UNIVERSITY OF VERONA

DEPARTMENT OF COMPUTER SCIENCE

GRADUATE SCHOOL OF NATURAL SCIENCE AND ENGINEERING

DOCTORAL PROGRAM IN COMPUTER SCIENCE

CYCLE XXXVIII

Faulty Behaviors Simulation in Industrial Cyber-Physical Systems for Safety Analysis

S.S.D. ING-INF/05

Coordinator:

Prof. Umberto Castellani

Tutor:

Prof. Franco Fummi

Doctoral Student:

Dott. Francesco Tosoni

Faulty Behaviors Simulation in Industrial Cyber-Physical Systems for Safety Analysis — Francesco Tosoni
PhD Thesis
Verona, March 30, 2026

Abstract

Industrial Cyber-Physical Systems (ICPS) represent a convergence of digital, physical, and networked domains, where ensuring functional safety is of paramount importance. This thesis presents a comprehensive methodology for simulating and analyzing faulty behaviors in ICPS, with a particular focus on multi-domain fault modeling, injection, and detection. By leveraging physical analogies, especially among electrical, mechanical, and thermal domains, the work introduces an innovative approach to extend standardized electrical fault injection techniques (e.g., ISO 26262, IEEE 2427-2025) to non-electrical domains. The proposed methodology enables the derivation of equivalent fault models across domains by exploiting analogies such as impedance and mobility, facilitating the simulation of complex fault scenarios in heterogeneous systems. The approach is validated through multiple case studies, including DC motors, MEMS accelerometers, and lithium-ion battery packs (from the automotive industry), which are modeled using Verilog-AMS and SystemC AMS. These models incorporate electrical, mechanical, and thermal behaviors, allowing for accurate fault injection and behavioral analysis. Furthermore, the thesis explores fault detection strategies based on contract-based monitoring and Time-Sensitive Behavioral Contracts (TSBCs), extending the analysis to software faults and control systems. The integration of Unreal Engine for immersive simulation and visualization, along with the development of a human digital twin framework, demonstrates the applicability of the methodology in Industry 4.0 contexts. The results highlight the effectiveness of the proposed multi-domain fault modeling and simulation framework in enhancing the robustness, safety, and diagnosability of ICPS. This work lays the foundation for future research in fault isolation, predictive maintenance, and the integration of real-time monitoring systems in complex industrial environments.

Abstract (Italian)

I Sistemi Cyber-Fisici Industriali (ICPS) rappresentano una convergenza di domini digitali, fisici e di rete, in cui garantire la sicurezza funzionale riveste un'importanza fondamentale. Questa tesi presenta una metodologia completa per la simulazione e l'analisi dei comportamenti di guasto negli ICPS, con una particolare attenzione alla modellazione, all'iniezione e al rilevamento di guasti multidominio. Sfruttando le analogie fisiche, in particolare tra i domini elettrici, meccanici e termici, il lavoro introduce un approccio innovativo per estendere le tecniche standardizzate di iniezione di guasti elettrici (ad esempio, ISO 26262, IEEE 2427-2025) ai domini non elettrici. La metodologia proposta consente la derivazione di modelli di guasto equivalenti tra diversi domini sfruttando analogie fisiche, facilitando così la simulazione di scenari di guasto complessi in sistemi eterogenei. L'approccio è validato attraverso molteplici casi studio, tra cui motori in corrente continua (DC), accelerometri MEMS e pacchi batteria agli ioni di litio (dall'industria automobilistica), i quali sono modellati utilizzando Verilog-AMS e SystemC AMS. Tali modelli integrano comportamenti elettrici, meccanici e termici, permettendo un'accurata iniezione dei guasti e una precisa analisi comportamentale. Inoltre, la tesi esplora strategie di rilevamento dei guasti basate sul monitoraggio per contratti (contract-based monitoring) e sui Time-Sensitive Behavioral Contracts (TSBCs), estendendo l'analisi ai guasti software e ai sistemi di controllo. L'integrazione di Unreal Engine per la simulazione e visualizzazione immersiva, unitamente allo sviluppo di un framework per lo Human Digital Twin, dimostra l'applicabilità della metodologia nei contesti dell'Industria 4.0. I risultati evidenziano l'efficacia del framework proposto per la modellazione e la simulazione di guasti multidominio nel migliorare la robustezza, la sicurezza e la diagnosticabilità degli ICPS. Questo lavoro pone le basi per future ricerche nell'ambito dell'isolamento dei guasti, della manutenzione predittiva e dell'integrazione di sistemi di monitoraggio in tempo reale in ambienti industriali complessi.

Contents

1

Introduction

The increasing integration of computational and networking capabilities into industrial and engineered systems has led to the proliferation of Industrial Cyber-Physical Systems (ICPSs). These systems, defined by the intersection of digital, physical, and network components, are central to the Industry 4.0 paradigm, enabling highly optimized production, predictive maintenance, and sophisticated monitoring. However, this complexity also introduces significant challenges, particularly concerning functional safety. Ensuring the functional safety of ICPS requires rigorous analysis of potential faults across all domains, electrical, mechanical, and thermal, and their potential interactions. A well-known method for ensuring the functional safety of a system is simulation: during this process, a time series of data is generated that represents the system's behavior over time in response to a given stimulus. The main advantage of simulation is that, potentially, we can recreate situations that would not be possible to recreate in the real world for testing purposes alone. In fact, although crash tests exist, their cost can be very expensive. Producing an object solely to destroy is an operation that, logically, can only be carried out a limited number of times. However, understanding how the system will react to faults, rather than to extreme conditions of use, is paramount for improving the design and maintenance of the system over time. The time series obtained as a result of fault simulations are very useful, if not crucial, for procedures such as fault behavior analysis, and automatic Fault Detection and Isolation (FDI). All of these are fundamental steps for predictive maintenance, which is particularly important in many manufacturing, industrial, aerospace, and automotive environments. In our experience, time series datasets are not easy to find, as data measured in the field is not always recorded and, secondly, is not always made publicly available by the companies. It is even rarer to find data from systems that are broken or nearly failing.

1.1 Challenges and Motivations

While the state-of-the-practice for digital and electrical fault injection is well-established, often guided by standards like ISO 26262 [1], this methodology is frequently limited to injecting a single fault at a time and only on one faulty branch.

A significant gap exists in the systematic modeling and, consequently, the simulability of non-electrical faults. The physical part of an industrial system, comprising multiple domains

(electrical, mechanical, and thermal), necessitates a fault simulation strategy comparable to that of the digital domain. Current challenges include:

- The lack of unified fault modeling and categorization: there is a need to accurately model non-electrical faults across domains, which are often closely tied to the specific system structure.
- The difficulty of fault injection in multidomain simulators: simulation tools often restrict fault injection to system interfaces, thereby inhibiting the comprehensive analysis of internal components or the exploration of cross-domain fault propagation effects.
- The need for seamless co-Simulation: The analysis of complex systems demands an integrated environment where cyber (controller/software) and physical (multidomain hardware) models can be efficiently co-simulated to inject and observe multidomain fault behaviors. For instance, modeling the complex electro-mechanical interactions (e.g., in a DC Motor) and their coupling with thermal aspects (e.g., in Battery packs) requires modeling languages (like Verilog-AMS or SystemC AMS) that support multiple disciplines and allow easy manipulation of internal component parameters to simulate faulty conditions.

Therefore, the primary motivation of this thesis is not only to develop novel methodologies for deriving non-electrical fault models, but also to integrate these models into efficient, multidomain simulation flows, enabling realistic fault injection and analysis for the functional safety assessment of ICPS.

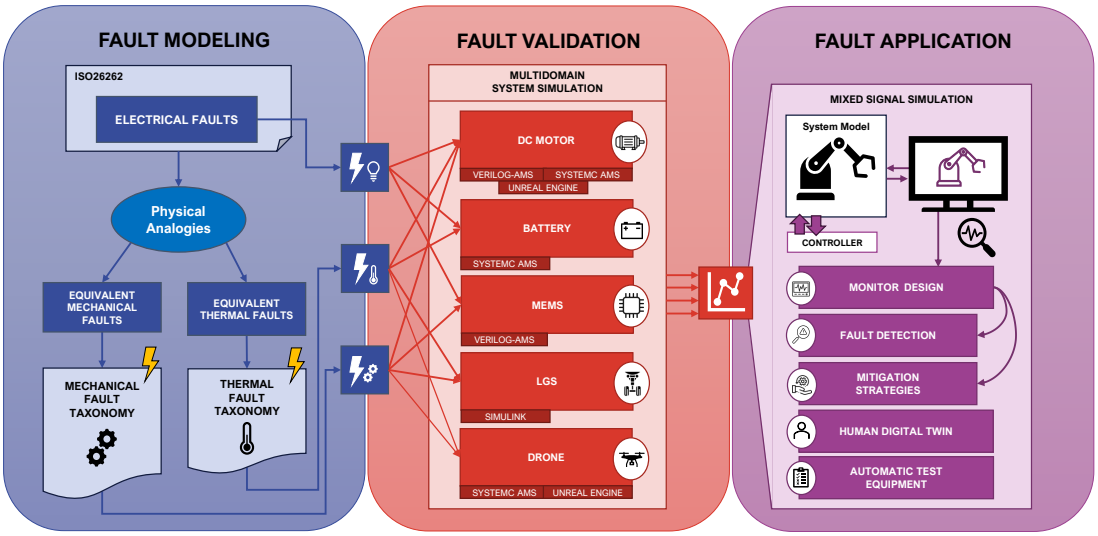


Fig. 1.1: Methodology proposed by this thesis. From left to right: inferring non-electrical fault models through physical analogy; testing such fault models to simulate and generate systems' faulty behavior; adopting the obtained fault models to realize different safety-oriented applications.

1.2 Research Contribution and Thesis Structure

This doctoral thesis addresses the functional safety challenge in ICPS by proposing and validating a novel, holistic, and cross-domain approach to fault modeling and simulation.

To fully appreciate the integration of the different disciplines proposed in this thesis, it is useful to visualize the overall methodology as an end-to-end "left-to-right" workflow, bridging traditional reliability engineering with modern simulation techniques.

The pipeline begins on the left of Fig. 1.1 with standard safety and reliability assessments, namely Hazard Analysis, Failure Mode and Effect Analysis (FMEA), and Fault Tree Analysis (FTA). These traditional activities are crucial for identifying critical failure modes, assessing their risks, and understanding their root causes across the system's different domains. However, traditional FMEA and FTA are predominantly static and document-based. This work pushes these analyses further to the right by translating the identified theoretical failure modes into concrete, simulable behavioral fault models. These multidomain faults (electrical, mechanical, thermal) are then injected directly into a Digital Twin of the ICPS. By simulating these faults in real time or accelerated time, the Digital Twin provides dynamic validation of the FMEA, observing how faults propagate across domains and evaluating the effectiveness of the system's safety mechanisms and monitoring contracts.

The main contributions of this doctoral thesis are shown in Fig. 1.1, and summarized as follows:

1. Fault Modeling: A methodology for applying physical analogies to systematically infer mechanical and thermal fault models from established electrical domain techniques (e.g., ISO 26262, IEEE 2427-2025). This led to the identification and classification of a non-electrical fault taxonomy.
2. Fault Validation and Multidomain Analysis: Validation of the proposed fault models across several Industrial Cyber-Physical Systems case studies, employing Verilog-AMS and SystemC-AMS to ensure easy manipulation and integration of the proposed fault models into industry-compatible simulation tools:
   - A DC Motor and Micro Electro Mechanical Systems (MEMS) accelerometer using multi-domain system simulation.
   - Thermal simulation and safety analysis on Battery Packs, including the development of an automatic tool for design exploration with various coolant layouts.
   - A Drone (multirotor) model, integrating mechanical, electrical, and thermal components in a closed-loop co-simulation environment (SystemC AMS and Unreal Engine).
   - Landing Gear System model of an aircraft, which is also composed of electrical, hydraulic, and mechanical components.
3. Fault Applications: Application of the derived fault models to high-value industrial activities:
   - Fault Detection and Monitoring: Implementation of a contract-based monitoring approach (Time-Sensitive Behavioral Contracts) to detect faults in multi-domain systems, extended to the control module (HW-SW Co-Simulation).

- Human Digital Twin: A framework for human-centered Digital Twins (IMHU) in the Industrial Metaverse for operator state detection (awake/drowsy).
- A battery pack automatic modeling and simulation tool, which helps the user to create complex cell-to-cell connected battery pack models.
- Automated Test Equipment (ATE) Integration: A Service-Oriented Methodology (MATE) for connecting ATE in Industry 4.0 environments.

The remainder of this thesis, which details these contributions, is structured as follows:

- Chapter 2, Fault Modeling: Discusses the theoretical framework and the construction of the non-electrical fault taxonomy based on the principle of physical analogies with the electrical domain.
- Chapter 3, Fault Validation: Presents the multi-domain simulation environment and the detailed analysis of the fault models across various case studies (DC Motor, MEMS, Landing Gear, and Drone).
- Chapter 4, Fault Applications: Explores the advanced applications enabled by the validated fault models, specifically detailing the Contract-Based Monitoring, Human Digital Twin, and ATE integration.
- Chapter 5, Concept of an Automotive Digital Twin, designed to integrate all the components of the previous chapters.
- Chapter 6, Conclusions and Future work directions.

2

Fault modeling through physical analogies

The pursuit of Functional Safety in Industrial Cyber-Physical Systems (ICPSs) necessitates a shift from verifying individual components in isolation to analyzing the system as a holistic entity. While Chapter 1 highlighted the need for a unified simulation approach, this chapter addresses the fundamental theoretical challenge: the semantic gap between physical defects and behavioral simulation.

In the digital and electrical domains, fault modeling is a mature discipline with established standards (e.g., ISO 26262) and abstractions (e.g., stuck-at faults, open/short circuits) that effectively map physical defects to behavioral deviations. Conversely, in the mechanical and thermal domains, failure analysis typically relies on Finite Element Analysis (FEA) or detailed geometric inspections of material degradation (e.g., cracks, wear, fatigue). While accurate, these physical descriptions are computationally too expensive for system-level simulation and often lack a direct translation into the behavioral differential equations used for control logic verification.

This chapter presents a methodology to bridge this gap. As Fig. 2.1 shows, by exploiting the mathematical isomorphism between different physical domains—specifically through Physical Analogies—we propose a framework to infer behavioral fault models for the mechanical and thermal domains starting from established electrical fault primitives. This approach enables the construction of a novel non-electrical fault taxonomy that is natively compatible with behavioral simulation languages, such as Verilog-AMS, allowing for the systematic injection of multi-physics faults.

2.1 Problem Statement

Nowadays, in every industrial field, the design of complex systems is evolving quickly, as a result of the adoption of digital technologies, in the form of the current trend named *Industry 4.0*, *i.e.*, the fourth industrial revolution. The new digital technologies allow indeed to closely monitor and mimic the evolution of a Cyber-Physical System (CPS), thus gaining more knowledge of the operating conditions, allowing to foresee future evolution and to observe complex interdependencies [2]. The modeling complexity, on the other hand, is rewarded by the possibilities it opens, allowing for the mirroring of the actual operation of the analog side of a CPS

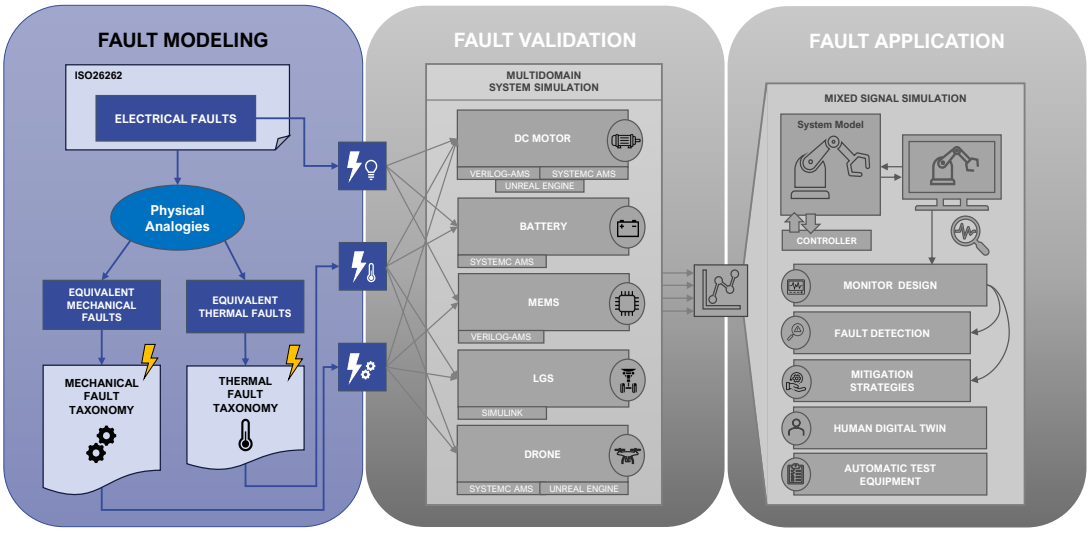


Fig. 2.1: Methodology proposed by this thesis. From left to right: inferring non-electrical faults models through physical analogy; testing such fault models to simulate and generate systems' faulty behavior; adopting the obtained fault models to realize different safety oriented application.

or a smart system. A smart system is designed to integrate heterogeneous components, such as digital, analog, and communication modules, as well as Micro Electro Mechanical Systems (MEMS), in one single, miniaturized device [3,4]. Furthermore, modeling such systems enables performing what-if analysis, estimating the impact of faults on the system or individual components, and better exposing the complex interdependencies between heterogeneous aspects in the generation and propagation of faulty behaviors [5].

This procedure, called *fault injection*, supports the creation of reliability mechanisms in the design phases of a smart system or the analog side of a CPS, which underlines weaknesses that could affect the system's safety [6]. However, applying fault injection to the context of CPSs (that are natively multi-domain systems) differs depending on the part of the system and the domain under analysis [7]. Overall, *fault injection* is the state of the practice for ensuring functional safety of both digital and analog circuits, as quoted by the ISO 26262 standard for the functional safety of road vehicles [1]. In the digital domain, the state-of-the-art fault models are stuck-at-0/1 faults [8]. For the analog domain, the analog faults have been defined and classified by the recent IEEE 2427-2025 standard. At the state-of-the-art, such techniques do not apply to other domains than the electrical one, for instance, in an electromechanical system, where fault models injected into the electrical part will deeply differ from purely mechanical ones, and this applies even more to other physical domains (*e.g.*, thermal, hydraulic). In the mechanical domain, the usage of fault injection techniques is not as widely adopted as in the electrical one. Thus, categorizing mechanical components and machinery working conditions is complex because of a paradigm shift that is too recent to be supported by an established and standardized procedure. Given the impact that mechanical fault injection could have on the evaluation of safety mechanisms, extending fault injection techniques to the mechanical domain appears to be an attractive idea. A key resource comes from the observation that *analogies* exist

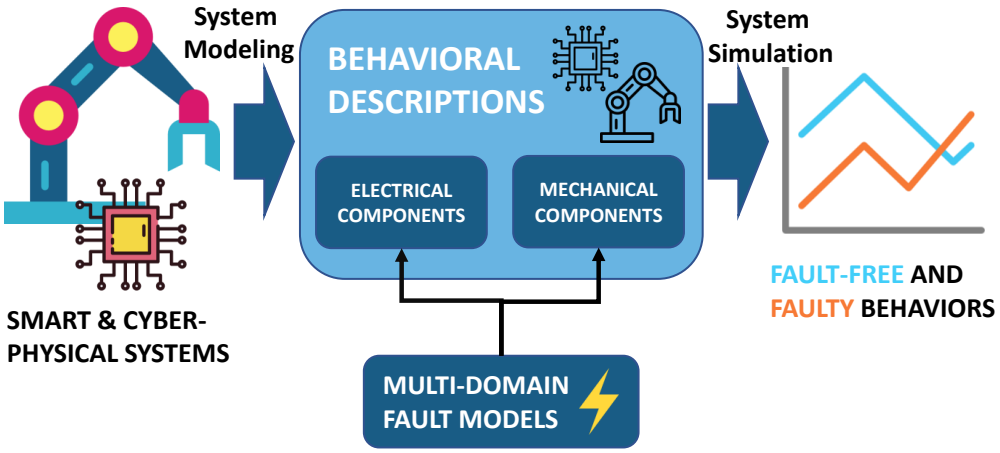


Fig. 2.2: Overview of the proposed methodology to model and inject multi-domain faults into the analog part of smart systems or CPSs described at the behavioral level through differential equations.

between the different physical domains. The idea is that different domain-specific behaviors can be mapped onto the same differential equations by interpreting the quantities involved in terms of their domain-specific meanings. The following sections will show how such *analogies* between mechanical and electrical systems can be exploited to perform safety analysis in the mechanical domain by using a new non-electrical fault taxonomy. Moreover, the purpose of the methodology presented in this paper is to model faulty behaviors of a generic system utilizing such analogies, regardless of the underlying physical domain. In this way, it would be possible to apply advanced fault injection techniques, well established in the electrical domain, to other domains, *e.g.*, the mechanical one. The goal, summarized in Fig. 2.2, is to model a smart system (*e.g.*, a car Electronic Control Unit (ECU)) or the analog side of a CPS (*e.g.*, a wind turbine [9]) with the adoption of different types of faults spanning across different domains, to alter its functionality in many ways, and to deeply explore the impact of faults on the different levels of a system. To summarize, the main innovations proposed in this article include:

1. Create a new mechanical fault taxonomy by identifying mechanical fault models and injecting them directly into the mechanical system;
2. Realize a new automatic tool for the automatic fault injection in multi-domain models (electrical and mechanical), where the standard electrical faults are injected in the electrical part, and the proposed mechanical faults are injected in the mechanical part.
3. Verify the behaviors of the proposed system under different faults and operating conditions; this allows to perform functional qualification of the verification testbench.

2.2 Background and State-of-the-art

This section introduces the state of the art of behavioral fault modeling in smart systems and CPSs through physical analogies. Then, it covers the starting point of this paper, summarizing the previous works in this context.

2.2.1 Behavioral fault modeling

Modeling and simulation of a CPS are key steps for ensuring the functional safety of such systems. At the state-of-the-art, there are two ways to represent and simulate physical systems: graphical tools, *e.g.*, Modelica-based tools and Simulink/Simscape, and adopting description languages, like Verilog-AMS, VHDL-AMS, and SystemC AMS. With the former solution, the system is built by connecting pre-defined blocks belonging to domain-specific libraries. Those pre-defined blocks hide their internal dynamic, making the internal equations visible only to the solver, which computes the system evolution over time. However, the designer can customize the blocks since they are parametrized. So, the fault injection process is limited in such blocks-only environments [10]. Few works focus on fault modeling in the literature. Several working groups deal with fault detection or diagnosis [11, 12], but fault modeling is still a poorly explored topic. The latter solution requires modeling system evolution explicitly as differential and algebraic equations enriched with the application of energy conservation laws. Although the effort during the model realization is higher, the designer can choose the accuracy of the model description. Thus, injecting faults and faulty behaviors is feasible by manipulating the equations.

A fault represents a wrong response in system behavior because of multiple scenarios, i.e., material aging, strain or breaking of an internal component, a production defect, or a digital failure [13]. Functional safety focuses on ensuring the correct performance of systems in the presence of faults through several procedures. *Fault injection* is one of these procedures, which is formally described for the electrical domain in the ISO 26262 standard [1]. Furthermore, a transistor-level circuit can be faulty by injecting specific fault models and using different fault injection techniques, which describe how and where faults are injected [14]. Instead, fault models and injection techniques are not as advanced in the mechanical and thermal domains. Some fault taxonomies exist for mechanical systems [15, 16], but they focus only on the geometry of the system or other physical properties. In the literature, the number of works that analyze the multi-domain faults in the mechanical domain is limited. Therefore, classifying fault models suitable for each mechanical model is complex due to the high heterogeneity of the existing mechanical systems, including MEMS. While in the thermal domain, faults in the form of abnormal temperatures are often the consequence of another disturbance, such as electrical or mechanical variations. Extending fault taxonomies and fault injection techniques to these domains is a way to deepen the study of functional safety in more complex, multi-domain systems. Moreover, all the real-world physical conditions that might vary the behavior of a system with a fault cannot be reproduced in a simulated environment. This work proposes a way to overcome these research gaps.

2.2.2 Modeling physical systems through analogies

In industry and research fields, the electrical equivalent circuits have been adopted increasingly to model complex systems over the years. Such equivalent models helped to bridge the gap between the electrical and mechanical disciplines, allowing engineers to leverage mathematical

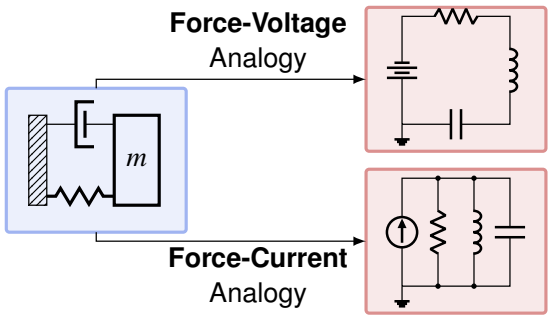


Fig. 2.3: Mapping of a mechanical system to an electrical one by exploiting the physical analogies between mechanical (left) and the electrical equivalent systems (right) according to the force-voltage (top) and the force-current (bottom) analogies.

tools and concepts developed for electrical circuits to the mechanical domain. These complex systems are built based on analogies that link physical quantities of a specific domain to quantities typical of another domain and exploit the concepts of conservation of energy laws [17] applicable to different domains like thermal, magnetic, and hydraulic, especially with the electrical one [18]. As an example, electrical circuits are used to model characteristic parameters variability in energy components [19], and ocean wave power takeoffs [20]. Nonlinear dynamical systems can be represented using this methodology by simulating their nonlinear behavior with active electrical components. As such, electrical circuits can be exploited to model translational/rotational mechanical systems through analogies. Using modern simulators (*e.g.*, Spectre and Eldo), simulating the equivalent circuits obtained is feasible in a fast and accurate way. Alternatively, such electrical equivalent circuits can be simulated via multi-physics simulators (*e.g.*, Simulink, Siemens AMEsim, Ansys Fluent, COMSOL, Modelica-based).

Table 2.1: Mapping between electrical and mechanical quantities in the force-voltage analogy.

Type	**Mechanical translation**	**Mechanical rotation**	**Thermal Domain**	**Analogous electrical**
Effort Variable	Force	Torque	Voltage	Temperature
Flow Variable	Velocity	Angular velocity	Heat Flow	Current
Others	Damping	Rotational resistance	Thermal resistance	Resistance
	Mass	Moment of inertia	-	Inductance
	Compliance	Rotational compliance	Thermal capacitance	Capacitance

2.2.3 Modeling multi-domain systems via Verilog-AMS

Verilog-AMS is the latest extension of the Verilog language created for combining the *digital* and the *analog* part. The communication between the *digital* and the *analog* part is possible through pre-defined language functions, *e.g.*, `timer()`, activating a timer inside an analog design, and `cross()`, describing a crossing routine when the monitored analog function crosses the zero value of magnitude.

The Verilog-AMS language also defines constructs to model systems belonging to different physical domains, in particular, electrical, mechanical, and thermal. Complex models can be defined by combining various physical domains, *e.g.*, by modeling electro-mechanical systems such as a direct current motor. A *discipline* represents a physical domain (*e.g.* electrical or mechanical), and it is composed by *natures*. For instance, the electrical domain is represented by the `electrical` discipline. The natures composing the discipline are the voltage and the current, and they are accessible respectively through the functions `V()` and `I()`. Conservative systems are defined by introducing *potential* and *flow* variables. Moreover, it is possible to define custom disciplines by changing the pre-defined *potential* and *flow* natures. The behavior of the conservative systems is modeled by the evolution of the natures of each domain over time, which needs to be specified. Building an electrical circuit is feasible by using the `branch` statement, which creates a connection between two nodes. Simulating behavioral Verilog-AMS models can be achieved in an efficient trend by using commercial SPICE-based simulators. The simulation can be handled by custom testbenches modules developed in SPICE-based code, *e.g.*, Eldo and Spectre languages.

2.3 Exploiting analogies for modeling mechanical systems and faults

The physical analogies correlating the mechanical and the electrical domains are used to build equivalent systems that share the same behavior [21]. In particular, those analogies are:

- *force-voltage analogy:* mathematical equations of mechanical systems are compared with mesh equations of the electrical system; this analogy is considered the easiest to use (top-right of Fig. 2.3);
- *force-current analogy:* mathematical equations of the mechanical system are compared with the nodal equations of the electrical system, thus being more conservative of the system structure (bottom-right of Fig. 2.3).

From a mathematical perspective, neither analogy is superior since they both lead to valid and consistent results. Therefore, the choice of which one is to be adopted remains arbitrary. The proposed methodology builds upon the *force-voltage* analogy because it is more intuitive than the *force-current* one [22].

2.3.1 Force-voltage analogy between the mechanical and the electrical domain

Through the *force-voltage* analogy, a mechanical system can be represented as an electrical one by mapping each mechanical component in a corresponding electrical one, as shown in the Table 2.1:

- force, *effort* variable in the mechanical domain, is represented by voltage in the electrical domain;
- velocity, the mechanical *flow* variable, is related to the current in the electrical domain;
- all the other equivalence relations between physical quantities in the two domains are mathematically derived from the first two equivalencies: *e.g.*, a damper is equivalent to a resistor because both represent energy loss in their domain.

Several mechanical systems and MEMS can be represented through an electrical circuit due to the mathematical analogies between the physical quantities, and they can be handled with electrical methodologies and tools. By translating the mechanical system into an electrical representation, well-established electrical techniques and tools can be directly used to analyze the mechanical system without the need for adaptations. Moreover, when an electromechanical system is transformed into an equivalent circuit, it can be analyzed as a unified electrical entity.

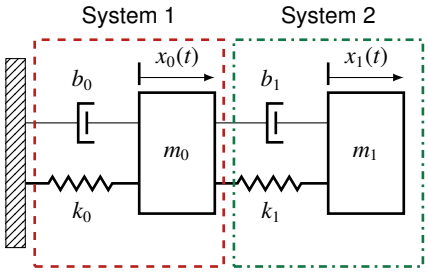


Fig. 2.4: Mechanical representation of a Tuned Mass-Spring-Damper system connected to a fixed reference.

Mechanical system

Let us consider the tuned Mass-Spring-Damper system shown in Figure 2.4 as an example of the mechanical system. This system is composed of a mass (m_0) connected to the ground reference by a spring (k_0) and a damper (b_0), and to a second mass (m_1) through a spring (k_1) and a damper (b_1). Usually, the first mass (m_0) is bigger than the second mass (m_1). This configuration of the system, composing two Mass-Spring-Dampers, allows the damping of the movement amplitude in one oscillator by installing a second oscillator on it. Thus, if tuned properly, the maximum oscillation amplitude of the first system, with respect to a periodic input signal, will be lowered. The Mass-Spring-Damper behavior is described by the differential equations shown in Eqs. (2.1) and (2.2).

$$m_0\ddot{x_0} = F + k_2(x_1 - x_0) + b_1(\dot{x_1} - \dot{x_0}) - k_0x_0 - b_0\dot{x_0} \tag{2.1}$$

$$m_1\ddot{x_1} = -k_1(x_1 - x_0) - b_1(\dot{x_1} - \dot{x_0}) \tag{2.2}$$

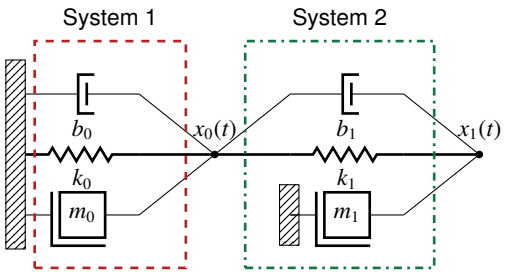


Fig. 2.5: Representation of a Tuned Mass-Spring-Damper system as a mechanical network.

Conversion to a mechanical network

In order to convert the mechanical model (Fig. 2.4) to its electrical equivalent, passing through a representation of the dual mechanical network is useful, as the *force-voltage* analogy does not preserve the topology of the mechanical network during the realization of the electrical circuit [22]. In Fig. 2.5 the two nodes x_0 and x_1 express the movements ($x_0(t)$ and $x_1(t)$) of the two masses in the mechanical system. Each node connects the mechanical components, or branches, which are exposed to the same displacement resulting from a force. The state space model of a mechanical system can be derived easily through its mechanical network description. Moreover, such conversion to the mechanical network is a very useful step to obtain the equivalent electrical circuit, especially using the force-voltage analogy. However, the choice of using mechanical networks is mainly because they can be drawn easily when adopting force-voltage analogy and they make the conversion step in the electrical domain simpler [23].

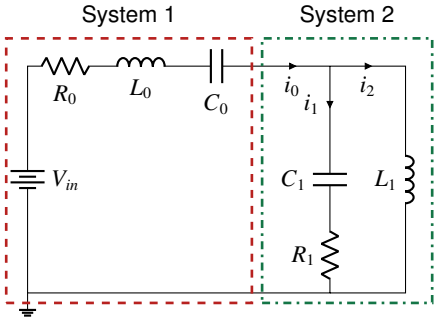


Fig. 2.6: Electrical equivalent representation of the tuned Mass-Spring-Damper system (double-RLC).

Electrical equivalent

The electrical equivalent circuit is a simple electrical system composed of two resistance-inductance-capacitance (RLC) branches shown in Fig. 2.6. Components of the mechanical network connected to the same displacement x_n are connected in series in the electric circuit since they are affected by the same electric current (*force-voltage* analogy consequence). Vice versa,

components connected to different displacements will be connected in parallel since they are not affected by the same electric current. The complete behavior of the double-RLC is defined through the differential equations shown in Eqs. (2.3) and (2.4).

$$0 = V_{in} - R_0 i_0 - L_0 \frac{di_0}{dt} - \frac{1}{C_0} \int i_0 dt + L_1 \frac{di_2}{dt} \tag{2.3}$$

$$0 = L_1 \frac{di_2}{dt} + \frac{1}{C_1} \int i_1 dt + R_1 i_1 \tag{2.4}$$

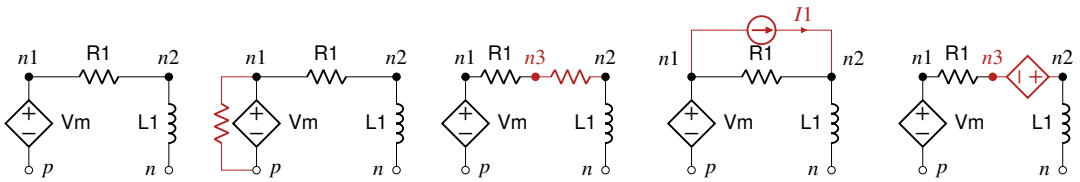
2.3.2 Standardized analog faults

For several years, the set of fault models was limited to open circuit, short circuit, sources, and parameter deviations for the electrical domain. However, what today is known as the IEEE 2427-2025 standard [14] has been produced by its working group. The development group of the 2427 standardized the defect modeling, simulation techniques, and coverage metrics for both analog and mixed-signal circuits. In order to determine the behavior of each analog fault, the IEEE 2427 standard specifies that faults need to be injected one model at a time and in a single point of the circuit. In details:

- *short* electrical fault is equivalent to a bridge fault in an analog circuit, where two points of the circuit are joined by a not intended connection: this fault model has been introduced to ensure a connection between two points of the circuit by injecting it in parallel to a circuit component. Usually, the short fault model is added through a small resistance in parallel to a component (see Fig. 2.7(b)).
- an *open* electrical fault represents a lack of a planned connection between two points in the circuit; thus, the electrons flowing in the circuit are not able to pass throuh the original branch anymore. Therefore, the open fault is injected with a large resistor in series before or after a single component (see Fig. 2.7(c)) to stop the current from flowing.
- a *current pulse* fault is related to extra-currents injected in the circuit, usually modeled as pulses and injected in parallel with a component (see Fig. 2.7(d)).
- a *voltage pulse* fault refers to an abnormal voltage difference added in an electrical circuit; it is usually realized as pulses, and it is injected in series with a component (see Fig. 2.7(e)).
- *parametric* faults are related to the variation of a parameter inside a model, *e.g.*, a smaller resistance value in a circuit due to errors in manufacturing.

2.4 A novel mechanical fault taxonomy

In literature, mechanical faults are categorized based on the component's physical characteristics [16] in relation to a potential cause (*e.g.* an overload) and a failure mode that specifies how the component fails. The fault outcomes mainly involve changes in materials, component geometries, shapes, and dimensions.



(a) Fault-free circuit. (b) Short circuit. (c) Open circuit. (d) Current pulse. (e) Voltage pulse.

Fig. 2.7: Representation of a simple electrical circuit in the fault-free configuration and in four faulty configurations.

Table 2.2: Mechanical fault taxonomy derived from the analysis of electrical faults injected in the equivalent mechanical circuit.

Mechanical Behavior	Mechanical Effect	Mechanical Fault	Electrical Fault Equivalent
Brake/ friction	Added/increased braking force on failed component	Galling/Seizure, Creep, Spalling, Wear/Corrosion	Open
Disconnection	The failed component detaches from the system	Rupture, excess of Backlash	Short
External force	An abnormal (external) force affects the component	It can cause deformations or cracks or ruptures (from impact of fatigue)	Voltage source
Limited movement	The direction of displacement of a component is abnormally modified	Rupture, Deformation, Wear	Current source
Parametric	Intrinsic characteristics of the failed component altered	Wear (component aging) and all the parameters changes	Parametric

The *behavior* of mechanical systems is the main focus of this paper; therefore, a methodology relying entirely on the *behavioral-level* of abstraction has been investigated. Representing mechanical systems as mechanical networks and describing their behavior using equations allows inferring a fault taxonomy closer to this level. The fault modeling and fault injection operations of the electrical domain are also suitable for mechanical networks representing our mechanical system since the injection is performed into single branches in both cases. Moreover, building a mechanical network is structurally very similar to electrical circuits since both descriptions consist of nodes and branches, which have their own equations. For this reason, modeling a mechanical system through an electrical circuit seems to be a good approach for simulating faulty mechanical behaviors.

It is important to note that physical analogies do not provide any information about faults. The behavior is mathematically equal across the domains, but this does not mean that they are functionally equal, and thus, the correspondence of electrical faults with mechanical ones is not obvious. If the behavior of a faulty system makes sense from both an electrical and a mechanical perspective, then the fault models belonging to different domains are equivalent. This correlation can be established by simulating equivalent faulty electrical circuits using electrical fault injection techniques and then by studying the obtained behavior from the mechanical point of view. Some mechanical faults can be assimilated to the faulty behaviors obtained through electrical fault simulation, as defined in [15, 16]. Table 2.2 shows how mechanical fault behaviors

can be derived by simulating faulty electrical circuits equivalent to mechanical systems. Now let us see how from electrical faults, the corresponding mechanical faults can be inferred.

2.4.1 Open circuit fault

The open fault simulates a break in the circuit line, resulting in a significant reduction of the current flow. Conceptually, the introduction of a braking agent in a mechanical system can be correlated to the open fault, as both results in a significant reduction in the flow variable (electric current or velocity, respectively). The fault can be thought of as increasing friction on the system's surface caused by many factors, such as temperature, wear, or debris.

2.4.2 Short circuit fault

The short circuit fault consists of an unintended connection between two points in the circuit that are not meant to be connected. This fault represents a disconnection of mechanical components from the remaining part of the system, such as springs or dampers. This fault can occur if the component is damaged by excessive backlash or a rupture.

2.4.3 Voltage and current sources faults

Voltage and current sources change the voltage and the current at a given point in the circuit in an unexpected trend. These sources of interference can be caused by surrounding electrical circuits or by alpha particles coming from outer space impacting the circuit. Considering the analogy, the voltage source is analogous to an unexpected external force on the faulty component. On the other hand, a current source can be considered as a changing factor of the velocity. Therefore, a current source changes the displacement or rotational velocity of the component affected by the fault.

2.4.4 Parametric faults

Parametric faults are equivalent in both domains since they consist of simple parameter variations.

The taxonomy shown in Table 2.2 is derived essentially from the simulation of several mechanical systems described through the electrical analogy: a mass-spring-damper, a tuned mass-spring-damper, a double pendulum, and a DC motor [24, 25]. Thus, the presented fault taxonomy can be extended by building equivalent circuits of more complex mechanical systems and simulating their behavior. After forming the taxonomy, fault analysis becomes multi-domain: faults are injected and then simulated in their own domain of belonging. For example, the mechanical faults produced are injected into mechanical systems or into the purely mechanical parts of the tested systems. In the following chapters, the procedures of injection and fault simulation become multi-domain and they are illustrated and then exemplified on multiple complex systems: both CPSs, and MEMS sensors.

2.5 Multi-domain fault injection

In this section, we present a methodology to inject the fault models introduced in Section 2.4 through the Verilog-AMS language.

2.5.1 Behavioral fault injection

Fault injection in an analog system treats separately the injection of faults in the different physical domains. In the case of an electromechanical system, the injection process follows different rules for electrical equations and mechanical equations. Thus, let us introduce the procedures for injecting faults in the electrical and mechanical disciplines.

Faults in the electrical discipline

Fig. 2.7 shows the location of the faults injected in a typical electrical circuit: Let us take as reference the short circuit fault between nodes `p` and `n1` shown in Fig. 2.7(b). We can model that fault in Verilog-AMS with the following statement:

```
I(p, n1) <+ V(p, n1) / rshort
```

modeling a small resistor inserted between two nodes `p` and `n`, which is in parallel to the voltage source Vm.

We can model an open-circuit fault in Verilog-AMS, like the one shown in Fig. 2.7(c), with the following statement:

```
V(n3, n2) <+ I(n3, n2) * ropen
```

which is a high resistive contribution to the series of branches that goes from node `n1` to node `n2` of the circuit. Conversely to what we did with the short circuit, here we need to add a new node `n3` between resistor `R1` and node `n2`, to inject our open fault.

Similarly to the short and open fault, voltage and current sources are also injected by following the same guidelines. These sources are unwanted external contributions in the branch in which they are injected, affecting its behavior. Adding a current source to a branch is done by injecting the equation in parallel on a specific branch, as described for the short fault. A voltage source is injected by adding the fault equation in series to the specific branch, as described for the open fault. Furthermore, parametric faults can be injected to alter the values of inductors and resistors.

Faults in the mechanical disciplines

In the mechanical domain, including both translational and rotational mechanics, faults are injected as a direct contribution to the branch that composes a mechanical system. The mechanical disciplines of Verilog-AMS are non-conservative; thus, the system is not modeled by a composition of mechanical nodes. The following statement represents a damper fault modeled in Verilog-AMS:

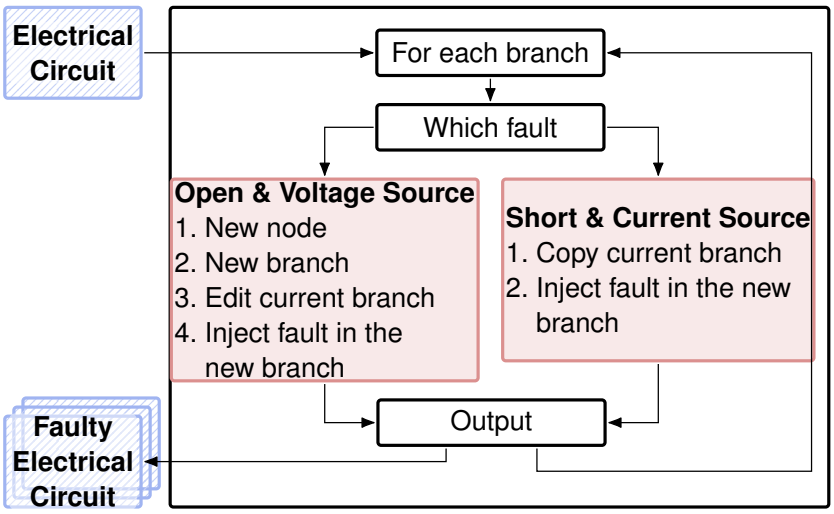


Fig. 2.8: Structure of the fault injection framework for altering electrical descriptions. Red boxes identify the injection process, while blue striped boxes the input and output circuits.

```
Tau(p, n) <+ Omega(p, n) * value
```

where *value* represents the value of resistance to motion. The higher this value, the greater the braking power will be to interrupt the component motion. The damper is injected in the same way into a translational mechanic model by replacing `Tau` with `F` and `Omega` with `Pos` (`kinematic` discipline) or `Vel` (`kinematic_v` discipline). Similarly, an external torque exerted on the rotational component included between the nodes `p` and `n` is modeled as:

```
Tau(p, n) <+ transition(100, 0, 0.1, 0.1)
```

where `transition` is a function that characterizes a transient contribution, *i.e.*, not constant during the time with an associated rise and fall time.

Note: The above examples have been written using Verilog-AMS to demonstrate that injecting the proposed fault models is straightforward. The same operation can be implemented in SystemC AMS, as we will see in the following chapters, where these languages will be explored in greater depth.

2.5.2 Fault injection framework

The faults presented in Section 2.5.1 are injected automatically into Verilog-AMS descriptions by a developed tool. Currently, the framework supports the manipulation of the electrical and mechanical discipline of Verilog-AMS. As mentioned before, only one fault model is injected at a time and at a single point in the system, according to the fault taxonomies shown in Section 2.4. The fault injection framework is built as an additional tool of HIFSuite [26], which allows manipulation of various Hardware Description Language (HDL) descriptions, including Verilog-AMS. Fig. 2.8 explains how the fault injection process takes place for the electrical part: the tool converts any circuit described in Verilog-AMS into XML as an intermediate modeling representation for simpler fault injection. Depending on the fault model being injected, there is

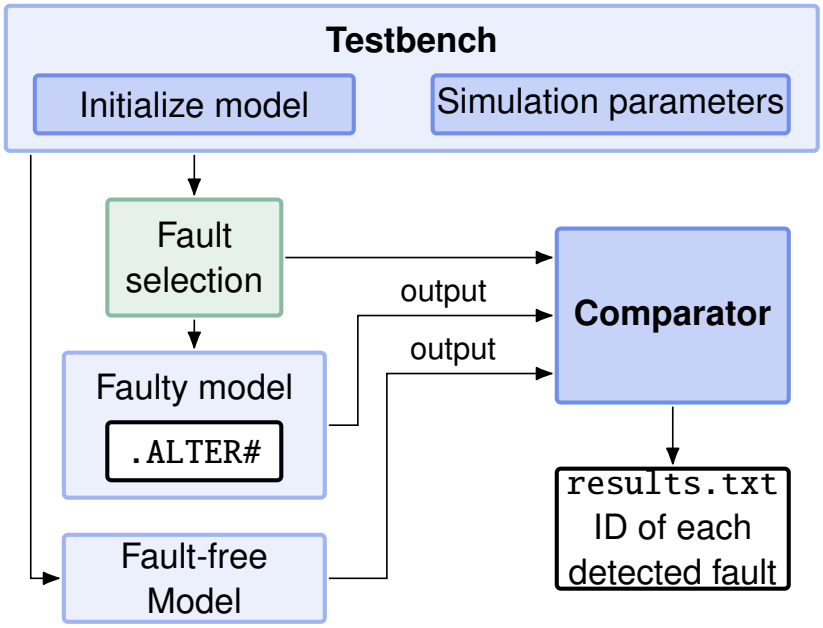


Fig. 2.9: Structure of the simulation flow used to test the different faulty models.

a procedure for faults injected in series to a branch (*i.e.* open and voltage source faults) and one for faults injected in parallel (*i.e.* short and current source faults). Finally, the tool returns all the faulty circuits, based on the electrical fault models, in their original HDL language.

The same procedure is followed for the injection of mechanical faults into Verilog-AMS models. The injection tool is able to generate multiple faulty descriptions of the original mechanical model with the same steps as with electrical circuits. Referring to Table 2.2, the braking agent and the external force faults will be injected once in each mechanical branch. The disconnection and the limited movement are injected between each couple of mechanical nodes in the Verilog-AMS module. The simulation is run by a testbench module, which instantiates both the fault-free model and the faulty models using the `alter` command (see Fig. 2.9). The instantiated modules provide their simulation output to the comparator, which will detect which fault pattern has been injected and at what point in the simulation time. This testbench module simulates every module only once for all comparisons, obtaining all results in a dedicated file.

2.5.3 Functional qualification of the verification testbench

The behavioral fault models presented in the previous sections are defined to be applied in analog descriptions that allow the representation of faulty continuous behaviors. This situation could present many variabilities due to the model itself and due to the stimulus provided to the model by the testbench. Thus the testbench quality is fundamental to adequately stimulate the model in the presence of nominal and faulty conditions [27]. These stimuli can be applied at different points of the model, depending on the physical system under analysis. For example, a direct current motor can be stimulated by applying different source waveforms to the electrical part enabling a different response of the motor dynamic. This implies that different input

waveforms can stimulate different faults injected into the model by changing the diagnostic coverage metrics calculated to asses the functional safety of complex systems. These variabilities due to the input waveforms are described in the literature for the analog domain as *testbench qualification* [28, 29].

The same concepts can be applied to systems described at the behavioral level, as proposed in this article. Consequently, an in-depth analysis should be performed to retrieve the range of values in which the input waveforms need to be positioned to stimulate the system correctly, *e.g.*, by applying a specific waveform frequency range as input. By combining ad-hoc testbenches with systematical fault injection campaigns in multi-domain systems, correct diagnostic coverage metrics can be calculated to guarantee the functional safety of the overall system. In this article, the proposed case study (see Chapter 3) has been analyzed under different faulty conditions (see Section 2.4) and various operating conditions. The latter scenario has been created by applying different stimuli to the system under test. These stimuli change the behavior of the system, and they are obtained through a refinement of the testbench module. Refining means building the testbench that stimulates the widest amount of the system's components, namely activating the largest number of faults among all the injected into the system.

2.6 Realizing thermal models through analogies

Temperature monitoring is critical for any class of industrial device, as it can change the inner parameters of the system belonging to other physical domains, such as friction or electrical resistance. Moreover, temperature tracking is important for determining the operating conditions of the system and for the safety of the surrounding environment. For this reason, realizing the thermal model of a system under test is crucial.

2.6.1 Thermal analogous model

Several approaches exist for building thermal models by exploiting the analogies introduced in the previous section. The main thermal characteristics that determine temperature change are resistance and thermal capacitance. These two parameters can be intuitively considered equivalent to electrical resistance and capacitance. Moreover, according to the analogy, the temperature is equivalent to voltage while heat flow to electric current. Therefore, it is simple to consider an RC electrical network as an equivalent thermal model, and many thermal simulators rely on this assumption [30–35]. These equivalent electrical circuits are known as Foster and Cauer networks [33]. The two networks, depicted in Figs. 2.10 and 2.11, are equivalent in implementation (they can be easily transformed into each other), but they have different characteristics [36]. Creating these networks allows describing the thermal behavior of various classes of systems since the temperature is described as an extra-functional property and modeled in the same way through couples of resistors and capacitors.

In the *Cauer network*, all capacitors are grounded to force heat to flow only in one direction, and each node in the circuit represents a different temperature value. Specifically, voltage on the

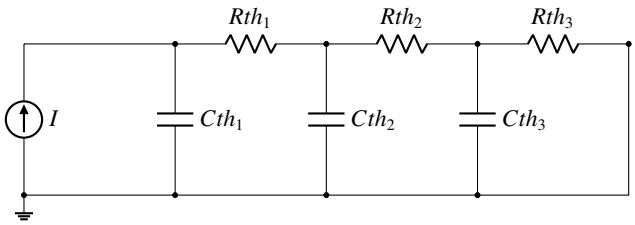


Fig. 2.10: Example of a Cauer network.

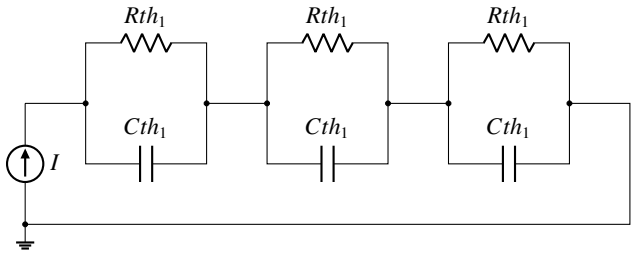


Fig. 2.11: Example of a Foster network.

capacitor nodes represents the temperature at a specific point in the system, while ground represents the ambient temperature. The potential difference between a node and ground represents temperature difference between the object and the environment, and resistance consists of the amount of heat that can flow through an object or a solid surface. For example, in a model related to transistors, each resistance represents the constructive and material differences of each layer of the component. However, the Cauer network is difficult to describe mathematically, so using it to extract parameters such as capacitance and resistance is complex.

In the *Foster network*, only the last capacitor is grounded; all the others are connected in parallel with the resistors. This network has no physical meaning from a thermal point of view, as the nodes do not respect the structure of the system (in the thermal domain, capacitance cannot take negative values). Nonetheless, the Foster network is useful for the extraction of resistance and heat capacity parameters and for making some analytical calculations.

For the reasons listed, the Cauer network representation was chosen in this paper as the equivalent thermal system to study the thermal behavior of the case studies. Furthermore, the accuracy of the thermal networks described in some articles as a Lumped-Parameter Thermal Network (LPTN) is proved to be comparable to a Finite Element Model (FEM) in the prediction of the effective temperature [37].

2.7 Thermal faults classification

Although the analogies allow the creation of valid models for systems belonging to different domains, this is not guaranteed concerning faults added as additional differential equations. A fault belonging to one domain may not have a physically valid and functionally relevant meaning in another domain, even if the purely mathematical behavior is the same. A physical

meaning of the obtained equivalent fault model has to be investigated anyway to validate the mapping between the two fault models belonging to different domains.

An extension of faults from the electrical to the mechanical domain has been proposed in [25]. Now, the focus shifts to a possible mapping between faults in the electrical domain and the thermal domain.

Due to the ability to model thermal systems as electrical circuits described in the previous section, injecting electrical faults into the equivalent thermal circuit and studying their physical effects on the whole system seems very promising. Then, the aim is not only to study the direct thermal effects of a thermal fault but also to understand what consequences it may have in a multi-domain system.

The mapping research between electrical and thermal fault models is first performed on isolated RC Cauer networks of various sizes. Fault injection has been executed on Cauer networks consisting of one to three resistor and capacitor pairs to understand whether the electrical fault actually could have a physical meaning as an equivalent thermal fault. Once the mappings between faults were determined, the analysis switched to a real model, as explained in Section 3.5.2, *i.e.*, where the Cauer network is supplied by the power loss of a real system.

Let us now present the mappings obtained between the electrical fault models and the thermal domain. Consider the analog faults proposed by the IEEE 2427 standard [14]: the analog faults proposed by this standard must be injected at a single point in the circuit, one fault model at a time. In this way, studying the variation of system behavior based on that fault model is feasible.

2.7.1 External heat source

One of the electrical faults proposed in the standard is the *current source* injected in a branch. This fault represents an unintended variation of electric current flow in a branch of the circuit. The cause of this fault may be due to external interference coming from adjacent branches or an external electromagnetic field. Referring back to the analogy, the coupling between electric current and heat flow is crucial when analyzing this fault model. Thus, by injecting electrical current into a branch of a Cauer network, the amount of current flowing in the circuit increases, compared to the current flow generated by power dissipation. Hence, at the thermal level, the system is exposed to a bigger heat flow than normal, causing a rise in the temperature of the component corresponding to the faulty branch. In general, the temperature rise in a mechanical component causes thermal dilatation of the materials composing the object, inducing internal stresses when subject to constraints [38]. Moreover, this anomaly temperature variation is usually a consequence of friction, defined as a passive source of heat that causes an increment in the system temperature, and a mechanical dilatation of materials.

2.7.2 Open Fault

As a resistance value higher than expected in a branch of the circuit. If the additional resistance has a very high value, the fault model simulates a line break in the circuit. The open fault

increases the resistance value on a branch: from a thermal point of view, it represents an increase in the thermal resistance value of the faulty component. This effect can be caused by a sudden change in the physical properties of the material due to non-optimal physical working conditions of the system.

2.7.3 Parametric

As the specific internal parameters change, the behavior of the system changes. In the case of a Cauer network, if the resistance value changes, the current flow decreases at that point of the circuit. Thus, the thermal resistance expressed by the circuit is altered, changing the physical properties of the material of the faulty component. One possible reason for this fault could be a production defect or a deterioration of the surface that no longer insulates heat in the way it was meant to. The same fault can also be applied to the capacitor, thus changing the heat capacity value of the faulty component's material.

The discussed fault patterns also have consequences on the components dissipating power by their usage. Those effects, which are external to the thermal domain, are analyzed in the next section by showing faulty simulation traces. The previously described thermal faults must be considered in a fault injection campaign for a multi-domain system because it reveals new fault scenarios and new fault modes for the system under test. Regarding the electrical domain, a real fault mode is the damage to an electrical circuit, changing its resistive and capacitive properties, due to temperature growth. It is very well known that overheating of electrical components deteriorates their performance [38]. The same scenario is also potentially dangerous mechanically, as anticipated earlier. The deformation of a mechanical component due to high temperature seriously changes its behavior. These new fault modes expand the possible set of failures that a mechanical system could encounter during its normal function due to different physical causes.

2.8 Summary and considerations on the proposed non-electrical fault models

This chapter has established the theoretical framework for a unified approach to fault modeling in ICPSs. By leveraging the mathematical isomorphism inherent in Physical Analogies, we have demonstrated that fault models, traditionally confined to specific physical domains, can be abstracted and translated across disciplinary boundaries. The derivation of the Mechanical and Thermal Fault Taxonomies from established electrical standards (such as IEEE 2427) provides a rigorous, systematic method to represent physical defects, such as friction, wear, rupture, or overheating, using behavioral electrical equivalents.

2.8.1 The advantage of behavioral simulability

The primary contribution of this methodology lies in overcoming the "simulability gap" of non-electrical faults. Traditionally, injecting a mechanical defect (*e.g.*, a broken gear tooth) or a thermal anomaly (*e.g.*, a hotspot) required modifying complex geometric models or finite

element meshes, a process computationally too expensive for system-level functional safety assessment.

Conversely, the proposed framework enables the seamless injection of these faults directly into the behavioral description of the system. By mapping physical defects to electrical primitives (Open, Short, Sources), complex multi-physics failure modes can be simulated by simply altering the differential equations or the netlist topology within standard HDL environments, such as Verilog-AMS or SystemC AMS. This approach enables Cross-Domain Analysis, allowing for the simulation of fault propagation across domains. For instance, a thermal fault modeled as an electrical resistance change can be instantly simulated to observe its impact on mechanical torque and electrical power consumption within the same solver environment.

2.8.2 Limitations of the analogous approach

The methodology proposed in this chapter relies on the Lumped Parameter assumption, mapping physical components to 0D or 1D electrical primitives (resistors, capacitors, inductors). While this abstraction is powerful for enabling multidomain interoperability, it inherently entails a simplification of the physical geometry and spatial dynamics. Specific mechanical failure modes that are strictly dependent on complex geometries or non-linear material deformations, such as asymmetric gear wear, backlash, or localized plastic deformation, are approximated here as variations in behavioral coefficients (*e.g.*, resistance or inductance changes) according to the derived taxonomy Table 2.2. However, this limitation serves as a strategic trade-off that enables System-Level Analysis. While FEA remains superior for designing the structural integrity of a single component, it is computationally unsuitable for simulating long operational cycles of ICPSs. The proposed lumped parameter approach, which approximates geometric detail, achieves the computational efficiency necessary to simulate the interaction between domains. It allows, for instance, observing how a thermal fault propagates to the electrical controller and triggers a software response, a holistic view that component-level physical simulations cannot provide efficiently.

2.8.3 Transition to validation

While the theoretical consistency of these analogies is grounded in energy conservation laws, their practical effectiveness must be proven against complex, real-world dynamics. The taxonomies derived in this chapter serve as the foundation for the experimental activities detailed in the remainder of this thesis.

The following Chapter 3 will assess the robustness and accuracy of these equivalent fault models. The validation will cover a diverse spectrum of applications, demonstrating the method's versatility across key sectors:

- **Industrial**: Through the analysis of an electro-mechanical DC Motor, evaluating how electrical, mechanical, and thermal faults could manifest in such systems.
- **Automotive** and **Smart Systems**: By applying the methodology to MEMS accelerometers and Battery Packs, exploring the critical coupling between thermal degradation and electrical performance.

- **Aerospace**: Through the modeling of a Multirotor Drone, testing the impact of component failures on vehicle dynamics and flight trajectory in a closed-loop simulation, and a Landing Gear System.

These case studies will confirm that the proposed behavioral fault models not only reproduce the expected physical failure modes but also enable the efficient generation of synthetic fault data, which is essential for designing robust diagnostic and safety mechanisms.

3

Multidomain faults validation

The theoretical framework established in Chapter 2 provided a rigorous methodology for inferring mechanical and thermal fault models from electrical primitives via physical analogies. However, the efficacy of any modeling methodology relies fundamentally on its ability to reproduce realistic behaviors in complex, practical scenarios. While the mathematical derivation ensures consistency, only empirical application can verify that the injected faults manifest as plausible physical failures that propagate correctly across different energy domains.

This chapter focuses on the extensive validation of the proposed multi-domain fault models. The primary objective is to demonstrate that the inferred taxonomies are not merely theoretical abstractions but are practically simulable and capable of generating synthetic data representative of real-world failure modes. To ensure the robustness and generalizability of the approach, the validation campaign is conducted across a diverse set of Industrial Cyber-Physical Systems (ICPSs) and Smart Systems, ranging from fundamental electromechanical actuators to complex, safety-critical aerospace systems.

3.1 Diversity of case studies and domains

The selected case studies were chosen to cover a wide spectrum of physical domains, dynamic behaviors, and industrial applications. This heterogeneity is essential to prove that the proposed fault modeling strategy is domain-agnostic and scalable. The validation process of the fault models presented in the Chapter 2 covers:

- **Actuators**: A DC Motor coupled with a gear train, serving as a fundamental building block for investigating electrical, mechanical, and thermal interactions with the surrounding environment.
- **Sensors**: A MEMS Accelerometer, representing the class of miniaturized smart systems where mechanical and electrical properties are tightly coupled at the micro-scale.
- **Energy Storage**: A Battery Pack (modeled on the Tesla Model S architecture), which introduces the critical dimension of thermal-electrical coupling and the simulation of cooling systems.
- **Complex Mechanical Systems**: A Landing Gear System (LGS), enabling the analysis of hydraulics and mechanical faults in a safety-critical avionics context.

- **Autonomous Systems**: A Multirotor Drone, which integrates aerodynamics, mechanics, thermal components and control electronics, offering a scenario for validating faults in a closed-loop, highly dynamic environment.

3.2 Interoperability and simulation environments

A key requirement for any modern industrial methodology is interoperability with existing design flows. Therefore, this chapter also emphasizes the implementation of these case studies across a variety of standard modeling languages and simulation environments.

Rather than relying on a single modeling language, the systems on which the fault models were injected have been coded using:

- **Verilog-AMS** and **SystemC AMS** for rigorous mixed-signal and multi-discipline behavioral modeling.
- **Simulink/Simscape** for high-level system analysis.
- **Unreal Engine** for exploring the integration of physics-based fault simulation within 3D visualization environments.

