Formal Verification of Safety-Critical Systems with Uncertainty for Industry 4.0 Applications

Xin Xin

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow

January 2024

Abstract

Industry 4.0 adopts Internet of Things (IoT) and service-oriented architectures to integrate Cyber-Physical Systems (CPSs) and Enterprise Planning systems into manufacturing operations. Furthermore, manufacturing processes typically involve the composition of various modular CPSs that work as a whole, such as multiple Collaborative Robots (cobots) working together as a production line, improving the production process's flexibility and resilience. On the other hand, it is still challenging to verify this kind of compositional process and take into account uncertainties from IoT sensors and decision-making algorithms. For example, the trustworthiness of the sensors is essential to guarantee performance, safety and product quality during operation. However, existing methodologies to test such systems often do not scale to today's sensor networks' complexity and dynamic nature.

Formal model verification techniques are a valuable tool that allows strong reasoning about the high-level design of CPSs. However, the uncertainty exhibited by the underlying sensor networks is often ignored. Moreover, existing model-checking tools are hard to adapt to the dynamic environment of Industry 4.0 applications during the operation stage, such as an Automated Guided Vehicle (AGV) joining in accompanying the manufacturing process at run time.

This thesis proposes a novel run-time formal verification framework for modular CPSs that combines sensor-level data-driven fault detection and system-level probabilistic model checking. The resulting framework can quantify sensor readings' trustworthiness, enabling formal reasoning for system operation behaviour and reliability analysis.

The proposed approach is evaluated on three use cases, including an industrial turn-mill machine equipped with a sensor network to monitor its main components continuously, a passenger lift with two sensor networks to monitor the door and cabinet car movements, and a two-cobot painting process running with Robotic Operation System (ROS). The results indicate that the proposed verification framework involving the quantified sensor's trustworthiness enhances the accuracy of the system failure prediction and potentially optimises manufacturing processes.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my heartfelt gratitude to all those who have contributed to the completion of this thesis. Their support, guidance, and encouragement have been invaluable throughout this journey.

First and foremost, I am deeply indebted to my supervisors, Dr Sye Loong, Dr Michele and Dr Martin, for their unwavering support, expert guidance, and insightful feedback. Their mentorship and belief in my abilities have been instrumental in shaping the direction of this research.

I would like to acknowledge the support and assistance received from Dr Andreas, Dr Teck Ping, Dr Hian Hian from TÜV SÜD Digital Service Singapore. Their commitment to the resources made available has been instrumental in the successful completion of this thesis.

I am grateful to my colleagues and friends for their camaraderie and motivation during challenging times. Their encouragement and willingness to lend a helping hand have been a source of strength throughout this research endeavour.

I extend my deepest appreciation to my family for their unwavering love, understanding, and encouragement. Their belief in my abilities has been a constant source of inspiration.

I would also like to express my gratitude to all the research participants who generously gave their time and insights, without whom this study would not have been possible.

Lastly, I acknowledge the financial support provided by the Singapore Economic Development Board (EDB) and TÜV SÜD Asia Pacific Pte Ltd, which facilitated the execution of this research.

In conclusion, I am thankful to all those who have been a part of this academic journey and have contributed to the successful completion of this thesis.

Declaration

I, Xin Xin, hereby declare that this thesis titled "Formal Verification of Safety-Critical Systems with Uncertainty for Industry 4.0 Applications" submitted to the University of Glasgow for the degree of Doctor of Philosophy is my original work, except where otherwise acknowledged. I affirm that this work has not been submitted for any other academic degree or qualification.

I acknowledge and give credit to all sources of information, data, and ideas that have been used in this thesis. All references and sources have been appropriately cited and included in the bibliography section.

Any copyrighted material or substantial portions of work done by others included in this thesis have been duly acknowledged, and permissions have been obtained as required.

I also declare that this thesis is the result of my independent research and effort, except for specific portions mentioned in the text where collaborations with other researchers or institutions are acknowledged.

Conflict of Interest: I declare that there are no conflicts of interest that might have influenced the outcome of this research or the writing of this thesis.

Funding and Support: This research is partially funded by Singapore Economic Development Board (EDB) through the Industrial Postgraduate Programme (IPP) Grant. Also, this research is supported by TÜV SÜD Asia Pacific Pte Ltd—Digital Service Singapore.

I understand the significance of academic integrity and take full responsibility for the content, findings, and conclusions presented in this thesis.

Chapter 1

Introduction

Industry 4.0, often called the "Fourth Industrial Revolution" or "I4.0", is a transformative movement in the industrial sector by integrating digital technologies, automation, and data exchange in manufacturing processes. It aims to create smarter, more efficient, and more interconnected production systems [1]. Industry 4.0 applications are revolutionising how we operate modern industrial systems, which are characterised by the increasing digitalisation and interconnection of products, value chains, and business models. Compared to the third industrial revolution, I4.0 focuses heavily on interconnectivity, automation, machine learning, and real-time data. It brings together Cyber-Physical Systems (CPSs), the Internet of Things (IoT), cloud computing, and cognitive computing to create a smart factory environment. Decentralised decision-making is the key advancement that allows components to make simple decisions on their own and become as autonomous as possible. For example, in a production process, when one machine fails, another can take over its tasks automatically without manual intervention. Sensor technologies, interoperability and robotics are fundamental to achieving the goal of Industry 4.0.

Sensor network-based systems and CPSs are foundational pillars for the Industry 4.0 paradigm that enables the digital transformation of manufacturing and industry. Sensor network-based systems play a pivotal role by acquiring real-time data, including machine status, operation data and environmental conditions. This is central to Industry 4.0, where data is harnessed from all parts of the production process. With the recent advancement in embedded sensor systems [2–6], there is a significant improvement in the modern smart manufacturing processes in terms of intelligent controls, predictive analytics and system automation. The ability to collect a manufacturing plant's shop-floor data in real-time has enabled the learning of each machine's state information and deriving insights that can subsequently be translated into self-control functions, hence automating the operations of the manufacturing plant. An automation program with initial configuration parameters is typically pre-loaded into the machine. The data collected from the machine, such as the machine status and work-piece qualities during run-time, are then used to update the configuration parameters continuously. This forms a closed feedback-control loop to maintain the smooth operation of the manufacturing plant. In such a situation, the quality and

trustworthiness of the sensor readings are crucial to effective control of the production process.

CPS is a complex system which connects sensor data acquisition systems with computer networks to utilise computer-based algorithms interacting with the physical world [7–10]. Compared to sensor network-based systems, CPSs are dynamic systems with tight integration between computational algorithms and physical components, enabling real-time monitoring, decision-making, and actuation. In contrast, sensor network-based systems primarily focus on data collection and monitoring without direct control over physical processes. Thus, safety-critical aspects of CPSs in the context of Industry 4.0 are vital as CPSs directly interact with or are near human operators. Malfunctions or unexpected behaviours from these systems can pose direct threats to human safety. For instance, a robotic arm used in assembly could harm an operator if it does not operate as intended. Moreover, a failure in a CPS can lead to shutdowns or damages, causing significant economic losses. This could be in the form of production halts, damage to produced goods, or the need for costly repairs.

With the increasing use of sensor network-based systems and CPSs, the reliability and safety of these systems become of paramount importance. For safety-critical systems, any failure can result in significant harm or loss, especially those used in transportation, healthcare, and energy production, which are particularly susceptible to failures that can result in severe consequences, including human injury, loss of life, and damage to the environment and infrastructure. To address this problem, a monitoring and verification approach is needed to ensure the safety of the deployment for Industry 4.0 applications. Hence, this research focuses on Industry 4.0 applications, especially sensor network-based systems and aims to provide a rigorous verification framework to ensure behaviour correctness.

1.1 Background of Manufacturing Systems

In the era of Industry 4.0, smart manufacturing plants are revolutionising traditional production processes that incorporate advanced technologies, automation, and intelligent systems to streamline operations, enhance productivity, and ensure high-quality output. Figure 1.1 illustrates a typical four-layer architecture of such manufacturing processes [11, 12].

- *Manufacturing Information System Layer* is the layer for high-level manufacturing management systems. For instance, Enterprise Resource Planning (ERP), Advance Planning and Scheduling (APS), Warehouse Management System (WMS) and Manufacturing Execution System (MES). These systems focus on production planning, material purchasing, resource and order management.

- *Task Management Layer* is a software layer that mainly focuses on the manufacturing process scheduling and dispatching tasks. Task management is crucial to ensure smooth operations, increased productivity, and timely delivery of products.

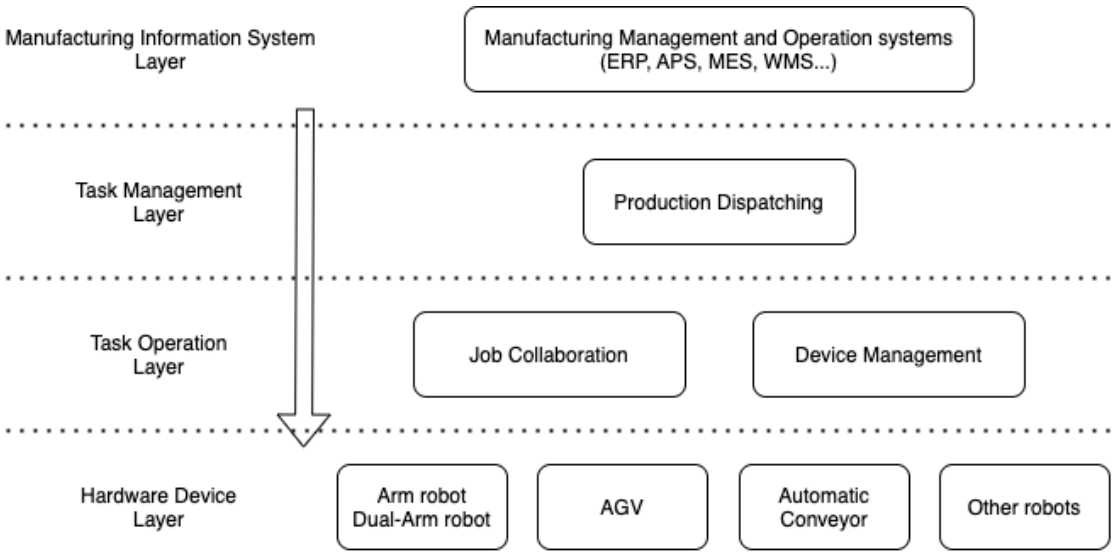

Figure 1.1: A typical Industry 4.0 manufacturing architecture.

- *Task Operation Layer* facilitates the execution and tracking of tasks. This layer operates and coordinates multiple machines, hardware executors and control systems, including distributing specific tasks to executors and monitoring their status.

- *Hardware Device Layer* contains all hardware equipment, including arm and dual-arm robots, Automated Guided Vehicles (AGVs), automatic conveyors or other operation machines. Moreover, sensor networks are equipped to monitor and provide feedback on working and environmental conditions.

All these components interact with each other and form a smart manufacturing plant. For example, a painting plant involves an ERP system, robot operation management system, AGVs and arm robots to process painting tasks for manufacturing automobile parts. In this case, firstly, the ERP system receives an order for a batch of car doors to be painted. The WMS checks the inventory and automatically triggers purchase orders to replenish the stock if any materials are insufficient. Subsequently, AGVs transport a batch of car doors to the pre-painting zone from the assembly line or warehouse. A robot arm moves car doors to painting booths. Another robot arm equipped with painting tools and sensors positions itself to precisely paint the doors. Meanwhile, the APS system optimises the painting sequence to minimise paint changeovers and downtime. Throughout the process, the operation management system monitors the AGVs' and robots' performance and adjusts parameters as needed for an effective paint process. By implementing advanced technologies and intelligent decision-making algorithms, the painting process streamlines the entire process, enhances productivity, and ensures consistent, high-quality manufacturing outputs.

In order to ensure smooth machine operation in such a smart manufacturing plant, firstly, there must be an efficient approach to accurately verify the machine's behaviour such that it

acts according to the expected behaviour over time. This significantly reduces the machine's downtime, and the replacement of faulty parts can be done at the right time, hence optimising the machine's utilisation. Secondly, the accurate prediction of machine behaviour depends very much on the accurate data acquisition from the sensors in the machine. This implies that the trustworthiness of the sensor readings is crucial. It is inevitable that inaccurate readings might occur due to factory calibration issues and sensor wear and tear, as well as malicious tampering, leading to deviation from the actual value. When using these inaccurate or untrustworthy readings in the production lines, it will significantly degrade the product outputs as well as affect the performance of the manufacturing system as a whole. Thirdly, considering the dynamic nature of the sensor networks as well as the possibility of unexpected sensor failures, the modelling methodology needs to take into account the uncertainties of sensor networks in order to represent the system accurately.

1.2 Challenges in Industry 4.0 Applications

Sensor network-based systems and CPSs play a crucial role in Industry 4.0 applications, as they are fundamental components that enable smart, connected, and highly efficient manufacturing environments. Sensor network-based systems are typically focused on gathering data from a network of sensors deployed in a particular environment. The connected sensors aim to collect and transmit sensory information from the physical world to a central processing unit or data centre for analysis and drive intelligent decision-making. Aside from sensor network-based systems, CPSs typically have a hierarchical architecture and integrate numerous physical components, such as sensors, actuators, and controllers, with the computational units and communication infrastructure. They involve the interaction between physical processes and software-controlled components to achieve desired functionalities. CPSs often exhibit real-time behaviour, feedback loops, and dynamic interactions between the physical and cyber components. Both kinds of these systems need to emphasise safety, reliability, and performance optimisation.

However, despite the many advantages of sensor network-based systems and CPSs, data accuracy and trustworthiness remain a major challenge. Sensor data can be susceptible to noise, drift, or calibration issues, affecting the reliability of the system. Sensor networks collect a vast amount of machinery operational data, and it is crucial to ensure that this data accurately reflects manufacturing operation status during the entire processing time. Consequently, the system's behaviour might change unexpectedly during the operation period, especially since the trustworthiness of the data quality is not well considered during the design phase. The challenge is that safety-critical systems require high levels of reliability to ensure continuous operation and fault tolerance. Sensor failures or communication breakdowns can have serious implications. Additionally, the complexity of sensor networks and the interoperability between different manufacturing work cells pose substantial challenges to maintaining the quality of

manufacturing processes. The traditional verification methodologies often assume the inherent trustworthiness of sensors and depend on the pre-defined models. However, these approaches struggle to accurately capture the dynamic complexities of sensor networks during run-time, particularly with regard to the trustworthiness of run-time sensor readings.

Two typical approaches are used to model and verify industrial applications, namely data-driven and model-checking approaches. The data-driven approach [13–16] might not be a suitable solution due to the lack of good training datasets. The model-checking approach [17–19], which requires prior specific domain knowledge of the process, that is not usually used in the industry. Moreover, the traditional model-checking techniques tend to assume the sensor run-time working conditions and depend on pre-defined static models, which makes it difficult to reflect the actual behaviour of the systems during the operation stage. Thus, it is hard to verify the run-time behaviour and feedback verification results to the physical systems. In Industry 4.0 systems, operations often involve interconnected components such as robots, sensors and sensor networks, actuators and CPSs, all working in real time to meet production goals and quality standards. These systems must adapt continuously to changes in environmental conditions, sensor reliability, and operational constraints. Feedback from a model-checking tool is crucial to provide actionable insights. For example, reconfiguring workflows, reallocating resources, or adjusting control parameters to address emerging issues or optimise system performance. Another example is in a smart factory, verification results could identify bottlenecks in production and suggest task redistribution to mitigate delays. In contrast, CPSs often prioritise static correctness and system safety, focusing on ensuring that the system behaves as intended under predefined scenarios. While verification remains important in CPSs, the need for feedback is typically less critical because many CPSs operate under fixed conditions or predefined control logic. This distinction highlights that while CPSs benefit from rigorous offline verification, Industry 4.0 applications demand an additional layer of operational adaptability enabled by feedback mechanisms from model-checking tools.

Additionally, it is a challenge to apply for run-time compositional structure systems. For instance, an operation management system composes multiple components at run-time according to the dynamic requirements, as shown in the example in Section 1.1, where a painting plant forms an automatic painting process that incorporates robots and AGVs according to the actual orders.

In summary, there is a need to verify the safety-critical manufacturing process at a system level, which takes into account sensor networks' trustworthiness and analysis of the impact during the system operation phase.

1.3 Thesis Statement

The aim of this research is to extend existing formal verification to develop a novel verification framework suitable for safety-critical systems with uncertainty of Industry 4.0 applications. Specifically, this work focuses on developing a novel run-time verification framework that can handle uncertainty in the modelling, analysis, and synthesis of safety-critical systems. Multiple approaches are explored to quantify and represent uncertainty, for instance, probabilistic models, compositional structures, or stochastic processes, and to evaluate their suitability for different safety-critical systems. More investigations are carried out on how to integrate uncertainty-aware verification with other aspects of the system lifecycle, including design, testing, and maintenance, to ensure the verified system operates correctly and safely under various conditions.

This resulting framework combines sensor-level fault detection and system-level model checking that is able to rigorously verify the Industry 4.0 applications while incorporating the quantified trustworthiness of sensor readings. The verification outcome can be utilised to inform application operations or improve the quality of the process. This enables formal verification of the system's run-time behaviour and contributes to optimising the manufacturing processes through the verification results.

The overall goal can be decomposed into the following three research questions:

RQ_1 : How to quantify the trustworthiness of the sensor and sensor network at run-time?

RQ_2 : How to model a sensor network-based system that reflects the impact of the changes in the sensor's trustworthiness?

RQ_3 : How to verify an Industry 4.0 application by detecting reachability of error states to ensure safety during the operation period, for example, a dynamic manufacturing process using multiple CPSs?

The key contribution of this research is providing a novel run-time formal verification framework, taking into account quantified sensor uncertainties to verify the correctness and behaviour of Industry 4.0 applications. The learning- and rule-based approaches are employed to quantify sensor trustworthiness during the system's operation period to reflect actual system running behaviour. This improves the efficiency and accuracy of model verification results. The sensor's trustworthiness is continuously quantified and used to update the system model at discrete intervals. The properties of the system model are checked at each update of the sensor trustworthiness changes, and the property-checking results are used to guide physical system updates to ensure the safety and quality of the Industry 4.0 applications. Moreover, this framework extends the traditional model structure with a new type of state, namely *interface states*. With this extension, the model can still be verified as usual during the design and operation phases. Subsequently,

these extended models can be composed to model interaction between multiple sensor networks using *interface states* to verify the higher-level system during the operation phase.

1.4 Research methodology

The objective of this research is to develop a run-time verification framework for applying formal verification techniques to ensure the reliability and safety of Industry 4.0 systems. Formal verification involves mathematical analysis and logical reasoning to provide a rigorous approach to verifying the behaviour and the correctness of the systems. This research aims to bridge the gap between existing formal verification methods and practical implementation for Industry 4.0 applications.

The initial phase of this research involves an extensive literature review of the uncertainty of Industry 4.0 applications, such as sensor failure types, detection methods and the uncertainties from soft computing algorithms. Subsequently, formal verification techniques and applications are reviewed. Existing research on model checking, theorem proving, and run-time verification methods are analysed to identify challenges and lessons learned.

The primary research components include:

- Formal verification techniques include symbolic model checking, parameterised model checking and compositional model verification.
- Quantification of the uncertainty from sensor networks, which are the crucial components of Industry 4.0 applications.
- Industry 4.0 applications cover manufacturing process, optional equipment and sensor network-base systems.

The research adopts a mixed-methodology approach, combining qualitative analysis of industry case studies and quantitative evaluation of the proposed framework's effectiveness in verifying through the experiments and using real industry environments as the testbed. Data are collected through literature reviews, interviews with industry experts, and experiments. Formal verification tools are also used for the proposed verification framework. Quantitative data are analysed using statistical methods.

1.5 Research Activities

Fig 1.2 shows an overview of the activities during the period of this research work. The research work was carried out in four phases. In the first phase, a state-of-the-art analysis was performed to understand the different techniques for sensor uncertainty detection and formal verification

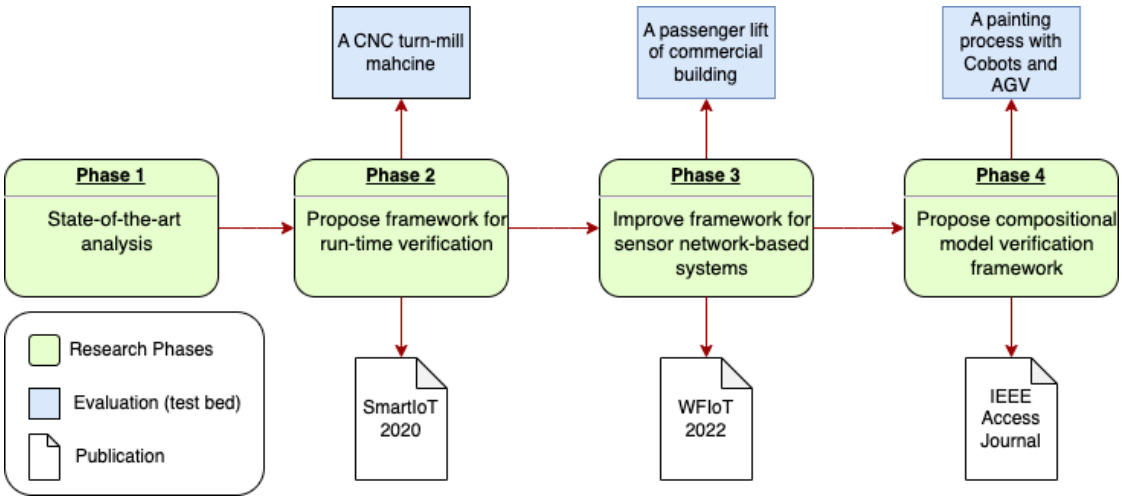

Figure 1.2: Phases of overall research.

techniques. These works were further analysed to identify the challenges and gaps. Based on this, an overall research goal was developed.

In Phase 2, according to phase one research, a conceptual approach was formulated. A run-time verification framework is proposed to verify a sensor-based system, taking into account the quantified sensor uncertainties. An industrial turn-mill machine was used to evaluate this proposed approach. The resulting approach was published in the proceedings of the IEEE International Conference on Smart Internet of Things (SmartIoT) 2020 [20].

In Phase 3, the proposed framework was enhanced to handle a sensor network-based system instead of a sensor-based machine. The enhancement mainly focuses on the quantification of sensor networks' uncertainties. An actual working passenger lift was used to evaluate the enhanced framework. The resulting approach was published in the proceedings of the IEEE 8th World Forum on Internet of Things (WF-IoT) 2022 [21].

Finally, the proposed verification framework was generalised to a compositional scenario to verify a simulated Industry 4.0 manufacturing process using two collaborative arm robots (cobots). The resulting improved approach was published in the journal of IEEE Access 10 (2022) [22].

Apart from the above research works, as the second author, another workshop paper about run-time risk analysis for sensor-based Unmanned aerial vehicles (UAV) was published in the proceedings of the IEEE 8th World Forum on Internet of Things (WF-IoT) 2022 [23]. This activity enriched the knowledge and experience of sensor-based systems, which helped to develop a more generalised verification framework for sensor network-based systems.

1.6 Thesis Structure

This thesis is organised as follows:

Chapter 2 provides background information and existing works that are related to this thesis. Firstly, it describes the types of sensor failure in the actual industry deployment. Furthermore, some failure detection methods are provided to identify and diagnose the sensor status. Subsequently, the existing system verification techniques are studied to analyse the impact according to the sensors' trustworthiness changes.

Chapter 3 presents a run-time verification framework that integrates data-driven methodologies and model-checking techniques to incorporate the trustworthiness of sensors and sensor networks into the run-time verification of system behaviour. Unlike traditional static model verification approaches, which rely on predefined models and assumptions about sensor accuracy, this framework enables run-time quantification of sensor trustworthiness and continuously updates models accordingly. By leveraging probabilistic model checking, the proposed framework addresses the limitations of static verification and provides actionable feedback from run-time verification results, thereby improving the operational decision-making process.

Chapter 4 demonstrates the process of applying the proposed run-time verification framework to a practical sensor-based system. The evaluation was conducted using an industrial CNC turn-mill machine as the experimental platform. The experiment was integrated with the algorithms designed to quantify the trustworthiness of sensor readings, as well as the model checker to verify the run-time system properties.

In Chapter 5, another use case is introduced, wherein the proposed run-time verification framework is applied to handle sensor networks rather than individual sensors. In this scenario, the aggregation of related sensors into a network aims to enhance the accuracy of trustworthiness quantification for the sensor network. The experiment employed a working passenger lift in a commercial building, which was equipped with two sensor networks to monitor its operation. Data from these networks was collected and processed to assess the performance of the run-time verification framework.

A more complex scenario within the context of Industry 4.0 applications is presented in Chapter 6. This scenario involves a manufacturing process that includes a painting process management system and two cobot arms. This manufacturing process was abstracted into a system model to evaluate the effectiveness of the proposed run-time verification framework. A mock-up physical system was set up as the experimental platform, and six test cases were designed to validate the property-checking results in a physically working environment.

Chapter 7 discusses the limitations of the proposed verification framework and the possibility of actual deployment for the manufacturers. The summary of the contribution of this thesis, answers to the research questions and future possible improvements are discussed in Chapter 8. Additionally, in the Appendix, detailed experiment data is provided.

