst points. If the function value still has not improved, a new point is generated between the best point and other points in the simplex. The method terminates if an optimum is found or the number of pre-set iterations runs out. The Nelder-Mead algorithm is relatively easy to implement and can handle non-smooth or non-convex functions that may cause gradient-based methods to get stuck in local minima. However, it may not converge to the global minimum for highly non-convex functions and may be sensitive to the choice of the initial simplex. Therefore, it is often used in conjunction with other optimization methods or with randomized initialization of the simplex to improve its performance.

Bayesian optimization is a technique used to optimize expensive objective function[40]. The objective function might be unknown or can only be evaluated through costly simulations or experiments [8]. It is particularly useful in situations where the objective function is non-convex, nonlinear, and has many local optima. Bayesian optimization has its name after the theorem it is built upon: Bayes theorem [40], which is stated as:

Definition 4.4.1 (Bayes theorem)**.**

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.16}$$

Bayes theorem describes the probability of an event based on prior knowledge of conditions that may be associated with the event [40]. Bayesian optimization works by maintaining a probabilistic belief about the objective function, called a surrogate function, and using an acquisition function to decide where to evaluate the function next. The surrogate function is built using Gaussian processes. For each iteration, the Gaussian processes are updated so the surrogate function becomes more and more like the true objective function.

Another benefit of Bayesian optimization is that it can balance the trade-off between exploration and exploitation and avoid evaluating the objective function at unpromising regions. Choosing exploration, the algorithm searches for points spread out across the whole range,

while with exploitation, the algorithm searches for points with low mean, i.e. points around the peak(s). This part is fundamental to a successful Bayesian optimization procedure. If there is no exploration, the algorithm can get stuck in local minima.

There are different acquisition functions. A common one is the Gaussian process upper confidence bound (UCB). For maximizing, it is:

Definition 4.4.2 (Guassian process upper confidence bound)**.**

$$UCB(x) = \mu(x) + \kappa\sigma(x) \tag{4.17}$$

where μ is the mean, σ is the standard deviation of the surrogate function and κ is a constant for determining exploration or exploitation [40].

The method is visualized in Figure 4.4 to Figure 4.7. The figures are made by [8]. The upper plot represents the true function, while the lower plot displays the acquisition function. Black dots on the true function indicate points where Bayesian optimization has identified promising locations. The pink region represents the interval with a high probability of containing the true function. The process involves maximizing the acquisition function to determine the next point for evaluating the function. In this case, the acquisition function employed is the UCB. As the iterations progress, the surrogate function, depicted in the upper plot, increasingly approximates the true objective function. Gradually, the algorithm approaches the true maximum.

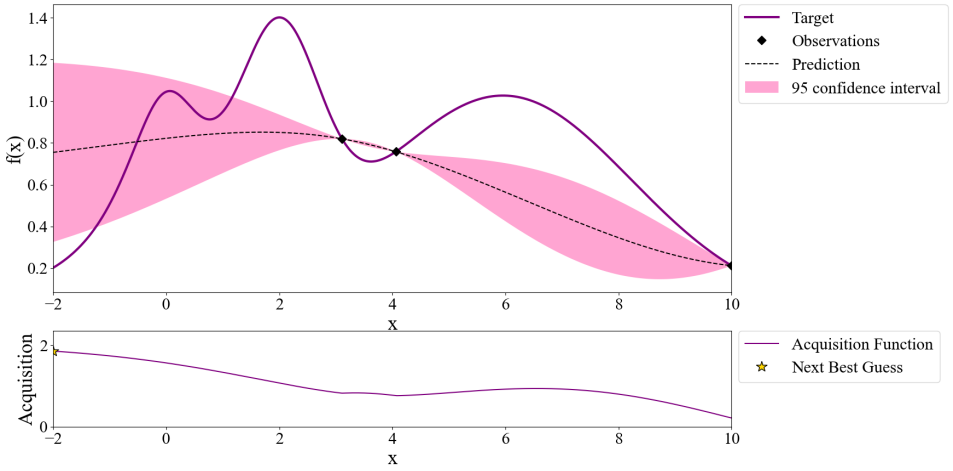


Figure 4.4: Three iterations of Bayesian optimization. Adapted from [8]

In Figure 4.4, three iterations of Bayesian optimization are depicted, yet the optimal solu-

tion has not been found. The yellow star on the acquisition function graph indicates the next guess for Bayesian optimization.

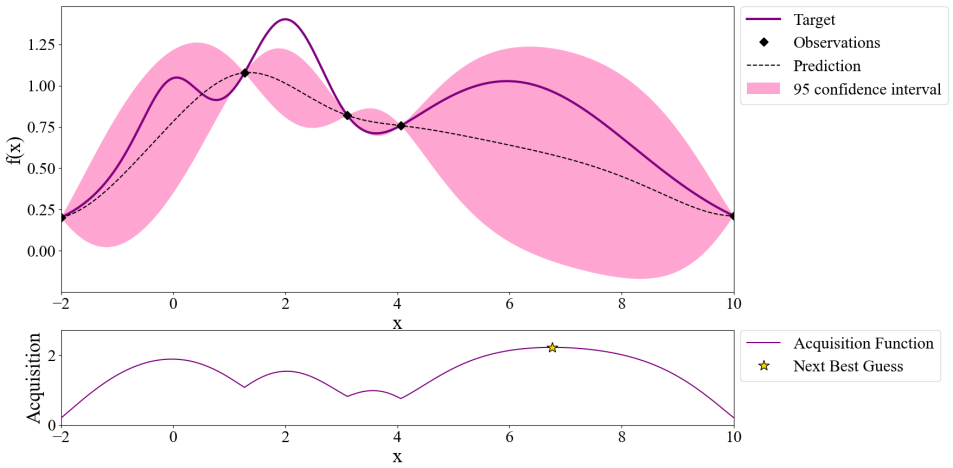


Figure 4.5: Five iterations of Bayesian optimization. Adapted from [8]

Moving on to Figure 4.5, five iterations of Bayesian optimization have been performed. Although the surrogate function shows an improved resemblance to the true function, further progress is still required. The next guess for the optimum is located in the upper-right section of the acquisition function.

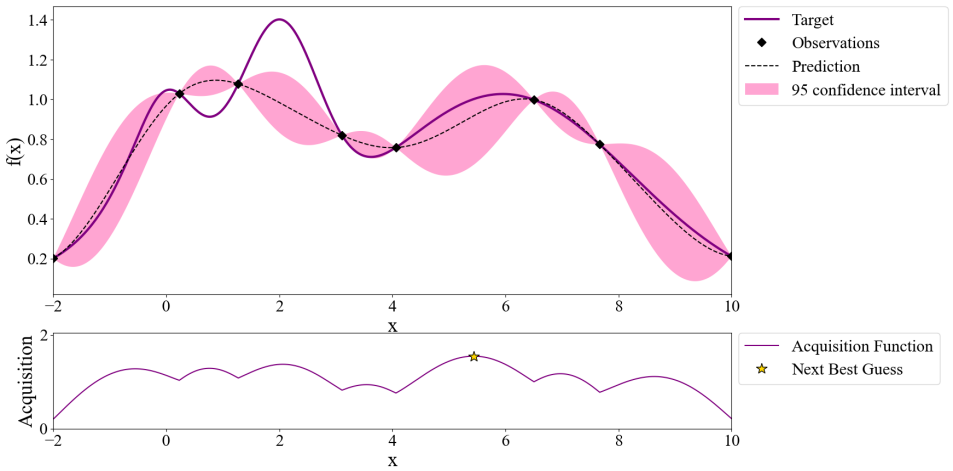


Figure 4.6: Eight iterations of Bayesian optimization. Adapted from [8]

After eight iterations of Bayesian optimization, the surrogate function closely resembles the target function, but the optimum is still not found in Figure 4.6

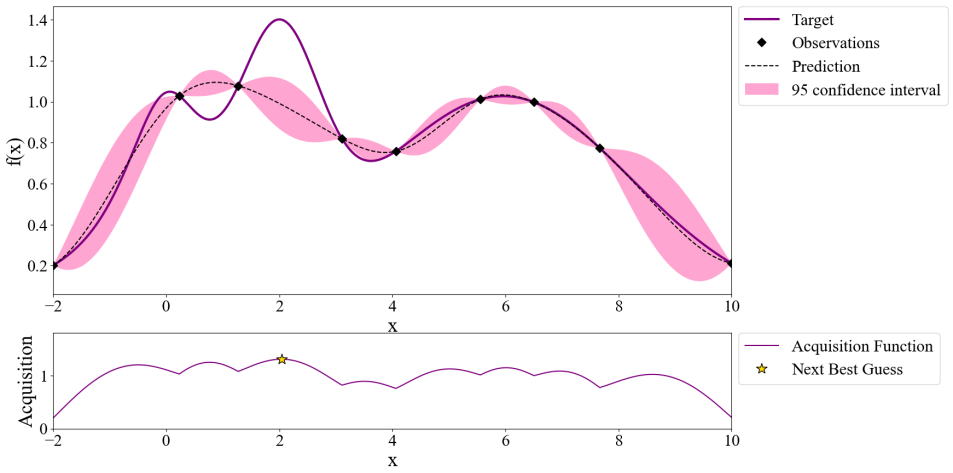


Figure 4.7: Nine iterations of Bayesian optimization. Adapted from [8]

Finally, Figure 4.7 reveals that the next guess appears to be the optimum, leading to the termination of the optimization in the subsequent iteration. Within only ten iterations, the method successfully discovers the optimal solution.

4.5 Optimization-guided falsification

The optimization algorithm uses the STL robustness score in its search for an optimum. The basic architecture of optimization-guided falsification is shown in Figure 4.8.

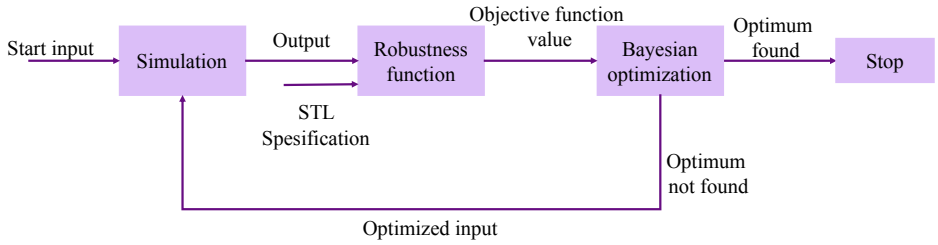


Figure 4.8: Overview of optimization-guided falsification. Adapted from [9]

Here are the steps:

1. Pick a start input x to all parameters
2. Compute the output by simulation
3. Calculate the robustness of the output using STL
4. Evaluate the robustness
5. If the robustness is lower than zero, a falsifying example is found, and the procedure can halt
6. If not, compute optimization to search for input values that yield lower robustness

Related work

Optimization-guided falsification uses an optimization algorithm and a requirement to guide the optimization. Which type of optimization algorithm used differs widely. Also, it can be done on different platforms. Here are some related works worth mentioning because of their broadness and their influence on this master's thesis.

There exist two notable Matlab toolboxes for optimization-guided falsification. The first toolbox is S-TaLiRo [41]. It accepts Simulink systems or user-defined functions of a model, along with an MTL formula, to search for instances of poor robustness using optimization methods. The user can select from various stochastic optimization algorithms, including Simulated Annealing, Ant Colony Optimization, Genetic Algorithms, and Cross Entropy.

The second toolbox is Breach [42]. Breach differentiates from S-TaLiRo by treating exogenous inputs uniformly and offering users greater flexibility in placing control points along the timeline [9]. Another difference is that Breach uses a nonlinear global optimizer based on the Nelder–Mead simplex-based algorithm.

Zahra Ramezani completed her Ph.D. thesis on Optimization-Based Falsification of Cyber-Physical Systems at the Chalmers University of Technology [34]. As part of her research, she has published several papers exploring various approaches to optimization-based falsification. Two of the papers are:

1. In [43], Ramezani et al. establish an optimization-free method called Hybrid Corner-Random. The method explores extreme values within the allowed parameter ranges. The method performs acceptably on some of the benchmark problems, but it cannot handle the hardest ones. Further, they suggest a line-search falsification method,

which uses the line-search optimization algorithm, which is a gradient-free optimization algorithm. This method, on the other hand, performs well in all test cases.

2. In [44], they explore the Bayesian optimization algorithm and its different alterations of it. In [34], Ramezani concludes that the Bayesian optimization method, based on the outcome of experiments, has a clear advantage over previously presented methods.

Falsification of safety conditions with Bayesian optimization is done in [45] where he presents pseudocode for a Bayesian optimization using upper confidence bound as the acquisition function.

4.6 Simulation-guided Lyapunov Analysis

Simulation-guided Lyapunov analysis is an emerging method for verification of cyber-physical systems [46]. This method uses Lyapunov's theorem for stability.

Definition 4.6.1 (Lyapunov's stability theorem [47])**.** *Let $x = 0$ be an equilibrium point for $\dot{x} = f(x)$ and $D \subset R^n$ be a domain containing $x = 0$. Let $V : D \to R$ be a continuously differentiable function such that*

$$V(0) = 0 \quad and \quad V(x) > 0 \quad in \quad D\backslash\{0\} \tag{4.18a}$$

$$\dot{V}(x) \leq 0 \quad in \quad D \tag{4.19a}$$

Then $x = 0$ is stable. Moreover, if

$$\dot{V}(x) < 0 \quad in \quad D\backslash\{0\} \tag{4.20a}$$

Then $x = 0$ is asymptotically stable.

If a system is asymptotically stable, it means that all system states converge to a particular value. By leveraging this concept, engineers and researchers can ensure that a system meets the desired performance criteria. For instance, if the goal is for the system state to converge to a particular reference value, asymptotic stability can be used to demonstrate that the system will indeed converge to this value.

Lyapunov functions can be used to not only prove stability but also to perform other verification tasks of systems [9]:
Stability: property ϕ = "is the system stable?"

Performance bounds: property ϕ = "The system remains within some set S."
Barrier certificate: property ϕ = "Given that the system is initiated in χ_0, it will never reach $\mathcal{F}$"

The method could be more mature and has been implemented on an academic scale [46]. This has been done in this article [10]. The article is made for hybrid systems, but the method works on non-hybrid systems as well.

While Lyapunov functions are a powerful tool for proving stability and safety in nonlinear and dynamical systems, finding a proper Lyapunov function is a challenging task. In [10], they turn the problem around. Rather than starting with a Lyapunov function, the process searches for a possible Lyapunov function, and if it finds one and the analysis is sound, the system is verified. This is done using simulation traces and the assumption of a sum-of-squares polynomial structure to obtain a candidate Lyapunov function. The process is shown in Figure 4.9.

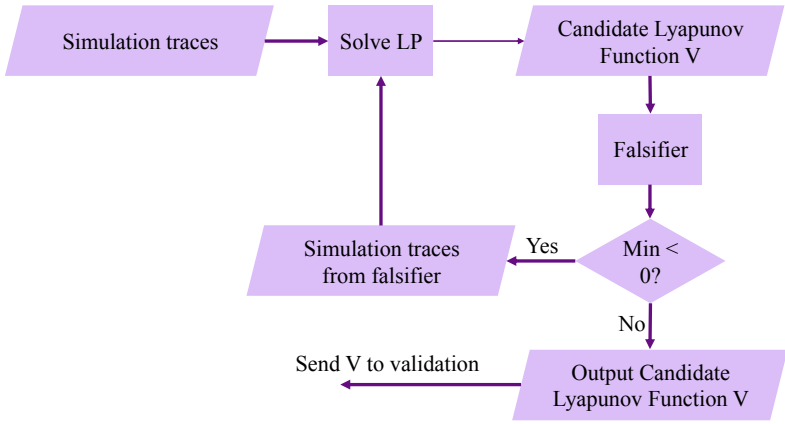


Figure 4.9: Overview of part one of Simulation-guided Lyapunov verification by [10]. Adapted from [10]

The process starts by obtaining a collection of simulation traces. These are then formulated as a linear program. For instance, the simulation traces are consolidated into a vector encompassing all state values, denoted as 'x'. This vector is combined with an unknown matrix 'P', resulting in the formulation V(x) = $x^T Px$. The objective of the linear program is to explore suitable values for 'P' that satisfy the Lyapunov conditions. If a suitable 'P' is discovered, it implies the identification of a potential Lyapunov function candidate.

The constraints in the linear program are given by:
simulation trace θ and candidate Lyapunov function $V(x) = x^T Px$. j is time steps, and i is for switched mode for hybrid systems. For a non-hybrid system, i will be 1.

$$V(\theta(t_j)) > 0 \tag{4.21}$$

$$V(\theta(t_j)) - V(\theta(t_{j+i})) - \gamma k||\theta(t_j)||^2 > 0 \tag{4.22}$$

$$\gamma > \epsilon \tag{4.23}$$

The constraints are formulated to check the Lyapunov stability definition. The first constraint guarantees the positive definiteness of the Lyapunov function. Meanwhile, the second and third constraints are employed to examine the negative definiteness of the derivative of the Lyapunov function.

At this point, the system is only tested on a finite collection of simulation traces. There can still exist simulations that do not hold for the stability criterion. The subsequent step is, therefore, a falsifier. In the article, they used the global optimizer Nelder-Mead to search for counterexamples that violate the Lyapunov conditions. In the falsifier, the Lyapunov conditions are stated as the objective function. If the objective function is less than zero, then a counterexample is found. The counterexample is then added to the constraints in the linear program problem, and a new one will be executed, giving a new candidate Lyapunov function. This loop continues until no new counterexample is found, resulting in a Lyapunov function candidate.

As global optimization techniques are not exhaustive, it is essential to conduct a thorough evaluation of the Lyapunov candidate [10]. Moving over to part two of the procedure, shown in Figure 4.10. In the next step, a Satisfiability Modulo Theories solver is used to validate the soundness of the Lyapunov candidates. Satisfiability Modulo Theories solvers are automated reasoning tools that can efficiently decide the satisfiability of logical formulas involving complex theories or constraints [48]. This synthesis process ensures that the Lyapunov function is constructed correctly and that its properties align with the desired stability requirements. It is also in this step that the property to check for is specified, whether it is a Lyapunov function to prove stability, a forward invariant set to check for performance bounds, or a barrier certificate. If the analysis is sound, the procedure halts, and the system is officially verified. [10] uses z3 [49] and dReal [50] and the symbolic tools in Mathematica [51] to verify the obtained Lyapunov candidate function.

In [10], they have proven that the method works on examples with nonlinear or hybrid

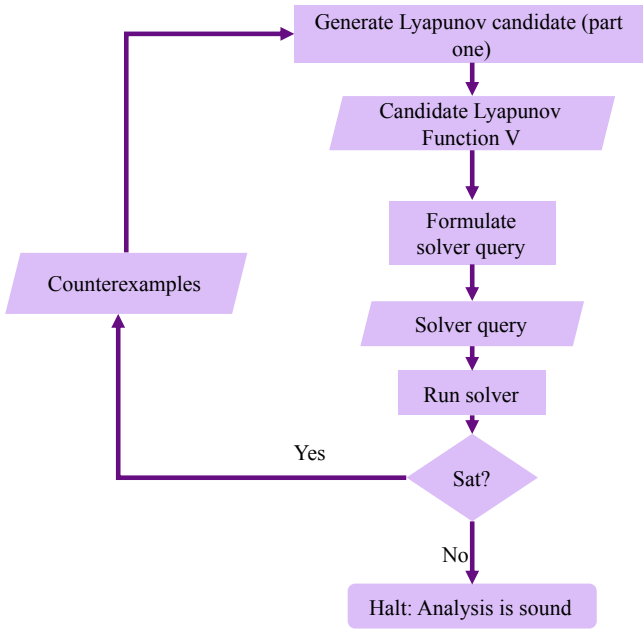


Figure 4.10: Overview of part two of Simulation-guided Lyapunov verification. Adapted from [10]

dynamical systems. They state that there exist no guarantees that the procedure will terminate with a sound analysis for all problems, but their examples prove that the method is capable of verifying challenging systems.

5

Lithium-ion Battery and BMS

This chapter goes deeper into the Li-ion battery and BMS part of the all-electric actuation control. Presenting failure modes of the Li-ion battery. Furthermore, the Matlab model will be presented, along with the selected test cases.

5.1 Lithium-ion battery

As the world shifts towards more renewable energy sources and electrification, the use of batteries is increasing. Li-ion batteries came on the market in 1991, and their popularity has exploded since [52]. The idea behind Li-ion batteries is that they are built of the lightest metal on the periodic table, Lithium. Because of this, they have a high energy and power density compared to other batteries today. The energy density of Li-ion, nickel-cadmium (Ni-Cd), and Lead-acid batteries are compared in Figure 5.1.

Another advantage of Li-ion batteries is their long cycle life, meaning they can be recharged and discharged many times before their capacity decreases. This sets them apart from other battery types, such as Ni-Cd batteries, which have a shorter cycle life and can suffer from memory effect. That is, when Ni-Cd batteries are charged after not fully discharged, they can ”remember” the shorter discharge cycle and gradually lose their ability to hold a full charge [53].

Li-ion batteries also have a low self-discharge rate, allowing them to retain their charge for longer periods of time when not in use [54]. This feature makes them an ideal choice for devices that are not frequently used. e.g. ESD valve.