This multi-language approach demonstrates that the proposed fault taxonomy is not tied to a specific solver but is a generalized methodology applicable wherever physical analogies can be described. The following sections provide a detailed analysis of each case study, presenting the nominal behaviors and the deviations caused by the injection of the multi-domain faults derived in Chapter 2.

3.3 Background and State-of-the-art

This section provides the necessary background regarding the modeling languages that were used to implement the case studies.

3.3.1 SystemC and SystemC AMS

SystemC [39] is a C++ library and modeling platform that enables the simulation and design of complex systems, particularly hardware/software ones. It provides constructs for describing concurrent processes, timed events, and communication between modules, making it suitable for system-level modeling, architectural exploration, and functional verification of digital systems. It supports event-driven simulation and offers a high-level abstraction suitable for hardware design and embedded systems development.

SystemC AMS [40] extends the capabilities of SystemC by introducing support for analog and mixed-signal (AMS) modeling. It allows the simulation of not only discrete-event digital systems, but also continuous-time and non-functional domains, such as mechanical, electrical, or other physical behaviors [2]. This makes SystemC AMS extremely flexible:

- Discrete-time computation (e.g., control logic) can be modeled either with event-driven semantics (using pure SystemC) or with static scheduling (Timed Data-Flow, TDF).

- Continuous-time models (e.g., mechanical and electrical subsystems) can be captured in two main ways: i) Signal processing modules are implemented using a library of predefined primitive modules (e.g., integrators, delays), known as the Linear Signal Flow (LSF) domain. ii) Circuit-level descriptions can be modeled using Electrical Linear Network (ELN) components through the instantiation of conservative primitives (e.g., resistors, capacitors).

The key advantage of SystemC AMS is the seamless coexistence of multiple abstraction levels within a single simulation environment. Timed events from the digital domain and continuous-time behaviors from the analog/physical domains can interact natively, thanks to the integration of the SystemC discrete-event simulator with a continuous-time solver.

This flexibility makes SystemC AMS a compelling choice for simulating diverse aspects of drone systems in a unified simulation run, enabling a more holistic approach to design and verification.

3.3.2 GVSoC ISS for RISC-V cores

GVSoC [41] is an open-source C++ event-driven simulation platform for RISC-V cores of the Parallel Ultra-Low Power (PULP) family [42], supporting the modeling of ultra-low-power CPUs and the definition of complex full-platforms, including multicore, multi-memory levels (i.e., on- and off-chip), complex I/O peripherals, and accelerators. GVSoC supports virtual prototyping and DSE through early-stage performance evaluation based, e.g., on hardware counters and almost cycle-accurate timing models.

GVSoC simulation is event-driven. A circular buffer contains every event generated in the system, enqueued based on the corresponding latency. At any time, the simulator identifies the next event, processes the corresponding actions, and updates the queue. When not used as a stand-alone component, GVSoC exposes APIs to activate its simulation and to collect execution statistics (e.g., timing or power estimations).

3.3.3 Unreal Engine

Unreal Engine is a widely used game engine developed by Epic Games, originally designed for video game creation but now also applied in simulations, architectural visualization, and virtual reality experiences [43]. Its robust physics and rendering capabilities make it an attractive platform for drone simulation [44, 45]. Unreal Engine supports drone simulation through its built-in Chaos Physics system, which models forces such as gravity, drag, and thrust, as well as rigid body and flight dynamics. The Niagara particle system adds realism by simulating environmental effects like wind or turbulence [46]. These features allow for high-fidelity physical interactions in 3D space. Unreal Engine enables the simulation of onboard sensors, such as cameras, GPS, and IMUs, either natively or through external integration. This enables closed-loop scenarios where perception data drives control, useful for testing autonomous behaviors and vision-based navigation. User control is supported via input mapping (e.g., keyboard, joystick).

Unreal Engine supports integration with external tools via socket communication (e.g., TCP/UDP) and APIs, enabling real-time co-simulation. It also provides a visual scripting sys-

tem (Blueprints) for fast prototyping, alongside full C++ support for advanced customization [47, 48]. Thanks to its high-quality graphics, real-time physics, and integration flexibility, Unreal Engine is well-suited for immersive drone simulation, especially when combined with external models that handle energy and control aspects.

3.3.4 Fault modeling using Simulink/Simscape

While Hardware Description Languages (HDLs) like Verilog-AMS and SystemC AMS offer rigorous control over equation-based modeling, a significant portion of industrial engineering—particularly in the automotive and aerospace sectors—relies on graphical, block-diagram-based environments. To validate the proposed fault modeling methodology across diverse industrial workflows, this thesis also adopts the MATLAB/Simulink ecosystem as a representative environment for Model-Based Design (MBD). MATLAB (Matrix Laboratory) serves as the computational engine, providing a high-level programming environment for numerical computation and data analysis [49]. Built upon this foundation, Simulink is a graphical programming environment for modeling, simulating, and analyzing multidomain dynamic systems [50]. The core strength of Simulink lies in its Causal Modeling approach: systems are represented as block diagrams where signals flow from inputs to outputs through transfer functions, integrators, and logic blocks. This paradigm is ideal for designing control systems (the "Cyber" part of Cyber-Physical System (CPS)), but can be limiting when modeling physical plants where bidirectional energy flow is dominant. In the domain of aviation engineering and simulation, the MATLAB environment, particularly through its Simulink and Simscape platforms, has established itself as a cornerstone for design, visualization, and testing [49]. The importance of these tools in the avionic world cannot be overstated, facilitating everything from initial design phases to comprehensive Hardware-in-the-Loop testing. A demonstration of their utility is seen in the development of digital twins of an aircraft in 2019 [51]. This achievement underscores the critical role that MATLAB and its associated tools play in advancing aerospace engineering and simulation practices. Simulink, renowned for its signal flow-based or causal modeling capabilities, facilitates the connection of system components via signals. This modeling paradigm, where signals transmitted between components are clearly defined, allows for straightforward simulation of system behaviors based on fixed input-output relationships.

The recent introduction of fault injection capabilities into the Simulink platform marked a significant advancement in simulation technology, allowing engineers and researchers to systematically introduce and study the effects of faults within their models. This feature enables the simulation of various fault conditions without modifying the original system design, offering a powerful means of conducting safety analyses, such as Failure Mode and Effect Analysis (FMEA), and optimizing fault detection and mitigation strategies. Simscape, extending Simulink's capabilities into physical modeling, adeptly handles the heterogeneity of cyber-physical systems by adopting an equation-based approach to represent complex physical phenomena across multiple domains [52]. This methodology facilitates high-fidelity simulation of physical systems, utilizing both built-in and custom components to accurately replicate real-world conditions. However, despite its prowess in physical system modeling, Simscape orig-

inally provided fault injection features only for its electrical domain, posing challenges for conducting comprehensive fault analyses within Simscape models.

Section 3.7 addresses this limitation by introducing custom fault injection blocks within the Simscape environment. By exploiting Simscape's robust component customization capabilities, we have developed blocks that can be programmatically manipulated to simulate a wide range of fault scenarios. Addressing this gap, our work pioneers the development of fault injection blocks within the Simscape environment. Leveraging Simscape's powerful component definition capabilities, we have created custom blocks that can be programmatically enabled and disabled, allowing for detailed and controlled simulation of fault scenarios. This approach enables the comprehensive analysis of the model under various fault conditions but also sets a precedent for extending fault injection capabilities to Simscape models.

3.4 Fault application to complex systems

The fault taxonomies presented in Section 2.4 are validated on multiple complex cases of study. Every physical domain that characterizes the system is subject to variations due to faults. Consequently, the different faults (adapted for specific physical domains) can be applied to different classes of systems, ranging from smart systems to the analog side of CPSs. If considered an open fault applied on a subcomponent of a smart system, *e.g.*, an RF switch Micro Electro Mechanical Systems (MEMS) [53], the movement of the internal switch will be blocked to one of its configurations, forcing the internal resistance in the on or off-state, limiting the functionalities of the smart system. The first case study is a Direct Current (DC) motor with a gear train shown in Fig. 3.1. The presented results are exemplified on a DC motor due to its extensive application in every industrial field, from miniaturization scale [4] to a bigger scale [54]. Every model presented in this chapter is initially modeled using differential equations, then written in one or more modeling languages, such as Verilog-AMS or SystemC AMS, and then simulated. *e.g.*, the Verilog-AMS code is simulated with a SPICE-based simulator.

3.4.1 Fault Impact Assessment and ICPS Fault Coverage

In traditional digital and analog testing, the impact and detectability of injected faults are evaluated using the concept of fault coverage, defined as the ratio of the number of observed (or detected) faults to the total number of injected faults for a given set of test stimuli. In the context of ICPSs, where continuous multidomain dynamics and discrete control logic coexist, evaluating the impact of a fault requires extending this metric. Additional considerations are required due to the continuous nature of physical dynamics and the presence of closed-loop control systems:

- Fault Masking by Control Loops: unlike open-loop digital circuits, an ICPS often features controllers (e.g., Proportional–Integral–Derivative controller (PID) controllers) designed to minimize errors. A minor parametric fault (*e.g.*, increased friction in a joint) might be completely compensated by the controller increasing the control effort. While the fault is physically present, it might not produce an observable failure at the system level.

- Tolerable Deviations: continuous physical signals cannot be evaluated with strict binary comparisons against a "golden model". Calculating fault coverage requires defining acceptable tolerance bands around nominal behavior.
- Temporal Constraints (Latency): in an ICPS, a fault is considered effectively "detected" or "covered" only if the monitoring system identifies it before it leads to an irreversible hazardous state.

In an ICPS, a fault may not immediately lead to a hard failure (like a digital stuck-at fault), but it might cause a progressive deviation in the system's physical parameters (*e.g.*, increased temperature, slower rotational speed). Therefore, we define the ICPS Fault Coverage as:

$$FC_{ICPS} = \frac{N_{detected}}{N_{injected}} \times 100 \tag{3.1}$$

where:

- $N_{injected}$ is the overall amount of multidomain faults injected into the system model during the simulation campaign.
- $N_{detected}$ is the number of injected faults that produce an observable and critical deviation in the system's behavior.

To objectively determine whether a fault is "detected" (*i.e.*, whether its impact is significant enough to be caught), we rely on the violation of safety requirements. In the framework proposed in this thesis, a fault is considered detected if it triggers a safety monitor or exceeds predefined operational thresholds monitored by the digital twin. This extended metric allows us not only to quantify the robustness of the ICPS against multi-physics disturbances but also to evaluate the effectiveness of the designed testbenches and real-time monitors in identifying hazardous behaviors before they cause catastrophic system failures.

3.5 DC motor

The first case study adopted is a DC motor (Fig. 3.1), a well-known machine used in the industrial and automotive fields as it allows to precisely control the speed of industrial applications. A DC motor, also known as a direct current motor, is an electric machine that converts electrical energy into mechanical energy by generating a magnetic field powered by a direct current. Upon activation, the stator generates a magnetic field that interacts with the magnets on the rotor, causing it to rotate through attraction and repulsion. The commutator linked to brushes connected to the power supply delivers current to the motor's windings in order to maintain continuous rotation. One of the reasons why the DC motor is essential for industrial applications is its ability to precisely control its speed. Now, let us analyze the system: it comprises an electrical actuating part, which powers the system, and a mechanical part, which generates the torque produced by the motor. The electrical part can be modeled through a circuit consisting of a voltage source conducting the current through the system, a resistor, and inductance connected in series. These two parameters represent the intrinsic resistance and the inductance of the armature. The electromotive force produced by the internal coils of the motor is used to show

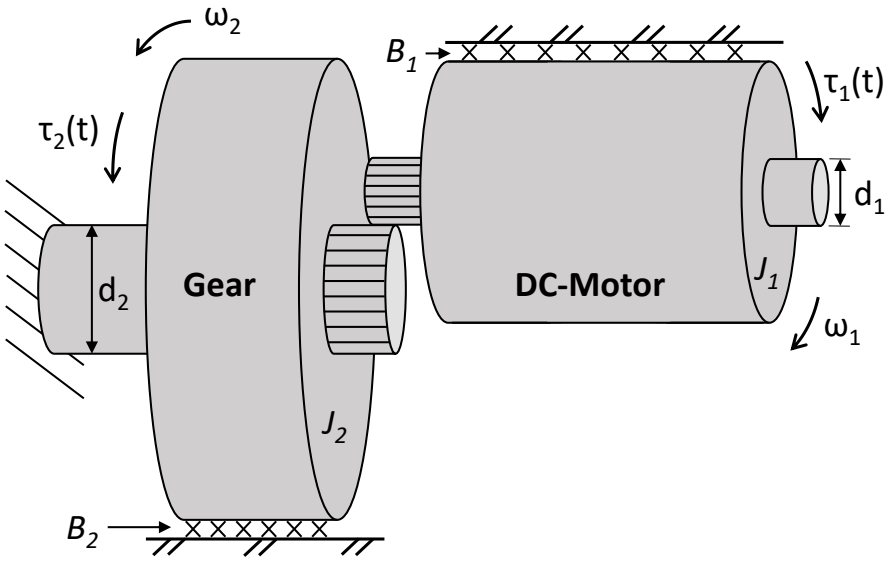


Fig. 3.1: DC motor with a gear train: the DC motor (on the right top of the figure) is connected with the gear train (center of the figure).

the conversion between electrical and mechanical energy, expressed as torque. The DC motor is an electromechanical component; consequently, it is modeled with electrical and mechanical differential equations, while the gear train component receives the torque from the connection with the DC motor and is modeled only with mechanical equations. The following constitutive relations model the dynamics of the motor connected to the gear train:

$$V_1 = K_E \cdot \omega_1 + R_A \cdot I_A + L_A \cdot \frac{d(I_A)}{dt}\,, \tag{3.2}$$

$$\tau_1 = K_T \cdot I_A - J_1 \cdot \frac{d(\omega_1)}{dt} - B_1 \cdot \omega_1\,, \tag{3.3}$$

$$\tau_2 = -N \cdot \tau_1 + J_2 \cdot \frac{d(\omega_2)}{dt} + B_2 \cdot \omega_2\,, \tag{3.4}$$

where V_1 represents the input voltage source for controlling the velocity of the rotor; the variable I_A represents the current flowing in the motor's windings; the angular velocity of the shaft is given by the variable ω, while the torque on the shaft by the variable τ. K_T and K_E are motor coefficients used to specify the size of the motor. Table 3.1 lists the value of each parameter used in this test case. Eq. (3.2) and Eq. (3.3) represent the electrical and mechanical rotational dynamics of the DC motor, while Eq. (3.4) represents the dynamic of the gear train that receives the opposite torque (*i.e.*, it turns in the opposite directions of the motor) from the motor shaft reduced by factor N representing the reduction factor. This mechanical system is used in many applications to increase the torque on the gear train shaft, *e.g.*, as the driver of the joints of an anthropomorphic manipulator [55].

Table 3.1: DC motor with gear train parameters.

Variable	Name	Value	SI Unit
Input voltage	V_1	120	V
Back-EMF coefficient	K_E	0.1785	V·s/rad
Torque coefficient	K_T	25.2756	N·m/A
Armature resistance	R_A	8.4	Ω
Armature inductance	L_A	0.0084	H
Motor inertia	J_1	0.0035	kg·m^2
Motor friction	B_1	0.064	-
Gear inertia	J_2	0.035	kg·m^2
Gear friction	B_2	2.64	-
Motor shaft radius	r_1	0.02	m
Gear shaft radius	r_2	0.16	m
Gear ratio	N	r_2/r_1	-

3.5.1 Multidomain fault injection and simulation

The fault taxonomy developed by applying the methodology outlined in this article is used to conduct a fault analysis of all the case studies presented in this chapter. The simulation environment is set up on a CentOS machine equipped with an Intel Core i7-9700 processor, operating at a frequency of 3.0 GHz, and 16 GB of RAM. The Verilog-AMS code is simulated with the Questa-ADMS tool (which uses Eldo as an analog simulator and Questa as a digital simulator) by Siemens EDA. The faults of the two domains composing the DC motor have been injected through the automatic fault injection tool presented in the previous sections.

The DC motor with gear train has been modeled using two different modules, described here in the Verilog-AMS language, and is shown in Listing 3.1. The first module contains the DC motor equations (Eq. (3.2) and Eq. (3.3)), while the second module includes the gear train equation (Eq. (3.4)). The two modules are interconnected by a `rotational_omega` port that allows the torque exchange between the two components. Regarding fault injection in the electrical part, consisting of three branches, the tool injected 3 open, 12 shorts, 3 voltage sources, and 12 current sources. On the other hand, the mechanical faults of Table 2.2, except the parametric, have been injected once per mechanical branch, then 2 faults. The entire model has been simulated in different operating conditions, and the faults from Table 2.2 have been tested on this system. Moreover, the system has been stimulated with different input curves, *e.g.*, step or sine curve, in order to analyze the different faulty behaviors. Fig. 3.2 shows the simulation results for this first case study. The plots are related to the angular velocity of the DC motor with the gear train simulated for 10 seconds: they report fault-free simulation (dotted blue line) and four different fault models:

- one from the electrical domain: an open fault in the electrical part of the motor (open-m-100, green line);
- three mechanical domain faults: an external torque source on the motor with a value of $120N \cdot m$ (τ-source-m-120, red line), a damper on the motor with a value of 1e03 (damper-m-1e03, magenta line), and a torque source on the gear train with a value of $40N \cdot m$ (τ-source-r-40, yellow line).

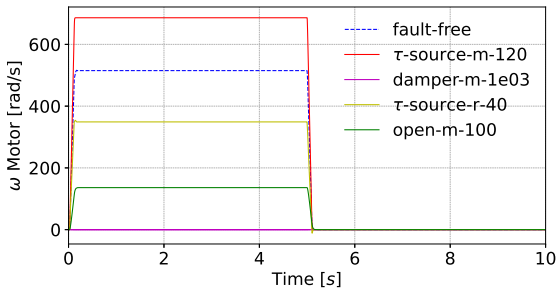
Note that Fig. 3.2(a) and Fig. 3.2(c) are related to the simulation of the system stimulated with a voltage step of 120 V for 5 seconds. Furthermore, Fig. 3.2(b) and Fig. 3.2(d) are related to the simulation of the system stimulated with a sin curve of 120 V with frequency 0.1 Hz for 5 seconds. The plots show the fault-free response of the system reflecting system equations (blue dotted line) and the effect of faults: the external torque accelerates the two components drastically (red line); the damper stops motor rotation (magenta line), and the open fault reduces the voltage feed for the motor, allowing the system to rotate less than the normal response (green line). The presented simulation results have been obtained by feeding the system with stimuli that allow the detection of all the injected fault models. We can notice how, by feeding the system with different stimuli, faulty behaviors act in a different way.

For example, with an insufficient supply voltage source to the motor, the effect of a damper on the motor cannot be visible because the motor would not move anyway. However, with a torque source on the gearbox, it is still possible to detect the faulty behavior of this fault (because everything should be stopped). Instead, a high input voltage source implies that the motor works at high velocity: a torque source on the gearbox cannot be noticed, whereas the effects of a damper on the motor will be detected. This type of analysis highlights how the quality of the testbench module affects the system behavior during the fault campaign simulation.

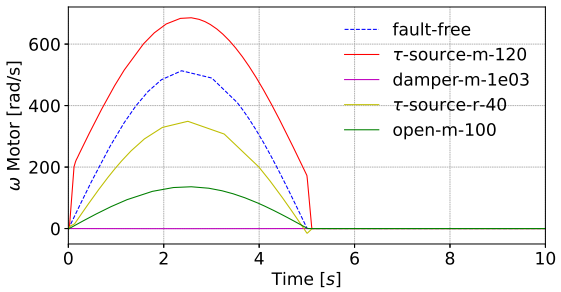
These considerations take place in the digital domain as *functional qualification of the testbench verification* analysis [56]. In the analog field, this approach needs to be considered to verify the correctness of the design by using the multi-domain fault modeling presented in this article. Consequently, an in-depth analysis should be performed to retrieve the range of values in which the input waveforms need to be positioned to stimulate the system correctly, *e.g.*, by applying a specific waveform frequency range as input [28].

3.5.2 Thermal analysis

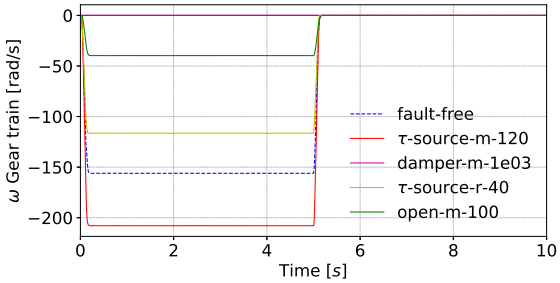
The DC motor system was also extended to test the thermal fault models (Fig. 3.3). As described in Section 2.6.1, the motor was extended with a Cauer electrical network, which models its thermal behavior (Fig. 3.4). Specifically, the network consists of two resistor-capacitor pairs, modeling (1) the difference between the internal temperature of the motor (rotor) and the shaft, and (2) the temperature difference between the shaft and the surrounding environment. These two pairs of resistor-capacitor represent a discretization into several regions of our system, belonging to different functional parts [57]. Power dissipation (P_{loss}) is the key value for calculating the input current in the Cauer network, and it is calculated as the current flowing in the motor armature times the armature resistance, as shown in the Eq. (3.5):



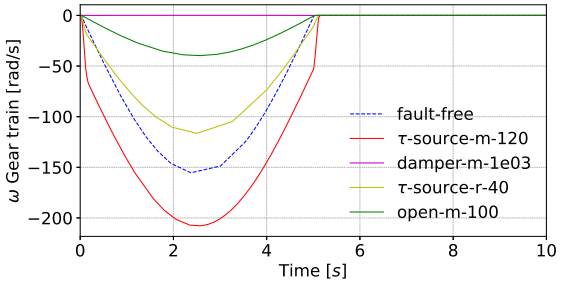
(a) Angular velocity of the motor, with a step as input.



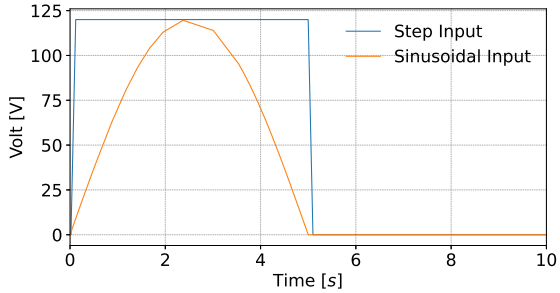
(b) Angular velocity of the motor, with a sinusoidal input.



(c) Angular velocity of the gear train, with a step as input.



(d) Angular velocity of the gear train, with a sinusoidal input.



(e) Input voltage to the DC Motor.

Fig. 3.2: Fault-free and faulty DC Motor waveforms.

$$P_{loss} = i^2 \cdot R_M \tag{3.5}$$

$$R_M = R_{M(0)} \cdot \Big(1 + t_{winding} \cdot (T - T_{(0)})\Big) \tag{3.6}$$

$$K_T = K_{T(0)} \cdot \Big(1 + t_{magnet} \cdot (T - T_{(0)})\Big) \tag{3.7}$$

The heat produced by this power turns into current following the analogy. Eqs. (3.6) and (3.7) show how the armature resistance value and torque constant vary as the motor temperature T changes. In the equations, the variables specified with (0) represent the initial value associated with each variable. The values $t_{winding}$ and t_{magnet} indicate the temperature coefficients of the motor windings, made of copper, and the permanent magnet, considered to be composed of ceramic material.

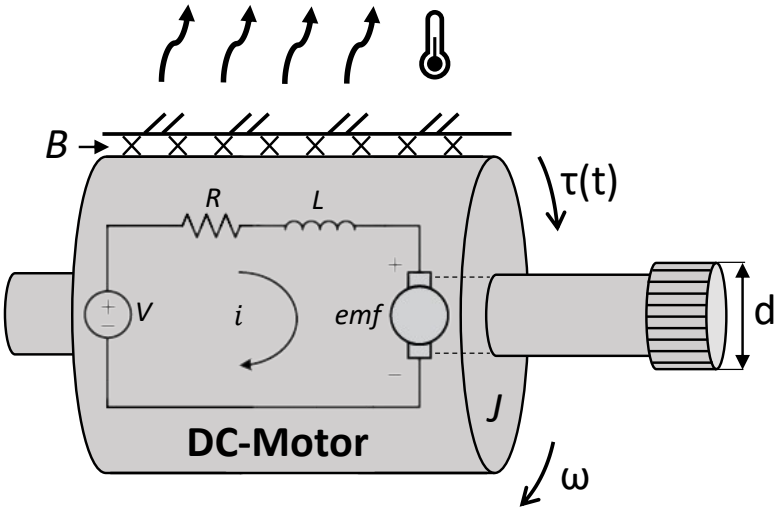


Fig. 3.3: A DC motor, an example of a multi-domain system (electrical, mechanical, thermal).

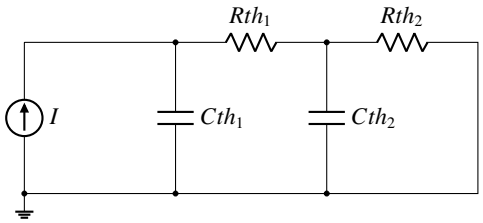


Fig. 3.4: Cauer network as the thermal model of the DC motor.

Listing 3.2 shows the Verilog-AMS code of the proposed system as a case study. There are two modules: one that implements the motor itself, including its electrical and mechanical equations (lines 1–38), and the other that realizes the Cauer network as a thermal module (lines 40–62). The branch `temp` represents the feedback of the thermal module on the parameters of the main system.

Let us now see the results of the thermal fault injection campaign on the DC motor. The goal is to highlight how the effects of faults belonging to one specific domain also impact other domains composing the system.

The system has been faulted in its thermal component by applying current sources, open faults, and varying the equivalent thermal parameters that characterize the motor. However, other fault models have been applied to the system, which do not belong to the thermal domain. Specifically, electrical faults have been injected into the original electrical component of the motor, and mechanical faults into the respective components.

As previously anticipated, one fault at a time has been injected into the system, thus simulating individual instances of motor fault situations. The fault models that have been applied for this case study, grouped for the different domains, are:

- **electrical** [24]: short, open, voltage source, current source, parametric;
- **mechanical** [25]: external damping effect, component disconnection, external force, limited movement, parametric;

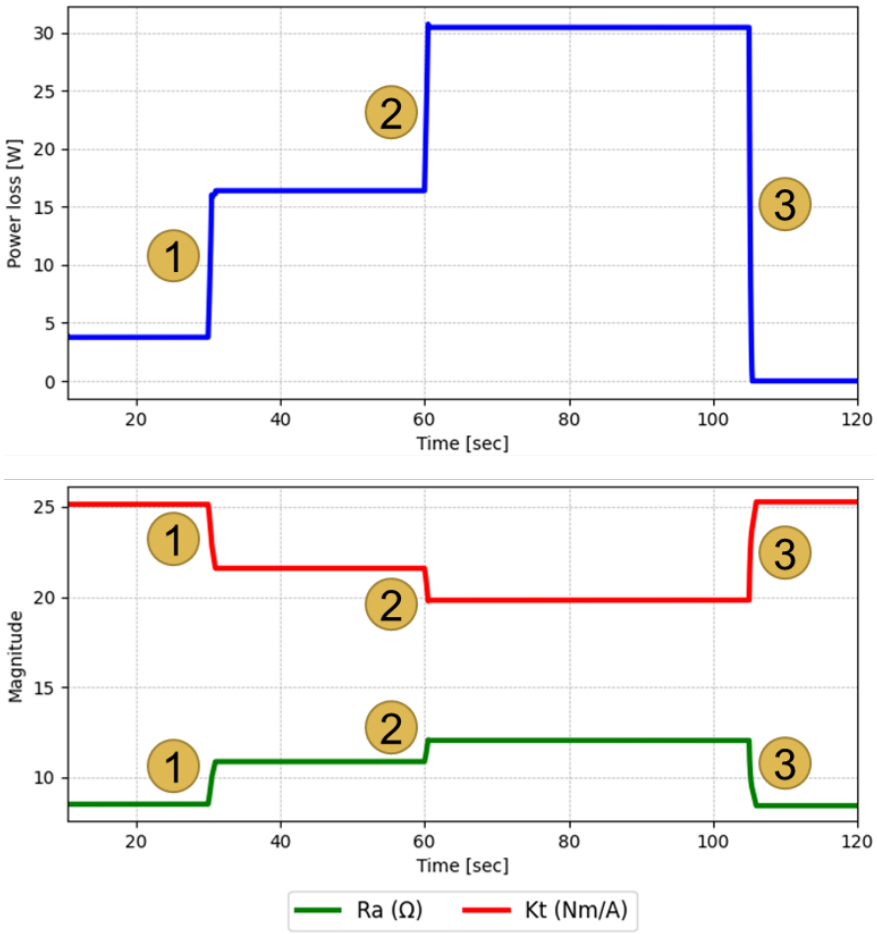


Fig. 3.5: Excerpt of the evolution of the DC motor power loss (upper graph) and internal mechanical parameters Ra (armature resistance) and Kt (torque constant) affected by an external heat source fault.

- **thermal**: external heat source, open thermal fault, parametric.

The results of this multidomain fault analysis are summarized in Table 3.2, and shown in the Figs. 3.6(a) to 3.6(c): Fig. 3.6(a) depicts the angular speed of the motor, Fig. 3.6(b) presents the temperature of the motor, while Fig. 3.6(c) shows the temperature on the shaft. Two fault observation points are considered in this analysis: the angular velocity of the motor and the temperatures at two points on the motor, one inside the motor and one on the shaft. The colors of the lines are common to all three graphs and represent the same scenario. The blue dashed lines represent the fault-free behavior of the system, which is stimulated with waves of different intensities: 12, 24, 48, and 64 V.

Let us now analyze the presented fault scenarios, which are represented by continuous lines and colored differently according to the injected fault model.

Regarding thermal faults, two models are shown in the figure: an abnormal heat source on the motor and an open fault, which increases the thermal resistance value of the motor. The first fault, named *thermal-source* and colored light blue, simulates the presence of an unwanted heat source on the motor, which impacts the mechanical movement of the motor by slowing

its rotational speed. This effect is caused by the increase in friction between the mechanical components: the temperature of the components rises, while the shaft speed decreases. The change in internal motor parameters due to the temperature increase is shown in the Fig. 3.5. Specifically, Fig. 3.5 shows how an increase in motor temperature corresponds to an increase in power dissipation (instant 1 and 2 in the figure) due to the change in the motor's internal parameters. These parameters are the electrical resistance of the armature (`Ra`, green color), and the mechanical torque constant (`Kt`, red color). In particular, temperature growth increases the value of `Ra` and decreases the value of `Kt`. At the moment the motor's power supply runs out, the motor slows down to stop its rotation, and so the power dissipated decreases (instant 3 in the figure). As a result, the `Ra` and `Kt` parameters of the motor also return to normal. The injected heat source takes effect after 30 seconds of simulation for a value of 50 W.

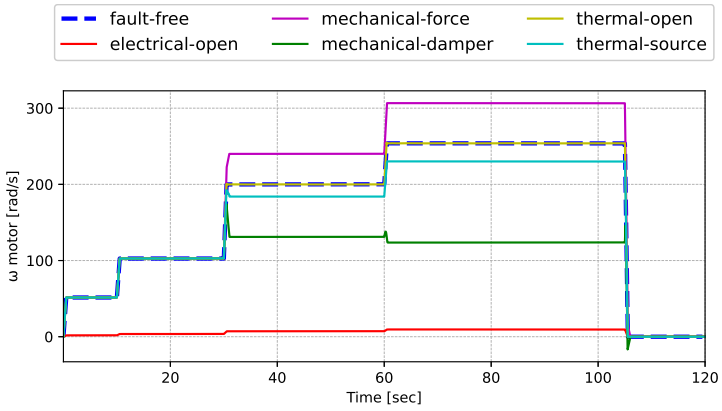
The second thermal fault model, called *thermal-open*, with a value of 1000 K/W and represented by the yellow line, is a change in the thermal resistance value of the system, which is increased. This fault causes a drop in the temperature rise of the motor components, as they are more insulated from the heat flow, while the angular velocity of the shaft remains unchanged compared to the fault-free version. The reported effects of thermal faults on temperature primarily affect the mechanical domain, impacting the motor's motion, and the electrical domain, increasing the power dissipated by it.

An open-type electrical fault, named *electrical-open*, has been injected with a resistance into the electrical part of the motor, simulating an increased resistance value of the motor's electrical circuit line. The resistor has been injected into the `ra` branch in the code shown in Listing 3.2, with the value of 1000Ω, and its response is highlighted by the red color. This fault causes a drastic drop in the power supply to the motor, which barely rotates, and the temperature of the motor and the shaft remains stable. Thus, an electrical fault model has direct effects on its own domain and consequential effects on the mechanical and thermal domains. This fault is injected by adding the following fault line in the Verilog-AMS code:

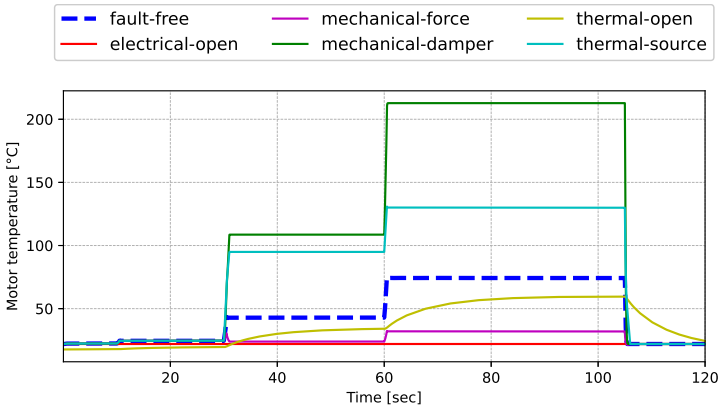
```
V(n1, n2) <+ I(n1, n2) * 1e03;
```

where `n1` and `n2` are the two nodes of the faulty branch. The open fault is injected in series to an existing branch, so one of the two nodes `n1` and `n2` has to be a new node, while the other is already existing.

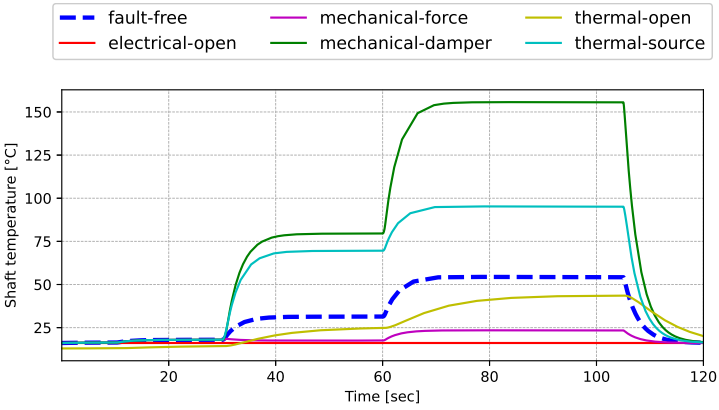
In addition, two mechanical faults were also injected and simulated: a *braking force* applied to the shaft and a force that contributes to the *shaft rotation*. In the first case, namely the *mechanical-damper* fault, the fault causes the motor to slow down its rotational speed, resulting in a drastic increase in power dissipation and, consequently, an increase in both motor and shaft temperatures. This faulty behavior is shown with a green line, and it has been injected after 30 seconds of the simulation, with a negative contribution of 50 Nm. In the second case, named *mechanical-force*, a positive contribution to the motor rotation has been injected. The motor rotates faster than it should: the higher this rotation is, the lower the amount of power is dissipated, reducing temperature growth. The violet line in the plots shows how motor speed increases from the time of fault injection (50 Nm after 30 seconds), while shaft motor temperatures grow much more slowly. Again, faults belonging to one domain, such as mechanical in



(a) Angular velocity of the DC motor in fault-free and faulty scenarios.



(b) Temperature of the DC motor's rotor in fault-free and faulty scenarios.



(c) Temperature of the DC motor's shaft in fault-free and faulty scenarios.

this case, also change mechanical behavior, as well as electrical and thermal ones. Simulations have been performed using the Verilog-AMS framework, instantiating the motor and Cauer network modules through a common testbench module. This top-level component powers the motor, providing the input voltage suitable for stimulating the applied fault correctly, with a given value, and at a specific time. The testbench has been built and refined during the analysis

to stimulate faults not only in the motor but also in the Cauer network, for example, by injecting heat into the branches of the thermal network. This type of analysis is necessary to define an optimal testbench that can stimulate the system in the context of functional qualification of the verification testbench [28], even in a multi-domain system. Moreover, verifying the testbench through fault simulation is useful for validating the system digital twin, *i.e.*, the DC motor.

Table 3.2: Fault Injection Campaign Summary for the DC Motor Case Study

Domain	Fault Models	Injected Faults	Est. Sim. Time
Electrical	Open, Short, V-Source, I-Source	30	~ 8 mins
Mechanical	Damper, Disconnection, Force, Limited Mov.	8	~ 2 mins
Thermal	Heat Source, Open Thermal, Parametric	3	~ 1 min
Total		**41**	**~ 11 mins**

Note: Simulation times are estimated CPU elapsed times.

3.5.3 Unreal Engine simulation

Game engines, which are part of the world of gaming product development, are powerful tools for creating interactive entertainment models. In recent years, the capabilities and accuracy of these tools have been growing dramatically. One such tool is Unreal Engine, a game engine developed by Epic Games that integrates an event-driven and continuous-time simulation engine [58, 59]. This software has the assets to develop simulations with high visual impact while maintaining remarkable physical accuracy [60]. One of the main features of game engines is the ability to visualize the behavior of the design under development in real time [61]. This is a primary feature of video game development but is very attractive for digital twin development. By combining the accuracy of graphical model reproduction with advanced physics modeling, we achieve a simulative environment not only for video game purposes but also on a technical and experimental level. In game development, the accuracy and precision of the reproduced reality are often compromised by the complexity of all the processes belonging to the game level. However, if the aim is to reproduce an industrial system, many purely gaming-related components would not be considered. This increases the depth achievable from the level of physics simulated within the game engine while keeping optimal simulation performances. With its advanced graphics rendering capabilities, real-time physics simulation, and robust scripting tools, Unreal Engine presents an exciting platform for engineers and researchers seeking to model and analyze complex industrial systems [62]. In this context, system modeling is advantageous for improving and optimizing the design and test phase before production. Unreal Engine seems an attractive environment for designing and simulating a system and creating a model of an existing machine [63]. Moreover, Unreal Engine's modeling capabilities can be exploited to simulate a system in nominal conditions and analyze its behavior in the presence of faults. Simulation of fault behaviors is crucial for studying system functional safety levels in industrial settings. Identifying the causes and ways in which a fault occurs enables not only the detection of faults

but also the prevention via Predictive Maintenance algorithms. Based on these premises, we propose this methodology focused on gaming engines for these reasons (see Fig. 3.6 for an overview):

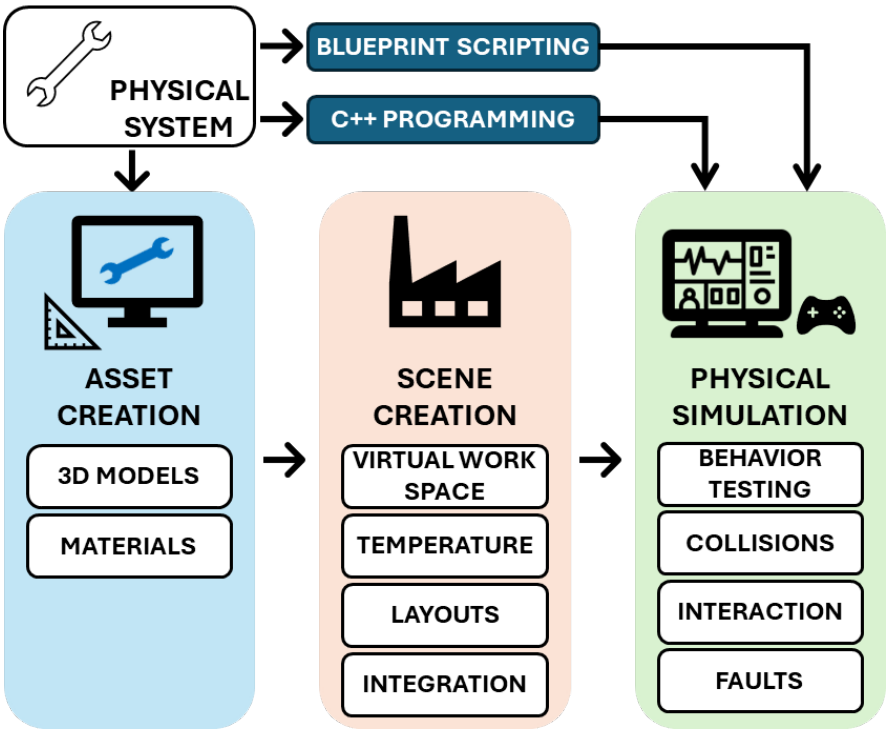


Fig. 3.6: Typical workflow adopted for the modeling and simulation of a physical system with the Unreal Engine gaming environment.

- Using the powerful programming components of a gaming engine to create and simulate a model of a DC motor;
- Combining graphical qualitative evaluation of modeled system behavior with precise measurement through data traces reported in appropriate graphs;
- Simulation of faulty behaviors of the developed system through injection of the fault models directly into the model code;
- The simulated system behavior coded in C++ can influence the 3D environment and receive feedback, allowing the variation of the equation's parameters based on these simulated physical properties, like gravity or collisions between objects.

3.5.4 Unreal Engine functional modeling

A specific and precise modeling procedure must be adopted to create and simulate a physical system inside a game engine. Performing the modeling steps sequentially promotes a smoother development: Fig. 3.6 is a diagram flow of this procedure.

System's Features Design and Description

As a first step, defining the system's assets is necessary. The graphics visualization part is crucial as it is one of the most characteristic aspects of game engines. Therefore, obtaining 3D models of the system or creating them through design tools is convenient. Defining the amount of meshes in the scene helps estimate the structural complexity to be achieved. However, the greater the number of meshes in the scene, the greater the impact on simulation performance. Another factor that increases visual complexity is the level of detail of the 3D assets used to create the model. Moreover, Unreal Engine allows the specification of the texture material of the 3D models [43]. This detail allows much more accurate modeling of the replicated system, both graphically and physically. Several physical properties can be modeled thanks to materials: any surface can be created, changing the physical properties of components deeply. Additionally, the weight of objects and their response to the surrounding environment can be determined. Both collisions, physics constraints, and rotation joints, which replicate the connections and constraints in the physical system, can be customized for each mesh separately. This allows for realistic movement and interaction between interconnected components. Once the 3D modeling part is complete, the created system is placed within the scene, i.e., the context in which it operates. This aspect is crucial when the system's interaction with other machinery or the external environment needs to be replicated [64]. For example, if the thermal aspect is critical for our system, we need to recreate the temperature of the environment to which it belongs, as well as any additional heat sources. When the system must interact with other previously modeled systems, integration into the overall functional context and responsiveness to stimuli from these systems are crucial. Once the graphical environment is prepared, we proceed to animate the 3D components in the scene. From the perspective of a physical system, this is achieved by modeling its dynamics. In Unreal Engine, this process can be accomplished through two tools: Blueprint and C++ programming. Development through Blueprint appears to be simpler and more intuitive due to its modular interface-based structure. This is particularly useful for users who are not experienced in programming, as well as for rapid prototyping and testing purposes without requiring any code. However, as a visual and interpreted scripting language, it has two main drawbacks compared to its C++ counterpart. The first is that C++ performance is much better overall at the simulation level for almost any system model. The second is that C++ programming allows the reproduction of very complex logic or models that require high levels of performance optimization. Additionally, in C++, it is much easier to reuse code and utilize features from libraries outside of Unreal Engine. To summarize, Blueprints offers rapid prototyping and ease of use, while C++ provides better performance, flexibility, and low-level control for complex simulations. After the development phase, the system modeled in Unreal Engine is tested. Factors often refined after the first tests include collision areas, i.e., whether and how the 3D models interact with each other, the surrounding environment, or the system's behavioral dynamics.

Behavior Modeling through C++ Programming

This work aims to demonstrate that Unreal Engine simulations, although originally designed for video game development, can also be leveraged for various purposes. In particular, functional safety aims to ensure the proper functioning of a system despite any adversities or faults. However, these studies require considerable effort to analyze the system's behavior under test. To ensure system functionality, one must first understand how the system can fail and what abnormal events may occur. An accurate simulation of the system behavior can help in analyzing the system response in a variety of working conditions, both normal and with faults. The physical behavior of a physical system is often described through differential equations. Within Unreal

Algorithm 1: Pseudo-code of the tick function called by the game engine.

```
1 function tick(CurrentTime, DeltaTime):
2     if DeltaTime > Threshold then
3         while Time ≤ (CurrentTime + DeltaTime) do
4             Perform a dynamic integration step;
5     else
6         Perform a single fixed integration step;
```

Engine, it is possible to accurately replicate the behavior identified by the equations through C++ programming. Moreover, by calculating the system's behavior, we get not only a numerical representation but also a visual one. Although waveform evaluations are more mathematically accurate, visual simulation helps us clearly visualize how the system reacts to certain stimuli. So, to model the dynamics of a multi-domain system, we describe its differential equations within the logic expressed by the C++ code. However, reproducing and simulating differential equations in C++ is not trivial. Solving differential equations associated with complex systems requires specialized methods based on numerical integration. Once the numerical method has been selected, we need to implement the algorithm in C++. This involves coding the mathematical equations, defining data structures to represent the state variables, and writing the numerical solver function that iteratively updates the state variables over time. Algorithm 1 exemplifies the approach we propose for integrating continuous-time simulation into the C++ code called by the game engine. The C++ function `tick`, containing the continuous-time model, is called at irregular intervals by the game engine, based upon the *frame rate*. Specifically, when the frame rate is too low, the simulated time span between one frame and another is high. This results in a *DeltaTime* provided to the `tick` function, which is also high. That is why in Algorithm 1, when the *DeltaTime* is above a given threshold, we need to perform multiple steps of numerical integration with a dynamic step. Otherwise, performing a single large integration step with a large *DeltaTime* can result in incorrect dynamics. We tested this approach and were able to reproduce the correct dynamics by performing multiple integration steps within the `tick` function. This iteration over time is very convenient in Unreal Engine: as a game engine, recalculating the dynamics of the actors in the scene is inherent in its normal operation. So, the iterative step of

calculating system behavior is placed within the `tick` method. Differential equations typically require initial conditions (e.g., initial values of state variables) and parameters (e.g., constants or coefficients) to be solved; those values are specified as parameters of the C++ class. In this way, we can accurately simulate the system's behavior, which is also recognized as the state of the art. Moreover, this behavior will be visualizable on a 3D moving model, which can be influenced by external conditions.

3.5.5 Experimental results

In this section, we present the experimental results achieved so far concerning the modeling of a case study and its simulation in Unreal Engine. In addition, some fault behavior has been modeled and simulated. We chose a reuse the DC motor as the first system to be built and run in Unreal Engine. This component is part of several industrial machines and a multidomain system. The equations which control the motor are Eqs. (3.2) and (3.3), and the parameters are the same as Table 3.1.

The behavior of this code simulated through the game engine is shown in Fig. 3.7. The graphs represent both the system's nominal behavior (dashed blue line) and three abnormal behaviors caused by behavioral faults injected within the motor. The motor acceleration is the

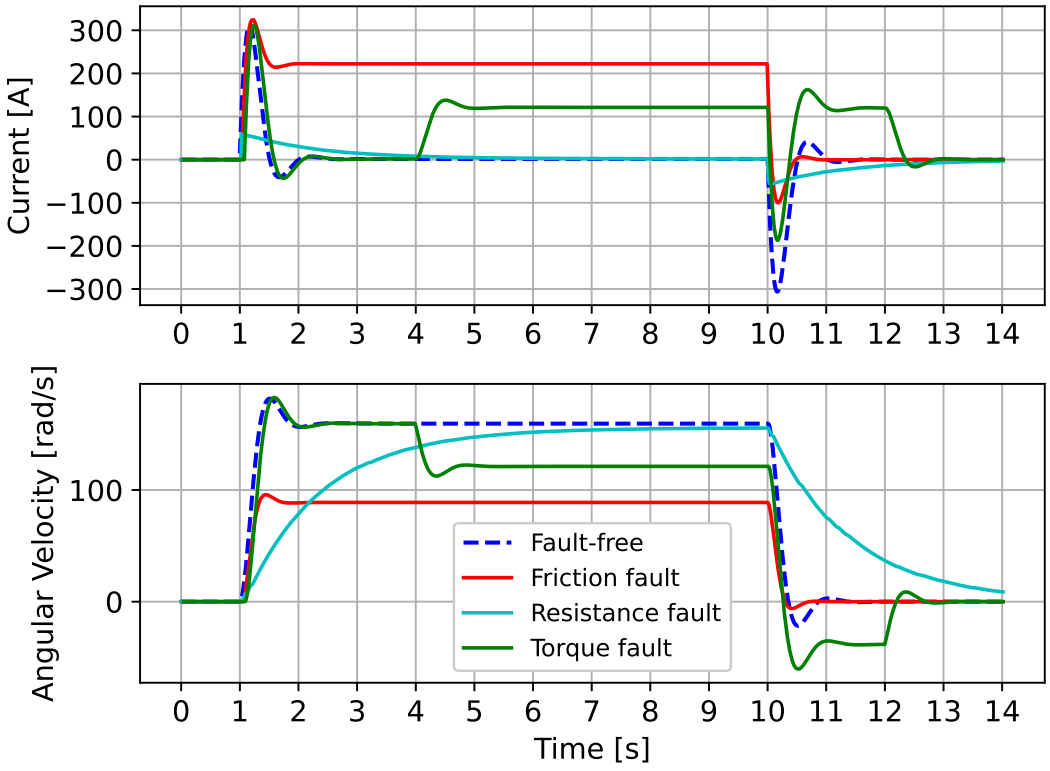


Fig. 3.7: Simulation results of both fault-free (dotted blue line) and faulty DC motor.

result of a 200 V input voltage that begins one second after the start of the simulation and lasts nine seconds. Specifically, the red line identifies a fault regarding the friction inside the motor. Several possible factors (e.g., dust, high temperature, etc.) could increase the coefficient of friction of internal mechanical components, reducing the maximum angular velocity reached by the motor. The fault is active for the whole simulation duration to simulate a more realistic fault condition. Applying a load torque to the shaft's rotation can produce a similar effect (represented by the green line). In this case, the fault is transient, i.e., its effect is not active for the entire simulation duration, but it activates after four seconds of simulation and remains active for eight seconds. Finally, the electrical resistance of the motor armature was increased (cyan line), simulating an internal electrical fault. The constant activation of this fault throughout the simulation prevents current from flowing through the windings normally, resulting in significantly slower motor acceleration than usual.

3.6 MEMS 3-axis accelerometer

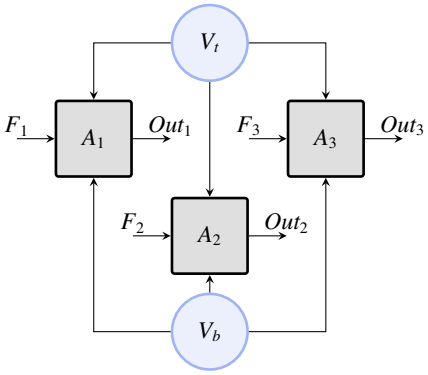
As the second case study, we chose the MEMS model of a 3-axis accelerometer, shown in Fig. 3.8. Until a few years ago, accelerometers were intended mainly for scientific, military, or civilian uses. However, due to the recent evolution of electronics, reduction of costs, and development of applications, accelerometers are widely used on everyday objects [65]. The sensor considered in this article was made by combining three individual accelerometers [66], orienting them in three different directions perpendicular to each other.

The structure of the 3-axis accelerometer is depicted in Fig. 3.8(a). The three accelerometers, identified as A1, A2 and A3, are identical and, for this reason, powered by the same voltage sources V1 and V2. Each accelerometer is stimulated by different external forces, F1, F2, and F3, since they are positioned orthogonally, thus producing three different output voltages. The internal structure of each single-axis accelerometer is shown in Fig. 3.8(b). Like the motor, this model is also a system composed of mechanical and electrical parts. First, the acceleration of the seismic mass is caused by the external force F. The displacement of the mass, limited by the spring and the damper, is directly proportional to the intensity of the acceleration experienced by the system. This purely mechanical behavior is expressed by Eq. (3.8),

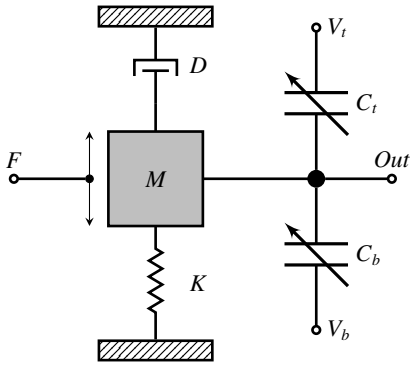
$$F(t) = Kx + D\frac{dx}{dt} + M\frac{d^2(x)}{dt^2} \tag{3.8}$$

where x is the displacement of the mass, M is the mass weight, D is the damping factor, and K is the spring stiffness. Thus, the displacement exerted by the mass affects the accelerometer's electrical components. It consists of two capacitors, both connected to a dedicated voltage source. The mass forms the middle plate of the differential capacitive circuit: the spacing between the plates changes when the mass is no longer in equilibrium, thereby altering the capacitive value of the entire circuit. These changes are expressed by Eq. (3.9),

$$C_t = \frac{\varepsilon A}{d_0 - x}, \qquad C_b = \frac{\varepsilon A}{d_0 + x} \tag{3.9}$$



(a) Structure of the 3-axis MEMS accelerometer system model.



(b) Schematic of the internal components of a single accelerometer model [66].

Fig. 3.8: The overall structure of the second case study.

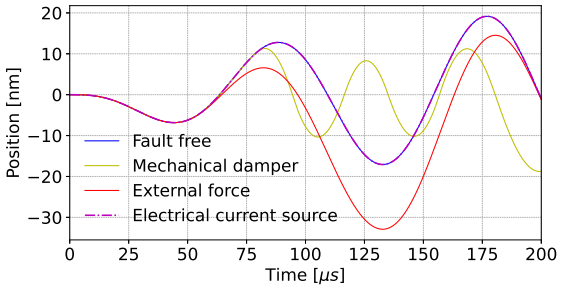
where A is the area of the capacitor plate, d_0 the gap between the plates at equilibrium, and ε is the vacuum permittivity constant (8.85e−12 F/m). Finally, Eq. (3.10) shows the behavior of the currents flowing through the two physical capacitors, producing the output voltage value of acceleration.

$$I_t = C_t \frac{dV_t}{dt}, \qquad I_b = C_b \frac{dV_b}{dt} \tag{3.10}$$

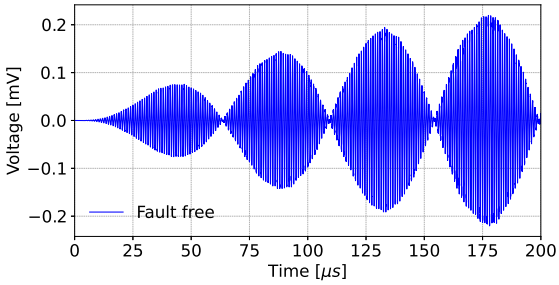
3.6.1 Multidomain fault injection and simulation

Moving to the fault simulation, the fault models have been injected into both physical disciplines that form the system. The statistical results of this fault simulation campaign are reported in Table 3.3. In the electrical part, we injected 6 open circuits, 27 short circuits, 6 voltage sources, and 27 current sources. In the mechanical part, we injected 12 faults, 3 for each present in the taxonomy (see Table 2.2). The model was stimulated under several scenarios using different forces applied to the system, e.g., shocks via force pulses or continuous acceleration via sine waves. Fig. 3.9 shows the simulation results for this second case study. In particular, all mass displacements related to the different scenarios, fault-free and faulted ones, have been included in Fig. 3.9(a). The colors represent different working conditions and each color is related to

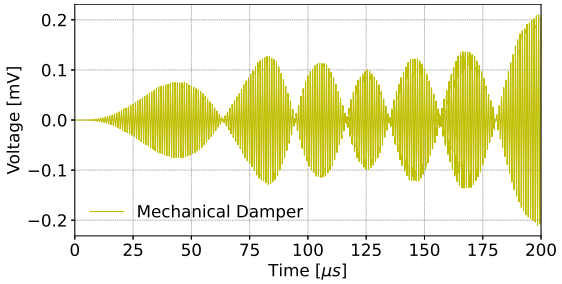
its own acceleration output in the next plots. For example, the fault-free displacement is represented with a blue line, and the related acceleration value, depicted in Fig. 3.9(b), is drawn with the same line format. The same rule applies to all the faulty behaviors shown as well. The model was simulated for 200 µs, and it was fed with a sinusoidal acceleration. Fig. 3.9(a) and Fig. 3.9(b) highlight that the measured acceleration is greater when the mass movement is more intense. Let us now analyze some examples of faulty behaviors in the MEMS accelerometer.



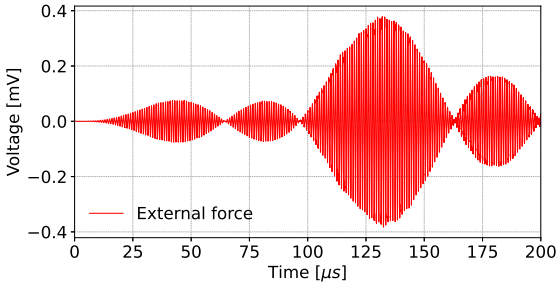
(a) Displacement of the internal mass of the accelerometer.



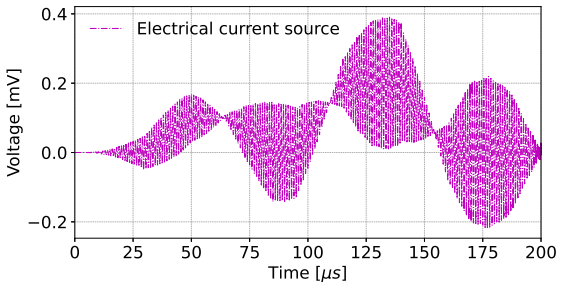
(b) Nominal acceleration output of the accelerometer.



(c) Damped acceleration value of the accelerometer.



(d) Modified acceleration value due to an external force.



(e) Modified acceleration value due to an external current source.

Fig. 3.9: Fault-free and faulty accelerometer waveforms.

Fig. 3.9(c) represents the model's response when suffering a damping fault, which can be caused by many factors, such as an increased friction value or an object that slows the movement of the mass. This fault has been represented both in Fig. 3.9(a) and Fig. 3.9(c) with a yellow line. This time, the fault is transient, meaning that it is not active for the entire simulation duration but only for a limited period. The motivation behind this choice is just to model a more realistic

fault scenario. The duration of the effect of this fault is 100 µs, specifically, from 80 µs to 180 µs. While this fault is active, the oscillation of the mass is limited (see Fig. 3.9(a)). Consequently, the accelerometer's measured acceleration will be incorrectly lower (see Fig. 3.9(c)).

Another fault model tested on the system is the unexpected external force shown in red. In particular, the system has been subjected to a force not included in the input provided to it. In this scenario, the mass undergoes a displacement not due to the force acting on the whole system, as shown in Fig. 3.9(a). As a result, the accelerometer measures an acceleration that does not fully reflect the force applied to the system, but rather the force indicated by the erroneous displacement of the mass. This transient fault is active from 50 µs to 130 µs, for a total duration of 80 µs. For this reason, the acceleration measured after the fault disappears reflects the behavior of the fault-free model.

Finally, Fig. 3.9(e) shows the effect of an electrical fault injected on the upper capacitor of the sensor, C_t. The fault represents an anomalous current source connected to the branch, thus adding additional current to the capacitor's current. Again, this is a transient fault: the source is active from 30 µs to 65 µs, and from 95 µs to 135 µs, but with doubled intensity. Fig. 3.9(e) depicts clearly how the current source changes the intensity of the acceleration measured by the accelerometer, changing the amplitudes in the first three oscillations.

Table 3.3: Fault Injection Campaign Summary for the MEMS Accelerometer Case Study

Domain	Fault Models	Injected Faults	Est. Sim. Time
Electrical	Open, Short, V-Source, I-Source	66	~ 17 mins
Mechanical	Damper, Disconnection, Force, Limited Mov.	12	~ 3 mins
Total		**78**	**~ 20 mins**

Note: Simulation times are estimated CPU elapsed times.

3.7 Landing gear system

Landing gear systems ensure aircraft safety during the critical phases of takeoff and landing. These systems, accounting for 2.5%-5% of the maximum takeoff weight and 1.5%-1.75% of the aircraft's cost, disproportionately contribute to 20% of the airframe's direct maintenance costs [67,68]. Hydraulic systems are primarily used for landing gear retraction/extension due to their ability to achieve smooth and precise speed control. However, ensuring the integrity and functionality of these systems necessitates robust fault detection and diagnosis strategies. The traditional *time-based maintenance* approach for aeronautical systems is undergoing a significant paradigm shift. By 2035, *condition-based maintenance* is expected to become the standard, with a further transition towards *predictive maintenance* utilizing Artificial Intelligence (AI) algorithms for real-time health monitoring [69–71]. This shift relies heavily on the availability of a vast amount of high-quality data, encompassing both healthy and faulty system behavior. However, acquiring or accessing real-world fault data from intricate aircraft systems, such as

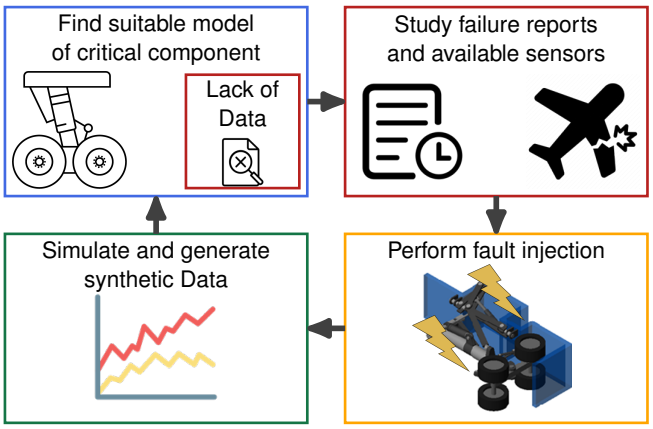


Fig. 3.10: Overview of the proposed methodology to model and inject multi-domain faults into the landing gear systems through differential equations to generate fault synthetic data.

landing gear, is challenging and often cost-prohibitive [72]. Deliberately causing faults in real systems can be unsafe and infeasible, rendering simulation a critical tool. The aviation industry has decades of experience with model-based design, making it a natural fit for fault injection.