1.7 Research Publications

The proposed verification framework has resulted in the following publications:

- Xin, Xin, Sye Loong Keoh, Michele Sevegnani, and Martin Saerbeck. 'Dynamic Probabilistic Model Checking for Sensor Validation in Industry 4.0 Applications'. In 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), 43–50. Beijing, China: IEEE, 2020.
  https://doi.org/10.1109/SmartIoT49966.2020.00016.

- Xin Xin, Sye Loong Keoh, Michele Sevegnani, and Martin Saerbeck. 'Run-Time Probabilistic Model Checking for Failure Prediction: A Smart Lift Case Study'. In 2022 IEEE 8th World Forum on Internet of Things (WF-IoT), pages 1–7, October 2022.
  https://doi.org/10.1109/WF-IoT54382.2022.10152177.

- Xin, Xin, Sye Loong Keoh, Michele Sevegnani, Martin Saerbeck, and Teck Ping Khoo. 'Adaptive Model Verification for Modularized Industry 4.0 Applications'. IEEE Access 10 (2022): 125353–64.
  https://doi.org/10.1109/ACCESS.2022.3225399.

- Yong Zhi Lim, Xin Xin, and Teck Ping Khoo. 'Enhancing UAV Flight Safety through Sensor-based Runtime Risk Assessment'. In 2022 IEEE 8th World Forum on Internet of Things (WF-IoT), pages 1–5, October 2022.
  https://doi.org/10.1109/WF-IoT54382.2022.10152064.

Chapter 2

State of the Art

Formal verification, by definition, is a rigorous method used in the field of computer science to ensure the correctness of a system or software [24]. It involves mathematical techniques and logical reasoning to verify a system adheres to the specifications and requirements. An extensive amount of literature work has been done, and multiple formal verification techniques are used for verifying sensor network-based systems and CPSs [25, 26]. An elaborate survey of formal verification approaches was presented by *Clarke et al.* [27]. This paper discusses two well-established approaches to verification: model checking and theorem proving. Model checking involves building a finite model of a system and checking if desired properties hold. Theorem proving involves expressing the system and its desired properties as formulas in mathematical logic and finding proof of the properties. Both approaches have been successfully used in hardware and software verification. The paper highlights several successful case studies using formal methods. For instance, IBM [28] used formal specifications to improve the quality of its Customer Information Control System (CICS), resulting in a reduction in errors and improved system quality. Another example is distributed fault-tolerant software for London's airspace using formal description and refinement techniques. Moreover, Lockheed [29] used formal methods to analyse and verify the avionic software for the Lockheed C130J, resulting in an improvement in software quality.

Compared to existing verification for hardware and software, Industry 4.0 application refers to integrating advanced digital technologies into various aspects of manufacturing and industrial processes, including hardware and software, such as sensors, conveyors, control systems and scheduling systems. It aims to create intelligent manufacturing and optimise operations by leveraging technologies such as sensor networks, artificial intelligence (AI), CPSs, and other automation systems. Typically, these technologies are soft computing-based and deal with problems that are complex, vague, and uncertain in nature. They are widely applied in many Industry 4.0 domains [30–34], such as 3d printing, optimising and scheduling manufacturing systems, analysing pharmaceutical hierarchy processes and so on so forth. Compared to traditional computing, in which problems are well-defined and precise, soft computing techniques are designed

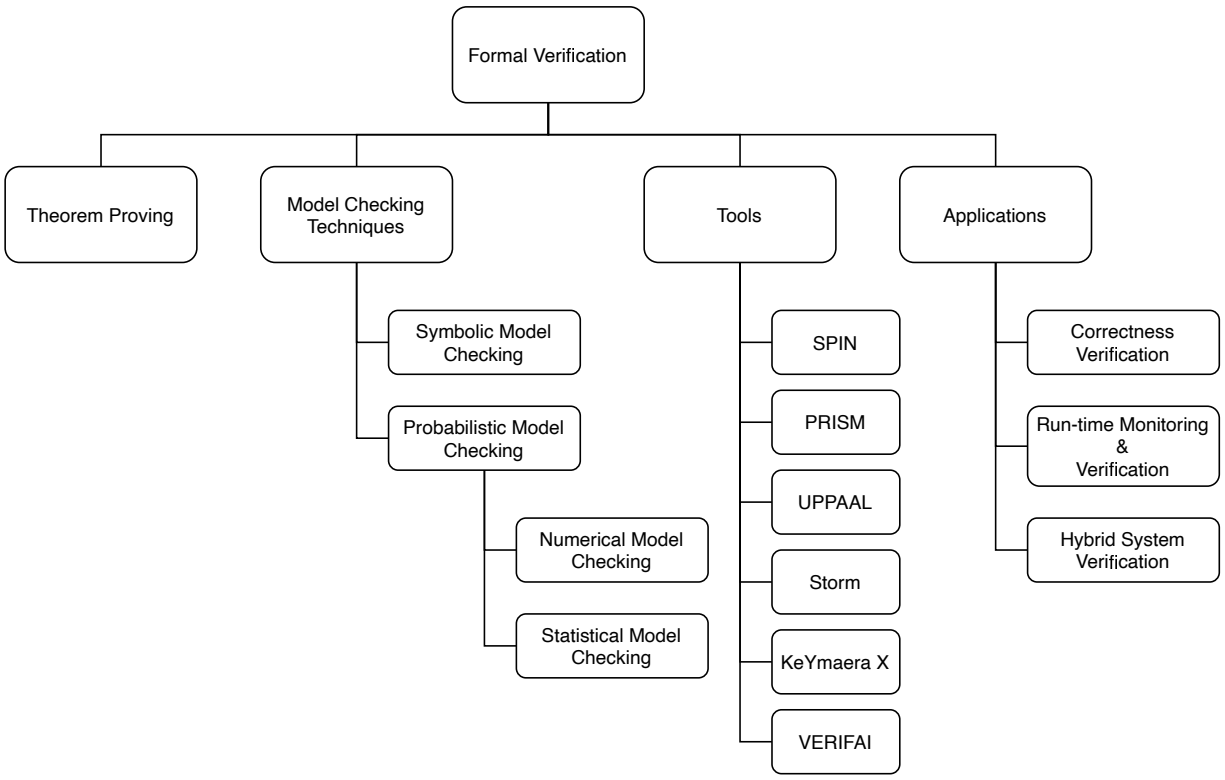

Figure 2.1: Formal verification topics.

to handle imprecision, ambiguity, and partial truth, for instance, in machine learning, fuzzy logic, genetic algorithms and neural networks. As a result, soft computing exhibits uncertainties during run time according to the working conditions.

In this chapter, formal verification techniques are discussed in section 2.1 that focus on model-checking techniques and applications. Categorisation and detection of uncertainty from sensors are presented in section 2.2. Finally, the applications of formal verification techniques about CPSs are in section 2.3.

2.1 Formal Verification Techniques

Formal model verification plays a crucial role in ensuring the correctness and reliability of sensor network-based systems. It is a rigorous and automated technique used to verify the correctness of a given system's specifications. Typically, formal verification provides a high level of assurance by exhaustively analysing all possible system behaviours, considering different input scenarios, and proving properties using formal methods [35]. Fig 2.1 illustrates a high-level model-checking technique to tackle the challenges, and below are the major techniques of formal verification.

- Model Checking is a popular technique for system verification as it exhaustively explores

the system's state space to verify desired properties. It is effective for small to medium-sized systems but faces challenges with scalability when dealing with large and continuous state spaces due to state space explosion [36, 37].

- Theorem Proving, on the other hand, provides a rigorous mathematical approach to proving correctness properties using logical reasoning. It allows for detailed and fine-grained analysis but requires significant expertise and effort in constructing formal proofs.
- Symbolic Model Checking allows for the analysis of systems by constructing symbolic representations of the system's state space. It can handle large state spaces effectively and supports verifying complex temporal properties.
- Probabilistic Model Checking extends the analysis to systems with probabilistic behaviours and is useful for reliability analysis and performance evaluation.

2.1.1 Theorem Proving

Theorem Proving, also known as Automated Deduction (AD), is a deductive method that applies mathematical logic to prove the correctness of a system. *Bibel* [38, 39] introduced the history and perspective of research of automated deduction and proposed a revised transition logic for a broad formal platform for reasoning about actions and change in general. By using AD, the system and properties are presented by logic formalism, and the goal is to construct a proof that the system satisfies the specification for all possible inputs and states. This technique widely uses for safety-critical applications to verify the correctness of software controlling critical systems, such as the aerospace and automotive industries [40–45].

Rashid et al. [46, 47] discusses the use of formal methods, specifically theorem proving, for modelling, analysis, and verification of CPSs, and presents case studies from the automotive, avionics, and healthcare domains. The authors present the application of formal methods for functional, performance, and dependability analysis of CPS, which includes formalising complex concepts such as Laplace and Fourier transform, dynamic dependability analysis, and probabilistic analysis. This paper suggests the need for formal libraries and dedicated theorem provers to support the analysis of hybrid systems within CPS, which involve both discrete and continuous dynamics. It highlights the growing importance of accurate analysis and verification of CPS due to their safety-critical nature and presents examples of formalisation and verification of various components and systems.

While theorem proving is extremely powerful, it comes with a set of challenges that can make its application in real-world industry scenarios quite difficult. It can handle very complex systems and specifications, but creating the necessary proofs is a difficult and time-consuming task. It requires the expertise of logic and formal methods, which are specialised skills uncommon in many industries. Moreover, Industry 4.0 applications often have a level of complexity

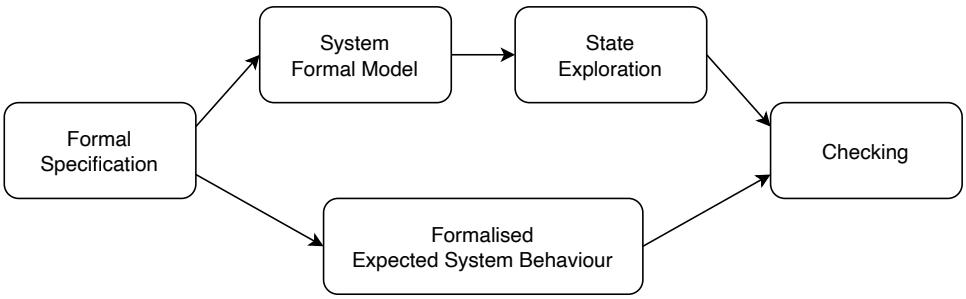

Figure 2.2: A process of verifying software system using model checker technology.

that is difficult to capture in a formal proof. This is particularly the case for legacy manufacturing systems, which might have been developed over many years and may have poorly documented or even unknown behaviours.

2.1.2 Model Checking

Model checking is also a formal verification technique used to analyse a system's behaviour and properties systematically. It is widely used to verify modern system behaviour and analyse system reliability. Model checking relies on formal specification languages and temporal logic to abstract system models and properties [48]. A formal specification language is a finite state-transition graph that provides an adequate abstraction for finite-state systems, such as software systems, communication protocols, or hardware systems. Additionally, temporal logic provides a framework for the description of correctness of properties for the state-transition systems. Model checking is especially valuable for safety-critical and mission-critical systems, where even small defects or failures can have severe consequences.

Figure 2.2 illustrates a process of verifying software systems using model checking, which typically has the following steps.

1. The first step in model checking is to define the formal specification of the software system. The specification describes the desired behaviour, properties, and requirements that the system should satisfy. It may include safety properties, for example, the system shall never enter an unsafe state, or the system will eventually reach a desired state.

2. A formal model of the software system is created based on the specifications. This model represents the system's behaviour using formal languages or mathematical notations, such as state transition graphs, finite automata, or temporal logic formulas. The model abstracts away unnecessary details while capturing essential aspects of the system's behaviour.

3. The properties specified in the first step are also formalised using suitable logical formulas. These formulas express the expected behaviour of the system in a language that the model checker can understand and analyse.

4. The core step of model checking is that the state space of the formal model is explored using suitable algorithms. The state space represents the system's possible states and how it can transit from one state to another based on its behaviour and the specified rules.

5. During state space exploration, the model checker systematically checks whether the expected properties hold true in all reachable states of the model. It checks if the system can violate safety properties, for example, reaching an error state, and verifies if aliveness properties are satisfied, such as eventual termination.

Some model-checking techniques have gained popularity and widespread use in the field of software and systems verification.

- *Symbolic Model Checking* is a formal verification technique used to verify whether a model or system satisfies specified properties rigorously. It involves mathematically proving the correctness of a model or system based on its formal specification [49].

- *Bounded Model Checking (BMC)* is a technique that focuses on verifying properties within a bounded number of steps or transitions in the system's state space. It is particularly useful for finding bugs and errors in software systems with large state spaces [50–52].

- *Explicit State Model Checking (EMC)* involves enumerating and storing individual states explicitly during the verification process. This approach is suitable for systems with a relatively small state space [53–56].

- *Satisfiability Modulo Theories (SMT) Solvers* are automated tools that can decide the satisfiability of logical formulas over theories such as arithmetic, bit-vectors, and arrays [57–59].

- *Quantitative Model Checking* is also a model-checking technique that performs quantitative model verification. A quantitative model checker focuses on assessing the performance, accuracy, and uncertainty of system models in terms of numerical measures [60–62].

- *Probabilistic Model Checking* deals with systems that involve probabilistic or stochastic behaviour. It verifies the properties of the system's probabilistic behaviour, making it suitable for modelling and verifying systems with uncertainty.

Clark et al. [63] demonstrated that model checking, an automatic verification technique for finite-state hardware and software systems, faces the "state explosion problem" as the number of state variables in the system increases, causing the system state space to grow exponentially. The lecture notes explain how the model-checking algorithms work, approaches to the state explosion problem, and focus on BMC. Ensuring correctness in software and hardware is crucial

for safety-critical systems, which has increased the interest in applying formal methods and verification techniques like model checking.

Beyer et al. [64] introduced a general approach to formal verification of software using model-checking techniques. The authors gave an overview of the history of formal verification theory and presented the development of software verification tools. The authors also introduced an important scientific method, competitions, to provide regular comparative evaluations of automatic tools for software verification. Moreover, this research work emphasises the maturity of software model checking as a research area, showcasing the development of numerous verification tools, their participation in competitions, and the integration of various verification systems and technologies. It highlights the continuous interest and community activity in the field, with an emphasis on areas of future research, including verification witnesses, concurrent programs, unbounded parallelism, termination, cooperative verification, machine-learning-based invariant generation, hyper-properties, and quantum programs.

2.1.3 Formal Specification Language and Temporal Logic

A formal specification language is a mathematical or formal language used to precisely describe the behaviour, properties, and requirements of a system. Temporal logic is a mathematical logic used to reason about the behaviour and properties of systems over time. It provides a formal language for expressing temporal relationships, constraints, and properties of system executions. Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are two prominent formal languages for expressing and reasoning about temporal properties. LTL is a widely used temporal logic that allows reasoning about sequences of states or events over time. It consists of temporal operators describing the states' ordering and relationship in a system's execution trace. The common LTL operators are:

- G (Globally): Specifies that a property holds in all future states. For example, $G(p)$ indicates property p is always true in the entire execution trace.
- F (Eventually): Specifies that a property holds at some point in the future. For example, $F(p)$ asserts that property p will eventually become true.
- X (Next): Specifies that a property holds in the next state. For example, $X(p)$ indicates that property p will be true in the next state.
- U (Until): Specifies that a property p holds until another property q becomes true. For example, pUq states property p holds until property q becomes true.

CTL is another widely used temporal logic that allows reasoning about temporal properties in branching execution paths or system behaviours. It introduces path quantifiers to express properties over all possible paths of a system. Below are operators commonly used by CTL:

- *AG* (All Global): Specifies that a property holds in all states along all possible paths. For example, $AG(p)$ states that property p holds in all states in all possible paths.
- *EF* (Exists Eventually in a Future path): Specifies that a property will eventually hold in some future state along at least one path. For example, $EF(p)$ indicates that property p will eventually become true in at least one future path.
- *AX* (All Next): Specifies that a property holds in the next state along all possible paths. For example, $AX(p)$ states that property p holds in the next state along all possible paths.
- *EU* (Exists Until): Specifies that a property p holds until another property q becomes true along at least one path. For example, $pEUq$ asserts that property p holds until property q becomes true along at least one path.

Probabilistic Temporal Logic (PTL) is an extension of traditional temporal logic that incorporates probabilistic aspects into reasoning about system behaviour over time. Probabilistic Computation Tree Logic (PCTL) extends CTL to allow for expressing and reasoning about temporal properties in systems where uncertainty or probabilistic behaviour is present. The following are two probabilistic operators that enable the verification of properties that involve probabilities and likelihoods.

- P (Probability): Specifies the probability of a property holding along a path. For example, $P > 0.5(p)$ states that property p has a probability greater than 0.5 of being true along a path.
- S (Probability Operator): Specifies the probability of a property holding until another property becomes true. For example, $pS >= 0.7q$ states that property p holds with a probability greater than or equal to 0.7 until property q becomes true.

With formal specification languages and temporal logic, a system can be modelled and abstracted with respect to the system specifications. Subsequently, this abstraction can be rigorously verified using a formal framework with probabilistic or uncertain behaviours.

A research work [65] demonstrates the model verification approach to address the challenge of the trustworthiness of sensor-driven systems. The paper introduces two concepts, frames of reference and frames of function, to organise models of sensor-based systems. These frames facilitate communication between modellers, analysts, and stakeholders, distinguishing the purpose of each model and contributing to the overall trust that the system should fulfil the requirements, such as geographic, economic, uncertainty or failures. The authors provide an example of a smart water distribution network to illustrate how different combinations of frames can allow the reduction of multiple dimensions of focus to make analysis more manageable. However, further developments are still needed to cover comprehensive frames and include sensor-driven systems' failure scenarios.

2.1.4 Model Checking at Run time

Run-time verification constitutes a methodology intent on analysing system behaviour and verifying properties during system execution. It focuses on monitoring and checking system executions against specified properties. *Calinescu et al.* [66] introduced Formal Method @ Runtime (FM@R) and Model Checking @ Runtime approaches. The FM@R methodology accomplishes two primary objectives. Firstly, it guides the system to satisfy objectives by assessing feasible configurations. Secondly, it addresses the major concern of the correctness of system objectives. Two main activities of run-time verification are execution monitoring and the evaluation of traces against the system specification. It provides a lightweight model-finding approach to specify system requirements using mathematical notation. An experiment with virtual machine consolidation shows acceptable overheads for systems comprising up to 30-40 components. However, the proposed approach highly depends on the expertise of the system specification, and the scalability is another challenge.

César Sánchez, et al. [67] summarised the frameworks, tools and challenges of Run-time Verification (RV) from multiple application domains, for instance, hardware verification for precise timing and non-disruptive operation, security and privacy protection in the context of European General Data Protection Regulation (GDPR). The authors categorised RV approaches and challenges into different categories, such as distributed systems, hybrid systems and huge, unreliable or approximated domains. In the context of hybrid and embedded systems, the paper identifies the coexistence of continuous and discrete behaviours as a challenge, along with constrained monitoring resources. In the hardware domain, precise timing and non-disruptive operation of monitors are identified as critical needs. Security and privacy domains require a suitable combination of static and dynamic analysis. Transactional information systems pose challenges related to monitoring modern information system behaviours and the compromise between expressivity and non-intrusiveness in monitors. The connection between legal and technical aspects of contracts and policies presents paramount challenges. Lastly, the paper discusses challenges related to monitoring systems that are unreliable or require aggregation or sampling due to large amounts of data.

For safety-critical systems, such as collaborative CPSs, run-time assurance is essential. *Bartocci, Ezio et al.* [68] summarised the techniques for qualitative and quantitative monitoring of CPS behaviours. Moreover, the authors also presented applications and tools supporting CPS monitoring using formal specification languages and temporal logic, such as LTL and STL for CPSs. *Junges, Sebastian et al.* [69] presented a tractable algorithm based on model checking that combines nondeterminism and probabilities to monitor running CPSs. *Forejt et al.* [70] introduced incremental verification techniques to address the evolution of adaptive software systems during the system operation period. With the ever-increasing autonomy of CPSs, *Faymonville et al.* [71] introduced a stream-based monitoring framework to ensure the safety of the system at run time, namely StreamLAB. The authors demonstrated the capability of StreamLAB on typi-

cal monitoring tasks for CPSs, such as sensor validation and system health checks, by analysing the specifications, including the computation of memory consumption and run-time guarantees.

2.1.5 Available Model Checking Tools

Model checking is a broad technique that can be used for multiple domains, such as software verification, hardware design, protocol, cybersecurity analysis and concurrent control systems. Some model checker tools are also available that can be used directly. Following are some popular tools that focus on the formal verification of the systems.

- *SPIN* [72] is a widely used software model checker tool that helps in the verification of concurrent systems. It was developed at Bell Labs by Gerard J. Holzmann in the 1980s and has since become one of the most influential tools in the field of formal methods.

- *PRISM* [73, 74] is designed to analyse and verify probabilistic models, including both discrete- and continuous-time Markov chains, Markov decision processes, and stochastic timed automata. The tool uses model-checking techniques to automatically explore the state space of these models and verify various properties specified in probabilistic temporal logics.

- *UPPAAL* [75] provides a modelling and verification environment for real-time systems that can be described using timed automata, especially for the formal verification of real-time systems. It is designed to analyse systems that involve both discrete and continuous behaviours with precise timing constraints.

- *NuSMV* (New Symbolic Model Verifier) is a powerful model-checking tool used in the formal verification of hardware and software systems [76, 77]. NuSMV is an extension of SMV that can process files written in the SMV language and supports various model construction modalities, reachability analysis, fair CTL model checking, quantitative characteristics computation, and counterexample generation. It is an open-source tool widely used in academia and industry for verifying the correctness of designs, protocols, and systems, including hardware designs, communication protocols, distributed systems, and software.

- *Storm* is designed to handle probabilistic systems [78], including discrete- and continuous-time Markov chains and Markov decision processes, which are explicit states and fully symbolic model checking, as well as a modular set-up for easy exchange of solvers and decision diagram packages. It also provides a Python API for prototype development.

- *KeYmaera X* [79] is a formal verification tool that focuses on the analysis and verification of hybrid systems. Hybrid systems are systems that exhibit a combination of continuous

dynamics, described by differential equations or differential inclusions, and discrete transitions, represented by automata or state machines. KeYmaera X is designed to reason about such systems' safety, stability, and other properties while considering both continuous and discrete behaviours.

- *VERIFAI* [80] for the formal verification of artificial intelligence (AI) systems, specifically those used in autonomous systems and robotics. VERIFAI is designed to analyse the behaviour of AI algorithms and ensure their safety and correctness under various operating conditions and environmental uncertainties.

Meenakshi et al. [40] present a case study of applying formal verification methods to verify the requirements of a generic aircraft flight control system, specifically focusing on the Mode Transition Logic (MTL) of an autopilot. The mode transition logic specifies the system's mode and functions, and the paper describes the modelling and verification of the autopilot's mode transition logic using three open-source model-checking tools: SPIN, NuSMV, and Symbolic Analysis Laboratory (SAL). These tools are used to evaluate the design of the MTL against a set of functional requirements, with each tool offering distinct benefits and techniques for verifying the system. This paper also highlights the challenges of state space explosion for explicit state model checkers like SPIN and the limitations of symbolic model checking. It introduces the concept of bounded model checking as an alternative, noting its effectiveness in managing time and memory consumption. Additionally, the paper discusses the benefits of formal verification, including the ability to provide thorough coverage of system behaviours and the generation of counterexamples in case of errors.

2.1.6 Model Checking Application of Industry

Calder et al. [81] introduced a stochastic probabilistic model checking framework for failure prediction of a critical communication system. It defines three status categories for each component within the system, *working, reduced-redundancy and no-service*. The model predicts not only the probability of system failure based on each component's status but also predicts future service availability. This methodology helps the operators to allocate resources optimally in the presence of component failures. This framework is based on a discrete space model and temporal logic to predict the likelihood of service failure within a given time bounds and quantify the impact of lower-level components on service availability. However, the proposed framework is hard to model run time behaviours due to the unreliable readings from sensors.

Kwiatkowska et al. [82] used the probabilistic model checker PRISM to abstract an Internet-of-Things (IoT) system with a probabilistic model. The resulting model is used to evaluate the performance and reliability of a sensor-network-based system with the injection of one or more sensor failures.

Sevegnani et al. [83] demonstrated a modelling and verification framework of a large-scale sensor network system on Bigraphical Reactive Systems (BRS) [84]. This paper proposes an extension to standard bigraph to allow overlap or intersect locations, which are essential in Industry 4.0 domains such as wireless signalling, social interactions, and audio communications. The proposed approach is able to capture not only the spatial operational aspects but also the temporal evolution to represent the dynamic behaviour of the sensor network. The paper also discusses an implementation called BigraphER, which provides an efficient implementation of computation, simulation, and visualisation for bigraphs with sharing. The authors present a case study based on real-world deployments of urban environmental monitoring to demonstrate the proposed approach. The experimental evaluation involves the generation of simulated events from a Cooja network simulator replaying actual data streams. The evaluation shows the proposed framework is capable of handling online verification of large-scale sensor network-based systems. However, to verify live Industry 4.0 systems, further research work is still needed.