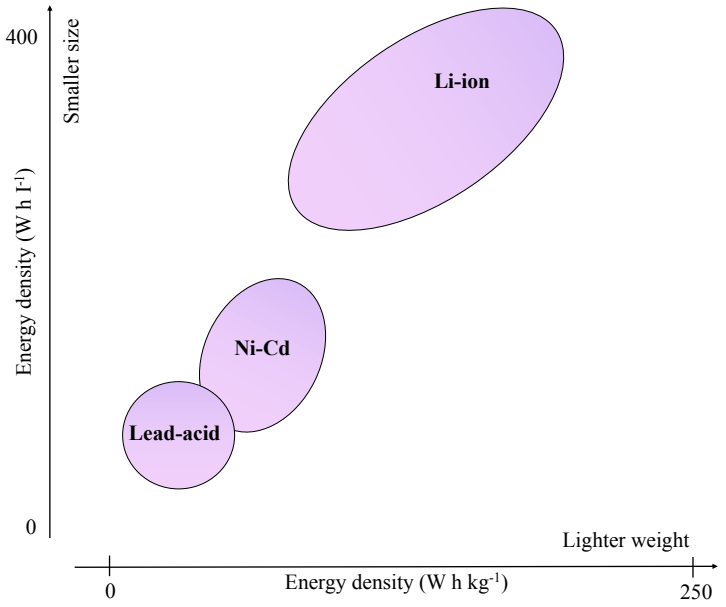


Figure 5.1: Comparison of battery types. Adapted from [11]

While Li-ion batteries offer many advantages, they do come with some drawbacks, including a higher cost and the risk of fire if not handled or charged properly. Safety can be ensured in multiple ways. On the battery cell level by space or isolation between the cells. The placement of the battery in the environment to ensure cooling conditions and fire detection [52]. Last but most importantly, Li-ion batteries need a BMS.

5.2 Battery management system

A BMS is an electronic system that plays a crucial role in ensuring the safety and optimal performance of a battery [54]. The BMS monitors and controls the condition of the battery. The main function of the BMS is to detect and prevent failures within the battery. A list of functions of the BMS is shown in Figure 5.2.

The article [12] gives an overview of the functions of the BMS. In general, the BMS monitors the state of the cells, battery health, voltage, current, temperature, and remaining runtime. The BMS optimizes the battery's performance by preventing and mitigating failures. The BMS ensures that each cell is charged and discharged properly to prevent any cell from being overcharged or overdischarged. This is called cell balancing, and it is typ-

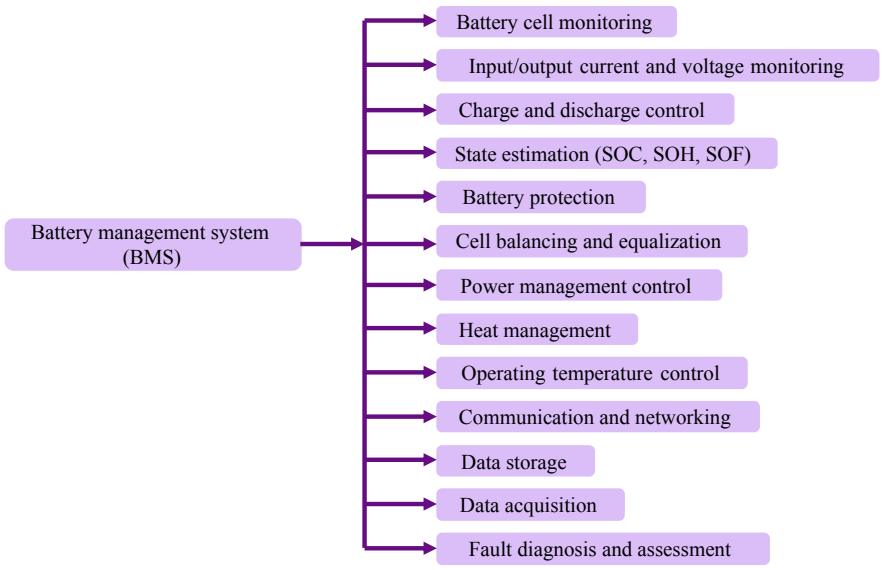


Figure 5.2: Overview of BMS functions. Adapted from [12]

ically done by diverting excess charge from the higher voltage cells to the lower voltage cells [12]. In the event of abnormal conditions, the BMS is to put into measures to prevent damage or combustion of the battery.

5.3 Faults of Lithium-ion batteries

The article [54] gives an overview of the failures of a Li-ion battery. The article builds upon the study by Lyu et al. [55] that extensively examines battery failures at a chemical level. This thesis will not go into the depth of the chemistry inside the Li-ion battery. Instead, the focus is on understanding and modeling the outcome of failures. The referenced article is utilized because it provides a broader overview, which aligns with the objectives of this thesis.

[54] distinguish between internal and external faults as shown in Figure 5.3. The article mentions more failures, but after conversations with the supervisor, some are not that relevant for the battery in the all-electric Xmas tree and, therefore, not taken into here.

Battery faults	
External	Internal
Sensor fault: - Voltage - Current - Temperature	Internal short circuit Overcharge Overdischarge Accelerated degradition Thermal runaway

Figure 5.3: Overview of internal and external faults of Li-ion batteries. Adapted from [10]

5.3.1 Internal short-circuit

An internal short circuit (ISC) is an issue that can occur in a battery when the separator layer between the electrodes fails, causing the two electrode materials to be internally and electronically interconnected [54]. This failure can be due to various reasons, such as high temperature, cell deformation, dendrite formation, or compressive shock. The result is high local current densities, self-discharge, and a temperature increase. As the lithium-ions and electrons are released at the anode and travel across the electrolyte toward the cathode, they trigger contact between the anode and cathode, leading to internal short-circuiting. This failure can lead to thermal runaway.

5.3.2 Overcharge

The occurrence of overcharge in the battery cells is a prevalent issue that stems from factors such as the variance in cell capacity within the pack, erroneous voltage and current measurement, or imprecise state of charge (SOC) estimation by the BMS [54]. A regular battery pack can also experience overcharge when the charger malfunctions. Overcharging Li-ion batteries instigates electrochemical reactions among battery components, resulting in the depletion of active materials. Furthermore, the accumulation of gases in sealed batteries can lead to the battery bursting. Overcharge can lead to ISC due to an increase in the battery's surface temperature and ultimately lead to thermal runaway.

5.3.3 Overdischarge

Overdischarge occurs when the battery is discharged below its recommended lower voltage limit. [56]. Overdischarge can be caused by the same factors as overcharge. This failure

can worsen the lifespan of the battery and lead to ISC and, ultimately, thermal runaway [57]. Figure 5.4 shows how it looks when cells have different charges.

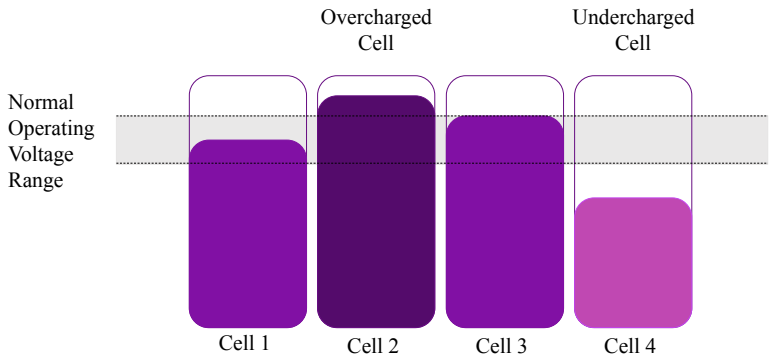


Figure 5.4: Example of disbalanced cells. Adapted from [12]

5.3.4 Sensor drift

Sensor drift is a failure in one of the sensors in the battery, e.g. temperature, voltage, or current, leading to incorrect measurements. The failure can be caused by physical factors such as vibration [58]. The sensors help the BMS monitor the battery and manage the battery operation effectively. Failure in temperature sensors can lead to inaccurate thermal management, ISC, overheating, and ultimately, thermal runaway [59]. Voltage sensors monitor cell voltages and help the BMS estimate the SOC of the cells [54]. If the BMS cannot estimate the SOC correctly, it can lead to the failures overcharge and overdischarge. A faulty current sensor can also result in an inaccurate estimation of SOC.

5.3.5 Accelerated degradation

Cell degradation is the gradual loss of a battery cell's capacity and performance over time, resulting from chemical and physical changes that occur during use [54]. The degradation process can be accelerated when the battery is stored at high temperatures [60]. This should not be a problem for the battery in this thesis as it is to be placed at the ocean bottom. Other factors are impedance increase, higher frequency of the charging and recharging cycle, change in SOC, and voltage rates [61; 62]. Accelerated degradation can lead to ISC and, ultimately, thermal runaway.

5.3.6 Thermal runaway

This failure is the most severe failure that can occur on the Li-ion battery [54]. It is triggered by a cascade of exothermic reactions within the battery, leading to a rapid increase in temperature and pressure that can ultimately cause the battery to explode and catch fire.

5.4 Matlab model

The DT of the battery is modeled with Simulink in Matlab by Ph.D. candidate Björklund. The DT was developed in parallel with this master. Different prototypes were made and tested on the way.

In this report, the result from two DTs is included:

DT prototype	Number of cells	Failures implemented
DT1: Battery	8	ISC
DT2: Battery and BMS	4	ISC, Cell balancing and sensor drift

5.4.1 Modeling

Battery

Accurately modeling the chemical reactions within a battery is challenging. To overcome this, an equivalent circuit model is commonly employed. An electrical circuit model is easier to model and simulate than chemical reactions, and studies show that they are reliable models with high accuracy [63]. In order to model the battery cells, a second-order equivalent circuit model is used as shown in Figure 5.5.

The equivalent circuit model represents battery cells by employing resistors and capacitors in series, effectively mimicking the behavior of a chemical cell [13].

To determine the parameters of the RC parallel network, i.e., the values of R1, R2, C1, and C2, a lookup table approach is utilized. A lookup table is a Matlab Simulink block that utilizes arrays of data to approximate mathematical functions to effectively map input values to corresponding output values. The lookup table allows the model to adapt to changing operating conditions to capture the expected behavior during higher temperatures, lower load demands, and a reduced SOC. Utilizing lookup tables for the battery cell parameters in the equivalent circuit model enables capturing behavior as a function of the current state of the operational conditions. Figure 5.6 show an example of a lookup table. The plot shows the R1 value for the different SOC and temperature values.

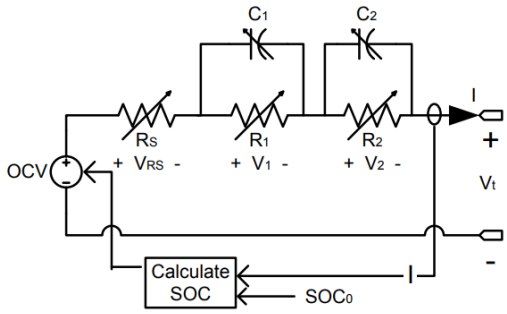


Figure 5.5: Equivalent circuit model of Li-ion battery proposed by [13]. Adapted from [13]

A battery exists of multiple cells. For each cell in the battery pack, an equivalent circuit model is used to capture the dynamic behavior. However, the parameters of the lookup tables are set to unique parameters to account for variance between cells in terms of physical properties.

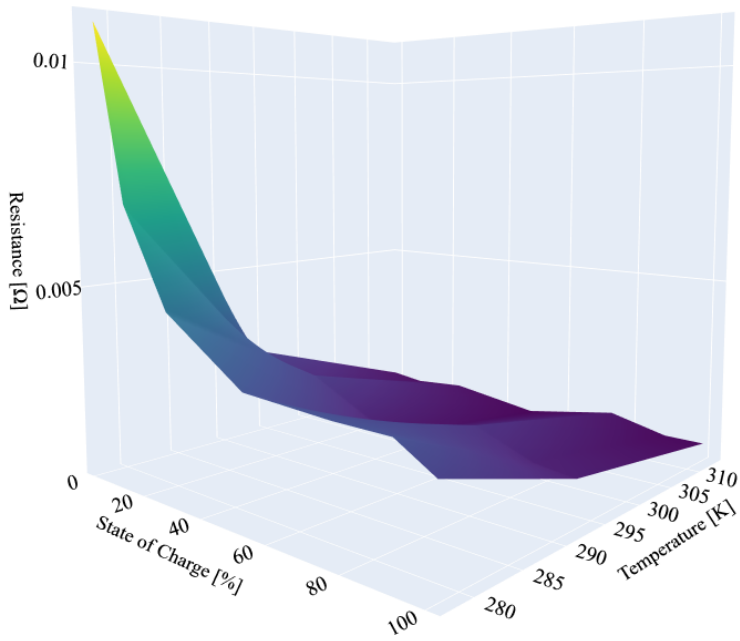


Figure 5.6: Example of a lookup table.

BMS

The BMS is, among other things, used for SOC estimation of each cell. This is modeled using an extended Kalman filter. The model and the measurements in the extended Kalman filter are weighted and calibrated during the design period of the DT. The BMS gathers the measured voltage and temperature of each cell. The model used in the Kalman filter is built upon the first-order equivalent circuit model, in contrast to the cells, which are modeled in the second order. The second order gives a more dynamic behavior.

In the extended Kalman filter, the model parameters are kept constant, whereas the DT of the battery involves changing parameters. This discrepancy implies that the model used in the extended Kalman filter differs from the DT model of the battery. Consequently, when the Kalman filter receives data from the battery, accurate estimation becomes more challenging due to the substantial divergence between the two models.

By modifying the internal parameters of the DT, it is possible to represent behavior caused by physical degradation. This approach allows to assess the extent to which the DT can deviate from the model before the extended Kalman filter can no longer provide accurate estimations.

The extended Kalman filter also controls the charge and discharge cycle with a switch. The BMS uses the cell with the smallest SOC value to turn it on and the cell with the highest SOC value to turn the switch off. An incorrect estimation of the SOC of an individual cell can cause a scenario of discharging/charging cells to accelerate degradation and reduce the remaining useful life of the battery pack.

The BMS is responsible for cell balancing. This is done by measuring the voltages of the cells and connecting a simple logic. Whenever a cell exhibits excessively high voltage, a circuit is closed to divert the current away from the cell. During charging, the closed circuit redirects current from the overcharged cell to an externally connected resistor promoting a more balanced charge.

5.4.2 Test cases

Below are a description of the test cases implemented in the DT and which are tested in this master thesis.

Internal short circuit

ISC is modeled by an extra circuit connected to all the cells with a resistance, R_ISC. When the R_ISCR is high, the current goes where it should go, through the battery, and

the battery gets fully charged. When the R_ISCR is small, some of the current will go through the other circuit and not to the cell, leading to ISC.

The ISC is influenced by the internal resistance of each cell, denoted as Rs. The amount of ISC is determined by the difference between Rs and R_ISCR. When Rs is significantly larger than R_ISCR, the current will flow through the alternate circuit, resulting in ISC. The internal resistance of a cell can increase due to failures in that cell.

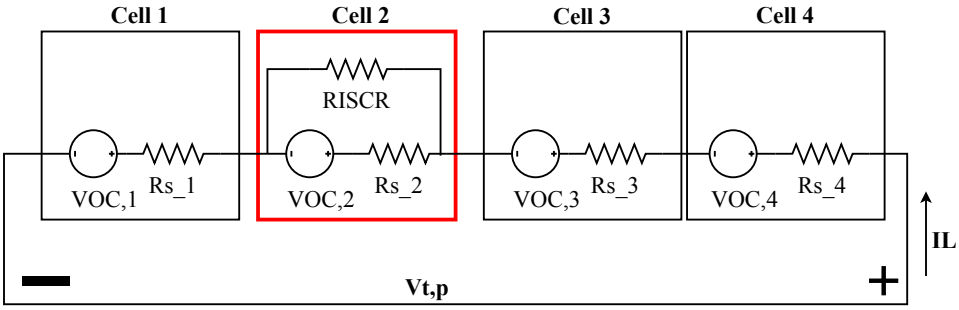


Figure 5.7: Circuit of modeled ISC circuit. Adapted from [14]

The BMS itself does not have the ability to detect ISCs at their current state, but this failure case is designed into the DT. Although a test case aimed at assessing the behavior of an ISC detection algorithm is unavailable at the current time, the overall performance is influenced by the short circuit.

Cell balancing

In order to evaluate the cell balancing function of the BMS, this test case involves modifying a cell to create a discrepancy among the battery cells, consequently making it harder for the BMS to efficiently balance the cells. This can be achieved by altering the parameters R1, R2, C1, and C2 of each cell. This is done by multiplying each parameter with a constant. The result of this can be seen in Figure 5.8. The above surface plot shows the original R1, while the surface plot under is the new one after R1 is multiplied by 0.3.

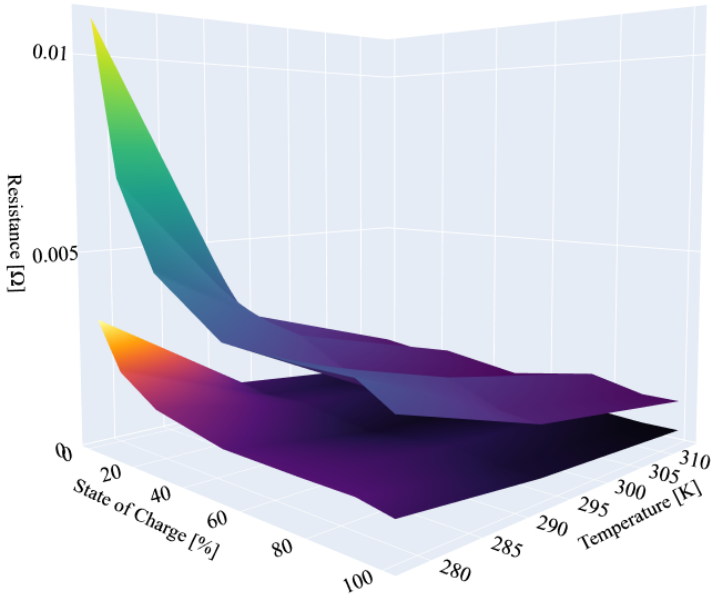


Figure 5.8: Example of lookup table after multiplying R1 with 0.3.

The performance of the BMS becomes evident when examining the voltage plot, as ideally, all cells should exhibit the same voltage. Additionally, the impact on SOC estimation can also serve as evidence of the imbalance among the cells.

Sensor drift

To ensure the safety and optimal performance of batteries, it is imperative to have a dependable sensor fault diagnostic scheme. Sensor drift was implemented by adding an offset to the original voltage. A constant is added to the measured V terminal of cell 3. In case of sensor drift, the BMS can still handle the battery quite well up until some point.

6

Implementation

This chapter describes the implementation of the DT from Matlab to Python, along with the implementation of the optimization-guided falsification elements STL and Bayesian optimization. Furthermore, this chapter presents the implementation of the GUI. To facilitate comprehension, code snippets are included throughout the chapter. However, for convenience, the complete code can be found in a compressed zip folder alongside this thesis. The appendix includes code from one of the test cases and the code of the GUI.

6.1 From Matlab to Visual Studio Code

The model of the battery and BMS was originally a Simulink model in Matlab, but for better universality, Python has been used. The Simulink model has been exported into a standalone Functional Mock-up Unit (FMU) using Functional Mock-up interface (FMI) version 2.0. FMI facilitates the exchange of dynamic simulation models between different programs. By exporting the model as an FMU, Python can simulate the model independently without running Matlab in the background.

To save the Simulink model as an FMU, follow these steps: select ”Save” $\rightarrow$ ”Export Model To” $\rightarrow$ ”Standalone FMU.”

It is important to note that all desired outputs in the standalone FMU must have an output port in the Simulink model before exporting.

The FMU is then opened in Python with the help of the package FMPy [64]. The following steps are involved:

- Starting the FMPy GUI with: Python -m fmpy.gui
- Open an FMU
- Tools → Create Jupyter Notebook

A Jupyter notebook is created, containing all inputs of the FMU as a dictionary and all outputs as another dictionary. Dictionaries are a type of hash table data structure. They consist of key-value pairs, where each value is associated with a specific key. The model is simulated using the *simulate_fmu* function from the FMPy package. The function takes as arguments the filename of the FMU, the start values dictionary, the output values dictionary, and the simulation time.

The results can be plotted by using *plot_result* function from the FMPy package, which utilizes the matplotlib package. Alternatively, the results can be stored in a Pandas DataFrame for further analysis. In addition, specific plotting functions have been developed for this thesis work, such as *plot_result_plotly* and *plot_robustness_plotly*, to improve code cleanliness and visualization.

Code Listing 6.1 and Code Listing 6.2 below demonstrates how the data is rendered in the Jupyter Notebook.

```
import fmpy
from fmpy import *

filename = 'BP_BMS_Passive_Cell_Balancing_SOC_Charging.fmu'

start_values = {
    # variable                          start   unit      description
    'Cell_1_C1_T[1,1]':            1834.8624,          # Cell_1_C1_T(1,1)
    'Cell_1_C1_T[1,2]':           12413.7931,          # Cell_1_C1_T(1,2)
    'Cell_1_C1_T[1,3]':                30000,          # Cell_1_C1_T(1,3)
    'Cell_1_C1_T[2,1]':            4492.7536,          # Cell_1_C1_T(2,1)
    'Cell_1_C1_T[2,2]':                18750,          # Cell_1_C1_T(2,2)
    'Cell_1_C1_T[2,3]':                32500,          # Cell_1_C1_T(2,3)
    'Cell_1_C1_T[3,1]':           23191.4894,          # Cell_1_C1_T(3,1)
    ...}
```

Code Listing 6.1: Jypiter notebook start part 1

```
output = [
    'SOC_Cell1',                    # SOC_Cell1
    'V_terminal_Cell1',             # V_terminal_Cell1
    'I_Cell1',                      # I_Cell1
    'SOC_Cell2',                    # SOC_Cell2
    'V_terminal_Cell2',             # V_terminal_Cell2
    'I_Cell2',                      # I_Cell2
    'SOC_Cell3',                    # SOC_Cell3
    'V_terminal_Cell3',             # V_terminal_Cell3
    'I_Cell3',                      # I_Cell3
    'SOC_Cell4',                    # SOC_Cell4
    'V_terminal_Cell4',             # V_terminal_Cell4
    'I_Cell4',                      # I_Cell4
    'T_Cell1',                      # T_Cell1
    'SOC_est_Cell1',                # SOC_est_Cell1
    'SOC_est_Cell2',                # SOC_est_Cell2
    'SOC_est_Cell3',                # SOC_est_Cell3
    'SOC_est_Cell4',                # SOC_est_Cell4
    'V_pack',                       # V_pack
    'V_terminal_Cell3_measured',    # V_terminal_Cell3_measured
]
sim_time = 20000

result_org = simulate_fmu(filename, start_values=start_values, output=
    output, stop_time=sim_time)
df_org = pd.DataFrame(result_org)
```

Code Listing 6.2: Jypiter notebook start part 2

When the FMU is imported in Python, all tables are divided into single elements with their own key instead of being as a whole table as they were in Simulink. An example of this is illustrated in Figure 6.1. This makes the handling of the data more challenging in Python. This issue is resolved in the FMI version 3.0. However, this version was not available in Matlab prior to the completion of this master thesis. Therefore, some functions were made to make the testing process simpler. These are presented in Table 6.1.