This work adopts a model-based approach to investigate the consistency between a system's actual and expected behavior in Landing Gear System (LGS). We introduce a novel methodology for gathering synthetic fault data by employing fault injection techniques within an LGS model (see Fig. 3.10). This approach facilitates the generation of synthetic fault data, bridging the gap imposed by the limitations of real-world data acquisition and enabling the exploration of a wide array of potential failures [73]. The generated synthetic data, encompassing both healthy and faulty system behavior, is instrumental in developing and validating robust PHM algorithms for future LGS, paving the way for a future of predictive maintenance in aviation.

The contributions of this research are delineated as follows:

- **Model-Based Fault Analysis with Synthetic Data Generation**: We propose a model-based approach for fault analysis in LGS. This approach is particularly valuable due to the limitations of acquiring real-world data for all possible fault scenarios. The study presents a methodology for generating synthetic fault data by utilizing fault injection techniques within the model. This methodology can be adapted to cover a broad spectrum of potential failures in various critical aircraft components, not limited to LGS.
- **Focus on Practical Relevance**: The selection of fault scenarios for detailed analysis prioritizes those with a high impact on real-world LGS operations. This ensures the practical relevance and applicability of the findings to improving landing gear safety and reliability.
- **Foundation for Future Research**: The study establishes a foundation for future research in LGS health management. The generated synthetic fault data paves the way for developing and refining robust predictive maintenance algorithms. Additionally, the research highlights the potential of merging model-based and data-driven approaches for even more effective LGS health management strategies.

While this research focuses on aircraft subcomponents, the proposed methodology can be applied to Cyber-Physical Systems and Industrial Cyber-Physical Systems, showcasing its versatility and relevance across domains where fault tolerance and system integrity are paramount.

3.7.1 Evolution and challenges in aircraft maintenance

The aviation sector stands at the threshold of a significant transformation, driven by the advancements in condition-based and predictive maintenance methodologies. This evolution reflects a broader industry trend towards embracing real-time, data-driven strategies for aircraft maintenance, which promises to redefine operational efficiencies and safety protocols. Central to this shift is the vision articulated by the Advisory Council for Aviation Research and Innovation in Europe (ACARE), which envisions condition-based maintenance becoming the state-of-practice by 2035 [71]. ACARE's projection is not merely aspirational but signals a decisive move towards adopting maintenance frameworks that are adaptive, predictive, and optimized for the dynamic demands of aviation operations. Presently, the industry largely operates under time-based maintenance protocols, where maintenance activities are scheduled at fixed intervals, regardless of the actual condition of the aircraft components. This approach, while systematic, often leads to either over-maintenance or under-maintenance, with the former incurring unnecessary costs and the latter posing risks to safety and reliability. Time-based maintenance's reliance on predetermined schedules rather than real-time data and condition monitoring represents a significant area where efficiency gains can be realized. The limitations of time-based maintenance have become increasingly apparent in light of the burgeoning capabilities for data collection and analysis offered by modern aircraft. These technological advances provide a compelling case for transitioning to condition-based and predictive maintenance methodologies, where decisions are informed by the actual wear and performance of aircraft systems. This transition is not merely technical but is also financial, with the industry grappling with the cost implications of maintenance strategies. In 2018 alone, airlines globally expended approximately $69 billion on maintenance, repairs, and overhaul, accounting for 9% of their total operational costs, as underlined in a recent review [72].

The escalation in maintenance requirements and data collection capabilities presents both a challenge and an opportunity for leveraging big data and AI technologies to enhance efficiency and reduce operational costs. With its foundation in real-time data analytics and extensive sensor networks, predictive maintenance promises to revolutionize aircraft maintenance by preempting equipment failures and optimizing repair schedules, enhancing operational safety and efficiency. Yet, this promising transition is fraught with complexities. Integrating AI and Machine Learning (ML) into predictive maintenance strategies entails navigating the burgeoning complexity of aircraft systems and the voluminous data they generate. The stark contrast between older aircraft models with fewer sensors and newer, more sensor-laden models necessitates adaptive predictive maintenance approaches, ensuring maintenance practices evolve alongside technological advancements.

The review also underscores the burgeoning potential and existing hurdles of predictive maintenance in aviation [72]. A critical challenge emerges in the scarcity of publicly avail-

able datasets for aircraft-specific components, especially hydraulics, which play a pivotal role in critical systems like landing gear. This scarcity hinders the development of bespoke predictive maintenance algorithms, emphasizing the need for industry-wide collaboration and open data initiatives. Further, studies comparing ML algorithms against hydraulic system data reveal an intriguing insight: traditional methods with feature engineering often surpass deep learning models in data-constrained scenarios. This not only highlights the limitations of current AI approaches but also points to the potential of hybrid methodologies that blend traditional model-based techniques and data-driven ML techniques for predictive maintenance.

3.7.2 Fault Injection for Synthetic Data Generation in Aircraft

The proposed methodology outlines a general approach for fault injection and synthetic data generation applicable to various critical aviation components. While the specific example used here focuses on LGS, the core steps can be adapted to other aviation systems by replacing component-specific details. The general flow of this methodology is shown in Fig. 3.11.

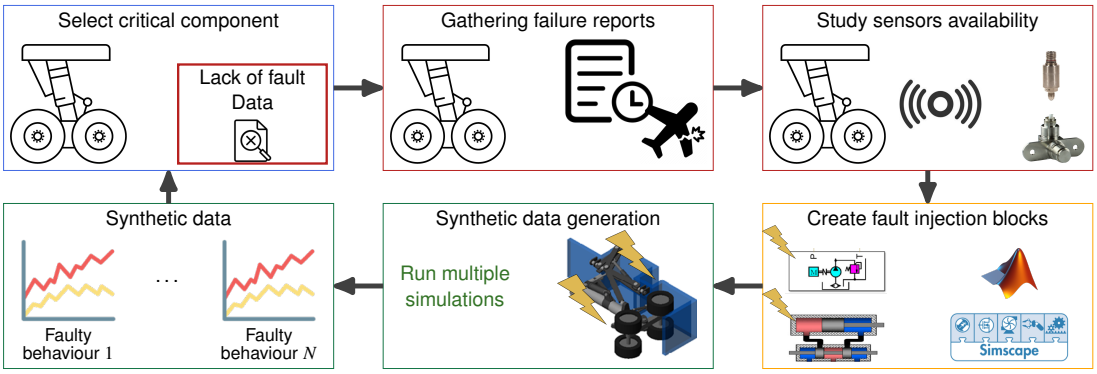


Fig. 3.11: Overview of the methodology for fault injection and synthetic data generation, with an example of a LGS. This approach encompasses critical component selection, literature analysis, sensor capability assessment, development of custom fault injection blocks, and synthetic data generation.

Select critical component

As depicted in Fig. 3.11, the first step involves selecting the critical component within the chosen aviation system for fault injection analysis. This selection can be based on factors like:

- **Safety Significance**: Components crucial for the aircraft's safe operation are given priority. Ensuring the reliability of these components is paramount to overall aviation safety.
- **Historical Failure Data**: Components with a higher frequency of reported failures should become prime candidates.
- **Availability of High-Fidelity Models**: The selection process is also influenced by the existence of high-fidelity models or the feasibility of developing such models for the component in question. High-fidelity models are essential for accurately simulating the component's behavior under normal and fault conditions, thereby ensuring the effectiveness of the fault injection analysis.

Gathering failure reports

A critical step in our methodology is conducting a thorough literature analysis of potential failure modes associated with the selected critical aviation component. This analysis is essential not only for engineers specializing in simulations who might not have extensive mechanical experience but also for more experienced technicians accustomed to hands-on work with these systems. By reviewing a wide range of data sources, including industry reports [74–77], academic publications, and incident databases [78,79], professionals across the spectrum can deepen their understanding of component behaviors and failure mechanisms. This comprehensive literature analysis empowers engineers and technicians alike to expand their knowledge base, enabling them to anticipate and simulate a broader range of failure scenarios. For engineers, it enhances their ability to create more accurate and comprehensive simulation models. For technicians, it provides a broader perspective on potential failure modes, enriching their practical experience with a deeper theoretical understanding. Ultimately, this approach fosters a more holistic understanding of aviation systems' vulnerabilities.

Study sensors availability

This phase of our methodology is devoted to examining the sensors equipped with the critical component, focusing on the types of data they collect. Understanding sensor capabilities and the data they provide is crucial for configuring our simulations to accurately mirror the system's behavior and performance under various fault conditions. It is essential to recognize that in a real-world scenario, components are not equipped with an unlimited number of sensors due to constraints such as cost, space, and practicality. Therefore, simulations strive to emulate these real-world limitations, ensuring that the sensor distribution within the simulation closely reflects what would be feasible on the actual component.

Create fault injection blocks

This step details the development of custom blocks within the chosen simulation environment (e.g. Simscape) to inject various faults into the model. These blocks aim to replicate the identified failure scenarios and enable a comprehensive analysis of the system's behavior under fault conditions. Leverage the simulation environment's capability to define custom components with functionalities tailored for fault injection. These blocks should be:

- **Parameterizable**: Allowing for customization of fault severity levels, providing flexibility in simulating various fault conditions from mild to severe.
- **Mathematically Represented**: Utilizing equations to simulate the desired fault behavior. Each block utilizes mathematical equations to simulate the specific behaviors associated with each fault accurately, ensuring the simulation outcomes closely mirror real-world phenomena.
- **Easily Enabled/Disabled**: These custom blocks should be designed for easy enabling and disabling, preserving the integrity and readability of the original model. This capability

maintains the system model's clarity by allowing users to quickly switch between the default and fault-injected states without modifying the model's fundamental structure.

Synthetic data generation

The final phase of this methodology involves generating synthetic data through fault injection experiments. This process involves executing simulations incorporating various fault conditions to assess the system's response. The data collected from these simulations serves as a foundation for subsequent analysis and model validation.

3.7.3 Model evaluation

This section demonstrates the application of our proposed methodology to LGS model, illustrating its practical implementation and effectiveness.

Choice of the critical component

Fig. 3.12 depicts the case study we chose; it is a LGS model that uses Simulink for control logic and Simscape for the mechanical and hydraulic subsystems. Beyond producing traditional data outputs like scope views and waveform plots, our simulation enhances comprehension through animated visualizations of the landing gear mechanics. This feature provides an intuitive understanding of deployment, locking, retraction, and unlocking sequences. This model is particularly valuable for its multidomain approach, accurately reflecting the integration of control, mechanical, and hydraulic systems within a single simulation environment. The simulation incorporates animation beyond static analysis, offering dynamic insights into the system's operational behavior. Such a multidomain and visually enriched simulation approach aligns with state-of-the-art aviation systems, offering a nuanced case study that bridges theoretical analysis with practical, real-world operational dynamics.

Gathering failure reports

To understand the fault landscape for effective synthetic data generation, our literature review targets hydraulic landing gear components, pinpointing the most common failure modes identified over the past century. This focused investigation draws from the following:

- **Industry Reports**: Aviation safety organization reports analyze trends in landing gear incidents, highlighting frequent failure modes like actuator leaks and pipeline wear [74–77].
- **Academia Articles**: Academic publications on LGS provide valuable information on the actual state-of-the-art in the academic research world regarding the simulations and the practices adopted to study these mechanisms [80–82].
- **Incident Databases**: Aviation incident databases like The Aviation Herald [79] and Aviation Safety Network [78] provide real-world insights into landing gear malfunctions and root causes, with The Aviation Herald established in 2008 and recognized for its independence. The Aviation Safety Network has operated under the Flight Safety Foundation since 1945.

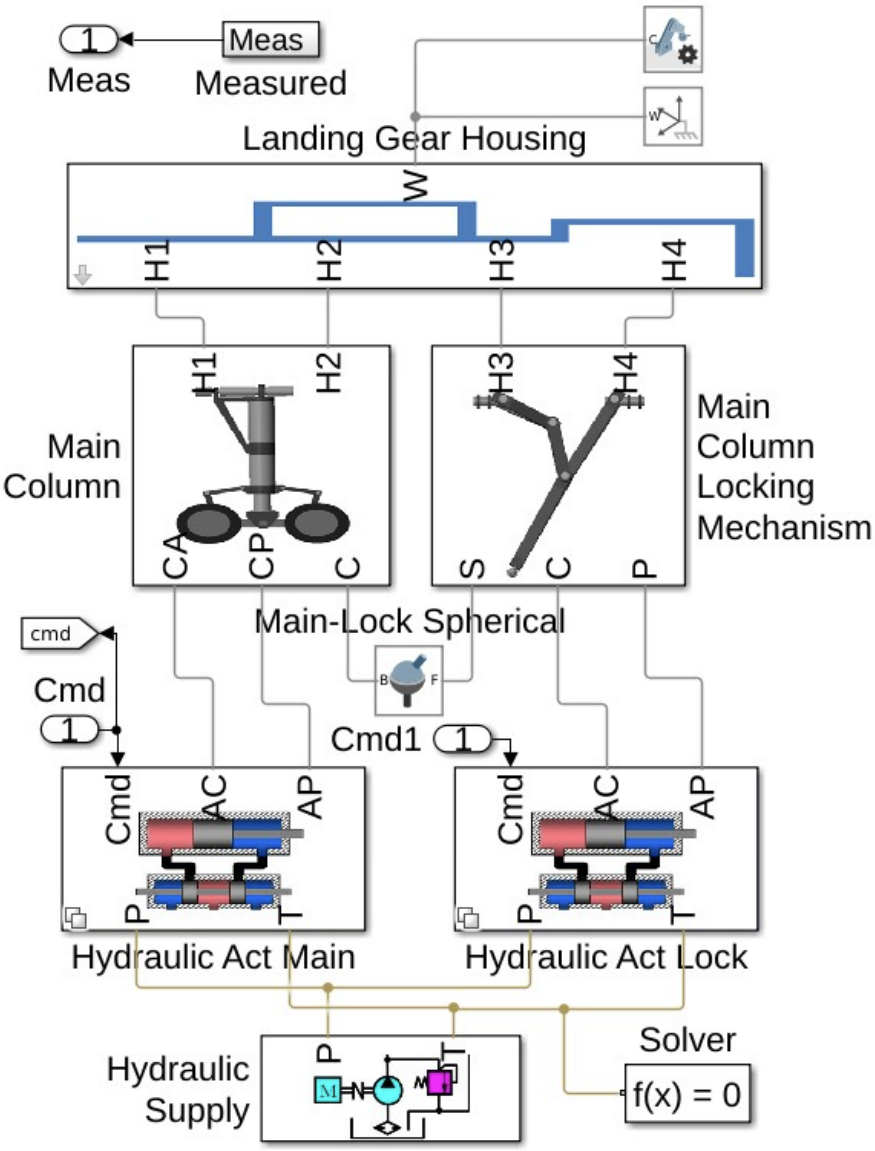


Fig. 3.12: Close-up view of the Simscape model representing the LGS [80]. Each block within the model represents a specific component or subsystem. These blocks can be further decomposed into sub-blocks, which mathematically describe the behavior of individual mechanisms using equations.

Our targeted literature review culminates in identifying the most recurrent faults documented through extensive research and analyses. The following faults were consistently highlighted as prevalent issues:

1. **Oil straining**: The obstruction of oil flow due to contaminants leads to reduced hydraulic efficiency.
2. **Oil filter blocked**: The clogging of oil filters hampers the cleanliness of the hydraulic fluid, affecting system performance.

3. **Pipeline leakage**: The deterioration of hydraulic lines over time can lead to leaks and reduced system pressure.
4. **Actuator external leakage**: The loss of hydraulic fluid from actuators when the hydraulic fluid flows from the system to the outside environment.
5. **Actuator internal leakage**: The loss of hydraulic fluid from actuators due to damage to actuator seals, which close the gap between the actuator piston and the cylinder wall.
6. **Stuck reversing valve core**: The malfunctioning of valves that control hydraulic flow direction, hindering the system's operation.
7. **Oil contains wear particles**: Metallic particles in the oil can wear down hydraulic components and reduce system life.
8. **Oil mixed with air**: Air entrapment within the hydraulic fluid leads to inefficiencies and potential system failure.

Sensors availability

After the cataloging phase, it is imperative to determine which onboard sensors are available to monitor these specific faults. This step is crucial for acquiring pertinent data that reflects the system's operational state and performance while in service. Based on common sensors found in LGS, the following parameters can typically be tracked:

- **Proximity**: These sensors are typically mounted on actuators and wheels. Proximity sensors detect the extension and retraction positions on actuators, providing binary or continuous positional data. Regarding wheels, they can measure the weight on wheels, indicating whether the aircraft is airborne or on the ground, which is critical for ensuring the correct timing of landing gear operations.
- **Pressure**: Such sensors monitor the hydraulic system's pressure levels, offering insights into the state of the entire aircraft's hydraulic support. Pressure sensors play a crucial role in detecting leaks, blockages, or failures in the hydraulic system that could impact the deployment or retraction of the landing gear.

While these types of sensors can be found almost on every LGS, the quantity and quality of the sensors, and consequently the data they provide, can vary significantly between different aircraft models and manufacturers.

3.7.4 Fault injection custom blocks

We have developed custom blocks designed to simulate various fault scenarios to advance the fault injection capabilities for the LGS model within Simscape. These blocks replicate the operational conditions under which faults like actuator friction and hydraulic supply limitations can impact the system's performance. Through the Simscape language, these custom components are detailed to capture the dynamics of faults using a foundation of mathematical equations and physical connections.

Actuator Friction Block

This block builds upon the translational friction model [83], extending its functionality to simulate the friction experienced by actuators. The model incorporates Stribeck, Coulomb, and viscous friction components to replicate the frictional forces accurately. These forces are crucial for understanding how friction can affect the deployment and retraction of the landing gear, especially under varying operational conditions. Adding a programmable parameter allows this block to be enabled or disabled, offering flexibility in simulating and analyzing the effects of actuator friction dynamically during the simulation runs.

$$F = \sqrt{2e} \cdot (F_{brk} - F_C) \cdot \exp\left(-\left(\frac{v}{v_{st}}\right)^2\right) \cdot \frac{v}{v_{st}} + F_C \cdot \tanh\left(\frac{v}{v_{Coul}}\right) + f_v \tag{3.11}$$

The total friction force F is a composite of:

- F_C represents Coulomb friction;
- F_{brk} represents breakaway friction;
- v_{brk} is the velocity at which breakaway; friction transitions to dynamic friction;
- v_{St} defines the Stribeck velocity threshold;
- v_{Coul} is the Coulomb velocity threshold;
- f denotes the viscous friction coefficient.

Hydraulic Supply Fault Block

Similarly, the Hydraulic Supply Fault block is designed to simulate restrictions in the hydraulic system, such as those caused by partial blockages or component wear. This block utilizes a Rotational friction approach to represent the hydraulic resistance and its impact on the system's efficiency [84]. By integrating this block into the hydraulic supply system of the LGS model, we can explore the consequences of diminished hydraulic pressure and flow on landing gear operation.

$$T = \sqrt{2e} \cdot (T_{brk} - T_C) \cdot \exp\left(-\left(\frac{w}{w_{st}}\right)^2\right) \cdot \frac{w}{w_{st}} + T_C \cdot \tanh\left(\frac{w}{w_{Coul}}\right) + f_w \tag{3.12}$$

The total friction torque T is a composite of:

- T_C represents Coulomb friction torque;
- T_{brk} is breakaway friction torque;
- w_{brk} is breakaway friction velocity;
- w_{St} is Stribeck velocity threshold;
- w_{Coul} is Coulomb velocity threshold;
- w is relative velocity;

- f represents the coefficient of viscous friction.

By implementing these custom fault injection blocks, our study pioneers a methodical approach to assessing the LGS's resilience and vulnerabilities under various simulated fault conditions. This development represents a significant step forward in utilizing Simscape for detailed fault analysis and underscores the potential of custom blocks in enriching simulations.

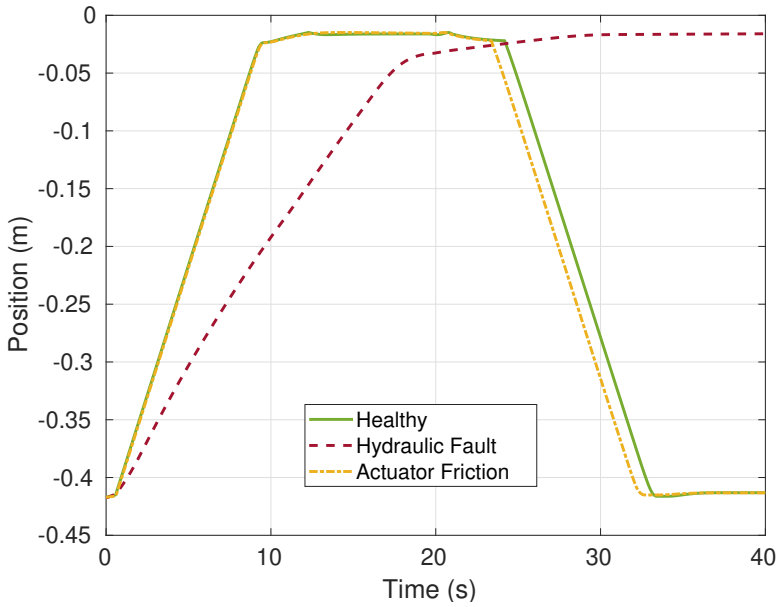


Fig. 3.13: Position of the Main Actuator.

Synthetic data generation

With the fault blocks in place, we move on to synthetic data generation. This step involves running multiple simulations at different severity levels for the injected faults. The resulting synthetic data serves two purposes: analyzing the system's behavior under various fault conditions and training robust ML algorithms for predictive maintenance.

The LGS model was subjected to simulations under three conditions:

- **Healthy**: This baseline scenario represented the system functioning normally, without any faults.
- **Hydraulic Supply Fault**: The corresponding fault block within the model was activated to simulate the hydraulic supply malfunction.
- **Actuator Friction Fault**: The dedicated fault block for actuator friction was enabled within the model.

For each fault scenario, simulations were run with varying parameters to explore a range of potential fault severities. In the absence of empirical data from real-world systems or literature, specific parameter values were selected to highlight the abnormal behavior of different failure modes and severities on the LGS:

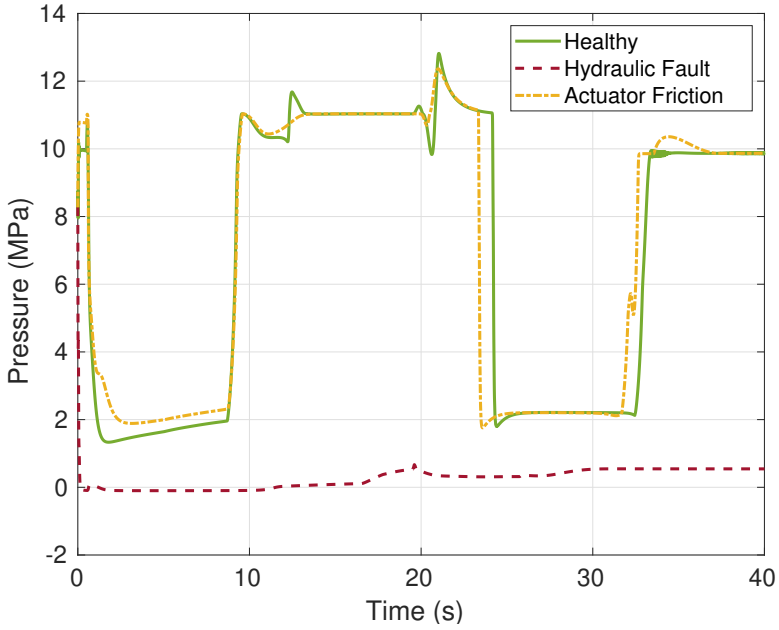


Fig. 3.14: Pressure of the Main Actuator.

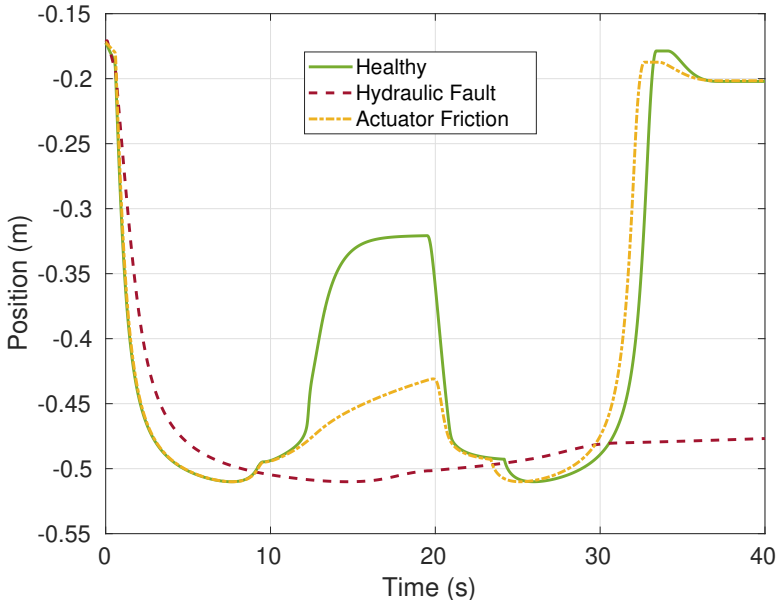


Fig. 3.15: Position of the Lock Actuator.

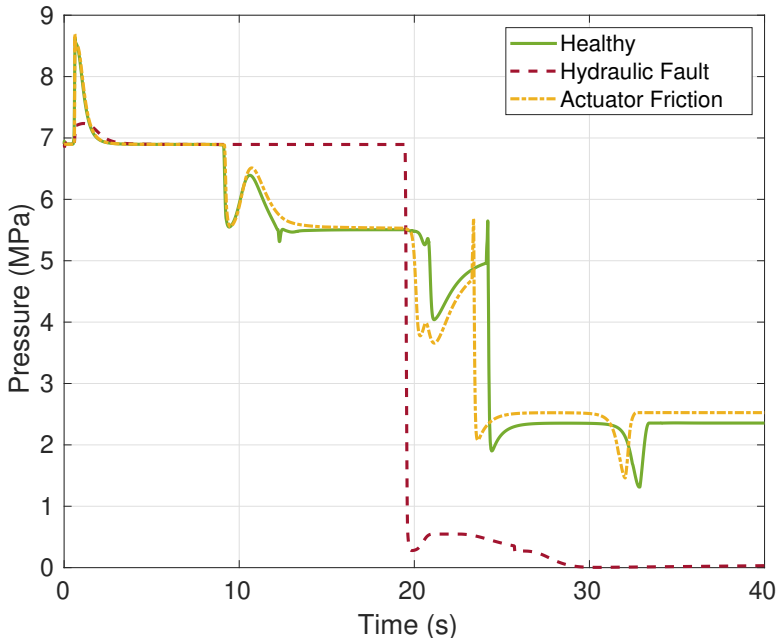


Fig. 3.16: Pressure of the Lock Actuator.

- For the **Actuator Friction Fault**, we applied the fault block across both actuators, using parameters that capture a broad range of friction-induced behaviors. These included a breakaway friction of 10 000 N, a breakaway friction velocity of 0.1 m/s, a coulomb friction force of 20 N, and a viscous friction coefficient of 100 N · s/m. This set of parameters was selected to demonstrate the fault's impact on actuator performance, specifically highlighting how friction can significantly affect the actuator's ability to retract and extend under various conditions.
- For the **Hydraulic Supply Fault**, the parameters we chose are a breakaway friction torque of 1.5 N · m, a breakaway friction velocity of 0.1 rad/s, a Coulomb friction torque of 1.5 N · m, and a viscous friction coefficient of 0.001 N · m · s/rad.

Table 3.4: Simulated failure blocks and fault cause.

Failure block	Failure cause
Actuator friction	1,4,5
Hydraulic supply fault	2,3,6,7,8

Figs. 3.13 and 3.14 illustrate the different effects of these faults on the main actuator's position and pressure, respectively, revealing critical insights into these specific fault types. For instance, actuator friction introduces a delay in retraction, which is attributed to pressure variations. Meanwhile, a hydraulic supply fault leads to a significant deployment delay and a failure to retract, underpinning a total pressure loss. Figs. 3.15 and 3.16 extend this analysis to the lock actuator, showcasing similar fault impacts and underscoring the comprehensive nature of these failure modes on LGS integrity. The selection of these two fault scenarios (hydraulic supply and

actuator friction) aimed to encompass a broad spectrum of potential LGS failure at a functional level. Although these fault blocks are distinct, they can be traced back to similar failure causes, albeit in different positions within the system. While further investigation into specific failure modes is warranted, these initial experiments provide valuable insights into the model's ability to capture critical system behaviors under fault conditions. Table 3.4 shows the relationship between the fault blocks and the possible cause.

3.7.5 Discussion

Our simulations targeted two critical fault conditions within the LGS hydraulic supply malfunction and actuator friction. These conditions represent a broad spectrum of potential LGS failures at a functional level. The analysis revealed distinct impacts on both main and lock actuators, aligning with expectations. Hydraulic supply faults caused significant deployment delays and prevented retraction, while actuator friction primarily introduced delays during retraction. These observations were supported by the pressure variations observed in the simulations. Moreover, the selection of representative cases prioritized faults with a high impact on system functionality and real-world relevance to LGS operations. These outcomes demonstrate the feasibility of the methodology for analyzing critical system behaviors under fault conditions and suggest that our simulation parameters effectively captured the dynamics of these fault conditions. By simulating various fault scenarios, designers can identify potential weaknesses and incorporate features that mitigate their impact. Additionally, the model can also be used to develop training programs for maintenance personnel and inform the development of robust Predictive Maintenance (PdM) algorithms that leverage synthetic data to continuously monitor the LGS for signs of faults and enable predictive maintenance.

It is becoming increasingly recognized among vendors that simulations incorporating fault scenarios can offer significant value, yet this is a nascent field requiring further research. Current simulation tools often lack the capabilities to easily incorporate and simulate faults, representing a significant barrier to widespread adoption and implementation. Addressing these gaps is crucial for advancing the field and fully realizing the potential of fault simulations.

In aviation, model-based design in aviation is a well-established practice, and access to high-fidelity models is crucial. However, researchers often face limitations in acquiring such models. Future work can delve deeper into specific failure modes and expand the fault library to encompass a more comprehensive representation of potential LGS issues. Real-world LGS health data can be used to further refine and validate the model in an iterative process, enhancing its accuracy and effectiveness. It is essential to acknowledge that the value of synthetic data generated by a model depends heavily on its quality. Effective LGS modeling requires a high level of expertise, which is often found within the aircraft domain, where high standards are necessary.

Table 3.5: Fault Injection Campaign Summary for the Landing Gear System Case Study

Domain	Fault Types	Injected Faults	Est. Sim. Time (CPU)
Hydraulic	Supply Fault (Blockage/Leakage)	11	~ 5 mins
Mechanical	Actuator Friction, Parameter Drift	142	~ 71 mins
Total		**153**	**~ 76 mins**

Note: Simulation times are estimated CPU elapsed times.

3.8 Multirotor Drone

Drones are captivating and increasingly ubiquitous systems that exemplify the core principles of cyber-physical systems, showcasing the intricate interplay between physical components (e.g., mechanical structures, energy systems) and digital elements (e.g., control software) [85, 86]. Their usage has significantly expanded in recent years, extending across various domains such as agriculture and surveillance [87–89].

Designing a drone is inherently *multidisciplinary*, with the primary challenges stemming not from isolated concerns like control software development or aerodynamic design, but from managing the inter-dependencies among these diverse domains. Modern drones require considerable computational power to process sensor data, execute machine learning algorithms, and maintain stable flight. This demand for processing capability competes with the need for energy-efficient systems to maximize flight time, all while adhering to strict weight constraints imposed by limited motors power and battery capacity. Moreover, drones depend on highly responsive and precise control systems, demanding careful integration of hardware and software. Achieving a balance between energy efficiency and precise control thus requires tight coordination and optimization across the entire system.

All these considerations significantly influence software development, particularly since drones operate in challenging environments where they may encounter unforeseen conditions, component variations, or failures. It is therefore essential to integrate fault scenario analysis into the design process and to build resilience into the system. This means the software must be fault-aware, capable of detecting, isolating, and responding to failures in real-time, while also supporting the evaluation of different hardware configurations, such as batteries, sensors, and other critical components. Since it is impossible to foresee every possible fault or performance variation, development efforts greatly benefit from a flexible and comprehensive simulation environment that covers multidisciplinary aspects. Such an environment allows designers to model various failure scenarios, compare alternative components, and develop tailored detection, mitigation, and optimization strategies.

The current state of the art features a plethora of drone simulators [86, 90], but most of them focus on specific design phases or domains. Some tools provide early development support but lack detailed physical modeling [91–93], while others emphasize accurate modeling to support flight algorithm design [94–96] or prioritize graphical simulation [43]. Integrating additional aspects, such as detailed power consumption, often requires coupling with external tools, which can be cumbersome and non-trivial.

The goal of this work (outlined in Fig. 3.17) is to propose an open-source solution that enables multi-physics drone simulation by integrating multiple facets of drone design into a single simulation run, with minimal integration effort. The proposed solution is built on top of three main pillars:

- *SystemC AMS*, a C++ library and modeling platform that supports the simulation and design of complex systems, with support for analog and mixed-signal modeling and multi-domain descriptions [39];
- *GVSoC*, an open-source C++ event-driven Instruction Set Simulator (ISS) for PULP RISC-V cores [41], supporting virtual prototyping and software performance estimation;
- *Unreal Engine*, a widely used game engine known for its advanced physics simulation, powerful rendering graphics, and extensive component support, combined with environmental support (e.g., gravity, wind) and user interaction (e.g., drone input commands) [43].

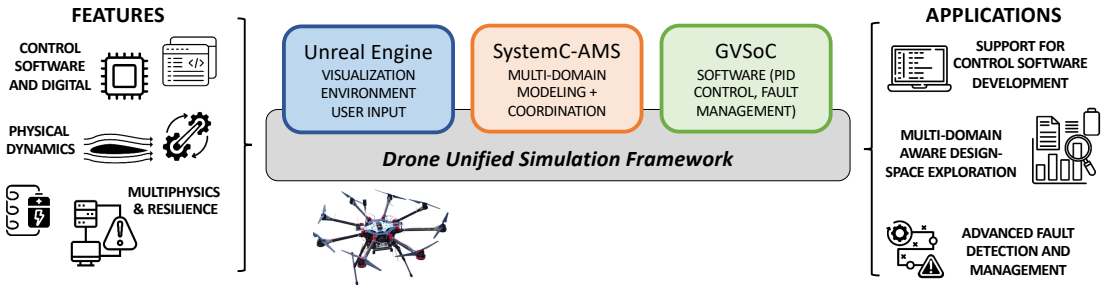


Fig. 3.17: Proposed co-simulation framework to enable modeling and analysis across mechanical, electrical, software, and environmental domains (left) with the integration of Unreal Engine, SystemC AMS, and GVSoC (middle), for effective drone design, exploration of alternatives, and development of software with fault management (right).

The key directions of the proposed simulation frameworks are thus a careful support for hardware, mechanical, power and multi-domain modeling (achieved with SystemC AMS), full support for software simulation and development (through the ISS), 3D modeling and environment support (thanks to Unreal Engine) and extensive support for fault detection, isolation, and recovery (enabled by the holistic view of the drone).

This work demonstrates the proposed approach using a state-of-the-art drone model described in [97]. This system-level model encompasses the aerodynamics of the rotor-propeller, electromechanical motor behavior, battery dynamics, and the rigid body dynamics of an eight-motor drone. We utilize this model to demonstrate how the implementation is realized in SystemC AMS and Unreal Engine, and to showcase specific design tasks enhanced by simulation, such as Design Space Exploration (DSE) of batteries and the development of advanced control and safety strategies.

Drone design

Drone design is inherently complex, requiring expertise across highly diverse domains. It often begins with the mechanical or airframe configuration, and subsequent domains, such as power

electronics, embedded control, sensor integration, and fault management, are added at later stages of the design flow [98, 99]. This impacts on the simulation and fault awareness support.

Simulation tools

Many drone simulation frameworks exist in the literature, each offering different trade-offs between fidelity, flexibility, and ease of use [86, 90]. Overall, most of the available tools are specialized for specific phases of the drone design process, either focusing on physical modeling, control development, or early-stage prototyping. Seamlessly combining functional and extra-functional properties (e.g., power, reliability) within a single, integrated simulation workflow remains a challenge, and often requires non-trivial effort to bridge the gap between multiple simulation environments.

Simulink, especially when used in combination with Simscape Multibody and Simscape Electrical, enables system-level multi-domain simulation by modeling both mechanical and electrical subsystems in a unified environment [100]. It is widely adopted for system-level modeling, control algorithm development, and Hardware-in-the-Loop (HiL) simulations. Its graphical modeling interface makes it accessible for rapid prototyping. However, Simulink has limitations when it comes to low-level hardware modeling, such as embedded processors or firmware execution. Additionally, implementing complex control strategies (e.g., deep learning-based control) often requires integration with external environments such as Python or C/C++ toolchains, which can complicate the workflow.

ANSYS, SolidWorks, and AirSim focus on high-fidelity physical simulation [96, 101, 102]. ANSYS and SolidWorks offer highly detailed finite-element and Computer Aided Design (CAD)-based simulations, particularly for structural, thermal, and fluid dynamics aspect, and are typically used during early mechanical design phases for evaluating aerodynamics and structural integrity. AirSim, developed by Microsoft, offers realistic physics and sensory environments for drones and autonomous vehicles, and supports integration with Unreal Engine or Unity for enhanced visuals [96]. However, these platforms generally lack out-of-the-box support for control system development or power modeling, requiring custom interfaces with other tools like Simulink or the Robot Operating System (ROS).

FlightGear, Gazebo, and OpenRocket are popular open-source simulators geared toward simulating real-world flight behavior [94, 95, 103]. Gazebo, for instance, is often used with ROS and supports physics-based simulation of robots and drones, including accurate modeling of joint dynamics, sensors, and controllers. While these tools are effective for simulating mechanical dynamics and Software-in-the-Loop (SiL) testing, they typically do not provide native support for detailed electrical systems or battery models, requiring co-simulation or external plugins.

Open-source projects such as ArduPilot and Webots are designed for algorithm development, prototyping, and educational use [91–93]. ArduPilot supports a wide range of vehicles and provides robust firmware-level control with strong community support. Webots offers a lightweight and accessible simulation environment with basic physics and sensor modeling. However, these tools often lack support for advanced physical modeling, including nonlinear

aerodynamics, detailed actuator behavior, or accurate energy consumption, making them less suitable for high-fidelity, system-level simulation.

Finally, game engine-integrated simulators, like Unity ML-Agents and Unreal Engine [43, 104], provide realistic visualization and sensory fidelity, but they do not consider the electrical or mechanical behavior of the components.

Drone fault detection, isolation, and recovery

Faults are a critical aspect of aerospace and robotics fault-tolerant systems. A fault in a drone is an unexpected deviation from normal operation in any hardware, software, or mechanical component that degrades or compromises system performance, reliability, or safety. Faults do not necessarily lead to failure immediately, but can escalate into one if not detected and mitigated.

Despite of the cause, faults can be classified as (Figure 3.18):

- Permanent faults: irreversible changes, due to deterioration of a component or a hazard during use;
- Transient faults: triggered by non-permanent conditions, such as high outside temperature, power surges, or electromagnetic interference [105], and thus rarely leave the system permanently broken;
- Intermittent faults: faults that occur frequently, but the system is not affected continuously by such faults. The healthy behavior is restored after a short time interval [106];
- Incipient faults: cause a gradual degradation of system performance, e.g., due to system wear and tear.

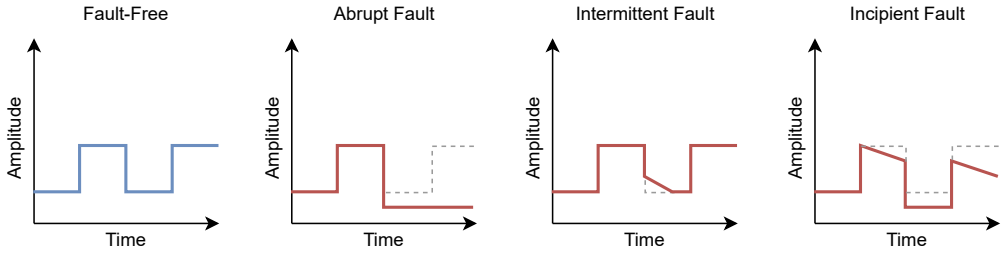


Fig. 3.18: Differences between fault-free behavior, abrupt fault, intermittent fault, and incipient faults.

Fault mitigation strategies include different actions. To detect faults, it is necessary to monitor system parameters (e.g., sensor values, motor Revolutions Per Minutes (RPM), battery levels) for anomalies, by using threshold-based checks or model-based estimations (e.g., Kalman filters). Once a fault occurs, it is then necessary to isolate it, determine the subsystem or component where it occurred, and identify possible causes. Recovery actions can then be defined, such as reconfiguring the system (e.g., switching to backup sensors), enabling graceful degradation (e.g., limiting maneuverability), or initiating safe procedures, like returning to base or performing an emergency landing. The time from fault detection to full fault mitigation is referred to as the reaction time, as it represents the time between the failure and the completion of the countermeasure action.

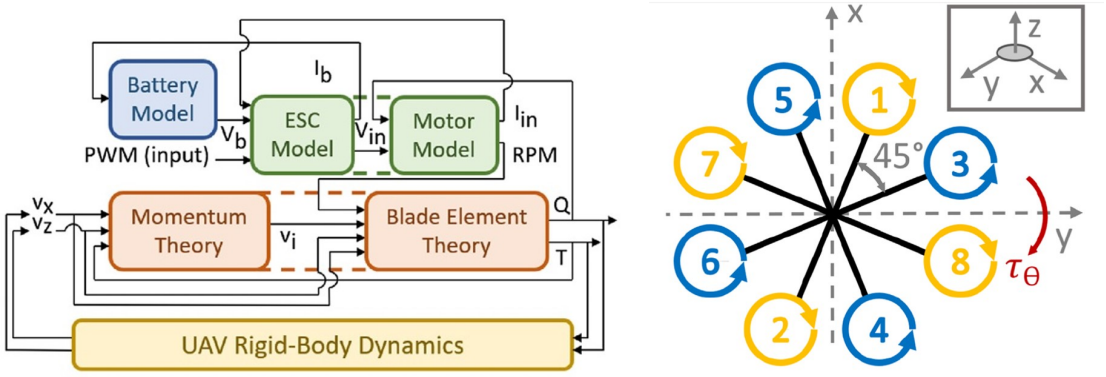


Fig. 3.19: Block diagram of drone model (left) and layout of rotors, axes, and torque (right) as reported in [97]. The drone model provides mechanical and power subsystems.

In the context of drones, faults can span across multiple domains: sensor faults (e.g., hard or intermittent failure); actuator faults (e.g., motor degradation causing a partial loss of thrust due to wear or overheating); power system faults (e.g., sudden battery voltage drop or capacity loss); software faults (e.g., due to latencies or to control instability under edge cases); and even environmental faults (e.g., wind gusts) [107]. In such complex scenarios, drone designers often develop fault management capabilities late in the process, as a reaction to faults observed during integration, rather than incorporating them into the early design models [98, 108].

On the other hand, fault management is a consolidated strategy in other contexts, such as embedded systems. Embedded systems are indeed widely used in domains such as automotive, aerospace, and industrial automation, which require very high reliability, determinism, and safety levels [1, 109, 110]. To ensure this, embedded system design anticipates faults during early design phases. In order to understand and improve the system, developers perform fault modeling and simulation, i.e., representing faulty behaviors or failure modes in a simulation of the overall system to analyze their impact on system performance, safety, and reliability [111]. This allows to identify weaknesses and to improve the design with error detection and recovery strategies.

3.8.1 Starting multi-rotor drone system-level model

To exemplify the proposed solution on a real drone, we adopted the drone model in [97] as a starting point for our work. This model captures the tightly coupled interdependencies between electrical, mechanical, thermal, and aerodynamic subsystems in a drone, allowing for the simulation of faults, the evaluation of their impact on system performance, and the validation of control strategies. The remainder of this paper will show its implementation in the proposed frameworks.

The structure of the drone propulsion and dynamics model in [97] is illustrated in Figure 3.19. The control inputs to the system are Pulse Width Modulation (PWM) signals, issued by the UAV flight controller to each rotor. These PWM commands instruct the **Electronic Speed Controller (ESC)** to regulate the motor input voltage as a fraction of the battery voltage V_B,

based on the PWM duty cycle. The output of the ESC model is the modulated voltage V_{in}, which serves as the input to the motor model.

The **Motor Model** simulates the behavior of a brushless DC motor driven by V_{in}. It computes the resulting motor current I_{IN} and angular speed (RPM), both of which depend on electrical characteristics and the mechanical load imposed by the connected propeller. The torque Q exerted by the propeller feeds back into the motor, influencing its rotational speed and current draw.

The motor's rotation drives the propeller, which is modeled using **Blade Element Theory (BET)**. BET breaks the propeller into small blade segments and integrates their contributions to compute the generated thrust T and torque Q. These forces are functions of rotational speed, local flow velocities, air density, and propeller geometry.

To account for airflow changes induced by the propeller's operation, the model incorporates a **Momentum Theory** module. It estimates the induced inflow velocity v_i at the rotor disk based on the UAV's translational velocity components (v_x, v_z). This velocity affects local flow conditions and is fed back into the BET model, closing the aerodynamic loop.

The computed thrust and torque are then used in the **UAV Rigid-Body Dynamics** module, which integrates them into Newton-Euler equations to update the UAV's position, velocity, and orientation. The resulting motion alters the inflow velocities at the rotor, further influencing the aerodynamic and motor models.

The motor current I_{IN} is also passed to the ESC, which uses it to calculate the battery current draw I_B. Finally, the **Battery Model**, based on an equivalent circuit with open-circuit voltage V_{OCV} and internal resistance R_S, responds to this current by updating its voltage output V_B. This voltage is supplied back to the ESC, thus completing the electrical feedback loop.

Overall, the model integrates ESC regulation, motor dynamics, aerodynamic loading, vehicle motion, and battery behavior into a closed-loop simulation. Each subsystem is interconnected through continuous feedback, enabling accurate analysis of UAV performance, transients, and cross-domain interactions. Software is not included in the simulation, which directly takes in input PWM signals, rather than user commands. Additionally, no environmental support is provided, either as input sensor data or as physical awareness (e.g., wind). This limits the development of control strategies on top of the drone model.

3.8.2 Methodology

This section outlines the methodology used to develop our comprehensive drone simulation framework. Simulating a drone, such as the one described in Section 3.8.1, is inherently multidisciplinary and requires consideration of user interaction, software development, and execution. All such aspects can not be easily modeled in a single framework, and are rather partitioned into three communicating subsystems:

- **Unreal Engine** is used to collect user inputs via keyboard commands, and as a visual front-end of our simulation, representing the drone and its environment interactively;

- **GVSoC** is used to transform user input into commands for the drone motors, by applying PWM and thrust calculation; it also performs PID control (i.e., stabilizes the drone's movement, control trajectory, and speed), and applies fault mitigation strategies;
- **SystemC AMS** is used to simulate the drone's multidisciplinary aspects described in Section 3.8.1, to reproduce aerodynamics, mechanical, power, and hardware aspects;

Communication between these three subsystems is handled separately between SystemC AMS and Unreal Engine (via socket) and between SystemC AMS and GVSoC (via API invocations). The overall system architecture is reported in Fig. 3.20. The next sections detail each subsystem.

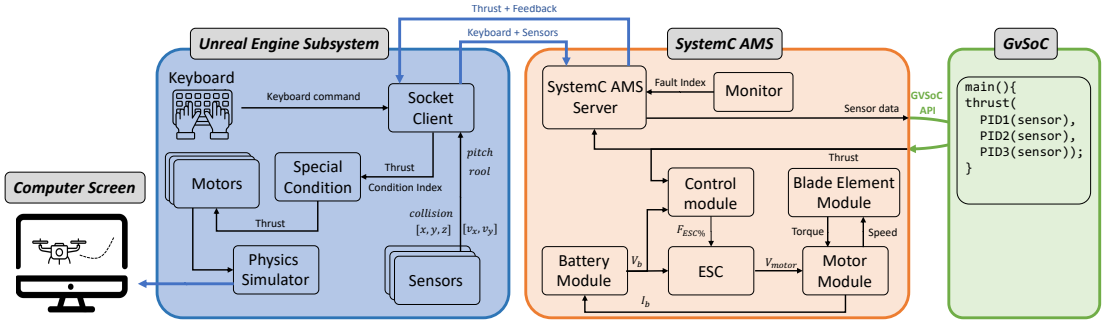


Fig. 3.20: Detailed simulation infrastructure: the drone model in Figure 3.19 is simulated in SystemC AMS; control software (including stabilization control and thrust estimation) is executed in GVSoC; user input, visualization, and environment simulation are managed by Unreal Engine. Communication with Unreal Engine happens via sockets, while the synchronization between SystemC AMS and GVSoC are managed through the invocation of GVSoC APIs.

SystemC AMS subsystem

The SystemC AMS subsystem is derived from the model in [97] and reproduces most of drone domains, including power, aerodynamics, mechanics, and hardware. This is possible thanks to the flexibility of SystemC AMS, which natively provides constructs that span across multiple notions of time and modeling semantics [2].

The mechanical and aerodynamics subsystem includes:

a) *Control module*: this module is responsible for accurately controlling engine thrust by translating the thrust request from the software (received from GVSoC, as will be explained later) into the appropriate Electronic Speed Controller (ESC) command $F_{ESC\%}$. This is achieved by performing an inverse computation that maps the required thrust to the corresponding ESC control percentage, i.e., the percentage of available battery voltage necessary to reach the required motor angular velocity. This is a crucial condition to ensure that the request can be satisfied and that the drone can move as desired.
The process requires to computing the necessary rotor angular velocity ω to generate the desired thrust T_{req}, as in Eq. (3.13), and then to convert ω into the ESC control percentage $F_{ESC\%}$, as in Eq. (3.14):

$$\omega = \sqrt{\frac{2T_{req}}{C_1 \rho A}} \tag{3.13}$$

$$F_{ESC\%} = \frac{\omega + b\omega^2 K_v^2 R_m}{V_b K_v} \tag{3.14}$$

where, C_1 is the thrust coefficient, ρ is the air density, A is the rotor disk area, K_v is the motor velocity constant, R_m is the motor resistance, b is a loss coefficient, and V_b is the battery voltage. This control strategy ensures that the physical thrust produced by the motors aligns closely with the flight control requirements.

b) *Blade element module*: this module estimates the torque Q and the thrust T generated by the propeller, based on the blade element theory, the current drone speed, and the required PWM. The original model in [97] is very accurate from a physics point of view, but it contains cyclic dependencies that create extremely long simulation times and simulation instability, as convergence is heavily dependent on the preset initial values. This is typical of dynamic systems that are not completely supported by the solver used by SystemC AMS (which relies on the Euler and trapezoid methods) [2].
To overcome this limitation, we adopted a simplified model, providing a linear relation between rotor speed and induced torque Q and thrust T of the blade elements [112], as in Eq. (3.15), Eq. (3.16) and Eq. (3.17).

$$L_b = -\frac{1}{2} C_1 \rho A W^2 \tag{3.15}$$

$$f_T = k\omega^2 \tag{3.16}$$

$$T_F \approx T_D = bw^2 \tag{3.17}$$

where L_b is the blade airfoil lift in air of density ρ, W is the airspeed of a blade, A is the blade area, $C1$ is the lift coefficient, ω is the propeller speed. L_b is set equal to the thrust f_T, and the torque acting on the frame T_F is equal to the drag torque of the rotor T_D, that is given in output.

c) *Electronic Speed Controller (ESC)*: the ESC module takes as input the control signal $F_{ESC\%}$, received from the control module, and converts it into energy that makes the motors spin at the desired speed. This is calculated by determining V_{motor} as the percentage of the battery voltage V_B necessary to realize the requested movement:

$$V_{motor} = V_b \cdot \ F_{ESC\%} \tag{3.18}$$

d) *Motor module*: This module transforms electrical power into mechanical power to produce the thrust necessary for flight. It receives the input voltage V_{motor} from the ESC and models the internal electromechanical behavior of the motor using a set of equations. The current demand I_{motor} is calculated from the torque Q, multiplied by a motor constant K_v (equal to the inverse of the motor velocity constant), plus a current demand due to motor inner resistance I_0 caused, e.g., by friction, as in Eq. (3.19) [113]. The current demand will be

given as output to the battery model and used to calculate the internal voltage from V_{motor} by applying the internal voltage drop across the motor's phase-to-phase resistance, R_m. The motor angular velocity ω is then derived by multiplying such voltage by the motor constant K_v (Eq. (3.20)).

$$I_m = Q \cdot K_v + I_0 \tag{3.19}$$

$$\omega = (V_{motor} - I_m R_m) \cdot K_v \tag{3.20}$$

As evident from Eq. (3.13) to (Eq. (3.20)), the mechanical and aerodynamics subsystem requires discrete-time computation. Each module can thus be realized as a SystemC AMS TDF module, thus exploiting the effectiveness of the static scheduling. An example for the control module is sketched in Fig. 3.21. Each module defines an interface, as input and output ports (lines 3-5), and implements predefined functions. In detail, the `set_attributes()` function sets the execution time step (lines 15-17). The `initialize()` function allows setting initial values of variables and ports (lines 18-24). Finally, the `processing()` functions define the discrete-time computation that must be updated at fixed time steps by implementing Eq. (3.13) and Eq. (3.14) (lines 25-28).

```
1. SCA_TDF_MODULE(CONTROL) {
2. public:
3.     sca_tdf::sca_in<double> thrust;
4.     sca_tdf::sca_in<double> Vb;
5.     sca_tdf::sca_out<double> Fesc;
6.
7.     SC_HAS_PROCESS(CONTROL);
8.     CONTROL(sc_module_name CONTROL);
9.
10.    double w, Kv, A, Rm, C1, b;
11.    void set_attributes();
12.    void processing();
13.    void initialize();
14.};
```

```
15.void CONTROL::set_attributes() {
16.    set_timestep(1, SC_MS);
17.}
18.void CONTROL::initialize() {
19.    w = 0;
20.    Kv = 406.5;
21.    Rm = 0.10;
22.    A = 0.002363;
23.    C1 = 0.03546;
24.}
25.void CONTROL::processing() {
26.    w = sqrt(2 * thrust.read() / (C1 * A * 8));
27.    Fesc = (w + b*w*w + Kv*Kv * Rm)/(Vb.read() * Kv);
28.}
```

Fig. 3.21: Example of SystemC TDF module implementing the control module.

The SystemC AMS subsystem also includes a power component, bridging the gap between power usage to activate the motors and power storage:

e) *Battery module*: This module is responsible for calculating the State of Charge (SoC) and the output voltage V_b of the battery. The internal dynamics of the battery are modeled using a first-order equivalent circuit, where V_b depends on the total current demand from the motors I_b (positive I_B indicates discharge), the available charge C_b, and the internal resistance R_{int}. The SOC represents the remaining charge in the battery relative to its nominal capacity C_{bat} and is estimated by integrating the current over time according to Equation Eq. (3.21):

$$\frac{dSOC}{dt} = -\frac{1}{C_{bat}} I_B \tag{3.21}$$

The terminal voltage of the battery is computed using the first-order equivalent circuit model shown in Eq. (3.22), where $V_{oc}(SOC)$ is the open-circuit voltage and is typically a nonlinear

function of SOC:

$$V_B = V_{OC}(SOC) - R_{int} \cdot I_b \tag{3.22}$$

This model enables the efficient and realistic simulation of battery behavior under varying load conditions by capturing both energy depletion and voltage drop effects resulting from internal resistance. Additionally, it allows for the variation of the battery's internal parameters (e.g., R_{int}) as a function of SOC, enabling the accurate modeling of battery internal dynamics. The variability of R_{int} does not allow for reproducing the battery with the ELN semantics of SystemC AMS, as the resistor primitive `sca_eln::sca_r` has a fixed Ohmic resistance value. It was thus necessary to solve the circuit equations, and to have coefficients whose values are adjusted to the battery state ($V_{oc} \in [2.5, 4.2]$, $R_{in} \in [0.0185, 0.0239]$).

GVSoC subsystem

Drone control requires the development of control firmware to perform drone stabilization, sensor data fusion and processing, and interpretation of user input commands. The development of such software cannot be done in isolation, and it rather benefits from development in an ISS, which allows for the execution of a specific software stack and enables the exploration and validation of performance, stability, and response to various commands.

In this work, we focus our attention mostly on two software components: the Flight Control System, for drone stabilization, and a Fault Management System, for detecting and managing faults, including battery faults, motor faults, etc. [114]. Such software components have been implemented in C and integrated in the GVSoC ISS, and play a central role in managing the drone's behavior during normal operation and fault conditions:

a) *Flight Control System*: this module is designed to ensure stable flight and precise maneuvering. It is structured around three independent Proportional-Integral-Derivative (PID) controllers, i.e., algorithms that constantly process data from sensors to understand how the drone is moving relative to the desired trajectory, and then send commands to the ESCs and motors to correct any errors. PID controllers are thus the responsible entities for regulating a specific subset of the drone's dynamics.
The drone's flight control system is organized around three independent PID controllers, each tasked with regulating a distinct aspect of the drone's dynamics. Each controller follows the standard PID formulation:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{3.23}$$

where $u(t)$ denotes the control signal at time t and $e(t)$ represents the control error. The proportional gain K_p produces a correction proportional to the instantaneous error, enabling fast response. The integral gain K_i accounts for the accumulation of past errors, ensuring the elimination of steady-state offsets. The derivative gain K_d anticipates the future trend of the error, contributing to a smoother response by reducing overshoot and improving overall stability.

The first controller (PID1) manages the *drone's altitude*, and thus requires as input the target altitude and the current altitude. For PID1, error is defined as the difference between the desired target altitude $h_{\text{target}}(t)$ and the current measured altitude throttle(t):

$$e_{PID1}(t) = \text{throttle}(t) - h_{\text{target}}(t) \tag{3.24}$$

The second controller (PID2) is responsible for regulating roll (i.e., *rotation around the longitudinal X-axis*), which primarily influences the drone's lateral linear velocity (along the Y-axis). PID2 requires x-axis rotation angle $\theta_x(t)$, and linear velocity along the y-axis $v_y(t)$ a velocity feedback gain that modifies the error based on linear velocity, aiming to improve stabilization and counter drift caused by the drone's rotational movement. The error signal incorporates both attitude and velocity components:

$$e_{PID2}(t) = \theta_x(t) - \theta_{x,\text{target}}(t) + C_n + v_y(t) \cdot C_v \tag{3.25}$$

where the term C_n is a damping factor applied directly to the error, designed to help suppress small residual rotational motions quickly, and C_v is a velocity feedback gain that modifies the error based on linear velocity, aiming to improve stabilization and counter-drift caused by the drone's rotational movement.
The third controller (PID3) performs a similar function for pitch (i.e., *rotation around the lateral Y-axis*) given y-axis rotation angle $\theta_y(t)$ and the linear velocity along the x-axis $v_x(t)$, thus primarily influencing the drone's forward/backward linear velocity (along the X-axis). The error signal is thus:

$$e_{PID3}(t) = \theta_y(t) - \theta_{y,\text{target}}(t) + C_n + v_x(t) \cdot C_v \tag{3.26}$$

All PID control algorithms thus require sensor data that can be generated by the Unreal Engine simulation and forwarded to the control software. The algorithms are implemented in C for easy integration and execution in the GVSoC ISS.

b) *Fault Management System*: this system is responsible for detecting, isolating, and mitigating malfunctions or anomalies across all critical components, including motors, ESCs, propellers, sensors, and batteries. Its main objective is to ensure operational safety by identifying faults early, isolating their sources, and executing appropriate response actions to transition the drone into a safe state, minimize potential damage, or, when feasible, maintain mission continuity. Including this component within the framework enhances the realism of the simulation, enabling it to closely replicate real-world system behavior and ensuring that the overall simulation remains consistent with real-world conditions. For example, if an engine fault index indicates a malfunction, the countermeasure system switches to a predefined emergency flight mode, and if sensor values remain static for an extended period (thus suggesting sensor failure), the system can infer the fault and respond accordingly, either by issuing warnings, disabling affected controls, or activating fallback strategies.
The development of such a system is extremely complex, as it requires monitoring and controlling heterogeneous domains, including batteries, sensor data, and mechanics-related

sensors. It is thus the software component that benefits the most from the development of a multi-domain simulation framework like the one proposed in this work, e.g., to simulate fault scenarios safely and repeatedly, with no physical harm, test a vast number of fault combinations, edge cases, and environmental conditions that would be impractical in the real world, and reduce the reliance on expensive physical prototypes and time-consuming flight tests [115]. Section 3.8.3 will focus on the development of some examples of fault management strategies for the drone under design.

Together, the flight control and the fault management system form a robust architecture that combines real-time feedback, high-level user intent, and fault detection to enable autonomous and resilient drone operation within a simulated environment.

Unreal Engine subsystem

As presented by Michel et al. in [86], a drone simulator includes many different aspects that go beyond the mathematical multi-physics equations of the system, including a control system, to determine the flight route, detect (and react to) collisions, and interact with the user in keyboard or pad-controlled simulators, and a visualization mode, to allow comprehension of the drone behavior over time. While SystemC AMS is well-suited for multi-physics modeling and GVSoC can support the software of the control system, they lack support for advanced visualization. Additionally, integrating location sensors (to enable visualization) would require explicitly calculating the position coordinates of the drone in the reference system, as well as rotation angles, etc., which is far from trivial. The role of the Unreal Engine subsystem is thus mostly twofold:

a) *Visualizer*: Originally developed as a game engine, Unreal Engine offers high-fidelity rendering and advanced environmental modeling capabilities. In our simulator, it is used to visually represent the drone and its surrounding environment in a realistic and structured manner. The drone model is configurable, with parameters such as size, weight, and appearance that can be adapted to reflect different real-world drone configurations. The simulated environment consists of static objects, such as buildings, walls, or trees, arranged to recreate realistic deployment scenarios and to evaluate the drone's behavior in the presence of physical constraints or obstacles.
b) *Sensors Modeling*: Unreal Engine is also responsible for emulating the sensors needed to provide data to the rest of the simulation system, particularly to the control software. Among these are a position sensor, which continuously reports the drone's $[x, y, z]$ coordinates in the 3D world; an orientation sensor, that captures roll and pitch angles ($\theta_x(t)$ and $\theta_y(t)$); and velocity sensors that measure the drone's translational speeds along the x and y axes (i.e., $v_x(t)$ and $v_y(t)$). Additionally, specialized sensors are implemented to detect collisions or exceptional conditions, such as impacts with obstacles or entering restricted areas. These event-driven sensors trigger corresponding reactions in the control logic, depending on how the controller is configured to handle such scenarios.
c) *Keyboard Interaction*: Another key component is the keyboard interface, which enables the user to directly interact with the drone to either control its desired movement or give inputs

to affect its movement, e.g., to simulate faults (as will be described later in the experimental sections).