Filieri et al. [85] introduced a run-time probabilistic model checking approach to evaluate the conformance of reliability requirements at run time. The authors defined two phases, namely, the design-time phase and the run-time phase. At design time, a Discrete-Time Markov Chain (DTMC) model is pre-computed, and a set of symbolic expressions is defined to represent satisfaction of the requirements. As this model transition values are known only at run time and may change over time, a set of variables is used to represent the transition probabilities. Subsequently, the verification is performed at run time by replacing the transition variables with the real values gathered by a monitoring system. However, the performance of this approach is not only dependent on the DTMC model itself, but also depends on the monitoring system inputs which is hard to apply to more general scenarios. *Li et al.* [86] presented a dynamic adaptation probabilistic model checker approach to improve self-adaptive systems' utility. They define a Markov Decision Process (MDP) [87] model as the system abstraction, and the operator provides initial transition parameters according to experiment results and experience. Subsequently, the transition parameters are updated onto the MDP model according to the operation parameters at run time. Over time, the MDP model adapts itself to the self-adaptive system's behaviour. This approach relies on the operator's actions and the effect on the system model that can be accurately measured. *Epifani et al.* [88] proposed another novel dynamic probabilistic model-checking framework based on KAMI (Keep Alive Models with Implementations). This framework is based on the Bayesian Estimation Theory (BET) to estimate the transition matrix according to the run-time system. Subsequently, the estimated transition matrix is applied to a DTMC model to increase the accuracy of failure prediction. Even so, quantifying the run-time variables, e.g., the trustworthiness of system-embedded sensors, is still a challenge.

2.2 Uncertainty of Industry 4.0 Applications

Uncertainty refers to epistemic situations involving imperfect or unknown information, including accuracy, reliability, ignorance, precision and clearness. In the domain of Industry 4.0, uncertainty is described as a state in which the future outcome is unpredictable or not determinable. Uncertain behaviour of Industry 4.0 applications is categorised into two types, namely known uncertain behaviour and unknown uncertain behaviour. Known uncertain behaviour includes faulty behaviour resulting from known potential faults during design time and sporadic behaviour with no occurrence pattern. Unknown uncertain behaviour includes emergent behaviour only known at run time and run-time faulty behaviour due to unknown failures [89].

Asmat et al. [90] provided a survey of the state-of-the-art approaches, tools, and causes for handling uncertainty in CPSs. The research result also applies to Industry 4.0 applications. This paper summarised that uncertain behaviour typically occurs at three levels and is categorised into three processes.

- *Application-level* uncertainties that occur due to user interactions and human behaviour impact the quality and reliability of the operating systems. For instance, incorrect user input or lack of knowledge about technical and environmental processes.
- *Infrastructure-level* uncertainties due to dependencies among physical units and cyber services. The main cause of this type of uncertainty could be communication failure, unreliable sensing and computational algorithm issues. For instance, network failure causes sensors and actuators not to function consistently and results in a loss, de-calibrated sensor leads to poor data collection that harms production quality or the lack of real-time computation leading to severe damages.
- *Natural process* is another root cause of uncertainty in Industry 4.0 applications, such as a sudden change in the environment leading to application unreliable.

Industry 4.0 applications' key characteristics are integrated connected sensors, streaming of real-time data, and leveraging artificial intelligence to operate and optimise manufacturing processes. In this combination, the connected sensors and real-time computation algorithms are two major sources of uncertainty.

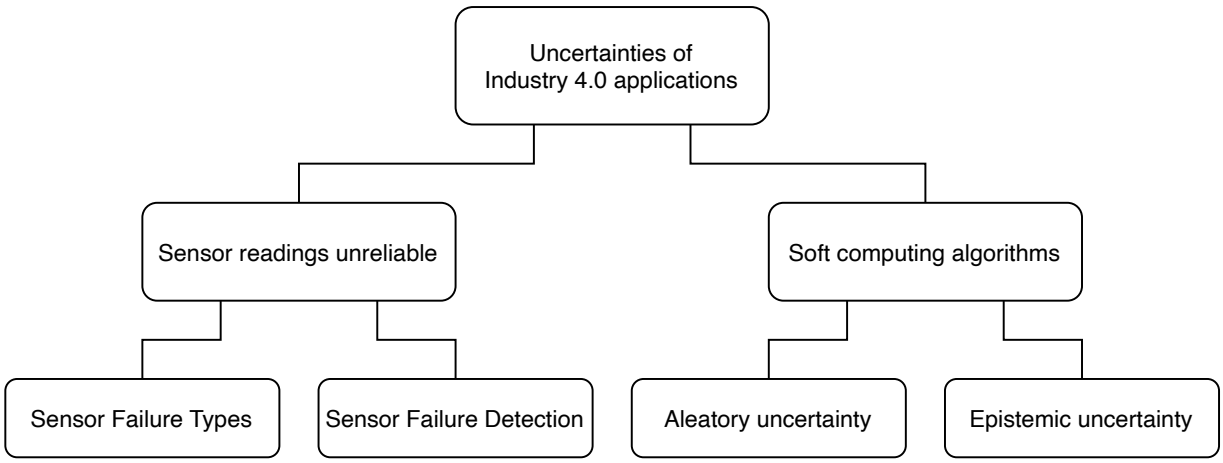

Figure 2.3: Uncertainties of Industry 4.0 applications.

Figure 2.3 illustrates uncertainty sources and structures.

2.2.1 Uncertainty of Sensor Networks

Sensor data is essential for making decisions and ensuring the overall reliability of systems. Researchers have made progress in categorising sensor failure types and detection methods to address sensor trustworthy issues.

Failure Types

Sensor faults are very common in sensor network-based systems, thus producing unreliable data that does not reflect the true state of the machine or environment. *Ni et al.* [91] defined a systematically characterised taxonomy of sensor data faults into three categories, namely *environment features*, *system features or specifications*, and *data features*. Such classification helps to identify and develop fault detection methods.

- *Environment Features* capture context and operational conditions of a sensor, including where the sensor is placed, temperature and humidity of the environment. It also includes the measured modality of the sensor and its boundary conditions. These features are crucial in determining the expected behaviour, which can then be used to determine sensor or system faults.

- *System Features or Specifications* include hardware components and calibration features. Hardware components describe the abilities of a sensor. Calibration describes the uncertainty of the mapping from input to output. For example, clipping may be a hardware or software limitation exhibited by a sensor when it maxes out due to operating conditions exceeding its limits or calibration conditions.

- *Data Features* are usually statistical in nature and calculated in either the spatial or temporal domain. For example, sensors are expected to retain similar characteristics as other co-located sensors and operate reliably for a period of time.

The *data features* is of particular interest as it includes two groups of typical sensor failures. One is observable from a data-centric point of view, where outliers, spikes and "stuck-at" faults can be identified statistically. The other group of sensor faults requires taking a system-centring view, including calibration faults, connection or hardware failures, low battery, out-of-range failures, and clipping.

Failure Detection

Sharma et al. [92] summarised four methods of sensor fault detection, which are rule-based, time series analysis-based, learning-based and estimation methods.

- The *Rule-based* approach is a common method that uses domain knowledge of sensor behaviour to develop heuristic rules to ensure that the sensor readings are in accordance with the modelled expectation.

- The *Time series analysis-based* approach uses temporal correlation in measurements to build a normal behaviour sensor model. Subsequently, data collected from the sensors in operation is compared against this normal behaviour model, often using time series forecasting to determine whether they are faulty.

- The *Learning-based* approach builds a sensor module to analyse the normal and faulty behaviours from historical data using statistical modelling or machine learning techniques. It often attempts to perform a root cause analysis.

  Based on the resulting inference models, sensor faults can be detected, and the reason for the fault can be inferred as well.

- The *Estimation* approach exploits spatial and temporal correlations in measurements from different sensors to generate the normal sensor behaviour.

Although this can help provide a guideline to detect sensor faults, it is still a challenge to quantify the impact of faulty sensors on the system as a whole.

Ramanathan et al. [93] introduced an application which applied sensor-level fault detection methods in an environmental monitoring sensor network. The authors discuss the challenges of calibration and fault detection in environmental sensor networks and present a procedure and fault detection system for addressing these issues. After forty-eight sensors were deployed in the field for twelve days to collect groundwater chemistry data, they found that a significant number of sensor readings were uninterpretable due to the prevalence of anomalous patterns. In order to identify the faults, a fault detection system was developed, which applied pre-configured rules

Table 2.1: Sensor faults and detection methods.

Fault Type	Description	Detection Method
Outlier	Outliers are the most common sensor faults. The sensor readings are out of range according to expectations.	Rule-based
Spike	The readings changed much greater than expected over a short period of time. And it may or may not return to normal afterwards.	Rule-based
Stuck-at / Constant	The readings are kept the same or almost the same for a period of time greater than expected.	Rule-based
Intermittent	Deviations from normal readings appear and disappear several times, the frequency of this signature is generally random.	Rule-based
High Noise / Variance	noise are common and expected in the sensor's data. However, unusually high noise is caused by hardware failure or low batteries, and the data may not present the expected behaviour.	Rule-based
Bias	A constant offset from the ground truth. The governing equation shall be $Y_{reading} = X + b + noise$, where X is the ground truth, and b is the constant offset.	Correlation-based
Drift	A time-varying offset from the ground truth of the sensor's signal. The equation should be $Y_{reading} = X + f(t) + noise$, where X is the ground truth and f(t) is the time-varying offset.	Time series analysis-based
Scale	A time-varying offset from the ground truth of the sensor's signal. The equation should be $Y_{reading} = X \times f(t) + noise$, where X is the ground truth and f(t) is the time-varying offset.	Time series analysis-based

to detect invalid data and identify faulty sensors. The paper also emphasizes the importance of rapid deployment and portable, reusable sensor networks in addressing environmental issues like arsenic contamination in groundwater. While the proposed system helps to verify the data integrity at the sensor level, it does not reflect the impact of the sensor faults at a higher level to highlight the potential risks.

2.2.2 Uncertainties of Computation Algorithms

The uncertainty sources in Industry 4.0 applications can be classified into two categories [94], namely *Aleatory uncertainty* and *Epistemic uncertainty*.

- *Aleatory uncertainty* is caused by natural variability, which is inevitable and irreducible. For instance, there is variability in material properties and working environment conditions. This intrinsic uncertainty can be quantified by fitting a probability distribution.

- *Epistemic uncertainty* is normally caused by limited historical data, lack of knowledge,

or model simplifications and assumptions. These uncertainties are reducible when more information or data becomes available. For example, Machine Learning (ML) models may have high predictive uncertainty if the training data set comes with bias or small volumes of the samples. This model uncertainty can be quantified by incorporating known physical monitoring data for physics-based analysis [94].

Hu et al. [95] provided an approach to modelling various uncertainty sources in the context of Additive Manufacturing (AM). The authors reviewed and summarised Uncertainty Quantification (UQ) and Uncertainty Management (UM) in the area of AM. The paper reviews the current research state of UQ/UM in AM processes, with a focus on laser powder bed fusion AM. It summarises the major methods and models then presents insights into how current UQ and UM techniques can be applied to AM to improve product quality. The paper concludes by using laser sintering of metal nanoparticles as an example to illustrate the application of UQ and UM in AM. This research work emphasises the importance of UQ and UM in addressing the variation in the quality of manufactured parts in metal-based AM processes and highlights the potential for improving product quality through the integration of UQ techniques.

Moreover, Artificial Intelligence (AI) and Machine Learning (ML) have gained significant traction in Industrial 4.0 applications and are transforming various sectors. They are used to automate tasks, improve decision-making, enhance product quality and optimise operations. While AI and ML offer tremendous opportunities, there are several challenges that industry applications face. *Dreossi et al.* [80] presented a software toolkit for the formal design and analysis systems that incorporate AI and ML components, namely VERIFAI. The paper addresses three key challenges in applying formal methods to AI/ML-based systems: perception, learning, and environment modelling. VERIFAI aims to address these challenges through its approach to simulation-based verification and synthesis guided by formal models and specifications. VERIFAI focuses on several use cases, including temporal-logic falsification, model-based systematic fuzz testing, parameter synthesis, counterexample analysis, and data set augmentation. This toolkit offers novelty in addressing these challenges through the integration of formal methods to provide a suite of use cases in an integrated fashion, unified by a common representation of an abstract feature space and accompanied by a modelling language and search algorithms. VERIFAI is structured to provide efficient communication with simulators, allowing for seamless integration and interoperability.

Fremont, Daniel J. et al. [96] proposed a new probabilistic programming language for the design and analysis of CPSs, especially machine learning-based systems, namely SCENIC. SCENIC is a domain-specific language describing scenarios distributed over scenes and the behaviours of the systems over time. It provides the syntax for spatial and temporal relationships, such as geometric relationships, which require non-linear expressions and constraints and parallel and sequential composition and interrupts for dynamic behaviours. Scenic allows specifying scenes, objects, and agents, as well as their temporal and spatial relationships in a concise

and flexible manner. This paper details the syntax, semantics, and various features of Scenic, including specifiers, behaviours, and composition of modular scenarios. Scenic provides functionalities for generating synthetic data, composing scenarios, defining and enforcing temporal and spatial constraints, and specifying complex behaviours for dynamic agents. It also integrates domain-specific algorithms for efficient sampling and provides the ability to monitor and interrupt scenarios. Additionally, Scenic allows the selection of scenarios based on predefined conditions or random selection under specific constraints. The language's features are demonstrated, including the capability to specify distributions, object definitions, and the manipulation of geometric and spatial entities. These features collectively enable the creation, analysis, and testing of complex scenarios for CPSs, which are the pillars of Industry 4.0 applications.

Torfah, Hazem et al. [97] presented a use case that uses VERIFAI and SECENIC to model and run-time assurance of an autonomous aviation application, where VerifAI was used to learn monitors for an experimental autonomous aircraft taxiing system developed by Boeing for Defence Advanced Research Projects Agency (DARPA) Assured Autonomy project. The study demonstrates the effectiveness of run-time monitors in improving the system's performance and ensuring its safe operation. This paper emphasises the role of run-time monitors in ensuring the safety of AI/ML-based autonomous systems within their operating environments. It outlines the challenges in constructing run-time monitors for capturing safe operating conditions for AI/ML-based components and highlights the importance of understanding how these components behave in complex operating environments. The paper also discusses the integration of the monitors into an architecture for run-time assurance and highlights the importance of implementing trustworthy and efficient run-time assurance modules. Additionally, the authors outline a wish list for run-time monitors and suggest prospects for using alternative learning methods, such as oracle-guided inductive synthesis [98,99] and introspective environment modelling [100, 101], further to enhance the construction of monitors for autonomous systems.

Along with the new technologies that have been applied and deployed for Industry 4.0 applications, an integrated and comprehensive approach is needed to address and quantify uncertainties to ensure safety and verify the behaviour of manufacturing processes.

2.3 Model Verification for CPSs

Considering the complexity and tight interactions of CPSs, Statistical Model Checker (SMC) is proposed [102] to tackle two obstacles [103] of modern CPSs. SMC is a simulation-based approach to sample the behaviours and check conformance to the temporal formula. *Younes et al.* [104] compared two probabilistic model checking techniques, Numerical- and Statistical-probabilistic model checking. The result showed that both techniques have similar performance, but the statistical approach scales better with the size of the state space and requires less memory. *Zarei et al.* [105] proposed another SMC approach to verify learning-based CPSs. This kind of

CPSs employs machine learning algorithm-based controllers, e.g., Neural Network, which increases complexity and non-linearity. Traditional verification techniques face state-space search and scalability challenges. Thus, they built the SMC based on the Clopper-Pearson confidence levels and defined specifications using Signal Temporal Logic (STL) to verify the reachability, safety and performance. The results showed that it is feasible to use statistical verification for learning-based CPSs. Even so, determining the CPS's run-time characteristics is still a challenge as the sensor readings are unreliable, which will significantly affect the reliability of the verification.

Apart from the SMC approach, *ModelPlex* [106] provides correctness guarantees for CPSs at run time. It combines *Model Monitor* to check the previous state and current state for compliance with the model, *Controller Monitor* checks the output of a controller implementation against the controller model, while *Prediction Monitor* checks the impact of deviation from the model to predict the eventual state that might cause failures. *ModelPlex* is based on *differential dynamic logic* dL [107] and has been applied in robotic applications [108–110] using the tool KeYmaera X [111, 112]. However, the run-time trustworthiness of the sensors' readings is not reflected through the *Controller monitor* and *Prediction monitor*.

Together with model checking, temporal logic is a popular formalism language for specifying reactive system behaviours. Temporal logic language has traditionally been used for formal verification, such as LTL, to capture safety and reachability requirements over Boolean predicates defined over the state space. Computation Tree Logic (CTL) allows the expression of requirements overall computations branching from a given state [48]. *Kamide et al.* [113] introduced a sequential Linear Temporal Logic (sLTL) and a sequential Computation Tree Logic (sCTL) by extending LTL and CTL to represent hierarchical information and structures. This research work defined the translations from sLTL and sCTL into LTL and CTL to verify hierarchical systems by reusing the standard LTL- and CTL-based model-checking algorithms.

Over the last few years, the Robot Operating System (ROS) has become a popular software framework for distributed robotics and CPSs. A ROS-based Run-time Verification (ROSRV) framework is an approach that incorporates a middle layer to intercept messages in order to verify the run-time system behaviour [114, 115]. The ROSRV provides a functional layer to intercept all messages between the slave layers to master layers, and by understanding the communication between them, the system is able to enforce the desired system behaviour based on safety policies. However, such verification systems only result in the system conforming to the behaviour and safety policies, but it has no ability to predict failures in advance. Furthermore, the middle layer is actually incurring overheads, and it can become a bottleneck when a large number of messages are exchanged in a large-scale deployment.

Ferrando et al. [116] introduced another ROS-based runtime verification framework, ROSMonitoring. This framework automatically verifies messages against formally specified properties by adding a monitor through ROS node instrumentation instead of creating a middle layer. It

provides the flexibility to scale up and choose the specification formalism, such as Linear Temporal Logic (LTL) or Signal Temporal Logic (STL). However, the ROSMonitoring approach can only be applied to ROS-based CPSs.

To describe and analyse distributed systems rigorously, *Paul et al.* [117] presented a Dynamic Input/Output Automata (DIOA) that allows the creation and destruction of components dynamically. *Civit et al.* [118] extended DIOA to a probabilistic framework to model a dynamic probabilistic system, for instance, an Industry 4.0 application using multiple CPSs to work on one manufacturing process. However, the DIOA models analyse a system only. It lacks an approach to verify such dynamic systems.

2.4 Challenges and Limitations

A significant challenge is that existing model-checking tools, such as PRISM or SPIN, handle probabilistic events or logical correctness effectively, but they lack the integration of dynamic trust metrics associated with run-time factors. These tools excel in handling probabilistic events and logical correctness but fall short when applied to real-world Industry 4.0 systems, where sensor trustworthiness is inherently dynamic. Factors such as ageing hardware, insufficient maintenance, environmental interference, or miscalibration can significantly impact sensor reliability over time. Without mechanisms to compute and update trustworthiness to pre-defined system models dynamically, the validity of verification outcomes in such contexts is limited, potentially leading to incorrect system behaviour being deemed “safe” or “correct”.

Another fundamental limitation lies in the passive nature of current verification frameworks. These tools are designed primarily to check properties like safety and correctness, generating outputs such as *property satisfied* or *counterexample found*. However, they rarely translate these results into actionable insights for the physical system. In dynamic environments like smart factories or industrial robotics, this disconnect limits the ability to adapt system behaviours or configurations in run-time based on verification outcomes. For instance, property checking results could inform operational adjustments, such as redistributing workloads to reduce the strain on unreliable components or recalibrating sensors to improve measurement accuracy. Additionally, the insights from verification models could provide invaluable evidence for refining control algorithms or optimising processes to enhance overall system performance and quality. This feedback mechanism is particularly critical in Industry 4.0 applications, where systems are expected to not only monitor operations but also actively adapt and optimise themselves in response to evolving performance metrics and working conditions. The absence of such mechanisms in existing model-checking frameworks creates a significant gap between formal verification processes and the real-world requirements of dynamic, interconnected Industry 4.0 applications.

To address these challenges, this research introduces an extended framework that bridges

the gap between sensor trustworthiness, run-time verification, and operational feedback. The proposed approach integrates sensor fault detection algorithms to monitor and update sensor trustworthiness during run-time. By dynamically computing quantified confidence values and embedding them into probabilistic models, the framework enables continuous updates to the verification process, ensuring that logical correctness is assessed with current and realistic system states. Furthermore, by incorporating these run-time verification results into actionable system adjustments and process improvements, this research not only enhances the accuracy and efficiency of verification outcomes but also aligns verification with the adaptive, optimisation-driven philosophy of Industry 4.0.

This approach is essential for advancing the state of the art in model-checking for Industry 4.0 applications, as it provides a comprehensive solution to evolving system conditions at run-time, strengthens the reliability of sensor networks, and promotes continuous operational improvement, all of which are critical to realising the full potential of Industry 4.0.

Chapter 3

Run-time Verification for Industry 4.0 Applications

This chapter proposes a new verification framework that combines the advantages of data-driven modelling and model-checking techniques to provide a run-time verification solution that can be used to verify the behaviour of sensor network-based Industry 4.0 applications. This approach leverages the data-driven learning ability to quantify sensors' trustworthiness at run-time and offers a model-checking advantage regarding rigorous verification capability.

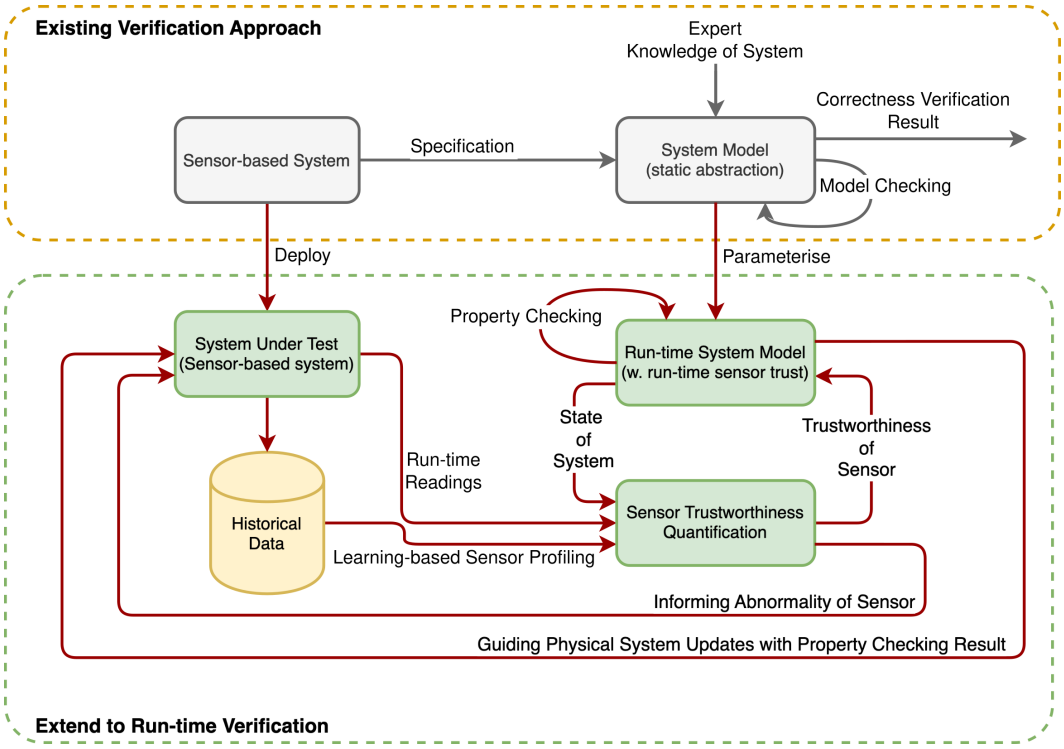

Figure 3.1: Architecture of run-time probabilistic model checker.

As illustrated in Figure 3.1, the current verification approaches are made statically in that a

formal model representing the system behaviour is first defined according to system specifications and expert knowledge. The formal model is essentially a static abstraction of the underlying sensor-based system with assumptions about their accuracy and reliability. These sensors are normally responsible for collecting data that reflects various operational metrics required for the system's function, such as temperature of working environment, pressure or motion. Subsequently, verification of the formal model through techniques such as model checker explores possible states and scenarios to identify violations or confirm adherence to the expected properties. If successful, the process produces a correctness verification result, which is an assurance that the system is operating within defined parameters. The sensor-based system is then deployed into the real world once the formal model has been verified. In this case, the verification of the formal model is done before the actual deployment. While the system is in operation, its behaviour will likely deviate from the formal model due to the underlying sensor malfunction, sensor reading drift and inaccuracy over time. Static verification assumes that all conditions remain constant and does not accommodate changes in sensor trustworthy or unexpected environmental impacts. This implies that the sensor-based system deployed in the field no longer conforms to the behaviour of its formal model, which was model-checked or verified prior to deployment.