Function name	Input	Output	Description
sort_start_values	start_values	start_keys, start_values, keys	Groups start values and keys associated with each input parameter. Creates a list of lists, where each sublist corresponds to a specific input parameter and contains the respective values. Similarly, it generates a separate list of lists containing the corresponding keys. For example, one sublist would consist of all the keys related to Cell_1_R1_T. Additionally, the function provides the key for each input parameter as part of its return value.
make_dict	keys, start_values	d	Links a key to each list in the list of lists, similar to the example to the left illustrated in Figure 6.1. This function is added to simplify the function *multiply_by_constant* and to make testing easier.
multiply_by_ constant	d, name, constant	d	Multiplies an input parameter with a given constant. Returns a new dictionary with updated values.
make_start_ values	d, start_key	d2	Rearranges the start values into their original format, allowing the function *simulate_fmu* to be called. This function makes the start values look like the right of Figure 6.1.

Table 6.1: Functions made in this master thesis for handling and altering the data for simulation and testing.

```
'Cell_1_R1_T[1,1]': 0.022893,
'Cell_1_R1_T[1,2]': 0.0060907,
'Cell_1_R1_T[1,3]': 0.0027303,
'Cell_1_R1_T[2,1]': 0.014339,
'Cell_1_R1_T[2,2]': 0.0049875,
'Cell_1_R1_T[2,3]': 0.0024937,
'Cell_1_R1_T[3,1]': 0.0088519,
'Cell_1_R1_T[3,2]': 0.0048968,
'Cell_1_R1_T[3,3]': 0.0024484,
'Cell_1_R1_T[4,1]': 0.0065021,
'Cell_1_R1_T[4,2]': 0.0030598,
'Cell_1_R1_T[4,3]': 0.0019124,
'Cell_1_R1_T[5,1]': 0.0060868,
'Cell_1_R1_T[5,2]': 0.0042423,
'Cell_1_R1_T[5,3]': 0.0025823,
'Cell_1_R1_T[6,1]': 0.0056501,
'Cell_1_R1_T[6,2]': 0.0030819,
'Cell_1_R1_T[6,3]': 0.0018834,
'Cell_1_R1_T[7,1]': 0.0048179,
'Cell_1_R1_T[7,2]': 0.0029252,
'Cell_1_R1_T[7,3]': 0.0018928,
```

```
'Cell_1_R1_T': [0.022893,
                0.0060907,
                0.0027303,
                0.014339,
                0.0049875,
                0.0024937,
                0.0088519,
                0.0048968,
                0.0024484,
                0.0065021,
                0.0030598,
                0.0019124,
                0.0060868,
                0.0042423,
                0.0025823,
                0.0056501,
                0.0030819,
                0.0018834,
                0.0048179,
                0.0029252,
                0.0018928]
```

Figure 6.1: To the left is how the start values are imported in Python, while the right is how they were originally in Simulink.

6.2 Signal temporal logic

To implement the STL component of the optimization-guided falsification process, the stlcg package [37] from GitHub was utilized. Various options were explored, including other Python packages developed by different authors. However, the stlcg package stood out due to its comprehensive documentation and accompanying YouTube video, which facilitated its implementation compared to other packages. Additionally, the decision to utilize a preexisting package instead of writing the code from scratch was driven by efficiency considerations.

The stlcg package provides functions for constructing STL formulas and calculating robustness traces from simulation traces.

calculate_robustness

The function *calculate_robustness* uses functions from stlcg to calculate the robustness of a simulation trace against the STL formula given in Equation 6.3. *calculate_robustness* takes a DataFrame, a key, and the maximum and minimum values as arguments. It calculates the robustness of the simulation trace according to the specified STL formula and adds it to the DataFrame. The function returns an updated DataFrame and the smallest value of the robustness trace.

$$\phi_1 = DataFrame[key] < max \tag{6.1}$$

$$\phi_2 = DataFrame[key] > min \tag{6.2}$$

$$\phi = \Box\phi_1 \wedge \phi_2 \tag{6.3}$$

The simulation traces must be of data type torch tensor for stlcg to calculate the robustness. A torch tensor is a matrix of multi-dimensional size containing elements of a uniform data type [65]. The traces are afterward transformed into pandas DataFrame for more straightforward implementation.

```
def calculate_robustness(dataframe, robustoutput, min, max):

    input = torch.tensor(dataframe[robustoutput], dtype = torch.float,
    requires_grad=False)
    input = torch.reshape(input, (1, len(input), 1))

    val_max = torch.tensor(max, dtype=torch.float, requires_grad=True)
    val_min = torch.tensor(min, dtype=torch.float, requires_grad=True)

    ϕ1 = stlcg.LessThan(lhs=robustoutput, val=val_max)
    ϕ2 = stlcg.GreaterThan(lhs=robustoutput, val=val_min)

    ϕ1 = stlcg.Always(subformula=ϕ1, interval = [0,2])
    ϕ2 = stlcg.Always(subformula=ϕ2, interval = [0,2])

    ϕ = stlcg.And(ϕ1, ϕ2)

    inputs = (input, input)
    pscale = 1      # "pscale" is the scale used for evaluating predicates
    scale = -1      # "scale" is the scale used in the maxish/minish
    function. <0 defaults to the true min/max
    rob = ϕ.robustness_trace(inputs, pscale=pscale, scale=scale)

    rob2 = rob.detach().numpy()
    robreverse = rob2[0,:]
    dataframe["Robustness"] = robreverse
    minval = dataframe["Robustness"].min()

    return dataframe, minval
```

Code Listing 6.3: Calculate robustness

6.3 Bayesian optimization

In conjunction with STL, falsification is performed using Bayesian optimization. A package from GitHub called Bayesian-optimization [8] was utilized to simplify the implementation process. This package was chosen due to its eminent documentation and ease of use. Notably, the package has garnered nearly 7k stars on GitHub, indicating its popularity and reliability. Moreover, it leverages the capabilities of Scipy and Scikit-learn, two widely-used Python packages specifically designed for optimization tasks.

The method begins by instantiating a BayesianOptimization object by specifying a function to be optimized f, and its parameters with their corresponding bounds, pbounds. This

is seen in Code Listing 6.4. The random_state parameter determines the number of random seeds for the Gaussian process initialization. However, in this thesis, the investigation of this parameter has been omitted, and it is consistently set to one in all of the test cases. Although this parameter has not been explored in this thesis, it could potentially impact the optimization process.

```
optimizer = BayesianOptimization(
    f=black_box_function,
    pbounds=pbounds,
    verbose=2, # verbose = 1 prints only when a maximum is observed,
    verbose = 0 is silent
    random_state=1)
```

Code Listing 6.4: Bayesian optimization object

```
def black_box_function(x1,  x2,  x3,  x4):
    s_key, s_value, keys = sort_start_values(start_values)

    d = make_dict(keys, s_value)

    d = multiply_by_constant(d, 'Cell_2_R1_T', x1)
    d = multiply_by_constant(d, 'Cell_2_R2_T', x2)
    d = multiply_by_constant(d, 'Cell_2_C1_T', x3)
    d = multiply_by_constant(d, 'Cell_2_C2_T', x4)

    d = make_start_values(d, s_key)

    result = simulate_fmu(filename, start_values=d, output=output,
    stop_time=sim_time)
    df = pd.DataFrame(result)

    df, minval = calculate_robustness(df,robust_output, min, max)
    res = minval

    return -res

# Bounded region of parameter space
pbounds = { 'x1' : (0.1,600), 'x2' : (0.1, 500),  'x3' : (0.1, 200), 'x4'
    : (0.1, 200)}
```

Code Listing 6.5: Black box function and parameter ranges

The Code Listing 6.5 demonstrates the cell balancing test case. The black_box_function takes four variables as input, which are to be optimized. Inside the function, the functions from Table 6.1 is employed to multiply the input parameters by the optimizer variables, simulate the FMU, and calculate the robustness. The value of robustness guides the optimization process.

It is important to note that the aim is to find the smallest value of the robustness, indicating a higher violation of the STL. However, the optimization process is set up as a maximization task. To address this, the function returns the negative value of the robustness, effectively transforming the minimization problem into a maximization problem.

Subsequently, in Code Listing 6.6, the utility function is set to UCB. Kappa is used for exploration and exploitation, explained in section 4.4. The parameter xi is not necessary for the utility function UCB and is set to zero. The BayesianOptimization object allows probing to guide the optimization. By default, these points are lazily explored by setting lazy=True, meaning they will be evaluated only when the maximize function is called next. This probing process occurs before the Gaussian process takes control. As the parameters are initially multiplied by one, all parameters are probed to have a value of 1. This is done in Code Listing 6.6.

```
utility = UtilityFunction(kind= "ucb", kappa=2, xi=0.0)

optimizer.probe(
    params={'x1' : 1, 'x2' : 1, 'x3' : 1, 'x4' : 1},
    lazy=True,
)
```

Code Listing 6.6: Utility function and probing

Lastly, the function maximize can be called. Shown in Code Listing 6.7. In maximize, two parameters need to be set:

- n_iter: How many steps of Bayesian optimization to perform. The more steps, the more likely the method finds a reasonable maximum.
- init_points: How many steps of random exploration to perform. Random exploration can help by diversifying the exploration space.

The maximize function generates step-wise output during its execution, which is displayed in Figure 6.2. Currently, the implementation does not include a stopping condition for when an optimum is reached. Consequently, the optimizer continues running until the

maximum number of iterations is exhausted. In practice, the optimizer may discover progressively improved optima or converge to a single optimum, depending on the characteristics of the graph.

```
optimizer.maximize(
    init_points=5,
    n_iter=2,
)
```

Code Listing 6.7: Maximize function

iter	target	x1	x2	x3	x4
1	0.4633	1.0	1.0	1.0	1.0
2	0.8188	250.3	360.2	0.1229	60.54
3	0.4426	88.14	46.26	37.33	69.18
4	0.4755	238.1	269.5	83.9	137.1
5	0.5058	122.8	439.1	5.575	134.1
6	0.486	250.4	279.4	28.16	39.7
7	0.4537	4.731	81.01	141.9	153.0
8	0.4635	265.6	206.8	109.6	148.9

Figure 6.2: Screenshot of the output of the maximize function. The pink row indicates the last optimal point found.

6.4 Graphical user interface

To improve user experience and facilitate visualization, the code has been integrated into a GUI using Dash. Dash is a Python framework developed by Plotly designed explicitly for creating interactive web applications. With its powerful capabilities, Dash enables the construction of interactive Python dashboards and provides convenient figure-plotting functionalities. It utilizes a Python-based language resembling HTML, making it user-friendly and well-documented.

6.4.1 HTML

Layout

The HTML code is implemented in the file app.py. Here the layout functions of the dashboard are set. The app.layout function is utilized to arrange and position all the components within the HTML structure. The layout is sectioned using the html.Div function, which divides the pages into sections. The layout of the dashboard is visualized in Figure 6.3, and the code relating is in Code Listing 6.8.

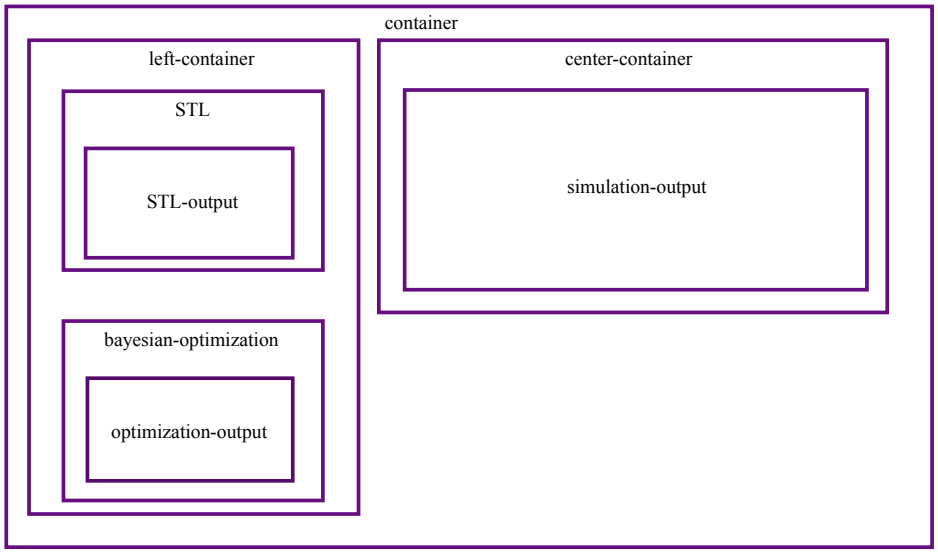


Figure 6.3: HTML layout

```
app.layout = html.Div([
    html.Div([
            html.Div([ ...
                    html.Div([], id = 'STL-output')
                    ], id = 'STL', ),
            html.Div([ ...
                    ], id = 'bayesian-optpimization'),
            html.Div([...
                    ], id = 'optimization-output'),
            ], id = 'left-container'),

    html.Div([
            html.Div([], id = 'simulation-output'),
            ], id = 'center-container'),
], id = 'container')
```

Code Listing 6.8: Code for HTML layout

Inside each html.Div other HTML components can be put, for example, a button or a header. In addition to the basic HTML components, Dash has some of their own called Dash Core Components(DCC). For instance, dcc.Input is used to gather user input. This component plays a crucial role in supporting callbacks, a vital function in Dash applications.

Callbacks

Updates are made using callbacks. They are implemented as functions that are triggered whenever there is a change in the property of an input component. Their purpose is to update a corresponding property in another component, which acts as the output.

To create a callback in Plotly Dash, we use the app.callback function. This function defines the inputs and outputs of the callback, while the associated function specifies the actions to be performed based on the input and output.

The Code Listing 6.9 shows an example from the GUI where the STL formula is displayed and updated. The inputs for this callback include a dropdown for selecting the output parameter of the DT, as well as the minimum and maximum values for the STL formula.

The output produced by Code Listing 6.9 will be placed in the designated html.Div element in the app.layout with the id ”STL-output” in Code Listing 6.10.

```
@app.callback(
    Output('STL-output', 'children'),
    Input('STLdropdown', 'value'),
    Input('STLmin', 'value'),
    Input('STLmax', 'value')
)
def update_STL(STLdropdown, STLmin, STLmax):
    return html.Div(["ϕ1 = {} is less than {}".format(STLdropdown,STLmax),
            html.Div(html.P([ html.B()])),
            "ϕ2 = {} is more than {}".format(STLdropdown, STLmin),
            html.Div(html.P([ html.B()])),
            "ϕ = Always  ϕ1 And ϕ2"])
```

Code Listing 6.9: Callback of STL formula

```
app.layout = html.Div([
    html.Div([
        html.Div([
            html.H2("STL"),
            dcc.Dropdown(['SOC_Cell1', 'V_terminal_Cell1', 'I_Cell1', '
    SOC_Cell2', 'V_terminal_Cell2', 'I_Cell2', 'SOC_Cell3', '
    V_terminal_Cell3', 'I_Cell3', 'SOC_Cell4', 'V_terminal_Cell4', '
    I_Cell4', 'T_Cell1', 'SOC_est_Cell1','SOC_est_Cell2', 'SOC_est_Cell3',
     'SOC_est_Cell4', 'V_pack', 'V_terminal_Cell3_measured'],
                        'V_terminal_Cell2',
                        clearable = False,
                        id = 'STLdropdown'),
            html.Div(html.P([ html.B()])),
            "Max:",dcc.Input('5', type = 'number', step = 0.01, id = '
    STLmax', style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            "Min:",dcc.Input('4', type = 'number', step = 0.01, id = '
    STLmin', style = {'width': '15%'}),
            html.Div(html.P([ html.Br()])),
            html.Div(id = 'STL-output')

        ],id = 'STL',  style = {'display': 'inline-block'}),

        ...])
```

Code Listing 6.10: app.layout of GUI

6.4.2 CSS

To style the GUI, a Cascading Style Sheet (CSS) is used. CSS is a style sheet language that defines the presentation and formatting of an HTML document. Its primary purpose is to enhance the visual appeal of the GUI and arrange elements in an aesthetically pleasing manner. CSS achieves this by specifying various properties, such as padding, which creates space around components, and sizing, which determines the dimensions of elements. CSS is used to place and size the html.Div elements on the page.

The provided Code Listing 6.11 below demonstrates the application of CSS to the optimize button with the id ”opt_btn”. It showcases how the CSS code defines the button’s size, color, hover effect, and other visual aspects.

The full CSS code can also be found in the attached zip folder or in section B

```
#opt_btn{
  width: 120px;
  height: 50px;
  cursor: pointer;
  border: 0px;
  border-radius: 5px;
  background-color:#a54cee98;
  color: rgb(255, 255, 255);
  text-transform: uppercase;
  font-size: 15px;
  border: 2px solid;
  border-color: #a54cee98;

}
#opt_btn:hover{
  background-color: #a54cee98;
}
```

Code Listing 6.11: CSS code of optimization button

7

Result

This chapter presents each test case with the input values chosen and the plots of the system outputs. Each optimization-guided falsification test cases present the Maximize function parameters and intervals. Tables show the optimization's complete output, where the pink rows indicate the last found optimum. Note that the output is a positive value but negative in the plots because the optimizer solves a maximization problem while, in practice, it is a minimization as mentioned in chapter 6. The manual test cases chose the inputs randomly.

	DT1: Battery		DT2: Battery and BMS	
Test cases	Manual	Optimization-guided falsification	Manual	Optimization-guided falsification
Internal short circuit	x	x		x
Cell balancing			x	x
Sensor drift				x

Table 7.1: Overview of test cases in the result section

7.1 Original simulation

These plots show the original simulation of the DTs with the original input values. The simulation time is 25000 seconds.

7.1.1 DT1: Battery

A pulse is sent to the current in the battery to show the behavior of the battery through charge and discharge.

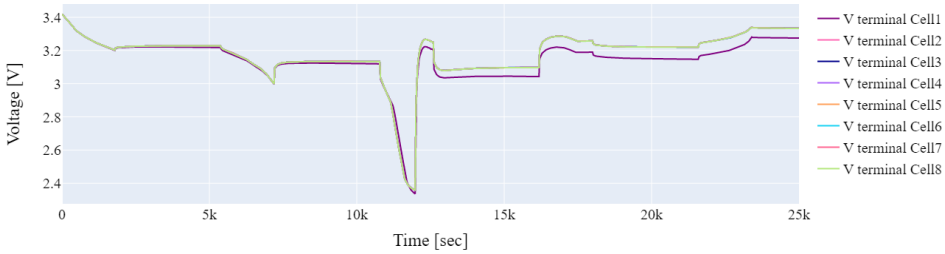


Figure 7.1: Voltage of cells

Figure 7.1 shows the voltages of each cell. During charge, the voltages decrease. Figure 7.1 highlights a notable distinction between the majority of the voltages at the battery terminals and the voltage observed in cell 1. This discrepancy can be attributed to substantially different internal parameters within that particular cell. This observation underscores the reliance on internal models for computing simulation iteration results. It suggests that the internal characteristics of individual cells play a crucial role in determining the overall behavior and performance of the battery system.

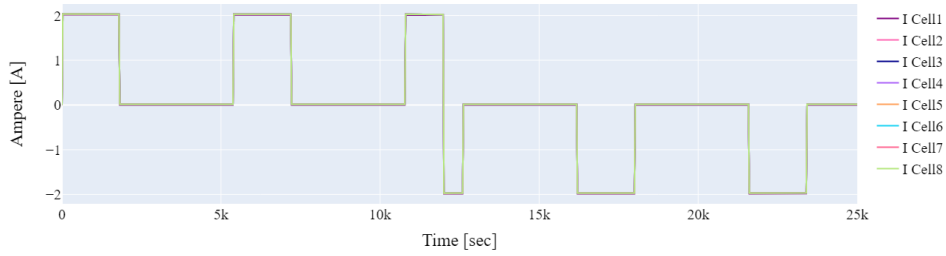


Figure 7.2: Current of cells

Figure 7.2 displays the current of the battery. A pulse of current is sent to the battery. The first half of the plot shows the current going out and discharging the battery. Then the second half shows the recharging of the battery.