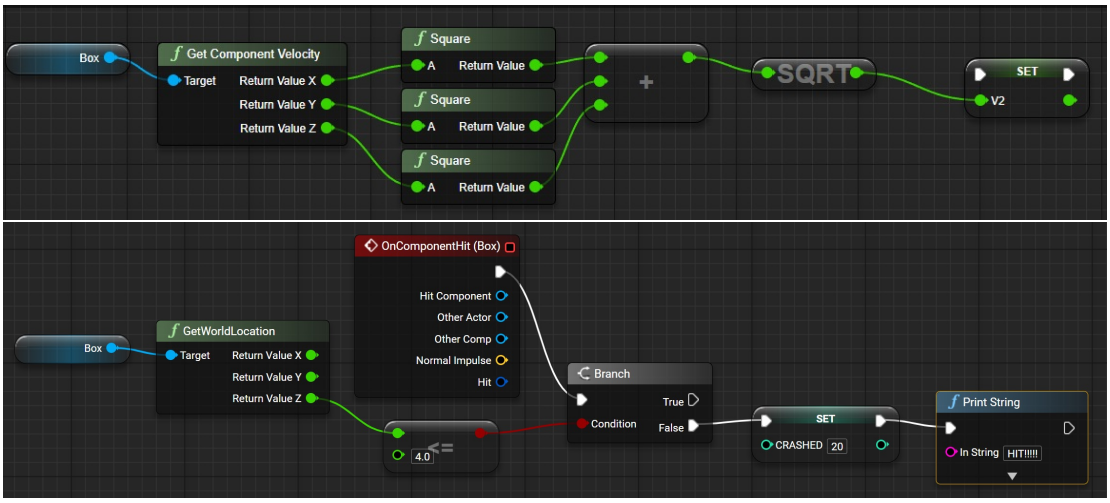


Fig. 3.22: Blueprints for estimating (top) drone velocity and (bottom) drone position to check for crashes.

All the described modules are implemented as Blueprints, i.e., graphically specified scripts in Unreal Engine that encapsulate logic through a node-based interface. Blueprints provide a visual programming model that is highly suited for rapid development and prototyping of interactive systems. In our simulator, they are used to define the behavior of sensors, environment elements, and drone dynamics in response to simulated physics and control commands. For example, in Fig. 3.22, the velocity sensor blueprint (top) continuously tracks the drone's velocity to send it to GVSoC, and the collision sensor blueprint (bottom) monitors the drone's position and physical interactions with the environment to trigger event-based responses.

Thanks to Blueprints, complex event-driven logic can be implemented and modified without altering C++ source code, thus improving maintainability and enabling fast iteration during development. Additionally, the Blueprint system is fully integrated with the Unreal Engine rendering and physics engines, ensuring tight coupling between visual feedback, physical simulation, and control logic from external tools (i.e., GVSoC).

Tool integration

The integration and synchronization of the tools are crucial aspects of the proposed simulation infrastructure, as sketched in Fig. 3.23. SystemC AMS has a lead role, as its *SystemC AMS server* block (Fig. 3.20) handles communication with both Unreal Engine and GVSoC.

Communication with Unreal Engine occurs through TCP/IP sockets, which enable simple message passing and synchronization at specified time steps. Synchronization with GVSoC instead relies on MESSY, an open-source solution that integrates GVSoC with SystemC AMS [116] for Design Space Exploration (DSE) of cyber-physical systems. The integration is realized by using SystemC AMS as master, as it periodically invokes the `step_until()` API

function of GVSoC to allow software execution for a given time step. This allows immediate time and data alignment of all frameworks, by defining a synchronization time step common to all simulators.

A visual representation of the overall synchronization and integration is depicted in Fig. 3.23, which adopts a timing diagram formalism. At initialization time (*INITIALIZE* fragment), SystemC AMS prepares its simulation environment by running all constructors and building the event queue. Additionally, the constructor of the SystemC AMS server block also instantiates the GVSoC ISS and its own event queue by starting SW execution, and it sets up a TCP/IP socket for communication with Unreal Engine. Then, Unreal Engine starts and connects to the TPC/IP socket, thus ending the initialization phase.

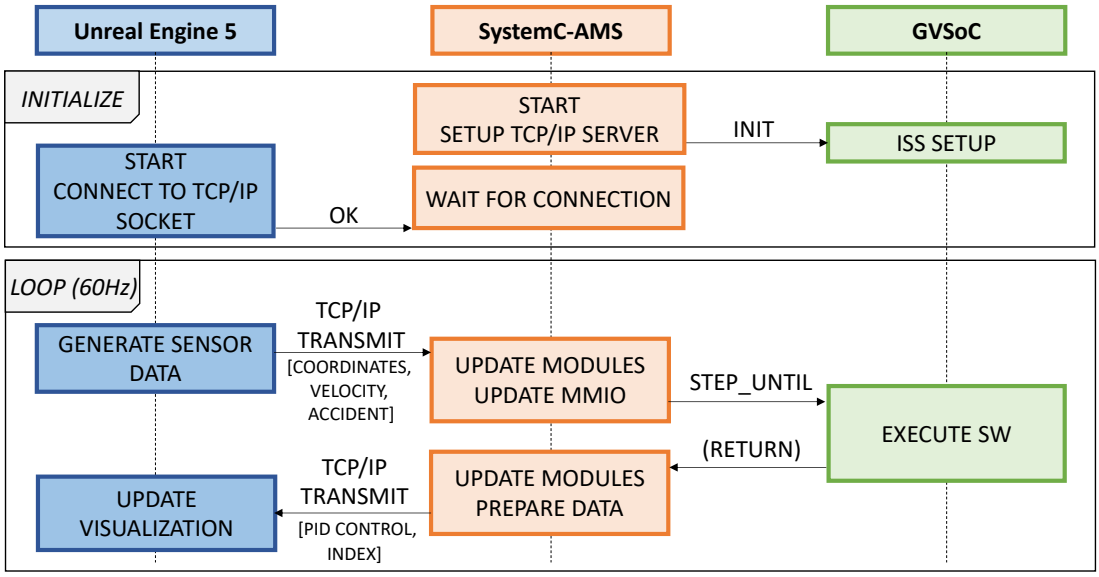


Fig. 3.23: Execution flow and synchronization between SystemC AMS, GVSoC, and Unreal Engine.

To run the simulation (*LOOP* fragment), it is necessary to define a synchronization mechanism that maintains a common notion of time across the three simulators, allowing for deterministic and coherent processing of events and communication. Unreal Engine allows the use of a variable timestep for its main game loop, but provides a fixed timestep feature to enable more stable and predictable physics, especially for complex simulations, by dividing a single frame time into smaller, fixed-duration sub-steps. Given the need for tight synchronization with the other simulators, we thus configured Unreal Engine with a 60Hz execution rate. Synchronization with SystemC AMS is based on such execution frequency.

The former choice (*i.e.*, system update at 60Hz) drives the main simulation loop. Every 16.67ms (i.e., at 60Hz), Unreal Engine recomputes sensor data (e.g., drone coordinates, position) and communicates it via a socket to SystemC AMS. The SystemC server block listens to the TCP/IP socket and is activated every 16.67 ms, upon receiving new data through the socket. In this way, the updated Unreal Engine information is aligned with the TDF execution cycle, allowing it to recompute the main core of the drone simulation synchronously with Unreal En-

gine. SystemC AMS then updates the Memory Mapped I/O (MMIO) memory region associated with GVSoC to allow data sharing. Activation of GVSoC occurs through the invocation of the `step_until()` API function passing the time step as a parameter (*i.e.*, 16.67ms). This allows re-executing SW (*e.g.*, PID control) with the same frequency as the data update, and to react accordingly to the evolving drone conditions and user requests. The resulting control outputs are written to the MMIO to SystemC AMS and then sent back to Unreal Engine via TCP/IP socket communication, allowing for the visualization update.

It is important to note that this design choice allows a tight coupling of the three simulators, which execute at the same simulation pace and deterministically. Other configurations are possible, *e.g.*, activating SystemC AMS less frequently to reduce the simulation overhead or only in response to new input values from Unreal Engine.

Framework performance

This section analyzes the performance of the proposed framework in terms of CPU utilization, memory consumption, and runtime. To collect these metrics, we conducted a 140-second drone flight simulation, which provided a sufficiently long execution window to capture the framework's impact on system resources. All experiments were executed on a Windows 11 laptop equipped with 16 GB of RAM, an Intel Core i9-9750H processor (6 cores), and an NVIDIA GeForce RTX 2060 GPU (used to accelerate the Unreal Engine rendering).

- **CPU usage**: The results are presented in the left plot of Fig. 3.24. We were able to isolate the CPU usage of SystemC AMS + GVSoC from that of Unreal Engine 5. The measurements show a stable CPU load throughout the simulation, with SystemC AMS averaging 6.54% per core and Unreal Engine averaging 5.02% per core.
- **RAM usage**: The same distinction between actors is applied for this metric as well. As expected, the results (reported on the right in Fig. 3.24) show that Unreal Engine dominates memory consumption, with an average of 5,595.73 MB over the length of the simulation. In contrast, the memory consumption of SystemC AMS can be considered negligible (avg. 82.72 MB). The reason behind these numbers is the memory intensity associated with 3D rendering, which involves handling high-resolution textures, complex geometries, and dynamic lighting effects throughout the simulation.
- **Runtime**: This metric indicates how much real-world time is needed to simulate a given amount of simulated time, allowing us to evaluate the overhead of the framework. In this experiment, as well as in all those presented in Section 3.8.3, the simulated time advances at a 1:1 ratio with real time. This shows that the framework is real-time: simulating 140 seconds requires exactly 140 seconds in the real world, while allowing full visibility onto all drone behaviors.

3.8.3 Framework application to enhance drone design

This section details a series of experiments conducted within our simulation framework, showcasing its capabilities in both optimizing initial design parameters and ensuring system re-

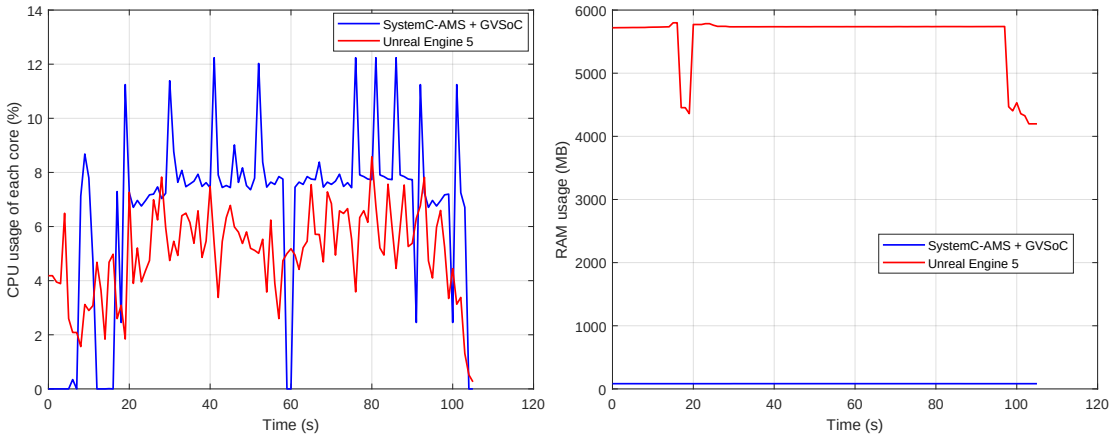


Fig. 3.24: CPU and memory usage of a 140s simulation

silience against common real-world challenges and failures. Collectively, these experiments underscore the power and flexibility of our simulation framework in enabling systematic DSE and fault scenario testing. This comprehensive approach is essential for developing optimized, reliable, and inherently safer autonomous aerial platforms.

Design Space Exploration

This section focuses on fundamental component selection, demonstrating how different battery configurations critically impact drone autonomy and environmental interaction. An experiment is conducted to investigate the drone's stability when subjected to external environmental disturbances, highlighting the importance of designing adaptive control algorithms to maintain stable flight. The goal is to demonstrate that a single simulation framework enables the consideration of multiple aspects during prototyping, allowing for careful design choices, including environmental effects that are not easily incorporated in such a preliminary phase.

Battery Choice

Battery choice is crucial to allow drone autonomy and effectiveness, and requires taking into account several parameters, including battery physical dimensions, which affect how it fits within the drone's frame; its capacity, which directly influences flight time; and its weight, which has a significant impact on the drone's lift requirements and energy efficiency. A well-balanced trade-off among these factors is essential to optimize both performance and endurance without compromising safety or maneuverability.

In this experiment, we compared three batteries with similar capacities (15Ah to 16Ah) and voltages (21.6 V to 22.2V), thus ensuring a fair baseline in terms of total energy available, but with different weights, ranging from 1.350 kg to 1.984 kg (approximately +46%). Full battery details are provided in Table 3.6.

The three batteries offer different tradeoffs in terms of electrical characteristics and weight. The Sony battery pack [117] is a good compromise in terms of energy density and peak current, and may thus seem the best design choice. The Samsung battery pack [118] offers the best

energy density but a lower maximum current, which may be negative in high-thrust or peak-load scenarios. Finally, the Tattuu battery pack [119] has a lower energy density, which could translate to reduced flight efficiency and shorter airtime, but with a maximum peak current and the highest pack voltage. Estimating the impact of such design choices with no simulation is far from trivial, as battery dynamics are nonlinear and difficult to predict [120].

Table 3.6: Parameters of the compared battery packs.

	Sony Li-Ion 18650 VTC6 [117]	Samsung INR21700-50S [118]	Tattu Lipo [119]
Cell Capacity (mAh)	3,000	5,000	16,000
Pack Capacity (Ah)	15	15	16
Pack Weight (g)	1,600	1,350	1,974
Pack Dimensions (mm×mm×mm)	200×375×285	70×125×75	65×190×76
Pack Voltage (V)	21.6	21.6	22.2
Pack Max Current (A)	150	45	480
Energy Density (mAh/g)	9.375	11.100	8.105
SoC at end of simulation (%)	50	52.3	51.3

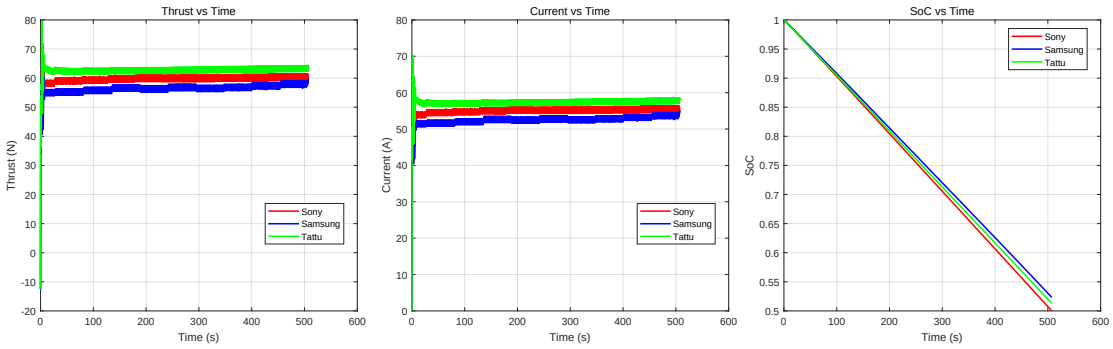


Fig. 3.25: Battery exploration: generated thrust (left), current (middle), and SoC evolution (right) with the batteries in Table 3.6 during the same flight. The results show that the best battery is the one offering the best energy density (i.e., the Samsung battery pack), as the lighter weight requires a lower thrust and current demand, and thus impacts less negatively on the battery SoC.

We conducted a comparative experiment using the three battery packs: the drone was programmed to take off and maintain a stable hover for a predefined duration (520s), while we continuously monitored the Thrust, Voltage, and State of Charge (SoC) of each battery pack. The experiment results are illustrated in Fig. 3.25. At the end of the simulation, all batteries had a SoC around 50%, as a result of the similar electrical characteristics (i.e., capacity and voltage). However, the Sony battery has the lowest SoC, with the Samsung battery preserving an additional 2.3%, and the Tattuu battery an additional 1.3%. Even if the SoC difference may seem negligible, it translates into a longer flight time (approximately 30 seconds) and thus higher drone autonomy, which will be increased in longer flights.

A closer look at the plots and at the battery characteristics allows to understand the reason for the SoC difference. The Tattuu battery is heavier, which implies a higher thrust and,

consequently, a higher current demand throughout the entire flight. This is only partially compensated by the highest capacity (16Ah instead of 15Ah). The Sony battery has a higher energy density (+15.6%) and a lower weight, but this does not compensate for its lower capacity; as a result, the SoC at the end of the simulation is the lowest. The best compromise is thus the Samsung battery, thanks to a higher energy density and lower weight, which compensates for the lower maximum current. As a result, the Samsung battery is the best choice for the drone under design, with the highest SoC at the end of the flight.

This initial experiment highlights how a seemingly simple design decision, such as the choice of battery, can have a significant impact on drone autonomy and efficiency. Thanks to the modular and extensible nature of our simulation framework, users can easily prototype and evaluate different battery configurations. This enables informed decision-making during the early stages of system design, ultimately leading to more optimized and better-performing aerial platforms.

Wind Effect

Drones are frequently deployed in outdoor environments where weather conditions can vary significantly; therefore, wind is one of the most critical environmental factors affecting drone stability. The robustness of the flight control system against such disturbances is essential to ensure a safe and reliable flight.

To investigate the drone's behavior under wind influence, we utilized Unreal Engine to easily model wind as a constant external force applied to the drone along a specific direction, thereby emulating the aerodynamic drag and lateral pressure typically encountered in real-world scenarios.

We then conducted a series of experiments in which the drone was commanded to hover in place while being subjected to simulated wind forces. The results are presented in Fig. 3.26 and reflect two scenarios. In the baseline scenario, the drone uses the default flight control system with unmodified PID control parameters. In the wind-tolerant scenario, the PID controller is enhanced with logic specifically designed to counteract wind disturbances. To enable wind compensation, we introduced a new correction mechanism into the rotation PID, as shown in Eq. (3.27) and Eq. (3.28) (that modify Eq. (3.25) and Eq. (3.26)). This enhancement (in blue) involves augmenting the error signal with a component proportional to the integral of the drone's linear velocity in the wind direction over time [121]. The integration term accumulates the displacement caused by wind, effectively providing an estimate of the drone's drift due to external forces. Wind disturbance causes a bias in velocity, which, over time, becomes a positional drift.

This wind-compensation strategy is conditionally enabled depending on the drone's flight state, as determined by the emergency logic running within GvSoC. Specifically, the integral term is only active during hover mode, where position drift due to wind can be meaningfully countered without affecting the drone's navigation logic.

$$e_2(t) = \theta_x(t) - \theta_{x,\text{target}}(t) + C_n + v_y(t) \cdot C_v + \int_0^t v_y(\tau)\, d\tau \tag{3.27}$$

$$e_3(t) = \theta_y(t) - \theta_{y,\text{target}}(t) + C_n + v_x(t) \cdot C_v + \int_0^t v_x(\tau)\, d\tau \tag{3.28}$$

The comparison clearly shows that the baseline controller is unable to maintain position effectively under wind stress, resulting in significant drift (blue line of the left subplot of Fig. 3.26, where wind is simulated as a drag force pushing the drone in parallel w.r.t. the plot y-axis). In contrast, the modified controller successfully stabilizes the drone after a few seconds of excessive correction, thus demonstrating the importance of considering and simulating real-world disturbances while designing flight control system algorithms. The simulation allowed to estimate that the countermeasure did not impact current consumption (right): the solution without the countermeasure requires an average of 49.762 A, while the solution with the countermeasure requires an average of 49.766 A (less than 1% increase). The difference is marginal and does not significantly impact the battery consumption (90% vs 89.99% SoC at the end of the experiment).

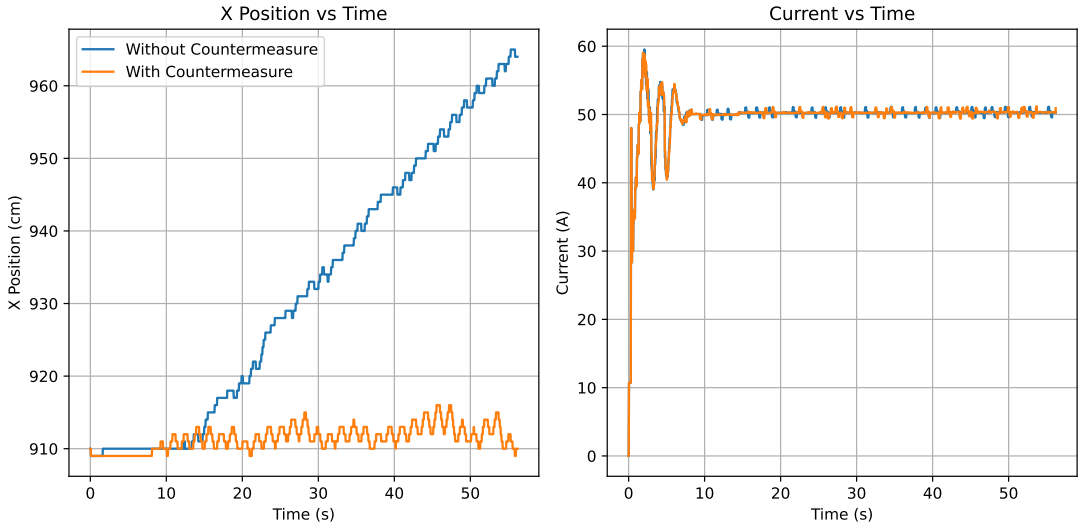


Fig. 3.26: Wind effect on drone trajectory (left) and current demand (right) with and without countermeasure.

This experiment highlights the critical role of control algorithms in adapting to real-world conditions and demonstrates how a simulation-based framework like ours can be used to test and iterate on such algorithms in a controlled and repeatable environment, with awareness also of complex power implications.

Development of the Fault Management System

This section focuses on the development of the fault management system, which requires simulating faulty behaviors to ensure correct detection and to implement effective countermeasures.

To this extent, we focus on two different aspects. First we analyze drone behavior under simulated Motor Failures, illustrating the critical role of real-time fault detection and reactive control strategies in enabling controlled emergency landing and preventing catastrophic events. Then, we focus on the complex dynamics of battery aging and thermal response to demonstrate how the power subsystem can impact performance and trigger essential safety mechanisms, thereby further validating the importance of comprehensive system monitoring. All the results regarding the fault simulations are summarized in Table 3.7.

Motor Failures

Motor failures can occur due to various reasons, including mechanical degradation, aging components, or the accumulation of debris and particles within the motor housing. Such failures can significantly compromise flight safety, especially in aerial systems that rely on multiple rotors for stability and control. For this reason, it is crucial not only to predict these failures in advance but also to design effective runtime countermeasures that can mitigate their impact and prevent catastrophic crashes.

To analyze this case, we considered two situations: in the first, the motor experiences a permanent fault and becomes non-operational; in the second, the motor undergoes a transient fault but later recovers its normal functionality. These two scenarios are examined to illustrate different approaches to fault management.

- **Permanent fault**
  To simulate the permanent fault, we created a new Unreal Engine blueprint to capture an interrupt to activate the faulty behavior at any time during drone operation (e.g., whenever the number '0' is pressed on the keyboard, the interrupt is triggered). The drone starts in its normal mode, taking off and reaching a stable hovering position. As soon as Unreal Engine detects the user instruction to activate the faulty behavior, it sends an event signal over TCP/IP communication.
  This affects both the SystemC AMS model and the drone software. In SystemC AMS, the simulation of the faulty motor is disabled, which also reduces battery power consumption accordingly. On the software side, the event is interpreted by the fault management system as a fault detection. It is worth noting that the system response was intentionally designed not to be immediate, in order to better reflect real-world conditions, where fault detection, validation, and corrective actions naturally introduce reaction time delays.
  The decision to trigger the fault through Unreal Engine was somewhat arbitrary, but it offers practical advantages: it centralizes user interactions within a single environment, avoids the need for additional mechanisms such as timers, and effectively emulates the role of telemetry equipment. Nonetheless, this is only one possible approach, and alternative fault modeling methods remain feasible, for example, using timed events directly within the SystemC AMS simulation at the ESC controller level.
  With no fault management, the drone tips toward the failed motor and starts yawing, as depicted on the left-hand side of Fig. 3.27. When the motor failure occurs (indicated by a blue dot), the drone briefly attempts to stabilize but ultimately loses control. Within seconds,

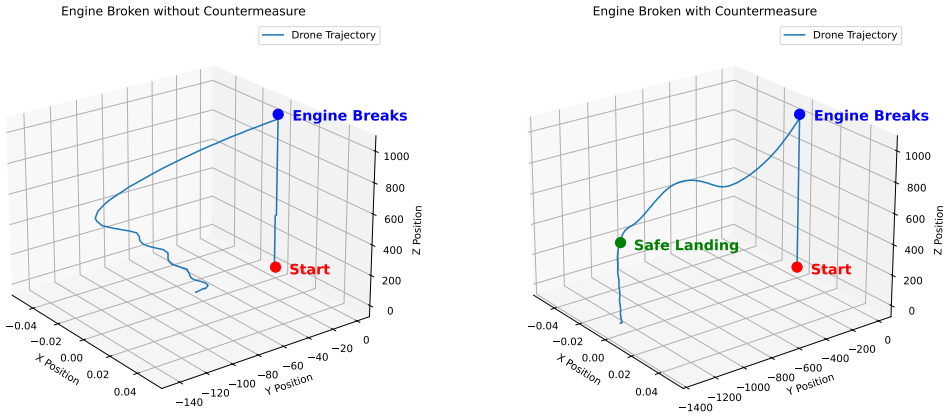


Fig. 3.27: Trajectory Comparison of the Permanent Engine Fault with and without countermeasures

it begins to spiral and fall uncontrollably, as evident from the chaotic and diverging path of its trajectory.

The simulation framework allowed to design a simple yet effective countermeasure: upon detecting the motor failure, the Fault Management System disables the motor that is symmetrically opposite to the failed one, attempting to regain balance and initiate a controlled emergency landing (as suggested by [122]).The right-hand side of Fig. 3.27 shows the effect of the fault management strategy, from failure detection to countermeasure activation and subsequent operations. As soon as the engine fault occurs (again marked by the blue dot), the opposite motor is also deactivated. This deliberate action balances torque distribution and allows the drone to regain stability. The drone then initiates a Safe Landing sequence, marked by the green dot, and successfully descends in a controlled manner, minimizing the risk of damage. It is important to highlight that this countermeasure significantly extends both flight distance and duration, as can be observed by comparing the left and right plots of Fig. 3.27. Without it, the drone would lose balance, collapse, and crash almost immediately.

- **Transient fault**

  To make the experiment more comprehensive, we also modeled a transient fault in the motor, i.e., the motor stops functioning for a limited period of time before returning to normal operation. We repeated the experiment under these conditions to evaluate the system's response.

  A transient fault is challenging, as it requires detecting not only the onset of the fault, but also when the motor resumes normal operation. If this is not monitored (left-hand side of Fig. 3.28), the just developed fault management strategy would intercept the motor malfunction (blue dot) and stabilize the drone by turning off the opposite motor. As soon as the faulty motor becomes active again (black dot), the motor configuration becomes unstable and causes once again a sudden yaw torque and a roll/pitch dip, thus causing an unpredictable behavior and crashing the drone.

Vice versa, as shown on the right-hand side of Fig. 3.28, the dynamic activation of the countermeasure enables the drone to remain airborne and to safely transition into the Safe Landing procedure as soon as the motor becomes active again.

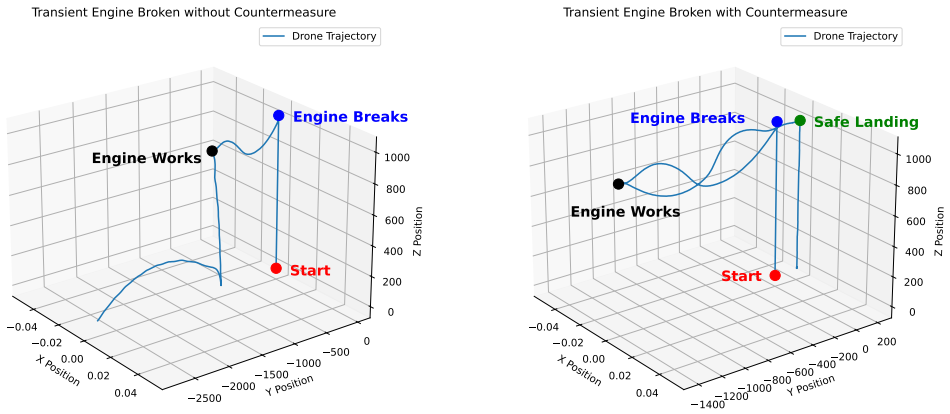


Fig. 3.28: Trajectory Comparison of the Transient Engine Fault without (left) and with (right) countermeasures.

This experiment highlights the importance of runtime fault detection and reactive control strategies in aerial robotics. The ability to handle hardware failures, particularly motor malfunctions, can significantly enhance the reliability and safety of drones operating in critical or unpredictable environments. Most importantly, our simulation framework enables this kind of fault-aware experimentation: by integrating real-time signals, sensor feedback, and custom logic within the digital twin, developers can prototype and validate fault detection mechanisms, test countermeasures, and assess system resilience in a controlled and repeatable setting. This makes the platform a powerful tool for safety-critical drone design and validation.

Battery dynamics

The battery is a fundamental component in drone design, affecting not only energy availability but also the overall reliability and safety of the system [123], as its long-term efficiency and thermal characteristics play significant roles in the drone's operational integrity. In particular, battery aging and temperature management are critical aspects that must be considered, as both can substantially impact performance during flight and under stress. To address these concerns, we integrated a model into our simulation framework that supports the evaluation of both battery aging and thermal behavior over time. This was possible because the battery model is white-box, rather than a predefined component, and it can thus be customized and extended beyond the initial model in described in [97].

- **Battery thermal model**
  Temperature is a crucial metric, especially under high-current loads such as takeoff or rapid maneuvering, where excessive heating may lead to reduced performance or even safety

hazards [124]. We thus extended the circuit model described in [97] with the generation and dissipation of heat, by using a physics-based approach. The heat generated in the battery is predominantly due to internal resistive losses and is described, thanks to basic electric-thermal laws, by the following equation:

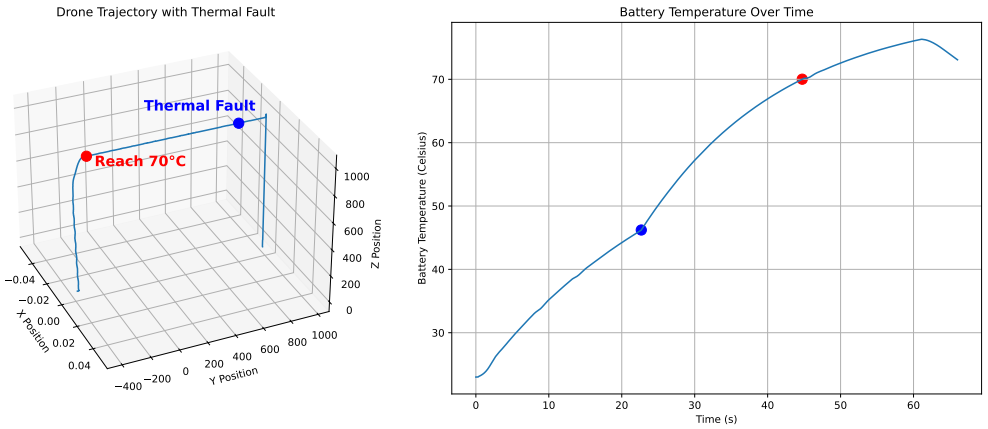


Fig. 3.29: Battery thermal fault: flight trajectory (left) and corresponding temperature evolution. The blue dot represents the beginning of the thermal fault, which generates an anomalous temperature increase, and the red dot marks the reaction of the Fault Management System, which activates a safe landing.

$$Q = \int_0^t \left[I_{\text{total}}^2 \cdot (R_{\text{int}} \cdot N_{\text{cell}} + R_{\text{pack}}) - P_{\text{coolant}} \cdot \Delta t \right] dt \tag{3.29}$$

Here, the total internal resistance contributes to heat generation, while P_{coolant} accounts for the heat dissipated to the environment through convective cooling. The temperature increase over time is given by:

$$\Delta t = \frac{Q}{C_{\text{pack}}} \tag{3.30}$$

where C_{pack} is the heat capacity of the battery pack. The cooling effect itself is modeled according to the Convective Heat Transfer Model:

$$P_{\text{coolant}} = h \cdot A \cdot \Delta t \tag{3.31}$$

where h is the convective heat transfer coefficient, A is the effective surface area, and Δt is the temperature differential between the battery and the ambient air.

An overheated battery cannot guarantee a reliable power supply for the motors, as the output voltage may drop sharply or even cease entirely in the case of thermal runaway. For this reason, our Fault Management System incorporates battery temperature monitoring. When the temperature exceeds a defined threshold, a thermal fault signal is generated, which in turn activates the Safe Landing routine and halts drone operation. This mechanism enhances resilience against thermal stress, particularly during extended missions or periods of high

power demand. The choice of threshold is critical and should take into account the thermal stress that the battery may undergo during the landing process, plus the altitude at the time of fault detection, to ensure that the battery remains within its safe operating range.
An example of this logic is illustrated in Fig. 3.29. In this scenario, the battery temperature rises uncontrollably due to a thermal fault that could be caused by battery wear, damage following an accident, excessively intense charge and discharge cycles, etc. The fault is activated in correspondence to the blue dot, and simulated by applying a multiplier that artificially increases the battery's heat generation. This leads to a faster temperature increase, until it reaches the identified safety threshold of 70°C (red dot) and the Fault Management System activates safe landing.

- **Battery aging**
  Battery aging is an inherent and irreversible process that progressively alters the electrochemical and physical properties of a cell over its operational lifetime. This leads to a reduction in the available charge storage capacity and an increase in internal resistance, ultimately impairing energy efficiency, power capability, and overall system reliability. To account for these effects in the battery model, we simulate both capacity degradation (by reducing the effective total capacity relative to the nominal specification) and the increase of the battery internal resistance (through a scaling factor). This enables a more realistic emulation of how batteries deteriorate under real-world operating conditions. The dual-parameter modeling approach, used in conjunction with the thermal model, improves modeling accuracy compared to approaches that only reduce capacity, as it captures the two primary electrochemical indicators of battery aging.
  Integrating this aging model also enables runtime monitoring mechanisms that enhance flight safety. By accounting for battery health within the Fault Management System, the drone can perform runtime monitoring of the battery, e.g., to trigger a Safe Landing if the estimated battery condition falls below a critical threshold. This not only prevents excessive stress on the battery but also facilitates timely replacement, thereby improving overall system reliability.
  To assess the interaction between battery aging and thermal behavior, we designed a dedicated experiment in which the drone takes off and hovers for a fixed period. Two simulations were conducted: one using a new battery and another with an aged battery characterized by reduced capacity and increased internal resistance. The temperature and the State of Charge profiles over time of both cases were recorded and are shown in Figure 3.30. The results indicate that aging has a significant impact on thermal performance, particularly during periods of high power consumption. Notably, the aged battery reached a peak temperature of 47.3 °C, compared to 46 °C for the new battery. Although the absolute difference may appear small, even slight temperature elevations can have serious implications for safety and operational limits, particularly under extreme conditions or with repeated use.

These experiments highlight the flexibility and extensibility of our simulation framework. Its equation-based modeling approach enables the incorporation of complex battery dynamics,

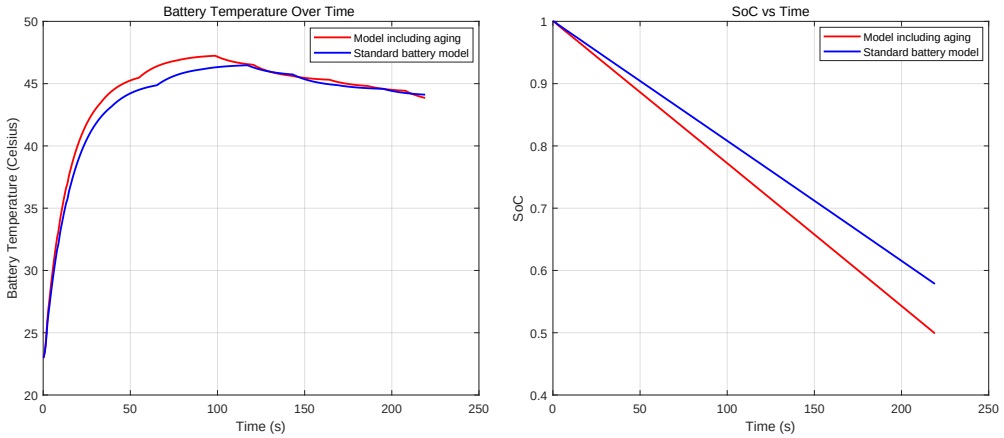


Fig. 3.30: Battery temperature and State of Charge over time for new and aged batteries during a hovering flight. The aged battery has a real capacity of 0.8× the nominal one.

Table 3.7: Fault Injection Campaign Summary for the Multirotor Drone Case Study

Domain	Fault Models	Injected Faults	Est. Sim. Time
Electrical	Short, Open, Parametric (ESC/Motor)	24	~ 56 mins
Mechanical	Friction, Propeller Degradation	12	~ 28 mins
Thermal	Component Overheating	2	~ 3 mins
Total		**38**	**~ 87 mins**

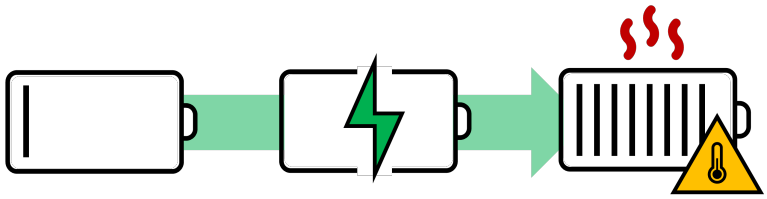
Note: Simulation times are estimated CPU elapsed times.

such as aging and heat dissipation, with a high degree of realism. Designers can accurately evaluate the long-term behavior of energy systems and test fault-handling strategies under various degradation scenarios. Such capabilities are crucial for ensuring the robust and safe operation of autonomous aerial systems.

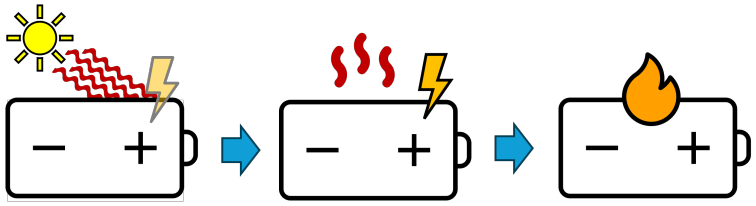
3.9 Lithium-ion battery

Nowadays, batteries are gaining increasing importance and appeal in modern technology, powering everything from portable electronics to electric vehicles and renewable energy systems [125]. Lithium-ion batteries dominate the battery market, and their composition constantly evolves to improve their performance and output [126]. Performance improvements typically refer to the amount of power the battery delivers, its capacity, the rate at which it can charge, and the temperatures it can withstand. This last characteristic is crucial, as sudden temperature changes can cause overheating, leading to serious problems, such as thermal runaway, short circuits, etc. [127]. It is well-known that once a critical state is reached in which the accumulated heat is much greater than the dissipated one, batteries can reach the point of igniting or even exploding, as pictured by Fig. 3.31 [128]. Obviously, these kinds of incidents are dangerous in any setting in which they occur and, thus, must be avoided at all costs. The relevance of this potentially harmful phenomenon rises in areas such as automotive, where interest in produc-

ing electric vehicles is growing. The battery packs installed in electric vehicles reach significant sizes, which means a bigger impact on humans and the environment. Unfortunately, recent cases of electric cars catching fire or exploding due to battery pack failures have been frequent [129]. For all these reasons, ensuring the regular operation of batteries is critical to their use in various



(a) Battery overheats while charging.



(b) Battery with a fault caused by an external heating source.

Fig. 3.31: Battery overheating caused by different factors.

systems, particularly for safety purposes, but not limited to. Using a battery safely and maintaining a linear input and output current flow helps avoid additional performance problems, such as the total capacity [126]. High temperatures can degrade the internal components, such as the electrolyte and electrodes, accelerating wear and reducing the battery's overall lifespan [130].

The design of a battery inherently involves a delicate trade-off between performance and safety, where pushing the limits of energy density and power output can increase the risk of overheating-related issues. Simulation is a powerful tool in this context, enabling engineers to accelerate the design process and optimize battery performance while ensuring safety [131]. By replicating various, even extreme, operating conditions in a virtual environment, simulation facilitates the evaluation and validation of safety levels, thereby contributing to the development of more robust and reliable energy storage systems. This paper analyzes the description of a circuit model of a lithium-ion battery, introducing fault models related to both electrical and thermal components. The injection of such faults inside the circuits representing the battery behavior allows us to simulate the system under several scenarios. The main points of this work are:

- Modeling the behavior of batteries as electrical circuits, which can be used both to design and to model already existing battery cells;

- The injection of fault models [73] into the equivalent circuit for simulating dangerous conditions related to overheating;
- Discussion related to the presented methodology based on simulation results: fault simulation helps test the batteries even in the worst conditions, sharpening their design phase.

3.9.1 Battery Modeling

Let us now see how to model a battery and its thermal components through equivalent electrical circuits. With this representation, we can handle fault injection directly, thus easily simulating many scenarios.

Equivalent Circuits for Batteries

Lithium-ion batteries are composed of several elements that generate an electrochemical reaction, thereby delivering the stored energy in response to a current demand. Studying the structural components of the battery in detail is the key element of its design: different amounts of certain elements rather than others will shift the balance of the construction compromise of this kind of battery. On the one hand, considering a specific volume occupied by the battery, filling up most of the space to increase energy storage enhances the battery's performance. On the other hand, battery safety is compromised, as there is no longer sufficient space for a potential cooling system. However, modeling all the chemical processes in detail is complex and unnecessary if one's goal is only to analyze battery behavior. Electrical equivalent circuit models are widely used for representing the dynamic behavior of batteries due to their simplicity, flexibility, and computational efficiency [131]. Modeling a battery using equivalent electrical circuits, as shown in Fig. 3.32, involves representing its dynamic electrical behavior with components like resistors, capacitors, and voltage sources [132]. Specifically, the circuit branch on the left side illustrates the section devoted to the evolution of state-of-charge (SOC), represented by V_{SOC}. The capacitor C_B represents the charge stored in the battery. It is discharged by the I_B load current, i.e., that required from the battery. In the right branch of Fig. 3.32, we can see the nonlinear dependence between the voltage delivered by the battery and the SOC. The R_S resistance models the voltage drop due to the inherent ohmic losses of the battery itself and is also affected by SOC. This dynamic relationship can be easily derived from the battery datasheet if multiple SOC vs. voltage curves are reported. This approach provides a simplified and effective way to capture key characteristics of the battery, such as open-circuit voltage (OCV), internal resistance, transient responses, and SOC dependencies. By capturing the electrochemical processes in an abstracted electrical framework, electrical equivalent circuits offer a practical approach to analyzing and predicting battery performance under various operating conditions. They are particularly effective for applications that require real-time simulations, such as battery management systems (BMS) in electric vehicles, where speed and accuracy are crucial. Those equivalent models can also be extended to include thermal and aging effects, allowing for a more comprehensive evaluation of battery performance and safety throughout its lifecycle. This approach facilitates optimization and validation during the design phase, supporting the development of efficient and reliable energy storage systems.

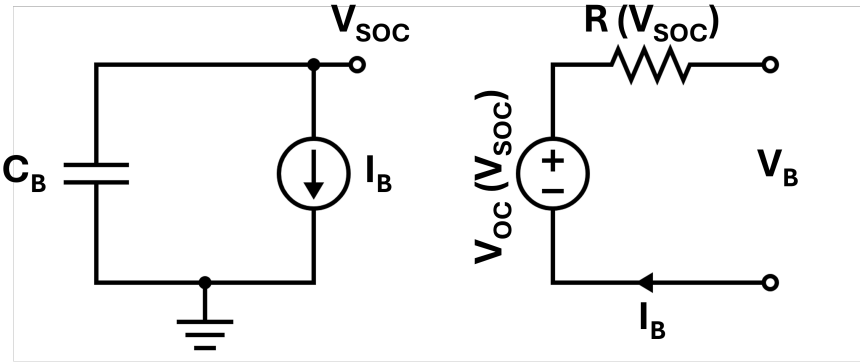


Fig. 3.32: Equivalent Circuit for modeling the behavior of a lithium-ion battery.

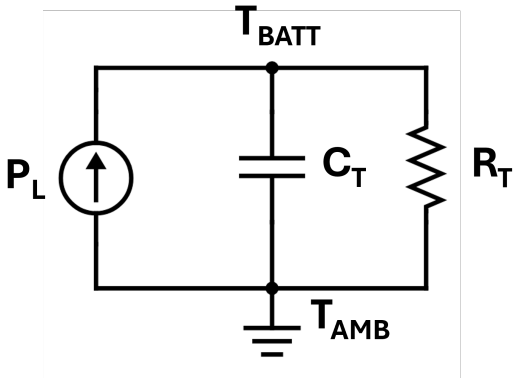


Fig. 3.33: Thermal Equivalent Circuit for measuring the temperature of a lithium-ion battery.

Thermal Battery Modeling

As anticipated in the previous section, extending the model of a battery with its thermal component facilitates its analysis and study from a safety perspective. Modeling the thermal equivalent circuit of a battery is highly useful because it provides a simplified and effective way to understand and predict how a battery manages heat during operation. Batteries generate heat due to internal resistance and electrochemical processes, and if not properly managed, this heat can lead to performance degradation, safety risks, and reduced lifespan. In this context, the concept of equivalent circuits returns as the focus of this important modeling phase for batteries. In fact, the thermal component is modeled through an electrical circuit, exploiting the analogy between the electrical and thermal domains. In this way, we can easily model heat flow through a current flow, the thermal resistance of materials as electrical resistance, and the thermal capacity through electrical capacitors. An example of this type of thermal model is depicted in Fig. 3.33, where the current generator P_L represents the power dissipated by the battery, which is then converted into heat that is emitted. R_T and C_T represent the thermal resistance and capacitance of the battery, respectively, which describe how the battery dissipates the heat produced. If we measure the voltage across this circuit, we obtain the temperature difference against the ambi-

ent. Then, feedback on the voltage-temperature relationship on the electrical side of the model is provided, as described in [133].

3.9.2 Thermal Fault Analysis of Batteries

In this section, several faults that can affect both electrical and thermal components of batteries are presented. Afterward, these fault models are applied to a case study based on the model shown in the previous section.

Multidomain Fault Modeling

One of the worst problems with lithium batteries is the risk of thermal runaway. This phenomenon occurs when the amount of heat the battery produces exceeds the amount of heat dissipated. This fatal consequence can happen for multiple reasons: an insufficient cooling system for the properties of the battery and extreme user conditions such as very high ambient temperature. Other causes may include excessive stress on the battery, such as rapid charge and discharge cycles. Thermal runaway is particularly problematic because it can cause battery fires or, in worst cases, explosions. There are other problems associated with overheating of batteries beyond thermal runaway, including capacity losses, battery swelling due to gases produced by the heat, electrolyte breakdown, and internal short circuits [134, 135]. In this context, several fault models that can cause thermal runaway are injected into the equivalent thermal circuit. The first one is a parametric fault with respect to the value of the battery thermal resistance, varying from the nominal conditions. This fault can be caused by multiple scenarios, such as inefficiency in the cooling system, assuming our battery is part of a larger, and therefore cooled, battery pack. A change in the battery's geometry due to an external impact can also lead to this fault. These circumstances prevent the battery from dissipating heat normally, resulting in an increase in temperature. This fault is introduced by varying the resistance value in the equivalent thermal circuit, thereby altering the battery's structural properties. Another fault that can cause battery overheating is the battery's exposure to external heat, which may be generated by external agents or nearby components. The unintended heat source could then raise the temperature of our battery, thus changing its performance. This fault is modeled by injecting a source of electric current (equivalent to heat) into the equivalent thermal circuit, affecting the measured voltage (equivalent to temperature) value.

After analyzing the possible thermal scenarios, let us examine the types of faults that can affect the battery from a purely electrical perspective. Since the battery is represented through an electrical circuit, electrical faults that can be injected are a state-of-the-practice, as outlined in several standards, such as ISO26262 [1].

An open-circuit fault occurs when electrical continuity is disrupted, preventing current flow. Causes include physical breaks, overheating, aging, or manufacturing defects. This fault is simulated by introducing a high-value resistor at the fault location. A short-circuit fault arises when normally separate circuit points make contact, creating an unintended low-resistance path. This allows excessive current to bypass the load, often due to insulation failure, external objects, or

faulty components. It is modeled by adding a new branch between two nodes. In batteries, short circuits can lead to fires by damaging internal components. Additional fault models include voltage or current sources to simulate external influences and parametric faults, which alter the nominal values of circuit elements.

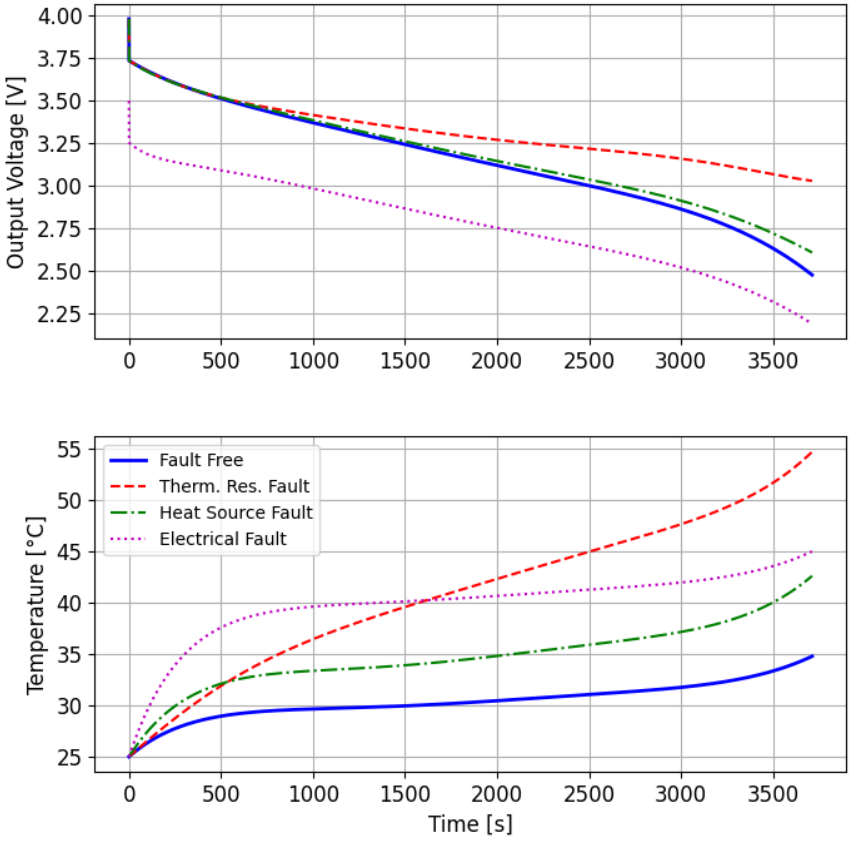


Fig. 3.34: Simulation of the battery model for a complete discharge cycle at 1C rate.

3.9.3 Fault Simulation inside Batteries

The battery model, following the structure shown in Figs. 3.32 and 3.33, was coded in SystemC AMS, with the Panasonic NCR18650B battery serving as a case study. This model was chosen because of the availability of datasheets and its use in multiple areas. The faults were added to the model by parametrization of the components meant to be faulty. A fault selector activates the single fault by selecting the faulty components using a control signal. As for the previous case studies, the overall results of this fault simulation campaign are reported in Table 3.8. In the simulations, a full discharge cycle was performed, starting from a 100% charged battery. We simulated this scenario with constant current loads to recreate a linear discharge. The scenario

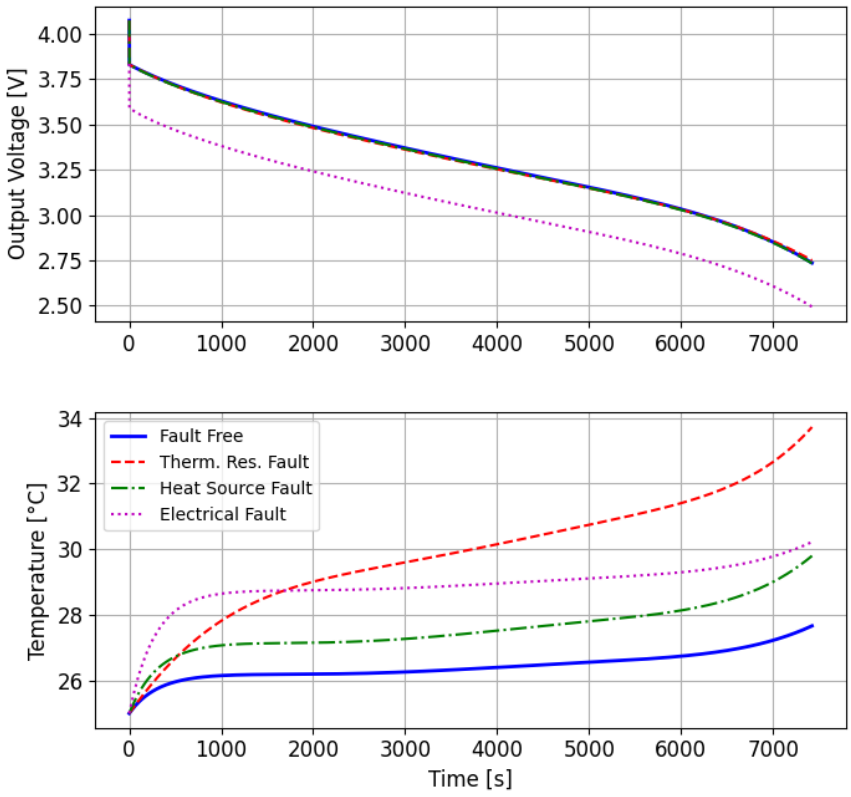


Fig. 3.35: Simulation of the battery model for a complete discharge cycle at 0.5C rate.

with a current load of 0.5C, *i.e.*, 1.6 A, is shown in Fig. 3.35, while a current load of 1C, or 3.2 A, is applied Fig. 3.34. The two graphs show four simulations each: with the blue color and solid line, the fault-free simulation is identified, which we need to make the comparison with the fault scenarios. The red color with a dashed line represents the thermal resistance fault, which means increasing the value of R_T (Fig. 3.33) from 6.7Ω to 30Ω in both simulations. As a result, we can see that the battery's temperature rises in both cases, but less heat is generated with a lower required current load. We can also observe that the output voltage increases, which is due to the fact that the battery remains within the optimal use temperature range. Higher temperatures enhance ion mobility and reduce the viscosity of the electrolyte, thereby facilitating better charge transport. This phenomenon persists until the optimal temperature range is exceeded; then, the components begin to be damaged by excessive heat. As shown in Fig. 3.35, with 0.5C load current, we notice that the voltage is not much subject to this phenomenon, as the temperature rises more slightly compared to the same fault scenario (Fig. 3.34). The green color and dash-dotted line identify the second simulated thermal fault, which is the presence of additional heat from outside. The effect of this fault is to double the power loss of the battery, which represents the heat produced by energy waste. Similar to the previous scenario, the internal battery temperature also increases, but this time due to additional and constant heat.

Although the heat growth appears more pronounced at the beginning, it then stabilizes and does not reach the values related to the thermal resistance fault. In multidomain systems, this setting can be very frequent, as the heat may come from one or more nearby batteries or other electromechanical components. Lastly, the magenta color with a dotted line indicates the electrical fault that has been injected. This fault represents a 0.15Ω increase in R_S resistance in Fig. 3.32, which may be due to multiple possible causes, including damage to the battery from impacts, aging, or other defects. The fault indicates that increased internal resistance causes the battery to dissipate more power, generating heat. This will result in reduced power output compared to normal, and consequently, a higher battery temperature.

Table 3.8: Fault Injection Campaign Summary for the Lithium-ion Battery Case Study

Domain	Fault Models	Injected Faults	Est. Sim. Time
Electrical	Increased Internal Resistance (R_S)	7	~ 7 mins
Thermal	Thermal Res. Fault, External Heat Source	8	~ 9 mins
Total		**15**	**~ 16 mins**

Note: Simulation times are estimated CPU elapsed times.

3.10 Summary and considerations on the fault models injection and simulation

This chapter has provided extensive empirical evidence supporting the multidomain fault modeling methodology defined in Chapter 2. By applying the proposed fault taxonomies to a diverse set of case studies, ranging from fundamental components like DC motors and MEMS accelerometers to complex systems such as battery packs, landing gear systems, and multirotor drones, we have demonstrated the versatility and robustness of the approach across the industrial, automotive, and aerospace domains.

The simulation results presented herein confirm that mapping physical defects to electrical primitives allows for the accurate reproduction of complex failure modes within standard behavioral simulation environments (Verilog-AMS, SystemC AMS, Simulink). Beyond mere reproduction, this validation campaign highlights several critical advantages for the engineering of safety-critical systems.

3.10.1 The strategic value of synthetic fault data

One of the most significant outcomes of these experiments is the capability to generate high-fidelity synthetic faulty data. In many industrial sectors, particularly aerospace and automotive, real-world fault data is scarce, expensive to obtain, or ethically problematic to generate (*e.g.*, inducing a real battery thermal runaway or a landing gear failure).

The methodology presented allows engineers to fill this "data gap" by:

- Overcoming Data Scarcity: As demonstrated in the Landing Gear System case study, the simulation framework can generate massive datasets covering rare failure modes that are statistically unlikely to appear in standard operational data but are critical for certification.
- Training AI/ML Algorithms: The availability of labeled faulty datasets is a prerequisite for training Data-Driven algorithms. The synthetic data generated by these models lays the foundation for developing robust PdM and Fault Detection and Diagnosis (FDD) systems, capable of identifying degradation patterns before catastrophic failure occurs.

3.10.2 Enhancing Design Robustness and Longevity Studies

The application of fault models during the design phase, rather than post-production, enables a "Design-for-Safety" approach. By injecting parametric faults (*e.g.*, resistance drift, friction increase) as a function of time, we can simulate the aging process of components. The battery pack simulations, for instance, demonstrated how thermal faults and aging affect the State of Charge (SoC) estimation and the overall lifespan of the energy storage system. The methodology supports rapid "what-if" analyses. The battery choice of the drone study exemplified how different architectural choices can be evaluated against their resilience to thermal faults, allowing engineers to select the most robust design before prototyping.

3.10.3 Interoperability and cross-domain propagation

Finally, the successful implementation of these models across heterogeneous environments, from rigorous mixed-signal analysis in Verilog-AMS to physics-based visualization in Unreal Engine, proves that the proposed fault taxonomy is platform-independent. This interoperability is crucial for capturing the propagation of faults across domains, as seen in the drone case study, where a thermal fault in the battery directly impacted the mechanical flight trajectory.

In conclusion, this chapter has validated that the proposed fault models are not only theoretically sound but also practically effective for generating the deep, multi-physics data required to assess system quality. The availability of such data opens the door to advanced applications. The next chapter, Chapter 4, will demonstrate how these faulty behaviors are utilized to implement concrete industrial solutions, including contract-based monitoring, human-in-the-loop digital twins, and automated testing frameworks.

3.10.4 Discussion on the approach limitations

While the simulation results presented in this chapter demonstrate a consistent functional behavior of the injected faults across diverse case studies, it is crucial to acknowledge the limitations regarding experimental validation against physical counterparts. The validation strategy adopted in this work is primarily qualitative and functional, relying on comparisons between nominal and faulty simulation traces and their alignment with theoretical predictions and literature data (e.g., CFD tools). A direct quantitative validation against real-world faulty data remains a significant challenge due to the inherent nature of the systems under analysis, but it is part of future work (whether possible). In safety-critical domains such as aerospace (Landing Gear System) or

high-energy storage (Battery Packs), inducing catastrophic failures, such as a hydraulic rupture or a thermal runaway, on physical prototypes involves prohibitive costs, safety risks, and ethical concerns. Consequently, publicly available datasets containing high-fidelity traces of such critical failures are extremely scarce. Therefore, since the aim of this thesis is functional safety, we should consider the proposed fault models as behavioral faults. They are designed to accurately reproduce the functional effect of a physical defect on the system's control loop (*e.g.*, the drop in pressure, the rise in temperature, or the loss of torque) rather than the micro-physical phenomenon itself (*e.g.*, the propagation of a crack in a specific alloy). We consider this level of abstraction sufficient and specifically intended for verifying system-level safety mechanisms and control logic robustness, as well as analyzing the functional safety of the case study.

```
module dcmotor(tau_drive, p, n, tau_load);
  // Parameters
  ...
  // Ports
  output tau_drive;
  input p, n, tau_load;
  // Nodes
  electrical p, n, n1, n2;
  rotational_omega tau_drive, tau_load, rgnd;
  // Reference nodes
  ground rgnd;
  // Branches
  branch (p, n1)  vm;    // motor voltage
  branch (n1, n2) ra;    // motor resistance
  branch (n2, n)  la;    // motor inductance
  branch (tau_drive, rgnd) motor; // DC motor
  // Behavior
  analog begin
    // Electrical motor internal dynamic
    V(vm)  <+ Ke * Omega(motor);
    V(ra)  <+ Ra * I(ra);
    V(la)  <+ La * ddt(I(la));
    // Motor dynamic
    Tau(motor) <+ + (Kt * I(vm));
    Tau(motor) <+ - (B1 * Omega(motor));
    Tau(motor) <+ - (J1 * ddt(Omega(motor)));
    // Gear contribution
    Tau(motor) <+ + (coef) * (Tau(tau_load));
  end
endmodule

module reductor(tau_drive, tau_load);
  // Parameters
  ...
  // Ports
  output tau_load;
  input tau_drive;
  // Nodes
  rotational_omega tau_drive, tau_load, gnd_mec;
  // Reference nodes
  ground gnd_mec;
  // Branches
  branch (tau_load, gnd_mec) gear;
  // Behavior
  analog begin
    N = r2 / r1;
    // Gear train dynamic
    Tau(gear) <+ - (N * Tau(tau_drive));
    Tau(gear) <+ + (B2 * Omega(gear));
    Tau(gear) <+ + (J2 * ddt(Omega(gear)));
  end
endmodule
```

Listing 3.1: Verilog-AMS modules implementing the DC motor and the gear train.

```
module motor(shaft, p, n, ploss, gnd);
    // PARAMETERS ------------------------
    // Motor parameters & Electrical/mechanical relations
     < equal to previous motor code >
    // Thermal model parameters
    parameter real Ta = 22; // Ambient temperature
    parameter real tco_wind = 0.004; // Winding coeff.
    parameter real tco_magn = -0.002; // Magnet coeff.
    // DYNAMIC COEFFICIENTS ------------
    real dyn_Ra, dyn_Kt;
    // PORTS ---------------------------
    inout  ploss, gnd;
    output shaft;
    input p, n;
    // NODES ---------------------------
    electrical p, n, n1, n2, ploss, gnd;
    rotational_omega shaft, rgnd;
    // Reference nodes.
    ground rgnd, gnd;
    // BRANCHES ------------------------
    branch (p, n1)  vm;         // motor vm
    branch (n1, n2) ra;         // motor resistance
    branch (n2, n)  la;         // motor inductance
    branch (shaft, rgnd) comp; // motor
    branch (ploss, gnd) temp;  // temperature
    // BEHAVIOR ------------------------
    analog begin
        // Electrical and mechanical motor internal dynamic
         < equal to previous motor code >
        // Ambient temperature + power loss
        V(temp) <+ Ta + (pow(I(vm),2) * Ra) * I(vm);
        // Temperature effects on motor parameters
        dyn_Ra = Ra * (1 + tco_wind*(V(temp) - Ta));
        dyn_Kt = Kt * (1 + tco_magn*(V(temp) - Ta));
    end
endmodule

module thermal(ploss, gnd);
    // PARAMETERS ------------------------
    parameter real R_th1 = 2.2;
    parameter real C_th1 = 4.1;
    parameter real R_th2 = 6;
    parameter real C_th2 = 10/r2;
    // NODES ---------------------------
    electrical ploss, t1, t2, gnd;
    inout ploss, gnd;
    // BRANCHES ------------------------
    branch(ploss, t1) in;
    branch(t1, t2)  R1;
    branch(t1, gnd) C1;
    branch(t2, gnd) R2;
    branch(t2, gnd) C2;
    // BEHAVIOR ------------------------
    analog begin
        I(R1) <+ V(R1) / R_th1;
        I(C1) <+ ddt(V(C1)) * C_th1;
        I(R2) <+ V(R2) / R_th2;
        I(C2) <+ ddt(V(C2)) * C_th2;
    end
endmodule
```

Listing 3.2: Verilog-AMS modules implementing the DC motor and the Cauer Network module.