This research extends the static verification approach by parameterising the initially verified system model as a run-time system model that is able to model the sensor-based system's behaviour based on run-time sensor readings. Once the sensor-based system is deployed, the sensors begin to collect run-time readings that reflect actual conditions, and the sensor readings are stored in a historical data repository. This repository serves as a point of comparison for identifying deviations in sensor readings. Through learning-based statistical analysis, the system uses historical data to detect trends, anomalies, or outliers during the sensor operation time. A statistical sensor model is built for each sensor to compare run-time readings against historical data. These statistical models identify patterns in sensor data that may indicate abnormal behaviour, such as sensor drift, malfunction, or environmental interference. This learning-based component thus helps to quantify the trustworthiness of each sensor by detecting changes in sensor behaviour that could impact system reliability. The sensor's trustworthiness is evaluated and assigned a score, representing its reliability at that moment. For instance, a sensor that frequently deviates from expected patterns may receive a lower trust score. When a sensor's trustworthiness drops below a certain threshold, this information is flagged, signalling a potential abnormality. Such feedback enables the system to handle unreliable data, maintaining model integrity despite sensor inconsistencies. The run-time system model, which is essentially a probabilistic state transition matrix of the system behaviour, gets updated, and the model is verified each time a sensor's quantified trustworthiness changes. Property Checking is applied to this run-time system model to ensure ongoing compliance with system specifications. In contrast to the one-time static verification, this real-time property checking continuously validates system

behaviour against specifications, taking current sensor trust levels into account. If the property checking process detects a violation or unexpected behaviour, it generates feedback indicating potential system errors or sensor malfunctions. A critical feature of the run-time verification framework is the adaptive feedback loop, which guides real-time updates to the physical system based on the results of property checking and sensor trust assessments. For example, if a sensor's readings are found to be unreliable, the system may initiate recalibration, increase redundancy by cross-referencing with other sensors, or adjust decision-making algorithms to mitigate the impact of unreliable data.

With the quantification of sensors' trustworthiness and its integration with model-checking methodologies, a run-time verification model is continuously adapted based on the results from property checking and sensor trustworthiness assessments. By recalibrating the model dynamically, the system maintains a closer alignment with real-world conditions, enhancing resilience against unforeseen sensor errors or environmental changes.

This chapter further proposes that although each individual sensor reading's trustworthiness can be quantified and evaluated, all the sensors in a sensor network also represent the collective reliability of interconnected sensors and, hence, feed into the broader trustworthiness of the entire sensor network. With this, a run-time model consisting of multiple sensor networks can be defined to monitor and verify Industry 4.0 applications that involve interactions between multiple sensor networks. Based on each sensor network's trustworthiness, the run-time model checking process is performed to verify the overall system behaviour in real-time. Finally, the run-time verification approach compositional modelling is proposed to support run-time model checking and ensure that system verification processes continuously adapt to sensor status updates, enhancing the robustness and accuracy of system monitoring. In summary, the following are the four key contributions of this run-time model-checking approach.

- Quantify the run-time sensor's trustworthiness using learning-based algorithms from historical data.
- Take into account the quantified sensor's trustworthiness to update the system model at discrete intervals to reflect run time system behaviour.
- Continually checking properties of the system model at each update of the sensor trustworthiness changes.
- The property checking results guide physical system updates to ensure the safety and quality of Industry 4.0 systems.

Figure 3.2 presents a more detailed and structured framework for understanding and analysing the runtime modelling and verification of critical sensor-based systems, gradually increasing in complexity from sensor-based to network-based systems, and followed by compositional modelling for multiple connected systems.

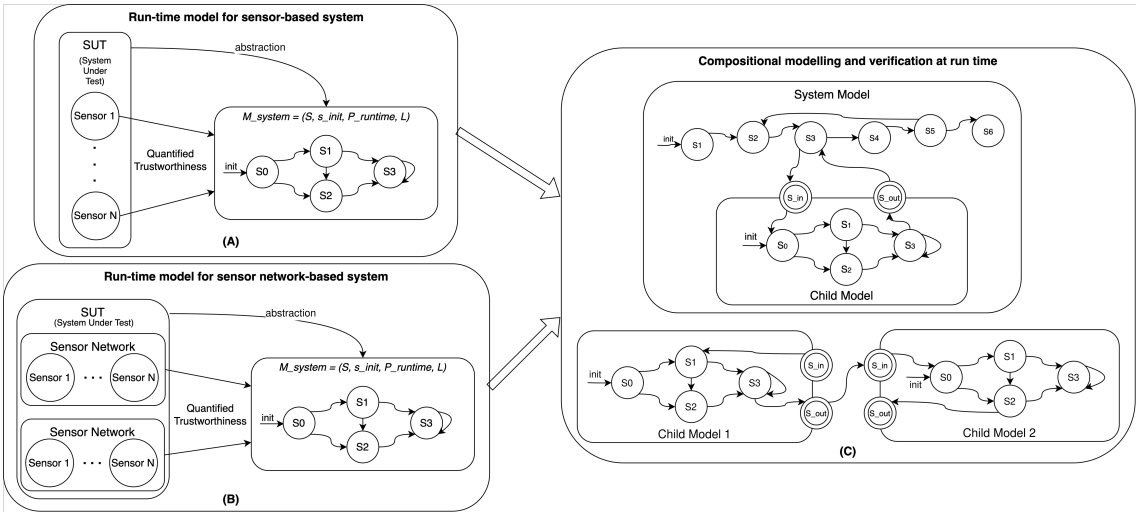

Figure 3.2: Overview of run-time probabilistic model checker (A) Run-time verification for sensor-based systems, (B) Run-time verification for sensor network-based systems, and (C) Compositional modelling and run-time verification.

Run-Time Model for Sensor-Based Systems The System Under Test (SUT) consists of multiple individual sensors, from Sensor 1 to Sensor N. These sensors continuously collect data from the physical environment. The sensor data is evaluated to provide a measure of *Quantified Trustworthiness*, which serves as an essential input for the abstracted run-time system model.

The system is abstracted into a formal model, represented as $M_{system} = (S, s_{init}, P_{runtime}, L)$, where S denotes the set of system states. s_{init} represents the initial state of the system. $P_{runtime}$ indicates the probabilistic transitions between states. L is a labelling function that associates observable properties with states.

The state transition diagram, denoted as S_0, S_1, S_2, S_3 in Figure 3.2, provides a graphical representation of how the system evolves over time, capturing the run-time behaviour of sensor and evaluating the effects at the system-level during the operation period.

Run-Time Model for Sensor Network-Based Systems The modelling framework is further extended to support a sensor network-based system. The SUT now comprises multiple sensor networks, each network containing several individual sensors, e.g., Sensor Network 1 with Sensors 1 to N. These interconnected sensor networks collectively monitor the system's environment, providing a richer, more complex dataset than a single sensor.

The abstraction process again results in the formation of an overarching run-time system model M_{system} with the same components $(S, s_{init}, P_{runtime}, L)$. In this context, M_{system} captures the behaviour and interactions of multiple sensor networks. The inclusion of sensor networks introduces a higher level of complexity, requiring more sophisticated modelling to accurately represent the run-time behaviour and trustworthiness of the system.

Compositional Modelling and Verification at Run Time The final component demonstrates the compositional modelling and verification approach for sensor- and sensor-network-based systems at runtime. Composition allows two distinct models, Model 1 and Model 2, to communicate and exchange data through defined interfaces, s_{in} and s_{out}, each with its own set of states, S_0 to S_3. These models represent modular components of the overall system.

The *System Model* M_{system} is an integrated representation that combines individual models into a unified structure. It includes *Child Model* nested within the larger system structure, which captures detailed internal behaviour. This compositional approach allows for scalable and modular verification, enabling the system to handle increased complexity while ensuring that both local (component-level) and global (system-level) properties are validated.

This chapter presents the run-time verification framework consisting of the following:

1. Quantification of Sensor Run-time Trustworthiness
2. Quantification of Sensor Network
3. Run-time model checking for sensor-based and sensor network-based systems
4. Compositional modelling and run-time model checking

3.1 Quantification of Sensor Run-time Trustworthiness

In order to quantify the trustworthiness of the sensors, Figure 3.3 illustrates a structured methodological framework for the profiling of sensors, encapsulating four primary stages.

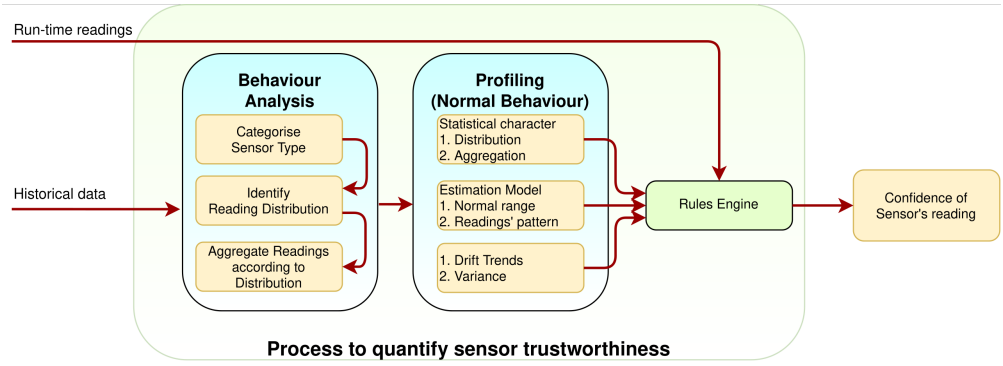

Figure 3.3: Sensor trustworthiness quantification process.

The first step of the process is the *Categorisation of Sensor Type*, which involves the systematic classification of sensors based on their functional characteristics, data output, or specific application domains. This step is critical as it sets the foundation for subsequent analysis by profilling the sensor within its appropriate category.

The second stage, *Identification of Reading Distribution*, entails a rigorous examination of the sensor's data output to discern the underlying statistical distribution of the readings. This involves the application of statistical tools and techniques to determine whether the data conforms to a known distribution model, such as a normal, uniform, or skewed distribution. Identifying the distribution pattern is essential in understanding the variability and central tendencies of the sensor data.

Following the identification of the distribution, the process advances to the *Aggregation of Readings* according to distribution. In this phase, the sensor readings are systematically aggregated based on the identified distribution model. This aggregation could involve summarising the data through statistical measures such as mean, median, or variance or applying more complex data aggregation techniques suited to the identified distribution. This process aims to consolidate the sensor data into a coherent form that accurately reflects the distribution characteristics.

The final stage is the *Profiling of the Sensor*, where a comprehensive sensor profile is developed based on the aggregated data. This profile encompasses various performance metrics, including accuracy, reliability, and consistency of the sensor's output. The sensor profile serves as an evaluative tool, providing insights into the sensor's normal behaviour and its overall operational availability.

This methodological approach ensures a thorough and systematic analysis of sensor behaviour, quantifying sensor trustworthiness during run time in sensor network-based industrial applications.

3.1.1 Categories of Sensors

In the industry context, a sensor is a device that detects and measures specific physical properties, such as temperature, pressure, humidity, or motion and converts these measurements into signals that can be interpreted by electronic systems. Sensors play a pivotal role in industry applications, serving as the primary interface between physical phenomena and digital systems. Sensors can be categorised into several types based on the nature of the data they produce, such as real number readings, enumerate readings, or binary readings. This research begins with a specific scenario, for instance, one type of sensor, to ground the discussion in a specific context before moving toward broader generalisations. Sensors used in industrial manufacturing are typically categorised into the following types:

- Real Number Readings: These readings provide continuous, quantitative values representing precise measurements. For example, temperature, humidity, pressure, and accelerometer sensors frequently produce real-number outputs to capture high resolution and accuracy that can represent slight variations in the measured phenomenon.

- Enumerate Readings: Enumerated readings represent discrete states or predefined categories, often used when sensors classify or detect types of objects, scenes, or environmental conditions. Enumerate sensors are typically used when the exact measurement is not critical, but rather the category or range of measurement is more relevant. For example, detecting positions such as "closed, open, intermediate" or level detectors, such as "empty, half-full, full".

- Binary Readings: The binary output is usually a sequence of bits processed that represents complex data, such as audio, visual, or other signal-based inputs. For instance, image sensors produce binary data that is processed to form digital images, while audio sensors may output binary data corresponding to sound waveforms.

In this research, time series readings are particularly focused. Time series sensor readings are sequences of data points collected or recorded at successive points in time, typically at uniform intervals. They capture how a particular measurement changes over time. Time series data can originate from any type of sensor, including real number sensors, enumerate sensors and binary sensors, as long as the readings are taken over time. They are commonly used in monitoring, predictive maintenance, trend analysis, and forecasting. These types of sensor data are the foundation of various industries, and the choice of data type significantly affects the methods used to quantify trustworthiness.

3.1.2 Identify Reading Distribution

Typical sensor failure types and detection methods have been discussed in Section 2.2. Data-driven techniques, such as statistical analysis, are commonly used to measure the trustworthiness of sensors and sensor networks at runtime, distinguishing between normal operational variations and sensor failures.

Two typical approaches are employed to detect the distribution of sensor readings in the manufacturing industry for sensor reading analysis, namely *Visual Inspection* and *Automated Statistics Calculation*.

- *Visual Inspection* involves using graphical methods to understand a dataset's shape, central tendency, spread, and other characteristics. By examining visual representations of data, one can gain insights into its underlying structure, identify patterns, detect anomalies, and make inferences about its distribution. Common tools for visualising data distributions include Histograms, Kernel Density Estimation (KDE) plots, and Quantile-Quantile (Q-Q) plots. Each method offers a unique perspective on the data, allowing for a comprehensive understanding of its distributional properties.

  Histogram is a type of bar chart representing a dataset's frequency distribution. It is constructed by dividing the data into intervals, known as "bins", and counting the number of

observations that fall into each bin. To create a histogram, the range of data is divided into consecutive, non-overlapping intervals or bins of equal width. The bins represent the range of values, and each bin's height corresponds to the number of data points within that range. The area of each bar in a histogram represents the frequency of data within each bin. Histograms provide a visual summary of the data's distribution, helping to identify its central tendency, including mean, median, mode, spread (range, variance, standard deviation), and overall shape, e.g., normal, skewed, and bimodal. The shape of a histogram reveals information about the distribution of the data. For example, if the histogram is roughly symmetrical with a peak in the centre, the data may be normally distributed. In case the histogram has a long tail on one side, the data may be skewed to the left or right. In another scenario, there are multiple peaks, the data may have various modes, indicating the presence of subgroups or a mixed distribution.

KDE plot is a non-parametric way to estimate the Probability Density Function (PDF) of a continuous random variable. It smothens the data points to produce a continuous curve that represents the data's distribution. KDE is constructed using a kernel function, e.g., a Gaussian function, applied to each data point. The kernel function spreads out each data point into a curve, and these curves are then summed to create a smooth density estimate. The bandwidth parameter controls the width of the kernel, affecting the smoothness of the KDE plot. KDE plots are used to visualise the distribution of a dataset in a smooth, continuous manner, making it easier to identify the underlying shape and structure of the data distribution without the binning artefacts present in histograms. The KDE plot shows peaks where data points are concentrated and valleys where data points are sparse. A smoother KDE plot can indicate a general trend, while a more rugged plot may reveal more detailed features of the data. A small bandwidth may result in a very wiggly plot with a lot of noise, while a large bandwidth can oversmooth the data, masking important features. KDE Provides a smooth and continuous estimate of the data distribution. Moreover, it reduces the artefacts and discontinuities introduced by histograms and provides flexibility in adapting to various data shapes with appropriate kernel and bandwidth choices. However, choosing the proper bandwidth is critical. If the bandwidth is too small or too large, it can misrepresent the data.

Q-Q plot is a graphical tool used to compare the distribution of a dataset to a theoretical distribution, often normal distribution or another dataset. It helps in assessing whether the data follows a particular distribution. Q-Q plots are created by plotting the quantiles of the data against the quantiles of a theoretical distribution or another dataset. If compared to a normal distribution, the data exhibits similar quantiles to a standard normal distribution. If the data follows the theoretical distribution, the points will approximately lie on a straight line. Q-Q plots are mainly used to visually assess the normality of a dataset, but they can be applied to compare any two distributions. The closer the points are to the reference line

(usually the 45-degree line), the more closely the data matches the theoretical distribution. The points will closely follow the reference line if the data is normally distributed. In another case, if the points curve away from the reference line at the ends, this indicates heavier tails than the theoretical distribution. Furthermore, a deviation in the middle or ends of the Q-Q plot may indicate skewness or other deviations from normality. Q-Q plot is more informative than a histogram or KDE when assessing normality and is effective for comparing the distribution of a dataset to a theoretical distribution. However, on the downside, the Q-Q plot can be subjective, especially for small sample sizes and does not measure the distribution's shape, spread, or central tendency.

- *Automated Statistics Calculation* refers to the process of programmatically computing key descriptive statistics and applying statistical tests to a dataset. This approach removes manual calculations and visual inspection, allowing for rapid, objective, and consistent analysis across large or multiple datasets. There are five key components of automated statistics calculation.

  Descriptive Statistics summarise sensor reading's main features through numerical calculations. These statistics offer a way to quickly understand the reading's central tendency, spread, and overall distribution, which is essential for extracting the sensor's working behaviour.

  Goodness-of-Fit Tests are statistical tests used to determine how well a sample data fits a distribution from a population with a specific theoretical distribution. These tests help decide whether the data follows a hypothesised distribution and whether deviations are statistically significant.

  Automated Distribution Fitting involves using algorithms to fit various statistical distributions to the data to determine which distribution best matches the observed data. This step is crucial for understanding the underlying patterns and modelling the data appropriately.

  Automated Parameter Estimation involves determining the parameters of the chosen statistical model or distribution that best describes the observed data. It is a critical step in modelling and helps summarise the data succinctly.

  Model Comparison Metrics are used to evaluate and compare multiple statistical models to determine which one provides the best fit for the data while balancing complexity and goodness-of-fit.

Both approaches are employed to quantify sensor trustworthiness for the proposed run-time verification framework. Comparing these two approaches, *Visual Inspection* is an intuitive initial step in data analysis that uses graphical methods to understand sensor readings distribution and identify patterns. In contrast, *Automated Statistics Calculation* provides an objective and consistent approach to analysing the readings through the programmatic computation of descriptive

statistics and the application of statistical tests. Together, both methods form a comprehensive approach to reading analysis, combining human intuition with computational efficiency and rigour.

In Industry 4.0 applications, the trustworthiness of the sensors refers to the degree to which they reliably provide accurate, consistent, and timely data. Each of these factors, *reliability*, *accuracy*, and *consistency*, plays an interdependent role in ensuring that sensors contribute to safe, efficient, and predictable industrial operations. According to *ISO/IEC 25012 data quality model*, the following are the considerations to quantify sensor trustworthiness.

- **Reliability** $R(S)$: The probability of sensor S_i's ability to function without failure over a specified period under predefined conditions.
- **Accuracy** $A(S)$: The degree to which the sensor data aligns with the true values it is intended to measure under similar conditions. The resulting values are normalised to a range between 0 and 1, ensuring that the sensors' results are scaled proportionally within this interval. This normalisation process preserves the relative magnitude of the original data while constraining it to a standardised scale, facilitating comparative analysis and improving the stability of subsequent computations.
- **Consistency** $C(S)$: The degree to which the attributes of sensor data are free from contradiction and are coherent with other data in a specific context of use. In the industrial sensor context, it refers to the stability of sensor readings over time, for example, the absence of significant variance in measurements and reading drift.

Thus, for an individual sensor, trustworthiness is expressed as:

$$T(S) = conf(R(S), A(S), C(S)) \tag{3.1}$$

The function $conf$ involves a weighted sum, where each factor is assigned a weight based on its importance to the specific application of the sensor network. For example, in real-time monitoring systems, low latency is prioritised, while in environmental monitoring, long-term accuracy and consistency are more critical. These weights quantify the influence or confidence of each factor in the decision-making process. Typically, the weight of each factor is decided by the following methods.

- **Expert Judgment**: Domain experts assign weights based on the perceived importance or reliability of each factor. This approach relies on human expertise and intuition. This is the most common way in the industry because it is hard to capture sufficient historical data to represent all possible scenarios and derive the weights using algorithms.
- **Statistical Analysis**: Historical data is analysed to determine how often each incoming reading leads to the ground truth. The reading data sets that have a higher trust rate are

assigned greater weights. This is the case after the system is deployed in the field and the reading is captured. This method is powerful enough to continue to optimise function $conf$ to desire an accurate representation of the sensor's trustworthiness.

- **Machine Learning Techniques**: Algorithms adjust the weights based on training data. Methods like reinforcement learning or optimisation algorithms can also fine-tune the weights to improve the performance of the function $conf$. Compared to statistical analysis, machine learning is better suited when the volume of sensor data is large and requires automation and handling of complex patterns that are difficult to model with traditional statistical methods.

With these two steps, the first is to quantify three factors of each sensor and, subsequently, assign a weight to each factor, the trustworthiness of the sensor can be computed at run time to reflect its real-world working conditions.

3.2 Trustworthiness of Sensor Network

A sensor network is a collection of spatially distributed sensors that collaboratively monitor and record working conditions over a particular area or system. The reliability of the data collected by sensor networks is crucial, as even a single sensor failure can lead to significant errors, impacting the accuracy and behaviour of the entire sensor network. Extending the quantification of the sensor's trustworthiness thus becomes an essential aspect of sensor network management. The ability to trust each sensor's data is essential for the overall reliability and efficiency of the sensor network and the entire system.

A relationship function is involved in quantifying the sensor contextual correlations within a sensor network. To define a relationship function that reflects the trustworthiness of a sensor network, the concept of sensor-network reliability is formalised as a function of individual sensor performance and overall network dynamics. Let $T(S_i)$ represent the trustworthiness of an individual sensor S_i within the network. The trustworthiness of the network, T_{net}, can be represented as an aggregate function of all n individual sensors' reliability, data accuracy, and consistency:

$$T_{net} = agg(T(S_1), T(S_2), \ldots, T(S_n)) \tag{3.2}$$

To reflect the trustworthiness of the overall sensor network, a function agg is performed to aggregate the trustworthiness of each individual sensor. A common approach is to use weighted averages or other data fusion methods, accounting for redundancy and correlation between sensor readings. The following are three common approaches proposed.

- *Statistical Correlation* measures the strength and direction of the relationship between two variables. It quantifies how one variable tends to change when the other variable changes.

Correlation is typically expressed by the correlation coefficient, which ranges from -1 to 1. Common methods to calculate correlation include *Pearson Correlation Coefficient*, *Spearman's Rank Correlation* and *Mutual Information*. *Pearson Correlation Coefficient* measures the linear relationship between two sensors' data. If the correlation coefficient is close to 1 or -1, the sensors are strongly correlated. *Spearman's Rank Correlation* is a non-parametric measure of rank correlation, it is useful when the relationship is not linear. *Mutual Information* quantifies the amount of information obtained about one sensor through another and is also widely used for non-linear relationships.

- *Regression Models* are statistical tools used that help in understanding relationships between multiple variables. In Industry 4.0, sensor networks are used to consolidate multiple sensor readings when making a decision. The following are several common regression models. *Linear Regression* assumes a straight-line relationship between the variables, *Logistic Regression* is used for binary readings, where the sensor reading is categorical, such as on/off, *Polynomial Regression* extends linear regression to model non-linear relationships by including powers of the independent variable(s) and *Multivariate Regression* is similar to linear regression. However, multiple dependent multivariate-sensors are considered simultaneously.

- *Machine Learning Models* are algorithms that enable computers to learn patterns from data for understanding the relationships between individual sensors. For instance, *Neural Networks* can model complex non-linear relationships between multiple sensors, *Long Short-Term Memory (LSTM) networks* are useful for time-series data, *Random Forest* can be used to model relationships by treating sensor data as features and finding patterns or correlations and *Support Vector Machines (SVM)* are useful for finding complex relationships in multi-dimensional data.

In summary, the relationship function $T_{net} = agg(T(S_1), T(S_2), \ldots, T(S_n))$ aggregates the trustworthiness of individual sensors to provide a comprehensive measure of the network's trustworthiness, which is crucial for ensuring the validity and resilience of sensor network-based systems.

In the next chapters, three experiments are presented to demonstrate the quantification process of sensor and sensor-network trustworthiness.

3.3 Run-time Model Verification with Trustworthiness

Industry 4.0 applications involve extending traditional verification techniques to address the complexities of highly interconnected, automated, and data-driven systems. In Industry 4.0, sensor network-based systems are characterised by dynamic interactions, uncertainty, and real-time data flows between devices, machines, and human operators. The proposed run-time prob-

abilistic model checking approach addresses these stochastic behaviours by modelling the sensor's uncertainties and incorporating these uncertainties to verify aspects such as system performance, potential failures, or decision-making processes. This involves creating formal models that embed the quantified trustworthiness of sensor-based devices and networked systems and composing multiple system models to verify at run time.

3.3.1 Run-time Model Verification

The proposed run-time model verification process involves three main steps, including defining the system model according to system specifications, defining property specifications using specific language, and verifying the system continuously when the sensor trustworthiness changes. The system model includes states and transitions, where transitions are associated with probabilities and actions.

Define run-time system model In line with the traditional formal model, the system states and transitions are defined according to the system specification, which means that they should not change during the operation according to the actual running environment. However, with the ability to quantify each sensor's and the sensor network's trustworthiness through $T(S)$ and correlation T_{net}, the transition probability can be updated dynamically. In this case, the run-time system model should employ the sensor network trustworthiness to form the run-time transition matrix during the system working stage. Below is the model definition of a run-time system model.

$$M_{runtime} = (S, s_{init}, P_{runtime}, L) \tag{3.3}$$

where $M_{runtime}$ is the run-time probabilistic model, S is a finite set of machine states of the system, $s_{init} \in S$ is the initial state, $P_{runtime} : S \times S \to [0,1]$ is the run-time transition probability matrix where $\sum_{s' \in S} P(s,s') = 1$ for all $s \in S$ and $L : S \to 2^{AP}$ are function-labelling states with atomic propositions.