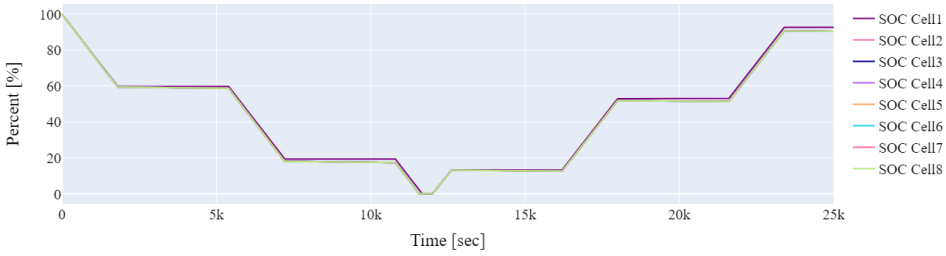


Figure 7.3: SOC of cells

Figure 7.3 shows how the SOC of the cells is full at the beginning. When the battery is discharging, the value of SOC goes down. When it is neither charging nor discharging, the SOC stands still. The SOC fluctuates within the range of 0% to 100%.

7.1.2 DT2: Battery and BMS

In this simulation, the battery is discharged and charged over and over. The simulation time is 20,000 seconds.

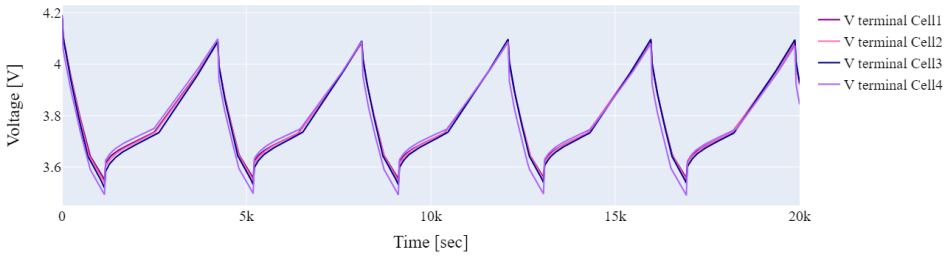


Figure 7.4: Voltage of cells

In Figure 7.4, cell voltages are shown. Initially, the cells are similar. During discharge, their voltages gradually decrease. When fully charged, cell voltages peak at around 4.1V, while when fully discharged, they reach around 3.5V.

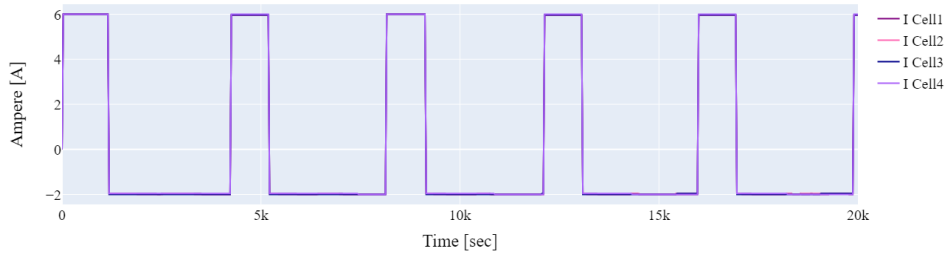


Figure 7.5: Current of cells

In Figure 7.5, the currents of each cell are presented. Compared to DT1 in Figure 7.2, a higher amplitude of the current is injected. In DT1, the current only goes up to 2, whereas in DT2, it goes up to 6.

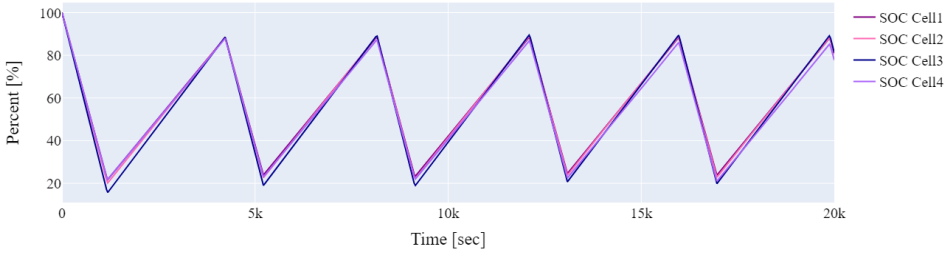


Figure 7.6: SOC of cells

In Figure 7.6, the SOC of each cell is presented. Initially, there is no cell degradation, and all the cells have a similar SOC. The SOC fluctuates within the range of 10% to 90%.

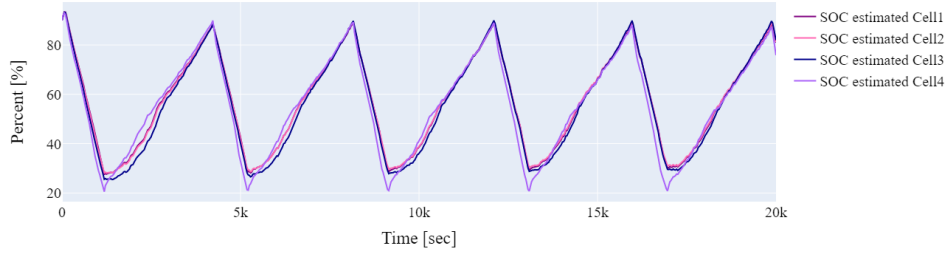


Figure 7.7: SOC estimated by BMS

In Figure 7.7, the SOC of each cell estimated by the BMS using an extended Kalman filter

is displayed. It closely resembles Figure 7.6, indicating that the BMS effectively estimates the SOCs of the cells.

7.2 DT1: Battery

7.2.1 Manual testing

Before implementing the BMS, testing was conducted on a DT of only the battery. This testing aimed to detect and analyze the battery's behavior in various failure scenarios, providing valuable insights into its performance. By thoroughly assessing the battery's response, the testing phase served as a validation process for the DT, ensuring its accuracy and reliability.

Internal short circuit

In this test case, the input parameter Cell_2_R_ISCR is multiplied by 0.04, which should lead to ISC. The simulation time is 20,000 seconds.

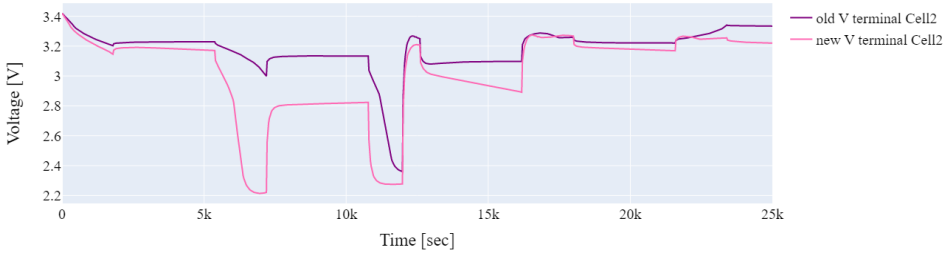


Figure 7.8: Voltage of cell 2

In Figure 7.8, the voltage of cell 2 is plotted both before and during an ISC event. During the ISC, the voltage of cell 2 drops significantly lower compared to its original voltage, indicating the impact of the internal short circuit on the cell's voltage level.

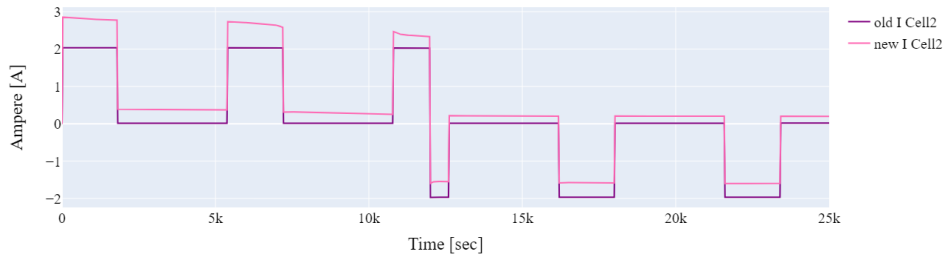


Figure 7.9: Current of cell 2

In Figure 7.9, it appears that there is a fault in the DT model. Normally, during an ISC, the current in the affected cell should be lower compared to the original scenario without an ISC. However, if the current in the cell with an ISC is shown to be higher than the original, it indicates a fault in the DT modeling.

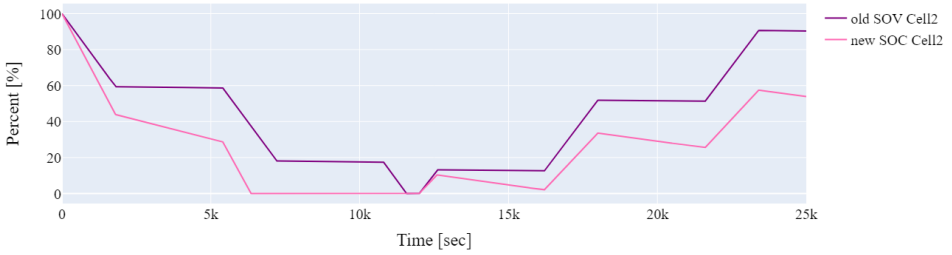


Figure 7.10: SOC of cell 2

In Figure 7.10, the SOC of cell 2 is presented before and after experiencing an ISC. The new SOC of cell 2 is noticeably lower than the old SOC, indicating a significant decrease in its capacity. The inability of the SOC to reach 100% after the ISC suggests degradation of this particular cell. This degradation could be attributed to the internal short circuit and highlights the impact of such events on the overall battery performance.

7.2.2 Optimization-guided falsification

The DT of only the battery served as an initial model for implementing optimization-guided falsification. In this context, a cell balancing test case was conducted. Without the BMS in place, there is no cell balancing.

Cell balancing

STL:

$$\phi_1 = V_{terminal}cell_2 < 5 \tag{7.1}$$

$$\phi_2 = V_{terminal}cell_2 > 3 \tag{7.2}$$

$$\phi = \Box\phi_1 \wedge \phi_2 \tag{7.3}$$

The value of the voltage of cell 2 should always be between 3 and 5.

Bayesian optimization:

κ	2
init_points	5
n_ iter	2

Table 7.2: Maximize function parameters

Input variable	Parameter	Interval
Cell_2_R1_T	x_1	$(0.1, 6)$
Cell_2_R2_T	x_1	$(0.1, 5)$
Cell_2_C1_T	x_1	$(0.1, 2)$
Cell_2_C2_T	x_1	$(0.1, 20)$

Table 7.3: Intervals for variables

Iteration	Target	x1	x2	x3	x4
1	0.64	1.0	1.0	1.0	1.0
2	0.8443	2.56	3.63	0.1002	6.116
3	0.5404	0.9659	0.5525	0.4539	6.977
4	0.7851	2.441	2.74	0.8965	13.74
5	0.5929	1.306	4.403	0.152	13.44
6	0.8688	2.562	2.838	0.3667	4.042
7	0.9219	2.912	3.807	0.1762	4.762
8	1.308	5.115	3.684	0.1	4.484

Table 7.4: Output of optimization

By applying a probe at one point and setting the number of iterations (n_iter) to two, along with five initial points (init_points), the Bayesian optimization process runs for a total of eight iterations until an optimal solution is found. In this optimization, the Kappa value is set to two, favoring exploitation over exploration.

The input variables Cell_2_R1_T, Cell_2_R2_T, Cell_2_C1_T, and Cell_2_C2_T are multiplied by the parameters to be optimized, denoted as x_1, x_2, x_3, and x_4, respectively.

The intervals for the parameters are: $x_1 \in (0.1, 6)$, $x_2 \in (0.1, 5)$, $x_3 \in (0.1, 2)$, and $x_4 \in (0.1, 20)$.

By multiplying Cell_2_R1_T by 5.115, Cell_2_R2_T by 3.684, Cell_2_C1_T by 0.1, and Cell_2_C2_T by 4.484, the lowest obtained robustness score is -1.308.

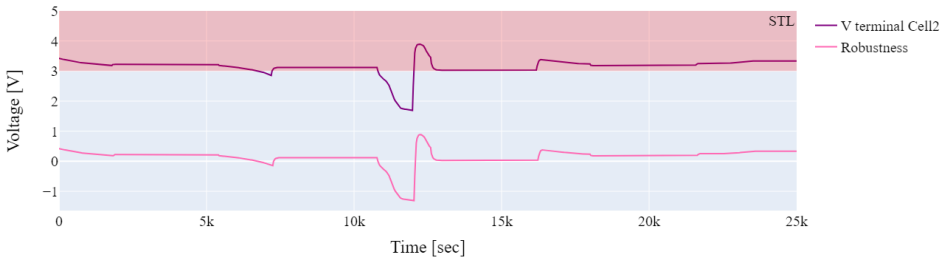


Figure 7.11: Robustness of voltage of cell 2

In Figure 7.11, the voltage of cell 2 is presented after the optimization process, along with its corresponding robustness value. The simulation trace generally adheres to the STL specification for the majority of the time but deviates below the desired threshold after approximately 11,000 seconds.

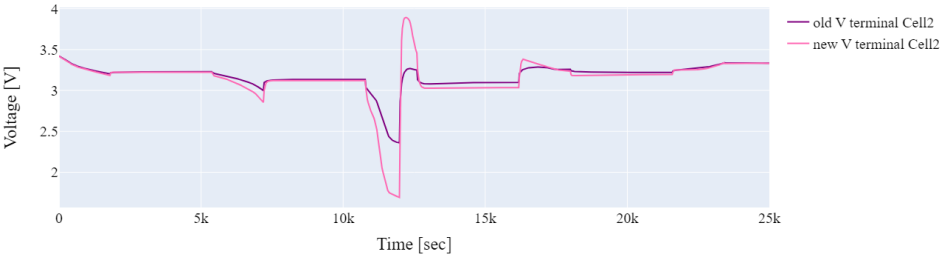


Figure 7.12: Voltage of cell 2

In Figure 7.12, the voltage behavior of cell 2 is displayed, showcasing a comparison between the new and old voltages. By modifying the values of resistors R1 and R2, as well as capacitors C1 and C2, the voltage behavior of the cell has been altered. These changes in component values have influenced the voltage response, resulting in a different voltage behavior compared to the original scenario.

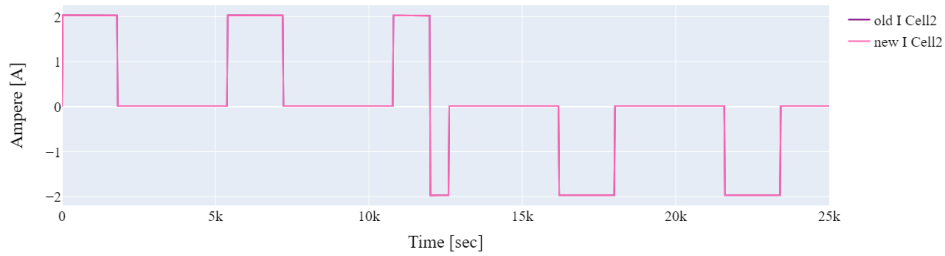


Figure 7.13: Robustness of voltage of cell 2

The new and old currents of cell 2 remain identical in Figure 7.13, indicating that the changes in component values do not affect the current behavior in this particular test case.

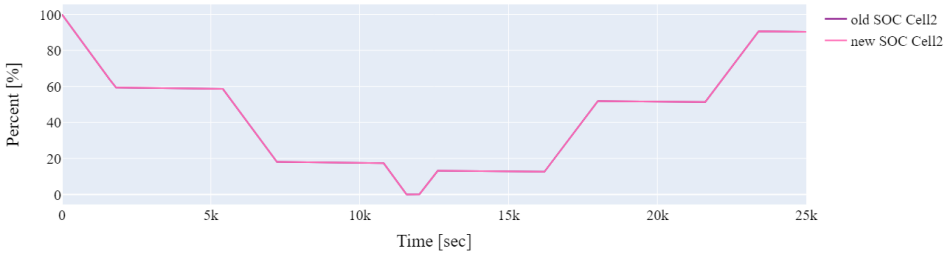


Figure 7.14: Robustness of voltage of cell 2

The new and old SOC values of cell 2 also remain identical. The changes in component values have no influence on the SOC in this particular test case, as seen in Figure 7.14.

7.3 DT2: Battery and BMS

The rest of the test cases is on the DT of both the battery and BMS. The simulation time is 20,000 seconds.

7.3.1 Manual testing

During the manual testing phase, various values were examined to assess their impact on the battery and BMS. One example of such testing is provided below. In manual testing, robustness calculations were not performed. The primary objective of manual testing was to observe how different inputs affected the outputs of the battery and BMS.

Cell balancing

Below Table 7.5 indicating the values multiplied by their corresponding constants. In the given test case, the simulation time is set to 25,000 seconds.

Input variable	Constant
Cell_2_R1_T	0.225
Cell_2_R2_T	200.34
Cell_2_C1_T	40.25
Cell_2_C2_T	0.054

Table 7.5: Constant for input

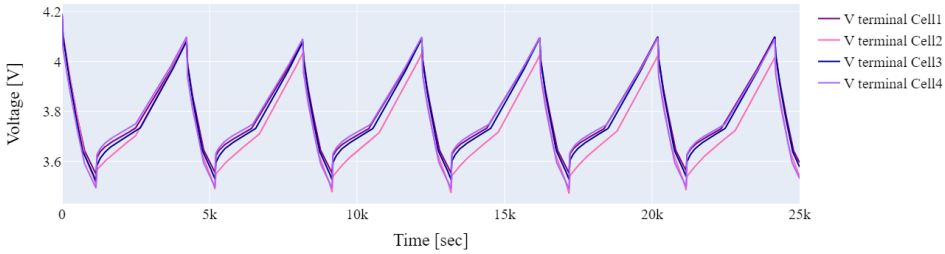


Figure 7.15: Voltage of cells

Upon reviewing Figure 7.15, it can be observed that the voltage of cell 2 deviates from the other cells. This discrepancy indicates a difference in the voltage behavior of cell 2 compared to the rest of the cells during the simulation.

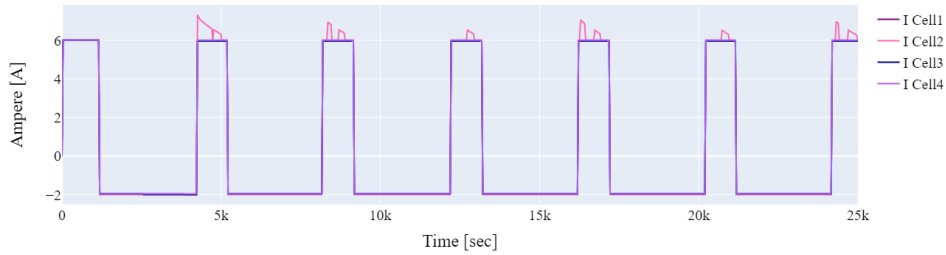


Figure 7.16: Current of cells

The current plot displayed in Figure 7.16 exhibits unusual behavior, suggesting that the inputs provided may not accurately reflect realistic conditions. The parameters R1, R2,

C1, and C2 have realistic values within a specific range. However, in this case, it appears that the values used for these parameters may fall outside of these ranges. This deviation could be attributed to the use of a relatively high multiplication factor.

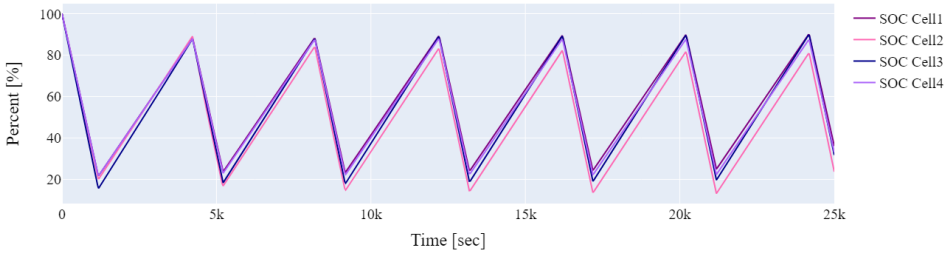


Figure 7.17: SOC of cells

In Figure 7.17, it can be observed that after 20,000 seconds, the SOC of cell 2 is slightly shifted to the right compared to the other cells. Additionally, the interval of SOC values for cell 2 appears to be lower than that of the other cells.

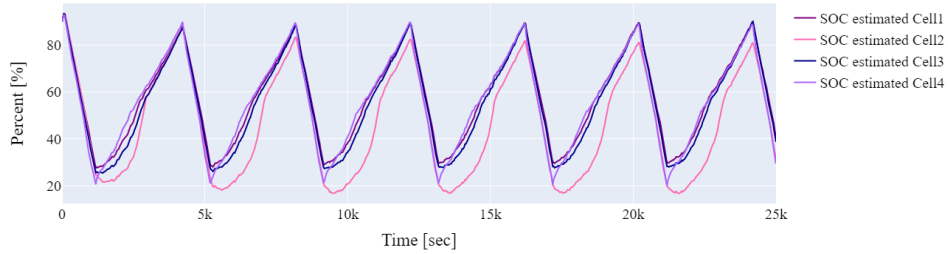


Figure 7.18: SOC of cells estimated by BMS

In Figure 7.18, it is evident that the BMS effectively estimates the SOC of the cells, including cell 2. Despite the differing SOC behavior of cell 2, the BMS accurately captures and reflects this behavior in its estimations.

7.3.2 Optimization-guided falsification

In this section, the multiplication constants were determined using optimization-guided falsification techniques. The simulation time is 20,000 seconds.

Cell balancing

STL:

$$\phi_1 = V_{terminal} cell_2 < 5 \tag{7.4}$$

$$\phi_2 = V_{terminal} cell_2 > 4 \tag{7.5}$$

$$\phi = \Box \phi_1 \wedge \phi_2 \tag{7.6}$$

The value of the voltage of cell 2 should always stay between 4 and 5.