4

Application of the fault taxonomy and techniques to real-world solutions

In the preceding chapters, this thesis has established a rigorous methodology for modeling multi-domain faults via physical analogies (Chapter 2) and has validated the physical fidelity of these models across a diverse range of simulation environments (Chapter 3). Having demonstrated the capability to generate high-fidelity synthetic faulty data, the fundamental question shifts from how to simulate a fault to how to exploit this capability to solve complex industrial challenges.

Simulation data, no matter how accurate, is of limited value if it does not drive decision-making or enhance system robustness. This chapter bridges the gap between theoretical modeling and industrial application. It demonstrates how the proposed fault injection framework serves as a foundational technology for advanced design automation and runtime verification.

The availability of the fault models developed in this research enables a paradigm shift in the development lifecycle of Industrial Cyber-Physical Systems (ICPSs). By integrating these models into higher-level workflows, we can address critical needs that standard testing approaches cannot satisfy:

- Scalability in Design: Moving from component-level analysis to system-level design exploration without the prohibitive costs of physical prototyping.
- Functional Safety: Utilizing faulty traces to mathematically verify safety monitors before deployment.
- Calibration and Robustness: Using fault injection to distinguish between environmental noise, calibration errors, and actual defects.

4.0.1 Chapter overview

This chapter presents several distinct applications that leverage the multi-domain fault models to deliver concrete engineering solutions:

Contract-Based Monitoring for Fault Detection

Building on the faulty datasets generated for the DC Motor and Drone, this section presents a runtime verification framework. It utilizes Time-Sensitive Behavioral Contracts to formally define the boundary between nominal and faulty behavior. We demonstrate how injecting faults

into the simulation allows for the rigorous testing and tuning of these contracts, ensuring they can distinguish between benign anomalies and critical failures in complex control loops.

Automated Design Exploration for Battery Packs

While Chapter 3 validated the single-cell model, this section presents an Automatic Design Tool for entire battery packs. By leveraging the scalability of the proposed thermal fault models, the tool enables the automated generation and simulation of complex pack architectures (*e.g.*, varying coolant layouts). This application demonstrates how fault injection can guide the design phase, identifying optimal thermal management strategies to mitigate propagation risks, such as thermal runaway.

Robust Calibration of MEMS Accelerometers

Specific challenges of smart sensors are addressed. We present a methodology that utilizes mechanical and electrical fault injection to validate an automated Calibration System. This application emphasizes the significance of synthetic fault data in ensuring that calibration algorithms are robust against internal defects and do not obscure genuine hardware failures.

D-MATE: Service-Oriented ATE Integration

Bridging the gap between simulated fault analysis and physical production testing, this section introduces D-MATE (Design Methodology for Connecting Automatic Test Equipment). We address the challenge of heterogeneous testing environments by proposing a service-oriented architecture. This application demonstrates how fault knowledge generated in simulation can be operationalized, enabling the seamless integration of testing machinery into the Industry 4.0 digital thread and standardizing the deployment of fault-detection routines.

Through these applications, this chapter provides the final evidence of the thesis statement: that physics-based multi-domain fault modeling is not merely a simulation exercise, but a critical enabler for the next generation of safe, reliable, and human-centric industrial systems.

4.1 Time-Sensitive Behavioral Contract monitors design

The increasing complexity of distributed control systems requires high reliability and precision from critical components to ensure smooth operation and early detection of faults, thereby maintaining efficiency and preventing costly downtime. A crucial component of many electromechanical systems is a DC motor, which is used for the actuation of industrial machinery, vehicles, or robots. Therefore, it serves as a running example throughout this work, as shown in Fig. 4.1. In this scenario, the motor controls the joint of a robotic arm, which performs operations on workpieces. Here, the DC motor is part of a complex setup comprising multiple actuators, sensors, and end effectors, all of which are regulated by a controller. Like any technical system, the DC motor can be subject to faults due to external influence factors as well as wear over time. This, in turn, could lead to potential faults or performance degradation, such as

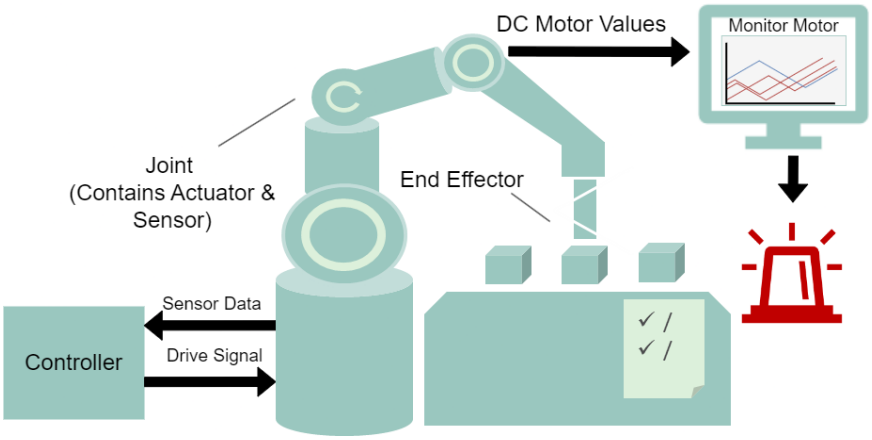


Fig. 4.1: Monitoring system behavior of industrial control systems.

a loss of the provided torque or variations in the necessary currents. These are examples of behavioral fault effects that can afflict the motor for multiple reasons, such as aging or changes in the system's working environment. Particular faults can have a significant impact on the overall production plant. Although most changes in the device under test do not turn out to be catastrophic, the system's functionality is compromised as it is no longer optimal. These types of faults can deteriorate the system's performance to the point of more serious failures, such as breakdowns that prevent the system from operating. Identifying the gradual deterioration in the performance of industrial instrumentation is crucial for maintaining optimal performance in an industrial process [136]. Such a Fault Detection and Isolation (FDI) process helps to avoid more severe damage in terms of repair costs and time to restore the production phase. Thus, early fault detection can make a significant difference in terms of operational efficiency, reliability, safety, and overall productivity.

By implementing a more proactive approach to fault detection, new maintenance strategies can enhance the reliability of distributed control systems. Contract-based design is often used to specify the expected behavior of system components in the form of assumptions about the components' environment and guarantees for the components' behavior under the condition that the assumptions are fulfilled [137]. Thereby, (non)-functional system requirements can be included during the design phase and checked for consistency throughout each design step. Moreover, the specified contracts serve as a basis for monitoring the system at runtime, enabling the early detection of deviations from expected norms and prompting early intervention in the event of potential faults.

This work addresses the challenges of FDI in multi-domain systems by combining the following concepts:

1. Contract-based design to distinguish between desired and undesired (faulty) system behavior.
2. Mapping of fault detection rules based on a threshold verification on contracts that can be evaluated in a co-simulation environment.

3. A simulation-based fault injection procedure highlighting the principles of the contract-based FDI rules.
4. A simulation of hardware monitors based on time-sensitive behavioral contracts to detect faults, validating the applicability for use during system operation.

By implementing the framework presented in this article, continued monitoring of system behavior will be achieved. As shown in Fig. 4.1, the monitors report an anomaly in system functionality, allowing the user to assess the severity of the fault. Based on the analysis, production can be restored by correcting the power supply or configuration parameters of the production system. Otherwise, if the fault is more significant, corrective actions will be taken. These actions may lead to the replacement of the component with a new or more appropriate one for the context. Further changes can be made at the system control strategy level to avoid, for example, accelerated system wear and tear.

4.1.1 Background

To prevent malfunctioning of control and automation systems, continuous runtime verification and validation are essential. Consequently, Fault Detection and Isolation (FDI) and tolerance in Cyber-Physical Systems (CPSs) have been extensively researched across various domains [138]. The importance of improving FDI through comprehensive modeling and simulation has been emphasized in the literature [139], while specific standards, such as IEEE 2427, have been introduced to focus on analog defect modeling [14]. Specific attention has also been highlighted regarding fault analysis in aerospace environments [140, 141].

Simulation plays a critical role in this context, underscored by the diverse ecosystem of frameworks available for CPS fault tolerance research. For instance, fault behaviors have been simulated in vehicle dynamics [142], and fault libraries have been provided for Modelica [143]. Furthermore, the complementary roles of SystemC and Matlab for system validation have been emphasized [144], along with the fault detection capabilities found in languages such as Verilog-AMS [145] and SystemC AMS [146]. Early FDI methods for distributed systems [147] and enhancements via co-simulation for IEC 61499 applications [148] have also been proposed.

Regarding the implementation of runtime verification, methods typically focus on software monitors. Monitors based on finite-state machines have been suggested [149, 150], as well as software monitors for behavioral and timing specifications based on assume-guarantee contracts [151]. While promising, running additional software to verify applications at runtime can result in severe performance overhead [152]. As a solution, hardware-based approaches utilizing Hardware Description Language (HDL) have been employed. Techniques include using a host machine to process monitoring data for real-time analysis [153], frameworks to translate Linear Temporal Logic (LTL) over finite traces (LTL_f) into automaton implementations for FPGAs [154], or transferring Property Specification Language (PSL) to Verilog code via compilers [155]. Hybrid hardware-software approaches have also been proposed to extract time traces and map variables to FPGA registers, though these may still incur significant performance overhead [156].

Recent research has also highlighted data-driven FDI approaches using machine learning [157, 158], which eliminate the need for system modeling by leveraging datasets from machinery operations. However, obtaining such datasets, especially those containing faulty data traces, remains challenging due to limited availability and confidentiality issues.

In contrast, design based on assumption-guarantee contracts alongside formal specifications has gained significant interest for CPSs fault detection [159–162], as it aligns with system requirements [163], robustness enhancement for analog systems [164], and predictive maintenance strategies within Industry 4.0 [165]. This work aims to consider the general concept of contract-based design [137] as a basis to derive formalized contract-based monitors. Similar to software-based runtime monitors suggested in the literature [151], these are intended to be general-purpose and applicable to verify any hardware component with a behavioral description. By formally specifying the nominal behavior of a multi-domain system, derived monitors can be used to validate fault detection. This approach is combined with a simulation environment for multi-domain systems where faulty models are injected [73], allowing the derived monitors to check whether the injected faults are correctly detected.

4.1.2 Enhancing Multi-Fault Detection with Contract-Based Co-Simulation

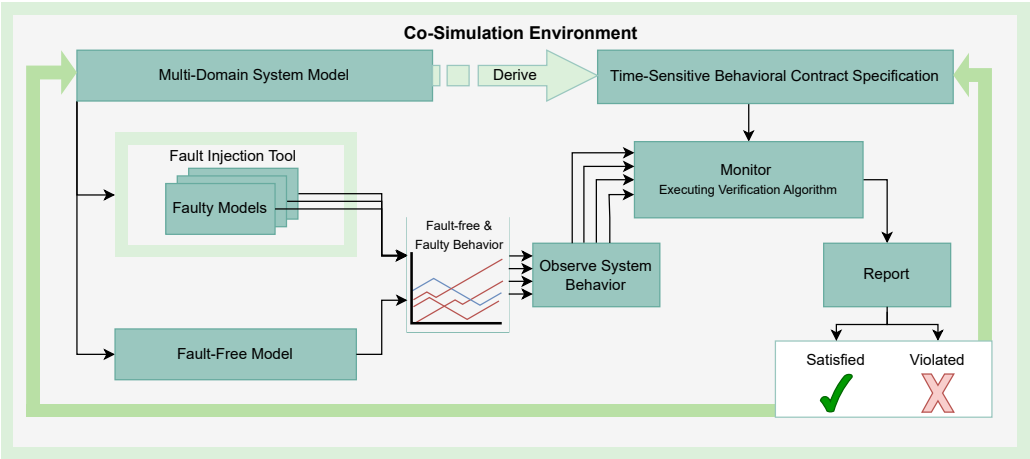


Fig. 4.2: Interaction of the multi-domain fault injection tool and the time-sensitive behavioral contract monitor simulation.

This section describes the co-simulation environment that enhances multi-fault detection for multi-domain system models with Time-Sensitive Behavioral Contract (TSBC) specifications and monitoring principle. The overall concept is visualized in Fig. 4.2. The contract-based design methodology is typically used to verify complex systems by providing a formal description of the system components, enhancing modularity and reusability. This work relies on the theoretical foundation as previously described [137]. Each system component is specified by a mathematical contract detailing its behavior. A contract $\mathbf{C}$ is defined by pairs of assumptions

A and guarantees **G**. An assumption **A** denotes the anticipated behavior of the environment, which could be a specific parameter or state value at a defined point in time. Guarantees **G** delineate the behavior of the component from activation to completion, provided the assumption is satisfied. Contracts can be refined into subcontracts, which require a valid conjunction and composition of all subcontracts. This mathematical framework enables the use of automated tools in the design and verification process, allowing potential design issues to be detected early and the system's reliability to be proven. The contract specification can be derived directly from the nominal behavior of the multi-domain system model. [151] proposed the concept of TSBCs within the contract-based design methodology to verify the functional behavior of industrial control systems. The observed system behavior can be monitored and evaluated with a verification algorithm that checks whether the expected system behavior is satisfied or violated. We combine this approach with the concept of multi-domain fault models of the previous chapters. Specifically, the faults are directly injected into the differential equations of the system, thereby altering its dynamic behavior. All these behaviors, generated under different working conditions through simulation, are exported as data series for the monitors to analyze.

The report of the TSBC monitor, which can be either valid or invalid, can be used in further design iterations, including adjustments to control strategies to refine the multi-domain system model or update the contract specifications. Accordingly, it is possible to relax the specification and allow a larger interval for the time and behavioral values, thereby preventing violations of the TSBCs. However, this does not rule out the occurrence of faulty behavior. Thus, adjustments to the system model could include strategies to prevent faulty behavior. A possible variation is to implement an alternative control strategy, refining the system's treatment in different work regimes. At runtime, in the event of a fault notification, the system can be stopped immediately, preventing further damage to the entire system. Subsequently, if the user is also the system's manufacturer, design updates can be made to it, changing its structural or functional parameters. Alternatively, after analyzing the cause of the fault, the user can update the system's functional requirements and then switch to a more suitable product for the context, *i.e.*, a system that satisfies the new reliability demand. Otherwise, the user can simply decide to replace the faulty component if it has reached the end of its life. Another improvement in the system's working condition involves the maintenance performed; monitoring system behavior can serve as the basis for predictive maintenance algorithms. Improving the maintenance cycle of the system helps to prevent the occurrence of faults [165]. In all these situations, the proposed co-simulation environment supports the design, development, and monitoring of a production line. The simulation and implementation of this framework refine the quality of testing and monitoring for the whole system. The functional quality of the verification testbench improves along with the accuracy and spectrum of failures that the monitors are able to intercept.

4.1.3 Contract-Based Multi-Fault Detection

A simulation model of a real system allows total freedom in recreating the most extreme working conditions without damaging the physical system. Therefore, simulation allows us to design the working environment without experiencing time and monetary losses. To perform simula-

tions for the previously introduced DC motor, a Verilog-AMS model was created, describing the equations in the `electrical` and `rotational_omega` domains. The system was then simulated using a SPICE-based simulator and a testbench module, which instantiated the motor module and fed it with the input voltage. White Gaussian noise was added to the simulated time responses to account for realistic sensor characteristics and to mitigate temporal discretization effects arising from implementations on digital signal processors. These data traces serve as the inputs for TSBC monitors. The presented potential fault models are injected directly into the differential equations of the motor. Regarding the Verilog-AMS code, fault injection is performed by adding branch contribution statements (identified by the symbol <+) to the existing equations, which represent the fault injection points. For example, the following code line shows how to inject an electrical fault:

```
V(p, n) <+ I(p, n) * open;
```

where `V(p, n)` is the voltage across the two electrical nodes `p` and `n`, and `I(p, n)` is the current across the nodes. The value `open` is the amount of resistance applied to this branch as a fault: if the value is very high, then we are simulating the interruption of the electrical line. The mechanical fault is injected equivalently, by specifying for example:

```
Tau(x, y) <+ Omega(x, y) * damp;
```

where `Tau(x, y)` is the mechanical torque produced across the two mechanical nodes `x` and `y`, and `Omega(x, y)` is the angular velocity across the nodes. The value `damp` is the damping coefficient we want to apply to this branch as a fault: the value of this coefficient determines how much the rotation is slowed down.

The expected behavior of the DC motor is visualized in 4.3. In the first milliseconds of the simulation, a current spike occurs during the acceleration phase, when the motor must overcome inertia to initiate motion. Then, the velocity stabilizes along with the current, maintaining a constant rotational velocity. After approximately 8 seconds of simulation, the input voltage is disconnected, and then the deceleration phase begins. Due to inertia, the motor continues to rotate for a few more milliseconds before finally coming to a stop. During the deceleration, the sign of the current becomes negative; however, it follows a similar behavior to that during the acceleration phase.

For TSBCs, the system behavior is delineated by the time variable t and data variable d. Assuming the physical time t is within a specified interval, the DC motor is supposed to guarantee a certain behavior d. Contracts are formally expressed as $C_i : t \in [t_1; t_2] \ ? \ d \in [d_1; d_2]$, which can be interpreted as follows: Is the value of t within the specified interval? If so, the data value d must also be within a certain interval. A violation of this specification is considered faulty behavior. To ensure compliance with the contract specifications, the TSBC-based monitors must track time and subsequently ensure that environmental data values are accurate. The progression of physical time t is referred to as the rate of clock cycles. TSBC monitors observe system behavior during specific actions triggered by events, which necessitates synchronization between the monitors and the actions they observe. The data variable d denotes system properties,

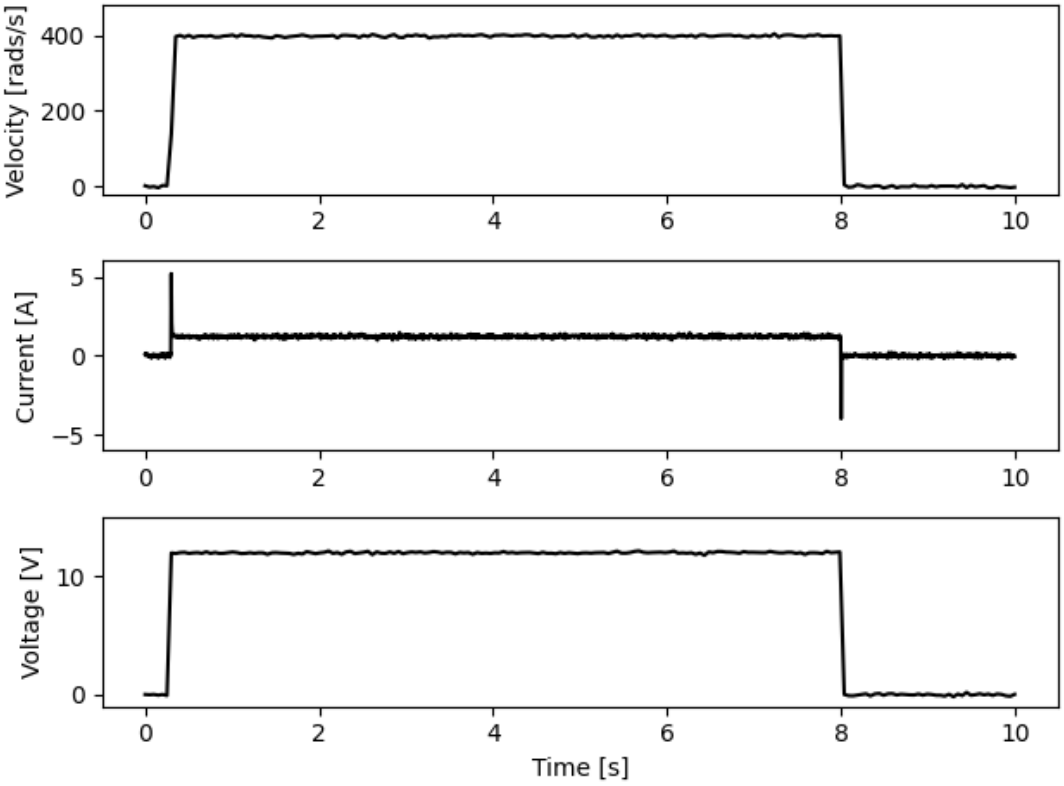


Fig. 4.3: Data traces of expected angular velocity, electrical current, and supply voltage of the DC motor over time.

accessible to monitors through measurement and digital representation via interfaces defined as *observers*. In the DC motor scenario, a TSBC specification could be defined with the supply voltage as a data value to ensure that the correct input is applied at the correct point in time (see 4.4). The contract specifications $Contract\_C_1$ through $Contract\_C_4$ enable the verification of functional parameters and the detection of faulty behavior in the supply voltage. The provided TSBC specification can also include an error for the supply voltage by 2% expressed by an interval, e.g. (11.76 V ≤ Supply Voltage ≤ 12.24 V). The following four contracts ($Contract\_C_1$ through $Contract\_C_4$) specify the assumed behavior as a means to verify that the proper voltage is being applied to the motor. This way, the monitor can evaluate and detect the wrong supply voltage, which could indicate an error in the motor's electrical supply.

$$Contract\_C_1 : t \in [0.2; 0.45] \; ? \; d \in [0; 12.24]$$

$$Contract\_C_2 : t \in [0.45; 7.9] \; ? \; d \in [11.76; 12.24]$$

$$Contract\_C_3 : t \in [7.9; 8.1] \; ? \; d \in [0; 12.24]$$

$$Contract\_C_4 : t \in [8.1; 10] \; ? \; d = 0$$

These contracts are critical, especially when the monitors are deployed in the real world. Providing an incorrect input voltage to the motor can happen due to human error or an external

failure. However, in the next simulations, the supply voltage to the motor remains the same, even in the faulty cases. Therefore, it is assumed that if any anomalies occur, they are not caused by human error or by a problem with the voltage supply. The focus of this paper is the consideration of behavioral analog faults that regard the motor itself.

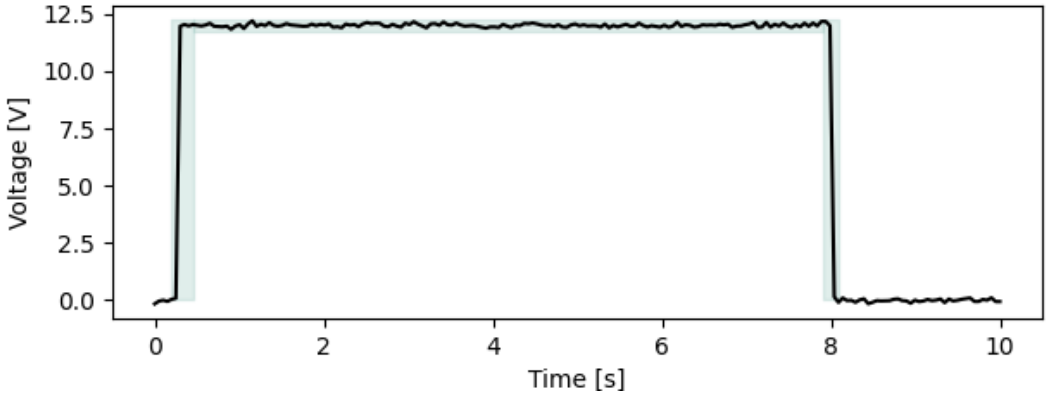


Fig. 4.4: Possible trace of the supply voltage, including the permissible range specified for the TSBC monitors.

Since the motor is an electromechanical system, we must consider both its electrical and mechanical aspects. Both components may be subject to change due to multiple reasons. Here, we considered the faults that best fit the context we want to recreate in simulation: a gradual deterioration of motor performance.

Turning to the electrical domain, we aim to simulate the reduced current resulting from an increase in armature resistance. Several factors can cause this fault, including wear and tear of the system's electrical components, a poorly placed connection, or external electromagnetic interference that causes a loss of current.

The electrical fault remains active throughout the simulation period, thereby simulating the constant wear and tear of the motor's electrical components. In this case, the fault primarily affects the electric current, decreasing the flow through the motor armature. To detect whether there is an electrical error and the current reaching the motor is less than expected, the following four contracts (*Contract_*C_5 through *Contract_*C_8), including a potential error of the current by 2% are specified:

$$Contract\_C_5 : t \in [0.2; 0.45] \ ? \ d \in [0, 6.12]$$

$$Contract\_C_6 : t \in [0.45; 7.9] \ ? \ d \in [1.037, 1.403]$$

$$Contract\_C_7 : t \in [7.9; 8.1] \ ? \ d \in [-4.08, 1.403]$$

$$Contract\_C_8 : t \in [8.1; 10] \ ? \ d = 0$$

The injected electrical fault also affects the angular velocity of the motor, which, although powered normally, is significantly decreased due to an insufficient power supply, preventing it from reaching the rated speed. The injected mechanical fault applies additional friction to the

rotary components of the motor. Multiple reasons can cause this fault, including the presence of debris or dust between the rotating parts, external agents slowing down the shaft rotation, worn surfaces due to motor aging, and more. Additionally, the fault is incremental, meaning its intensity increases gradually throughout the simulation. This change was adopted to simulate an increasingly severe effect of the fault. The effect of this fault is a larger braking force than the normal friction present in the system, thus decreasing the angular velocity. Accordingly, the effect of the faults (mechanical or electrical) can be noticed when observing the angular velocity (rad/s) as data condition by applying the following four contracts ($Contract\_C_9$ through $Contract\_C_{12}$), including a potential error of 2%:

$$Contract\_C_9 : t \in [0.2; 0.45] \; ? \; d \in [0; 408]$$

$$Contract\_C_{10} : t \in [0.45; 7.9] \; ? \; d \in [392; 408]$$

$$Contract\_C_{11} : t \in [7.9; 8.1] \; ? \; d \in [0; 408]$$

$$Contract\_C_{12} : t \in [8.1; 10.00] \; ? \; d = 0$$

The possible traces of both electrical and mechanical faults are presented in 4.5 and 4.6, respectively, integrating the contract specifications and highlighting the allowed interval of the time-dependent angular velocity and current.

Based on the provided TSBC specifications and observed behavior, monitors execute their verification algorithm. For contracts with both time and data conditions (normalized contracts) the verification algorithm follows a specific procedure starting with evaluation of the time condition. Afterwards, if the time condition is met, the monitor assesses the inner data condition. Otherwise, the monitor evaluates the outer data condition. Only if the evaluated data condition is satisfied, the entire contract is regarded as satisfied. Alternatively, it is considered violated, and monitors must decide how to proceed.

TSBC monitors analyze data traces and report on the correctness of system execution or contract violations. After a violation is detected, monitors may either continue verification in case of success in the future, known as recoverable monitoring, or halt verification until a subsequent request is received, also defined as unrecoverable monitoring. Recoverable monitoring assumes that monitors failing once may pass within the same request, allowing the environment to refine its response based on continued observation. Conversely, unrecoverable monitoring assumes that contracts violated once remain so for the duration of the current request, which is a suitable approach for monitoring critical fault conditions intolerant of any violations. When a critical fault is reported, identifying the cause of the problem is crucial. Once the cause of the problem is identified, a decision must be made: are corrections needed? Improvements can be made to either the simulation model or the monitors.

Regarding the model, investigating the nature of the fault is essential to understanding the severity of the problem. Simulating multiple fault scenarios helps define the system's limits. This determines whether the motor has the necessary specifications for the context. Another possible development related to the motor is to simulate the fault for a longer time. In this way, the effect of a persistent fault in the system can be analyzed. Based on the severity of the fault,

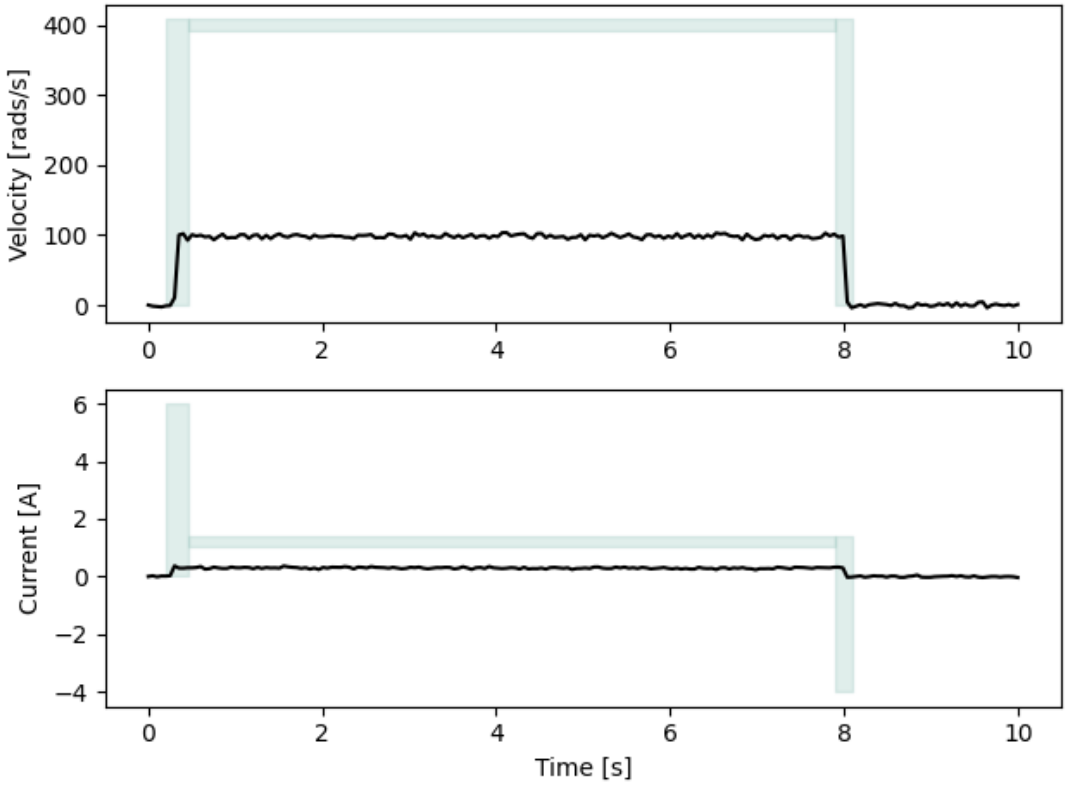


Fig. 4.5: Possible trace of the electrical fault model with highlighted permissible contract area.

the user decides whether and how to intervene. If that fault occurs, the system can be stopped immediately to avoid serious damage. Otherwise, a simple change to the system configuration, such as correcting the input signal, would fix the operation. In addition, simulating for a longer period would reveal whether and how the system controller would mitigate the fault.

Alternatively, changes to the contract specifications could include relieving the interval specifications and broadening the spectrum of allowed values. However, this does not improve the reliability of the system itself - it just prevents the detection of *smaller* deviations. Accordingly, specification of contracts should be done systematically, considering the precise requirements and potential behavior of the system.

4.1.4 Discussion

The proposed simulations clearly show that faults have non-negligible effects on the motor. Following a systematic design for TSBC-based monitors allows for the specification of a suitable fault detection mechanism that can be tested during the design phase. TSBC monitors report the problem, notifying the user as soon as the recorded values exceed the range accepted by the contracts. This approach was designed and tested in simulation by performing a case study. The introduced analog system was analyzed from the perspective of behavioral-level faults, thereby examining its functional aspects.

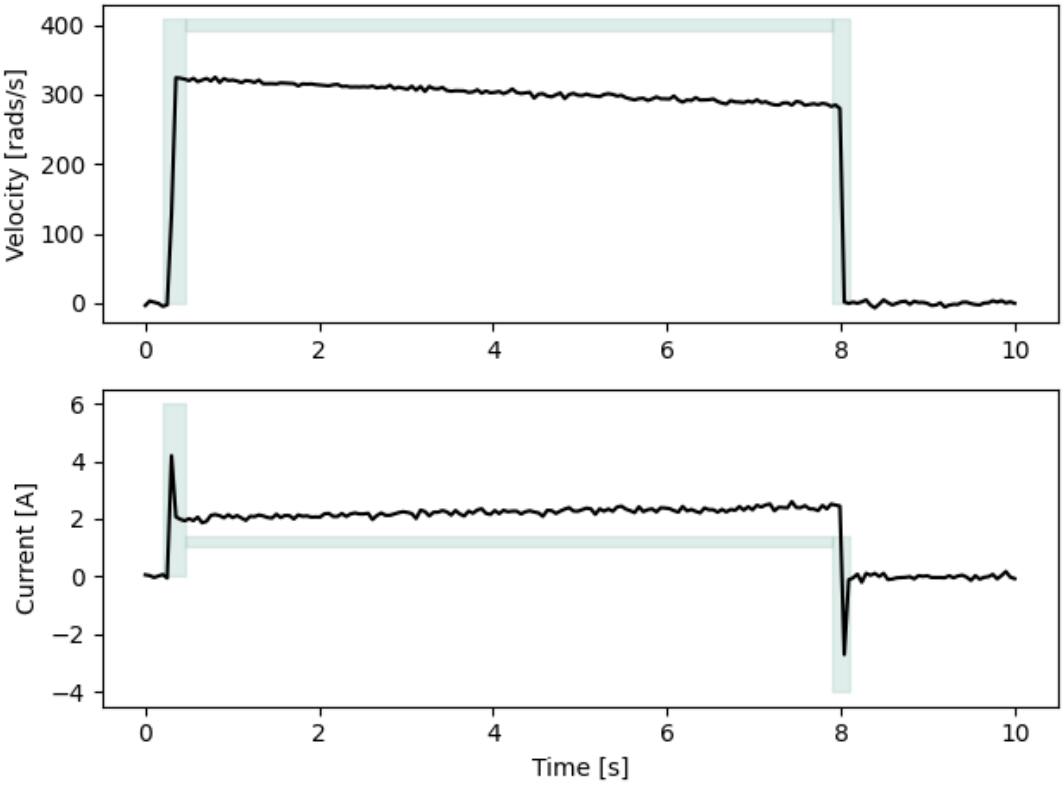


Fig. 4.6: Possible trace of a mechanical fault model with highlighted permissible contract area.

The implementation of this methodology remains applicable to various systems that incorporate physical components. Application to other systems requires establishing the system model and its nominal behavior to derive the contract specifications. Analog fault detection is a key task in many industrial scenarios that include electrical and mechanical components. However, the monitoring of these parameters can be extended to more system components. For example, the user may want to monitor the motor's operating temperature or the machinery's vibration level. To do this, the motor model would be extended to generate data on these properties. New monitors could be implemented with contracts related to these additional features. Moreover, currently, we directly forward measurement signals to the monitor (i.e., select specific signals that we want to observe). However, based on control-oriented state estimation, it is possible to check the valid behavior for signals that are not directly measured or measurable [166, 167]. Threshold monitoring for fault detection is state-of-the-art in control technology [168]. An alternative approach would be to convert neural-network-based FDI rules [169] into the presented TSBC structure. For that purpose, the TSBC-based monitoring approach provides a systematic methodology for monitoring specification. Besides pure threshold classifiers, a precise specification of permissible behavior could also utilize extended specifications, such as considering a gradient within a contract. This enables a finer granularity of monitoring and even earlier detection of potential violations. For example, zooming in on the possible trace for the motor's

current (4.7) shows the actual course of data values. The previously specified TSBC monitors are not capable of specifying the precise progression of values and would allow a much broader scope of values. While this is not an issue per se, a more precise specification of the monitors can significantly improve fault detection. The proposed approach considers a co-simulation

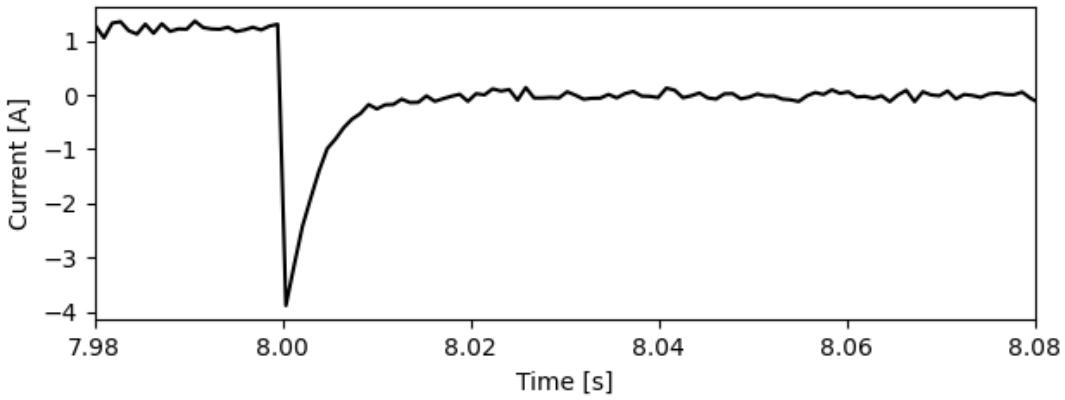


Fig. 4.7: Detdetailed view of a potential current trace adhering to the specification of $Contract\_C_7$.

environment for the hardware components. However, in typical industrial control systems, numerous possible scenarios can be encountered during various control steps, and the effects of faults may also be visible in the software. The more faults that are detected in the life cycle of a control system, the more resources are required to rectify these faults. It is crucial to determine whether a potential monitoring system functions in conjunction with the software's behavior. One approach would be to extend the co-simulation environment with a simulator for the control software's functionality, as proposed in [170], and integrate TSBC monitors to create an additional layer for fault detection.

The framework was developed in simulation, but we plan to deploy it in the real world as a future development. Instead of analyzing the system through data traces produced, the motor will be monitored in real-time. Execution data will be transmitted to the controller via special sensors integrated into the system. Currently, fault detection is the initial task in this workflow. One of the following directions is to extend fault detection to predictive maintenance. The goal is to predict the occurrence of a fault through system monitoring and notify the user in advance. In this way, the system signals when a fault is about to occur, which optimizes maintenance. The goal is to avoid costly stops in the production phase due to breakdowns. Developing a mitigation strategy would enable the controller to automatically determine whether to halt the system due to an emergency or adjust the system configuration and continue the production cycle. In the future, this systematic approach for fault detection could be used at runtime to reduce the need for periodic maintenance schedules and instead establish the basis for predictive maintenance, based on the acquired data. This is an appealing topic and a further aspect that makes our approach even more valuable and exciting. Not only can we highlight that we are in a critical phase with our motor, but we can also adjust to deviations in motor performance in real-time. Also, the user will be notified that maintenance should be considered.

4.2 Extension to co-simulated systems

This section extends the previous one with a novel method that interfaces simulation of control software with detailed physical simulation, both enhanced with injected behavioral faults and contract-based monitors, to evaluate fault tolerance in CPSs. Fig. 4.8 exemplifies a use case in which a control unit regulates the speed and position of a conveyor belt driven by a DC motor: faults can be injected into both the DC motor and the controller, representing the classes of physical and digital faults. Our approach enhances system reliability and robustness by providing a solid foundation for FDI and fault mitigation, such as continuous monitoring and timely corrective actions. It helps developers assess how long a system can maintain acceptable performance under various fault conditions.

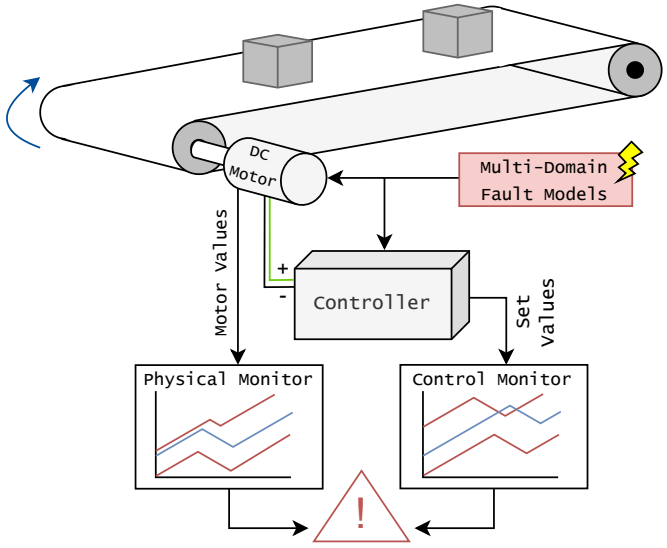


Fig. 4.8: Co-simulation of software and physical components for multi-domain fault detection in CPS.

This is achieved by integrating contract-based monitoring for all subsystems, refining them through fault injection, and by modeling faulty behavior to improve monitor accuracy and robustness. This systematic, fault-driven approach enhances CPSs design, offering deeper insights into system robustness and fault-tolerant strategies. Now the focus is on:

1. **Consistent and comprehensive** assessment, by maintaining a uniform level of detail across control software and physical simulations for a wide range of faults.
2. **Early fault detection** and accurate specification of requirements, with a focus on **single faults** and the extensibility for multiple simultaneous faults.
3. Multiple **systematically designed** contract-based monitors to observe set values and measured values to effectively detect potential faults.

4. **Formal representation** of monitors grounded in assume-guarantee specifications.

Our methodology aims to integrate into both the design and monitoring phases of CPS, promoting a model-based, top-down development approach.

Our work integrates contract-based monitoring across digital and analog subsystems during design and monitoring phases, using fault injection to assess FDI capabilities of the introduced monitors and to iteratively refine the CPS design. Unlike traditional methods, we incorporate faulty behaviors to enhance detection accuracy and robustness, offering a comprehensive strategy for FDI and mitigation in CPS. This approach represents a significant advancement in designing fault-tolerant complex systems.

4.2.1 Hardware & Physical Components Simulation with Fault Injection and Monitoring

This section presents the proposed framework for modeling control software and physical components of a CPS, incorporating fault injection, and specifying contract-based monitors for fault detection during simulation, cf. Fig. 4.9.

The design of a CPS typically involves iterative refinement steps that address the specific requirements of both control software (left) and physical elements (right). The modular nature of contracts enables testing and verification of individual sub-systems (monitors A to E), which can then be combined through hierarchical contracts to form a reliable overall system. Through this process, a specification of the system's nominal behavior is developed. We leverage a contract-based methodology to design monitors that can be simulated alongside both software and physical models, allowing for FDI by integrating monitoring mechanisms within the simulation. Each simulation run enables the evaluation of observed behavior and an analysis of the FDI capability of the monitors. In Fig. 4.9, simulation run #1 exemplifies a simulation with no faults detected, while simulation run #2 shows the activation of three monitors on both the software and the physical parts, as a result of detected faults.

Let us now discuss key concepts related to fault categorization and injection, simulation of software and physical parts' interactions, and the design of contract-based monitors.

Control & Physical Components: Models & Simulation

A CPS is subject to numerous requirements, influencing the choice of model for each component. For the physical part, the dynamics are commonly described with Differential Algebraic Equations (DAEs). On the software side, developers have access to a variety of modeling languages to meet system requirements and ensure functionality. Common choices include SysML for system-level design and specification, as well as standards like IEC 61131 or IEC 61499 [171], widely used for designing and programming industrial automation systems.

Various simulation approaches are essential to validate CPSs models, ensuring they meet performance, safety, and reliability standards [172]. Model-in-the-Loop (MiL) and Software-in-the-Loop (SiL) simulations enable early testing without the need for physical hardware. Hardware-in-the-Loop (HiL) simulations integrate real hardware with simulated environments

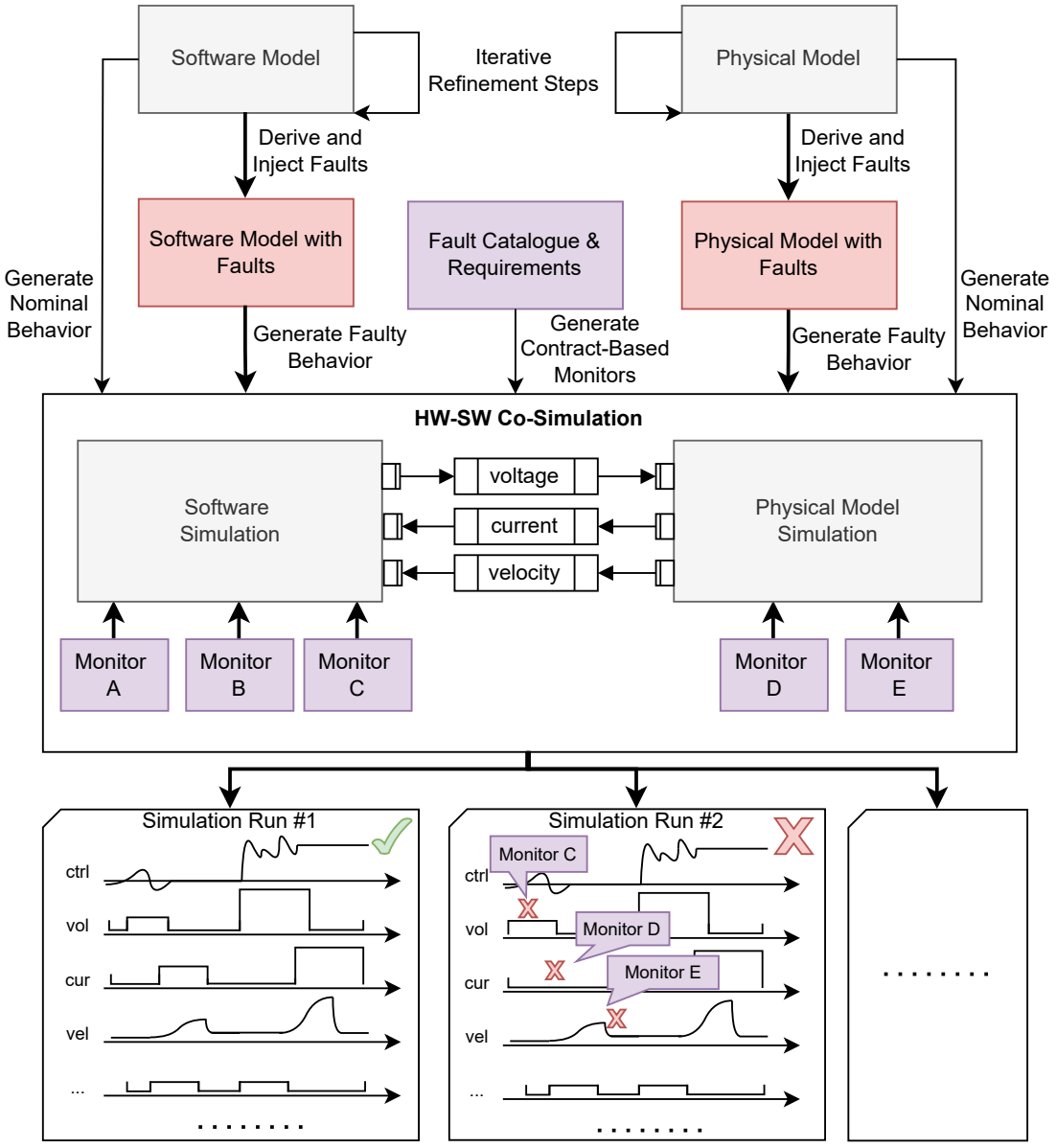


Fig. 4.9: Modeling software and hardware of a CPS with an injection of faults and contract-based monitors for FDI.

to test interactions in real-world scenarios. Co-simulation allows simultaneous testing of physical elements (e.g., circuit dynamics) and software (e.g., control logic) to ensure seamless integration. Combined with formal methods (e.g., model checking and runtime verification), simulation ensures comprehensive validation, reducing risks and improving reliability before deployment.

The methodology presented in this paper is based primarily on system simulation. From the languages outlined in Section 4.1.1, SystemC with its AMS extension is chosen as it covers a wide range of domains in a single environment (e.g., electrical and mechanical components) [2].

These different types of models can be used simultaneously in the same simulation due to the seamless interaction through the underlying SystemC simulation kernel.

Contract-Based Monitors

The underlying methodology for contract-based monitors is Contract-Based Design (CBD) [173], which enhances modularity and reusability. Each system component is described by a contract composed of assumptions and guarantees. The assumptions define expected conditions from the environment, such as specific parameter values, while the guarantees describe the component's behavior if the assumptions hold. This approach enables the early detection of potential design issues and enhances overall system reliability. In combination with fault injection, we propose using contract-based monitors as a means to determine the root cause of an error that may occur during runtime. Thereby, establishing the basis for fault isolation and estimation. Contracts can be expressed in different formal languages, such as MTSL [174] or StSTL [175]. For this work, we chose the concept of TSBCs [151]. In TSBC monitoring, system behavior is defined by time (t) and data (d) variables. A contract is expressed as $t \in [t_1; t_2] \; ? \; d \in [d_1; d_2]$, which checks if a specific behavior occurs within a given time interval. If the time condition is met (assumption), the corresponding data condition must also hold (guarantee). Any deviation from this specification is flagged as a fault. The required parameters for contract-based monitors can be specified during design time or derived at runtime as functions of reference and corresponding control signals.

By synchronizing with system events, TSBC monitors can assess system properties in real-time. If a contract violation occurs, monitors report the fault and determine whether to continue verification (recoverable monitoring) or halt further checks until the next request (unrecoverable monitoring).

Depending on the severity of the fault, adjustments can be made to the simulation model or contract specifications. This could involve refining system behavior through additional simulations, model adjustments, or modifying the contract to more precisely specify permissible behavior. Integrating multiple monitors enhances the precision of FDI but also increases system overhead. This approach is best reserved for critical system components where high monitoring accuracy is essential. In contrast, relaxing contract specifications should be approached with caution, as it may mask potential issues rather than improving system reliability.

4.3 Proof-of-Concept Implementation

This section presents the simulation framework used as a proof of concept, demonstrated through the DC motor example, which is omnipresent in industrial machinery. We introduce the physical model, which incorporates injected faults and contract-based monitors, followed by the control software, its associated fault scenarios, and the corresponding contract-based monitors. In our illustrative scenario, the DC motor serves as the main drive of a conveyor belt. As we can see from the previous sections, the DC motor's behavior can be affected by various faults in both its electrical and mechanical submodels. In the results presented in the following

sections, we restrict ourselves to two faults to show the effectiveness of fault injection: (i) an electrical fault resulting from an increased resistance to of 20.5Ω; (ii) a mechanical fault leading to an increase of the velocity-proportional friction to 50Nms/rad.

To detect faulty behavior, we employ three monitors: one for the electric current in the armature circuit, one for the angular speed of the shaft, and one for the applied voltage. These monitors are based on the previously introduced TSBCs and are specified in Table 4.1, where the motor contracts have the identifier `hw`, while the software contracts are marked by `sw`.

Table 4.1: Overview of example contract specifications for the velocity value observable at the software and physical component.

Contract ID	Contract Specification
C_SW_Vel1	$t \in [0.00; 0.15]\ ?\ d \in [-1.78; 19.54]$
C_HW_Vel1	$t \in [0.00; 0.15]\ ?\ d \in [-1.84; 20.23]$
C_SW_Vel2	$t \in [0.15; 1.1]\ ?\ d \in [11.69; 12.18]$
C_HW_Vel2	$t \in [0.15; 1.1]\ ?\ d \in [11.72; 12.19]$

4.3.1 Control Software Model

Due to the wide use of motors in ICPSs, IEC 61499 was chosen as the modeling language for specifying the control software implementation, cf. Fig. 4.10.

The control software consists of five Function Blocks (FBs) and represents a standard input–process–output model with the FB `E_Cycle` as clock generator to trigger execution every millisecond. The interfaces between software and physical are covered within the FBs `ReadSensor` and `SetMotor` which convert analog to digital signals and vice versa. The `ReadSensor` FB gets the values `velocity`, `current`, and `position` as input for the software model. The output of `SetMotor` provides the input `voltage` to the DC motor. Based on the angular position, `DriveControl` may specify different reference values for the Proportional–Integral–Derivative controller (PID) FB. For simplicity, the desired speed profile is specified as an acceleration to a velocity of 12 rad/s, followed by a slowing down to 3 rad/s after a given time span, before the `velocity` set point is set to zero. The PID is implemented based on [171] and consists of a proportional (`PID_Cal`), integral (`IntegralReal`), and derivative (`DerivativeReal`) part. The PID can be tuned with respect to the following inputs: MODE, a boolean value that controls whether the set point is configured automatically or manually (true = auto or false = manual); ManOut (manual set output); the process value and set point; KP (proportional gain); KI (integration constant 1/sec); TD (derivative time, sec); and lastly, Cycle, used as the discretization step size of the integral and time derivative.

The simulator for the software model is built in SystemC and mimics the execution behavior of 4diac FORTE [176] as a specific implementation of IEC 61499. The control software is created within the open-source 4diac IDE. It is then transformed into SystemC by a model-to-model transformation.

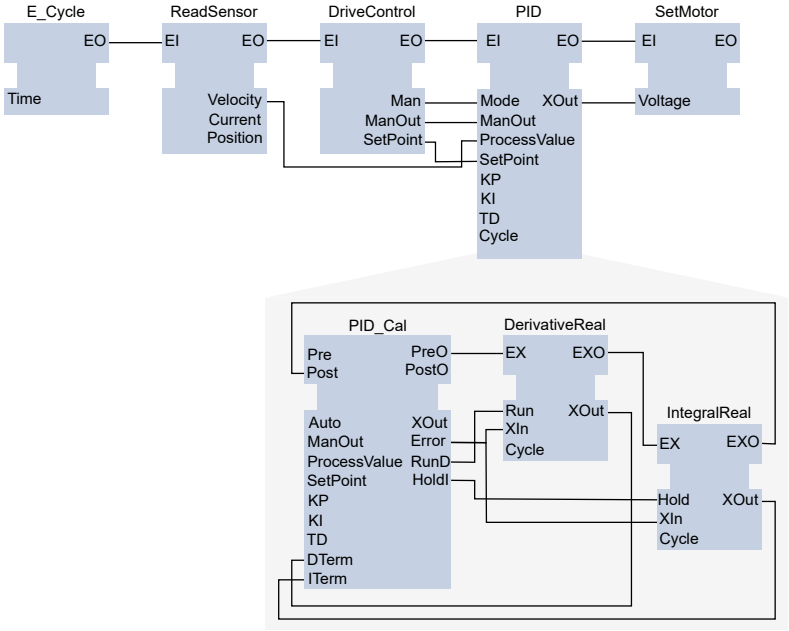


Fig. 4.10: Software model using a PID-controller in IEC 61499.

Two different faults are prepared to be injected into the software component. Initially, the last input value will be used with a probability of 6% instead of the new values for a random period between 1 and 10 cycles of 10 ms. The second one corresponds to a random output perturbation of FB `SetMotor` by $\pm -10\,\%$. While the values of the first fault can be monitored, those of the second one can not.

Similarly to the motor simulation, runtime monitors for the set value in `SetMotor` FB or the measured `velocity` in the `ReadSensor` FB are implemented in the software simulation. These monitors observe contracts based on TSBC. For example, see Table 4.1.

4.3.2 Co-simulation results

Fig. 4.11 summarizes the simulations in four plots, showing the observed voltage, velocity, current, and the integral value of the PID, along with their corresponding monitor values. Voltage, velocity, and current are shown for both the physical (red) and software (blue) sides, while the integral value is only available in the software model. Monitors respond to valid behavior, returning *false* upon detecting violations. Admissible ranges are shaded in light grey for software and dark grey for physical quantities. The plots are divided into five scenarios, each with a 3-second simulation window.

The first scenario illustrates the model's *nominal behavior* using a PID controller with initial parameters set by the developer, potentially based on arbitrary but seemingly adequate choices. White Gaussian noise is added within the `ReadSensor` FB to provide more realistic data. During nominal operation, the system exhibits a stable voltage, velocity, and current, with monitors

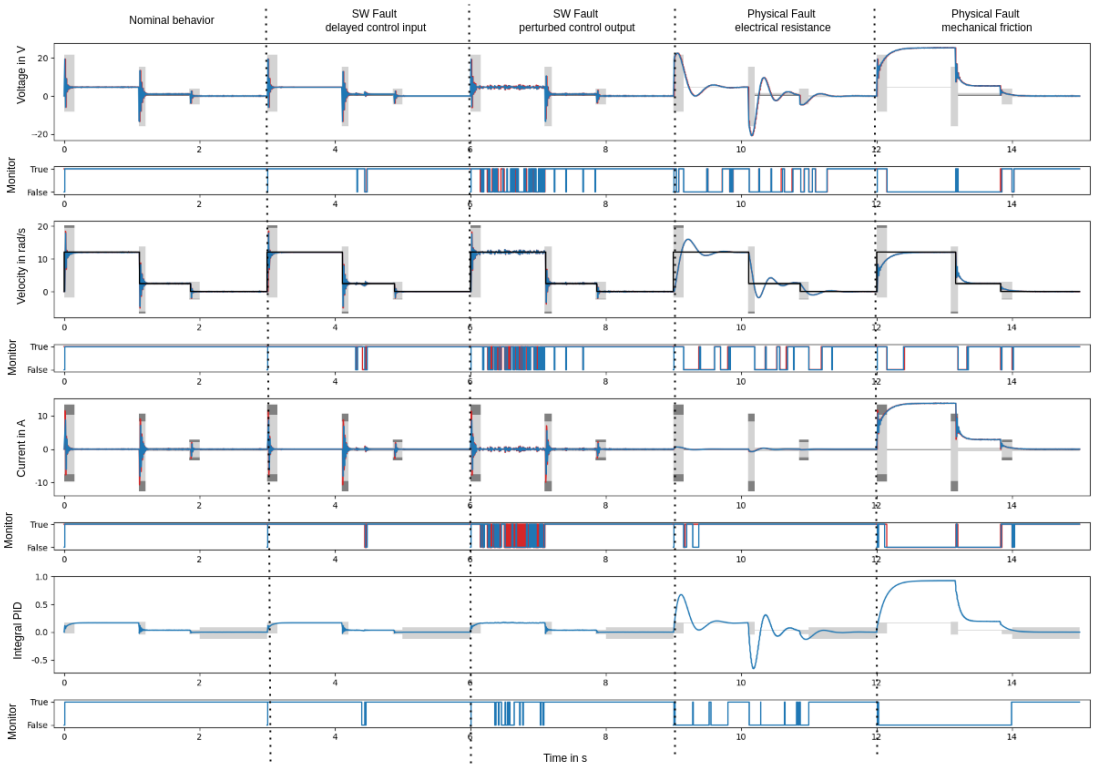


Fig. 4.11: Trace of the physical model (red signals) and SW (blue signals) values and the corresponding monitor outputs. Valid intervals are highlighted within the corresponding plots for software (light grey) and physical side (dark grey).

consistently indicating *true* for all quantities. The PID controller ensures steady-state accuracy. However, variations of the parameters of the DC motor component without re-tuning the PID parameters could lead to inefficiencies. To mitigate this risk, well-known automatic PID tuning mechanisms could be integrated in the control block.

In the *first software fault scenario*, a delayed control input disrupts the system, causing transient deviations in voltage and velocity, which the monitors flag as faults. At first sight, the current appears unaffected. However, this statement is only true as long as the introduced delays are not shorter than the system's time constants, which otherwise serve as a natural low-pass filter (at least partially) suppressing this effect. The *second software fault*, involving a perturbed control output, has a more significant influence. Voltage, velocity, and current exhibit fluctuations, and the monitors detect continuous violations. The PID controller's integral value also increases, showing the system's inability to maintain proper control.

Next, with the introduction of a *physical fault affecting the electrical resistance*, the system experiences oscillations and loses its rapid settling behavior. The monitoring system repeatedly detects abnormal behavior in the voltage, velocity, and current. However, the current is way smaller than expected, which goes unnoticed based on the monitor specification. This could be solved by specifying a more precise monitor. The PID controller shows a flagged deviation from its expected trajectory, reflecting the increased effort of maintaining control. In the final

phase, a *mechanical friction fault* affects the system's velocity and current. As friction increases, velocity drops, and current spikes to compensate for the load. The monitors accurately detect these faults while voltage remains stable, highlighting the need for multi-domain monitoring.

The monitoring system efficiently detects both software and physical faults with precise timing. Software faults with an intermittent behavior cause transient issues, while physical faults lead to more extended disruptions. Multi-parameter monitoring is crucial for reliable FDI, especially as control algorithms may mask physical issues. Note that scalability may become a concern as the complexity of control algorithms and the number of hardware components increase. Automating the generation of monitors from contract specifications can facilitate integration. Our approach enables developers to efficiently and reliably place monitors to detect a variety of potential faults.

4.4 Accelerating battery pack design via modular electro-thermal simulation frameworks

Among the critical challenges in battery pack design, thermal management plays a crucial role in ensuring operational safety, extending battery lifetime, and maintaining performance under varying load conditions [177]. Inadequate heat dissipation can lead to uneven temperature distribution, accelerated aging, and, in extreme cases, thermal runaway events, thus making accurate thermal modeling and design exploration an essential step in the design process [178].

Existing modeling approaches exhibit a trade-off between computational efficiency and physical fidelity. Lumped models enable fast simulation but fail to capture cell-level interactions and localized thermal behavior. Conversely, high-fidelity computational fluid dynamics (CFD) models offer detailed temperature fields but incur prohibitively high computational costs, making them unsuitable for early-stage exploration or evaluating many design choices [179]. This limitation is further compounded by the influence of the cooling system, whose geometry, coolant type, and flow strategy strongly affect safety, energy efficiency, and total parasitic losses [180]. Early-stage assessment of cooling strategies is therefore fundamental for selecting optimal performance and cost-effective designs.

To address these limitations, this work proposes an *automated electro-thermal design-space exploration framework* for battery packs. The tool targets the design of next-generation automotive and heavy-duty mobility systems, where the ability to evaluate electrical performance and thermal behavior under diverse operating conditions is critical for ensuring reliability and enabling system-level optimization.