Property specification In the Industry 4.0 context, the properties to be verified are typically expressed using temporal logic, such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL), which can specify requirements like safety or expected behaviours. Probabilistic Computation Tree Logic (PCTL) is an extension of Computation Tree Logic (CTL) designed to express and verify properties of systems that exhibit probabilistic behaviours. It is widely used in the verification of stochastic models such as Markov Decision Processes (MDPs) and Discrete-Time Markov Chains (DTMCs). PCTL extends the expressiveness of traditional temporal logic by incorporating probability thresholds, enabling reasoning about the likelihood of specific states occurring. The logic allows for the specification of quantitative properties, such

as the probability of reaching a certain state or the likelihood of a sequence of events occurring within a given time period. Below are some example use cases of PCTL specification.

Example 1: The following expression specifies that the probability of eventually reaching the state *target_state* is at least 0.95:

$$P_{\geq 0.95}[F\,\text{target\_state}]$$

This means that with a probability of at least 95%, the system will eventually reach the state where the property *target_state* holds. For instance, in industry applications, the final completion state is defined as the target state, this property checking can be used to verify the system's 95% availability to ensure the whole process quality.

Example 2: The probability that a system eventually moves to an error state:

$$P_{=?}[F\,\text{error\_state}]$$

This computes the exact probability that the system will reach the error state at some point in the future. For example, an automated passenger lift relies on algorithms to make real-time decisions. A critical error state, such as door movement to stop a passenger from entering or colliding with an obstacle, could lead to accidents. This property checking helps calculate the probability of such an error state occurring, factoring in sensor failures, communication delays, and software bugs. Furthermore, this probability assessment helps manufacturers improve safety measures, reduce accident risks, and meet regulatory standards for autonomous systems.

Example 3: The specifies that the probability of reaching state *goal* within 10 steps is at least 0.9:

$$P_{\geq 0.9}[F^{\leq 10}\,\text{goal}]$$

This means the system will reach the goal state within ten steps with a probability of at least 90%. Consider a robotic assembly line in a manufacturing plant that produces electronic components. The assembly process has several stages, each with specific tasks that must be completed in sequence. For the production line to operate efficiently, the system needs to reach a goal state where a component is fully assembled and quality-checked within ten operational steps. By using this property checking, manufacturers can assess whether their production line meets target efficiency and reliability metrics. Ensuring a high probability of reaching the goal state within a specified number of steps supports consistent production output, reducing downtime and minimising costs associated with rework and delays.

This means that with a probability of at least 80%, *state A* will hold continuously until *state B* eventually holds.

Verification The model verification process involves exhaustively exploring all possible states of the system to determine whether the model satisfies these properties using model checker tools. In the case that the system does not meet the specified property, the model checker provides a counterexample, which is a sequence of states leading to the violation of the property. The proof of correctness is automated through algorithms that systematically explore the state space of the model, ensuring both completeness and correctness. PRISM is the robust probabilistic model-checking tool that supports various models, including DTMCs, CTMCs, MDPs, and PTAs, enabling the verification of probabilistic properties using logic like PCTL and CSL. In the following experiences, PRISM is used for quantitative analysis of the proposed verification approach.

In chapter 4, an experiment was set up to evaluate this approach using an actual sensor-based machine.

3.4 Compositional Modelling and Run-time Verification

To handle the probabilistic nature of such run-time industrial processes, the proposed run-time model is extended to two types, namely *Child model* and *System model*. *Child model* aims to abstract minimum and completed working unit, e.g., a turn-mill machine that can work independently to complete a designated work. *System model* targets modelling the system at a higher level, for instance, manufacturing management systems or scheduler systems, which is shown in Figure 1.1, where it coordinates multiple working modules (child models) to complete a more complex job. These two types of models work together and are capable of abstracting Industry 4.0 applications on a modular basis and composing at run time to represent an operation process.

Child Model In order to support the verification of compositional structure, a new type of state of the system model is introduced as interfaces with other models. This new state type is a special state that acts as the edge of a model to interact with not only internal states but also other models. These interface states have only one direction for the transitions. For instance, the interface state S_{in} only accepts incoming transitions from external and outgoing to internal states. Conversely, the interface S_{out} transits internal states to external models only.

Figure 3.4 shows the model with interface states. On the left side is the traditional model representing a basic isolated system with four states: S_0, S_1, S_2, and S_3. S_0 is the initial state, and transitions are defined between the states. Arrows between states represent transitions, showing how the system can move from one state to another. S_3 features a self-loop, indicating an internal or recurring process within this state. On the right side is an extended model, named

Child Model, where additional states (S_{in} and S_{out}) have been introduced. The core states S_0, S_1, S_2, and S_3 from the original model are retained, with the same transition relationships as before. The addition of s_{in} and s_{out} introduces an interface layer for the child model, allowing it to be integrated with other models in a larger system. S_{in} acts as an entry point for interactions from an external or parent system. It connects to state S_0 as an incoming synchronisation or message receipt mechanism. S_{out} provides an exit mechanism to interface with external systems. It connects to S_3, indicating that this state can lead to an external synchronisation or signal that the process is complete or requires interaction outside the child model.

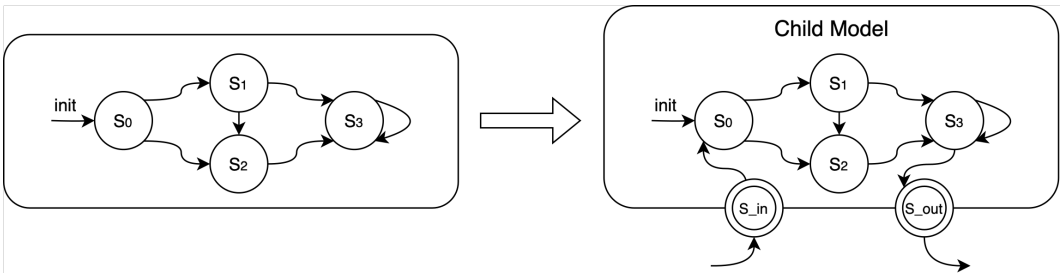

Figure 3.4: Child model with interface extension.

Meanwhile, the traditional verification approaches are still applicable as per normal with the definition of states and transitions. With this, extended child model interacts with other models, the internal states are transparent to external models and only interface states are exposed and represent this child model.

The *child model* is defined as a six-tuple to provide the capability to be integrated with other models.

$$M_{child} = (S, s_{init}, P_{runtime}, L, S_{in}, S_{out}) \tag{3.4}$$

where S, s_{init}, $P_{runtime}$ and L are defined as in the run-time probabilistic model, M_{child} and two new elements, S_{in} and S_{out} are added. $S_{in} \subset S$ is a set of the entry states of the model, while $S_{out} \subset S$ is a set of the exit states of the model. Moreover, the existing property specifications are still applicable to the M_{child}.

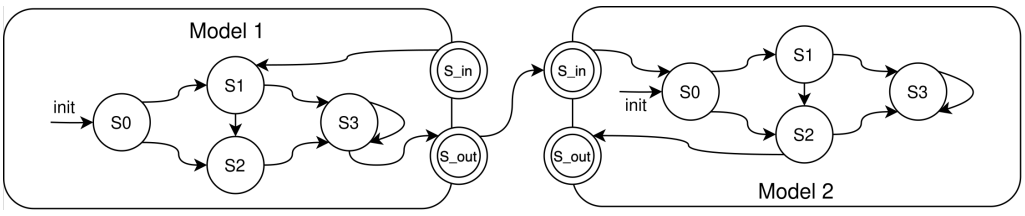

Figure 3.5: Interaction of child models.

Figure 3.5 shows two child models that interact with each other through extended interface states. In this scenario, *Model 1* and *Model 2* all come with S_{in} states and S_{out} states. The key aspect of this diagram is the bidirectional synchronisation between *Model 1* and *Model 2* through their shared S_{in} and S_{out} states. Model 1's S_{out} connects to Model 2's S_{in}, indicating that Model 1 can send a signal to trigger state changes in Model 2. This interaction represents a compositional model where different subsystems communicate through well-defined interface states, allowing verification of their individual behaviours in the context of the overall system. This extended model is capable of representing normal *Task Operation* (c.f. Chapter 1) of Industry 4.0 applications, which is able to work and complete a task independently.

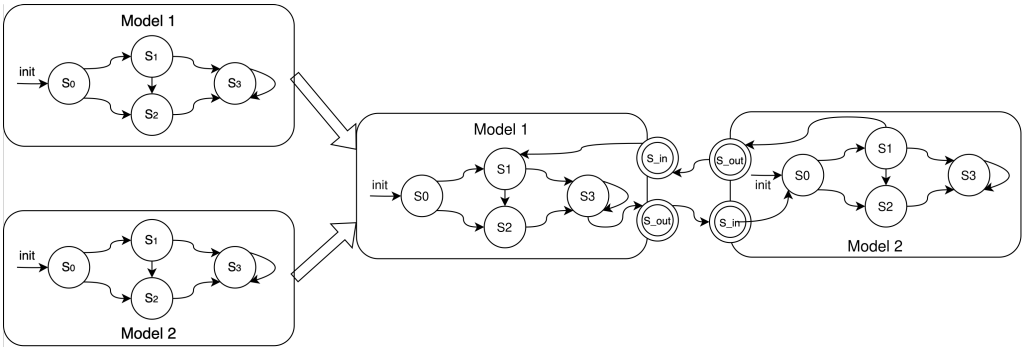

Figure 3.6: Interaction of child models.

Figure 3.6 shows another scenario in which two additional transitions form an interface between states: S_{in} and S_{out}. These transitions S_{in}, S_{out} serve as *entry* and *exit* transitions between the two systems, establishing a synchronised communication mechanism. For instance, S_{in} connects S_0 to the initial state of Model 2, S_{out} is connected from S_1 in Model 2 and connects to the S_{in} of Model 1. This model represents a scenario where external inputs or synchronisation points are introduced to enable inter-model communication in a distributed or compositional system. This case is suitable for the workflow of a manufacturing process in that one work cell completes the task and informs the next work cell.

System Model For higher-level management or scheduler systems, the traditional model or the *child model* makes it hard to abstract the interactions. Figure 3.7 represents another improvement of the traditional system model to represent the system from the system management level, which needs to manage and coordinate sub-systems to work as a process. Using the interface state, the *system model* is capable of representing both management systems and task operation sub-systems, which is defined in Figure 1.1. This management system model is named *system model*, and the task operation layer model is *child model*. The system model interacts with the child model through the child model's interface states.

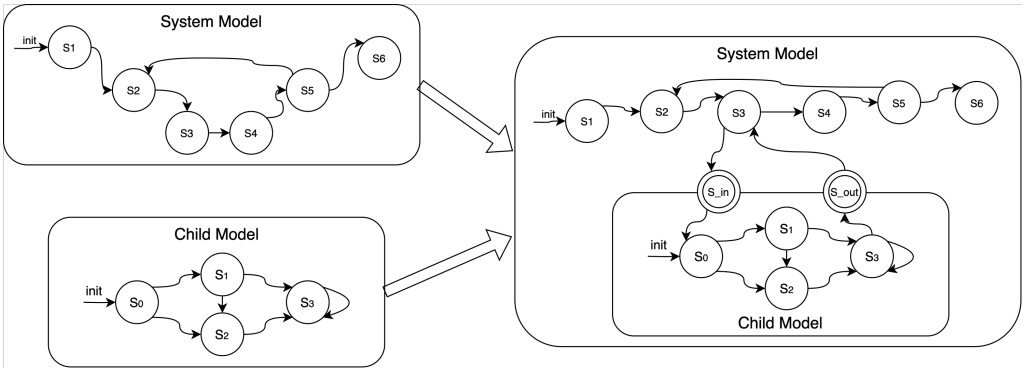

Figure 3.7: Structure of compositional modelling.

Figure 3.7 shows both the child model and the system model. These two models collaborate through interfaces of input and output states such that they can be integrated with each other easily. In the upper part of the figure, the system model is represented, which contains its own sequence of states: S_1 through S_6. Within this system model, the previously defined child model is embedded as a modular component. In the lower part of the figure, a child model is defined as an independent state machine consisting of four states: S_0, S_1, S_2, and S_3. Transitions between these states define the internal behaviour of the model. The transition system also includes two additional states, S_{in} and S_{out}, which serve as input and output interaction points for integrating this child model with the larger system, enabling the larger system model to interact with the internal states of the child model. This composition structure allows for localised verification of the child model's correctness independently while still ensuring that its interactions within the system model are consistent with the overall system requirements. The figure highlights the composition approach in model verification, where smaller, independently verifiable models are composed into a complete system model. Such an approach is used in probabilistic model checking, where the state space of large systems can be reduced by verifying sub-components individually. This method facilitates modular verification, enhancing scalability and reducing the complexity of system-wide property checking.

The *system model* is defined as below:

$$M_{system} = (S, s_{init}, P_{runtime}, L, M_{child_1}||...||M_{child_k}) \tag{3.5}$$

where M_{system} is the top level system model, S, s_{init}, $P_{runtime}$ and L are defined as in $M_{runtime}$, while $M_{child_1}||...||M_{child_k}$ is a set of the child models as defined in Equation 3.4) representing the sub-system of the top-level system. The operator $||$ represents the connection using the interface states of the child models.

Considering the relationship between the child models and the system model, the following requirements must be satisfied:

1. $S_{system} \cap S_{child_i} = S_{in_i} \cup S_{out_i} \quad \forall\, i \in \{1,...,k\}$

2. $S_{in_i} \cap S_{out_i} = \emptyset \quad \forall\, i \in \{1,...,k\}$

3. $S_{child_i} \cap S_{child_j} = \emptyset \quad \forall i \neq j$

4. $P_{system}(s,s',t) = 0$
   $\forall\, s \in S_{in_i}, s' \in S,\, t \in \mathbb{T}$ *and* $i \in \{1,...,k\}$

5. $P_{system}(s,s',t) = 0$
   $\forall\, s \in S,\, s' \in S_{out_i},\, t \in \mathbb{T}$ *and* $i \in \{1,...,k\}$

The conjunction of system model S_{system} and child models S_{child_i} should be the only interface states S_{in_i} and S_{out_i}. Also, there should not be overlap states between S_{in_i} and S_{out_i}, similarly for all the child models with no overlap states between child models. No transition of the system model is from S_{in_i} states or to S_{out_i} states at any time $t \in \mathbb{T}$, and the child model $i \in \{1,...,k\}$.

At the top level of the system model, there should not be input state(s) and output state(s) to ensure the system has finite states.

The semantics of the proposed approach are stated as follows:

1. $S = S_{system} \cup S_{child_i} \quad \forall\, i \in \{1,...,k\}$

2. $P(s,s',t) = P_{system}(s,s',t)$
   $\forall s \in (S_{system}/S_{in_i}), s' \in S, t \in \mathbb{T},$ *and* $i \in \{1,...,k\}$

3. $P(s,s',t) = P_{child_i}(s,s',t)$
   $\forall s \in (S_{child_i}/S_{out_i}), s' \in S, t \in \mathbb{T},$ *and* $i \in \{1,...,k\}$

In order to verify the system as a whole, the global state space is represented as S, which is the union set of all the system model's states S_{system} and child models' states S_{child_i}. The global transition matrix P concatenates the transitions of both the system model and child models.

With the semantics of the proposed approach and the three constraints above, the design guarantees that there are no overlapped states between the system-level model and the child models except the interface states.

Figure 3.8 shows the proposed approach that can be applied to industry 4.0 applications. With both types of *child model* and *system model*, the proposed approach allows child modules to be modelled and verified separately, the high-level system can be composed during operation stage.

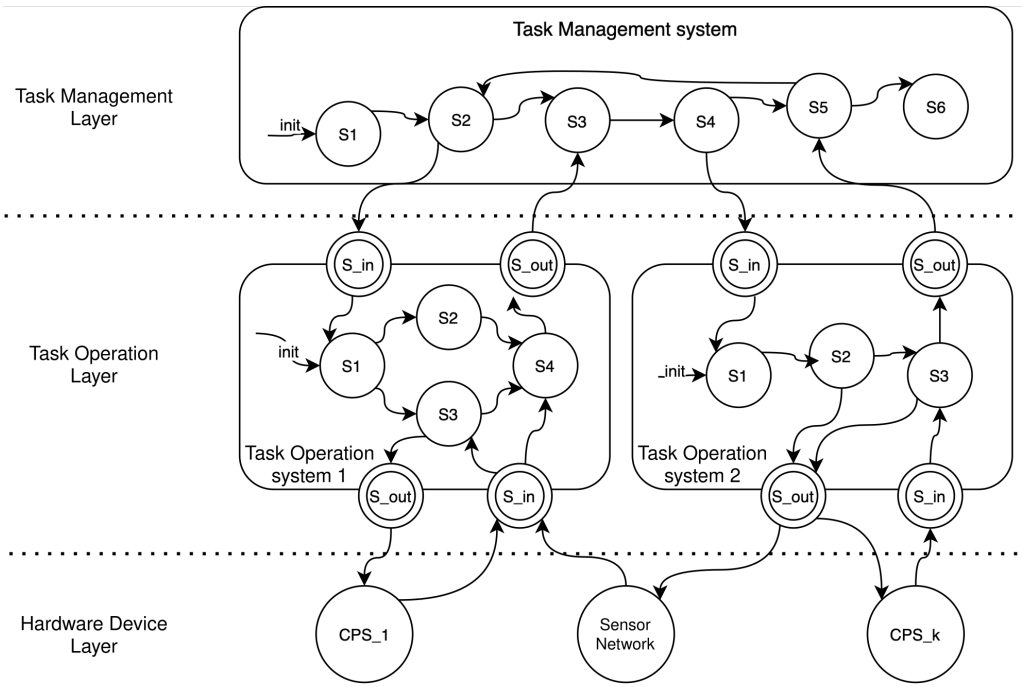

Figure 3.8: Structure of models for Industry 4.0 applications.

Task Management Layer : The topmost layer represents the Task Management System, which is responsible for managing high-level tasks and system states. The state machine within this layer consists of several states (S_1 through S_6), which capture the progression of tasks from initialisation to completion. The transitions between these states encapsulate the logic of task sequencing at a management level.

Task Operation Layer : Beneath the Task Management Layer, the Task Operation Layer is composed of two subsystems: Task Operation System 1 and Task Operation System 2. These systems operate in parallel and interface with the Task Management System through input and output states (S_{in} and S_{out}), which allow them to communicate with the higher-level management layer. Each operation system includes its own internal state machines, consisting of multiple states, S_1 through S_4, to handle the operational sub-tasks. This layer manages specific actions required for completing the high-level tasks managed by the top layer.

Hardware Device Layer : At the lowest layer, the Hardware Device Layer interfaces with physical components of the system, such as CPS and sensor networks. The state machines within this layer represent the physical devices' interactions, which may include hardware components like sensors or actuators. These devices communicate with the Task Operation Layer through states like S_{in} and S_{out}, bridging the gap between software operations and hardware-level actions. Multiple CPS components, CPS_1 through CPS_k, are shown interacting with the task operation systems and sensor networks, enabling real-world data collection and control.

Figure 3.8 demonstrates how this multi-layer system can be decomposed for modular verification. Each layer can be verified individually for internal correctness using model-checking techniques, ensuring that transitions between states within a layer conform to the desired properties. Once the individual layers are verified, the interactions between layers—specifically through states, such as S_{in} and S_{out}, can be verified to ensure that the task management system, task operation systems, and hardware devices operate cohesively.

This compositional structure supports modular verification, a technique that improves scalability by enabling localised reasoning about sub-systems. In the context of CPSs, this approach is crucial for managing the complexity of verifying both the software, task management and operation layers, the physical devices, and the hardware device layer.

In summary, the proposed approach supports modularity in that the top-level system model can be decomposed into lower-level child models to provide flexibility in verifying the system at different levels of abstraction. Each child model can be verified independently through traditional model checking during the design phase and they can be composed into a system model and then verified through model checking at run-time using the proposed approach to update its transition probability matrix. An experiment was set up using this scenario described in Chapter 6.

In summary, this methodology provides a comprehensive structure for capturing, analysing, and verifying the trustworthiness, reliability, and run-time behaviour of critical sensor-based systems in Industry 4.0 applications.

Chapter 4

Run-time Model Checking with Sensor Trustworthiness

This chapter explains how the proposed run-time verification model is applied in an Industry 4.0 context by modelling a turn-mill machine to facilitate continuous real-time verification of its behaviour. A turn-mill machine is a multi-functional machine tool that combines the capabilities of both turning and milling in a single setup. Turn-mill machines are widely used in Industry 4.0 applications due to their integration into smart manufacturing systems. These machines contribute to enhanced productivity, precision and flexibility, aligning with the goals of Industry 4.0, which emphasises automation, real-time data exchange and interconnectivity. A turn-mill machine is typically equipped with sensors, advanced control systems and data analytics functions to perform continuous monitoring and optimisation of the production process. The turn-mill machine's correct operations rely on accurate sensor data as its control actions are triggered based on the machine's sensor reading in real time. Therefore, any inaccurate sensor reading in the turn-mill machine may cause the machine to receive faulty feedback, leading to improper tool adjustments or incorrect machining parameters being provisioned. Consequently, this can potentially result in dimensional inaccuracies, defective parts and reduced product quality. Furthermore, faulty or inaccurate sensor readings could disrupt predictive maintenance algorithms thus causing missed maintenance schedules, as well as increasing the risk of unexpected machine failures or downtime. By modelling the turn-mill machine using the proposed run-time verification approach, the sensor reading's trustworthiness can be computed at run-time and fed into a probabilistic verification model, such that the system behaviour can be modelled more accurately based on the trustworthiness of the sensor readings real-time. With the model's verification output, the operators can then plan and take actions in advance if the turn-mill machine is predicted to fail, hence guaranteeing the manufacturing process quality.

The turn-mill machine was set up as the test bed that serves as an environment to explore run-time trustworthiness metrics under variable conditions. By incorporating the proposed run-time probabilistic model checking, the resulting analysis not only highlights the trust levels associ-

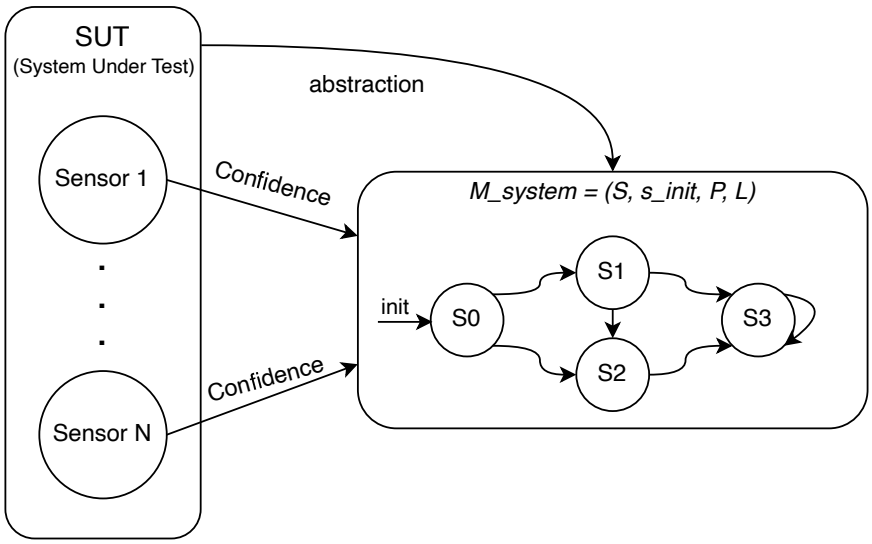

Figure 4.1: Overview of the run-time probabilistic model checker.

ated with sensor readings but also informs decision-making processes for real-time adjustments to maintain the efficiency of industrial operations.

4.1 Architecture of Run-time Model Checking

Figure 4.1 illustrates the architecture of the proposed run-time probabilistic model checker, which integrates model checker and data-driven sensor trustworthiness quantification approaches.

The *SUT* is a sensor-based system that consists of several sensors. The sensors collectively monitor certain conditions or parameters in an environment. During the run-time, all the sensors are continuously evaluated to quantify sensor trustworthiness using data-driven algorithms, resulting in a *confidence* score that represents each sensor's trustworthy level. Subsequently, these confidence scores are fed to the probabilistic system model to compute the transition probability between machine states, reflecting the real-time behaviour of the system. Due to the decision-making algorithms highly depending on the sensor outputs [119–121], the trustworthy level of the sensor affects the probability of transition of machine states, which affects the system behaviours accordingly. The confidence from each sensor influences the system model's probability of reaching certain states.

In this process, two crucial modules are designed and implemented, namely *Sensor Fault Detection (SFD)* and *System Model Verification (SMV)*. SFD is used to profile sensor behaviour and quantify run-time sensor trustworthiness. Meanwhile, the SMV module handles the probabilistic model checking and module evolution. Figure 4.2 illustrates the process of the implementation.