Bayesian optimization:

κ	2
init_points	5
n_iter	2

Table 7.6: Maximize function parameters

Input variable	Parameter	Interval
Cell_2_R1_T	x_1	$(0.1, 6)$
Cell_2_R2_T	x_2	$(0.1, 5)$
Cell_2_C1_T	x_3	$(0.1, 2)$
Cell_2_C2_T	x_4	$(0.1, 20)$

Table 7.7: Intervals for variables

By probing one point and setting n_iter to two and init_points to five, the Bayesian optimization iterates for a total of eight iterations as shown in Table 7.8. The value of Kappa was set to two, favoring exploitation in the optimization process.

The input variables Cell_2_R1_T, Cell_2_R2_T, Cell_2_C1_T, and Cell_2_C2_T were multiplied by their corresponding parameters to be optimized: x_1, x_2, x_3, and x_4, respectively. The parameter intervals for these variables were defined as follows: $x_1 \in (0.1, 6)$, $x_2 \in (0.1, 5)$, $x_3 \in (0.1, 2)$, and $x_4 \in (0.1, 20)$.

During the optimization process, three instances of worse robustness scores were found, indicated by pink highlights in Table 7.8. The last instance was discovered by multiplying Cell_2_R1_T with 5.372, Cell_2_R2_T with 5.0, Cell_2_C1_T with 0.1, and Cell_2_C2_T with 4.96. This particular combination yielded the lowest robustness score of -0.5276 among the optimization iterations.

Iteration	Target	x1	x2	x3	x4
1	0.4633	1.0	1.0	1.0	1.0
2	0.504	2.56	3.63	0.1002	6.116
3	0.4616	0.9659	0.5525	0.4539	6.977
4	0.4901	2.441	2.74	0.8965	13.74
5	0.4713	1.306	4.403	0.152	13.44
6	0.4997	2.562	2.838	0.3667	4.042
7	0.519	3.118	4.159	0.117	4.876
8	0.5276	5.372	5.0	0.1	4.96

Table 7.8: Output of optimization

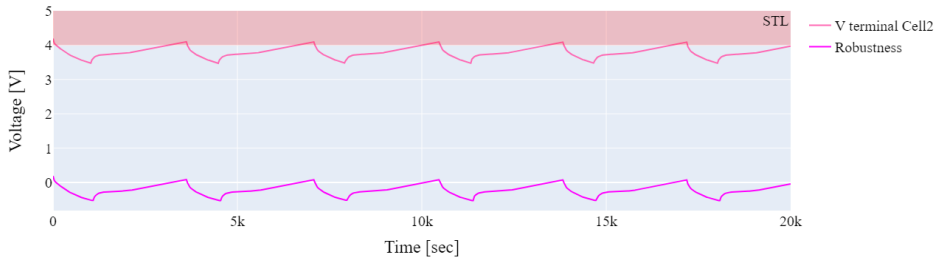


Figure 7.19: Robustness of voltage of cell 2

In Figure 7.19, the allowed STL region is indicated by the red section on the y-axis, spanning from 4 to 5. It is evident that the voltage of cell 2 consistently violates the STL criteria, as it frequently falls outside the designated red region. This violation can be observed in the voltage plot of cell 2, where the values predominantly lie outside the specified range. Additionally, the robustness plot consistently remains below zero, further indicating the non-compliance of cell 2 with the desired STL constraints.

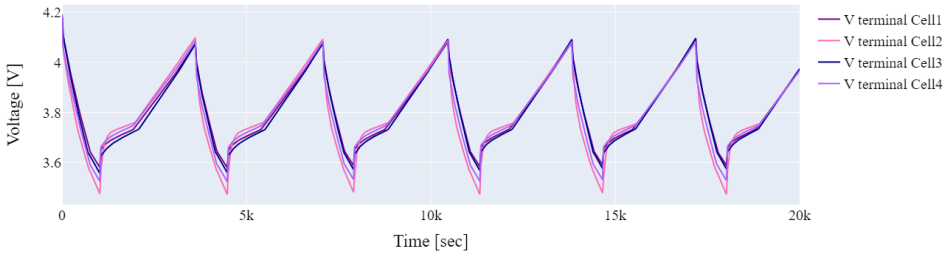


Figure 7.20: Voltage of cells

In Figure 7.20, the voltage behavior of cell 2 is noticeably lower compared to the other cell voltages. This discrepancy is attributed to the fact that the STL constraints are consistently violated by cell 2, leading to attempts to adjust the voltage plot in an effort to bring it outside the STL region. As a result, the plots are drawn down, resulting in a worse robustness score. Despite finding a test case that does not exhibit the worst behavior, the adjustments made to meet the STL criteria ultimately lead to a decreased robustness score.

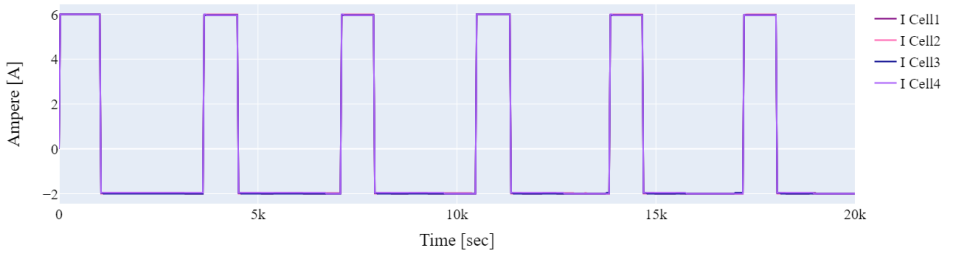


Figure 7.21: Current of cells

The currents of the cells, particularly cell 2, are not significantly affected by the new inputs for R1, R2, C1, and C2. This can be observed in the plot shown in Figure 7.21, which remains similar to the original plot displayed in Figure 7.5.

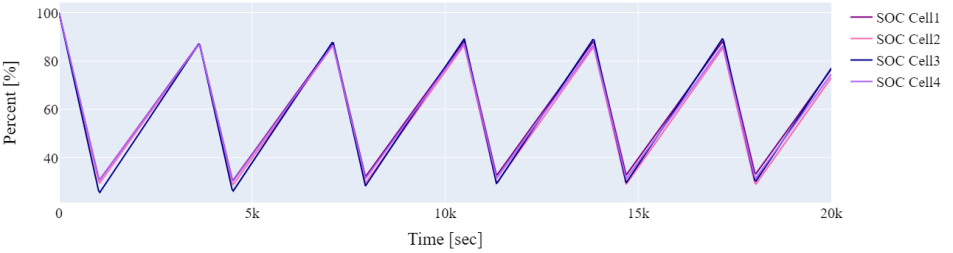


Figure 7.22: SOC of cells

In Figure 7.22, the SOC of cell 2 exhibits some variation compared to the other cells, but overall, the plot closely resembles the original simulation plot shown in Figure 7.6.

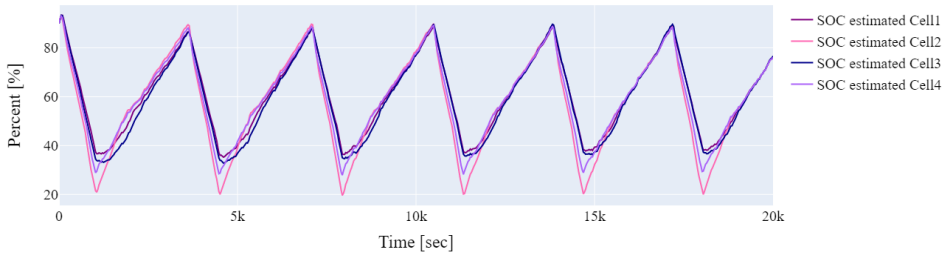


Figure 7.23: SOC of cells estimated by BMS

In Figure 7.23, the SOC estimation by the BMS for cell 2 appears to be lower compared to the SOC values observed in Figure 7.22.

The test case conducted in this scenario was not particularly extreme, suggesting that there may be more room for the optimization algorithm to explore different possibilities. For example, by setting Kappa to more exploration or making the intervals for x_1, x_2, x_3, and x_4 larger, or by increasing the number of iterations.

Internal short circuit

In this particular test case, the focus is on evaluating the effects of two parameters: the internal resistance of cell 1, Rs, and the internal short-circuit resistance, R_ISCR. The simulation time is 20,000 seconds.

STL:

$$\phi_1 = V_{terminal}cell1 < 5 \tag{7.7}$$

$$\phi_2 = V_{terminal}cell1 > 4 \tag{7.8}$$

$$\phi = \Box\phi_1 \wedge \phi_2 \tag{7.9}$$

The value of the voltage of cell 1 should always stay between 4 and 5.

Bayesian optimization:

κ	5
init_points	5
n_iter	4

Table 7.9: Maximize function parameters

Input variable	Parameter	Interval
Cell_1_Rs_T	x_1	$(2, 5)$
Cell_1_I_R_ISCR_T	x_1	$(0.02, 1)$

Table 7.10: Intervals for variables

In this test case example, Bayesian optimization is performed with the following configu-

ration: probing one initial point and setting the number of iterations (n_iter) to four, along with five initial points (init_points). The optimization process consists of a total of 10 iterations, and the results are presented in Table 7.11.

In this optimization, a value of Kappa is set to 5, indicating a balance between exploration and exploitation. The input variables, Cell_1_Rs_T and Cell_1_I_R_ISCR_T, are multiplied by optimization parameters x_1 and x_2, respectively. The parameter intervals for x_1 and x_2 are defined as $x_1 \in (2, 5)$ and $x_2 \in (0.02, 1)$.

In this particular test case, a lower robustness is found three times, but subsequently, the optimization fails to find a lower value. The final point obtained has a robustness score of -0.5474, achieved by multiplying Cell_1_Rs_T with 5.0 and Cell_1_I_R_ISCR_T with 0.497.

Table 7.11: Output of optimization

Iteration	Target	x1	x2
1	0.4518	1.0	1.0
2	0.5045	3.251	0.7259
3	0.4885	2.0	0.3163
4	0.4871	2.44	0.1105
5	0.4913	2.559	0.3586
6	0.5023	3.19	0.548
7	0.5412	4.634	0.8239
8	0.5496	5.0	0.02
9	0.5489	5.0	1.0
10	0.5474	5.0	0.497

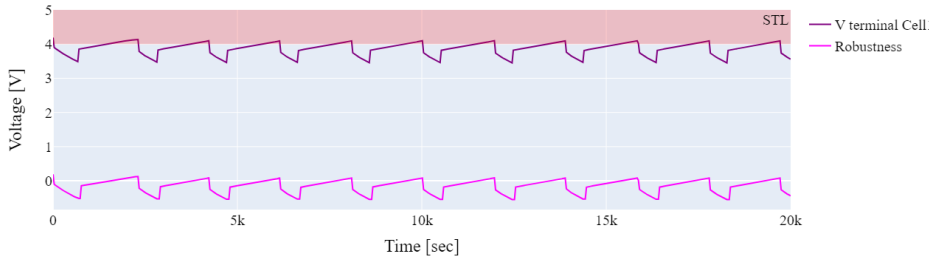


Figure 7.24: Robustness of voltage of celltwo

In Figure 7.24, the red interval from four to five on the y-axis represents the STL interval. The dark purple line represents the voltage of cell 1. It can be observed that only the peaks of the graph fall within the STL interval, while the majority of the data points lie

outside of it. The robustness line, indicated by the purple line, consistently remains below zero except for the peaks, indicating a significant violation of the STL and poor robustness performance.

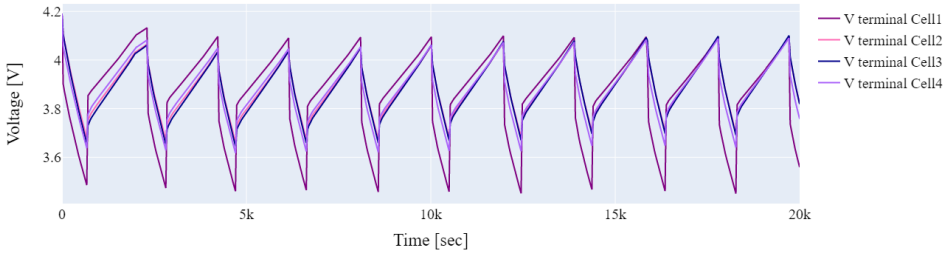


Figure 7.25: Voltage of cells

In Figure 7.25, it can be observed that the voltage of cell 1 experiences a larger drop compared to the rest of the cells. This behavior is attributed to the presence of an ISC.

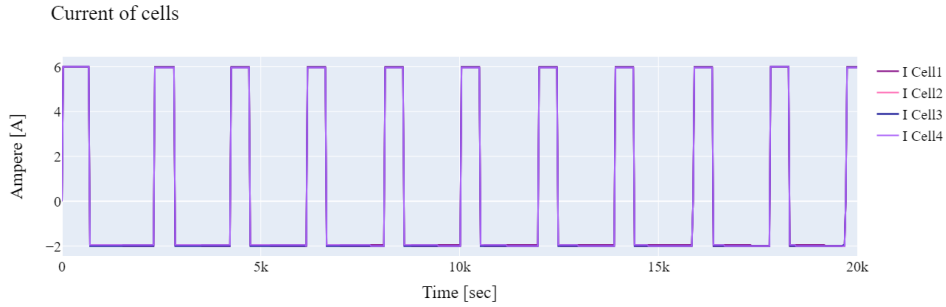


Figure 7.26: Current of cells

In Figure 7.26, the effects of the ISC can be observed. The frequency of charge and discharge cycles is higher compared to the original simulation. This is due to the presence of the ISC. The ISC provides an additional discharge path, leading to accelerated energy loss and a higher demand for recharging to compensate for the reduced capacity.

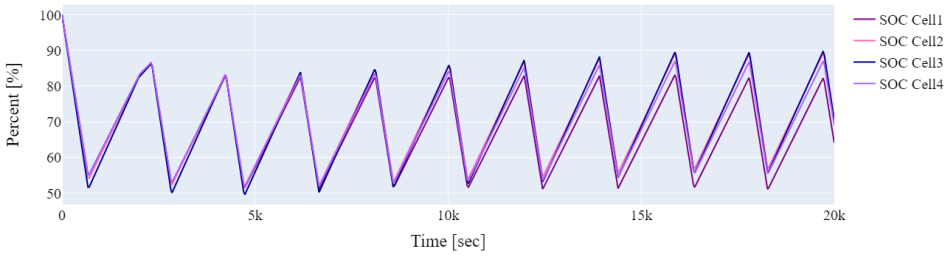


Figure 7.27: SOC of cells

Because of the internal short circuit, the SOC does not fluctuate between 10% and 90%, only between 50% and 90%. The whole battery pack is affected by the ISC. Due to the presence of the ISC, the SOC of the battery cells is affected in Figure 7.27.

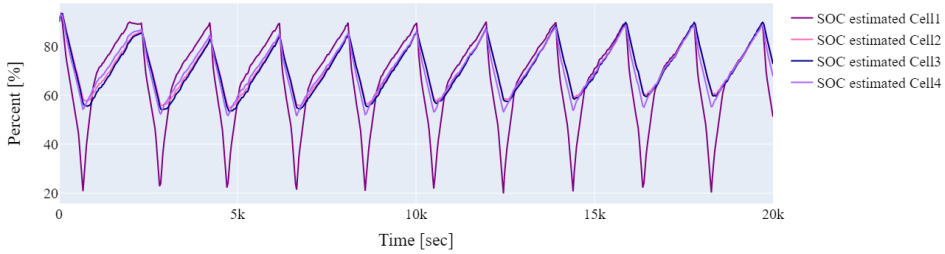


Figure 7.28: SOC of cells estimated by BMS

In Figure 7.28, the estimation of the SOC of cell 1 is significantly different from the actual SOC shown in Figure 7.27. The BMS utilizes the SOC values of the highest and lowest cells to determine the charging and discharging thresholds. As a result, the battery's performance is compromised, and its ability to deliver its full capacity and power is diminished.

Sensor drift

The last test case shows the sensor drift. The simulation time is 20,000 seconds.
STL:

$$\phi_1 = V_{terminal}Cell3_{measured} < 4.08 \tag{7.10}$$

$$\phi_2 = V_{terminal}Cell3_{measured} > 3.5 \tag{7.11}$$

$$\phi = \Box\phi_1 \wedge \phi_2 \tag{7.12}$$

The value of the measured voltage of cell three should always stay between 3.5 and 4.08.

Bayesian optimization:

κ	10
init_points	5
n_iter	10

Table 7.12: Maximize function parameters

Input variable	Parameter	Interval
V_cell3_offset	x_1	$(0.2, 7)$

Table 7.13: Intervals for variables

By probing one point and setting n_iter to ten and init_points to five, a total of 16 iterations of Bayesian optimization were performed in this test case, as shown in Table 7.14. The value of Kappa was set to 10, indicating a preference for exploration during the optimization process.

In this specific test case, the input variable V_cell3_offset was multiplied by a parameter x_1 in order to optimize its value. The parameter interval for x_1 was set to $(0.2, 7)$.

During the optimization process, lower robustness values were found three times. The last point with a robustness value of -0.3175 was obtained by multiplying V_cell3_offset with a value of 5.367. It is important to note that in all other test cases, V_cell3_offset was set to zero, but in this particular test case, it was initially set to 0.01 before the multiplication, resulting in an offset of 0.05367.

The original robustness value was 0.3039, and the optimizer was able to find worse robustness values through the optimization process.

Iteration	Target	x1
1	0.3039	1.0
2	0.3013	3.036
3	0.3011	5.098
4	0.2981	0.2008
5	0.3046	2.256
6	0.2994	1.198
7	0.3175	5.302
8	0.3006	5.847
9	0.3046	2.256
10	0.3175	5.367
11	0.3171	5.519
12	0.317	6.998
13	0.3174	6.812
14	0.3174	6.57
15	0.3005	6.343
16	0.3118	4.035

Table 7.14: Output of optimization

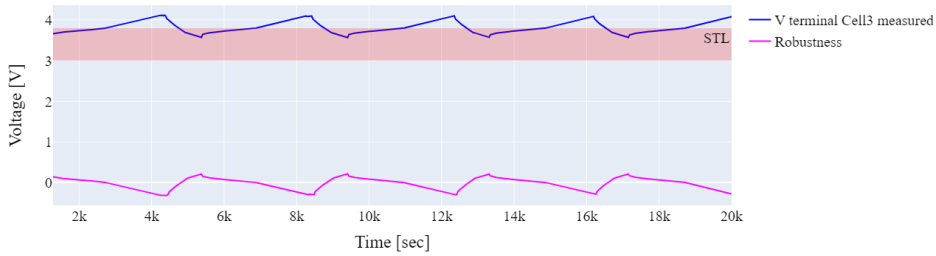


Figure 7.29: Robustness of measured voltage of cell 3

In Figure 7.29, The peaks in the graph of the voltage of cell 3 exceed the STL criteria. This violation of the STL results in negative robustness values at those specific points in time.

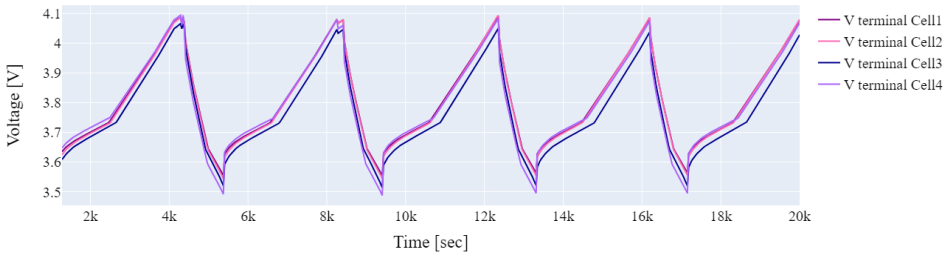


Figure 7.30: Voltage of cells

In Figure 7.30, the voltages of the cells are plotted. It is observed that the voltage of cell 3 deviates slightly from the voltages of the other cells.

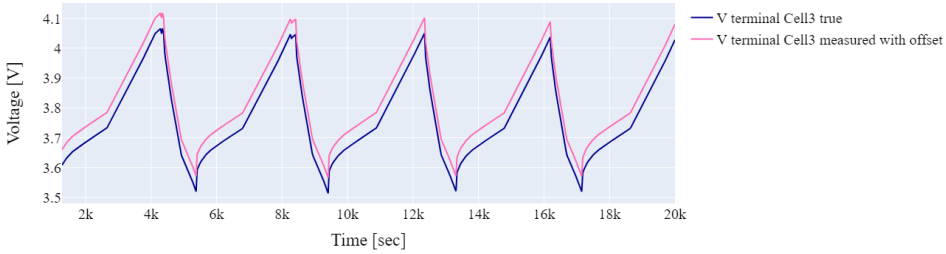


Figure 7.31: Voltage of cell 3: true and measured

In Figure 7.31, the real voltage of cell three is plotted alongside the measured voltage with the offset. It is observed that the offset is slightly higher than the real voltage of cell 3. This means that the BMS perceives the voltage of cell 3 to be higher than its actual value. As a result, the BMS may provide less charge to cell 3, leading to a lower voltage compared to the other cells.

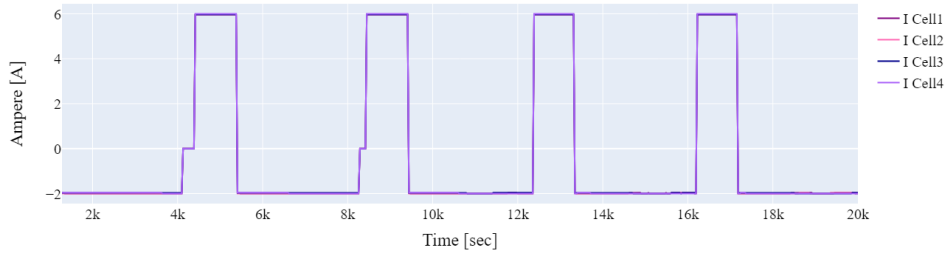


Figure 7.32: Current of cells

The current is not significantly affected by the sensor drift. In Figure 7.32, the plot of the current fluctuates between 0 and 6 as in the original simulation Figure 7.5.