The contributions of this work are summarized as follows:

- A modular and compositional framework that supports the construction of battery packs of arbitrary size, where each cell is modeled individually with coupled electrical and thermal dynamics.
- Automatic construction of interconnections using series-of-parallel (SOP) and parallel-of-series (POS) configurations, following common automotive design practices [181].

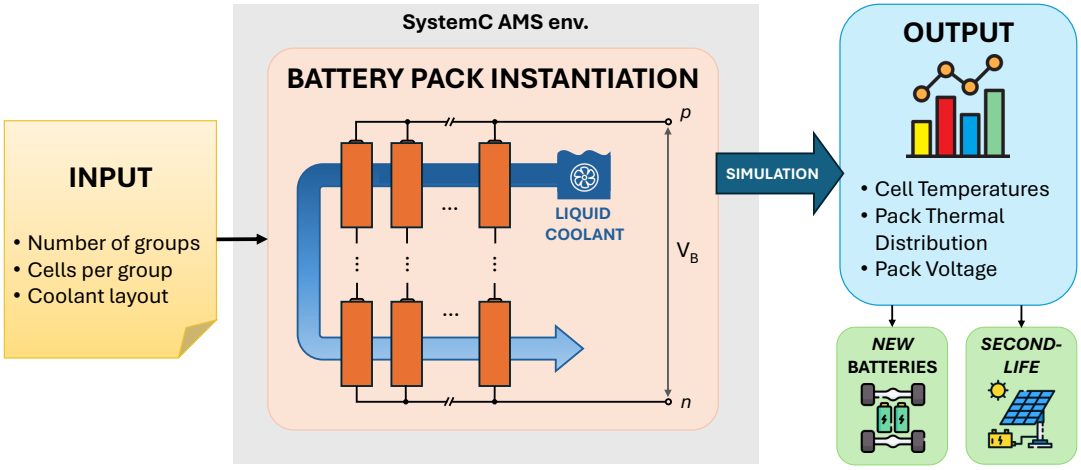


Fig. 4.12: Overview of the automatic compositional battery-pack framework with coupled electro-thermal cell models and configurable liquid cooling.

- Generation of an equivalent thermal network that captures conduction between cells and convection to the coolant.
- A configurable liquid-cooling subsystem supporting multiple geometric layouts and coolant types, enabling early optimization of thermal management strategies.

An overview of the proposed framework is shown in Fig. 4.12. By combining flexibility, scalability, and simulation efficiency, the approach bridges the gap between detailed physical modeling and fast system-level evaluation. The implementation leverages SystemC AMS [40], which enables modular construction and efficient support for multi-domain models, and to foresee the integration of battery simulation with other domains, including mechanics and digital control [182].

4.4.1 Battery pack simulation and thermal management

Thermal behavior in battery packs has been extensively studied due to its impact on safety, performance, and cycle life. Existing modeling approaches span a wide spectrum of fidelity and computational cost.

Commercial CAE Tools (e.g., MATLAB/Simscape Battery, GT-SUITE/AutoLion, Thermo-Analytics) offer integrated electro-thermal battery models and cooling system simulation. While powerful, these tools typically rely on lumped or semi-distributed thermal representations, require manual setup, and involve high licensing costs. This hinders rapid, automated design-space exploration during early development.

High-Fidelity CFD approaches (ANSYS Fluent, COMSOL, STAR-CCM+) provide detailed representations of heat transfer and fluid flow, making them suitable for late-stage validation. However, their computational cost remains prohibitive for evaluating large automotive-scale packs or exploring multiple cooling strategies [179].

Low-order thermal models approximate the pack using coarse thermal nodes [183, 184]. Their low runtime makes them appealing for control and system-level studies, but they lack cell-level granularity and do not accurately capture localized hot spots, conduction paths, or cooling non-uniformities. As a result, they are unsuitable for assessing modern cooling architectures or heterogeneous pack configurations.

Recent research has explored hybrid and modular modeling approaches that integrate electrical and thermal dynamics at the cell level [185, 186]. While these methods represent a meaningful step forward, they rely on fixed cell arrangements or predefined module structures, offer limited support for modeling liquid-cooling layouts or selecting different coolant types, and lack automation for generating series/parallel electrical configurations. Moreover, few provide a unified co-simulation environment capable of natively coupling digital control with continuous-time electro-thermal physics, thus restricting their applicability for large-scale, automotive-grade design exploration.

Positioning of This Work

Although recent advances have introduced hybrid or modular battery modeling approaches, existing solutions still lack a unified methodology capable of supporting early-stage, system-level electro-thermal design exploration for large automotive-scale packs. Current tools either prioritize fidelity at the expense of runtime or achieve fast simulation by relying on coarse abstractions that limit physical accuracy and design flexibility. Moreover, integrated evaluation of electrical behavior, thermal interactions, and cooling strategies remains fragmented across separate modeling environments, complicating consistent system-level analysis.

The present work aims to bridge this gap by providing a coherent, simulation-oriented perspective that enables physically grounded yet computationally efficient exploration of battery-pack architectures and thermal management strategies within a single modeling framework. Additionally, the choice of the versatile language SystemC AMS enables foresight into integrated simulations that cover multiple aspects of a vehicle in a single run, in conjunction with digital control logic, as mandated by current Software-Defined Vehicle trends [187, 188].

4.4.2 Proposed Thermal Model

This article presents a framework that automatically generates a simulatable model of a battery pack exploiting the capabilities of the SystemC AMS language and simulation core. The overall objective is to enable fast design-space exploration with cell-level fidelity, allowing layout choices (e.g., coolant topology, fluid type, pipe routing) and operating scenarios to be evaluated early and iteratively. This philosophy aligns with the literature's call for models that bridge the gap between detailed physics and system-level speed by coupling electrical and thermal dynamics at the cell level [189].

Battery cell model

The single electric cell is modeled using an equivalent circuit that mimics its behavior [181, 190]. Each cell is modeled by two coupled submodels: an *equivalent electrical circuit*, including

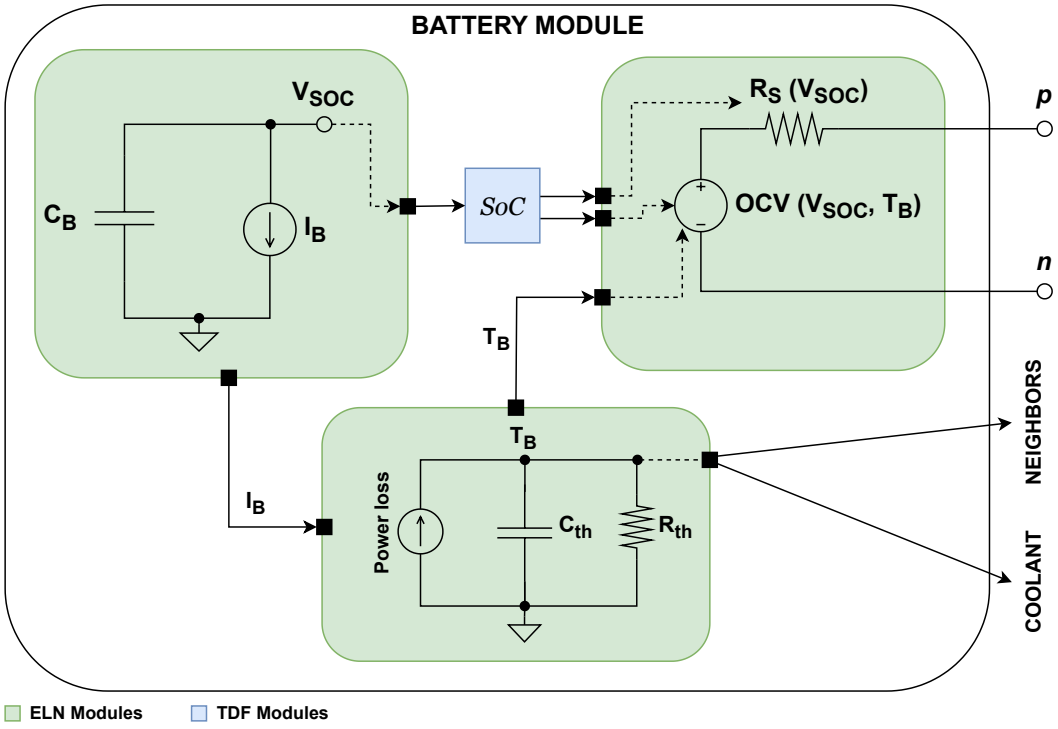


Fig. 4.13: Schematic of the single battery cell model used by the framework. The green color represents the ELN MoC modules, while the blue color represents the TDF MoC modules.

a Coulomb counting stage, and a *thermal RC network*. These two modules are cross-coupled through temperature-dependent open-circuit voltage (OCV) and internal resistance, as well as the conversion of electrical losses into heat. This architecture follows established practices in the electrothermal modeling of lithium-ion cells, allowing for computational efficiency while maintaining dependencies on the state of charge (SoC) and temperature [191, 192].

The structure of the single battery model is shown in Fig. 4.13 (presented in clockwise order). In the top-left corner, we can find the section of the model that implements the SoC evolution via the well-known Coulomb counting technique, which is a standard, low-overhead method to propagate SoC in equivalent-circuit battery models. [193]. In particular:

$$SoC = SoC_0 + \frac{1}{C_{nom}} \int_{t_0}^{\tau} I_B(t)\, dt \tag{4.1}$$

where C_{nom} is the nominal capacity of the battery, and I_B is the current flowing through the battery. The SoC is manipulated through polynomial transformations in the TDF module (Fig. 4.13, blue) at the top center to calculate the open-circuit voltage (OCV) and the value of the internal battery resistance R_s. Then, the module on the top-left of Fig. 4.13 implements the relation:

$$V_B = OCV(V_{SoC}, T_B) - I_B R_s(V_{SoC}) \tag{4.2}$$

where V_B is the output voltage of the battery, measured between p and n nodes. The relation highlights how OCV captures the equilibrium voltage versus SoC and battery temperature, while Rs represents the instantaneous ohmic drop, based on the SoC. Such equivalent-circuit approaches are well-documented and commonly compared with higher-fidelity electrochemical or CFD-coupled models as a speed-accuracy compromise.

Finally, at the bottom of Fig. 4.13, we find the equivalent thermal circuit. Exploiting the electrical–thermal analogy [194], each cell's temperature node T_B is governed by:

$$C_{th}\frac{dT_B}{dt} = I_B^2 R_S - \frac{T - T_{cool}}{R_{th,cool}} - \sum_{j=0}^{n} \frac{T_B - T_j}{R_{th,j}} \tag{4.3}$$

where C_{th} is the cell heat capacity (thermal capacitor), $I_B^2 R_S$ is the power loss by the battery, converted as a source of dissipated heat, $R_{th,cool}$ captures heat exchange to the coolant, and $R_{th,j}$ are thermal resistances to neighboring cells.

Coolant model

To capture fluid–solid heat exchange at high speed, the coolant loop is modeled using a compact, network-based methodology inspired by [189]. We adopt the same principles of multi-port cavities/channels, represented by reduced-order elements and a coupled solid–fluid RC formulation, to emulate embedded channels in battery packs.

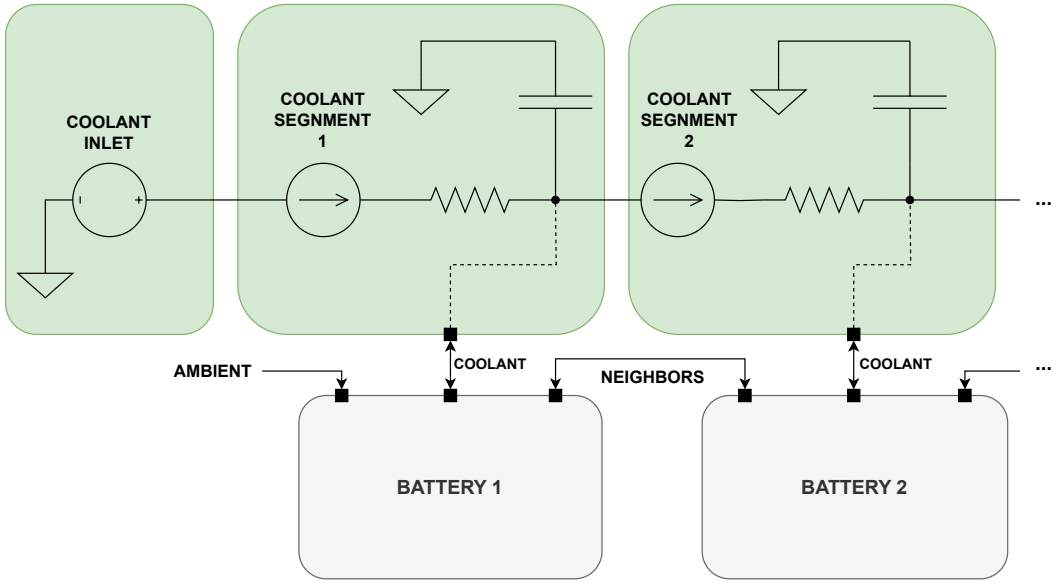


Fig. 4.14: Schematic of how the coolant flows through the battery modules (of Fig. 4.13). The current sources in each coolant segment are controlled by the voltage (or temperature) difference between the current coolant segment and the previous one.

The proposed coolant concept is shown in Fig. 4.14. The coolant fluid circulates through channels in thermal contact with the cells, allowing it to absorb the heat generated during

charge/discharge phases. To enable its simulation, the physical coolant is conceptually split into segments, one acting as a source and the others simulating the coolant path.

Following the equivalence between the thermal and electrical domains, the source is a voltage source that sets the input temperature of the coolant fluid. Starting from this inlet source, the segment modules are connected in series one after the other. A coolant segment is the section of coolant connected to the battery cells, and more than one cell can be connected to it, depending on the coolant topology. The module simulating one segment is thus composed of:

- a capacitor, representing the amount of heat that can be dissipated/absorbed by that coolant section, given by parameters such as the volumetric heat capacity of the fluid;
- a current source, representing the flow of the coolant, controlled by the volumetric heat capacity of the fluid, the average velocity of the fluid through the cell in the direction of flow, and the temperature difference between its coolant segment and the previous one [189];
- a resistor connecting the segment to the next one, calculated with the heat transfer coefficient through the fluid and the dimensions of the cooling channel;
- the temperature of each coolant segment is measured across its capacitor.

Coolant topology

Different liquid cooling topologies have been developed in order to balance efficiency, cost, manufacturability, and safety [195]. The choice of topology is strongly influenced by the cell format (e.g., cylindrical, prismatic, pouch), the required thermal uniformity, and the available packaging space.

In this paper, we focus on four layouts (Fig. 4.15), selected to show different characteristics in terms of cell refreshing performance and energy cost to power the cooling system:

- “S” topology: one serpentine tube between rows of cylindrical cells; this allows close contact with multiple rows of cells, while maintaining a lightweight construction;
- “C” topology: coolant flows around the module or through walls surrounding the cells, offering a simpler structural integration, though with lower direct efficiency;
- “SS” topology: the coolant flows through the cells with the tubes following a “S”-shaped path, each slithering around a portion of the battery rows;
- “E" topology: separates the coolant in parallel channels, to better handle large-scale battery packs. The multiple, thinner tubes require less cooling power, as they only touch a portion of the total cells.

Tool automatic workflow

An automatic tool implements a fully automated pipeline that constructs the electrical and thermal models in SystemC AMS from a compact design specification.

Design Specification

the user provides crucial data for the simulation: type of electrical structural connection (i.e, SOP or POS), coolant characteristics (layout, inlet flow rate, and specific heat), and the input

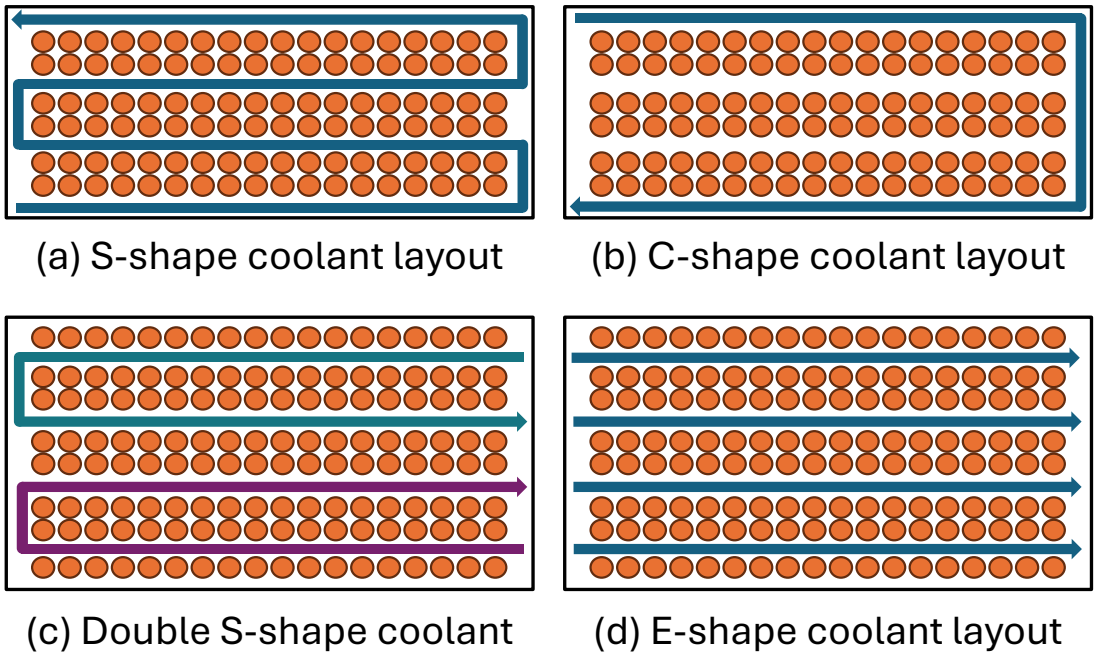


Fig. 4.15: Coolant topologies: arrows represent coolant flow in coolant channels, while orange circles represent cells.

current for the battery pack. Fig. 4.16 shows an example of the input file, configured to simulate 12 Panasonic NCR18650 cells organized in 4 groups of 3 cells each, with the specified position in the space (reflecting the electrical organization) and an “S" shaped cooling topology. The resulting topology is shown on the right-hand side, where orange represents cells, and blue represents coolant. Each square contains the ID of the element (i.e., B for cell number, C for coolant segment number), representing the position of each element in the generated model.

Cell Power Model

the tool creates the number of battery cell models requested. The model is built starting from user-provided information (Fig. 4.16) and a current-voltage plot provided in the datasheet [190].

Electrical Connection

battery cells are now connected, using SOP or POS paradigm. Both connection paradigms allow us to obtain a fully connected electrical model of the battery pack.

Battery Topology

in modern battery configurations, groups of cells may be electrically connected, e.g., in parallel, but are often arranged across different rows, primarily for spacing and cooling efficiency [196]. Physical proximity of cells is specified with a table-like notation. The assumption is that:

- the electrical connection reflects the numerical order, e.g., cells from 1 to the number of cells in a group are part of the same group (in Fig. 4.16, cells 1 to 3). Therefore, the connection does not need to be explicitly defined;

LAYOUT 3x4 (TOT CELLS 12) "S" SHAPE COOLANT

```
CELL NOM VOLTAGE = 3.6
CELL CAPACITY = 3350

ELECTRICAL CONNECTION = SOP
NUM GROUPS = 4
CELLS PER GRP = 3

LAYOUT = {{1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}}

NUM COOLANT SGM "S" = 9
COOLANT FLOW RATE = 0.033125
COOLANT SPECIFIC HEAT = 3400
```

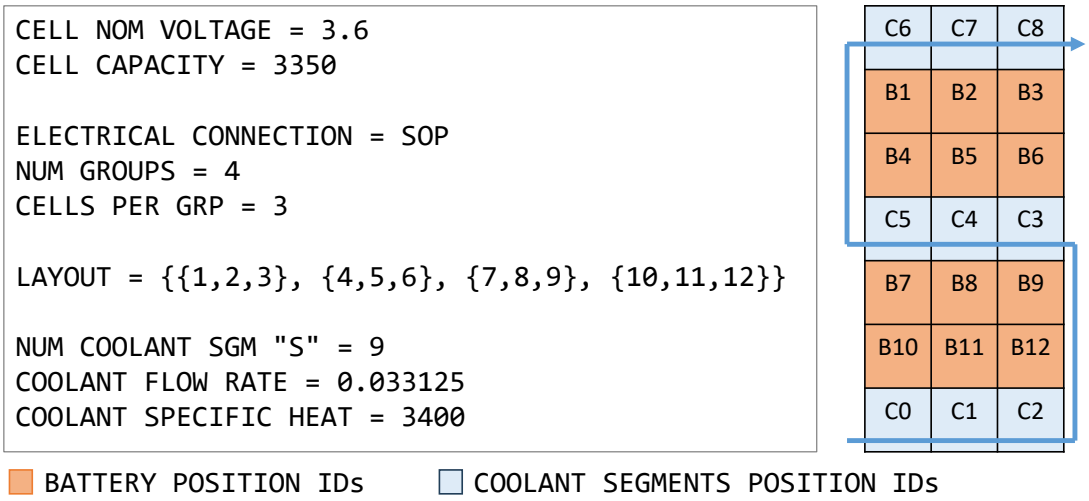


Fig. 4.16: Input provided by the user to construct a 3×4 topology of Panasonic-NCR18650 cells with a “S" shape coolant.

- a table-like notation is used to specify space proximity: brackets wrap the IDs of cells positioned on the same row. E.g., if the user specifies `{{1,2,4},{3,5,6}}` this implies that the first (electrical) group includes batteries with IDs 1, 2 and 3, but that the first row (for thermal and proximity considerations) includes IDs 1, 2 and 4.

Thermal Network Creation

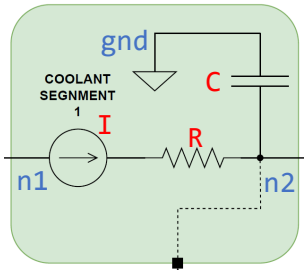
here, the thermal connection between adjacent cells is made. The cells are connected in a square pattern (above, below, right, left) according to the pack layout provided in the previous step.

Coolant Creation and Connection

the coolant is first created using the parameters provided by the user. Different types of coolant require different lengths and power levels, also based on the number of cells. Once created, each segment of the coolant is connected to its corresponding nodes, concluding the battery pack model creation.

Model Simulation

The model is now generated with SystemC AMS primitives, the power is applied to the battery pack terminals, and the behavioral simulation starts. The snippet in Fig. 4.17 allows us to highlight that SystemC AMS code implementation is quite straightforward, as it consists of a 1-1 mapping from network elements (e.g., a resistor) onto SystemC AMS primitives (e.g., `sca_r`). Most of the complexity of the proposed framework lies in the automatic generation of the model (steps *b-f*) given the initial user specifications (*a*), allowing for an easy setup with no manual intervention.




```
sca_eln::sca_node n1, n2;
sca_eln::sca_node_ref gnd;
sca_eln::sca_r R;
sca_eln::sca_c C;
sca_eln::sca_tdf::sca_isource I;

I.p(temp); I.n(n1); I.inp(heat_flow);
R.p(temp); R.n(n1);
C.p(n1);   C.n(gnd);
```

Fig. 4.17: Excerpt of the SystemC AMS code for the simulation of a coolant segment, where the resistor is mapped onto a `sca_r`, the capacitor onto a `sca_c`, and the current source onto a `sca_isource` instance.

4.4.3 Battery pack experimental results

For the experimental application, we focused our analysis on four cell configurations of increasing complexity to examine the impact of coolant and its resulting simulation performance. We decided to use a single cell for all experiments, in the interest of readability. The choice fell on the Panasonic NCR18650B cell (see Table 4.3) [197], which is the cell type used for the Tesla Model S. This choice stems from the fact that the Tesla Model S is one of the most documented in literature and on the web, making parameterization less complex [198]. Furthermore, two of the four configurations presented in this example originate directly from the automotive industry, illustrating the practical application of the proposed tool.

Table 4.2: Simulation time for the 8×5 configuration with different coolant shapes when increasing simulated time.

Simulated time (s)	Simulation time (s)			
	"S" shape	"C" shape	"SS" shape	"E" shape
250	0.861	0.696	0.846	0.844
500	1.818	1.579	1.816	1.652
750	2.699	2.542	2.678	2.686
1,000	4.027	3.763	3.611	2.992
1,250	6.360	6.375	6.325	4.494

Table 4.5 reports the simulation results for four different sizes: 4×3 (used as illustrative example), 8×5, 74×6, and 86×6. The latter two are inspired by the Tesla Model S battery packs (P85 and P100D) [199]. For all configurations, we simulated all four coolant shapes as part of a design space exploration to compare the different thermal distributions. Experiments were conducted with a fixed time step of 0.5 seconds to standardize the measurement. Simulations were performed on a Ubuntu 24 virtual machine with 3 cores at 3GHz and 8GB RAM.

Table 4.3: Specifications of Panasonic NCR18650B cell.

Parameter	Value
Format	18650
Nominal voltage	3.6 V
Capacity	3350 mAh
Charge voltage	4.20 V
Cutoff voltage	2.50 V
Diameter	18.5 mm
Height	65.0 mm
Mass	48.5 g

4×3 configuration

The first configuration corresponds to the specifications in Fig. 4.16: a 4×3 configuration, comprising 4 groups of 3 cells. The simulation compares the evolution with four different coolant shapes, while keeping all other parameters constant. Table 4.5 highlights that, despite the same number of battery cells, the simulated SystemC AMS versions are different, as a result of different coolant topologies that require a varying number of electrical primitives. All such configurations were generated almost instantaneously by varying one line of the input configuration file (`NUM COOLANT SGM` in Fig. 4.16, which indicates the coolant topology and the number of coolant segments to be generated, thereby tuning the accuracy of the coolant simulation). The different configurations have an impact on the average cell temperature, with the “E" topology achieving a more effective cooling effect.

Fig. 4.18 depicts the resulting evolution as thermal maps of the temperature of cells and coolant segments at different time steps. Both simulations start with ambient temperature (18°C). While providing the requested 32A current, the cell temperature increases unevenly. In the "C" shape configuration, cells closest to the coolant maintain a lower temperature, while cells in the center and on the left are hotter, with a ≈2°C difference. In the case of the "S" shape configuration, cells tend to have an overall lower temperature due to the coolant path that separates cell groups. This is also highlighted by the average cell temperature after 730 seconds of simulated time, which is 0.6°C lower for the "S" shape (26.5°C vs. 27.1°C).

8×5 configuration

The 8×5 configuration features a similar setup to the former experiment and is used to comment on the scalability of the simulation. To this extent, we simulated the SystemC AMS code for a variable length of simulated time, from 250s to 1250s, and reported the results in Table 4.2. The numbers show that the amount of time spent on the simulation is proportional to the simulated time, with a slight increase in the case of a complete battery pack discharge simulation (1,250s). Such measurements demonstrate that the initial instantiation cost of creating and connecting the battery pack (constant for all simulations shown in the graph) does not significantly impact the

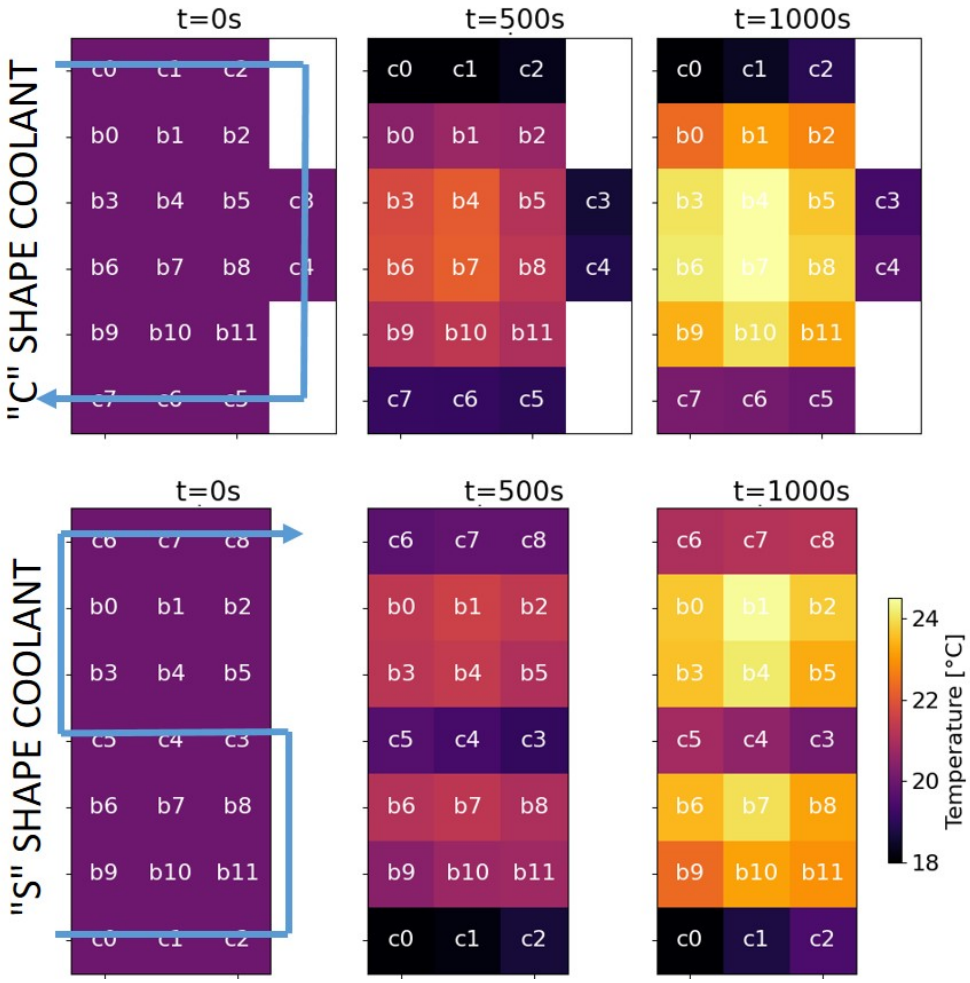


Fig. 4.18: Thermal map for the 4×3 configuration with "C" shape (top) and "S" shape (bottom) coolant. The maps show the thermal distribution at the beginning of the simulation, after 500s and 1,000s of simulated time, to show the temperature evolution of cells and coolant segments. The blue arrows highlight coolant flow.

simulation time. Additionally, the simulation time is significantly lower than the simulated time, as a 20-minute simulated time can be easily evaluated with less than 7 seconds per configuration.

Tesla Model S inspired 74×6 configuration

To demonstrate the capabilities of the proposed framework, we modeled a battery pack equivalent to a module of the Tesla Model S P85, featuring a 74×6 configuration of cylindrical cells for an overall 85kWh storage. The parameters of the single battery cell are reported in Table 4.3, while the specifications of this Tesla Model S module are described in Table 4.4. The original battery pack design incorporates a liquid cooling system featuring a serpentine aluminum tube that runs between the rows of cells (i.e., the “S" topology coolant layout), ensuring efficient heat transfer and uniform temperature distribution throughout the module. In this experiment, we compare the original “S" shape coolant with the other shapes.

Table 4.5 shows that the simulated SystemC system is significantly more complex than previous versions, with 8,800 to 9,800 primitives instantiated to simulate both battery cells and coolant segments. However, the code generation still allows for handling such complex systems, and to achieve fast simulation (on average, 105 seconds of runtime for a 16.5× longer simulated time). With these larger experiments, it becomes clear that the number of SystemC AMS primitives significantly impacts simulation time, as more primitives result in more equations that need to be solved at runtime. Additionally, each coolant topology achieves a different average cell temperature, with a delta of almost 3°C. This analysis may be useful to consider replacing the “S" topology coolant with either the “SS" or “E" configurations, which achieve better average cooling power.

Table 4.4: Tesla Model S battery module specifications (74p6s).

Parameter	Value
Configuration	74p6s (74P × 6S)
Cells	444 (18650, NCR18650B)
Chemistry	Li-ion, NCA
Nominal voltage	22.8 V
Max voltage	25.2 V
Capacity	~232 Ah
Energy	~5.3 kWh
Cont. discharge	225–500 A
Peak (10 s)	750 A
Dimensions	685×300×75 mm
Weight	25–26 kg
Cooling	Liquid, serpentine tube
Coolant flow rate	0.033125 kg/s
Coolant specific heat	3400 J/kg×°C

Fig. 4.19.a-b represents the thermal distribution as heatmaps for the original “S" shape coolant (Fig. 4.19.a) and for the “C" shape coolant (Fig. 4.19.b), to visually show the differences in terms of temperature distribution. The heat distribution pattern reveals significant differences already, in the middle of the simulation. The “S" shape coolant (Fig. 4.19.a) shows a heterogeneous temperature distribution. As the coolant flows away from the inlet, its temperature increases due to the heat generated by the cells, which are operating to provide the desired current. This increases the coolant temperature of ≈2°C at the end of the coolant path, but allows for a lower average battery cell temperature. Additionally, the battery cell temperature increases as the serpentine path is followed, as the coolant becomes less efficient at cooling the battery cells.

The “C" shape coolant (Fig. 4.19.b) has a more regular thermal distribution. The coolant is heated marginally (≈1°C), and cools only the neighboring cells following the segment shape. The hottest cells are those in the middle, creating a hot spot in the central region that could accelerate aging and compromise safety margins.

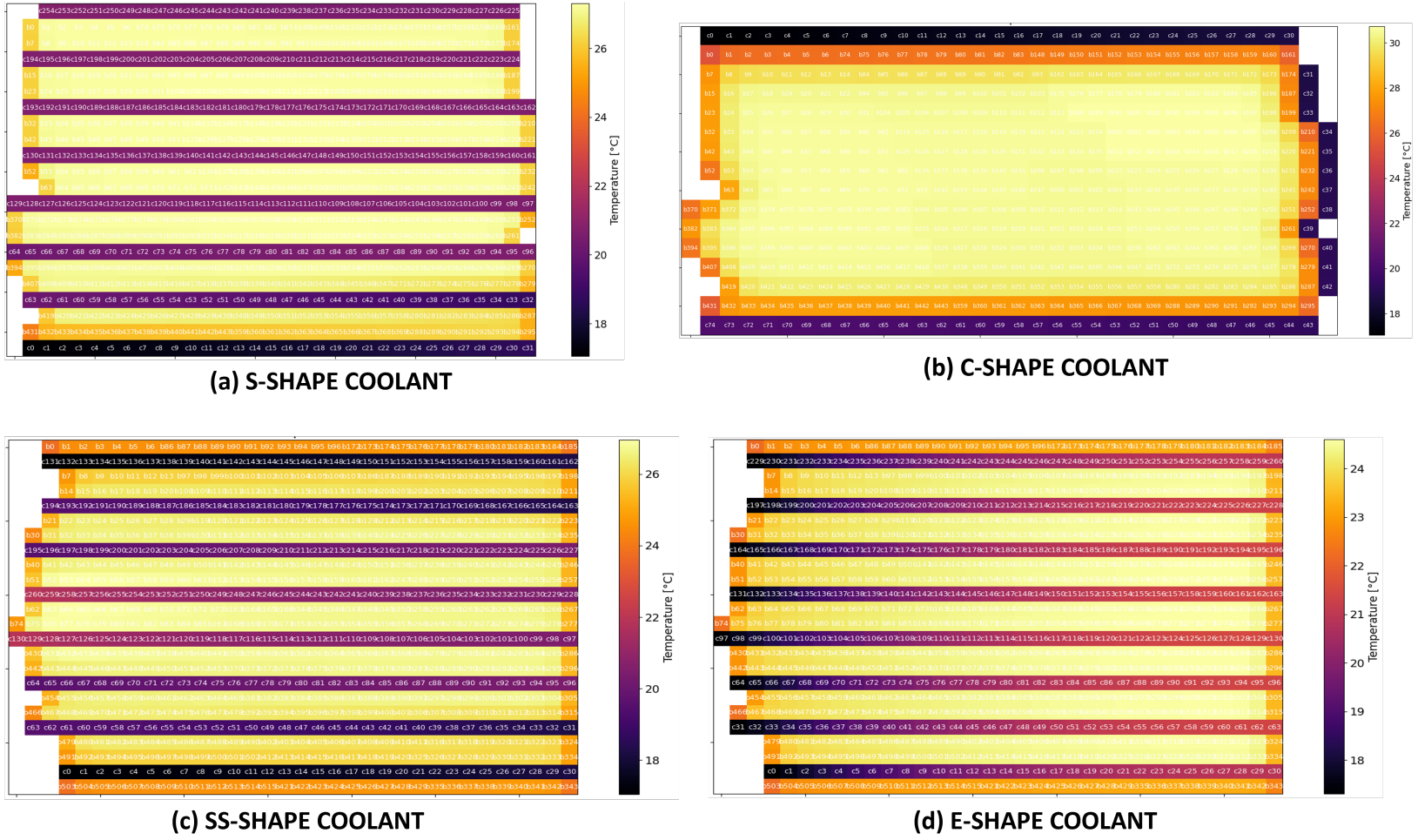


Fig. 4.19: Thermal map of some of the configurations of Table 4.5: the top reports the thermal distribution for the 74×6 configuration with “S" shape coolant (a) and “C" shape coolant (b); the bottom reports the thermal distribution of the 86×6 configuration with “SS" shape coolant (c) and “E" shape coolant (d).

This experiment demonstrates, through a rapid simulation (overall 3.5 minutes), that the “S" shape coolant is more effective in cooling the cells and preserving their operation, despite the higher technical complexity implied by increased pumping power and potentially more robust manifolds.

Tesla Model S inspired 86×6 configuration

As the last experiment, we simulate a battery pack module inspired by the Tesla Model S P100D. This module is an update to the P85 model simulated in the former section, obtained by are dividing the large “S" shape cooling tube into two (i.e., obtaining a “Double-S" shape). This reduces the tube diameter and creates space for two additional rows of cells, without modifying the case. We recreated this scenario, highlighting the improvements over the previous module.

The “SS" topology was thus generated by changing the initial configuration of Fig. 4.19.a. The new thermal map Fig. 4.19.c clearly highlights the different heat distribution: the sides of the pack are colder, due to the direction of the two coolant flows, and average cell temperature is lower, despite of the higher number of cells (i.e., 26.1°C vs. 28.9°C).

To further leverage the modeling flexibility of our tool, we generated the same battery pack with an “E" shape coolant layout. Fig. 4.19.d shows how the average temperature is lower (24.2°C), but highlights also that the heat gradient shifts to one side of the battery, due to the natural effect of the fluid flowing in a single direction. This characteristic may be taken into account in the design exploration, and determine its future position within the target EV.

Table 4.5: Various battery pack layouts simulation results.

Pack configuration	Simulated time (s)	Discharge current (A)	Coolant shape	Generation time (s)	Simulation time (ms)	SystemC AMS modules	Avg. cell temperature (°C)
3×4 (12 cells)	730	32	S	10.804	1.078	279	26.5
			C	11.058	1.243	274	27.1
			SS	6.081	1.292	264	25.1
			E	6.888	1.976	266	24.8
8×5 (40 cells)	1,220	48	S	17.607	7.069	891	25.1
			C	14.759	6.344	846	27.2
			SS	14.161	6.182	868	24.7
			E	15.168	5.228	872	24.5
74×6 (444 cells)	1,740	500	S	185.083	116.866	9,722	28.9
			C	205.129	78.107	8,827	29.8
			SS	183.596	115.513	9,564	27.8
			E	187.52	110.208	9,574	27.0
86×6 (516 cells)	1,910	550	S	203.772	156.259	11,265	27.2
			C	180.674	130.017	10,195	28.8
			SS	217.142	160.670	11,117	26.1
			E	226.303	154.232	11,129	24.2

Performance considerations

As a final analysis, Fig. 4.20 reports the simulation time and the average pack temperature of the 74×6 (blue) and the 86×6 (green) configurations. We also report the same statistics for two CFD-powered tools [198, 200] applied to the 74×6 configuration with “S" shape coolant (gray). Even if it was not possible to reproduce exactly the same experiment, the figure highlights that the simulation time of our framework is up to one order of magnitude faster than the CFD tools, while still exposing temperatures in the same range. More extensive comparisons will be performed as future work.

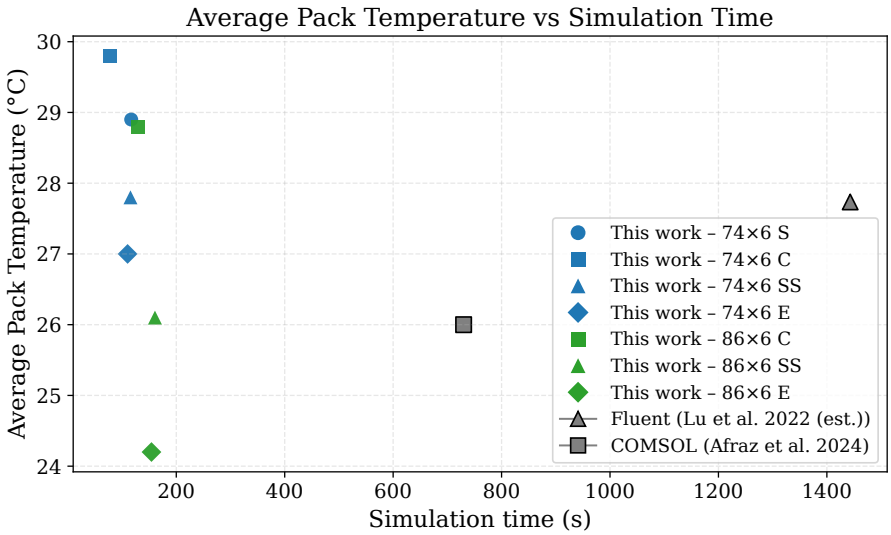


Fig. 4.20: Comparison between the final average temperature and the simulation times between the proposed tool and two literature CFD tools [198, 200].

Overall, these results highlight how different coolant solutions can significantly alter the operation of the battery pack, and that the proposed automatic tool may aid in the design phase by facilitating “what if" analysis and exploration of alternative configurations. The proposed framework supports the study of optimizing design spaces in terms of thermal safety, energy efficiency, and cost, thereby enabling designers to quantify quality-cost trade-offs early in the design process.

4.5 Smart systems robustness analysis

Mixed-signal systems are at the heart of the Industry 4.0 phenomenon, as they represent a significant portion of the intelligent systems employed in various industrial fields, such as automotive, avionics, aerospace, and Internet of Things (IoT). These systems, also called *smart systems*, are often characterized by a control part, and a sensing and actuation part, implemented through digital and analog components [201]. The reliability of such systems is vital, as they often need to react to unexpected or critical events within a precise time window and with an accurate set

of actions [202]. However, smart systems are typically designed by combining individual components that have been individually designed. Although the individual components are designed to ensure a certain level of safety, in most cases, this design approach does not comprehensively preserve the safety guarantees [139]. In particular, the interaction between components can cause several problems at the level of functionality: the effect of a fault in one component (e.g., caused by aging or deterioration) can propagate to other components, thereby damaging them or, more simply, affecting their proper functionality. Such incidents can cause various issues, depending on the specific application context, such as damage to other components, monetary loss, or safety concerns for the involved users. The current state of the art encompasses studies

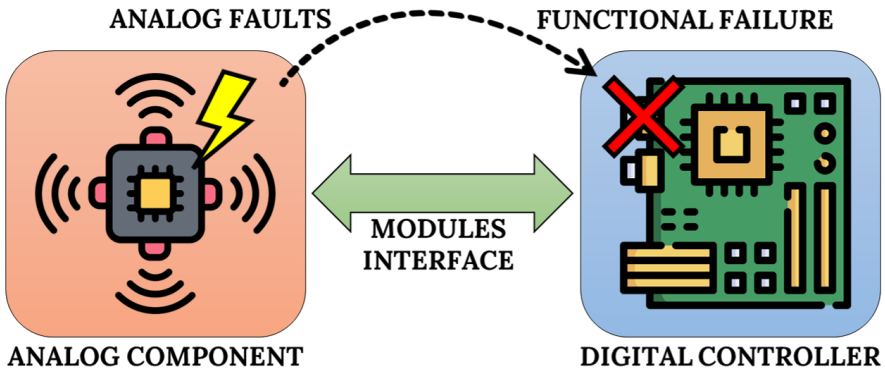


Fig. 4.21: Mixed-signal system setup: when an analog fault appears, how could the applicative software running on the digital controller react to these anomalous behaviors?

and analyses of faults in various specific domains (e.g., digital, analog, and physical) [203], with the goal of developing smart systems that prevent failure situations or mitigate challenging conditions [204].

The main claim of this work is that this compartmentalized approach is a weakness in smart system design, and that a system-synergic approach is rather necessary [205]. This consideration has been proposed in the past to address the challenges in Analog-Mixed Signals (AMS) design, where digital and analog components require careful co-design [202], and is here extended to cover other domains that go beyond pure functionality.

As Fig. 4.21 shows, faults in the analog system section could cause problems not only within the analog component but also at the digital level or even at the software level, if not handled correctly. For example, in the case of safety-critical systems, an incorrect behavior on the analog side due to altered physical conditions or faults may not always be intercepted and handled by the digital or software part. Such situations can result in system safety failures, leading to catastrophic consequences. On the contrary, if the same faults are tested together with the other parts of the system in a synergic approach, they would not propagate into the other components because the system can be effectively designed to handle them, thus avoiding further damage to other parts of the system. The main innovations proposed are:

- an analysis that highlights how integrating multi-domain fault models into the fault analysis process is critical for improving the robustness of the project design. Thus, adding an

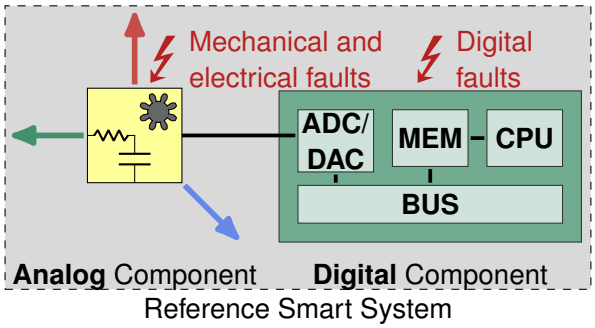


Fig. 4.22: Exemplification of a mixed-signal system and all the faults that can happen in the different domains.

analog comparator with digital output that controls the Micro Electro Mechanical Systems (MEMS) accelerometer response improves the digital robustness;

- functional qualification of the verification testbench analysis, for example, by improving the online calibration phase of a MEMS accelerometer by stimulating its functionality through the best stimulus. This analog wave can activate a higher percentage of multi-domain analog faults, resulting in a general improvement in the robustness of the entire smart system.

In this context, a smart system case study will highlight how the lack of holistic system fault evaluation during the design process could impact the system's functionality.

4.5.1 Multidomain faults and smart systems

The goal of this work is to argue that it is necessary for system designers to build safety mechanisms that consider and span multiple domains when designing systems comprising both analog and digital components. This *holistic vision* of the system is necessary to maintain adequate functional safety standards, as focusing on digital faults when designing safety mechanisms is not sufficient anymore. If the simulation also includes the analog part, then the digital component can be designed specifically to work with that analog device. Moreover, by extending the fault simulation, which already occurs in the digital section, to the analog part, we can design a digital component that deals more easily with the analog component's faulty behaviors. Improving the digital part's robustness increases the system's value, making it more reliable.

Standards like ISO 26262 [1] define how to assess electrical and/or electronic functional safety levels, but they are mostly oriented on how to model and inject faults in the digital domain [139]. This is a critical limitation when safety mechanisms must react to abnormal conditions even in the presence of failures generated by multi-domain faults, *e.g.*, wear and tear of an electromechanical component inside a MEMS.

As a motivation example, let's consider the accelerometer (*e.g.*, a MEMS placed on board a smart system) already introduced in Section 3.6 and its calibration phase, which could be performed during the physical production or directly on board (*e.g.*, at each ignition of the car that mounts such device inside of a control unit).

Functionality validation is very important in the manufacturing process of sensors, such as accelerometers. One way to test the correct behavior of an accelerometer is to calibrate it after

the production phase. The calibration verifies the accuracy and reproducibility of measuring instruments, such as sensors and measuring systems. A precise calibration is the precondition for accurate, reliable, and reproducible measurement results. The accuracy of this procedure is crucial to determining whether the sensor under test is functioning properly. For this reason, properly stimulating the accelerometer and verifying the returned result is crucial to determine whether the component is defective. The calibration accuracy is given by the number of faults this process can detect. Therefore, to make the calibration more precise, we need to stimulate the accelerometer to activate the highest possible number of faults. Thus, when one or more stimuli are selected as calibration inputs, finding those that activate as many faults as possible is crucial. The correct and effective method to detect the mentioned stimuli is through analog fault simulation. This analysis allows us to increasingly refine the input actions required to activate potential faults that may affect the system. Simulating fault behavior or altered conditions enhances the quality of the system calibration process, thereby making the component more robust to mutations in working conditions (*i.e.*, aging) and more resilient to analog faults as well.

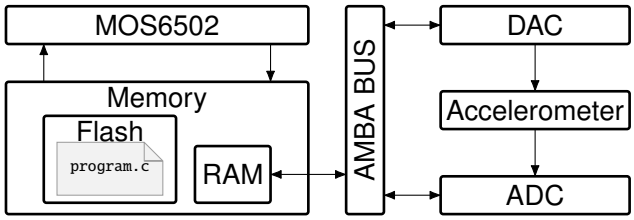


Fig. 4.23: Analog-Mixed Signal (AMS) case of study implemented as a Virtual Platform (VP) composed by a MOS 6502 microcontroller and a three-axis accelerometer.

4.5.2 Analog-Mixed Signal (AMS) case of study

Let's introduce the mixed-signal case study used throughout the article to support our claim. For the digital part, we adopted a MOS 6502 microcontroller (widely used in industrial applications) [206, 207], while for the analog part, we chose a three-axis accelerometer, characterized by a clear multi-domain composition. Since this work focuses on the analog part and its integration into the system, we chose a simple and well-known controller as the digital component. In contrast, the accelerometer is a simple sensor that mixes two physical disciplines: electrical and mechanical. The sensor was built based on a system known in the literature [66]. The considered mixed-signal system is simulated as a VP, allowing the combined simulation of the digital and analog parts together [208].

Fig. 4.23 shows a complete view of the VP: the fundamental blocks are the MOS 6502 microcontroller, a memory, an Advanced Microcontroller Bus Architecture (AMBA), and various peripherals connected to the AMBA, including an analog three-axis accelerometer. The accelerometer module is also coupled with an Analog-to-Digital Converter (ADC) signal converter that allows the discretization of the acceleration values to be written to the AMBA bus.

The three-axis accelerometer is an analog MEMS model that can measure three-dimensional acceleration values, whose structure is depicted in Fig. 3.8(a). The component comprises three identical one-dimensional accelerometers, `x`, `y`, and `z`, oriented in the three dimensions. As shown in 3.8(b), each accelerometer combines the mechanical domain (*e.g.*, mass-spring-damper subsystem) and the electrical domain (*e.g.*, capacitors). It is thus a suitable case study to show how electrical and mechanical faults could impact a digital controller. The power supply is the same for each accelerometer, while the force stimuli differ for `x`, `y`, and `z`.

This VP represents a typical mixed-signal system that can be used in any scenario where acceleration needs to be monitored and managed. For example, it may control the airbag on a vehicle, or a stability control or vibration measurement system on industrial machinery.

4.5.3 Impact of analog faults in the calibration phase of the accelerometer

Let us see how faults from different physical domains influence the robustness of the case study presented in the previous section.

Simulation setup

We developed a standard application, *i.e.*, the calibration process necessary for any MEMS accelerometer [209,210]. This calibration process can occur during the system's production phase, during installation within a more complex scenario, during reconfiguration, or by the user. The implemented calibration process involves monitoring the acceleration value produced by the accelerometer in response to a specific movement by the user. After the user initiates the process, the acceleration value produced in a precise time interval is requested from the accelerometer module. The acceleration value recorded by the accelerometer is converted by the ADC module and supplied to a comparator. Additionally, the value is placed on the bus to be communicated to the software component, which will display and store it. If the value the application measures is correct, the user is notified of a successful calibration. However, if the value falls outside the predetermined range, the application warns the user that a problem has occurred during the procedure. In this scenario, we assume that the user stimulus is always optimal; this allows us to perform analysis only on the integrity of the system, not on the correctness of the input provided from outside. So, in the case of incorrect calibration following optimal input, we assume that the problem may be internal to the hardware platform. A possible scenario could involve incorrect transmission of the acceleration data due to a software error, a digital fault, or an analog fault. To avoid this kind of error in the digital part, we chose to compare the acceleration value more than once with each measurement. By introducing a comparator module that reads the value from the DAC module, we perform an additional reading of the acceleration value from the accelerometer, thereby increasing the overall system's robustness. The Verilog-AMS code of the comparator is shown in Listing 4.1. The code shows that the module's behavior is controlled by the threshold parameters `positive_th` and `negative_th`. The real values of those parameters that truly define when the triggering signal is raised are set by the module, which instantiates the comparator together with the other analog modules. The comparator checks the received acceleration value `vin` against the thresholds. If the measured acceleration exceeds the

```
1 `include "disciplines.vams"
2 `include "constants.vams"
3 `timescale 1us / 1us
4
5 module comparator(
6     input electrical vin,         // Input voltage.
7     input wire       reset,       // Reset signal.
8     output reg [1:0] dout = 2'b00 // Output value.
9 );
10     // Positive threshold value.
11     parameter real positive_th = +1.00;
12     // Negative threshold value.
13     parameter real negative_th = -1.00;
14     // Track compactor status, initialize to ready.
15     reg ready = 1'b1;
16     // Positive threshold crossed.
17     always @(cross(V(vin) - positive_th)) begin
18         if (V(vin) > positive_th && ready == 1) begin
19             dout  = 1; // Set output to 1.
20             ready = 0; // Comparator fired.
21         end
22     end
23     // Negative threshold crossed.
24     always @(cross(V(vin) - negative_th)) begin
25         if (V(vin) < negative_th && ready == 1) begin
26             dout  = 2; // Set output to 2.
27             ready = 0; // Comparator fired.
28         end
29     end
30     // Reset the comparator and make it ready.
31     always @(posedge reset) begin
32         dout  <= 2'b00; // Set output to 0.
33         ready <= 1'b1;  // Comparator ready.
34     end
35 endmodule
```

Listing 4.1: Verilog-AMS implementation of the comparator module.

positive threshold, it sets `dout` to 1; if it is below the negative threshold, it sets `dout` to 2; otherwise, the `dout` value remains zero. The value inside the register `dout` is then memory-mapped in a specific region of memory to make it accessible from the software. Once the software has correctly read the status of the comparator for a test, it can reset its status through the `reset` signal, which is also memory-mapped.

A successful calibration occurs when the motion results in a reading that exceeds or falls below those thresholds. However, suppose the acceleration value produced by the user's stimuli does not cross the expected thresholds for the correct force input. In that case, the comparator keeps the value of `dout` to zero. In the context of a wrong calibration, investigating and determining what caused this result is crucial in the perspective of the next calibration. By adopting this solution, the comparator would respond differently if the application detected an incorrect calibration due to digital faults or software errors. Thus, it will be up to the user to determine whether to repeat the calibration or consider a component inspection, which would be necessary

when the comparator spots an analog fault. In the case of discordant responses from software applications and hardware comparators, the user can estimate where the fault is most likely to be located. Again, assuming the input is correct, the fault could be in the digital part if the application reports a failed calibration while the comparator's result is the opposite. Otherwise, the fault could be in the accelerometer if both give incorrect calibration as an answer. Consequently, a negative result following a calibration attempt can be due to multiple problems, not just a missed or incorrect movement by the user. In this scenario, considering faults is necessary as a possible cause of calibration failure.

The system was simulated for five seconds of operation, an interval deemed sufficient for testing the system's functionality.

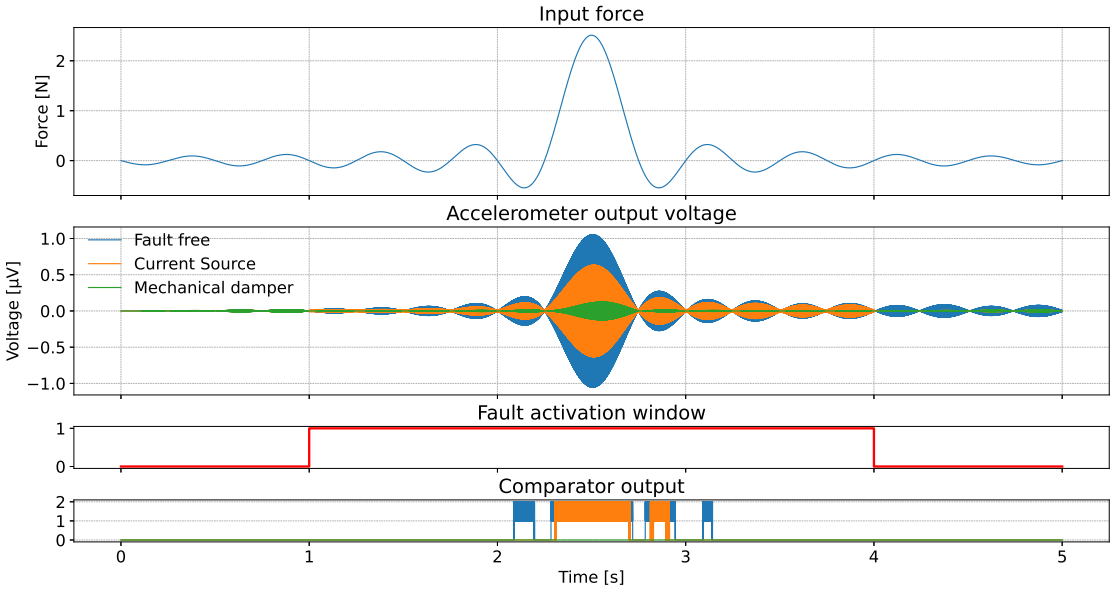


Fig. 4.24: From top to bottom, we have the input force exerted on the **x**-axis of the accelerometer, its **x**-axis output voltage in different conditions (fault-free, and electrical and mechanical faults), the time window when the fault is active, and finally the output from the comparator.

Correct system operation

The wave in the upper graph in Fig. 4.24 illustrates how a force input of approximately $2.6N$ is applied to the system. This simulated user input remains constant for all simulations presented below. This situation is a suitable scenario to keep the system input unaltered and study its response in relation to different working conditions. Between seconds 2 and 3, we can see a peak in the force on the system, indicating that calibration occurs within this specific time interval. Consequently, in our experiments, the cause of any calibration failure is not related to an incorrect input but to some other error or system failure.

The graph depicted in Fig. 4.24 shows the measured output acceleration in three different scenarios. The output data in the graph were generated by a simulation conducted using the QuestaSim software, which was used to simulate the entire AMS case study. First, a *fault-free*

simulation, which is identified by a blue line. Concerning the input plot, we can see how the movement of user input generates an acceleration signal. The small oscillations before and after the central portion of the signal are not relevant for calibration and are simply due to the system's positioning.

Fault injection

Let us now introduce the other two waveforms, which represent the accelerometer's behavior affected by two fault models. The first thing to note is that both faults presented have an activation window that lasts 3 seconds and begins after 1 second of simulation.

Regarding analog faults, we have already conducted an in-depth fault analysis and simulation on this accelerometer as well (see Section 3.6). Here, we present two of the analog fault models to demonstrate their impact not only on the purely analog part of the system but also on the other system parts, *i.e.*., digital and software, within the context of a sensor calibration process.

The first fault model we consider is a *mechanical damper*, identified by the green-colored line in Fig. 4.24. This mechanical fault consists of an increase in the resistance of motion of the mechanical part of the accelerometer. This fault can occur because of multiple causes, such as increased friction on the moving parts, hardening of the elastic components, or the unexpected presence of debris in the motion trajectories. This fault model has a precise effect on the accelerometer: the movement of mechanical parts is slowed down and restrained. Therefore, the measured acceleration will be of lower intensity even though the user has correctly performed the calibration movement. A similar effect can be seen in the second fault presented: an *unexpected current source* in the electrical part of the accelerometer. The fault simulates a slight leakage of electric current due to the influence of an electric field generated by other electric components, which acts directly on the accelerometer's electrical branches.

Impact of the digital subsystem

Although the motion was perfectly executed, the acceleration turned out to be very different from the fault-free data. The results plotted in Fig. 4.24 show that the current source affects the measured acceleration value slightly more than the mechanical damper fault. However, in both cases, the effect of the fault also propagates directly outside the accelerometer. The acceleration value determines whether the calibration was successful; an acceleration value lower than expected from a human causes the process to fail. In the case of the mechanical damping fault (green-colored line), the comparator module receives acceleration values that fall outside the expected range for the entire simulated period. Therefore, both the software component and the comparator report the failure of the process, despite the user executing the correct action. Instead, in the case of the electrical fault (orange-colored line), we observe a different situation: the comparator produces a successful calibration output, although the system response differs from the expected one. This scenario is a great example of how analog faults are crucial for refining the system parameters, such as the calibration threshold. Let us imagine what could have happened if a faulty accelerometer, affected by that slight electrical fault, were mounted on a

safety-critical system, where precision is key. The comparator range should be adjusted to also detect this kind of fault, in order to avoid non-negligible consequences. Without simulating this fault model, we wouldn't have been able to make this important refinement during the system's design phase.

4.5.4 Discussion

The situation reported by this case study is just one example of how analog faults can affect the analog module and the rest of the components of a more complex system. Analog faults span several important physical domains, all of which are important, as each fault highlights different behaviors in the system [73].

Let us imagine a real context for our simulation, where our simulated case study is the electronic part of an airbag system. This type of system must react with precise behavior to a specific stimulus applied to one axis, even during calibration; otherwise, devastating consequences for the overall system safety will occur.

As simulations show, a fault can significantly alter the functionality of a system, even to the point of being critical. Similar to our example, a fault within the accelerometer can prevent the control part of an airbag from triggering the protection system, even when an input is received that requires the device to deploy. The consequences of such a scenario would be non-negligible. A fault, such as a mechanical damper or an electrical influence, must also be studied and simulated in the design phase to reduce the risk of harm to people and property in various situations. There are numerous applications for an accelerometer in safety-critical scenarios: a fault in such a system that is not detected in time can lead to significant monetary or even safety issues.

This example highlighted that the design of multi-domain systems can be improved by including both digital and analog fault patterns. When applied to simulation models, faults become a crucial element in studying the proper functioning of individual modules and the overall system. Designing safer systems improves their quality and shortens the time frame for testing, verification, and potential corrections to be applied to the product. Thus, a smart system creation environment, such as the one described in this article, is the key to designing, simulating, and testing systems whose overall functionality must be guaranteed. Building a system that considers eventual system faults early in the design phase can also aid later analyses regarding the functional safety of a smart system.