SFD is a data-driven approach following the proposed method in section 3.1 to profile sensor readings based on the historical dataset, namely the sensor's *normal behaviour*. The historical dataset is first collected with human supervision to ensure the sensor operates correctly. For

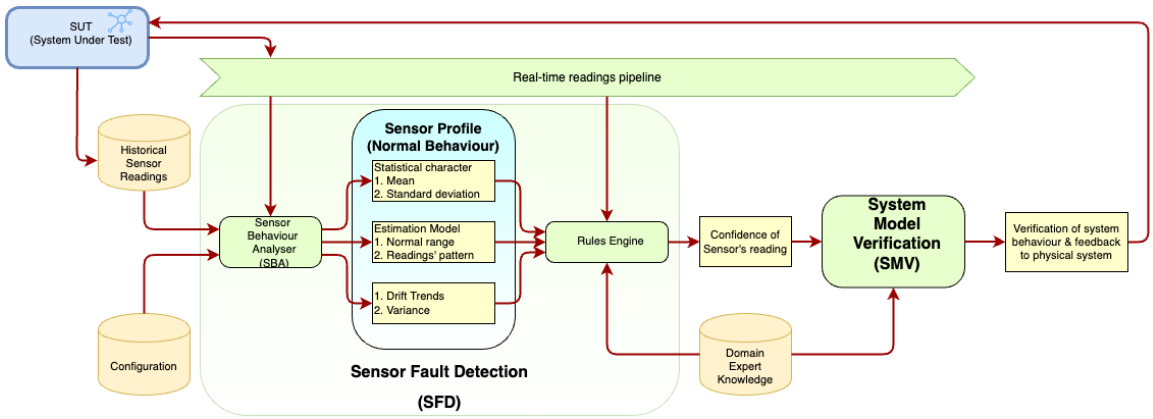

Figure 4.2: Modules of the run-time probabilistic model checker.

instance, this dataset can be collected just after the sensor's regular maintenance or calibration. Subsequently, while the machine is in operation, the run-time sensor's readings are continuously compared against the sensor's normal behaviour using time series analysis and estimation methods so that any deviation from the norm can be detected at run-time. The result is the quantified sensor trustworthiness, termed as the sensor run-time *confidence*. Whenever a sensor fault or deviation is detected, the state transition matrix of the base model is updated with a lower sensor *confidence*. Conversely, once the sensor turns back to normal working condition, the sensor *confidence* will be increased accordingly.

SMV is responsible for updating the model by performing property checking based on run-time sensor confidence. Firstly, a system-based model is defined according to the system specification or requirements to provide a system abstraction. This means manually specifying the initial transition probability matrix of the system model. Typically, this model is used to verify the system design at design time. Secondly, the sensors' confidence scores are fed into SMV, and SMV updates the transition matrix accordingly to reflect run-time system behaviour. Finally, the updated model is employed to verify system properties, with the verification results fed back to the physical system to guide operational processes or enhance overall quality.

With the combination of SFD and SMV, the system model evolves over time as the system continuously updates the sensor's behaviour through profiling and analysis of past sensor data. Essentially, this means that the model is updated continuously, taking into account run-time sensor data to derive the appropriate probability of state transitions. Hence, the proposed framework tracks both the sensor readings and the system expectation for verification and quantification. It enables the verification of the system's behaviour at run-time and, at the same time, assesses the system's reliability.

4.2 Sensor Fault Detection (SFD)

SFD is based on a set of quality evaluation metrics that are built from historical sensor data. In combination with a rule engine, SFD is able to quantify the trustworthiness of the sensor data at run-time. As shown in Figure 4.2, the SFD consists of two major modules, namely *Sensor Behaviour Analyser (SBA)*, and the *Rules Engine*.

SBA periodically learns about sensor behaviour from historical data, for instance, on a daily basis. In addition, SBA analyses run-time data obtained from the sensor reading pipeline in order to determine data patterns. The resulting *normal behaviour* profile for each sensor comprises the following three elements:

- *Statistical characteristics* — calculated over a window of samples. Windowing is done over the temporal domain on an individual sensor basis. The *mean* and *standard deviation* are the basic statistical measures often used to measure the reliability of a sensor. When the standard deviation is high (relative to expectation), it adds evidence to be classified as faulty data.

- *Estimation model* — uses data mining techniques, e.g., machine learning methods, to forecast data range and reading patterns. This process does not require knowledge from domain experts, the system learns from sensor data and machine states.

- *Drift trend* — is the direction in which the sensor reading is moving, usually away from its normal behaviour. Drift is typically a result of sensor wear and tear and calibration errors. As the sensor's readings are in time series format, an ARIMA [122] model is used to calculate the trend component in order to determine whether there are consistent deviations (in increasing or decreasing order) in the sensor readings over time.

The *Rules Engine* determines a sensor's *confidence score* based on the run-time sensor readings obtained from the pipeline by evaluating the degree of deviation from the sensor *normal behaviour*. As the ground truth is unknown, this deviation is usually termed as *fault*. Table 2.1 shows a list of fault types that are defined and can be detected by SFD. In order to detect faults, Equation 3.1 is applied with a set of rules defined based on the domain expert's knowledge. The sensor's *normal behaviour* includes the factors representing reliability, accuracy and consistency that are calculated by SBA and each rule is mapped to the factor. All incoming run-time readings are evaluated against these pre-configured rules and the sensor's *normal behaviour* to derive the *confidence score*. A sensor's *confidence score* is thus calculated as follows:

$$Conf_{sensor} = \sum_{i=0}^{n} factor_i \times weight_i \tag{4.1}$$

where n is number of the rules defined, $factor_i$ is the output of each rule i, and $weight_i$ is the weight of the factor of the rules.

Note that the sensor's normal behaviour comprises the three elements, *reliability*, *accuracy* and *consistency*, that are used to compute the sensor's confidence score, which is a representation of the trustworthiness of the sensor readings. This confidence score is subsequently fed into the SMV to update the transition matrix of the model during the system operation period.

4.3 System Model Verification (SMV)

SMV is a module of a probabilistic model-based model checker that verifies the working behaviour of SUT at run-time. Figure 4.3 illustrates the processing steps.

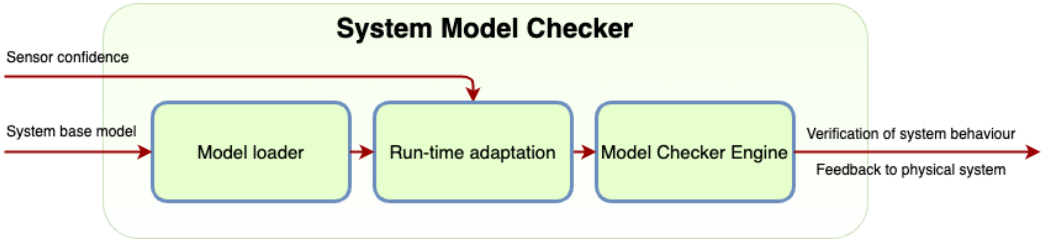

Figure 4.3: Run-time probabilistic model checker.

During the system operation phase, a system base model is first loaded into SMV. As defined in section 3.3, the system base model M_{base} as follows:

$$M_{base} = (S, s_{init}, P_{init}, L) \tag{4.2}$$

where M_{base} is the probabilistic model of SUT, S is a finite set of machine states of SUT.

With this definition, the states and transitions of the model are fixed according to the system specification and should not be changed during run-time. Only the probability of state transition in the model may be dynamically updated due to the uncertainty of run-time sensor status and its working conditions, e.g. unexpected sensor failure or drifted sensor readings. The initial probability of state transitions in the model must be defined based on the domain expert knowledge and the system specification. Subsequently, the transition probability matrix P evolves over time according to the sensor's confidence score obtained from SFD. The system model evolves to a run-time system model $M_{runtime}$ as follows:

$$M_{runtime} = (S, s_{init}, P_{runtime}, L) \tag{4.3}$$

where S is a finite set of machine states of the system, $s_{init} \in S$ is the initial state, $P_{runtime} : S \times S \times T \to [0,1]$ is the run-time transition probability matrix that the transition keeps updating over the finite time period $T \subset \{0,...,n\}$, where $\sum_{s' \in S} P(s,s',t) = 1$ for all $s \in S$, $t \in T$, and $L : S \to 2^{AP}$ are function-labelling states with atomic propositions.

This effectively models the behaviour of the overall system more accurately whenever faults are detected in the sensor readings, thus leading to better prediction of system failure.

In the next section, an experiment was set up to describe how the sensor confidence score is computed and how it updates the transition probability matrix.

4.4 Implementation and Experiment Design

A Computer Numerical Control (CNC) turn-mill machine is modelled using the proposed run-time probabilistic model checker. CNC machines, which rely on pre-programmed software to control the movement of tools and machinery, are widely utilised in industrial processes such as cutting, milling, turning, and drilling. Equipped with sensors, these machines generate substantial volumes of operational data that can be analysed to improve processes, enhance machine utilisation, and optimise energy consumption. As key components of Industry 4.0, CNC machines integrate traditional manufacturing with modern digital technologies. Verifying the operational behaviour and the trustworthiness of the sensors of CNC machines is crucial for ensuring the quality and reliability of the manufacturing process. Figure 4.4 shows a CNC turn-mill machine NTX1000 that is equipped with a sensor network to monitor its main spindle and cutting tool. Each monitored machine part consists of three types of sensors, namely *current* sensors, *vibration* sensors and *temperature* sensors. These sensors are connected to a data acquisition system, and the extracted sensor readings are streamed to the server via the Open Platform Communications-Unified Architecture (OPC-UA) protocol.

The proposed run-time probabilistic model checker was implemented to verify the CNC turn-mill machine based on the data flow. Specifically, the software modules implementation is presented with details of SFD (Section 4.4.1) and SMV (Section 4.4.2) as well as the system and environment properties (Section 4.4.3).

4.4.1 Sensor Fault Detection (SFD) Module

SFD was implemented using Python, and the interfaces were developed following RESTful architecture patterns. Historical data and configurations were provided through a web-based interface. Based on the historical readings, the sensor's normal behaviour was derived on a daily basis. During the system operation phase, the sensor readings were generated by SUT and published to *run-time readings pipeline*. With this, all processing modules subscribed to the *readings pipeline* to obtain the sensor data at run-time. SFD received the run-time readings to compute a confidence score for each sensor against its normal behaviour. Afterwards, this confidence score was fed into SMV to update the model's transition matrix.

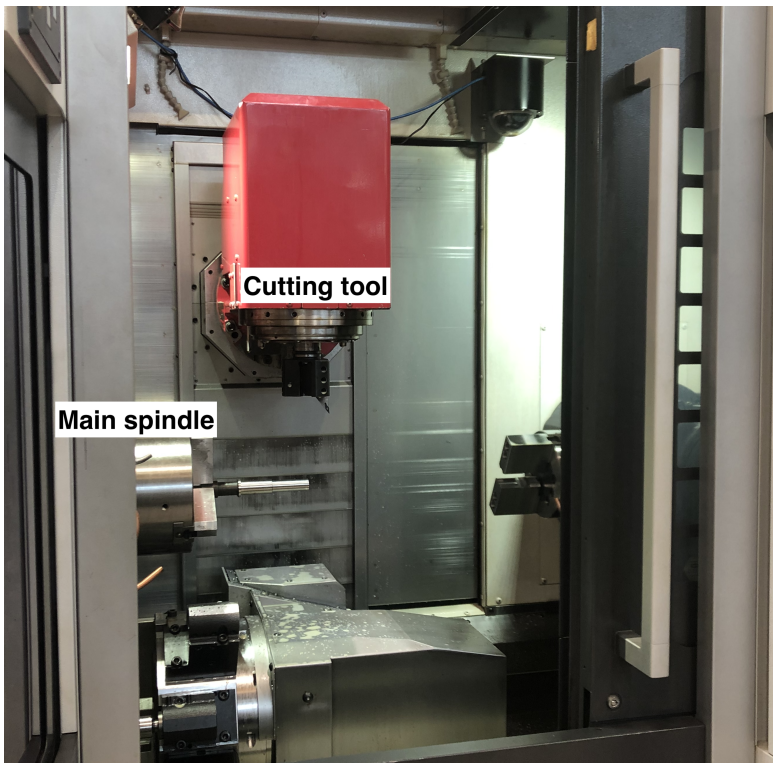

Figure 4.4: The location of sensors in a CNC turn-mill machine.

4.4.2 System Model Verification (SMV) Module

The SMV was implemented using PRISM 4.5 with a Java wrapper. In order to interact with the SFD, a web server was set up as a container to run the SMV module and exchange parameters.

A system probabilistic model was developed by domain experts based on domain knowledge and operating experiences. An actual machine operation case is illustrated to evaluate the proposed approach. Figure 4.5 shows the CNC machine model, which includes five states:

$$S = \{idle, mount, cut, unmount, error\} \tag{4.4}$$

and the initial transition matrix is defined as:

$$P_{init} = \begin{bmatrix} 0 & 0.999 & 0 & 0 & 0.001 \\ 0 & 0 & 0.999 & 0 & 0.001 \\ 0 & 0 & 0.6995 & 0.2995 & 0.001 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.5}$$

where the probability of each state transition in the matrix was defined according to expert knowledge and historical data. The initial probability of the transition matrix, P_{init}, assumes that the sensors are new and that they behave normally. The probability of system failure is

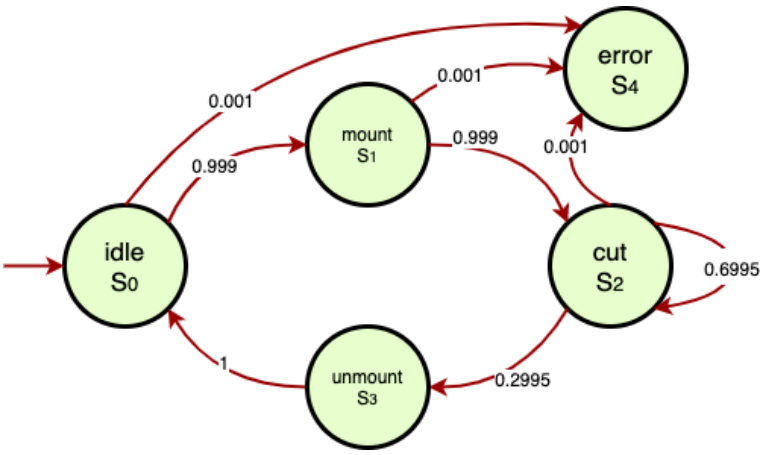

Figure 4.5: The state transition model of CNC turn-mill machine.

expressed by the following PCTL formula:

$$P_{error} =? \left[F^{\leq 60\times 60\times 24\times 7} (S_4) \right] \tag{4.6}$$

where P_{error} is the probability of system error in seven days. The error state S_4 is defined in the machine state in Figure 4.5. Industry typically operates on a weekly schedule, from Monday to Sunday, aligning maintenance activities with the end or start of a production week. Seven days minimise disruption to production and allow for systematic planning, facilitating resource allocation and staff scheduling. The typical failure cases that happen during idle, mounting and cutting stages, such as continuous operation without maintenance, lead to the spindle or bearing overheating if lubrication is insufficient.

Ideally, the system model should accurately reflect the system state at all times. In practice, however, the sensor-based system might be different, such as the unexpected sensor readings drift. To reflect run-time sensor trustworthiness is a crucial step that existing methods tend to ignore. As discussed, the deviation may be explained, among others, by the wear and tear of the sensors or sensor readings drift. In order to make the system model reflect the run-time system behaviour (updating evaluation and expectation in unison), the probability of each transition is updated according to the run-time *confidence score* of each sensor from the SFD. As each machine state is monitored by a set of sensors, the probability of transition to the *error* state, S_4 in Figure 4.5 should be derived based on the sensors' *confidence score*. The failure rate of each state is updated according to the following:

$$R_{failure} = \sum_{i=0}^{n} (1 - conf_i) \times \lambda_i \times w_i \tag{4.7}$$

where the transition to error state depends on the trustworthiness of n sensors, $conf_i$ is the confidence score of sensor i obtained from SFD, λ_i is the Mean-Time-To-Failure (MTTF) of

sensor i, and w_i is the weight assigned to sensor i in the current state.

Consequently, by knowing $R_{failure}$, the system's initial probability model, P_{init} can be dynamically updated whenever sensor faults are detected. The $P_{runtime}$ is a continuously updated probability transition matrix according to the actual running status of the system. The matrix is defined as follows:

$$P_{runtime} = \begin{bmatrix} 0 & 1-R_{failure} & 0 & 0 & R_{failure} \\ 0 & 0 & 1-R_{failure} & 0 & R_{failure} \\ 0 & 0 & 0.7\times(1-R_{failure}) & 0.3\times(1-R_{failure}) & R_{failure} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where the coefficients 0.7 and 0.3 were defined based on the domain expert's experience for this specific turn-mill machine. Hence, domain expertise is not only used for the initial model but also to guide the update of expectations. With the run-time sensor's confidence score, the system failure probability of the whole system should reflect the real system states more accurately.

4.4.3 Experiment Settings and Properties

The proposed run-time probabilistic model for the turn-mill machine was implemented based on the settings defined below:

1. The initial transition probability matrix, P_{init} was first defined by experienced operators. After the system started its operation, the probability of transition, $P_{runtime}$, was updated dynamically based on the sensor's *confidence score*.

2. The system model of this implementation was based on two modules only, which are the main spindle and cutting tool.

3. Each machine part was monitored by three types of sensors: *current, vibration* and *temperature* sensor.

4. An experienced operator sets the weights of the three sensor contributions of the module as

   (a) Current sensor: 0.3

   (b) Vibration sensor: 0.5

   (c) Temperature sensor: 0.2

5. Initial sensor *confidence score* was set as 0.99 for all sensors.

6. To simplify the complexity of the model, all sensors' MTTF were assumed to be 1,000,000 hours.

4.5 Experimental Results

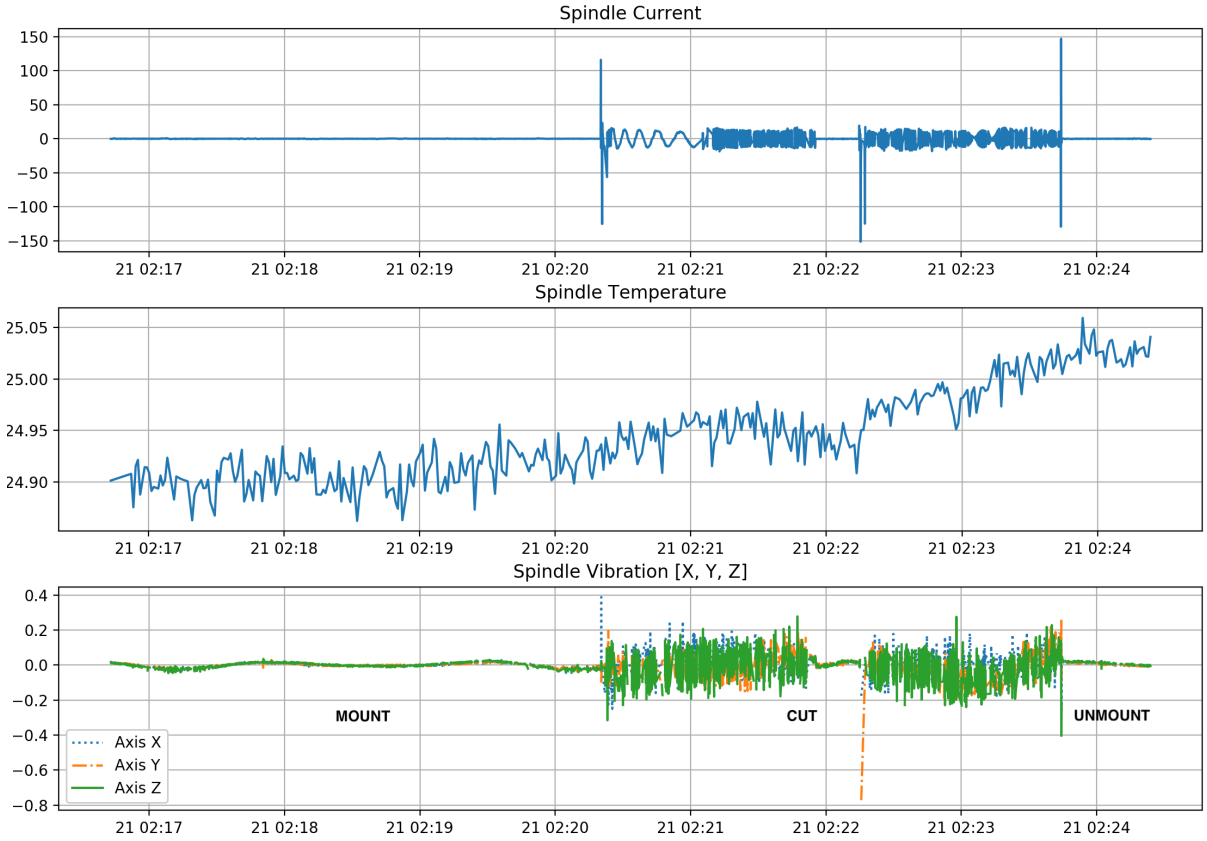

Figure 4.6: The sensor readings of the main spindle.

Three machine states were observed, *MOUNT, CUT* and *UNMOUNT*, to evaluate the proposed framework. Figure 4.6 shows the readings of *current* sensor, *temperature* sensor and *vibration* sensor from the main spindle. It is clear that the sensors exhibited different patterns at different machine states. During the transition from *MOUNT* to *CUT* and then to *UNMOUNT* state, three different patterns of the sensors' readings were observed. At the *MOUNT* stage, the temperature and vibration readings were relatively stable, while the current was almost zero. Subsequently, when the machine started the cutting operation, the temperature reading increased consistently, and the vibration amplitude was much greater. Once the cutting job was completed, the machine moved to the *UNMOUNT* state, and it was observed that the vibration re-stabilised. The temperature reading remained elevated while the current returned to zero.

In this implementation, the sensor readings were segregated according to the machine states in order to determine the sensor's *normal behaviour*. Each sensor's normal behaviour according to the machine states was analysed using the same algorithm. Table 4.1 shows the snapshots of two machine states: *MOUNT* and *CUT*, where all three sensors' readings were analysed. For each sensor and stage, the table shows the mean value and standard deviation, which highlights

the central tendency and variability of the readings at the different machine states. Additionally, the estimated reading range, represented by upper and lower bounds, provides insight into the expected readings under normal operating conditions. A drift trend metric quantifies deviations, indicating potential changes in sensor behaviour over time. A value of 0.0000 signifies no drift was detected in the respective stage, which is particularly evident for the current sensor in both the MOUNT and CUT stages. This suggests that these sensors remained stable without any noticeable long-term deviation in their readings during these periods. Notably, the vibration readings exhibit increased mean values and variability during the CUT stage, likely reflecting intensified mechanical activity, with significant drift trends along the x and z axes of 0.8095 and 0.8263 during the MOUNT stage, which could suggest wear or misalignment. Similarly, current readings during the CUT stage display increased variability slightly from 0.1561 to 8.3310, while the temperature remains relatively stable but shows a minor drift trend value of 0.3184 during this stage. These findings provide a quantitative basis for detecting anomalies and calculating the sensor confidence scores.

Table 4.1: Normal Behaviour (Sensor Profile) of current, temperature and vibration sensor

Sensor	Stage	Statistical Characteristics		Estimated Reading Range		Drift Trend
		Mean	Std Dev.	Upper	Lower	
Current	MOUNT	0.0781	0.1561	0.2296	-0.1894	0.0000
	CUT	-0.1910	8.3310	15.3485	-15.6400	0.0000
Temperature	MOUNT	24.8901	0.0158	24.9203	24.8618	0.0019
	CUT	24.9440	0.0440	24.9303	24.8694	0.3184
Vibration(x)	MOUNT	0.0033	0.0077	0.0202	0.0103	0.8095
	CUT	0.0064	0.0452	0.0771	-0.0821	0.0000
Vibration(y)	MOUNT	0.0034	0.0071	0.0167	0.0068	0.5780
	CUT	-0.0051	0.0466	0.0645	-0.0715	0.01953
Vibration(z)	MOUNT	0.0064	0.0099	0.0222	0.0090	0.8263
	CUT	-0.0065	0.0568	0.0971	-0.1010	0.0008

In SFD, there is a *Rules Engine* that is responsible for detecting sensor faults. Having defined the sensor's normal behaviour in Table 4.1, the following rules were configured in line with the domain of the case study to evaluate the concept. The exact parameter values are of less importance and, hence, were chosen conservatively. It was expected that the performance could be improved by fine-tuning the values. To illustrate the methodology, the following example rules suffice:

1. If there is zero variation (standard deviation) in the sensor readings for more than 30 seconds, the sensor is deemed as having *Stuck At* fault.

2. According to the experiences, if more than 50% of the readings are missing, the sensor is deemed as having an Intermittent fault. Otherwise, the percentage ratio of received readings and the total number of expected sensor readings is returned.

3. Compute the deviation from the run-time sensor readings to the expected reading range, which is derived from the sensor's normal behaviour.