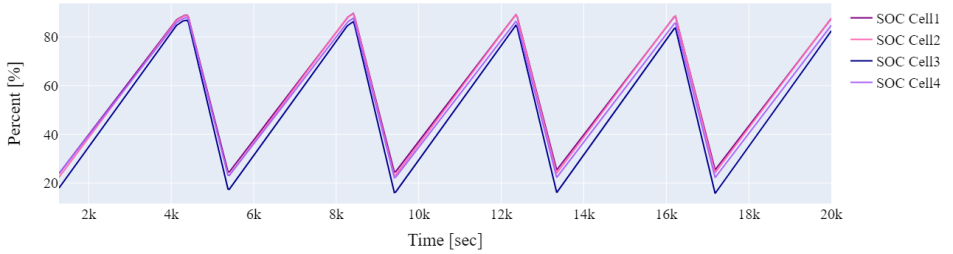


Figure 7.33: SOC of cells

In Figure 7.33, it is evident that the offset introduced in the voltage measurement leads to a lower SOC for cell 3.

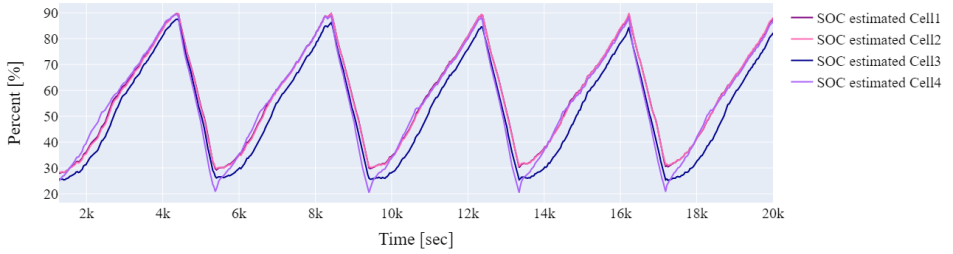


Figure 7.34: SOC of cells estimated by BMS

The BMS estimates the SOC of cell 3 to be higher than it actually is. In Figure 7.34, the smallest value of SOC of cell three starts at 25%, while in Figure 7.33, the SOC of cell three starts at 18%.

7.4 Graphical user interface

This GUI shows the cell balancing test case on the DT of the battery and BMS. This is the only test case implemented in the GUI. Figure 7.35 shows the dashboard when starting. To the left, the user can change the values for the optimization-guided falsification.

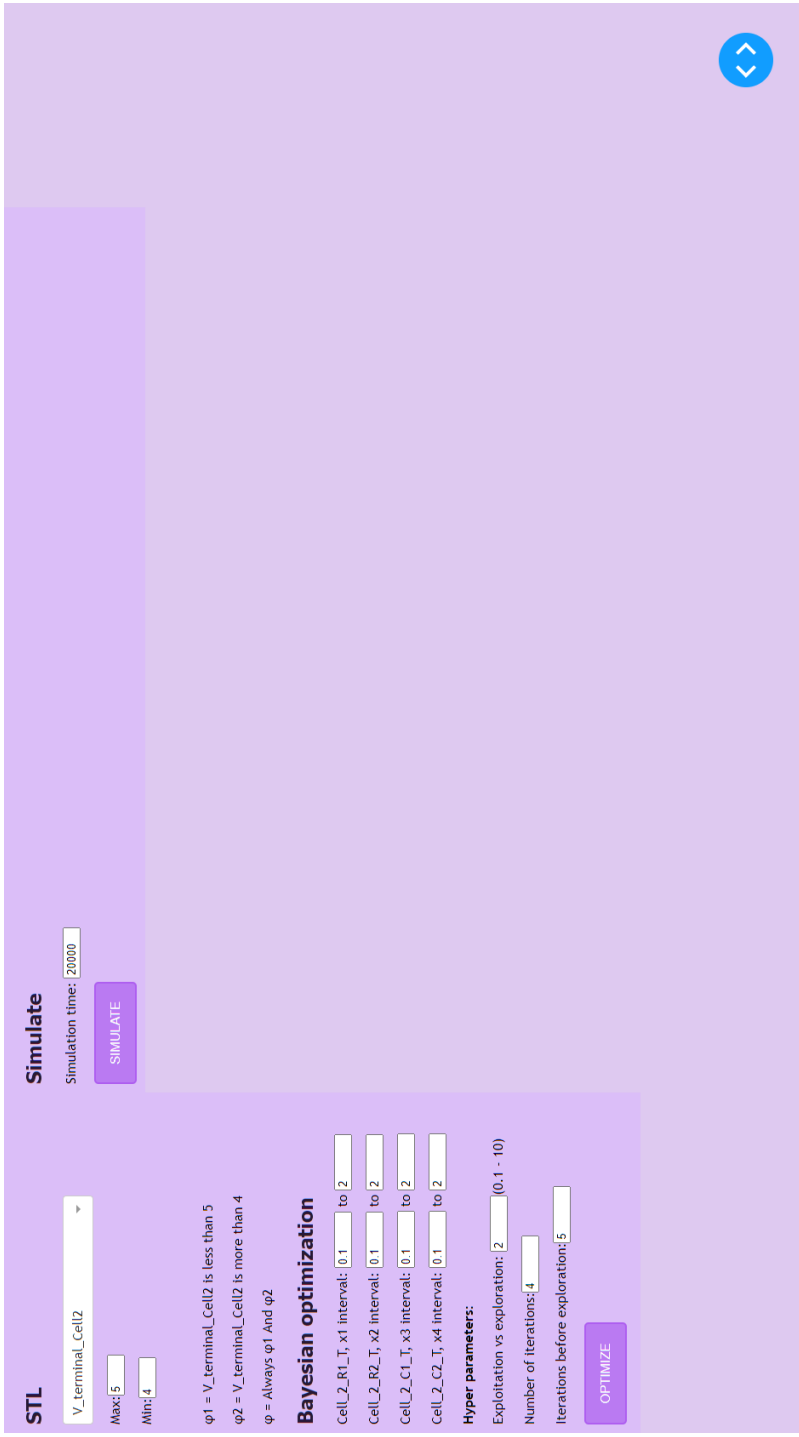
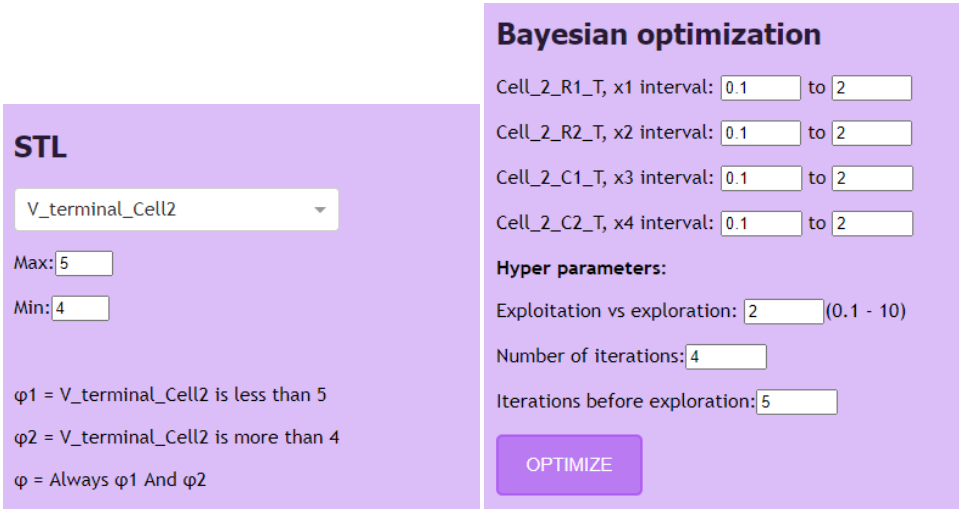


Figure 7.35: Screenshot of GUI startpage



(a) STL **(b)** Bayesian optimization

Figure 7.36: Closer screenshot of STL and Bayesian optimization parts of the GUI.

The STL part of the GUI, shown in Figure 7.36a, lets the user choose which output of the DT to check the robustness for. Right now, the only input that works in the dropdown is V_terminal_cell2, but the dropdown is there to present an example of future extensions. Text boxes are provided to insert max and min values. The complete STL formula is printed below. The text is updated as the input text boxes are updated.

The Bayesian optimization part, shown in Figure 7.36b, lets the user define the inputs for the Bayesian optimization. First, one can choose which intervals to let the x1, x2, x3, and x4 have in the optimization. Maximize function parameters need to be set. Kappa for exploration or exploitation. The number of iterations and the number of iterations before the exploration starts. Here the Maximize function parameters are written out for more user-friendliness instead of using Kappa, n_iter, and init_point. After preferred values are chosen, the button ”optimize” can be pushed. The optimization is then initiated, and after a short while, the output of the optimization is written out.

When the code is done optimizing, the result from the optimization is shown in Figure 7.37. The last found optimum is shown in pink. Here the target is shown with a minus sign before to indicate that the robustness is negative even though the optimization is a maximization problem.

iter	target	x1	x2	x3	x4
0	-0.463	1	1	1	1
1	-0.464	0.892	1.469	0.1	0.674
2	-0.45	0.379	0.275	0.454	0.757
3	-0.459	0.854	1.124	0.896	1.402
4	-0.451	0.488	1.768	0.152	1.374
5	-0.466	0.893	1.162	0.367	0.476
6	-0.478	1.439	1.425	0.579	0.477
7	-0.513	2	1.942	0.869	0.1
8	-0.508	2	2	1.932	0.1
9	-0.478	2	2	2	1.698

Figure 7.37: Screenshot of optimization output after pushing the optimize button. The pink row is the optimum found.

When the optimization is done, the button simulate can then be pressed, and plots will show up. A textbox is implemented for the user to decide the simulation time. This textbox is stated as an input in the code, so whenever it is altered, the plots will change without having to push simulate again. Figure 7.38 shows the GUI after pushing the simulate button.

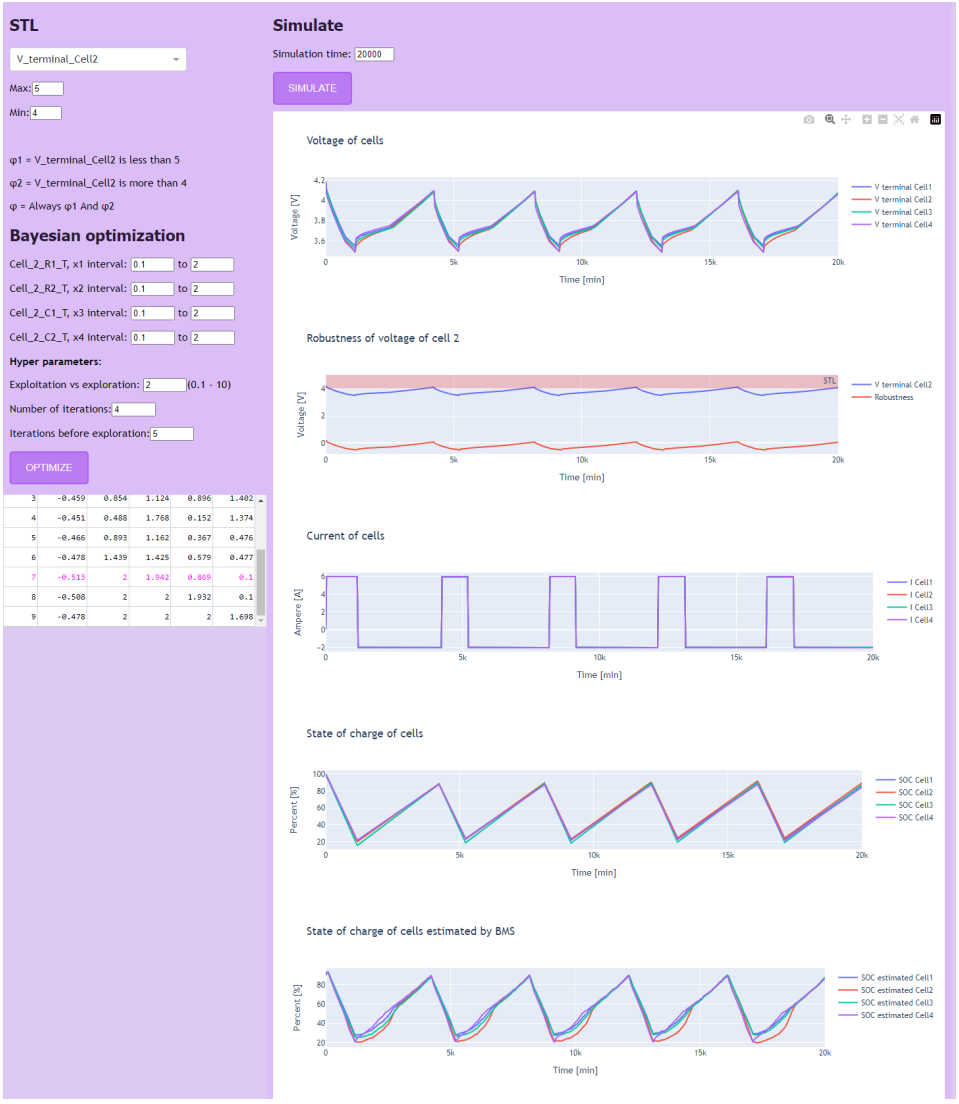


Figure 7.38: Screenshot of whole GUI after pushing the simulate button. Plots can be zoomed or panned.

8

Discussion

8.1 Optimization-guided falsification for safety demonstration

Traditionally, test case selection for safety demonstration has been a manual process based on risk analyses and experience. One advantage of optimization-guided falsification is its efficiency compared to manual testing. Simulation-based approaches offer faster and more flexible testing capabilities. This allows for exploring numerous test cases without subjecting individuals to potential risks or hazardous situations. Optimization-guided falsification also diminishes the number of simulations needed to find failures. By guiding the simulation and attempting to optimize for finding physical parameters that can cause issues with the software, the number of required iterations for identifying possible failure is reduced.

The main goal of safety demonstration is to demonstrate compliance with specific safety requirements. The falsification process is useful in proving that the system fails to meet such requirements. If a model has been successfully verified, it cannot be falsified, and vice versa. If the optimizer can still identify falsifying points, it indicates that the model cannot be verified. However, if the parameters or the combination of parameters leading to an issue can be considered highly unlikely, it could be arguable to pass the system despite the identified parameters. Especially if the parameters are trackable, i.e., the degradation can be foreseen and integrated into the maintenance schedule. For example, if the methodology shows, on a qualified and assured digital twin, that the SOC of the estimator will cause

issues if the battery degradation reaches a certain point, then this can be overcome by suggesting a rejuvenation strategy for the model parameters of the extended Kalman filter, attempting to update the parameters.

Falsification works especially well for the battery and BMS, where the goal is to inject failures rather than correct behavior. However, the quality of the testing depends on the forming of STL requirements and the ranges put on the parameters to be optimized. It is interesting to compare the manual and optimization-guided testing of cell balancing. The optimization is guided by poorer robustness. Therefore the values are more drawn downwards. In comparison, manual testing is only decided by random input values. It should be noted that forming the STL for the particular test cases could be done another way to enhance the safety argument.

Optimization-guided falsification offers flexibility with multiple parameters to guide the optimization process. It offers the freedom to select from various acquisition functions that can influence the optimization procedure. Although this thesis did not extensively explore these different acquisition functions, utilizing UCB proved sufficient to identify poor robustness areas.

Falsification can be valuable for debugging the model as it provides important information to the user. Testing the digital twin uncovered faults, a critical aspect of digital twin development. Even small faults can easily arise in complex models, making simulation a valuable tool for verifying the behavior of the digital twin.

However, while simulation-based testing offers a solid verification platform, combining it with valid processes for test case selection, evaluation of results, and test coverage assessment is crucial. This thesis acknowledges the limited knowledge about the correct behavior of the DT and highlights the need for further research in this area. This does not affect this method as, in the future, the DT will be validated and assured, and the inputs will have a clear and defined range of operability. Provided by the DT developer.

8.2 Simulation-guided Lyapunov opportunities

In theory, the simulation-guided Lyapunov analysis can be implemented on the DT of the battery and BMS. The approach would involve simulating the system multiple times using different inputs known to be safe. These simulations would generate data points representing safe system behavior. To utilize this data for Lyapunov analysis, the collected instances of safe behavior could be compiled into a linear program. And further, use a falsifier as described in section 4.6

It is worth noting that this method may incur significant computational costs due to the large number of inputs of the DT of the battery and BMS. However, this challenge can be addressed by selectively modifying certain inputs, similar to the approach used in STL. Different inputs affect different failures of the battery.

The property to verify in this context would be a safe set or a barrier certificate, ensuring the battery's safe performance.

Simulation-guided Lyapunov analysis offers several benefits for a safety demonstration. It provides a verification process and employs a falsifier to search for traces that do not meet the specified properties, making it an exhaustive method. Furthermore, the use of external verification tools can further enhance its reliability.

Generally, simulation-guided Lyapunov analysis offers stronger proof than optimization-based falsification alone, making it an appealing approach to explore in future research endeavors.

8.3 Limitations

This thesis primarily focuses on implementing the testing method rather than the execution of perfect test cases for the battery. The thorough examination of parameters to be tested and their corresponding ranges has not been fully conducted. The test case selection was conducted in collaboration with DT developer Björklund, but determining appropriate parameter ranges requires further attention.

One notable limitation of the test method is the potential error when using excessively large parameter ranges, leading to the optimization algorithm failing. However, such a scenario is not necessarily feasible in reality. To address this, the tester should have a wider understanding of the system. Suitable test cases and parameter ranges should be provided by the DT developer. This applies to both the development of the STL requirements and the selection of parameters in Bayesian optimization.

Creating accurate and useful requirements in any testing and verification approach is a crucial but often overlooked task. It is essential to carefully craft requirements that precisely capture the system's intended behavior, especially for a safety demonstration.

Another limitation arises from the limited prior knowledge of Python before undertaking this master's thesis. This research endeavor was a valuable opportunity to enhance proficiency in Python programming. Consequently, the organization of the code may have room for improvement.

9

Conclusion

The focus of this thesis has been on safety demonstration and simulation-guided testing methods of a DT of a battery and BMS that is to be implemented into a bigger DT of an all-electric subsea valve. The focus has been on implementing the method in Python and using the method on different test cases. This thesis has implemented a way of using optimization-guided falsification for a DT of a battery and BMS for an all-electric safety valve. The DT will play an important part in the safety of the valve, and using good and efficient testing methods is important.

This thesis aimed to investigate methods for automatic select test cases for implementing fault scenarios into a DT for simulation-based testing of the control system as part of the safety demonstration process. This master has implemented a falsifying method that finds bad inputs to inject faults to the DT for the BMS, i.e., the control system to handle. It is paramount in safety demonstration to have correct and valid test cases. In this thesis, the input parameters may not be valid, so safety demonstration can be available. So rather, this method serves as part of the safety demonstration by searching and providing falsifying examples that the technology is not safe.

The main contribution of this work is implementing a strategy to automate the selection of parameters, limiting the number of simulations needed to find a fault scenario on a DT, along with the implementation of the DT and a test environment in Python. Furthermore, the creation of a GUI for a universal design further streamlines the use of the method. The GUI has been implemented to facilitate the method's usability for test engineers.

This work has significantly contributed to validating the DT through simulation and eval-

uating the results. The successful implementation and testing of optimization-guided falsification on the DT of the battery and the BMS demonstrate its effectiveness.

9.1 Further work

This thesis has implemented a way of using Bayesian optimization for the falsification of a DT of a battery and BMS. If it is relevant to work further with this project, several points can be mentioned as further work with this report.

As mentioned, this thesis did not look into how to perfectly select parameter ranges. In order to fully utilize this recommended methodology for safety demonstrations using DTs, the developer of the DT should specify the physical limitations modeled in the DT, the boundaries of the parameters, and the assumptions and limitations of the DT. Furthermore, specifying the important parameters could help generalize testing.

STL requirements options are broad, and there could be more exploration of different ways to write the STL arguments. This is easier to do with more information about the DT and parameters. Some of the other formulations can be more relevant for different failures. The STL could be implemented using the VBools mentioned in chapter 4.

The GUI has room for expansion and further improvements. Currently, it only includes one test example. However, it should be enhanced to allow for modifying all inputs and testing of various test cases, including those not covered in this thesis. To enhance the GUI, visual representations of the battery could be implemented. This could involve displaying a figure illustrating the battery pack of cells and their corresponding voltages, SOC levels, and current. It would be helpful to highlight cells close to failure by blinking them in red.

Currently, the GUI only displays plots from the final optimum found. However, if there are multiple simulations, it would be beneficial to provide an option to plot multiple results simultaneously.

Consider implementing the GUI using a more advanced framework like React, which can offer enhanced functionality and a more visually appealing interface. Plotly Dash may not be efficient enough for large web applications.

Furthermore, demonstrating compliance with standards such as IEC 61508 should be considered more in-depth. Future work can aim to identify what parts of the standard can be reached utilizing a DT-based falsification approach. Therefore the GUI can be augmented and adjusted to cover and present the simulation results that will be used as evidence to make a claim about the SIL of the system.

To prevent the optimization process from exploring the same points repeatedly, it is advisable to save the falsified points. If the STL argument is altered for a particular test case, the optimizer can avoid revisiting the previously falsified points.