Another advantage of behavioral fault simulation in a mixed-signal system is the increased accuracy of the testbench module. In fact, during the production phase, each system must pass a test procedure performed by a testbench module separate from the system. To successfully simulate faulty behavior, a signal that accurately highlights the presence and effect of the fault itself is required as input. Thus, while faults are injected, the system requires an input that enables the faults to be detectable, allowing for further analysis. By simulating fault behavior, we can enhance the quality of the testbench module by enabling it to trigger as many faults as possible during testing. Furthermore, with an advanced testbench, not only do we increase the

possibility of identifying a faulty system at the production stage, but we also build and test fault detection algorithms. These types of solutions would further enhance the system's robustness.

4.6 D-MATE: A Design Methodology for Connecting Automatic Test Equipment in Industry 4.0

In the era of rapid technological advancement under Industry 4.0, the demand for reliable semiconductor devices has surged across critical sectors like automotive, avionics, and healthcare. In these safety-critical fields, even minor component failures have serious consequences. As Chris Miller highlights in *Chip War: The Fight for the World's Most Critical Technology* [211], semiconductors are the backbone of modern technology, and ensuring their quality and reliability is paramount. Applying Automatic Test Pattern Generation (ATPG) to these semiconductor-based devices helps identify and intercept defective units. Machines that perform this operation automatically on an industrial scale are called Automated Test Equipment (ATE). These machines are essential for ensuring the quality and reliability of semiconductor products. ATE machines validate the functionality of chips and electronic components, improving product safety and operational integrity.

Despite their critical importance, ATE machines remain underexplored in the context of Industry 4.0. As chip production scales to meet the demands of interconnected smart technologies, the need for standardized, efficient, and automated testing processes grows. The main challenge is integrating ATE seamlessly into the broader industrial automation architecture, commonly referred to as the *automation pyramid*.

Although Industry 4.0 emphasizes smart manufacturing and interconnected systems, seamlessly integrating ATE machines into the automation pyramid remains challenging. Traditional ATE systems often operate in isolation from other manufacturing processes, resulting in inefficiencies and a lack of interoperability. The lack of standardized communication protocols for ATE, especially in Printed Circuit Board (PCB) testing, limits their potential to meet Industry 4.0 goals, such as interpretability and information transparency.

This work addresses the integration gap by introducing D-MATE, a novel approach that adopts the OPC UA standard and proposes a Companion Specification for ATE in PCB testing (see Fig. 4.25). The primary innovations of this work are as follows:

1. Integration of OPC UA with ATE: We present a methodology for incorporating OPC UA into ATE communication architectures, enhancing interoperability and integration within Industry 4.0 ecosystems;
2. Development of a new Companion Specification for ATE: We present a detailed Companion Specification specifically tailored for ATE machines in PCB testing, standardizing information models and communication protocols to improve operational efficiency and consistency.
3. Validation in a real-world scenario: We implement and test the proposed approach in the Industrial Computer Engineering (ICE) Laboratory, demonstrating its practicality and effectiveness in an industrial setting.

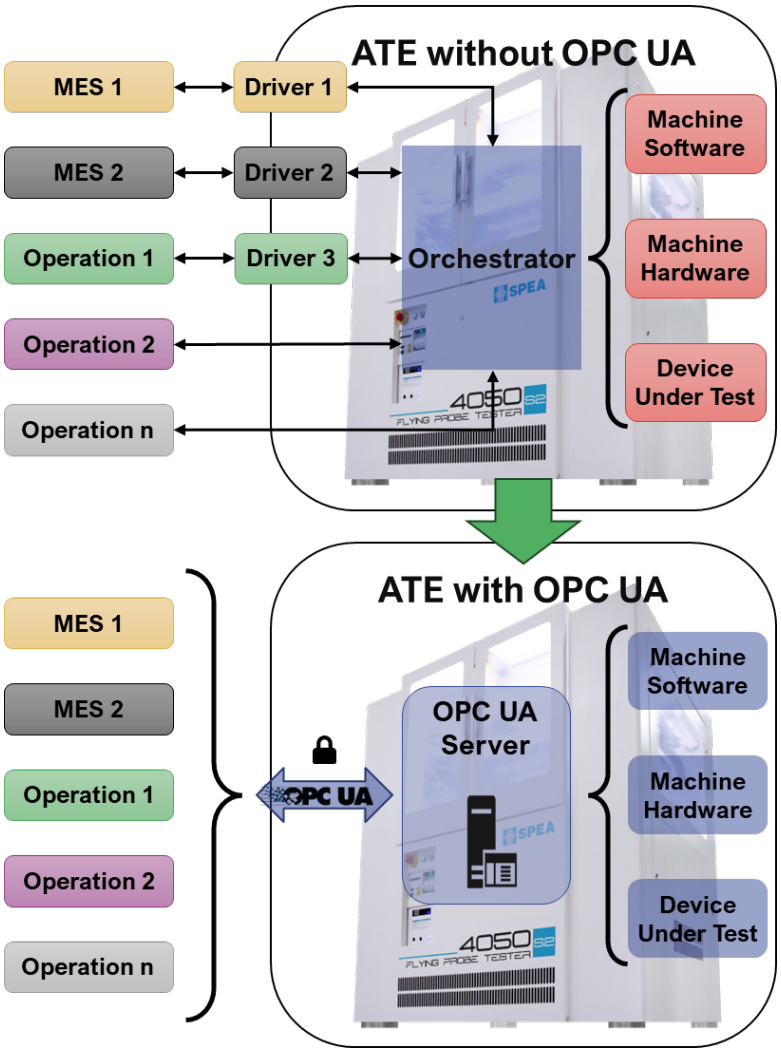


Fig. 4.25: This architecture uses Open Platform Communications Unified Architecture (OPC UA) as the central interface for accessing machine data and controlling operations. With OPC UA, all external communications, previously non-standardized and customized for different stakeholders, are unified under a single TCP/IP interface, enabling streamlined data transmission, improving management, and interoperability.

4.6.1 OPC UA Background

OPC UA is a flexible, platform-independent, service-oriented architecture compatible with diverse hardware and software environments (see Fig. 4.26). OPC UA effectively integrates industrial devices with cloud systems, microcontrollers, and various IoT applications. Its client-server architecture functions seamlessly across diverse platforms—including Windows, Linux, and mobile operating systems—making OPC UA ideal for the dynamic and rapidly advancing field of industrial automation [212–215].

The OPC UA framework includes essential components that enable secure, adaptable, and reliable communication. At its core are the *Information Model* [217], *Address Space* [218], and *Services* [219]. These elements standardize object representation and interactions between

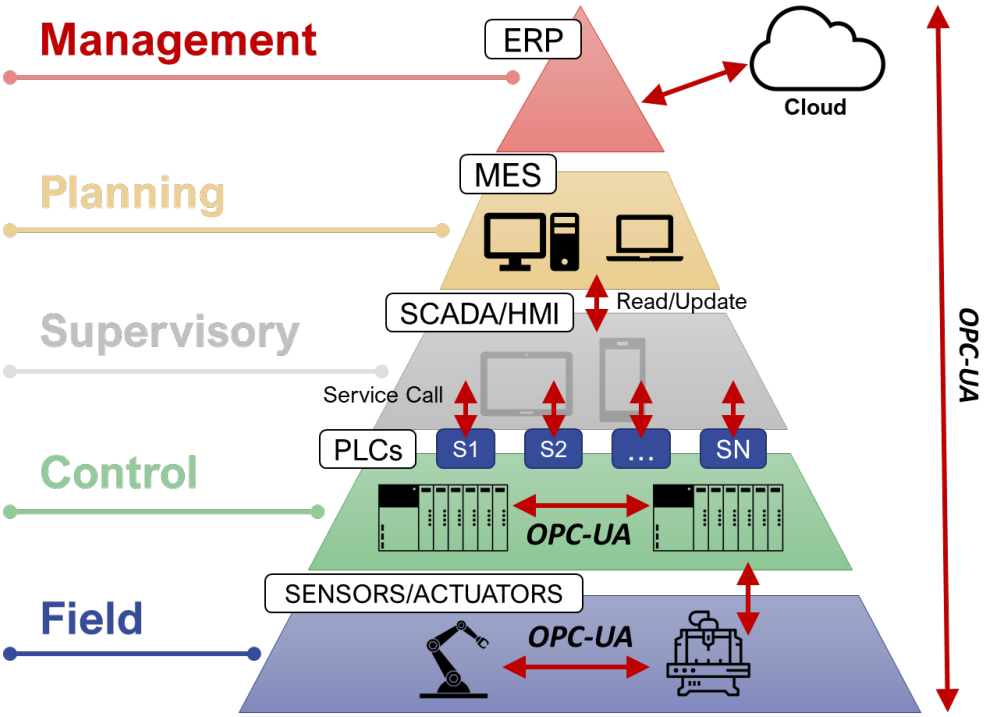


Fig. 4.26: The figure shows the automation pyramid and highlights the critical role of OPC UA across its levels, enabling data communication from field sensors and actuators to ERP systems in the cloud [216].

servers and clients. The *Information Model* provides a scalable structure for defining data types and their relationships, which are essential for representing complex industrial processes [220]. The *Address Space* organizes data and services, allowing clients to effectively navigate and manipulate them. As a service-oriented protocol, OPC UA relies on its *Services* component to provide fundamental operations that manage client-server interactions. These services include `Discovery`, which enables clients to locate servers (IP address and port), and `SecureChannel`, which establishes secure communication channels. Together, these components support binary and XML encodings, ensuring versatility across various industrial applications.

Security in OPC UA

The popularity of OPC UA largely stems from its robust security model (see Figure 4.27), which incorporates multiple encryption protocols to ensure data confidentiality and integrity. The currently supported **Security Policies** include:

- `None`: this policy applies no encryption or authentication.
- `Basic256Sha256`: uses the SHA-256 hashing algorithm with 256-bit encryption to secure data.
- `Aes256-Sha256-RsaPss`: combines AES-256 encryption with SHA-256 hashing and RSA-PSS for digital signatures.

- `Aes128-Sha256-RsaOaep`: uses AES-128 encryption, SHA-256 hashing, and RSA-OAEP for encryption and authentication.

A key aspect of the security model is **User Authentication**, which supports the following procedures for authentication:

- `AnonymousIdentityToken`: no authentication is required.
- `UserNameIdentityToken`: via username and password.
- `X509IdentityToken`: via X509 certificates.
- `IssuedIdentityToken`: via an external OAuth2 server.

These security mechanisms provide the flexibility and robustness essential for operation in industrial environments, where data protection and access control are paramount.

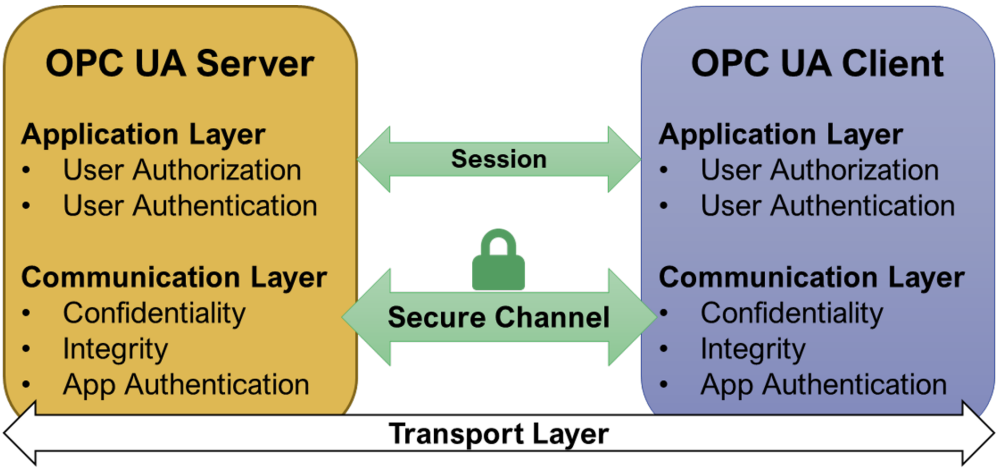


Fig. 4.27: This figure shows the OPC UA security architecture [221], as outlined in OPC 10000-6 [222]. The architecture supports various security objectives through different mappings, enabling security measures at multiple levels.

Companion specifications

A core feature of OPC UA is its ability to model complex data and processes in a standardized format using Information Models [223]. These models define data structure and exchange between devices, ensuring interoperability across systems. Industry groups and standardization bodies develop Companion Specifications to provide domain-specific guidelines on applying OPC UA [224]. For example, in the context of ATE, a Companion Specification can define the precise structure and semantics needed to standardize the communication of testing parameters, results, and machine states. Companion Specifications ensure that different ATE machines, even from various manufacturers, communicate using the same standardized language, promoting interoperability and scalability. Adhering to these specifications enables manufacturers to ensure compatibility with a wide range of systems, reducing integration costs and improving overall system reliability.

UA-.NETStandard library

The UA-.NETStandard library is an open-source .NET implementation of the OPC UA specification, designed to enable secure, platform-independent communication for industrial applications. We chose the UA-.NETStandard library for its comprehensive support of the OPC UA specification, including transport protocols such as TCP and HTTPS. The library supports both binary and XML encodings, making it versatile for developing scalable OPC UA applications. Additionally, the UA-.NETStandard library uniquely enables server timeout configuration through XML files or `ApplicationConfiguration.cs`. The UA-.NETStandard library is recognized as a leading open-source OPC UA implementation for its scalability [225]. It ensures the creation of fully compliant applications that adhere to the latest OPC UA standards. Notably, the UA-.NETStandard library fully supports all the security features specified by the OPC UA Foundation, making it ideal for implementing server architecture for ATE machines [226, 227].

4.6.2 Connecting automatic test equipment

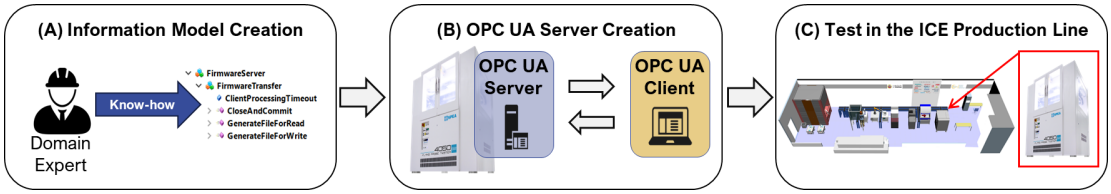


Fig. 4.28: The D-MATE framework begins with the creation of an information model (A), where machine operations and user roles are clearly defined. This is followed by the development of an OPC UA server to expose these operations and a client to simulate a generic user interacting with the system (B). Finally, the server is tested within the Industrial Computer Engineering (ICE) production line using tailored recipes, validating its functionality in a real-world environment (C).

The D-MATE framework comprises three key stages, depicted in Fig. 4.28 and outlined in this section.

Information model creation

The first stage of the framework involves creating an information model that serves as the system's foundation. This model defines the structure and semantics needed for effective communication between ATE machines and other components in the production environment. The IM standardizes interactions between the ATE and its surrounding systems by mapping all machine operations, test parameters, and data types. At this stage, we ensure that the information model is compatible with the OPC UA standard, facilitating interoperability and scalability across various industrial scenarios.

Server implementation

With the information model established, the next step is implementing the OPC UA server. The server is the central hub for managing and exposing the ATE machine's services and data.

In this stage, we integrate the information model into the server infrastructure, enabling it to handle client requests, such as initiating tests or retrieving data. The server also includes custom methods for key functions like file transfers and secure communication, providing a robust platform for real-time machine management.

Test in the ICE production line

The final stage validates the D-MATE framework in a real-world industrial setting, specifically on the ICE Laboratory's production line. Here, we deploy the OPC UA server and a general-purpose client to interact with ATE machines in a fully functional manufacturing environment. The production line, with various manufacturing cells and dynamic workflows, serves as a testbed for assessing the system's efficiency, scalability, and flexibility. By executing customized production recipes and monitoring the ATE performance, we evaluate the framework's robustness and practical applicability for seamless integration in an Industry 4.0 setting.

4.6.3 Implementing the connection

This section exemplifies the implementation of the methodology through the design and development of an information model and OPC UA servers for an ATE machine.

Information Model Creation

In the first phase, to address the lack of standardization in ATE machine communication and operation when handling PCBs, we developed a tailored information model based on OPC UA. This model provides a structured, consistent approach to managing various aspects of ATE machines, ensuring interoperability and efficiency in line with Industry 4.0 requirements. The information model organizes essential details into key categories detailed in this section. Each category specifies essential *Attributes* and *Methods* (if available) to streamline data handling and communication. The proposed information model standardizes communication, operation, and maintenance management of ATE machines in PCB testing. Adopting this model enables interoperability, efficiency, and reliability, aligning industry practices with the principles of Industry 4.0.

ATE Machine Identification

This category provides a unique identifier for each ATE machine and records fundamental details for asset management within the production environment. It contains no methods.
Attributes:

- `MachineID`: A unique identifier for the ATE machine.
- `Manufacturer`: The name of the machine's manufacturer.
- `Model`: The specific model number or name.
- `SerialNumber`: The serial number for traceability.
- `FirmwareVersion`: The installed firmware version.
- `Location`: The machine's physical or logical location within the facility.

System Configuration

This component captures the setup and operational parameters of the ATE machine, enabling configuration and connectivity.
Attributes:

- `ConfigurationID`: A unique identifier for the machine's configuration.
- `HardwareSetup`: Details of hardware components and their setup.
- `SoftwareSetup`: Information on software components and their versions.
- `NetworkConfiguration`: Network settings to ensure secure and effective communication.

Methods:

- `LoadConfiguration()`: Loads a specified configuration onto the ATE machine.
- `SaveConfiguration()`: Saves the current configuration for later reference.

Test Configuration

This component defines the setup and management of testing processes, enabling operators to establish and control various testing parameters.
Attributes:

- `TestID`: A unique identifier for each test configuration.
- `TestName`: A descriptive name for the test.
- `TestParameters`: Specific parameters and settings for the test.
- `TestSequence`: The sequence of operations required for the test.
- `TestLimits`: Defines acceptable limits and thresholds for test results.

Methods:

- `CreateTestConfiguration()`: Creates a new test configuration.
- `UpdateTestConfiguration()`: Updates an existing test configuration.
- `DeleteTestConfiguration()`: Deletes a specified test configuration.
- `GetTestConfiguration()`: Retrieves details of a particular test configuration.

Test Execution

This component manages each test's lifecycle, enabling real-time monitoring and giving operators control over testing processes.
Attributes:

- `ExecutionID`: A unique identifier for each test execution.
- `StartTime`: The timestamp that indicates when the test starts.
- `EndTime`: The timestamp that indicates when the test concludes.
- `Status`: The current status of the test (e.g., Running, Completed, Failed).

Methods:

- `StartTest()`: Initiates a test based on the specified configuration.

- `StopTest()`: Stops an ongoing test.
- `PauseTest()`: Pauses an active test.
- `ResumeTest()`: Resumes a paused test for operational flexibility.

Results Reporting

This component standardizes the recording and management of test results, ensuring consistent documentation for analysis.
Attributes:

- `ResultID`: A unique identifier for each test result.
- `TestID`: References the test configuration used.
- `ExecutionID`: Associates the result with a specific test execution.
- `Timestamp`: The timestamp when the result was generated.
- `ResultData`: Contains detailed data collected from the test.
- `PassFailStatus`: Indicates whether the test passed or failed.

Methods:

- `GetTestResults()`: Retrieves test results based on specified criteria.
- `ExportResults()`: Exports results in standard formats (e.g., CSV, XML).

Maintenance and Diagnostics

This category maintains machine health through scheduled maintenance and diagnostics, ensuring reliability and performance.
Attributes:

- `MaintenanceSchedule`: A schedule for regular maintenance tasks.
- `DiagnosticsData`: Data from diagnostic checks on machine performance.
- `ErrorLogs`: Records of errors encountered by the machine.
- `PerformanceMetrics`: Tracks metrics such as uptime and throughput.

Methods:

- `RunDiagnostics()`: Runs a diagnostic check.
- `GetDiagnosticsReport()`: Retrieves a report from diagnostics.
- `ScheduleMaintenance()`: Schedules a maintenance task.
- `GetMaintenanceHistory()`: Accesses the machine's maintenance history.

Security Considerations

Given the critical nature of ATE machines, security is a core aspect of the model, ensuring secure communication, authentication, and authorization protocols.
Attributes:

- `UserRoles`: Defines user roles and permissions.
- `AccessLogs`: Logs access and operational activities for monitoring.

Methods:

- `AuthenticateUser()`: Authenticates users based on secure credentials.
- `AuthorizeOperation()`: Verifies if a user is authorized for specific operations.

OPC UA Server Creation

After creating our information model, we used the model compiler [228] to convert the XML-based model into source files compatible with our OPC UA server environment. This process ensures compliance with the OPC UA standard and establishes the architecture needed for effective client-server communication. Two primary classes manage the server implementation: `RCServer` and `RCNodeManager`. The `RCServer` class manages overall server properties, node manager creation, and server lifecycle events. Specifically, it configures the server's URI, handles requests, and loads essential properties like manufacturer and software version details. Meanwhile, the `RCNodeManager` is responsible for setting up and maintaining the server's address space. It loads predefined nodes, converts them to the required typed nodes, and links methods to appropriate callbacks, enabling interaction with ATE functionalities.

Our OPC UA server design includes a custom request-handling system to manage prioritized command execution. This handler ensures deterministic management of high-priority commands, particularly those associated with safety-critical operations, by giving them precedence in processing. This system maintains operational integrity, allowing the server to efficiently manage and execute diverse client requests within the production environment.

4.6.4 Case Study: The ICE Laboratory

The ICE Laboratory at the University of Verona is a state-of-the-art research facility for demonstrating and testing computational technologies for Industry 4.0 and 5.0 applications. The laboratory features a reconfigurable production line with various manufacturing cells, including robotic assembly, quality control, additive and subtractive manufacturing, autonomous logistics, and functional testing.

The facility's functional testing cell incorporates the SPEA 4050S2 Automatic Flying Probe Tester [229], a state-of-the-art ATE system for electronic board testing. This tester is seamlessly integrated into the laboratory's service-oriented manufacturing software architecture and is deployed via a Kubernetes cluster. This architecture framework comprises two core components: the *Data collection architecture* [230] and the *Automation Manager* [231].

OPC UA-Based Integration

Each machine in the laboratory, including the SPEA 4050S2, is equipped with an OPC UA server that exposes its services to the *Automation Manager*. The *Data collection architecture* monitors equipment status, stores gathered data, and routes machine commands between laboratory devices and the *Automation Manager* through brokers like RabbitMQ and MQTT. The

Fig. 4.29: The SPEA 4050S2 Automatic Flying Probe Tester in the ICE Laboratory. The highlighted area shows a mini-pallet sliding toward the side bay to position a board for testing and programming.

Automation Manager orchestrates production by dynamically executing work orders, responding to real-time changes in the production environment, and controlling the SPEA 4050S2's operations according to predefined recipes.

Testing Process

During the demonstration, the SPEA 4050S2 performed various tests on electronic boards, including in-circuit tests, power-on tests, functional tests, boundary scans, and onboard programming. Boards were fed into the machine via a mini-pallet conveyor system, thanks to its seamless integration within the production line. Using the implemented OPC UA server, the ATE was fully integrated into the laboratory's automated workflow, allowing interaction with other cells in the production line (see Fig. 4.29).

The demonstration thoroughly exercised all facets of the companion specification, validating the server's capability to handle diverse operational scenarios. We rigorously tested each model component, from machine identification and configuration to test execution and result reporting, verifying its support for various production needs. Additionally, we evaluated the server's security framework by testing all security levels and access permissions. These tests demonstrated the system's robustness in managing secure communications and controlling access at multiple levels, highlighting the companion specification's flexibility and scalability in an industrial setting.

Dynamic Production Scenarios

The laboratory setup demonstrated the flexibility of the ATE in modern production environments by executing different production recipes. In one scenario, the SPEA 4050S2 acted as a testing customer, downloading customized firmware from the board producer's OPC UA server and flashing it onto the electronic boards after successful testing. This setup showcased the production line's ability to accommodate highly customizable workflows, emphasizing the flexibility and efficiency of the OPC UA-based integration.

4.7 Human-Centered Digital Twin

The transition from Industry 4.0 to Industry 5.0 necessitates a paradigm shift, where the human operator is no longer viewed merely as an external supervisor, but as an integral and active component of the CPS [232]. Consequently, the functional safety assessment of an industrial system cannot be complete without modeling the behavior of the human element [233, 234]. Specifically, "biological faults" such as fatigue, distraction, or drowsiness must be treated as parametric deviations in the system's control loop, as they can critically impact production safety and efficiency.

To address this challenge, this section presents the IMHU (Integrated Metaverse for Human-centric bUilding) framework. This application leverages the multi-domain modeling concepts established in this thesis to create a high-fidelity Human Digital Twin (HDT) capable of real-time monitoring and bi-directional interaction with the physical production line.

4.7.1 The six-stage development lifecycle

Applying the thesis methodology to the human operator requires defining the nominal and faulty states of the "human component". Unlike mechanical systems described by differential equations, human behavior is visually complex and non-deterministic. Therefore, the framework adopts Unreal Engine and MetaHuman technology [43] to create a hyper-realistic virtual replica that acts as a Digital Shadow.

The framework operates through a rigorous six-stage lifecycle, designed to bridge the gap between physical reality and virtual simulation (see Fig. 4.30):

- **MetaHuman Creation (A)**: The process begins with the creation of a high-fidelity 3D model using the MetaHuman Creator. This allows for precise customization of the operator's physical attributes, treating the human body with the same modeling rigor applied to CAD mechanical components.
- **Digital Shadow Creation (B)**: Real-time synchronization is established using the LiveLink protocol [235]. Unlike traditional motion capture, which requires expensive suits, this approach utilizes standard iOS depth-sensing cameras to stream facial tracking data (ARKit blend shapes) to the Unreal Engine instance. This creates a Digital Shadow capable of mirroring micro-expressions and head movements with low latency (see Fig. 4.31).

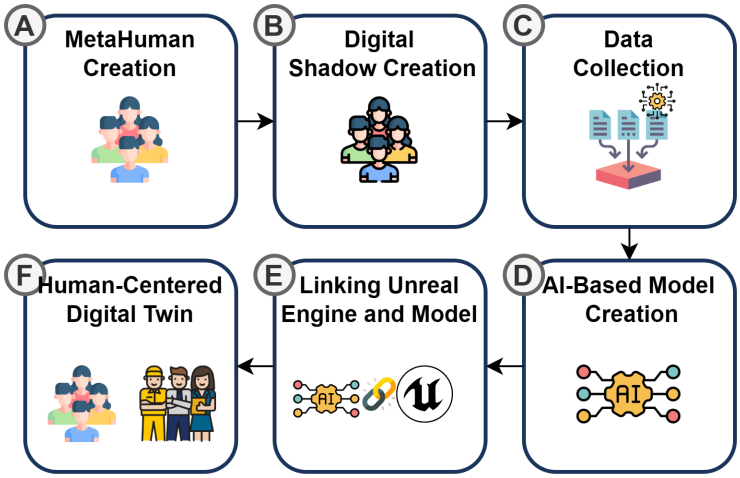


Fig. 4.30: An overview of the IMHU framework for implementing a human-centered Digital Twin in an Industry 5.0 context. (A) The process starts by creating a MetaHuman, a virtual replica of the real human. (B) The connection between the real human and the virtual replica is established using the LiveLink plugin, forming the digital shadow that allows real-time data acquisition. (C) Then, images were acquired and labeled to capture the operator's real-time movements under awake and drowsy conditions. (D) A state-of-the-art deep learning-based model, YOLOv8, is trained to detect the operator's state. (E) A client-server architecture is set up to facilitate the communication between YOLOv8 and Unreal Engine. (F) Finally, the Digital Twin reflects the operator's state (in our case, awake or drowsy) and provides alerts based on real-time conditions.

- **Synthetic Data Collection (C)**: Consistent with the thesis's focus on synthetic data generation (Chapter 3), the Digital Shadow is used to generate a labeled dataset. The operator simulates various states (*e.g.*, alert, yawning, eyes closing) within the virtual environment, and a Python script automatically captures annotated screenshots from multiple angles.
- **AI-Based Model Creation (D)**: The generated dataset is used to train a deep learning model. Specifically, the framework utilizes YOLOv8 (You Only Look Once) [236], a state-of-the-art object detection architecture. The model is trained to classify the operator's cognitive state into two classes: Nominal (Awake) and Faulty (Drowsy), effectively acting as a runtime monitor.
- **Unreal Engine-Model Linkage (E)**: A robust client-server architecture connects the simulation engine with the AI inference engine.
- **Human-Centered Digital Twin Deployment (F)**: The system is deployed to provide real-time feedback, closing the loop by triggering alerts when the "faulty" state persists.

4.7.2 The drowsiness detection model

The core of the monitoring system is a deep learning model based on the YOLOv8 architecture [236], chosen for its balance between speed and accuracy in real-time object detection. The network is composed of two primary blocks:

- Backbone: Utilizes EfficientNet-B4 [237] to extract multi-scale features from the input video frames.

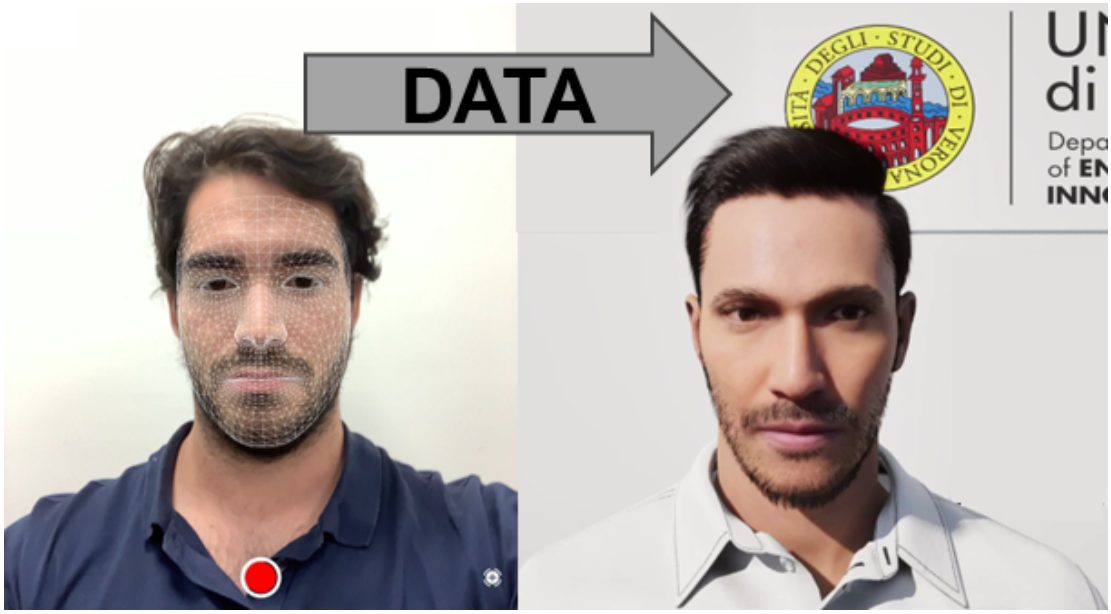


Fig. 4.31: Digital Shadow result. With this, we enable a real-time interaction between the physical operator and the MetaHuman. We used the LiveLink Face application [235], developed by Epic Games, for facial motion capture.

- Detection Head: Employs a NAS-FPN (Neural Architecture Search Feature Pyramid Network) [238] for feature fusion, combining feature maps through element-wise addition and max-pooling to handle objects at varying scales.

Loss function design

The YOLOv8 loss function consists of three main components: *localization loss*, *confidence loss*, and *class loss*. The *localization loss* measures the discrepancy between the predicted bounding box and the ground truth bounding box using the Intersection over Union (IoU) metric. Given the predicted box $\hat{B}$ and the ground truth box B, the IoU is defined as:

$$IoU(\hat{B}, B) = \frac{\hat{B} \cap B}{\hat{B} \cup B} \, . \tag{4.4}$$

The localization loss is typically computed as the negative logarithm of the IoU score or its variants (e.g., GIoU, DIoU), penalizing poor overlap between the predicted and true bounding boxes. As a result, the loss can be expressed as:

$$\mathcal{L}_{loc} = 1 - IoU(\hat{B}, B) \, . \tag{4.5}$$

The *confidence loss* penalizes incorrect predictions of object presence in each grid cell. It is computed as the binary cross-entropy loss between the predicted object confidence $\hat{C}$ and the ground truth confidence C, where $C = 1$ if an object is present, and $C = 0$ otherwise:

$$\mathcal{L}_{conf} = -[C \log(\hat{C}) + (1 - C) \log(1 - \hat{C})] \, . \tag{4.6}$$

This ensures that the model correctly identifies the presence or absence of objects in each predicted bounding box.

Finally, the *class loss* quantifies the error in classifying the object in each bounding box. It is typically computed as the cross-entropy loss between the predicted class probabilities $\hat{p}_i$ and the true class labels p_i over all classes i, as follows:

$$\mathcal{L}_{class} = -\sum_{i=1}^{N} p_i \log(\hat{p}_i) \, . \tag{4.7}$$

The total loss $\mathcal{L}_{total}$ is a weighted sum of these three components:

$$\mathcal{L}_{total} = \lambda_{loc}\mathcal{L}_{loc} + \lambda_{conf}\mathcal{L}_{conf} + \lambda_{class}\mathcal{L}_{class} \, , \tag{4.8}$$

where λ_{loc}, λ_{conf}, and λ_{class} are hyperparameters that balance the contributions of each loss term.

Training and Performance

The model takes in a video frame that captures a person's face or upper body (see Fig. 4.32). The model outputs bounding boxes representing the areas in the image where it detects the human face. It also predicts a confidence score for each bounding box, indicating the probability that the detected object is present. Each detected object is assigned a class label, *e.g.*, awake or drowsy.

The model was implemented in PyTorch [239] and trained for 100 epochs using the AdamW optimizer [240] with a learning rate of 1×10^{-2} on an NVIDIA RTX 3060 GPU. The dataset consisted of manually labeled images (using CVAT [241]) capturing diverse operator poses. As reported in Table 4.6, the model achieved exceptional performance metrics, with Accuracy, Precision, and Recall all reaching approximately 99.50% for both the "Wakefulness" and "Drowsiness" classes.

4.7.3 Real-time monitoring and system latency

To integrate this model into the control loop, a robust Client-Server Architecture was developed using Python.

- Data Acquisition: The system captures real-time screenshots of the operator from the Unreal Engine viewport using the `pyautogui` library [242].
- Inference Pipeline: The server processes these frames via the YOLOv8 model. The inference time is approximately 30 ms per frame.
- Feedback Loop: The prediction results (state, bounding box coordinates) are transmitted back to the Unreal Engine client via socket communication.

The total system latency, measured from the physical operator's movement to the Digital Twin's reaction, is approximately 250 ms. This responsiveness is sufficient to detect rapid onset of fatigue. A safety mechanism triggers an acoustic alarm if the "Drowsy" state is detected for more than 3 consecutive seconds.

Table 4.6: Accuracy (in percentage), precision, and recall (in a range from 0 to 1) of the YOLOv8 model.

Class	Accuracy ↑	Precision ↑	Recall ↑
Wakefulness	99.50 %	1	0.998
Drowsiness	99.50 %	0.965	1
Average	99.50 %	0.988	0.999

4.7.4 Human-Centered Digital Twin

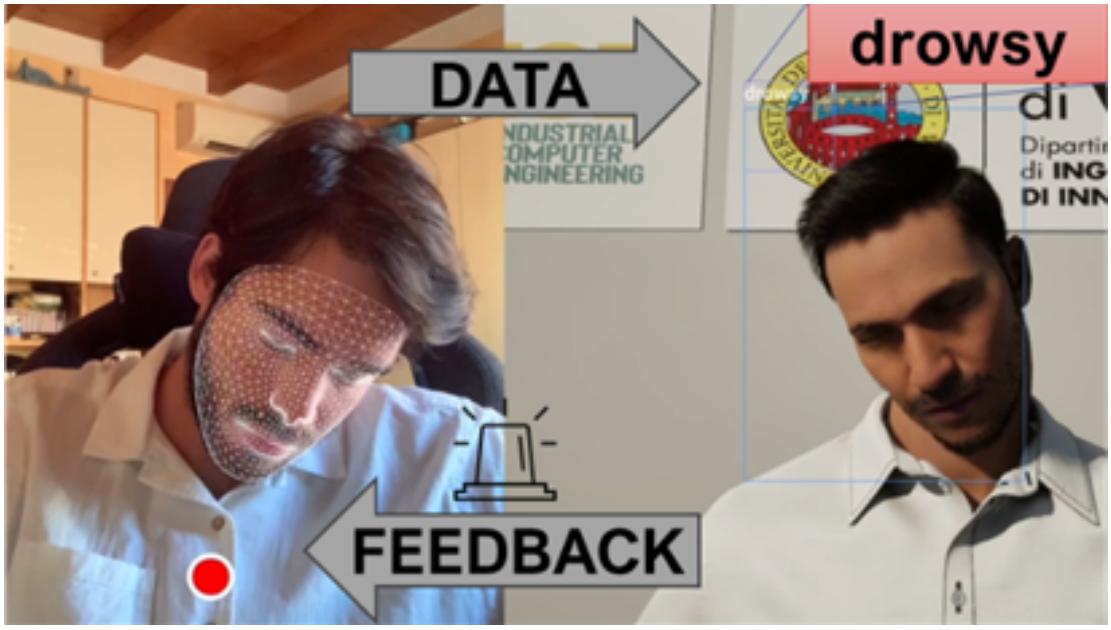


Fig. 4.32: A screenshot of our proposed human-centered Digital Twin.

4.7.5 Integration in the ICE Laboratory

The industrial applicability of IMHU was validated by deploying it within the ICE production line at the University of Verona (see Fig. 4.33).

The integration relies on a Service Oriented Architecture (SOA) managed by a Data Integration Hub (DIH). Depicted in Fig. 4.34, in this topology:

- The Human Digital Twin acts as an IoT edge device, communicating via MQTT [243] and OPC UA protocols.
- The operator's state (e.g., drowsiness level) is encapsulated as a variable within the OPC UA information model.
- This data is exposed to the Meta-MES [244], allowing the factory's automated supervisory system to react dynamically—for instance, by slowing down a robot or pausing a process if the operator is compromised.

Fig. 4.33: Overview of the ICE laboratory and our human-centered Digital Twin integrated within the manufacturing Digital Twin.

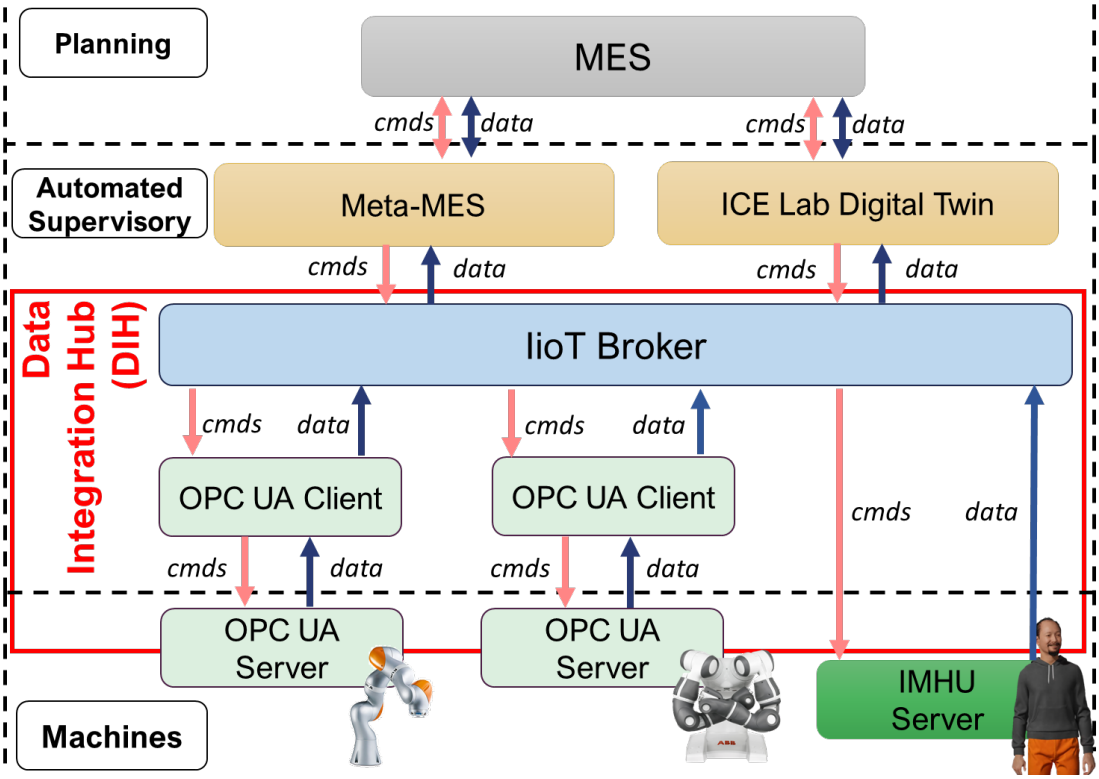


Fig. 4.34: Close-up view of the integration of our IMHU integration in the ICE lab SoM-enabled architecture.

This experiment demonstrated that by treating the human operator as a monitorable service, the framework enables seamless integration into legacy Industry 4.0 infrastructures, paving the way for the adaptive, human-centric collaboration envisioned by Industry 5.0.

4.8 Summary and discussion

This chapter has demonstrated that the multidomain fault modeling methodology, defined in Chapter 2 and validated in Chapter 3, establishes a critical link between theoretical simulation

and concrete industrial engineering challenges. By moving beyond the mere reproduction of failure modes, we have shown how high-fidelity synthetic fault data serves as a fundamental enabler for design automation, runtime verification, and smart manufacturing strategies.

The applications presented herein demonstrate the versatility of the proposed framework across various stages of the system lifecycle, showing that a unified approach to fault modeling can drive value in diverse contexts.

4.8.1 From "testing safety" to "designing safety"

The Automated Battery Pack Design Tool exemplifies the shift from reactive testing to proactive design. By scaling thermal fault injection from a single cell to complex pack configurations, the methodology enables designers to evaluate resilience against critical failures during the architectural phase. Similarly, the Contract-Based Monitoring application demonstrates that injecting faults into multidomain simulations provides the necessary "negative examples" to rigorously tune assumption-guarantee contracts. This ensures that runtime monitors can reliably distinguish between nominal operating variations and genuine faults, thereby bridging the gap between physical dynamics and formal cyber specifications.

4.8.2 The human as a system component

A significant contribution of this chapter is the alignment with the Industry 5.0 paradigm through the IMHU Framework. By applying the fault modeling methodology to the human operator, treating drowsiness as a parametric fault within the control loop, we have demonstrated that functional safety must encompass the biological domain. The integration of the Human Digital Twin into the factory's Service-Oriented Architecture proves that human states can be monitored and managed with the same rigor applied to mechanical assets.

4.8.3 Closing the Reality Gap

Finally, the integration with ATE via the D-MATE methodology and the robust MEMS calibration workflow demonstrates the framework's value on the factory floor. By defining standard interfaces and using simulated faulty behaviors to validate calibration routines, we ensure that the digital thread remains an important part of the testing production process.

In conclusion, this chapter demonstrates that the proposed multidomain fault models are not merely simulation artifacts but rather actionable engineering tools. They enable a holistic approach to reliability that spans from the early design of battery thermal management systems to the real-time monitoring of human operators, ultimately paving the way for more resilient, adaptive, and human-centric ICPSs.

5

Towards a unified digital twin for electric vehicles

5.1 The convergence of multidomain modeling

Throughout this thesis, I have presented methodologies for modeling, simulating, and validating faults across specific physical domains and individual subsystems. Chapter 2 established the theoretical framework of physical analogies; Chapter 3 validated these models using isolated components, such as DC motors, battery packs, and MEMS sensors; and Chapter 4 demonstrated advanced applications, including human-centric monitoring and automated design exploration.

However, the ultimate potential of these contributions lies not in their isolation but in their integration. A modern Electric Vehicle (EV) represents the quintessential Industrial Cyber-Physical System (ICPS), aggregating every challenge addressed in this work: high-performance electromechanical actuation, critical energy storage, pervasive sensing, complex control logic, and the indispensable role of the human driver. Furthermore, the automotive sector is one of the most sensitive to the issue of functional safety; the ISO 26262 standard [1] proposes fault models to test vehicle behavior. However, models are strictly necessary for testing situations that differ from the ordinary ones in which a vehicle may end up.

This chapter proposes a *conceptual* framework for a Unified EV Digital Twin. By synthesizing the technologies developed in previous chapters, I outline a comprehensive simulation architecture that supports the vehicle lifecycle from early design to operational safety monitoring.

5.2 Background on automotive digital twin

The concept of the Digital Twin (DT) has permeated the automotive industry, evolving from simple CAD representations to complex, data-driven replicas used for design, manufacturing, and Predictive Maintenance (PdM) [245]. Although the DT that have been implemented are mostly used for PdM, the usage of such models for seek of design is of high interest [246]. However, a review of the current scientific literature reveals that most existing solutions focus on isolated subsystems or specific functionalities [245, 247], lacking a truly unified multi-domain perspective that integrates the physical vehicle dynamics with the human operator's

physiological state. Research has been presented on monitoring overheating and the PdM of braking systems [248–250]. Several studies have proposed DTs for electric powertrains to optimize efficiency and monitor health, while other approaches focus on optimizing the fuel or the electrical energy consumed to extend the reachable range [251,252]. Interesting approaches have also been proposed to monitor the communications around the car wiring [253]. Another cornerstone challenge in the automotive field is the real-time data transfer from the car to the DT on the cloud [254,255]. For this reason, we propose a concept of a model that can run on the computers installed in the car, which are increasingly gaining more computing power. Furthermore, since the model would be installed and run directly on the car, there would be none of the well-known privacy issues associated with moving such sensitive data to the cloud [245].

While multidomain simulation tools exist, integrating fault injection across these heterogeneous domains remains an open challenge. Current methodologies often focus on single-domain fault propagation, *i.e.*, the one of the specific DT [256]. What is currently missing in the state of the art is a holistic simulation framework that can simultaneously:

- Model the cross-domain propagation of physical faults (*e.g.*, a thermal battery fault limiting motor torque).
- Integrate the human element not just as a disturbance, but as a faulty component within the control loop.
- Support bi-directional interaction where the vehicle's health affects the driver (*e.g.*, vibrations causing fatigue) and the driver's state affects the vehicle (*e.g.*, erratic inputs stressing the battery).

5.3 Architecture of the Unified EV Digital Twin

The proposed Unified Digital Twin is not merely a collection of models but a system-of-systems where the interaction between domains generates emergent behaviors. Fig. 5.1 illustrates the architectural integration of the components developed in this thesis into a complex EV simulation.

In particular, we can observe the following from the diagram:

1. The DC motor model from Section 3.5; here, we can choose between the Verilog-AMS and SystemC AMS code available (from the monitor study, see Section 4.1). Note: we also have the thermal model of this component (see Section 3.5.2). Therefore, we can also simulate overheating or other types of thermal failures, as well as mechanical or electrical malfunctions.
2. The battery pack, since we are designing an EV model. As we can see from the blue and orange arrow, this system is directly feeding the electrical motor. Here, we can simulate a particular thermal condition of the battery pack, also taking into account the environmental temperature, which can affect the battery.
3. Micro Electro Mechanical Systems (MEMS) sensors, including accelerometers. Nowadays, modern vehicles (electrical-powered or not) are equipped with numerous sensors, making our MEMS fault analysis and testing methodology (see Section 3.6) a suitable fit for the

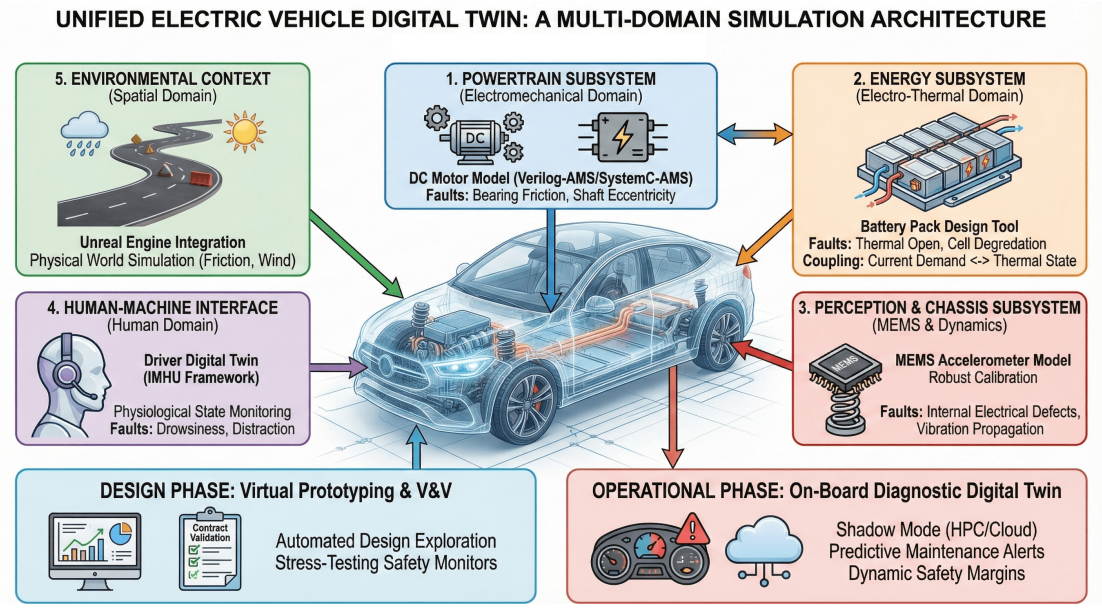


Fig. 5.1: An overview of the Unified Electrical Vehicle Digital Twin model.

context. Also the behavioral monitors fits the role of the dynamic checkers for the car behavior (see Section 4.1):

4. The driver, interfacing directly with the car, and what makes automotive such a functional safety environment sensible. The IMHU methodology (see Section 4.7) fits perfectly the car's driver role: the focus on the road must be maintained.
5. Environment around the vehicle, which can change the car conditions or state in any moment. The simulations and testing done for the multirotor drone (Section 3.8 demonstrate that Unreal Engine is a suitable tool to simulate a system in a 3D environment. Although we are currently in the early stages of this process, the initial results are promising for a future behavioral simulation model.

5.3.1 The Powertrain Subsystem

At the heart of the EV lies the propulsion system. This first subsystem is shown in Fig. 5.2. The DC Motor model with gear train validation serves as the high-fidelity physics engine. Instead of ideal torque sources, the vehicle dynamics are driven by the Verilog-AMS/SystemC AMS motor models. Moreover, we can inject specific mechanical faults (*e.g.*, bearing friction or shaft eccentricity) derived from the taxonomy in Chapter 2. This allows the simulation of subtle degradation modes that cause vibrations detectable only by specific sensors, rather than just catastrophic failures. The other side of the powertrain is the battery pack module. In Chapter 4, we presented an automatic tool that actually provides the energy backbone for our large model. Unlike standard battery models, which treat the pack as a monolithic voltage source, this subsystem models the thermal and electrical states of individual cells and cooling paths. The motor's current demand directly impacts the battery's thermal state. Conversely, a thermal fault

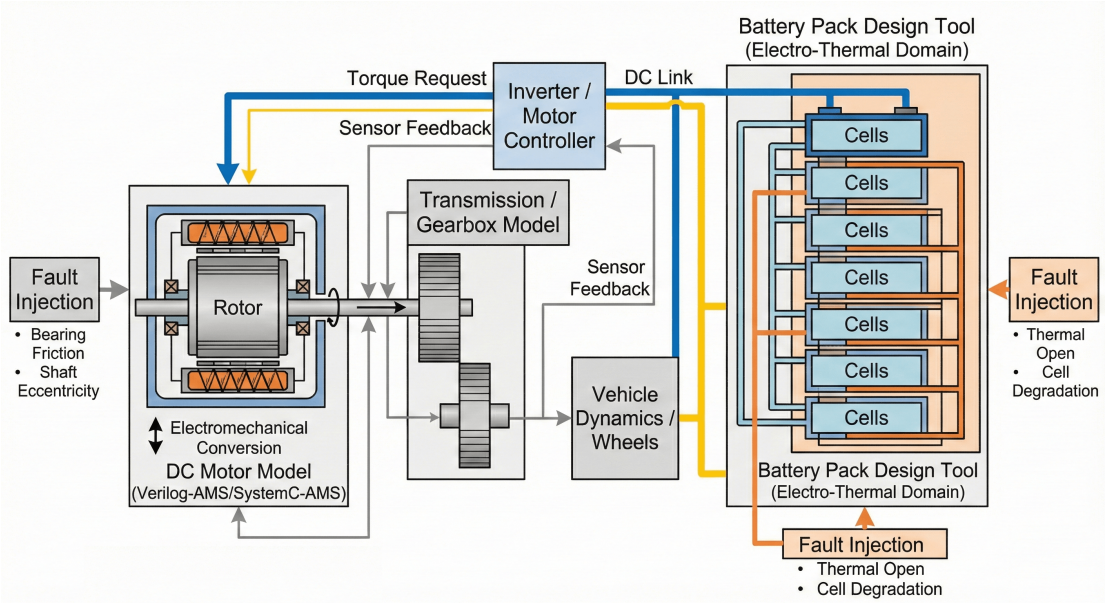


Fig. 5.2: First of the Unified Electrical Vehicle Digital Twin submodel: the powertrain, which includes the motor and the battery pack.

in the battery (e.g., blocked coolant channel) limits the power available to the motor. This coupling enables the analysis of cascading faults, such as how a thermal fault in the battery cooling system might force the powertrain into a degraded "limp mode" to prevent thermal runaway.

5.3.2 The Perception Sensors Subsystem

Sensors are crucial in modern cars, and ensuring they function properly is almost as important as installing them in the first place. The vehicle's stability control (ESP) and active suspension rely on MEMS Accelerometers. The model validated in Chapter 3 can be integrated into the chassis to provide raw sensor data. Obviously, there are not only accelerometers in a car, but a large number of them. Other types of sensors can be modeled similarly to the accelerometer to make the model more complex. The MEMSs are subject to the mechanical vibrations, *e.g.*, generated by the faulty motor model. Using the calibration methodology from Chapter 4, the virtual sensors are continuously tested against "phantom" accelerations caused by internal electrical defects (e.g., Current Source injection in the capacitive bridge), ensuring the control logic can distinguish between a pot-hole and a sensor glitch. Fig. 5.3 gives an overview of this part of the overall Unified Electrical Vehicle Digital Twin model.

5.3.3 Real-world Interface

Let us now focus on what interacts with the car externally: the driver and the surrounding environment, both of which are components related to Unreal Engine. The IMHU Framework (Section 4.7) can be integrated into the cabin monitoring system (see Fig. 5.4). The driver is no longer an external variable but a modeled component of the control loop. Therefore, the

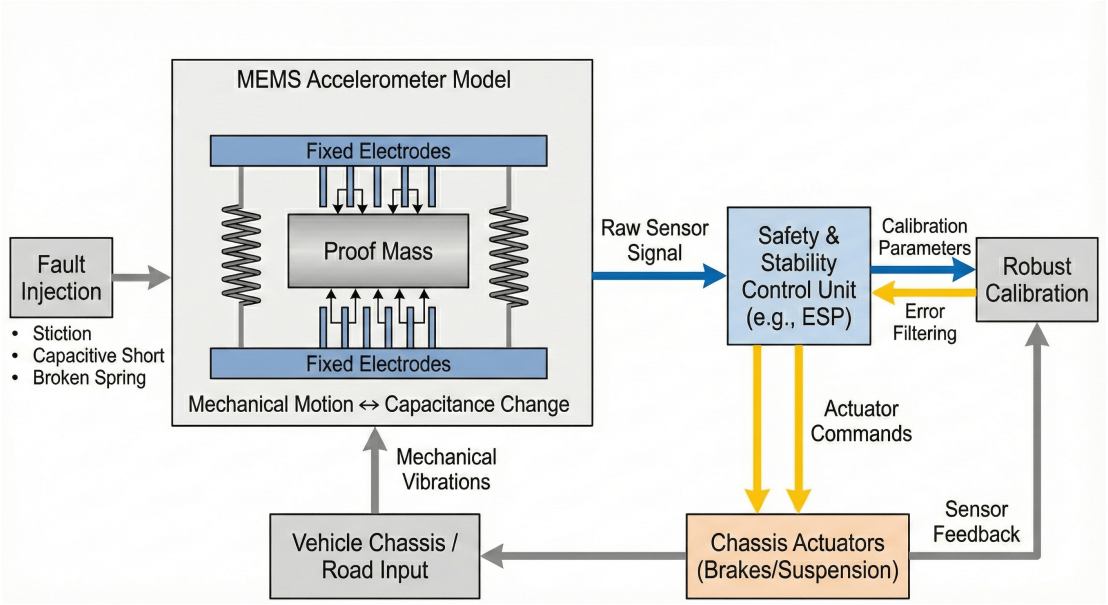


Fig. 5.3: Second of the Unified Electrical Vehicle Digital Twin submodel: the MEMS sensors, which represent a big component of nowadays cars.

resulting "Driver Digital Twin" monitors the driver's physiological states. Note: this feature could become useful even on autonomous drive vehicles. In this example, if the passenger falls asleep, the car may still move, but the vehicle would be immediately notified that no human is checking the system anymore. If the Drowsiness parameter exceeds a threshold, the system interacts with the Powertrain (*e.g.*, increasing regenerative braking to wake the driver) or the Advanced Driver Assistance Systems (ADAS).

Finally, the environmental integration used for the Drone case study (Chapter 3) provides the spatial context. It simulates the physical world, road friction, weather conditions, wind resistance, other cars or vehicles, and obstacles, closing the loop between the vehicle's physics and its environment.

5.4 Dual-Purpose Utilization: from Development to Operation

Considering the nature of the model, the true value of the proposed Unified Digital Twin concept lies in its versatility. The same underlying multidomain fault models can serve two distinct phases of the vehicle's lifecycle. On one hand, during development, the Unified Model acts as a virtual testbed. *e.g.*, engineers can use the Automated Design Exploration capabilities to optimize the battery cooling layout specifically for the heat profiles generated by the specific motor chosen for the powertrain. Furthermore, the Contract-Based Monitoring framework can be stress-tested. By injecting thousands of multi-domain faults, engineers can mathematically verify that the safety monitors will trigger correctly in edge-case scenarios that are tricky to recreate on a physical track.

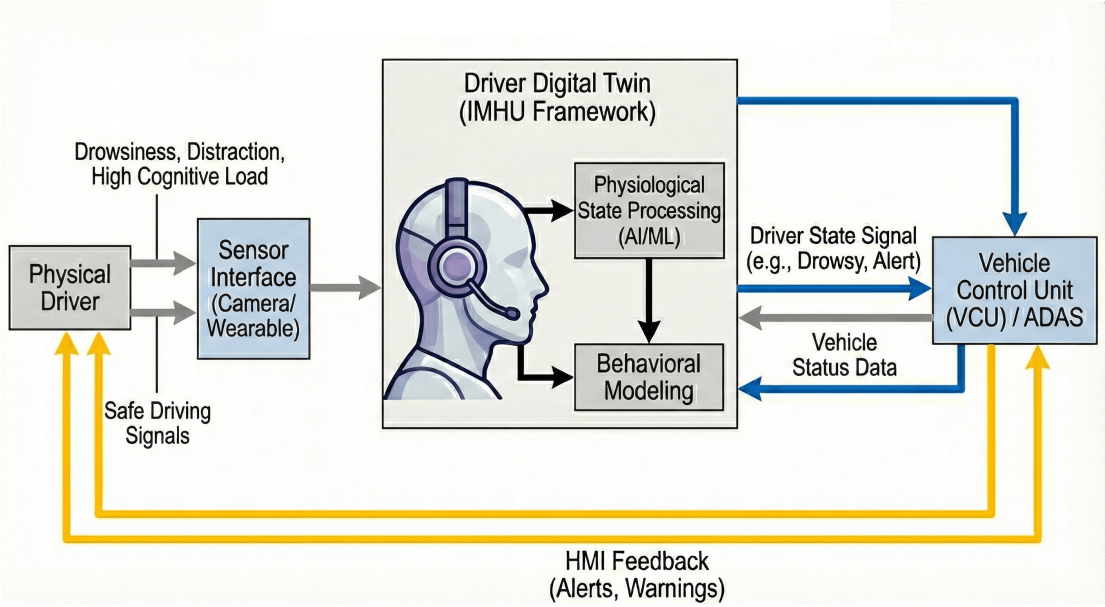


Fig. 5.4: Third of the Unified Electrical Vehicle Digital Twin submodel: the IMHU methodology applied to the driver and/or passenger of the car.

On the other hand, once the vehicle is on the road, a lightweight version of this Unified Model can run on the vehicle's High-Performance Computing (HPC) unit or in the cloud as a Shadow Mode Digital Twin. The model would continuously compare real telemetry with the simulation. A deviation in the motor's vibration profile, correlated with a specific thermal pattern in the battery, could trigger a PdM alert for a specific mechanical bearing fault weeks before it becomes audible to the driver. Regarding the driver, if the Human Digital Twin detects high distraction, the vehicle can instantaneously update the safety contracts of the powertrain (e.g., limiting max torque), effectively re-configuring the car's physics to match the driver's capability in real-time.

5.5 Challenges in realizing the unified digital twin

The conceptual architecture of the Unified EV digital twin presented in this chapter represents a significant leap in simulation complexity. Moving from the theoretical design to a fully operational implementation introduces several technical challenges that define the roadmap for future research.

5.5.1 Computational complexity and real-time constraints

Integrating high-fidelity physics (SystemC AMS/Verilog-AMS), Artificial Intelligence (AI) inference (YOLOv8), and photorealistic rendering (Unreal Engine) into a single closed-loop simulation imposes a heavy computational burden. Achieving real-time performance (essential for Human-in-the-Loop scenarios) may require advanced Model Order Reduction (MOR)

techniques or distributing the simulation load across heterogeneous hardware (*e.g.*, FPGA for physics, GPU for AI and rendering).

5.5.2 Synchronization and latency

The proposed architecture relies on the co-simulation of diverse Models of Computation (MoCs): the discrete-event nature of the digital controllers, the continuous-time solvers for the physical plant, and the frame-based execution of the game engine. Maintaining strict temporal synchronization between these domains is crucial for achieving a fine simulation. Specifically, the latency introduced by data exchange interfaces (*e.g.*, TCP/IP sockets between Unreal Engine and SystemC AMS) must be rigorously managed.

5.6 Summary

This conceptual case study demonstrates that the heterogeneous modeling methodologies developed in this thesis are not isolated academic exercises. When integrated, they form the blueprint for a next-generation Automotive Digital Twin. By simulating the interplay between electrical faults, mechanical degradation, thermal limits, and human physiology, we build the way for EV that are not only efficient but intrinsically self-aware and resilient.

6

Conclusion and suggestions for future research

To conclude this thesis, let us summarize the methodologies that have been proposed throughout this journey and suggest directions of future research that build on the proposed approaches.