4. If the drift trend is greater than 0.5, the sensor is deemed as having a *Drift* fault.

Table 4.2: Sensor confidence score of CUT stage

Time	Current	Temperature	Vibration
2020-02-21 10:20:43	0.9543	0.7306	0.7113
2020-02-21 10:21:13	0.9474	0.7424	0.8530
2020-02-21 10:21:43	0.9154	0.6834	0.8540
2020-02-21 10:22:13	0.9547	0.7387	0.8483
2020-02-21 10:22:43	0.9464	0.7180	0.8677
2020-02-21 10:23:13	0.9217	0.6077	0.8747
2020-02-21 10:23:43	0.9381	0.5849	0.7699

Table 4.2 shows a snapshot of a sensor confidence score during the CUT stage. Based on the confidence score, it is rather easy to identify the working condition of each sensor. Furthermore, with the proposed run-time probabilistic model checker, the resulting confidence scores were used to update the transition matrix of the model. The deviation of the sensor's working condition was reflected in the machine's overall working status.

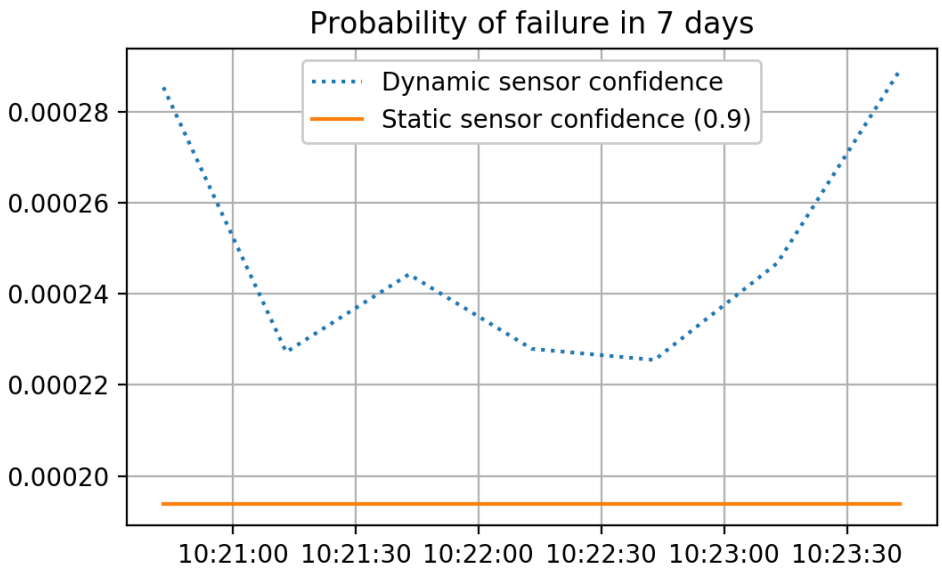

Figure 4.7: Comparison of system failure probability.

As shown in Figure 4.7, a traditional static model is assumed to have a constant confidence score of 0.9 for all sensors. This translated to a seven-day machine failure probability of 1.94E-4, which is constant at all times.

With a run-time model based on the dataset collected, sensor uncertainties were detected in the system, which resulted in the degradation of the sensor's confidence score. As shown in Figure 4.7, this had an effect on the overall system's failure probability, in that depending on the weight and the corresponding sensor's confidence score, the failure probability of the machine changed dynamically. The result shows a strong correlation between the system's failure probability and the sensor's confidence score is effectively established. Compared to the traditional static model with a constant failure rate of 1.94E-4, the proposed run-time model checker demonstrates the worst failure rate, which is 2.89E-4. It is about 1.5 times higher than the traditional static model. The proposed run-time probabilistic model checker is perceived as a better reflection of the turn-mill machine's behaviour as compared to a static model. This improvement helps the operator to manage the system more efficiently to avoid unexpected failure. For instance, identify the impact of a sensor failure to maintain or calibrate sensors in advance.

4.6 Summary

In this chapter, a run-time model checking approach is proposed that integrates a data-driven sensor failure detection and a system probabilistic model to form a run-time probabilistic model. The evaluation results highlight the importance of quantifying sensor trustworthiness in the domain of Industry 4.0, especially because all consequential decisions will be driven by automatically collected data. Moreover, the proposed methodology explicitly models the SUT with sensor uncertainties and verifies the system behaviour effectively at run time.

Chapter 5

Model Checking with Improved Quantification Method

In Chapter 4, the proposed run-time probabilistic model-checking approach is evaluated using an industrial CNC turn-mill machine as an experiment to monitor and verify the machine's behaviour during the operation period. The experiment result shows a strong correlation between sensor trustworthiness and the system failure probability. However, in real-world deployment, sensor networks are more commonly used [123–125], which compose correlated sensors in order to have comprehensive readings compared to a single sensor to improve process quality. For instance, a lift is equipped with a sensor network with a barometric air pressure sensor and an accelerometer sensor to monitor the lift's movement and stability. The barometric pressure sensor is used to measure the vertical movement of the lift, while the accelerometer measures the vibration of the lift car to determine the state of the lift system, whether it is in a moving or idle state. In this scenario, unreliable readings from the accelerometer sensor affect the entire sensor network's trustworthiness, regardless of the pressure sensor's accuracy. Thus, it further affects the lift system's behaviour. Therefore, to reflect the actual impact, the probabilistic run-time model has been extended to consider the trustworthiness of the sensor network.

5.1 Improved Sensor-Network Trustworthiness Quantification

Figure 5.1 shows an enhanced run-time probabilistic verification framework with an intermediate layer representing sensor network trustworthiness between SUT and the system model. This intermediate layer consolidates related sensors' confidence scores and calculates the corresponding sensor network confidence at run-time. Similar to Chapter 4, a base model is first defined according to the system specification, and the SFD (c.f. Section 4.1) module of the proposed framework starts to learn each sensor's behaviour and calculates confidence scores for each sensor during operations. Instead of feeding the sensor's confidence score to the system model directly, an intermediate layer is introduced to aggregate the confidence scores according

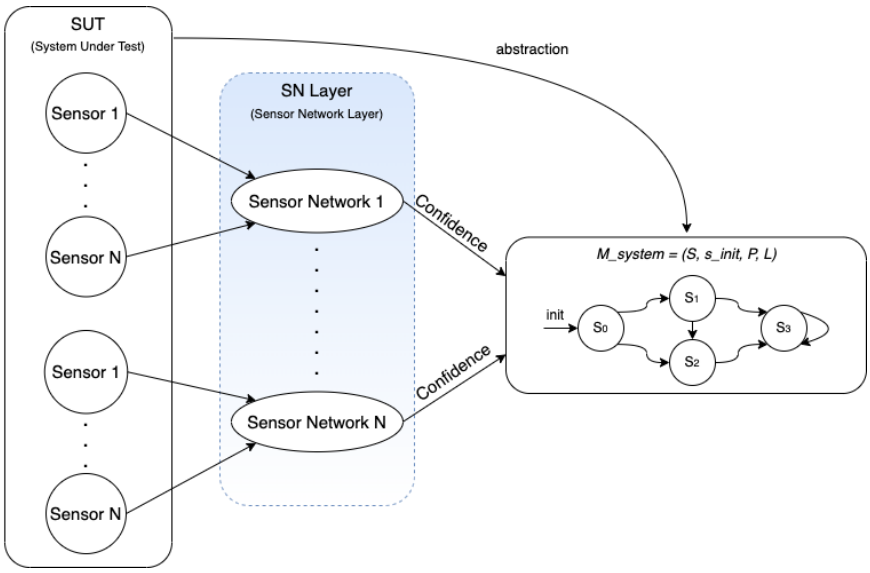

Figure 5.1: Run-time probabilistic model checker for sensor network-based system.

to the sensor network structure and the relationship of the sensors within the network.

$$Conf_{network} = \sum_{i=0}^{n} Conf_{sensor\_i} \times coef_i \tag{5.1}$$

where the confidence score of a sensor network $Conf_{network}$ depends on the conformance of the sensors' confidence score and relationships within the network, $Conf_{sensor\_i}$ is the confidence of each sensor i, $coef_i$ is the coefficient variable of sensor i in the network, e.g., if a linear correlation of the sensors in the network, the $coef_i$ will be a constant number represents the weight of the sensor in this network.

With this definition, if any deviation from the sensor's *normal behaviour* is detected, the sensor's *confidence score* will be lowered at run-time. Consequently, this sensor network's *confidence score* will be affected simultaneously according to the sensor's relationship to the network. For instance, if a primary sensor fault is detected, regardless of whether other related sensors are working normally or not, the sensor network's *confidence score* will be degraded accordingly. Subsequently, the transition probability matrix of the base model is updated based on the sensor network's confidence scores to reflect the impact of this sensor's failure. The *confidence score* of the sensor network leads to the continual updating of the transition matrix of the base probabilistic model to reflect the system behaviour at run time. Essentially, the model continually evolves over time through this process, considering the trustworthiness of run-time sensor networks to derive the appropriate probability of state transitions. It allows the verification of the system's behaviour at run-time and, at the same time, assesses the reliability of the system.

The enhanced approach is evaluated by an experiment using a passenger lift with two sensor networks and the probabilistic model checker PRISM.

5.2 Verification of Passenger Lift

Modern lifts are equipped with sensors to capture and process real-time data to monitor load conditions, detect abnormal behaviour and estimate when maintenance should be performed. An operational passenger lift system is used as an experiment to demonstrate how the sensor network's *confidence score* is computed and influences the probability model during the operation period.

The lift is modelled using the proposed run-time probabilistic model-checking framework to evaluate the proposed runtime verification approach. Internal institutional lift experts provided a list of lift parameters to be monitored, with the lift motion status and door states having the highest priorities. Hence, two sensor networks for a passenger lift were set up. One monitors the door's state, while the other is responsible for the car's movement. The door module comprises two sensors, namely *accelerometer* sensor and *magnetometer* sensor. The car module includes *accelerometer* sensor and *barometric air pressure* sensor. Figure 5.2 shows the sensor network structure of the lift model.

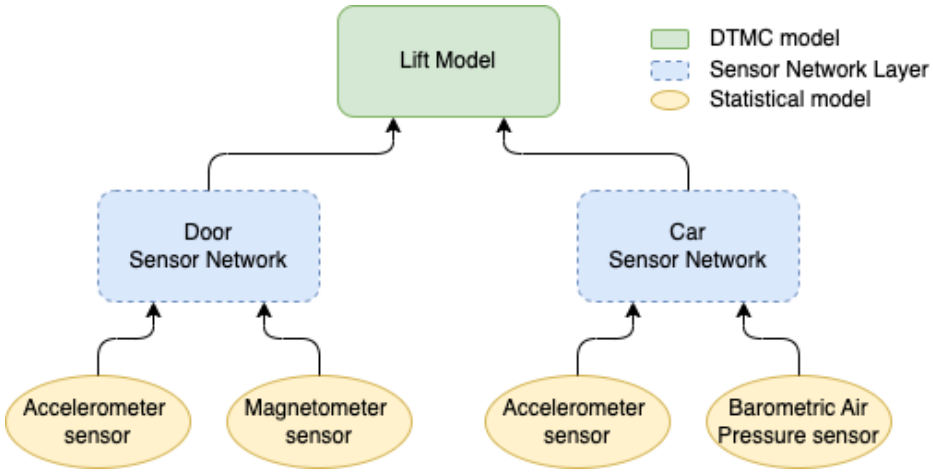

Figure 5.2: The sensor networks of a passenger lift.

These sensors were connected to a gateway located in the lift cabin. The extracted sensor readings were streamed to the back-end data processing pipeline via *Advanced Message Queuing Protocol (AMQP)*.

The SFD module was re-used from the experiment of the CNC turn-mill machine, but it was extended with an aggregation function to calculate the sensor network's confidence score. Historical data and configurations were provided through a web-based interface. SFD inspects historical readings to extract the sensor's *normal behaviour* on a monthly basis. During run-time, the sensor readings were generated by the lift and published to a cloud-based AMQP service. With this, all processing modules subscribed to the service to obtain the data to be processed in five-minute intervals. SFD received the run-time readings to compute a confidence score for

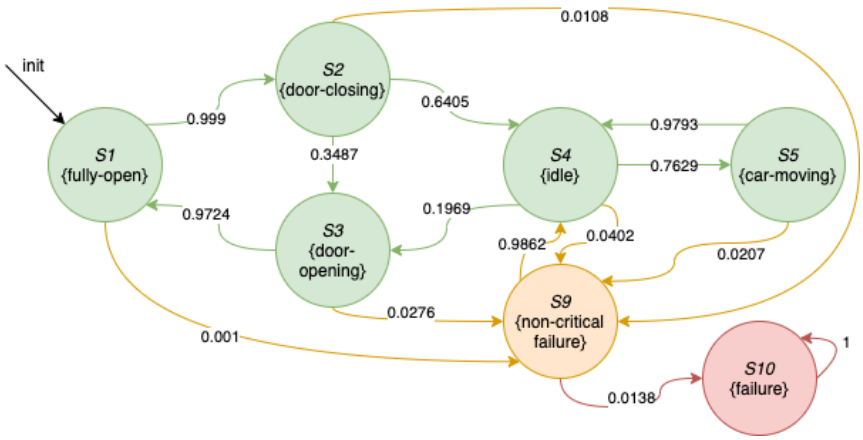

Figure 5.3: The probabilistic model of a passenger lift.

each sensor against its *normal behaviour*. Subsequently, the sensor network's confidence score was aggregated according to the network structure and relationships. Afterwards, the sensor network's confidence scores were fed to SMV to update the transition probability matrix.

Utilising an operational lift, a probabilistic system model was derived following the methodology outlined in [126], employing the Evidence-Driven State-Merging (EDSM) algorithm [127] using the lift's historical operation data. Subsequently, domain experts assigned labels to each derived state based on their domain knowledge and operational experience. Figure 5.3 shows the lift model with the derived transition matrix, which includes five working states and two error states:

1. *fully-open* — The doors of the passenger lift are fully opened. As advised by the operators, this is the initial state of the passenger lift. The sensors' readings are most stable at this stage.

2. *door-closing* — The doors start to close until they are fully closed. At this stage, any interruption to the door movement will result in the doors turning open again. The lift's state is then moved to *door-opening*. During this stage, the door's sensor network readings reflect the movement behaviour. However, the car sensor network's readings remain stable.

3. *door-opening* — The doors start to open until they are fully opened. During this stage, the door's sensor network readings reflect the movement behaviour, but the car sensor network's readings remain stable.

4. *idle* — The doors are fully closed, and the lift cabin car is stationary. During this stage, both door and car sensor networks' readings are stable.

5. *car-moving* — The doors are fully closed, and the lift cabin car is moving. During this stage, the door sensor network's readings are stable, while the car sensor networks' readings demonstrate the moving behaviour.

6. *non-critical failure* — The lift is working in an unexpected condition. However, it can be recovered automatically.

7. *failure* — This is a state that indicates the lift is working with unexpected behaviour that may cause subsequent hazard or injury.

With this, the probabilistic lift base model is defined as the following:

$$Lift_{base} = (S, s_{init}, P_{init}, L) \tag{5.2}$$

where $Lift_{base}$ is the probabilistic model of the passenger lift, S is the set of lift states, $s_{init} \in S$ is the initial state, P_{init} is the initial transition probability matrix that derived from historical data:

$$P_{init} = \begin{bmatrix} 0 & 0.999 & 0 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.3487 & 0.6405 & 0 & 0.0108 & 0 \\ 0.9724 & 0 & 0 & 0 & 0 & 0.0276 & 0 \\ 0 & 0 & 0.1969 & 0 & 0.7629 & 0.0402 & 0 \\ 0 & 0 & 0 & 0.9793 & 0 & 0.0207 & 0 \\ 0 & 0 & 0 & 0.9862 & 0 & 0 & 0.0138 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and $L : S \rightarrow 2^{AP}$ are function-labelling states with atomic propositions:

$$S = \{fully\text{-}open, door\text{-}closing, door\text{-}opening, idle, car\text{-}moving, non\text{-}critical failure, failure\}$$

The probability of system failure is expressed by the following PCTL formula:

$$P_{failure =?} \left[F^{\leq 24 \times 30} \left(S_{10} \right) \right] \tag{5.3}$$

where $P_{failure}$ is the probability of lift failure in thirty days. The failure state S_{10} is defined in the lift model in Figure 5.3.

Each lift state is monitored by two sets of sensors as presented in Figure 5.2, and one set is independent of the other. With this, a sensor network's confidence score is calculated as a whole to represent the set of sensors. And the individual sensor's confidence score can be computed according to Equation 4.1. In this lift context, the confidence scores of two sensor networks are calculated, i.e., the *door network* and *car network* as follows:

$$Conf_{door} = \{Conf_m \times w_m\} + \{Conf_a \times w_a\} \tag{5.4}$$

$$Conf_{car} = \{Conf_b \times w_b\} + \{Conf_a \times w_a\} \tag{5.5}$$

where the $Conf_m$, $Conf_a$ and $Conf_b$ are the confidence scores of the magnetometer, accelerome-

ter and barometric sensor, respectively. In the networks, all the sensors have a linear relationship to the network. And, w_m, w_a and w_b are the weights assigned to the sensors in the current network. Subsequently, the probability of each transition can be updated according to the run-time sensor network confidence score based on the following:

$$P_{runtime} = \begin{cases} P_{init} \times Conf_{door} & state \in \{S1, S2, S3\} \\ P_{init} \times \{Conf_{door}, Conf_{car}\} & state \in \{S4\} \\ P_{init} \times Conf_{car} & state \in \{S5\} \end{cases} \tag{5.6}$$

where the run-time probability of transition between states depends on the confidence scores of two sensor networks, $Conf_{door}$ and $Conf_{car}$ are sensor networks' confidence scores obtained from Equation 5.4 and Equation 5.5, P_{init} is the initial probability matrix.

Consequently, by knowing $P_{runtime}$, the system's initial lift probabilistic model, P_{init} is dynamically updated whenever sensor trustworthiness has changed. The probability transition matrix $P_{runtime}$ is continually update, the definition is as follows:

$$P_{runtime} = \begin{bmatrix} 0 & 0.999 \times C_d & 0 & 0 & 0 & 1 - 0.999 \times C_d & 0 \\ 0 & 0 & 0.3487 \times C_d & 0.6405 \times C_d & 0 & 1 - 0.3487 \times C_d - 0.6405 \times C_d & 0 \\ 0.9724 \times C_d & 0 & 0 & 0 & 0 & 1 - 0.9724 \times C_d & 0 \\ 0 & 0 & 0.1969 \times C_d & 0 & 0.7629 \times C_c & 1 - 0.1969 \times C_d - 0.7629 \times C_c & 0 \\ 0 & 0 & 0 & 0.9793 \times C_c & 0 & 1 - 0.9793 \times C_c & 0 \\ 0 & 0 & 0 & 0.9862 & 0 & 0 & 0.0138 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where C_d and C_c represents door ($Conf_{door}$) and car ($Conf_{car}$) sensor network's confidence score respectively. The confidence coefficients are defined based on a domain expert's experience for this specific lift model. Hence, domain expertise is not only used for the initial model but also guides the update of expectations. With the run-time sensor confidence score, the system failure probability of the whole system should reflect the real system states more accurately.

5.2.1 Experiment settings and configurations

The enhanced run-time probabilistic model for the passenger lift was implemented based on the settings configured as below:

1. The initial transition probability matrix, P_{init}, was first derived using Evidence-Driven State-Merging (EDSM) algorithm. Subsequently, the experienced operators fine-tuned this matrix according to the actual running status.

2. An experienced lift engineer sets the weights of all sensors in the network. The weights of sensors in the door sensor network:

   (a) Magnetometer sensor: 0.6

   (b) Accelerometer sensor: 0.4

The weights of sensors for the car sensor network:

(a) Barometric air pressure sensor: 0.5

(b) Accelerometer sensor: 0.5

3. The initial sensor's *confidence score* was set as 0.99 for all individual sensors.

The rules were configured in consonance with the domain of lift management to validate the concept. The actual parameter values were related to the individual lift's characteristics and, hence, were chosen conservatively. They were generally fine-tuned during the deployment stage to improve performance. The following rules were sufficient to evaluate the proposed approach:

1. If the lift status is not *idle*, but there is no variation (standard deviation) in the sensor readings for more than thirty seconds, the sensor is considered as having *Stuck At* fault.

2. According to the experiences, in case more than 50% of the readings are missing in the window of five minutes, the sensor is deemed as having an *Intermittent* fault. Otherwise, the normalised ratio of the received readings and the total number of expected sensor readings are returned.

3. Compute the distance between the actual sensor reading pattern and the sensor's normal behaviour model. This distance is used as the factor to detect *out of range* faults.

4. If the drift trend value is greater than 0.5, which should be close to 0, the sensor is deemed as having a *Drift* fault.

5.3 Experimental Results

Five lift operation states were observed, *fully-open, door-closing, door-opening, idle* and *car-moving* and used to evaluate our implementation.

In this implementation, the sensors were grouped into two sensor networks: the door network and the car network. Each sensor network's normal behaviour was calculated based on the individual sensors according to equation 4.1, 5.4, and 5.5. For the confidence score of sensors and sensor networks in each lift state, the same algorithm (c.f. Section 4.4.1) was used to analyse the readings. Appendix B shows the snapshots of five lift states, where all sensors' readings were analysed.

Figure 5.4 presents a snapshot of the confidence scores of two sensor networks over a typical working day. Both networks exhibit a similar trend, maintaining greater stability during the early morning hours compared to standard working hours (approximately 08:00 to 16:00). As ride activity increased, the resulting vibrations were detected by the sensor networks, thus leading to fluctuations in the sensor confidence scores. However, these variations did not result in

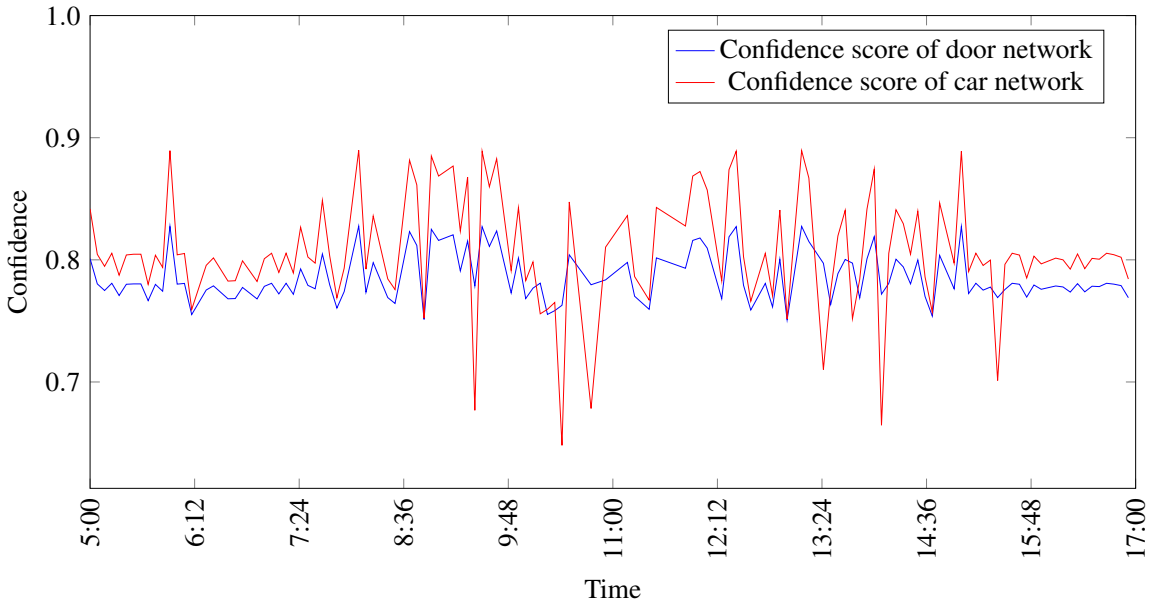

Figure 5.4: Sensor Network confidence score of Lift-Door and Lift-Car.

permanent sensor or network degradation, as the confidence scores stabilised during off-working hours.

While the overall patterns for the car sensor network and the door sensor network were similar, the car sensor network exhibited greater fluctuations. This suggests that operators should prioritise inspections of the lift's car sensor network and its associated components, such as the electric motor, pulleys, and metal cables, particularly if the confidence score of the car sensor network continues to deteriorate.

It is relatively straightforward to identify the working condition of each sensor according to the confidence score. However, it is quite challenging to determine the impact of the lift's overall operating situation in line with the sensor's working condition. With the proposed run-time probabilistic model checking approach, the resulting confidence scores are employed to update the transition matrix of the model at run-time.

Subsequently, both sensor networks' confidence scores are fed into run-time probabilistic model checking to demonstrate the reflection of the passenger lift's behaviour compared to a static model. Figure. 5.5 shows a static model that assumes all sensor networks retain a steady confidence score of 0.9. The resulting model provided a thirty-day failure probability of 0.0432, which is constant at all times.

In the actual situation, sensor faults are detected based on the collected dataset, leading to a degradation in the sensor confidence score, as illustrated in Figure 5.4. This degradation directly impacts the overall system's failure probability. Figure 5.5 presents the variation in the failure probability of the passenger lift when the model is updated at run-time. A strong correlation is observed between the system's failure probability and the confidence score of the sensor network.