Lastly, as a recommendation for further work, the exploration and evaluation of the Lyapunov method show great promise. Therefore, it should be considered a valuable avenue for future investigation and development.

Bibliography

[1] *Safety Capability of an All-Electric Production System*, ser. OTC Offshore Technology Conference, vol. Day 3 Wed, May 08, 2019, 05 2019, d031S041R003. [Online]. Available: https://doi.org/10.4043/29472-MS

[2] K. Berg, A. Hafver, O. I. Haugen, K. Kvinnesland, M. van der Meulen, T. Myhrvold, F. B. Pedersen, B. Søgård, M. A. Lundteigen, N. A. Zikrullah, A. Falck, R. Flage, C. B. Nyvik, and H. Kim, *Demonstrating safety of software-dependent systems*. Høvik, Norway: DNV AS, 2022.

[3] C. Mahler and M. Glaser, "Application of functional safety in all-electric control systems," in *Underwater Technology conference*, Bergen, Norway, 2018.

[4] NORSOK, "NORSOK S-001 Technical safety," *Standard, NORSOK*, 2021.

[5] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318316021

[6] M. Grieves and J. Vickers, *Transdisciplinary Perspectives on Complex Systems*, 2017.

[7] T. R. Torben, J. A. Glomsrud, T. A. Pedersen, I. B. Utne, and A. J. Sørensen, "Automatic simulation-based testing of autonomous ships using gaussian processes and temporal logic," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 0, no. 0, p. 1748006X211069277, 0. [Online]. Available: https://doi.org/10.1177/1748006X211069277

[8] F. Nogueira, “Bayesian Optimization: Open source constrained global optimization tool for Python,” 2014–. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[9] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, “Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques,” *IEEE Control Systems*, vol. 36, pp. 45–64, 12 2016.

[10] J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Arechiga, “Simulation-guided lyapunov analysis for hybrid dynamical systems,” in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 133–142. [Online]. Available: https://doi.org/10.1145/2562059.2562139

[11] J. Tarascon and M. Armand, “Issues and challenges facing rechargeable lithium batteries,” *Nature*, vol. 414, pp. 359–67, 12 2001.

[12] M. A. Hannan, M. M. Hoque, A. Hussain, Y. Yusof, and P. J. Ker, “State-of-the-art and energy management system of lithium-ion batteries in electric vehicle applications: Issues and recommendations,” *IEEE Access*, vol. 6, pp. 19 362–19 378, 2018.

[13] L. W. Yao, J. A. Aziz, P. Y. Kong, and N. R. N. Idris, “Modeling of lithium-ion battery using matlab/simulink,” in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, 2013, pp. 1729–1734.

[14] M. Seo, T. Goh, M. Park, and S. W. Kim, “Detection method for soft internal short circuit in lithium-ion battery pack by extracting open circuit voltage of faulted cell,” *Energies*, vol. 11, p. 1669, 06 2018.

[15] SUBPRO, “SUBPRO annual report 2021/2022,” 2022.

[16] M. Ghobakhloo, “Industry 4.0, digitization, and opportunities for sustainability,” *Journal of Cleaner Production*, vol. 252, p. 119869, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652619347390

[17] U. A. Lindgren. Latex template. [Online]. Available: https://www.itk.ntnu.no/ansatte/lundteigen_ma/tips

[18] ISO/IEC, *NEK ISO/IEC GUIDE 5*, 2014.

[19] *Real Application of Electric Controls Technology to Subsea Systems: Success, Learnings and Recommendations*, ser. OTC Offshore Technology Conference,

vol. Day 3 Wed, May 03, 2017, 05 2017, d031S038R004. [Online]. Available: https://doi.org/10.4043/27657-MS

[20] IEC, "IEC 61508, functional safety of electrical/electric/programmable electric safety-related systems," *Standard, IEC*, 2010.

[21] T. Winter, M. Glaser, B. Bertsche, S. Imle, and J. Popp, "Analysis of an all-electric safety subsea actuation system architecture," in *2020 Annual Reliability and Maintainability Symposium (RAMS)*, 2020, pp. 1–7.

[22] O. NORGE, "070 – Offshore Norge application of IEC 61508 and IEC 61511 in the Norwegian petroleum industry," *Guideline, NOG*, 2020.

[23] DNV, *DNV-RP-A204, Qualification and assurance of digital twins. Recommended practice*, 2020.

[24] A. Parrott and L. Warshaw, "Industry 4.0 and the digital twin," *Deloitte*, 2017.

[25] DNV, *DNV-RP-A203, Technology qualification. Recommended practice*, 2021.

[26] J. Wang, L. Ye, R. X. Gao, C. Li, and L. Zhang, "Digital twin for rotating machinery fault diagnosis in smart manufacturing," *International Journal of Production Research*, vol. 57, no. 12, pp. 3920–3934, 2019. [Online]. Available: https://doi.org/10.1080/00207543.2018.1552032

[27] R. Sargent, "Verification and validation of simulation models," vol. 37, 01 2011, pp. 166 – 183.

[28] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Comput. Surv.*, vol. 28, no. 4, p. 626–643, dec 1996. [Online]. Available: https://doi.org/10.1145/242223.242257

[29] ——, "Formal methods: State of the art and future directions," *ACM Comput. Surv.*, vol. 28, no. 4, p. 626–643, dec 1996. [Online]. Available: https://doi.org/10.1145/242223.242257

[30] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler, "Recent progress in continuous and hybrid reachability analysis," in *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, 2006, pp. 1582–1587.

[31] B. Meyer and M. Nordio, "Tools for practical software verification," in *Lecture Notes in Computer Science*, 2012.

[32] K. Sen, "Concolic testing," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 571–572. [Online]. Available: https://doi.org/10.1145/1321631.1321746

[33] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski, "Multiple shooting, cegar-based falsification for hybrid systems," in *International Conference on Embedded Software*, 2014.

[34] Z. Ramezani, "On optimization-based falsification of cyber-physical systems." Chalmers, 2022.

[35] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 1977, pp. 46–57.

[36] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166.

[37] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," in *Workshop on Algorithmic Foundations of Robotics*, 2020.

[38] K. Claessen, N. Smallbone, J. Eddeland, Z. Ramezani, and K. Åkesson, "Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 408–415, 2018, 14th IFAC Workshop on Discrete Event Systems WODES 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318306633

[39] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[40] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *CoRR*, vol. abs/1012.2599, 2010. [Online]. Available: http://arxiv.org/abs/1012.2599

[41] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 254–257.

[42] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds., Springer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 167–170.

[43] Z. Ramezani, K. Claessen, N. Smallbone, M. Fabian, and K. Åkesson, "Testing cyber–physical systems using a line-search falsification method," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2393–2406, 2022.

[44] Z. Ramezani, K. Šehić, L. Nardi, and K. Åkesson, "Falsification of cyber-physical systems using bayesian optimization," 09 2022.

[45] T. Akazaki, "Falsification of conditional safety properties for cyber-physical systems with gaussian process regression," in *Runtime Verification*, 2016.

[46] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-guided approaches for verification of automotive powertrain control systems," in *2015 American Control Conference (ACC)*, 2015, pp. 4086–4095.

[47] H. K. Khalil, *Nonlinear systems; 3rd ed.* Upper Saddle River, NJ: Prentice-Hall, 2002, the book can be consulted by contacting: PH-AID: Wallet, Lionel. [Online]. Available: https://cds.cern.ch/record/1173048

[48] C. Barrett and C. Tinelli, *Satisfiability modulo theories*. Springer, 2018.

[49] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 337–340.

[50] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *Automated Deduction – CADE-24*, M. P. Bonacina, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 208–214.

[51] S. Wolfram, *The MATHEMATICA® book, version 4*. Cambridge university press, 1999.

[52] H. Weydahl, "Li-ion batteries - hazards and mitigation," *IF´S RISK MANAGEMENT JOURNAL*, no. 2, p. 16–17, jan 2017. [Online]. Available: https://www.if-insurance.com/globalassets/industrial/files/risk-consulting-magazine/risk-consulting-2017-2.pdf

[53] S. Petrovic, *Nickel–Cadmium Batteries*. Cham: Springer International Publishing, 2021, pp. 73–88. [Online]. Available: https://doi.org/10.1007/978-3-030-57269-3_4

[54] M.-K. Tran and M. Fowler, "A review of lithium-ion battery fault diagnostic algorithms: Current progress and future challenges," *Algorithms*, vol. 13, no. 3, 2020. [Online]. Available: https://www.mdpi.com/1999-4893/13/3/62

[55] D. LYU, B. Ren, and S. Li, "Failure modes and mechanisms for rechargeable lithium-based batteries: a state-of-the-art review," *Acta Mechanica*, vol. 230, 03 2019.

[56] A. Manthiram, "An outlook on lithium ion battery technology," *ACS Central Science*, vol. 3, no. 10, pp. 1063–1069, 2017, pMID: 29104922. [Online]. Available: https://doi.org/10.1021/acscentsci.7b00288

[57] D. Ouyang, M. Chen, J. Liu, R. Wei, J. Weng, and J. Wang, "Investigation of a commercial lithium-ion battery under overcharge/over-discharge failure conditions," *RSC Adv.*, vol. 8, pp. 33 414–33 424, 2018. [Online]. Available: http://dx.doi.org/10.1039/C8RA05564E

[58] M.-K. Tran and M. Fowler, "Sensor fault detection and isolation for degrading lithium-ion batteries in electric vehicles using parameter estimation with recursive least squares," *Batteries*, vol. 6, no. 1, 2020. [Online]. Available: https://www.mdpi.com/2313-0105/6/1/1

[59] N. E. Galushkin, N. N. Yazvinskaya, and D. N. Galushkin, "Mechanism of thermal runaway in lithium-ion cells," *Journal of The Electrochemical Society*, vol. 165, no. 7, p. A1303, may 2018. [Online]. Available: https://dx.doi.org/10.1149/2.0611807jes

[60] W. Diao, Y. Xing, S. Saxena, and M. Pecht, "Evaluation of present accelerated temperature testing and modeling of batteries," *Applied Sciences*, vol. 8, no. 10, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/10/1786

[61] B. Xu, Y. Shi, D. S. Kirschen, and B. Zhang, "Optimal regulation response of batteries under cycle aging mechanisms," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE Press, 2017, p. 751–756. [Online]. Available: https://doi.org/10.1109/CDC.2017.8263750

[62] J. Xu, R. D. Deshpande, J. Pan, Y.-T. Cheng, and V. S. Battaglia, "Electrode side reactions, capacity loss and mechanical degradation in lithium-ion batteries," *Journal of The Electrochemical Society*, vol. 162, no. 10, p. A2026, jul 2015. [Online]. Available: https://dx.doi.org/10.1149/2.0291510jes

[63] M. Chen and G. Rincon-Mora, “Accurate electrical battery model capable of predicting runtime and i-v performance,” *IEEE Transactions on Energy Conversion*, vol. 21, no. 2, pp. 504–511, 2006.

[64] D. Systemes, “Fmpy,” 2023. [Online]. Available: https://github.com/CATIA-Systems/FMPy

[65] T. L. Foundation. Pytorch. [Online]. Available: https://pytorch.org/docs/stable/tensors.html

Appendix

A Inputs and outputs of the digital twins

Digital twin 1: Battery

Inputs:

Cell	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8
Cell_1_C1_1_bp	Cell_2_C1_1_bp	Cell_3_C1_1_bp	Cell_4_C1_1_bp	Cell_5_C1_1_bp	Cell_6_C1_1_bp	Cell_7_C1_1_bp	Cell_8_C1_1_bp
Cell_1_C1_SOC_bp	Cell_2_C1_SOC_bp	Cell_3_C1_SOC_bp	Cell_4_C1_SOC_bp	Cell_5_C1_SOC_bp	Cell_6_C1_SOC_bp	Cell_7_C1_SOC_bp	Cell_8_C1_SOC_bp
Cell_1_C1_table	Cell_2_C1_table	Cell_3_C1_table	Cell_4_C1_table	Cell_5_C1_table	Cell_6_C1_table	Cell_7_C1_table	Cell_8_C1_table
Cell_1_C2_1_bp	Cell_2_C2_1_bp	Cell_3_C2_1_bp	Cell_4_C2_1_bp	Cell_5_C2_1_bp	Cell_6_C2_1_bp	Cell_7_C2_1_bp	Cell_8_C2_1_bp
Cell_1_C2_SOC_bp	Cell_2_C2_SOC_bp	Cell_3_C2_SOC_bp	Cell_4_C2_SOC_bp	Cell_5_C2_SOC_bp	Cell_6_C2_SOC_bp	Cell_7_C2_SOC_bp	Cell_8_C2_SOC_bp
Cell_1_C2_table	Cell_2_C2_table	Cell_3_C2_table	Cell_4_C2_table	Cell_5_C2_table	Cell_6_C2_table	Cell_7_C2_table	Cell_8_C2_table
Cell_1_Capacity	Cell_2_Capacity	Cell_3_Capacity	Cell_4_Capacity	Cell_5_Capacity	Cell_6_Capacity	Cell_7_Capacity	Cell_8_Capacity
Cell_1_OCV	Cell_2_OCV	Cell_3_OCV	Cell_4_OCV	Cell_5_OCV	Cell_6_OCV	Cell_7_OCV	Cell_8_OCV
Cell_1_R1_1_bp	Cell_2_R1_1_bp	Cell_3_R1_1_bp	Cell_4_R1_1_bp	Cell_5_R1_1_bp	Cell_6_R1_1_bp	Cell_7_R1_1_bp	Cell_8_R1_1_bp
Cell_1_R1_SOC_bp	Cell_2_R1_SOC_bp	Cell_3_R1_SOC_bp	Cell_4_R1_SOC_bp	Cell_5_R1_SOC_bp	Cell_6_R1_SOC_bp	Cell_7_R1_SOC_bp	Cell_8_R1_SOC_bp
Cell_1_R1_table	Cell_2_R1_table	Cell_3_R1_table	Cell_4_R1_table	Cell_5_R1_table	Cell_6_R1_table	Cell_7_R1_table	Cell_8_R1_table
Cell_1_R2_1_bp	Cell_2_R2_1_bp	Cell_3_R2_1_bp	Cell_4_R2_1_bp	Cell_5_R2_1_bp	Cell_6_R2_1_bp	Cell_7_R2_1_bp	Cell_8_R2_1_bp
Cell_1_R2_SOC_bp	Cell_2_R2_SOC_bp	Cell_3_R2_SOC_bp	Cell_4_R2_SOC_bp	Cell_5_R2_SOC_bp	Cell_6_R2_SOC_bp	Cell_7_R2_SOC_bp	Cell_8_R2_SOC_bp
Cell_1_R2_table	Cell_2_R2_table	Cell_3_R2_table	Cell_4_R2_table	Cell_5_R2_table	Cell_6_R2_table	Cell_7_R2_table	Cell_8_R2_table
Cell_1_R_ISCR	Cell_2_R_ISCR	Cell_3_R_ISCR	Cell_4_R_ISCR	Cell_5_R_ISCR	Cell_6_R_ISCR	Cell_7_R_ISCR	Cell_8_R_ISCR
Cell_1_R_NO_ISCR	Cell_2_R_NO_ISCR	Cell_3_R_NO_ISCR	Cell_4_R_NO_ISCR	Cell_5_R_NO_ISCR	Cell_6_R_NO_ISCR	Cell_7_R_NO_ISCR	Cell_8_R_NO_ISCR
Cell_1_Rs_1_bp	Cell_2_Rs_1_bp	Cell_3_Rs_1_bp	Cell_4_Rs_1_bp	Cell_5_Rs_1_bp	Cell_6_Rs_1_bp	Cell_7_Rs_1_bp	Cell_8_Rs_1_bp
Cell_1_Rs_SOC_bp	Cell_2_Rs_SOC_bp	Cell_3_Rs_SOC_bp	Cell_4_Rs_SOC_bp	Cell_5_Rs_SOC_bp	Cell_6_Rs_SOC_bp	Cell_7_Rs_SOC_bp	Cell_8_Rs_SOC_bp
Cell_1_Rs_table	Cell_2_Rs_table	Cell_3_Rs_table	Cell_4_Rs_table	Cell_5_Rs_table	Cell_6_Rs_table	Cell_7_Rs_table	Cell_8_Rs_table
Cell_1_SOC	Cell_2_SOC	Cell_3_SOC	Cell_4_SOC	Cell_5_SOC	Cell_6_SOC	Cell_7_SOC	Cell_8_SOC
Cell_1_SOC0	Cell_2_SOC0	Cell_3_SOC0	Cell_4_SOC0	Cell_5_SOC0	Cell_6_SOC0	Cell_7_SOC0	Cell_8_SOC0
Cell_1_t_ISCR							

Outputs:

Current	State of charge	Voltage	Other
I_charge_discharge_current_Cell1	SOC_Cell1	V_terminal_Cell1	V_t_Cell1
I_charge_discharge_current_Cell2	SOC_Cell2	V_terminal_Cell2	V_pack
I_charge_discharge_current_Cell3	SOC_Cell3	V_terminal_Cell3	
I_charge_discharge_current_Cell4	SOC_Cell4	V_terminal_Cell4	
I_charge_discharge_current_Cell5	SOC_Cell5	V_terminal_Cell5	
I_charge_discharge_current_Cell6	SOC_Cell6	V_terminal_Cell6	
I_charge_discharge_current_Cell7	SOC_Cell7	V_terminal_Cell7	
I_charge_discharge_current_Cell8	SOC_Cell8	V_terminal_Cell8	

Inputs :

Cell_1	Cell 2	Cell 3	Cell 4	Other
Cell_1_C1_T	Cell_2_C1_T	Cell_3_C1_T	Cell_4_C1_T	P0
Cell_1_C1_T_KF	Cell_2_C1_T_KF	Cell_3_C1_T_KF	Cell_4_C1_T_KF	Q
Cell_1_C2_T	Cell_2_C2_T	Cell_3_C2_T	Cell_4_C2_T	R
Cell_1_Capacity	Cell_2_Capacity	Cell_3_Capacity	Cell_4_Capacity	S
Cell_1_R1_T	Cell_2_R1_T	Cell_3_R1_T	Cell_4_R1_T	SOC_vec
Cell_1_R1_T_KF	Cell_2_R1_T_KF	Cell_3_R1_T_KF	Cell_4_R1_T_KF	T_vec
Cell_1_R2_T	Cell_2_R2_T	Cell_3_R2_T	Cell_4_R2_T	V0_mat
Cell_1_R_ISCR	Cell_2_R_ISCR	Cell_3_R_ISCR	Cell_4_R_ISCR	V_cell3_offset
Cell_1_Rs_T	Cell_2_Rs_T	Cell_3_Rs_T	Cell_4_Rs_T	V_cell3_offset_ramp
Cell_1_Rs_T_KF	Cell_2_Rs_T_KF	Cell_3_Rs_T_KF	Cell_4_Rs_T_KF	V_cell3_offset_start_time
Cell_1_SOC0	Cell_2_SOC0	Cell_3_SOC0	Cell_4_SOC0	h
Cell_1_SOC_T	Cell_2_SOC_T	Cell_3_SOC_T	Cell_4_SOC_T	thermal_mass
Cell_1_Temp_T	Cell_2_Temp_T	Cell_3_Temp_T	Cell_4_Temp_T	
Cell_1_Vocv	Cell_2_Vocv	Cell_3_Vocv	Cell_4_Vocv	
Cell_1_bal	Cell_2_bal	Cell_3_bal	Cell_4_bal	
Cell_1_dV0_mat	Cell_2_dV0_mat	Cell_3_dV0_mat	Cell_4_dV0_mat	

Outputs:

State of charge	Voltage	Current	State of charge estimated	Other
SOC_Cell1	V_terminal_Cell1	I_Cell1	SOC_est_Cell1	V_terminal_Cell3_measured
SOC_Cell2	V_terminal_Cell2	I_Cell2	SOC_est_Cell2	V_pack
SOC_Cell3	V_terminal_Cell3	I_Cell3	SOC_est_Cell3	T_Cell1
SOC_Cell4	V_terminal_Cell4	I_Cell4	SOC_est_Cell4	