6.1 Summary of the proposed approach

Nowadays, the functional safety of Industrial Cyber-Physical Systems (ICPSs) is no longer a challenge confined to the reliability of isolated electronic components. As Industry 4.0 matures into the human-centric paradigm of Industry 5.0, the definition of a "system" has expanded to encompass multi-physics hardware, complex control software, and the interaction of human operators cannot be ignored anymore. Therefore, not only do the components involved increase in number and complexity, but they also interact directly with each other. This thesis emphasizes the importance of linking, for example, physical domains within a simulation model, rather than just modeling them as individual components. We have seen how the electrical or mechanical domain is closely linked to the thermal domain, especially in terms of safety-critical conditions. This thesis aims to address the critical need for a unified simulation framework that can analyze reliability and safety across this heterogeneous landscape.

Starting from this system's complexity, we initially sought the simplest way to represent all these components in a single model. The central hypothesis of this research was that the mathematical isomorphism existing between physical domains could be exploited to bridge the gap between physical defects and behavioral simulation. By leveraging Physical Analogies, we have demonstrated that it is possible to systematically infer mechanical and thermal fault models from well-established electrical standards.

This work has successfully transitioned this theoretical concept into a practical engineering toolset. After deriving the mechanical and thermal fault taxonomies, we began to create models of real systems. By injecting and simulating the fault models obtained, we were able to test them across a very diverse set of case studies. In addition, we tested the methodologies using different system modeling languages, starting with Verilog-AMS and SystemC AMS, as well as MATLAB/Simulink and Unreal Engine. Each of these languages has its own unique characteristics and peculiarities. Given the variety that characterizes today's programming world, testing the methodology on different languages seemed an interesting choice, as well as making the

methodology itself independent of any language. After rigorously validating the faults across diverse simulation environments, we applied them to several safety-related industrial applications. The most immediate development was to utilize the faulty time series generated by the simulations to construct behavioral monitors that would oversee system behavior. Another use of the faulty time series was to create a device for testing the calibration of a Micro Electro Mechanical Systems (MEMS) sensor. By simulating the faulty behavior of the sensor, it is possible to better calibrate the activation thresholds of an actuator, even in the event of a malfunction. Having analyzed electric motors and sensors, we decided to also focus on batteries. First, we tested faults, particularly thermal faults, on batteries. Second, we observed that in most cases, the battery pack is typically represented as a single large cell. Therefore, we developed a tool to support battery pack simulation and modeling of cell-to-cell interactions. In this way, it is possible to have a functional safety-oriented thermal simulation of the battery pack. During these tests and the production of faulty data sets, we wondered whether it would be possible to visualize our case studies in 3D. For this reason, we decided to test our methodologies on Unreal Engine. Starting with the DC motor and progressing to the development of a multirotor drone model and its safety mechanisms, we added an extra layer of quality to the fault simulation. Additionally, we developed a digital twin of the human face to recognize fatigue in industrial operators. If we consider fatigue or distraction as a fault, then we can say that this was an additional step forward in terms of human safety in an industrial context. Finally, to demonstrate that all the work done is not an end in itself, I proposed a way to collect all the proposed models within a single framework. The idea, as a starting point for the future, is to create a simulation model that contains most of the main components of an Electric Vehicle (EV). This concept was designed to be used both as a model in the design phase and as a digital twin, considering, for example, contract-based runtime monitors. Ultimately, this thesis offers a comprehensive solution to the problem of multi-domain simulability.

6.2 Directions for future research

New opportunities arise from the work of this thesis, some born from the limitations of the proposed techniques, and others from curiosity and eagerness to improve and explore. Follows a list of future research linked to this work, some of which we are already under study:

- The actual implementation of the concept shown in Chapter 5, connecting all the modules that comprise it and testing its performance in simulation.
- Although the accuracy of the proposed models has always been of paramount importance to us and our work, further validation of the fault models on the real counterparts of the models is necessary. Despite the fact that the results are qualitatively satisfactory, a comparison of the simulated conditions with real objects is currently underway.
- Contract-based runtime monitors are a good way to perform fault detection, but our research now focuses on ways to generate them quickly and/or automatically. Additionally, a comparison with other Machine Learning (ML)-based fault detection techniques is a necessary development, given the importance of Artificial Intelligence (AI) today.

- Given the importance of batteries today, especially in the automotive sector, development of the automatic tool continues. We plan to model other types of coolants and cell layouts. The modeling and integration of a Battery Management System (BMS) is another promising development, as it is a fundamental component of battery packs. It is responsible, for example, for cell balancing and managing shutdown procedures in the event of an emergency (crucial for safety studies).

Summary of the proposed innovative contributions

This chapter reports the innovative contributions to the state of the art by the works proposed in this thesis, organized in chronological order. The articles are grouped into four main topics as follows:

- the first group is related to physical analogies and the *novel fault taxonomy* produced, which comprises Chapter 2;
- the second group shows the *validation of the presented fault models* through different case studies, reported in Chapter 3;
- the final group comprises various solutions that utilize fault models to achieve *functional safety-related solutions*, which are all listed in Chapter 4.

Fault Modeling through Physical Analogies

1. N. Dall'Ora, **F. Tosoni**, E. Fraccaroli and F. Fummi, "Inferring Mechanical Fault Models from the Electrical Domain," 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS), Coventry, United Kingdom, 2022, pp. 01-08, doi: 10.1109/ICPS51978.2022.9817009.
2. **F. Tosoni**, N. Dall'Ora, E. Fraccaroli and F. Fummi, "A Framework for Modeling and Concurrently Simulating Mechanical and Electrical Faults in Verilog-AMS," 2022 Forum on Specification & Design Languages (FDL), Linz, Austria, 2022, pp. 1-8, doi: 10.1109/FDL56239.2022.9925655.
3. **F. Tosoni**, N. Dall'Ora, E. Fraccaroli, S. Vinco and F. Fummi, "Thermal Digital Twin of a Multi-Domain System for Discovering Mechanical Faulty Behaviors," 2023 IEEE 21st International Conference on Industrial Informatics (INDIN), Lemgo, Germany, 2023, pp. 1-8, doi: 10.1109/INDIN51400.2023.10218266.
4. **F. Tosoni**, N. Dall'Ora, E. Fraccaroli, S. Vinco and F. Fummi, "Multidomain Fault Models Covering the Analog Side of a Smart or Cyber–Physical System," in IEEE Transactions on Computers, vol. 73, no. 3, pp. 829-841, March 2024, doi: 10.1109/TC.2023.3345135.

Fault Models Injection, Simulation, and Validation

5. **F. Tosoni**, N. Dall'Ora, E. Fraccaroli, S. Vinco and F. Fummi, "Assessing Robustness of Smart Systems via Multi-domain Analog Fault Simulation," 2024 IEEE 30th International

Symposium on On-Line Testing and Robust System Design (IOLTS), Rennes, France, 2024, pp. 1-3, doi: 10.1109/IOLTS60994.2024.10616082.
6. F. Biondani, N. Dall'Ora, **F. Tosoni** et al., "Fault Injection for Synthetic Data Generation in Aircraft: A Simulation-Based Approach," 2024 IEEE 22nd International Conference on Industrial Informatics (INDIN), Beijing, China, 2024, pp. 1-8, doi: 10.1109/INDIN58382.2024.10774347.
7. **F. Tosoni**, N. Dall'Ora, E. Fraccaroli, S. Vinco and F. Fummi, "Cross-domain Analog Fault Injection for Designing Robust Smart Systems," 2024 Forum on Specification & Design Languages (FDL), Stockholm, Sweden, 2024, pp. 1-8, doi: 10.1109/FDL63219.2024.10673865.
8. **F. Tosoni**, S. Vinco and F. Fummi, "Modeling and Simulation of Thermal Faults in Batteries for Enhanced Safety," 2025 IEEE 28th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Lyon, France, 2025, pp. 115-118, doi: 10.1109/DDECS63720.2025.11006771.
9. R. Ni, G. Pollo, **F. Tosoni** et al., "A Multi-domain Simulation Framework for Enhanced Drone Design and Fault Management," submitted to ACM Trans. Cyber-Phys. Syst. (2025).

Application of the Fault Taxonomy and Techniques to Real World Solutions

9. **F. Tosoni**, N. Dall'Ora, E. Fraccaroli and F. Fummi, "The Challenges of Coupling Digital-Twins with Multiple Classes of Faults," 2022 IEEE 23rd Latin American Test Symposium (LATS), Montevideo, Uruguay, 2022, pp. 1-6, doi: 10.1109/LATS57337.2022.9937026.
10. A. Beghi, **F. Tosoni** et al., "VIR2EM: VIrtualization and Remotization for Resilient and Efficient Manufacturing: Project-Dissemination Paper," 2023 Forum on Specification & Design Languages (FDL), Turin, Italy, 2023, pp. 1-8, doi: 10.1109/FDL59689.2023.10272156.
11. **F. Tosoni**, M. I. Amin, N. Dall'Ora, E. Fraccaroli and F. Fummi, "Exploring Multidomain Faults in Digital Twin: A Gaming Engine Perspective: Wild-and-Crazy-Idea Paper," 2024 Forum on Specification & Design Languages (FDL), Stockholm, Sweden, 2024, pp. 1-5, doi: 10.1109/FDL63219.2024.10673841.
12. F. Bruns, **F. Tosoni**, A. Rauh, F. Fummi, S. Mehlhop and F. Oppenheimer, "Analyzing Fault Behaviors in Multi-Domain Systems with Contract-Based Monitors," 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), Padova, Italy, 2024, pp. 1-8, doi: 10.1109/ETFA61755.2024.10710990.
13. F. Bruns, **F. Tosoni** et al., "Validating the Design of CPS: Interfacing Simulations of Multi-Physics Components and Software with Contract-Based Monitoring," 2025 29th International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland, 2025, pp. 35-40, doi: 10.1109/MMAR65820.2025.11150819.
14. F. Biondani, **F. Tosoni** et al., "Special Session: D-MATE: A Design Methodology for Connecting Automatic Test Equipment in Industry 4.0," 2025 IEEE 26th Latin American Test Symposium (LATS), San Andres Islas, Colombia, 2025, pp. 1-6, doi: 10.1109/LATS65346.2025.10963954.
15. **F. Tosoni**, Y. Chen et al., "An Open Source Design Exploration Tool for Battery and Coolant Configuration," accepted to be published in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2026.

16. **F. Tosoni**, Y. Chen et al., "Accelerating Battery Pack Design via Modular Electro-Thermal Simulation Frameworks," submitted to IEEE 35th International Symposium on Industrial Electronics (ISIE), 2026.
17. F. Biondani, **F. Tosoni** et al., "MATE: A Methodology for Connecting Automatic Test Equipment in Industry 4.0," under submission to IEEE Access, 2025.
18. F. Biondani, L. Capogrosso, **F. Tosoni** et al., "Human-Centered Digital Twin Framework for Industry 5.0," under submission to Journal of Manufacturing Systems, 2025.

References

1. *ISO 26262 – Road vehicles – Functional safety*, ISO, 2018.
2. E. Fraccaroli and S. Vinco, "Modeling cyber-physical production systems with systemc-ams," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 2039–2051, 2023.
3. M. Lora, S. Vinco, and F. Fummi, "Translation, abstraction and integration for effective smart system design," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1525–1538, 2019.
4. A. C. Fernandez-Pello, A. P. Pisano, K. Fu, D. C. Walther, A. Knobloch, F. Martinez, M. Senesky, C. Stoldt, R. Maboudian, S. Sanders, and D. Liepmann, "Mems rotary engine power system," *IEEJ Transactions on Sensors and Micromachines*, vol. 123, no. 9, pp. 326–330, 2003.
5. Z. Huang, C. Wang, M. Stojmenovic, and A. Nayak, "Characterization of cascading failures in interdependent cyber-physical systems," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2158–2168, 2015.
6. C. Bolchini and A. Miele, "Reliability-driven system-level synthesis for mixed-critical embedded systems," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2489–2502, 2013.
7. M. Moradi, B. Van Acker, and J. Denil, "Failure identification using model-implemented fault injection with domain knowledge-guided reinforcement learning," *Sensors*, vol. 23, no. 4, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/4/2166
8. I. Pomeranz and S. Reddy, "Location of stuck-at faults and bridging faults based on circuit partitioning," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1124–1135, 1998.
9. R. Stetter, "Approaches for modelling the physical behavior of technical systems on the example of wind turbines," *Energies*, vol. 13, no. 8, p. 2087, Apr. 2020. [Online]. Available: https://doi.org/10.3390/en13082087
10. X. Yang, C. Yang, T. Peng, Z. Chen, B. Liu, and W. Gui, "Hardware-in-the-loop fault injection for traction control system," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 6, no. 2, pp. 696–706, 2018.
11. M. Fernandes, J. M. Corchado, and G. Marreiros, "Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: a systematic literature review," *Applied Intelligence*, vol. 52, no. 12, pp. 14 246–14 280, 2022.
12. L. Tang, H. Tian, H. Huang, S. Shi, and Q. Ji, "A survey of mechanical fault diagnosis based on audio signal analysis," *Measurement*, p. 113294, 2023.
13. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.
14. I. S. Association, *IEEE-SA 2427 Standard for Analog Defect Modeling and Coverage*. IEEE, 2025. [Online]. Available: https://standards.ieee.org/project/2427.html
15. E. Balaban, P. Bansal, P. Stoelting, A. Saxena, K. F. Goebel, and S. Curran, "A diagnostic approach for electro-mechanical actuators in aerospace systems," in *2009 IEEE Aerospace conference*. IEEE, 2009, pp. 1–13.
16. S. J. Uder, R. B. Stone, and I. Y. Tumer, "Failure analysis in subsystem design for space missions," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 46962, 2004, pp. 201–217.
17. J. Bougie and A. Gangopadhyaya, "Conservation laws and energy transformations in a class of common physics problems," *American Journal of Physics*, vol. 87, no. 11, pp. 868–874, nov 2019.

18. N. Hogan and P. C. Breedveld, "The physical basis of analogies in physical system models," in *The mechatronics handbook*. CRC Press/Balkema, 2002.
19. R. R. Thakkar, "Electrical equivalent circuit models of lithium-ion battery," *Management and Applications of Energy Storage Devices*, 2021.
20. C.-A. Chen, X. Li, L. Zuo, and K. D. Ngo, "Circuit modeling of the mechanical-motion rectifier for electrical simulation of ocean wave power takeoff," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 4, pp. 3262–3272, 2020.
21. P. Gardonio and M. J. Brennan, "On the origins and development of mobility and impedance methods in structural dynamics," *Journal of Sound and vibration*, vol. 249, no. 3, pp. 557–573, 2002.
22. J. López-Martínez, J. C. Martínez, D. García-Vallejo, A. Alcayde, and F. G. Montoya, "A new electromechanical analogy approach based on electrostatic coupling for vertical dynamic analysis of planar vehicle models," *IEEE Access*, vol. 9, pp. 119 492–119 502, 2021.
23. M. Smith, "Synthesis of mechanical networks: the inerter," *IEEE Transactions on Automatic Control*, vol. 47, no. 10, pp. 1648–1662, 2002.
24. N. Dall'Ora, F. Tosoni, E. Fraccaroli, and F. Fummi, "Inferring Mechanical Fault Models from the Electrical Domain," in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2022, pp. 01–08.
25. F. Tosoni, N. Dall'Ora, E. Fraccaroli, and F. Fummi, "A Framework for Modeling and Concurrently Simulating Mechanical and Electrical Faults in Verilog-AMS," in *2022 Forum on Specification & Design Languages (FDL)*, 2022, pp. 1–8.
26. N. Bombieri, G. D. Guglielmo, L. D. Guglielmo, M. Ferrari, F. Fummi, G. Pravadelli, F. Stefanni, and A. Venturelli, "HIFSuite: Tools for HDL code conversion and manipulation," in *2010 IEEE International High Level Design Validation and Test Workshop (HLDVT)*. IEEE, Jun. 2010.
27. H.-Y. Lin, C.-Y. Wang, S.-C. Chang, Y.-C. Chen, H.-M. Chou, C.-Y. Huang, Y.-C. Yang, and C.-C. Shen, "A probabilistic analysis method for functional qualification under mutation analysis," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 147–152.
28. M. Hassan, D. Große, H. M. Le, T. Vörtler, K. Einwich, and R. Drechsler, "Testbench qualification for SystemC-AMS timed data flow models," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 857–860.
29. N. Bombieri, F. Fummi, V. Guarnieri, and G. Pravadelli, "Testbench Qualification of SystemC TLM Protocols through Mutation Analysis," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1248–1261, 2014.
30. Y. Chen, S. Vinco, E. Macii, and M. Poncino, "Fast thermal simulation using systemc-ams," in *Proc. of the Great Lakes Symposium on VLSI*, 2016, p. 427–432.
31. K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," *SIGARCH Comput. Archit. News*, vol. 31, no. 2, p. 2–13, 2003.
32. N. An, M. Du, Z. Hu, and K. Wei, "A high-precision adaptive thermal network model for monitoring of temperature variations in insulated gate bipolar transistor (igbt) modules," *Energies*, vol. 11, no. 3, p. 595, 2018.
33. O. Alavi, M. Abdollah, and A. H. Viki, "Assessment of thermal network models for estimating igbt junction temperature of a buck converter," in *2017 8th Power Electronics, Drive Systems & Technologies Conference (PEDSTC)*. IEEE, 2017, pp. 102–107.
34. D. Troncon and L. Alberti, "Case of study of the electrification of a tractor: Electric motor performance requirements and design," *Energies*, vol. 13, no. 9, p. 2197, 2020.
35. G. Swift, T. S. Molinski, and W. Lehn, "A fundamental approach to transformer thermal modeling. i. theory and equivalent circuit," *IEEE transactions on Power Delivery*, vol. 16, no. 2, pp. 171–175, 2001.
36. K. Murthy and R. Bedford, "Transformation between foster and cauer equivalent networks," *IEEE Transactions on Circuits and Systems*, vol. 25, no. 4, pp. 238–239, 1978.
37. L. Mo, T. Zhang, and Q. Lu, "Thermal analysis of a flux-switching permanent-magnet double-rotor machine with a 3-d thermal network model," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 2, pp. 1–5, 2019.
38. J. W. Li, W. J. Zhang, G. S. Yang, S. D. Tu, and X. B. Chen, "Thermal-error modeling for complex physical systems: the-state-of-arts review," *The International Journal of Advanced Manufacturing Technology*, vol. 42, no. 1-2, pp. 168–179, Jun. 2008.

39. Accellaera, “Ieee standard for standard systemc® language reference manual,” *IEEE Std 1666-2023 (Revision of IEEE Std 1666-2011)*, pp. 1–618, 2023.
40. Accellera, “Ieee standard for standard systemc(r) analog/mixed-signal extensions language reference manual,” *IEEE Std 1666.1-2016*, pp. 1–236, 2016.
41. N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, “Gvsoc: A highly configurable, fast and accurate full-platform simulator for risc-v based iot processors,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 409–416.
42. D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, and A. Marongiu, “Energy efficient parallel computing on the pulp platform with support for openmp,” in *2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, 2014.
43. E. Games, “Unreal Engine,” https://www.unrealengine.com/, 2025.
44. X. Yang, D. Zou, L. Pei, D. Sartori, and W. Yu, “An efficient simulation platform for testing and validating autonomous navigation algorithms for multi-rotor uavs based on unreal engine,” in *China Satellite Navigation Conference (CSNC) 2019 Proceedings*, J. Sun, C. Yang, and Y. Yang, Eds. Singapore: Springer Singapore, 2019, pp. 527–539.
45. D. L. Smyth, F. G. Glavin, and M. G. Madden, “Using a game engine to simulate critical incidents and data collection by autonomous drones,” in *2018 IEEE Games, Entertainment, Media Conference (GEM)*, 2018, pp. 1–9.
46. I. Hallinan, T. Yuan, B. Geng, and X. Deng, “Realistic 3D drone simulation with path-planning in unreal engine 5,” https://irina694.github.io/3d-drone-sim/.
47. E. Games. (2025) Unreal engine blueprints. Accessed 7-May-2025. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine
48. ——. (2025) Unreal engine c++ programming. Accessed 7-May-2025. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-with-cplusplus-in-unreal-engine
49. T. M. Inc., “Matlab version: 24.1.0.2498408 (r2024a),” Natick, Massachusetts, United States, 2024. [Online]. Available: https://www.mathworks.com
50. H. Klee and R. Allen, *Simulation of dynamic systems with MATLAB® and Simulink®*. Crc Press, 2018.
51. Airbus, “Airbus Defence and Space Establishes Framework for Digital Twin Aircraft,” accessed on Feb 27, 2024. [Online]. Available: https://it.mathworks.com/company/user_stories/case-studies/airbus-defence-and-space-establishes-framework-for-digital-twin.html
52. E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.
53. G. Zorpette, “Rf mems deliver the "ideal switch": After two decades of development, mems-based rf switches are finally finding real-world uses,” *IEEE Spectrum*, vol. 57, no. 8, pp. 8–9, 2020.
54. D. A. Barkas, G. C. Ioannidis, C. S. Psomopoulos, S. D. Kaminaris, and G. A. Vokas, “Brushed DC motor drives for industrial and automobile applications with emphasis on control techniques: A comprehensive review,” *Electronics*, vol. 9, no. 6, p. 887, May 2020.
55. S.-H. Yen, P.-C. Tang, Y.-C. Lin, and C.-Y. Lin, “A sensorless and low-gain brushless DC motor controller using a simplified dynamic force compensator for robot arm application,” *Sensors*, vol. 19, no. 14, p. 3171, Jul. 2019.
56. “Ensuring functional safety for self-driving cars,” https://semiengineering.com/ensuring-functional-safety-for-self-driving-cars/, 2019.
57. K. Nishi, T. Hatakeyama, S. Nakagawa, and M. Ishizuka, “Transient thermal analysis of the microprocessor system one-dimensional thermal network with power estimation equation,” in *Fourteenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2014, pp. 982–989.
58. J. V. Sørensen, Z. Ma, and B. N. Jørgensen, “Potentials of game engines for wind power digital twin development: an investigation of the unreal engine,” *Energy Informatics*, vol. 5, no. Suppl 4, p. 39, 2022.
59. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
60. K. H. Sharif and S. Yousif Ameen, “Game Engines Evaluation for Serious Game Development in Education,” in *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2021, pp. 1–6.

61. S. Robyns, W. Heerwegh, and S. Weckx, "A digital twin of an off highway vehicle based on a low cost camera," *Procedia Computer Science*, vol. 232, pp. 2366–2375, 2024, 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924002321
62. Y.-L. Xin, G.-P. Ge, W. Du, H. Wu, and Y. Zhao, "Design of an Optical Physics Virtual Simulation System Based on Unreal Engine 5," *Applied Sciences*, vol. 14, no. 3, 2024.
63. M. Matulis and C. Harvey, "A robot arm digital twin utilising reinforcement learning," *Computers & Graphics*, vol. 95, pp. 106–114, 2021.
64. T. I. Erdei, R. Krakó, and G. Husi, "Design of a Digital Twin Training Centre for an Industrial Robot Arm," *Applied Sciences*, vol. 12, no. 17, 2022.
65. A. D'Alessandro, S. Scudero, and G. Vitale, "A review of the capacitive mems for seismology," *Sensors*, vol. 19, no. 14, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/14/3093
66. F. Pecheux, C. Lallement, and A. Vachoux, "VHDL-AMS and verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, 2005.
67. G. Roloff, "The Evolution of a System Aircraft Landing Gear," Airbus-Deutschland GmbH, Tech. Rep., 2002.
68. F. A. ADMINISTRATION, "Aviation Maintenance Technician Handbook–Airframe," U.S. Department of Transportation FEDERAL AVIATION ADMINISTRATION, Tech. Rep., 2018.
69. R. Ahmad and S. Kamaruddin, "An overview of time-based and condition-based maintenance in industrial application," *Computers & Industrial Engineering*, vol. 63, no. 1, pp. 135–149, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835212000484
70. W. J. C. Verhagen, B. F. Santos, F. Freeman, P. van Kessel, D. Zarouchas, T. Loutas, R. C. K. Yeun, and I. Heiets, "Condition-based maintenance in aviation: Challenges and opportunities," *Aerospace*, vol. 10, no. 9, 2023. [Online]. Available: https://www.mdpi.com/2226-4310/10/9/762
71. C. Windmeijer, "A paradigm shift to more efficient aircraft fleet maintenance," ReMAP's Public Relations, Tech. Rep., 2021.
72. I. Stanton, K. Munir, A. Ikram, and M. El-Bakry, "Predictive maintenance analytics and implementation for aircraft: Challenges and opportunities," *Systems Engineering*, vol. 26, no. 2, pp. 216–237, 2023.
73. F. Tosoni, N. Dall'Ora, E. Fraccaroli, S. Vinco, and F. Fummi, "Multidomain fault models covering the analog side of a smart or cyber–physical system," *IEEE Transactions on Computers*, vol. 73, no. 3, pp. 829–841, 2024.
74. AIRBUS, "A statistical analysis of commercial aviation accidents 1958-2023," Airbus, Tech. Rep., 2024.
75. Boeing, "Statisticalsummary of commercial jet airplane accidents worldwide operations 1959-2022," Boeing, Tech. Rep., 2023.
76. EASA, "Easa annual safety review 2023," European Union Aviation Safety Agency, Tech. Rep., 2023.
77. ICAO, "Statistical summary of commercial jet airplane accidents worldwide operations," International Civil Aviation Organization, Tech. Rep., 2022.
78. F. S. Foundation, "Aviation safety network," 2024, https://aviation-safety.net/ [Accessed: Feb 2024)].
79. N. S. D. G. mit beschränkter Haftung, "The aviation herald," 2024, https://avherald.com/ [Accessed: Feb 2024)].
80. Steve Miller, "Landing Gear Model in Simscape," accessed on Feb 27, 2024. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/47534-landing-gear-model-in-simscape
81. Y. Cao, S. Duan, Y. Li, X. Li, Z. Zhao, and X. Wang, "Fault detection of landing gear retraction/extension hydraulic system based on bond graph-linear fractional transformation technique and interval analytic redundancy relations," *Applied Sciences*, vol. 12, p. 9667, 09 2022.
82. S. Duan, Y. Li, Y. Cao, X. Wang, X. Li, and Z. Zhao, "Health assessment of landing gear retraction/extension hydraulic system based on improved risk coefficient and fce model," *Applied Sciences*, vol. 12, no. 11, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5409
83. The MathWorks Inc., "Translational Friction," accessed on Feb 27, 2024. [Online]. Available: https://it.mathworks.com/help/simscape/ref/translationalfriction.html
84. ——, "Rotational Friction," accessed on Feb 27, 2024. [Online]. Available: https://it.mathworks.com/help/simscape/ref/rotationalfriction.html

85. J.-S. Um, *Drones as Cyber-Physical Systems Concepts and Applications for the Fourth Industrial Revolution*. Springer, 2019.
86. A. Mairaj, A. I. Baba, and A. Y. Javaid, "Application specific drone simulators: Recent advances and challenges," *Simulation Modelling Practice and Theory*, vol. 94, pp. 100–117, 2019.
87. P. Boccadoro, D. Striccoli, and L. A. Grieco, "An extensive survey on the internet of drones," *Ad Hoc Networks*, vol. 122, p. 102600, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870521001335
88. A. Rejeb, A. Abdollahi, K. Rejeb, and H. Treiblmaier, "Drones in agriculture: A review and bibliometric analysis," *Computers and Electronics in Agriculture*, vol. 198, p. 107017, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169922003349
89. A. Gohari, A. B. Ahmad, R. B. A. Rahim, A. S. M. Supa'at, S. Abd Razak, and M. S. M. Gismalla, "Involvement of surveillance drones in smart cities: A systematic review," *IEEE Access*, vol. 10, pp. 56 611–56 628, 2022.
90. J. Glossner, S. Murphy, and D. Iancu, "An overview of the drone open-source ecosystem https://arxiv.org/abs/2110.02260," 2021.
91. ArduPilot. (2025) Ardupilot. Accessed 14-April-2025. [Online]. Available: https://ardupilot.org
92. Cyberbotics. (2025) Webots. Accessed 14-April-2025. [Online]. Available: http://www.cyberbotics.com
93. O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.
94. O. Robotics. (2025) Gazebo. Accessed 14-April-2025. [Online]. Available: https://gazebosim.org
95. FlightGear. (2025) Flightgear. Accessed 14-April-2025. [Online]. Available: https://www.flightgear.org
96. M. Research. (2025) Airsim. Accessed 14-April-2025. [Online]. Available: https://microsoft.github.io/AirSim
97. N. Michel, P. Wei, Z. Kong, A. K. Sinha, and X. Lin, "Modeling and validation of electric multirotor unmanned aerial vehicle system energy dynamics," *eTransportation*, vol. 12, p. 100173, 2022.
98. G. Schweiger, C. Gomes, G. Engel, J.-P. Schoeggl, A. Posch, I. Hafner, and T. Nouidu, "An empirical survey on co-simulation: Promising standards, challenges and research needs," *Simulation Modelling Practice and Theory*, vol. 95, 2019.
99. F. Cappuzzo, V. Dezobry, D. Bianchi, and S. DiGennaro, "A novel co-simulation framework for verification and validation of GNC algorithms for autonomous UAV," in *Proc. of Vertical Flight Society's 78th Annual Forum*, 2022.
100. MathWorks. (2025) Simulink. Accessed 14-April-2025. [Online]. Available: https://www.mathworks.com/products/simulink.html
101. Ansys, "Navigating the future: autonomous systems for aerospace and defense," https://www.ansys.com/, 2025.
102. D. Systèmes. (2025) Soldiworks. Accessed 14-April-2025. [Online]. Available: https://www.solidworks.com/
103. OpenRocket. (2025) Openrocket. Accessed 14-April-2025. [Online]. Available: https://openrocket.info/
104. U. Technologies, "ML-Agents," https://docs.unity3d.com/Packages/com.unity.ml-agents@3.0/manual/index.html, 2024.
105. J. Park, R. Ivanov, J. Weimer, M. Pajic, S. H. Son, and I. Lee, "Security of cyber-physical systems in the presence of transient sensor faults," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, no. 3, May 2017. [Online]. Available: https://doi.org/10.1145/3064809
106. S. Safari, M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, and J. Henkel, "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues," *IEEE Access*, vol. 10, pp. 12 229–12 251, 2022.
107. M. Saied, H. Shraim, and C. Francis, "A review on recent development of multirotor uav fault-tolerant control systems," *IEEE Aerospace and Electronic Systems Magazine*, vol. 39, no. 9, pp. 146–180, 2024.
108. J. Mabboux, V. Pommier-Budinger, S. Delbecq, and J. Bordeneuve-Guibe, "Co-design of a multirotor uav with robust control considering handling qualities and motor failure," *Aerospace Science and Technology*, vol. 144, p. 108778, 2024.
109. *IEC 61508 – Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*, IEC, 2010.

110. R. Altawy and A. M. Youssef, "Security, privacy, and safety aspects of civilian drones: A survey," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, no. 2, Nov. 2016. [Online]. Available: https://doi.org/10.1145/3001836
111. N. Bombieri, F. Fummi, and G. Pravadelli, "Hardware design and simulation for verification," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2006, pp. 1–29.
112. C. Yao, J. Krieglstein, and K. Janschek, "Modeling and sliding mode control of a fully-actuated multirotor with tilted propellers," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 115–120, 2018, 12th IFAC Symposium on Robot Control SYROCO 2018.
113. D. Lawrence and K. Mohseni, "Efficiency analysis for long duration electric mavs," in *Infotech@ Aerospace*. AIAA, 2005, p. 7090.
114. M. Verhaegen, S. Kanev, R. Hallouzi, C. Jones, J. Maciejowski, and H. Smail, *Fault Tolerant Flight Control - A Survey*. Springer, 2010, pp. 47–89.
115. F. Prochazka, S. Kruger, G. Stomberg, and M. Bauer, "Development of a hardware-in-the-loop demonstrator for the validation of fault-tolerant control methods for a hybrid UAV," *CEAS Aeronautical Journal*, vol. 12, pp. 549–558, 2021.
116. M. A. Hamdi, G. Pollo, M. Risso, G. Haugou, A. Burrello, E. Macii, M. Poncino, S. Vinco, and D. J. Pagliari, "Integrating systemc-ams power modeling with a risc-v iss for virtual prototyping of battery-operated embedded devices," in *Proceedings of the 21st ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, 2024. [Online]. Available: https://doi.org/10.1145/3637543.3652873
117. Sony, "Sony battery," https://www.thunderheartreviews.com/2019/03/sony-vtc6-18650-3000mah-30a-test.html, 2025.
118. Samsung, "Samsung battery," https://citylion.pl/en/celle-samsung-inr21700-50s-4900mah-50s-35a/, 2025.
119. Tattu, "Tattu battery," https://www.aerialclick.com/tattu/353-tattu-16000-30.html, 2025.
120. Y. Chen, D. Baek, A. Bocca, A. Macii, E. Macii, and M. Poncino, "A case for a battery-aware model of drone energy consumption," in *Proc. of IEEE International Telecommunications Energy Conference (INTELEC)*, 2018.
121. F. Chen, W. Lei, K. Zhang, G. Tao, and B. Jiang, "A novel nonlinear resilient control for a quadrotor uav via backstepping control and nonlinear disturbance observer," *Nonlinear Dynamics*, vol. 85, no. 2, pp. 1281–1295, 2016.
122. J. K. Tan, A. K. Kamath, K. Singh, and M. Feroskhan, "Hexmorph: Fault tolerance against single and dual rotor failure using geometric morphing on hexacopter," *Robotics and Autonomous Systems*, vol. 193, p. 105047, 2025.
123. T. Zhao, Y. Zhang, M. Wang, W. Feng, S. Cao, and G. Wang, "A critical review on the battery system reliability of drone systems," *MDPI Drones*, vol. 9, 2025.
124. S. Wang, H. Wang, M. Chang, J. Xu, J. Wang, X. Yang, and J. Bai, "A novel battery thermal management system for an unmanned aerial vehicle using the graphene directional heat transfer structure," *Journal of Power Sources*, vol. 588, 2023.
125. B. Xu, J. Lee, D. Kwon, L. Kong, and M. Pecht, "Mitigation strategies for li-ion battery thermal runaway: A review," *Renewable and Sustainable Energy Reviews*, vol. 150, p. 111437, 2021.
126. Y. Chen, Y. Kang, Y. Zhao, L. Wang, J. Liu, Y. Li, Z. Liang, X. He, X. Li, N. Tavajohi *et al.*, "A review of lithium-ion battery safety concerns: The issues, strategies, and testing standards," *Journal of Energy Chemistry*, vol. 59, pp. 83–99, 2021.
127. A. Mauger and C. Julien, "Critical review on lithium-ion batteries: are they safe? sustainable?" *Ionics*, vol. 23, pp. 1933–1947, 2017.
128. S. Mallick and D. Gayen, "Thermal behaviour and thermal runaway propagation in lithium-ion battery systems–a critical review," *Journal of Energy Storage*, vol. 62, p. 106894, 2023.
129. P. Sun, R. Bisschop, H. Niu, and X. Huang, "A review of battery fires in electric vehicles," *Fire technology*, vol. 56, no. 4, pp. 1361–1410, 2020.
130. Y. Wang, Q. Gao, G. Wang, P. Lu, M. Zhao, and W. Bao, "A review on research status and key technologies of battery thermal management and its enhanced safety," *International Journal of Energy Research*, vol. 42, no. 13, pp. 4008–4033, 2018.

131. O. Erdinc, B. Vural, and M. Uzunoglu, "A dynamic lithium-ion battery model considering the effects of temperature and capacity fading," in *2009 International Conference on Clean Electrical Power*, 2009, pp. 383–386.
132. M. Petricca, D. Shin, A. Bocca, A. Macii, E. Macii, and M. Poncino, "An automated framework for generating variable-accuracy battery models from datasheet information," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 365–370.
133. L. Gao, S. Liu, and R. Dougal, "Dynamic lithium-ion battery model for system simulation," *IEEE Transactions on Components and Packaging Technologies*, vol. 25, no. 3, pp. 495–505, 2002.
134. P. Liu, S. Li, K. Jin, W. Fu, C. Wang, Z. Jia, L. Jiang, and Q. Wang, "Thermal runaway and fire behaviors of lithium iron phosphate battery induced by overheating and overcharging," *Fire Technology*, vol. 59, no. 3, pp. 1051–1072, 2023.
135. S. Dey, H. E. Perez, and S. J. Moura, "Model-based battery thermal fault diagnostics: Algorithms, analysis, and experiments," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 2, pp. 576–587, 2019.
136. Y.-J. Park, S.-K. S. Fan, and C.-Y. Hsu, "A Review on Fault Detection and Process Diagnostics in Industrial Processes," *Processes*, vol. 8, no. 9, 2020.
137. A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, *Contracts for System Design*. Now Foundations and Trends, 2018.
138. M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault-Tolerant Control*, 2nd ed. Springer Berlin, Heidelberg, 01 2006.
139. A. Nardi, S. Camdzic, A. Armato, and F. Lertora, "Design-for-safety for automotive ic design: Challenges and opportunities," *2019 IEEE Custom Integrated Circuits Conference (CICC)*, 2019.
140. E. Balaban, P. Bansal, P. Stoelting, A. Saxena, K. F. Goebel, and S. Curran, "A diagnostic approach for electromechanical actuators in aerospace systems," in *2009 IEEE Aerospace conference*, 2009.
141. S. Uder, R. Stone, Associate, and I. Tumer, "Failure analysis in subsystem design for space missions," in *ASME Design Theory and Methodology Conference*, vol. 3, 01 2004.
142. A. Kolesnikov, D. Tretsiak, and M. Cameron, "Systematic simulation of fault behavior by analysis of vehicle dynamics," in *Proceedings of the 13th International Modelica Conference, Regensburg, Germany,*, 02 2019, pp. 451–460.
143. J. Gundermann, A. Kolesnikov, M. Cameron, and T. Blochwitz, "The fault library - a new modelica library allows for the systematic simulation of non-nominal system behavior," *Proceedings of the 2nd Japanese Modelica Conference Tokyo, Japan*, 2019.
144. X. Pan, C. Zivkovic, and C. Grimm, "Virtual Prototyping of Heterogeneous Automotive Applications: Matlab, SystemC, or both?" in *24th Asia and South Pacific Design Automation Conference, Tokyo, Japan*, 2019, pp. 544–549.
145. A. Ballo, M. Bottaro, A. D. Grasso, and G. Palumbo, "Regulated Charge Pumps: A Comparative Study by Means of Verilog-AMS," *Electronics*, vol. 9, no. 6, p. 998, 2020.
146. B. Vernay, A. Krust, G. Schröpfer, F. Pêcheux, and M.-M. Louerat, "SystemC-AMS Simulation of a Biaxial Accelerometer based on MEMS Model Order Reduction," in *Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP), Montpellier, France*. IEEE, 2015.
147. P. H. Feiler, J. Hansson, D. de Niz, and L. Wrage, "System architecture virtual integration: An industrial case study," *Software Engineering Institute Technical Report, CMU/SEI-2009-TR-017*, 2009.
148. J. Cabral, M. Wenger, and A. Zoitl, "Enable co-simulation for industrial automation by an fmu exporter for IEC 61499 models," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 449–455.
149. D. Tabakov and M. Y. Vardi, "Optimized Temporal Monitors for SystemC," in *Runtime Verification*, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, Eds. Berlin, Heidelberg: Springer, 2010, pp. 436–451.
150. K. Havelund, "Runtime Verification of C Programs," in *Proceedings of the 20th IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems: 8th International Workshop, Tokyo, Japan*, ser. TestCom '08 / FATES '08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 7–22.
151. D. Do Tran, J. Walter, K. Grüttner, and F. Oppenheimer, "Towards time-sensitive behavioral contract monitors for iec 61499 function blocks," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, 2020, pp. 27–34.

152. A. Nassar, F. J. Kurdahi, and W. Elsharkasy, "NUVA: Architectural Support for Runtime Verification of Parametric Specifications over Multicores," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems, Amsterdam, Netherlands*, ser. CASES. IEEE Press, 2015, p. 137–146.
153. N. Decker, B. Dreyer, P. Gottschling, C. Hochberger, A. Lange, M. Leucker, T. Scheffel, S. Wegener, and A. Weiss, "Online Analysis of Debug Trace Data for Embedded Systems," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), Dresden, Germany*, 2018, pp. 851–856.
154. T. Tracy, L. M. Tabajara, M. Vardi, and K. Skadron, "Runtime Verification on FPGAs with LTLf Specifications," in *Formal Methods in Computer Aided Design (FMCAD), Haifa, Israel*, 2020, pp. 36–46.
155. H. Lu and A. Forin, "Automatic Processor Customization for Zero-Overhead Online Software Verification," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 10, pp. 1346–1357, 2008.
156. D. Solet, S. Pillement, J.-L. Béchennec, M. Briday, and S. Faucou, "HW-based Architecture for Runtime Verification of Embedded Software on SoPC systems," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Edinburgh, UK*, 2018, pp. 249–256.
157. Y.-J. Park, S.-K. S. Fan, and C.-Y. Hsu, "A Review on Fault Detection and Process Diagnostics in Industrial Processes," *Processes*, vol. 8, no. 9, 2020.
158. A. Abid, M. T. Khan, and J. Iqbal, "A review on fault detection and diagnosis techniques: basics and beyond," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3639–3664, 2021.
159. B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, 1992.
160. W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand, "Using contract-based component specifications for virtual integration testing and architecture design," in *2011 Design, Automation & Test in Europe*, 2011.
161. A. Cimatti and S. Tonetta, "Contracts-refinement proof system for component-based embedded systems," *Sci. Comput. Program.*, vol. 97, no. P3, p. 333–348, Jan 2015.
162. V. P. Ivannikov, A. S. Kamkin, A. S. Kossatchev, V. V. Kuliamin, and A. K. Petrenko, "The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models," in *Programming and Computer Software*, vol. 33, 2007.
163. T. Bouhadiba, F. Maraninchi, and G. Funchal, "Formal and executable contracts for transaction-level modeling in systemc," in *Proceedings of the Seventh ACM International Conference on Embedded Software*, ser. EMSOFT '09, New York, NY, USA, 2009, p. 97–106.
164. P. Nuzzo and A. Sangiovanni-Vincentelli, "Robustness in analog systems: Design techniques, methodologies and tools," in *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, 2011, pp. 194–203.
165. T. Zonta, C. A. Da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive Maintenance in the Industry 4.0: A Systematic Literature Review," *Computers & Industrial Engineering*, vol. 150, p. 106889, 2020.
166. J. Baranowski, P. Bania, I. Prasad, and T. Cong, "Bayesian Fault Detection and Isolation using Field Kalman Filter," *EURASIP Journal on Advances in Signal Processing*, vol. 2017, p. 79, 2017.
167. F. Nemati, S. M. S. Hamami, and A. Zemouche, "A Nonlinear Observer-Based Approach to Fault Detection, Isolation and Estimation for Satellite Formation Flight Application," *Automatica*, vol. 107, pp. 474–482, 2019.
168. J. Chen, P. M. Frank, M. Kinnaert, J. Lunze, and R. J. Patton, *Fault Detection and Isolation*. London: Springer London, 2001, pp. 191–207.
169. J. Cao, M. U. B. Niazi, M. Barreau, and K. H. Johansson, "Sensor Fault Detection and Isolation in Autonomous Nonlinear Systems Using Neural Network-Based Observers," in *2024 European Control Conference (ECC), Stockholm, Sweden*, 2024.
170. S. Mehlhop and J. Walter, "Model-Aware Simulation of IEC 61499 Designs," in *27th IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany*, 2022.
171. A. Zoitl and R. Lewis, *Modelling Control Systems Using IEC 61499*, 2nd ed. Institute of Electrical and Electronics Engineers (IEEE), 2014.
172. J. Sini, M. Violante, and R. Dessì, "Computer-aided design of multi-agent cyber-physical systems," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 677–684.

173. A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems," *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.
174. E. Böde, W. Damm, G. Ehmen, M. Fränzle, K. Grüttner, P. Ittershagen, B. Josko, B. Koopmann, F. Poppen, M. Siegel, and I. Stierand, "MULTIC-Tooling," in *FAT-Schriftenreihe 316*, 2019.
175. C. Oh, M. Lora, and P. Nuzzo, "Quantitative verification and design space exploration under uncertainty with parametric stochastic contracts," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22, New York, NY, USA, 2022.
176. Eclipse Foundation, "What is Eclipse 4diac," https://eclipse.dev/4diac/, 2024.
177. M. Shahjalal, T. Shams, M. E. Islam, W. Alam, M. Modak, S. B. Hossain, V. Ramadesigan, M. R. Ahmed, H. Ahmed, and A. Iqbal, "A review of thermal management for Li-ion batteries: Prospects, challenges, and issues," *Journal of Energy Storage*, vol. 39, p. 102518, 2021.
178. C. Bibin, M. Vijayaram, V. Suriya, R. Sai Ganesh, and S. Soundarraj, "A review on thermal issues in li-ion battery and recent advancements in battery thermal management system," *Materials Today: Proceedings*, vol. 33, pp. 116–128, 2020, international Conference on Future Generation Functional Materials and Research 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785320320976
179. S. S. Madani, C. Ziebert, and M. Marzband, "Thermal behavior modeling of lithium-ion batteries: A comprehensive review," *Symmetry*, vol. 15, no. 8, p. 1597, 2023.
180. R. D. Widyantara, S. Zulaikah, F. B. Juangsa, B. A. Budiman, and M. Aziz, "Review on battery packing design strategies for superior thermal management in electric vehicles," *Batteries*, vol. 8, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2313-0105/8/12/287
181. D. Shin, M. Poncino, E. Macii, and N. Chang, "A Statistical Model-Based Cell-to-Cell Variability Management of Li-ion Battery Pack," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 252–265, 2015.
182. S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino, "A layered methodology for the simulation of extra-functional properties in smart systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1702–1715, 2017.
183. T. Özdemir, A. Amini, Özgür Ekici, and M. Köksal, "Experimental assessment of the lumped lithium ion battery model at different operating conditions," *Heat Transfer Engineering*, vol. 43, no. 3-5, pp. 314–325, 2022.
184. H. Shi, L. Wang, S. Wang, C. Fernandez, X. Xiong, B. E. Dablu, and W. Xu, "A novel lumped thermal characteristic modeling strategy for the online adaptive temperature and parameter co-estimation of vehicle lithium-ion batteries," *Journal of Energy Storage*, vol. 50, p. 104309, 2022.
185. J. Sun, D. Dan, M. Wei, S. Cai, Y. Zhao, and E. Wright, "Pack-level modeling and thermal analysis of a battery thermal management system with phase change materials and liquid cooling," *Energies*, 2023.
186. O. Champhekar, X. Hu, and A. Wakale, "Validation of a lumped electro-thermal model of a 14S1P battery module with 3D CFD results," *SAE Technical Paper*, 2021.
187. C. Irimia, M. Grovu, G.-M. Sirbu, A. Birtas, C. Husar, and M. Ponchant, "The modeling and simulation of an electric vehicle based on Simcenter Amesim platform," in *2019 Electric Vehicles International Conference (EV)*, 2019.
188. "Smart systems for software-defined and hardware-enabled vehicles," https://www.smart-systems-integration.org, European Association on Smart Systems Integration (EPoSS), Tech. Rep., 2025.
189. A. Sridhar, A. Vincenzi, D. Atienza, and T. Brunschwiler, "3D-ICE: A Compact Thermal Model for Early-Stage Design of Liquid-Cooled ICs," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2576–2589, 2014.
190. M. Petricca, D. Shin, A. Bocca, A. Macii, E. Macii, and M. Poncino, "An automated framework for generating variable-accuracy battery models from datasheet information," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 365–370.
191. L. Gao, S. Liu, and R. Dougal, "Dynamic lithium-ion battery model for system simulation," *IEEE Transactions on Components and Packaging Technologies*, vol. 25, no. 3, pp. 495–505, 2002.
192. L. Mattia, H. Beiranvand, W. Zamboni, and M. Liserre, "Lithium-ion battery thermal modelling and characterisation: A comprehensive review," *Journal of Energy Storage*, vol. 129, 2025.
193. A. Basia, Z. Simeu-Abazi, E. Gascard, and P. Zwolinski, "Review on state of health estimation methodologies for lithium-ion batteries in the context of circular economy," *CIRP Journal of*

Manufacturing Science and Technology, vol. 32, pp. 517–528, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1755581721000249

194. W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.
195. O. Kalaf, D. Solyali, M. Asmael, Q. Zeeshan, B. Safaei, and A. Askir, "Experimental and simulation study of liquid coolant battery thermal management system for electric vehicles: A review," *International journal of Energy Research*, 2020.
196. A. Sharma, P. Zanotti, and L. P. Musunur, "Enabling the electric future of mobility: Robotic automation for electric vehicle battery assembly," *IEEE Access*, vol. 7, 2019.
197. Panasonic. Panasonic NCR18650B Datasheet. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/597043/PANASONICBATTERY/NCR18650B.html
198. M. Afraz, Z. Ali Mohammadi, and G. Karimi, "A novel compact thermal management model for performance evaluation of Tesla-like Lithium-ion battery packs," *Energy Conversion and Management*, vol. 300, p. 117927, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890423012736
199. "Understanding the different Tesla Model S versions," https://www.clutch.ca/blog/posts/understanding-the-different-tesla-model-s-versions, Clutch Technologies Inc., Tech. Rep., 2024.
200. B. Xu and Z. Arjmandzadeh, "Parametric study on thermal management system for the range of full (tesla model s)/compact-size (tesla model 3) electric vehicles," *Energy Conversion and Management*, vol. 278, p. 116753, 2023.
201. K. Kundert, H. Chang, D. Jefferies, G. Lamant, E. Malavasi, and F. Sendig, "Design of mixed-signal systems-on-a-chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 19, no. 12, pp. 1561–1571, 2000.
202. P. Nuzzo and A. Sangiovanni-Vincentelli, "Robustness in analog systems: Design techniques, methodologies and tools," in *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, 2011, pp. 194–203.
203. S. Sunter and K. Jurga, "Automated observability analysis for mixed-signal circuits," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–6.
204. S. Sunter, M. Wolinski, A. Coyette, R. Vanhooren, W. Dobbelaere, N. Xama, J. Gomez, and G. Gielen, "Quick analyses for improving reliability and functional safety of mixed-signal ics," in *2020 IEEE International Test Conference (ITC)*, 2020, pp. 1–10.
205. C. Grimm and C. Radojicic, "Verification and validation of ams systems: Towards coverage of uncertainties," in *2015 IEEE 20th International Mixed-Signals Testing Workshop (IMSTW)*. IEEE, 2015, pp. 1–6.
206. "Reconstruction of the MOS 6502 on the Cyclone II FPGA," 2013. [Online]. Available: https://www.cs.columbia.edu/~sedwards/classes/2013/4840/reports/6502.pdf
207. "Development of the MOS Technology 6502: A Historical Perspective," 2022. [Online]. Available: https://www.embeddedrelated.com/showarticle/1453.php
208. M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, 2018.
209. O. Dürr, P. Fan, and Z. Yin, "Bayesian calibration of mems accelerometers," in *ArXiv*, 2023.
210. Z. Zhai, X. Xiong, L. Ma, Z. Wang, K. Wang, B. Wang, M. Zhang, and X. Zou, "A scale factor calibration method for mems resonant accelerometers based on virtual accelerations," in *Micromachines 2023*, vol. 14, no. 1408, 2023.
211. C. Miller, *Chip war: The fight for the world's most critical technology*. Simon and Schuster, 2022.
212. J. Luth, "OPC 10000-1 UA Part 1: Overview and Concepts," *OPC Foundation*, 2022.
213. W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.
214. O. Givehchi, K. Landsdorf, P. Simoens, and A. W. Colombo, "Interoperability for industrial cyber-physical systems: An approach for legacy systems," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3370–3378, 2017.

215. N. C. Găitan, I. Zagan, and V. G. Găitan, "Proposed modbus extension protocol and real-time communication timing requirements for distributed embedded systems," *Technologies*, vol. 12, no. 10, p. 187, 2024.
216. O. Konradi, A. Mankowski, L. Wisniewski, and H. Trsek, "Towards an Industrial Converged Network with OPC UA PubSub and TSN," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–4.
217. J. Harding, "OPC 10000-5 UA Part 5: Information Model," *OPC Foundation*, 2023.
218. ——, "OPC 10000-3 UA Part 3: Address Space Model," *OPC Foundation*, 2023.
219. M. Damm, "OPC 10000-4 UA Part 4: Services," *OPC Foundation*, 2023.
220. S. Cavalieri and M. G. Salafia, "Insights into mapping solutions based on opc ua information model applied to the industry 4.0 asset administration shell," *Computers*, vol. 9, no. 2, p. 28, 2020.
221. P. Hunkar, "OPC 10000-2 UA Part 2: Security," *OPC Foundation*, 2023.
222. R. Armstrong, "OPC 10000-6 UA Part 6: Mappings," *OPC Foundation*, 2023.
223. A. Busboom, "Automated generation of opc ua information models—a review and outlook," *Journal of Industrial Information Integration*, p. 100602, 2024.
224. M. Majumder, B. Wiesmayr, and A. Zoitl, "Extending the OPC UA Companion Specification for an IEC 61499-based Control System," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1–4.
225. N. Mühlbauer, E. Kirdan, M.-O. Pahl, and G. Carle, "Open-Source OPC UA Security and Scalability," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 262–269.
226. A. Erba, A. Müller, and N. O. Tippenhauer, "Security Analysis of Vendor Implementations of the OPC UA Protocol for Industrial Control Systems," in *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy*, ser. CPSIoTSec '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–13.
227. F. Biondani, D. S. Cheng, and F. Fummi, "Adopting OPC UA for Efficient and Secure Firmware Transmission in Industry 4.0 Scenarios," in *2024 IEEE 33rd International Symposium on Industrial Electronics (ISIE)*. IEEE, 2024, pp. 1–6.
228. OPCFoundation, "UA-ModelCompiler," accessed on Feb 27, 2023. [Online]. Available: https://github.com/OPCFoundation/UA-ModelCompiler
229. SPEA S.p.A., "4050 Automatic Flying Probe Tester," accessed on Feb 27, 2023. [Online]. Available: https://www.spea.com/en/products/4050-s2-flying-probe-tester/
230. N. Dall'Ora, K. Alamin, E. Fraccaroli, M. Poncino, D. Quaglia, and S. Vinco, "Digital Transformation of a Production Line: Network Design, Online Data Collection and Energy Monitoring," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, pp. 46–59, 2022.
231. S. Gaiardelli, S. Spellini, M. Panato, C. Tadiello, M. Lora, D. S. Cheng, and F. Fummi, "Enabling Service-Oriented Manufacturing Through Architectures, Models, and Protocols," *IEEE Access*, vol. 12, pp. 85 259–85 274, 2024.
232. M. Hermann, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," in *49th Hawaii International Conference on System Sciences (HICSS)*, 2016.
233. X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and Industry 5.0—Inception, conception and perception," *Journal of Manufacturing Systems*, vol. 61, pp. 530–535, 2021.
234. C. Zhang, Z. Wang, G. Zhou, F. Chang, D. Ma, Y. Jing, W. Cheng, K. Ding, and D. Zhao, "Towards new-generation human-centric smart manufacturing in industry 5.0: A systematic review," *Advanced Engineering Informatics*, vol. 57, p. 102121, 2023.
235. Epic Games, "Live Link Face," https://apps.apple.com/it/app/live-link-face/id1495370836?l=en-GB, accessed: 2025-01-21.
236. R. Varghese and S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness," in *International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 2024.
237. M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *36th International Conference on Machine Learning (ICML)*, 2019.
238. G. Ghiasi, T.-Y. Lin, and Q. V. Le, "NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

239. G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO Vision," https://github.com/ultralytics/ultralytics, accessed: 2025-01-21.
240. I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," *arXiv preprint arXiv:1711.05101*, 2017.
241. CVAT.ai Corporation, "Computer Vision Annotation Tool (CVAT)," https://www.cvat.ai, accessed: 2025-01-21.
242. Al Sweigart, "PyAutoGUI," https://github.com/asweigart/pyautogui, accessed: 2025-01-21.
243. Standard, OASIS, "MQTT version 3.1.1," http://docs.oasis-open.org/mqtt/mqtt/v3, accessed: 2025-01-21.
244. S. Gaiardelli, S. Spellini, M. Panato, M. Lora, and F. Fummi, "A Software Architecture to Control Service-Oriented Manufacturing Systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.
245. S. Deng, L. Ling, C. Zhang, C. Li, T. Zeng, K. Zhang, and G. Guo, "A systematic review on the current research of digital twin in automotive application," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 180–191, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667345223000251
246. J. Pfeiffer, D. Fuchß, T. Kühn, R. Liebhart, D. Neumann, C. Neimöck, C. Seiler, A. Koziolek, and A. Wortmann, "Modeling languages for automotive digital twins: A survey among the german automotive industry," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 92–103. [Online]. Available: https://doi.org/10.1145/3640310.3674100
247. X. Li, W. Niu, and H. Tian, "Application of digital twin in electric vehicle powertrain: A review," *World Electric Vehicle Journal*, vol. 15, no. 5, 2024. [Online]. Available: https://www.mdpi.com/2032-6653/15/5/208
248. R. Magargle, L. Johnson, P. Mandloi, P. Davoudabadi, O. Kesarkar, S. Krishnaswamy, J. Batteh, and A. Pitchaikani, "A simulation-based digital twin for model-driven health monitoring and predictive maintenance of an automotive braking system." in *Modelica*, vol. 132, 2017, p. 35.
249. P. Rajesh, N. Manikandan, C. Ramshankar, T. Vishwanathan, and C. Sathishkumar, "Digital twin of an automotive brake pad for predictive maintenance," *Procedia Computer Science*, vol. 165, pp. 18–24, 2019, 2nd International Conference on Recent Trends in Advanced Computing ICRTAC -DISRUP - TIV INNOVATION , 2019 November 11-12, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920300697
250. G. Davidyan, J. Bortman, and R. S. Kenett, "Development of an operational digital twin of a freight car braking system for fault diagnosis," *Advanced Theory and Simulations*, vol. 7, no. 6, p. 2301257, 2024. [Online]. Available: https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/adts.202301257
251. Nagy, Emil and Pázmány, József and Tollner, Dávid and Török, Árpád, "Toward predictive diagnostics: Real-time digital twin for electric vehicle power systems," in *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2025, pp. 176–183.
252. C. Li, J. Lei, L. Yang, W. Xu, and Y. You, "Research on electric vehicle powertrain systems based on digital twin technology," *Electronics*, vol. 13, no. 20, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/20/4103
253. L. Popa, A. Berdich, and B. Groza, "Cartwin—development of a digital twin for a real-world in-vehicle can network," *Applied Sciences*, vol. 13, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/1/445
254. H. Liu, B. Zhang, V. Wu, X. Yang, and L. Wang, "Review of digital twin in the automotive industry on products, processes and systems," *International Journal of Automotive Manufacturing and Materials*, vol. 4, no. 1, p. 6, Mar 2025.
255. D. Piromalis and A. Kantaros, "Digital twins in the automotive industry: The road toward physical-digital convergence," *Applied System Innovation*, vol. 5, no. 4, 2022. [Online]. Available: https://www.mdpi.com/2571-5577/5/4/65
256. F. Xie, G. Li, Q. Fan, Q. Xiao, and S. Zhou, "Optimizing and analyzing performance of motor fault diagnosis algorithms for autonomous vehicles via cross-domain data fusion," *Processes*, vol. 11, no. 10, 2023. [Online]. Available: https://www.mdpi.com/2227-9717/11/10/2862

List of Figures

List of Tables

List of Acronyms

A/G	Assume-Guarantee
AC	Automation Controller
ACARE	Advisory Council for Aviation Research and Innovation in Europe
ADAS	Advanced Driver Assistance Systems
ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
AMBA	Advanced Microcontroller Bus Architecture
AMQP	Advanced Message Queuing Protocol
AMS	Analog-Mixed Signals
API	Application Programming Interface
ATE	Automated Test Equipment
ATPG	Automatic Test Pattern Generation
BDD	Block Definition Diagram
BMS	Battery Management System
CAD	Computer-aided design
CBD	Contract-Based Design
CFD	Computational Fluid Dynamics
CPPS	Cyber-Physical Production System
CPS	Cyber-Physical System
CSP	Constraint Satisfaction Programming
DAC	Digital-to-Analog Converter
DAE	Differential Algebraic Equation
DIH	Data Integration HUB
DSE	Design Space Exploration
DSL	Domain Specification Language
DT	Digital Twin
ECU	Electronic Control Unit
EDA	Electronic Design Automation
ELN	Electrical Linear Network

ERP Enterprise Resource Planning
ESL Electronic System Level
EV Electric Vehicle
FB Function Block
FDD Fault Detection and Diagnosis
FDI Fault Detection and Isolation
FEA Finite Element Analysis
FEM Finite Element Model
FMEA Failure Mode and Effect Analysis
FMI Functional Mock-up Interface
FMU Functional Mock-up Unit
FSM Finite State Machine
FTA Fault Tree Analysis
HDL Hardware Description Language
HIF Heterogeneous Intermediate Format
HMI Human-Machine Interaction
HPC High-Performance Computing
HRM Hardware Resource Modeling
HW Hardware
IBD Internal Block Diagram
IC Integrated Circuit
ICE Industrial Computer Engineering
ICPS Industrial Cyber-Physical System
IIoT Industrial Internet of Things
IoS Internet of Services
IoT Internet of Things
IP Intellectual Property
ISA International Society of Automation
ISS Instruction Set Simulator
LGS Landing Gear System
LPTN Lumped-Parameter Thermal Network
LTL Linear Temporal Logic
MBD Model-Based Design
MBSE Model-based System Engineering
MEMS Micro Electro Mechanical Systems
MES Manufacturing Execution System
ML Machine Learning
MoC Model of Computation
MOR Model Order Reduction
NES Networked Embedded System
NFP Non-Functional-Property

NoC	Network on Chip
OPC UA	Open Platform Communications Unified Architecture
OVP	Open Virtual Platform
PBD	Platform-Based Design
PCB	Printed Circuit Board
PdM	Predictive Maintenance
PID	Proportional–Integral–Derivative controller
PLC	Programmable Logic Controller
PSL	Property Specification Language
QC	Quality Checking
RMSE	Root Mean Square Error
RPC	Remote Procedure Call
RTL	Register-Transfer Level
RTN	Resource Task Network
SCADA	Supervisory Control and Data Acquisition
SCNSL	SystemC Network Simulation Library
SDK	Software Development Kit
SME	Small and Medium Enterprise
SOA	Service Oriented Architecture
SoC	System on a Chip
SOM	Service Oriented Manufacturing
SVM	Support Vector Machine
SW	Software
SysML	System Modeling Language
TLM	Transaction-Level Modeling
TSBC	Time-Sensitive Behavioral Contract
UML	Unified Modeling Language
VLSI	Very Large Scale Integration
VP	Virtual Platform
WSN	Wireless Sensor Networks
XMI	XML Metadata Interchange