B Python code

Bayesian optimization of DT2 testcase cell balancing

```
from functions import *
from DT_inputs_output import *

start_values_DT2['V_cell3_offset'] = 0 #set to zero when no sensor drift

sim_time = 20000

result_org = simulate_fmu(filename_DT2, start_values=start_values_DT2,
    output=output_DT2, stop_time=sim_time)
df_org = pd.DataFrame(result_org)

robust_output = 'V_terminal_Cell2'
min = 4
max = 5

s_key, s_value, keys = sort_start_values(start_values_DT2)

df_org, minval = calculate_robustness(df_org,robust_output, min, max)

print(minval)

plot_result_plotly(df_org, 20)
plot_robustness_plotly(df_org, 20, robust_output, min, max)


def black_box_function(x1,  x2,  x3,  x4):
    s_key, s_value, keys = sort_start_values(start_values_DT2)

    d = make_dict(keys, s_value)

    d = multiply_by_constant(d, 'Cell_2_R1_T', x1)
    d = multiply_by_constant(d, 'Cell_2_R2_T', x2)
    d = multiply_by_constant(d, 'Cell_2_C1_T', x3)
    d = multiply_by_constant(d, 'Cell_2_C2_T', x4)

    d = make_start_values(d, s_key)

    result = simulate_fmu(filename_DT2, start_values=d, output=output_DT2,
     stop_time=sim_time)
    df = pd.DataFrame(result)

    df, minval = calculate_robustness(df,robust_output, min, max)
    res = minval
```

```
42
43      return -res
44
45  # Bounded region of parameter space
46  pbounds = { 'x1' : (0.1,600), 'x2' : (0.1, 500),  'x3' : (0.1, 200), 'x4'
        : (0.1, 200)}
47
48  optimizer = BayesianOptimization(
49      f=black_box_function,
50      pbounds=pbounds,
51      verbose=2, # verbose = 1 prints only when a maximum is observed,
        verbose = 0 is silent
52      random_state=1,
53  )
54
55
56  utility = UtilityFunction(kind="ucb", kappa=2, xi=0.0)
57
58  optimizer.probe(
59      params={'x1' : 1,  'x2' :1,'x3' :1,  'x4' : 1},
60      lazy=True,
61  )
62
63  optimizer.maximize(
64      init_points=5,
65      n_iter=2,
66  )
67
68  optimalpoint = optimizer.max
69  print(optimalpoint)
70
71  params = optimalpoint.get('params')
72  x1 = params.get('x1')
73  x2 = params.get('x2')
74  x3 = params.get('x3')
75  x4 = params.get('x4')
76
77  s_key, s_value, keys = sort_start_values(start_values_DT2)
78  d = make_dict(keys, s_value)
79  d = multiply_by_constant(d, 'Cell_2_R1_T', x1)
80  d = multiply_by_constant(d, 'Cell_2_R2_T', x2)
81  d = multiply_by_constant(d, 'Cell_2_C1_T', x3)
82  d = multiply_by_constant(d, 'Cell_2_C2_T', x4)
83  print(s_key)
84  d = make_start_values(d, s_key)
85
```

```
result = simulate_fmu(filename_DT2, start_values=d, output=output_DT2,
    stop_time=sim_time)


df = pd.DataFrame(result)

df, minval = calculate_robustness(df,robust_output, min, max)

plot_result_plotly(df, 20)
plot_robustness_plotly(df, 20, robust_output, min, max)
```

Graphical user interface

```
from functions import *
from DT_inputs_output import *

df_opt = pd.DataFrame(columns= ['iter', 'target','x1','x2','x3','x4'])

app = dash.Dash(__name__)

app.layout = html.Div([
    html.Div([
        html.Div([
            html.H2("STL"),
            dcc.Dropdown(['SOC_Cell1', 'V_terminal_Cell1', 'I_Cell1', '
    SOC_Cell2', 'V_terminal_Cell2', 'I_Cell2', 'SOC_Cell3', '
    V_terminal_Cell3', 'I_Cell3', 'SOC_Cell4', 'V_terminal_Cell4', '
    I_Cell4', 'T_Cell1', 'SOC_est_Cell1','SOC_est_Cell2', 'SOC_est_Cell3',
     'SOC_est_Cell4', 'V_pack', 'V_terminal_Cell3_measured'],
                        'V_terminal_Cell2',
                        clearable = False,
                        id = 'STLdropdown'),
            html.Div(html.P([ html.B()])),
            "Max:",dcc.Input('5', type = 'number', step = 0.01, id = '
    STLmax', style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            "Min:",dcc.Input('4', type = 'number', step = 0.01, id = '
    STLmin', style = {'width': '15%'}),
            html.Div(html.P([ html.Br()])),
            html.Div(id = 'STLoutput')

        ],id = 'STL',  style = {'display': 'inline-block'}),

        html.Div([
            html.H2("Bayesian optimization"),
            "Cell_2_R1_T, x1 interval: ", dcc.Input('0.1', type = 'number'
    ,step = 0.01, id = 'x3_bottom', style = {'width': '15%'}), ' to ', dcc
```

```
    .Input('2', type = 'number', max = 1000,step = 0.01, id = 'x3_top',
    style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            "Cell_2_R2_T, x2 interval: ", dcc.Input('0.1', type = 'number'
    , step = 0.01,  id = 'x4_bottom', style = {'width': '15%'}), ' to ',
    dcc.Input('2', type = 'number', max = 1000, step = 0.01, id = 'x4_top'
    , style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            "Cell_2_C1_T, x3 interval: ", dcc.Input('0.1', type = 'number'
    , step = 0.01, id = 'x1_bottom', style = {'width': '15%'}), ' to ',
    dcc.Input('2', type = 'number', max = 1000,step = 0.01, id = 'x1_top',
     style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            "Cell_2_C2_T, x4 interval: ", dcc.Input('0.1', type = 'number'
    , step = 0.01,id = 'x2_bottom', style = {'width': '15%'}), ' to ', dcc
    .Input('2', type = 'number',max = 1000, step = 0.01, id = 'x2_top',
    style = {'width': '15%'}),

            html.Div(html.P([ html.B()])),
            html.P(["Hyper parameters:"], style = {'font-weight':' bold'})
    ,
            "Exploitation vs exploration: ",dcc.Input('2', type = 'number'
    , step = 0.1, id = 'kappa', style = {'width': '15%'}), "(0.1 - 10)",
            html.Div(html.P([ html.B()])),
            "Number of iterations:", dcc.Input('4', type = 'number', step
    = 1, id = 'n_iter', style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            "Iterations before exploration:", dcc.Input('5', type = '
    number', step = 1, id = 'init_points', style = {'width': '15%'}),
            html.Div(html.P([ html.B()])),
            html.Button('Optimize', type = 'submit', id = 'opt_btn'),
            html.Div(html.P([ html.B()])),
        ], id = 'bayesianopt'),
        html.Div([
            html.Div(id = 'opt_output'),
        ] ,id = 'simulation')
    ], id = 'left-container',  style = {'display': 'inline-block' }),

    html.Div([
        html.H2('Simulate'),
            dcc.Store(id='store-data', data=[], storage_type='memory'), #
    'local' or 'session'
            "Simulation time: ", dcc.Input('20000', type = 'number', id =
    'sim_time', style = {'width': '5%'}, step = 1000),
            html.Div(html.P([ html.B()])),
            html.Button('Simulate', type = 'submit', id = 'sim_btn'),
```

```
            html.Div([],id = 'outputsim'),
        ], id = 'center-container', style = {'display': 'inline-block' }),

], id = 'container')

###################################

@app.callback(
    Output('STLoutput', 'children'),
    Input('STLdropdown', 'value'),
    Input('STLmin', 'value'),
    Input('STLmax', 'value')
)
def update_STL(STLdropdown, STLmin, STLmax):
    return html.Div([" 1  = {} is less than {}".format(STLdropdown, STLmax
    ),
                     html.Div(html.P([ html.B()])),
                     " 2  = {} is more than {}".format(STLdropdown, STLmin
    ),
                     html.Div(html.P([ html.B()])),
                     "   = Always   1  And  2 "])

########################################

@app.callback(
    Output('store-data', 'data'),
    Input('opt_btn', 'n_clicks'),
    State('sim_time','value'),

    State('STLmin','value'),
    State('STLmax','value'),

    State('x1_bottom', 'value'),
    State('x1_top', 'value'),
    State('x2_bottom', 'value'),
    State('x2_top', 'value'),
    State('x3_bottom', 'value'),
    State('x3_top', 'value'),
    State('x4_bottom', 'value'),
    State('x4_top', 'value'),

    State('kappa', 'value'),
    State('init_points', 'value'),
    State('n_iter', 'value'), prevent_initial_callback = True,
)

```

```
def optimize(opt_btn, sim_time, STLmin, STLmax, x1_bottom, x1_top,
    x2_bottom, x2_top, x3_bottom, x3_top, x4_bottom, x4_top, kappa,
    init_points, n_iter):
    if (sim_time == 0) or (x1_bottom==0) or (x1_top == 0) or (x2_bottom)
    == 0 or (x2_top == 0) or (x3_bottom == 0) or (x3_top == 0) or (
    x4_bottom == 0) or (x4_top == 0):
        return dash.no_update
    elif opt_btn:
        start_values_DT2['V_cell3_offset'] = 0 #no sensor drift
        result_org = simulate_fmu('
    BP_BMS_Passive_Cell_Balancing_SOC_Charging.fmu', start_values=
    start_values_DT2, output=output_DT2, stop_time=sim_time)
        df_org = pd.DataFrame(result_org)

        df_org, minval = calculate_robustness(df_org, 'V_terminal_Cell2',
    float(STLmin), float(STLmax))
        def black_box_function(x1,  x2,  x3,  x4):

            s_key, s_value, keys = sort_start_values(start_values_DT2)
            d = make_dict(keys, s_value)

            d = multiply_by_constant(d, 'Cell_2_R1_T', x1)
            d = multiply_by_constant(d, 'Cell_2_R2_T', x2)
            d = multiply_by_constant(d, 'Cell_2_C1_T', x3)
            d = multiply_by_constant(d, 'Cell_2_C2_T', x4)

            d = make_start_values(d, s_key)

            result = simulate_fmu('
    BP_BMS_Passive_Cell_Balancing_SOC_Charging.fmu', start_values=d,
    output=output_DT2, stop_time=sim_time)
            df = pd.DataFrame(result)

            df, minval = calculate_robustness(df,'V_terminal_Cell2', float
    (STLmin), float(STLmax))
            res = minval

            return -res

        pbounds = { 'x1' : (float(x1_bottom), float(x1_top)), 'x2' : (
    float(x2_bottom), float(x2_top)),  'x3' : (float(x3_bottom), float(
    x3_top)),
                    'x4' : (float(x4_bottom), float(x4_top))}

        optimizer = BayesianOptimization(
            f=black_box_function,
```

```
            pbounds=pbounds,
            verbose=2, # verbose = 1 prints only when a maximum is observed, verbose = 0 is silent
            random_state= 1,
        )
        utility = UtilityFunction(kind="ucb", kappa=int(kappa), xi=0.0)

        optimizer.probe(
            params={'x1': 1, 'x2' :1, 'x3' :1, 'x4' :1},
            lazy=True,
        )

        optimizer.maximize(
            init_points=int(init_points),
            n_iter=int(n_iter),
        )

        data = optimizer.res
        return data



##############################################
@app.callback(
    Output('opt_output', 'children'),
    Input('store-data', 'data'), prevent_initial_callback = True,
)
def utskrift(opt_data):
    if (opt_data is None): #<--- correct the condition
        return dash.no_update
    else:
        iter_list = []
        target_list = []
        x1_list = []
        x2_list = []
        x3_list = []
        x4_list = []
        for i in range(0, len(opt_data)):
            iter_list.append(i)
            target_list.append(round(opt_data[i]['target'], 3))
            x1_list.append(round(opt_data[i]['params']['x1'], 3))
            x2_list.append(round(opt_data[i]['params']['x2'], 3))
            x3_list.append(round(opt_data[i]['params']['x3'], 3))
            x4_list.append(round(opt_data[i]['params']['x4'], 3))


```

```
        neg_target_list = [ -x for x in target_list]

        df_opt_result = pd.DataFrame(columns= ['iter', 'target','x1','x2','x3','x4'])
        df_opt_result['iter'] = iter_list
        df_opt_result['target'] = neg_target_list
        df_opt_result['x1'] = x1_list
        df_opt_result['x2'] = x2_list
        df_opt_result['x3'] = x3_list
        df_opt_result['x4'] = x4_list

    return html.Div([
            dash_table.DataTable(df_opt_result.to_dict('records'), [{"name": i, "id": i} for i in df_opt_result.columns], style_table={'height': '200px', 'width': '400px', 'overflowY': 'auto'},  style_header={
        'color': 'black',
        'fontWeight': 'bold'
        }, style_cell_conditional=[
        {'if': {'column_id': 'iter'},
         'width': '20px'},
        {'if': {'column_id': 'target'},
         'width': '10px'},
        {'if': {'column_id': 'x1'},
         'width': '10px'},
         {'if': {'column_id': 'x2'},
         'width': '10px'},
         {'if': {'column_id': 'x3'},
         'width': '10px'},
         {'if': {'column_id': 'x4'},
         'width': '10px'},
        ], style_data_conditional=[{'if': {'filter_query': '{{target}} = {}'.format(df_opt_result['target'].min())},
                                    'color': 'magenta'}]),
        ])

########################################

@app.callback(
    Output('outputsim', 'children'),
    State('store-data', 'data'),
    Input('sim_btn', 'n_clicks'),
    Input('sim_time', 'value'),
    State('STLmin','value'),
    State('STLmax','value'),
)

```

```
def runsimulation(opt_data, sim_btn, sim_time, STLmin, STLmax):
    if (opt_data is None) or (sim_time == 0):
        return dash.no_update
    elif sim_btn:
        filename = 'BP_BMS_Passive_Cell_Balancing_SOC_Charging.fmu'
        opt_max = max(opt_data, key=lambda x:x['target'])
        opt_max_params = opt_max['params']
        x1 = opt_max_params.get('x1')
        x2 = opt_max_params.get('x2')
        x3 = opt_max_params.get('x3')
        x4 = opt_max_params.get('x4')

        s_key, s_value, keys = sort_start_values(start_values_DT2)
        d = make_dict(keys, s_value)
        d = multiply_by_constant(d, 'Cell_2_R1_T', x1)
        d = multiply_by_constant(d, 'Cell_2_R2_T', x2)
        d = multiply_by_constant(d, 'Cell_2_C1_T', x3)
        d = multiply_by_constant(d, 'Cell_2_C2_T', x4)

        d = make_start_values(d, s_key)

        result = simulate_fmu(filename, start_values=d, output=output_DT2,
     stop_time=sim_time)

        df = pd.DataFrame(result)
        df, minval = calculate_robustness(df,'V_terminal_Cell2', float(
    STLmin), float(STLmax))

        fig1 = go.Figure()
        fig1.add_trace(go.Line(x=df["time"], y=df["V_terminal_Cell1"],
    name = "V terminal Cell1"))
        fig1.add_trace(go.Line(x=df["time"], y=df["V_terminal_Cell2"],
    name = "V terminal Cell2"))
        fig1.add_trace(go.Line(x=df["time"], y=df["V_terminal_Cell3"],
    name = "V terminal Cell3"))
        fig1.add_trace(go.Line(x=df["time"], y=df["V_terminal_Cell4"],
    name = "V terminal Cell4"))
        fig1.update_layout(
            font_family = "Trebuchet MS",
            title_font_family = "Trebuchet MS",
            title = 'Voltage of cells',
        )
        fig1.update_xaxes(title_text="Time [min] ")
        fig1.update_yaxes(title_text="Voltage [V]")

        fig2 = go.Figure()
```

```
262         fig2.add_trace(go.Line(x=df["time"], y=df["V_terminal_Cell2"],
    name = "V terminal Cell2"))
263         fig2.add_trace(go.Line(x=df["time"], y=df['Robustness'],  name = "
    Robustness"))
264         fig2.update_layout(
265             font_family = "Trebuchet MS",
266             title_font_family = "Trebuchet MS",
267             title = 'Robustness of voltage of cell 2',
268         )
269         fig2.update_xaxes(title_text="Time [min] ")
270         fig2.update_yaxes(title_text="Voltage [V]")
271         fig2.add_hrect(y0=STLmax, y1=STLmin, line_width=0, fillcolor="red"
    , opacity=0.2, annotation_text="STL")
272 
273 
274         fig3 = go.Figure()
275         fig3.add_trace(go.Line(x=df["time"], y=df["I_Cell1"],  name = "I
    Cell1"))
276         fig3.add_trace(go.Line(x=df["time"], y=df["I_Cell2"] , name = "I
    Cell2"))
277         fig3.add_trace(go.Line(x=df["time"], y=df["I_Cell3"], name = "I
    Cell3"))
278         fig3.add_trace(go.Line(x=df["time"], y=df["I_Cell4"], name = "I
    Cell4"))
279         fig3.update_layout(
280             font_family = "Trebuchet MS",
281             title_font_family = "Trebuchet MS",
282             title = 'Current of cells',
283         )
284         fig3.update_xaxes(title_text="Time [min] ")
285         fig3.update_yaxes(title_text="Ampere [A]")
286 
287         fig4 = go.Figure()
288         fig4.add_trace(go.Line(x=df["time"], y=df["SOC_Cell1"], name = "
    SOC Cell1"))
289         fig4.add_trace(go.Line(x=df["time"], y=df["SOC_Cell2"], name = "
    SOC Cell2"))
290         fig4.add_trace(go.Line(x=df["time"], y=df["SOC_Cell3"], name = "
    SOC Cell3"))
291         fig4.add_trace(go.Line(x=df["time"], y=df["SOC_Cell4"], name = "
    SOC Cell4"))
292         fig4.update_layout(
293             font_family = "Trebuchet MS",
294             title_font_family = "Trebuchet MS",
295             title = 'State of charge of cells',
296         )
```

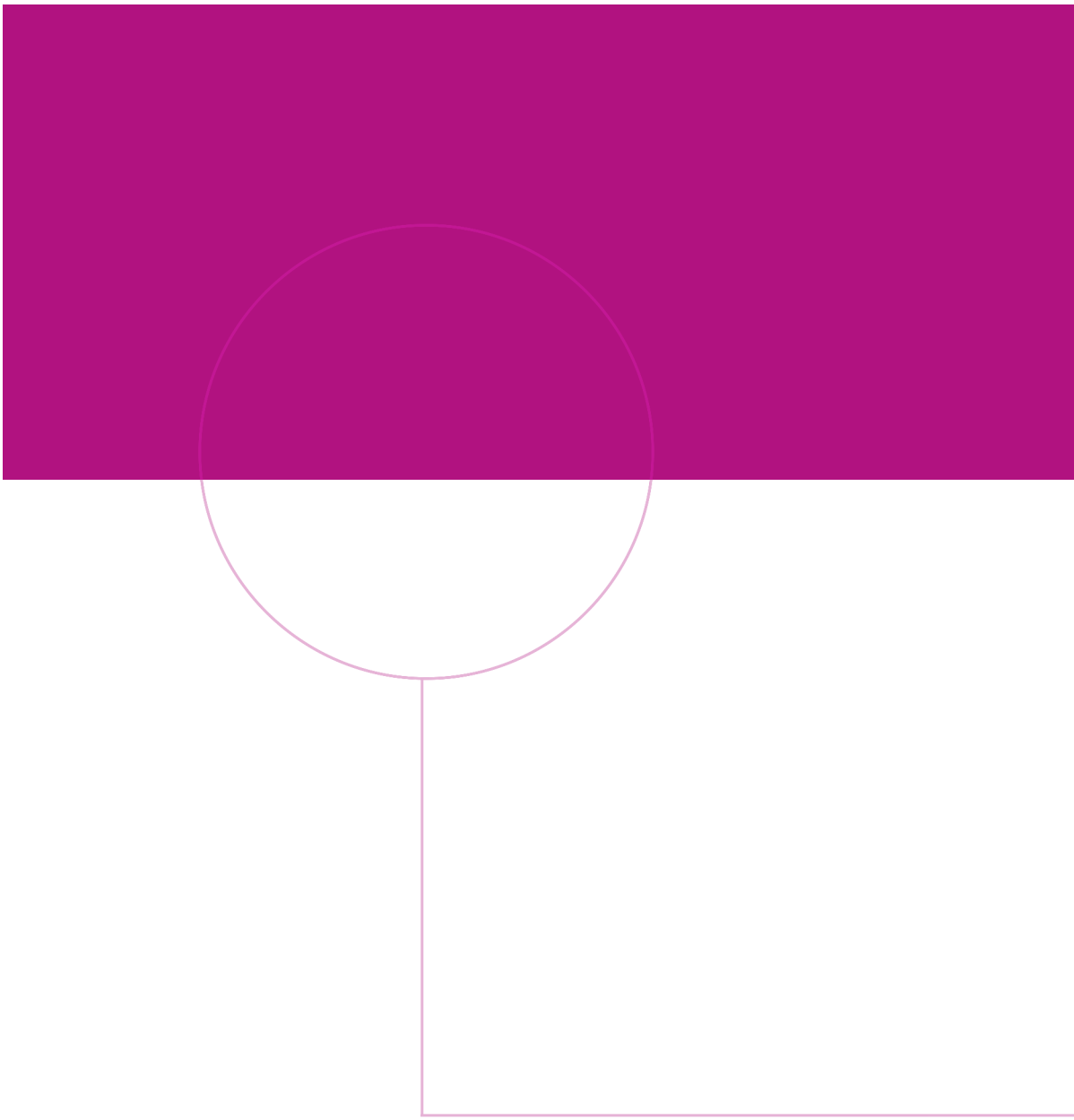
```
        fig4.update_xaxes(title_text="Time [min] ")
        fig4.update_yaxes(title_text="Percent [%]")

        fig5 = go.Figure()
        fig5.add_trace(go.Line(x=df["time"], y=df["SOC_est_Cell1"], name =
    "SOC estimated Cell1"))
        fig5.add_trace(go.Line(x=df["time"], y=df["SOC_est_Cell2"], name =
    "SOC estimated Cell2"))
        fig5.add_trace(go.Line(x=df["time"], y=df["SOC_est_Cell3"], name =
    "SOC estimated Cell3"))
        fig5.add_trace(go.Line(x=df["time"], y=df["SOC_est_Cell4"], name =
    "SOC estimated Cell4"))
        fig5.update_layout(
            font_family = "Trebuchet MS",
            title_font_family = "Trebuchet MS",
            title = 'State of charge of cells estimated by BMS',
        )
        fig5.update_xaxes(title_text="Time [min] ")
        fig5.update_yaxes(title_text="Percent [%]")

        return html.Div([dcc.Graph(id = 'figure1',
                            figure = fig1),
                         dcc.Graph(id = 'figure2',
                            figure = fig2),
                         dcc.Graph(id='figure3',
                            figure = fig3),
                         dcc.Graph(id='figure4',
                            figure =  fig4),
                         dcc.Graph(id='figure5',
                            figure = fig5),])


##################################

if __name__ == "__main__":
    app.run_server(debug=True)
```



NTNU

Norwegian University of
Science and Technology