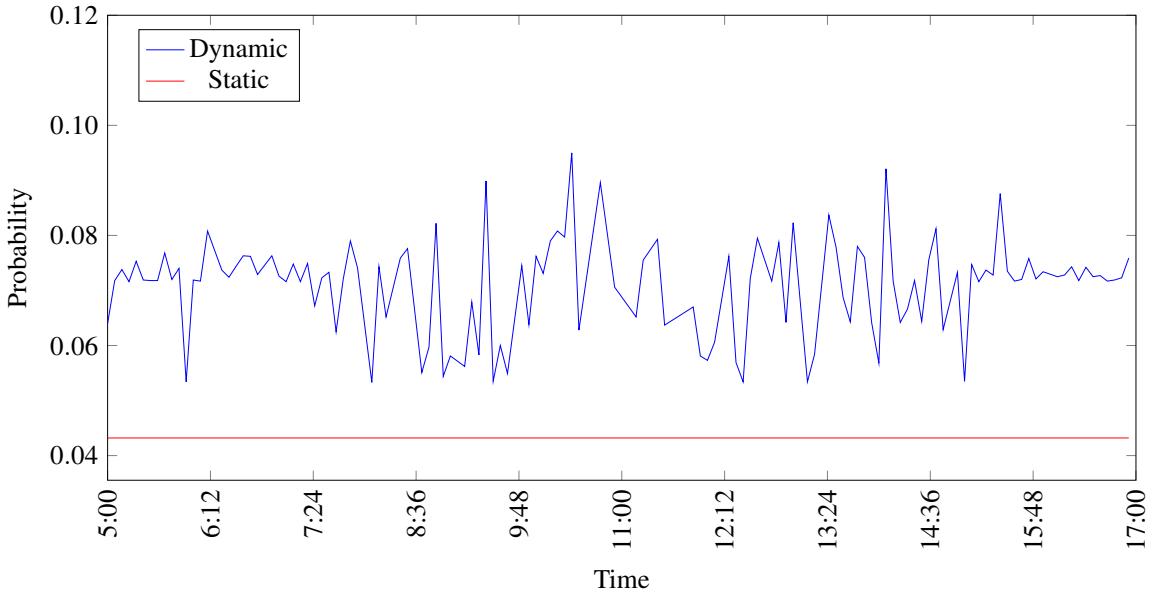

Figure 5.5: Comparison of system failure probability.

5.4 Validation and Insights

In this experiment, validating the system model against the operational passenger lift presents significant challenges when failure injection is impractical due to safety constraints. In such cases, validation relies on indirect yet rigorous methodologies. One effective approach is run-time monitoring, where the lift's live operational status is compared with the model's predicted failure to detect discrepancies. Additionally, operational data is systematically recorded to facilitate failure analysis, allowing past fault scenarios to be replayed on the model for assessing its ability to accurately capture and represent failure states. By integrating both run-time monitoring and historical data analysis, the validation framework enables continuous evaluation of the proposed approach's performance under real-world conditions. Furthermore, the collected operational data can be leveraged to optimise the computation algorithm for confidence scoring and refine the transition matrix of the system model periodically.

The proposed approach has established a correlation between sensor network confidence and failure reachability. The results demonstrate that a degradation in sensor network confidence will lead to an increased probability of failure. Although it was not possible to inject faults into the operational passenger lift system, the SFD module (c.f. Section 4.4.1) ensures accurate data aggregation from the sensor network when actual faults occur. By computing a confidence score that reflects the operational conditions of the sensor network, this will have an effect on the model's probability of reaching the failure state. Typically, maintenance teams aim to replace components or sensors within the sensor network before they fail. Thus, the probability of reaching a failure state can serve as a guideline for system maintenance, ensuring smooth operation rather than relying solely on a fixed maintenance schedule. The experimental results

were reviewed and discussed with the domain experts and experienced operators, who validated the findings and concurred with the outcomes.

This experiment has also provided valuable insights. A detailed analysis and discussion of the experimental results are presented as follows:

1. The system failure probability is in inverse proportion to the sensor's *confidence score*. When the sensor's confidence score is low, this results in a high probability of failure. In the case of passenger lift, a linear function is applied to the sensor-network confidence score to the transition probability matrix. The choice of function depends very much on the logical relationship between the sensors and the system, and this can be configured and customised accordingly. Extensive discussions are held with the passenger lift domain experts, and it is appropriate to define a linear function for this use case. Nonetheless, machine learning might be another great option if the necessary dataset is available.

2. As the sensor is dynamic in nature, a transient fault in the sensor leading to the system failure will recover automatically. Thus, the overall system failure probability is observed to fluctuate over time and does not show a trend towards high failure probability. For instance, when unexpected interference happened, this caused abnormal fluctuation in the sensor readings. Whenever this interference has stopped, the environment is back to normal, which means that the sensor reading is back to normal behaviour as well. Naturally, the overall system's failure probability will be reduced accordingly. In fact, some recoverable failures are captured, e.g., an obstacle blocking the cabin doors, which had led to a door closing failure. Once the obstacle had been removed, the lift recovered automatically. A general state named *non-critical failure* is used as in Figure. 5.1 to capture these transient faults, with the assumption that there is a high probability of recovery. Even so, the operator should pay more attention to such non-critical failures in case the lift is misused, or the ageing parts need to be replaced. However, if abnormal fluctuation is very frequent, sensor tear and wear issues might be the cause. The operator should reserve the spare parts or re-calibrate the sensor accordingly.

3. Throughout the 2-month monitoring of the passenger lift, there is no persistent sensor failure observed, and no actual lift failure has occurred. Hence, the probability of the system failure is rather low. This could be attributed to the frequent maintenance and the replacement of sensors to ensure the reliability of the lift. With the proposed verification framework, the maintenance schedule can be optimised by examining the system failure probability such that when a threshold is reached, it triggers the maintenance process to take place.

The experiment result shows a strong correlation between the sensor network's trustworthiness and the probability of system failure. Considering the dynamic nature of the sensor net-

work, this framework provides a novel approach for monitoring and maintaining sensor network-based systems, especially for safety-critical systems.

5.5 Summary

An enhanced run-time verification framework is constructed for sensor network-based systems using the passenger lift as an evaluation use case. This framework can be used to predict potential system failure probability dynamically, model the sensor network's behaviour, and quantify the trustworthiness of a sensor network at run-time. The proposed approach is demonstrated by combining a data-driven sensor failure detection module for quantifying the sensor trustworthiness and a probabilistic system abstraction from a run-time probabilistic model. A base probabilistic model can be updated at run time with the sensor network's confidence scores that explicitly reflect sensor uncertainty during the operation stage. The methodology forms a unified run-time model that presents more accurate prediction results of impending system failures, even while the system is running, explicitly modelling sensor uncertainty and updates expectations on sensor readings with the system under test. The evaluation results highlight the efficiency of explicitly modelling sensor trustworthiness, especially because all consequential decisions in the domain of sensor network-based systems will be driven by automatically collected sensor data.

Chapter 6

Compositional Modelling and Run-time Verification

In Chapter 4 and Chapter 5, the run-time model verification framework is introduced and applied to use cases involving multiple sensors and sensor networks within a single system. Specifically, these include a turn-mill machine and a passenger lift. In a real-world Industry 4.0 environment, manufacturing processes are significantly more complex, typically comprising multiple sensor network-based systems or CPSs.

Sensor network-based systems are primarily employed for sensing and communication tasks, emphasising on monitoring and data collection. In contrast, CPSs integrate computation, networking, and physical processes, facilitating bidirectional interactions between the physical and cyber domains. CPSs often incorporate sensors, actuators, control mechanisms, and computational models to enable real-time decision-making. These systems are typically application-specific and work in coordination to complete production processes. For example, in a manufacturing context, in Industry 4.0, AGVs transport workpieces between different operational areas, while collaborative robots (cobots) perform tasks on these workpieces to complete production activities. Once tasks are finished, AGVs then transfer the completed workpieces to storage facilities. This process demonstrates the interplay of multiple CPSs, where the quality of the final output relies on both the accuracy of sensor inputs and the performance of each CPS. Additionally, CPSs are designed for distinct purposes, including painting cobots, welding cobots, and logistic AGVs. Completing an entire production process generally requires the integration and coordination of multiple CPSs.

To achieve greater flexibility, an Industry 4.0 application is dynamically configured based on run-time requirements and overseen by a higher-level management system. Therefore, the proposed run-time model verification framework is crucial for supporting use cases with a compositional structure in Industry 4.0 applications, which frequently involve both sensor network-based systems and CPSs.

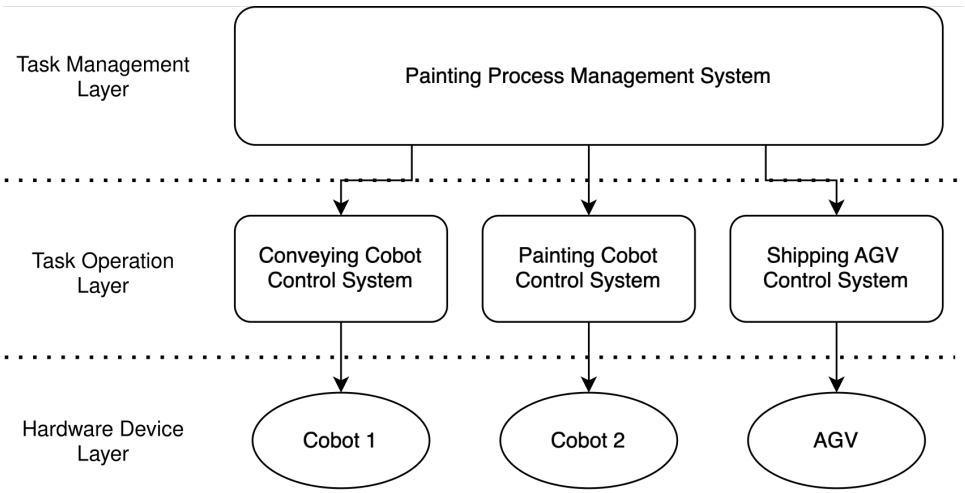

Figure 6.1: Structure of automated painting process.

6.1 Compositional Industry System

This chapter employs an automated industrial painting process to demonstrate the application of the proposed approach to a real-world industrial system. In contemporary industrial painting operations, the integration of various automation systems, including AGVs and cobots, facilitates efficient and synchronized workflows. Within this process, an AGV initially transports a workpiece from its storage location to the designated working area. *Cobot 1* then transfers the workpiece from the AGV platform to the painting station, where *Cobot 2* carries out the painting operation. Upon completion, *Cobot 2* returns the workpiece to the AGV, which subsequently transports it back to the warehouse.

Figure 6.1 illustrates the layered structure of an industrial painting process management system. Each layer serves distinct functions to ensure smooth operation and coordination of tasks among hardware devices, control systems, and high-level management. This layered structure allows each component to be developed, managed and verified independently while still working cohesively in the system. The primary layers in this diagram are as follows.

- *Task Management Layer* is the top layer, responsible for overseeing the entire workflow of the painting process. The *Painting Process Management System* at this layer handles task scheduling, resource allocation, and high-level decision-making. It manages the workflow from the arrival of a workpiece in the working zone to the painting operation and final move back to the warehouse. By controlling and directing tasks at a high level, this layer ensures that the operations performed by lower layers are aligned with production goals and quality standards.

- *Task Operation Layer* is positioned below the management layer, and *Task Operation Layer* houses the control systems responsible for the execution of specific tasks. It in-

cludes the *Conveying Cobot Control System*, *Painting Cobot Control System* and *Shipping AGV Control System*. Each control system in this layer acts as an intermediary between the management layer's commands and the physical devices in the hardware layer, translating high-level instructions into specific operations. This layer plays a vital role in coordinating the activities of different devices, for example instructing *Cobot 1* to pick up the workpiece from the AGV, directing *Cobot 2* to carry out the painting process, and guiding the AGV to transport the workpiece between locations. By isolating the control logic from the hardware, this layer allows flexibility in device management and enables easier updates or replacements of control systems without impacting the overall workflow.

- *Hardware Device Layer* is the foundational layer that contains the physical devices required to perform the painting operation. It includes *Cobot 1*, *Cobot 2*, and the *AGV*. These devices carry out the physical actions, including moving, handling, and painting the workpiece, based on instructions from the control systems in the *Task Operation Layer*. The AGV is responsible for shipping the workpiece from the warehouse to the painting zone and back. *Cobot 1* handles the transfer of the workpiece from the AGV to the painting area, and *Cobot 2* performs the actual painting process. The actions of these hardware devices are essential for the realisation of tasks set by the upper layers, and they must operate reliably to ensure the overall efficiency and accuracy of the process.

Although hardware verification can be handled by manufacturers to ensure the general reliability of physical devices, these verification results do not fully address the operational accuracy of the entire process. This is primarily due to the fact that the control systems that manage the interactions between AGVs and cobots are often developed and operated by different vendors. Thus, to ensure the integrity and reliability of the automated painting process, the proposed verification approach that encompasses the task control layer and the task management layer is essential.

In the subsequent discussion, the automatic painting system is presented as a use case to illustrate the proposed run-time verification process. Initially, the task management layer of the system is abstracted and modelled, referred to as the *System Model* or *Parent Model*. This system model is then further decomposed into task operation layer models, referred to as *Child Models*. This compositional modelling approach offers flexibility in verifying the system at different levels of abstraction and effectively managing the complexities associated with Industry 4.0 applications.

6.2 Task Operation Layer Model (Child Model)

The child model refers to the model of a control module in the task operation layer that is capable of completing single tasks independently. Also, it can be embedded in the manufacturing process

as a working module as the process requires, for instance, the operation systems of AGVs and cobots or a turn-mill machine if it is part of a processing pipeline.

As proposed in Chapter 3, the run-time probabilistic model is enhanced with input states and output states as a child model (Section 3.4). In this case, the child model is an abstraction of a minimum fully functional system in the automated painting process to represent the cobot or the AGV control systems. Typically, this is defined during the design phase to verify all system behaviours that satisfy the specification. The child model is defined as below:

$$M_{cobot} = (S, s_{init}, P_{runtime}, L, S_{in}, S_{out}) \tag{6.1}$$

where M_{cobot} is the model of the cobot control system, S is the state set of cobot working states, s_{init} is the initial state and $P_{runtime}$ is the run-time probability matrix of the cobot state transition. L is the labelling function for the transition, and two interface states for composing with external systems, S_{in} and S_{out}. $S_{in} \subset S$ is the entry state of the model, while $S_{out} \subset S$ is the exit state of the model.

6.3 Task Management Layer Model (System Model)

The task management layer typically involves the management system itself and sub-modules to control working tasks. The proposed *System Model* (Section 3.4) is capable of representing this structure. The sub-modules are abstracted using child models. Multiple child models and the task management system can be composed to form a task management layer system model. This fits well with Industry 4.0 applications as the manufacturing process is formed dynamically by multiple fully functional sub-systems with a centralised management system. In this case, two cobots and one AGV with a central management system form an automated painting system. With this context, the cobot is defined as a child model, and the central management system is abstracted as a system model. This system model is defined as:

$$M_{system} = (S, s_{init}, P_{runtime}, L, M_{cobot_1} || M_{cobot_2}) \tag{6.2}$$

where M_{system} is the task management system model, S, s_{init}, $P_{runtime}$ and L are the states and transition matrix of M_{system}, while $M_{cobot_1} || M_{cobot_2}$ is the set of the cobot models (Equation 6.1) representing the Task Operation systems, cobots for task operations.

6.4 Temporal Logic Property Query

Once the system model has been defined, the properties of the model could be verified and then queried to predict the system's potential failure at run-time. PCTL is used to evaluate the model properties for this experiment. For example, the probability of eventually moving to a

Figure 6.2: The experiment setup.

failure state. For instance, the probability of system failure is expressed by the following PCTL formula:

$$P_{failure} =? [F^{\leq t} (S_{failure})] \tag{6.3}$$

where $P_{failure}$ is the system failure in the next t time. The failure state $S_{failure}$ is defined in the system model.

6.5 Experiment Design

The experiment was designed to simulate a real-world industrial painting process using two cobot arms. These cobots were programmed to autonomously synchronise their actions to complete the painting task for a single workpiece. In this experiment, one cobot was responsible for preparing the workpiece while the other applied the paint, simulating the actual manufacturing process.

Figure 6.2 illustrates the experimental setup in the lab, showing the cobots in action within a controlled environment designed to replicate the constraints and requirements of an actual industrial painting system. This setup serves as a physical system to validate the proposed run-time verification approach.

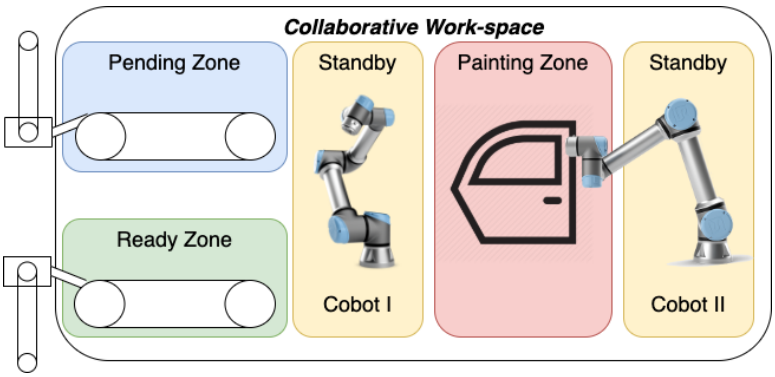

Figure 6.3: The collaborative workspace for painting process.

Figure 6.3 illustrates three working zones in the collaborative workspace. Cobot I is in charge of moving the workpiece from the *pending* zone to the *painting* zone, while Cobot II executes the painting work at the *painting* zone. When the painting work is completed, Cobot I must then move the workpiece to the *ready* zone. During the painting process, one cobot must be in the *standby* zone with a standby position when another cobot is working. Each cobot is equipped with a depth camera to detect the working zone and workpiece using image processing techniques.

The electrical current in Amperes (A) of the cobots is the key indicator to determine the machine's working condition. For instance, if the cobot is obstructed, the current reading will be much higher than the normal reading range. The current readings of the base joint, shoulder joint, elbow joint and wrist joint were observed in this experiment. Firstly, the cobot's *Sensor Normal Behaviour* was profiled, and then the *Sensor Confidence Score* was determined.

6.5.1 Settings and Properties

In order to focus on the main actions and to simplify the painting process model, the following assumptions were defined according to the expert:

- All cobots were working in the collaborative workspace, including *pending* zone, *ready* zone, *painting* zone and *standby* location.
- There was no shared space between *pending* zone, *ready* zone, *painting* zone and *standby* location.
- The three critical parameters, zone detection accuracy, workpiece detection accuracy and current readings of cobot's joints, were fetched through the painting system and cobot systems' Application Programming Interfaces (APIs).
- Five sensor fault types were used to compute the *Sensor Confidence Score* at run time. In addition, the following weights for each fault type derived based on the occurrence and

severity were assigned based on the advice from the expert.

  - Intermittent fault: 0.1
  - Stuck-at fault: 0.1
  - Spike fault: 0.1
  - Follow estimated reading range: 0.1
  - Sensor data pattern match: 0.6

- The servicing cycle was twenty days.
- The cobot's safe working range was defined according to ISO/TS 15066:2016. The *pressure* should be within 160 N/cm^2 of quasi-static contact and 2 N/cm^2 of transient contact. Additionally, the *force* should be within 210 N of quasi-static contact and 2 N of transient contact.

6.5.2 Task Operation model – Cobot Arm

As shown in Fig. 6.4, an operating cobot can be represented by eleven states to illustrate its working status.

S_1 : *Standby* is the entry state in which a cobot is ready for duty. In this state, the *force* and *pressure* should be in the safe working range.

S_2 : *Idle* is the state that the cobot prepares to move its arm from *standby* location to the working zone, e.g., *painting* zone, *pending* zone. In this state, the *force* and *pressure* are kept in a safe range. The cobot turns on the camera and triggers its image-processing engine to determine the position of the workpiece.

S_3 : *Zone detection* detects and confirms the target zone location. The cobot should move the gripper to a more precise position.

S_4 : *Workpiece detection* detects and calculates the precise location and the size of the workpiece. The cobot operates its gripper to grip and pick up the workpiece accordingly.

S_5 : *Execute mission* means that the cobot performs the task as instructed.

S_6 : *Controlled parking* is the state that the cobot reduces the Tool Centre Point (TCP) speed safely, stops working and moves as commanded by the controller. In this state, the *force* and *pressure* are kept in the safe range.

S_7 : *Stop* is an exit state in which the cobot is stationary and does not work for any purpose. In addition, the *force* and *pressure* are in a safe range.

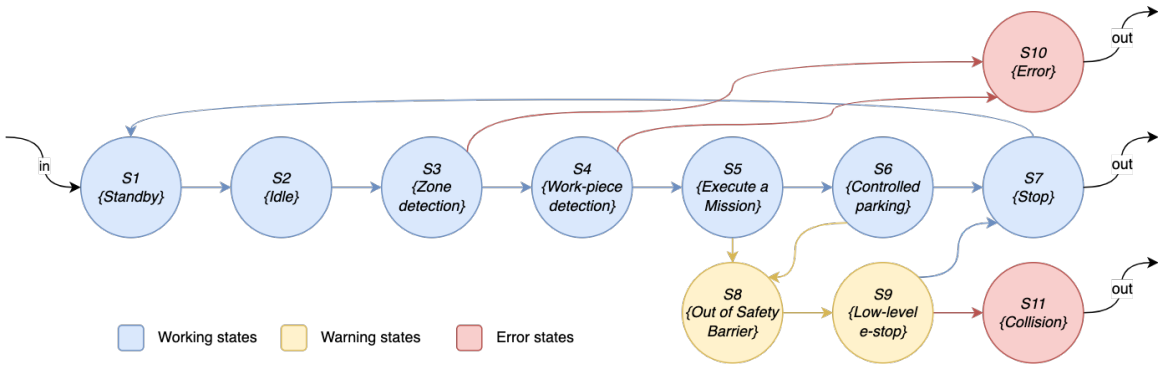

Figure 6.4: The cobot child model M_{cobot}.

S_8 : *Out of safety barrier* is the state that one or both *force* and *pressure* is out of the safe range. In this state, the cobot must stop working immediately.

S_9 : *Low-level e-stop* means the cobot triggers the emergency-stop function.

S_{10} : *Error* is a state indicating that there is a zone or workpiece detection error.

S_{11} : *Collision* is a state that cobot stopped and one or both *force* and *pressure* is still out of the safe range.

The child model (c.f. Eq. 6.1) is used to verify the cobot system behaviour individually. Three factors according to the run-time conditions, namely *zone detection accuracy* represented by $zone_{acu}$, *workpiece detection accuracy* represented by wp_{acu} and *sensor confidence score* represented by sn_{conf} are derived at run time to update the transition matrix, P_{cobot}, which reflects the run-time behaviour of the cobot during the operation stage.

$$P_{cobot} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & zone_{acu} & 0 & 0 & 0 & 0 & 0 & 1-zone_{acu} & 0 \\ 0 & 0 & 0 & 0 & wp_{acu} & 0 & 0 & 0 & 0 & 1-wp_{acu} & 0 \\ 0 & 0 & 0 & 0 & 0 & sn_{conf} & 0 & 0 & 1-sn_{conf} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & sn_{conf} & 0 & 1-sn_{conf} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & sn_{conf} & 0 & 0 & 0 & 0 & 1-sn_{conf} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.4}$$

6.5.3 Task Management Layer Model - Painting System

The system model is an abstraction of the painting process that includes five machine states and is composed of two instances of the cobot model as the child models. In particular, Cobot I is responsible for moving the workpiece between the pending zone, painting zone, and ready zone, while Cobot II paints the workpiece only in the painting zone. In this system, both cobots should not execute their tasks concurrently in order to avoid a potential collision. For example, while Cobot I is moving a workpiece to the painting zone, Cobot II should be stationary at the standby location with a safe standby pose.

Fig. 6.5 illustrates the parent model that defines the five system states and two child cobot models:

S_1 : *Idle* is the initial state of the painting process, the system checks all the modules' statuses and waits for the task to be executed.

S_2 : *Plan* is the state that the system retrieves the task details, plans the steps, and adjusts running parameters.

S_3 : *Waiting workpiece* is a state that keeps detecting the workpiece in the pending zone. If a workpiece is detected, the system should trigger the planned action.

S_4 : *Confirm status* is a state that ensures all modules are ready to move to the next step. It is assumed that there may be a random error with this state leading to the transition to S_5 *Failure* state since the painting system checks with cobot systems and process control modules at this stage. According to the empirical rule and statistical studies of industrial processes [128], it is assumed to be a quarterly failure event, which approximates to 0.9875 ($\mu \pm 2.5\sigma$).

S_5 : *Failure* means the painting system works with an unexpected behaviour that may cause subsequent hazard or injury.

C_1 : $C1_{in}$ *and* $C1_{out}$ are the input and output states of a sub-task assigned to a dedicated cobot to shift the workpiece to the working zone.

C_2 : $C2_{in}$ *and* $C2_{out}$ is the action to paint the workpiece. This task is allocated to the second cobot in the system.

The painting process is abstracted using an extended MDP. The symbol $M_{painting}$ represents the system-level model as follows:

$$M_{painting} = (S, s_{init}, P, L, M_{c1} || M_{c2})$$

where $M_{painting}$ is the compositional system model, S is a five-state set of the system, $s_{init} \in S$ is the initial state S_1, $P : S \to 2^{Act \times Dist(S)}$ is the transition probability function, where Act is a set of

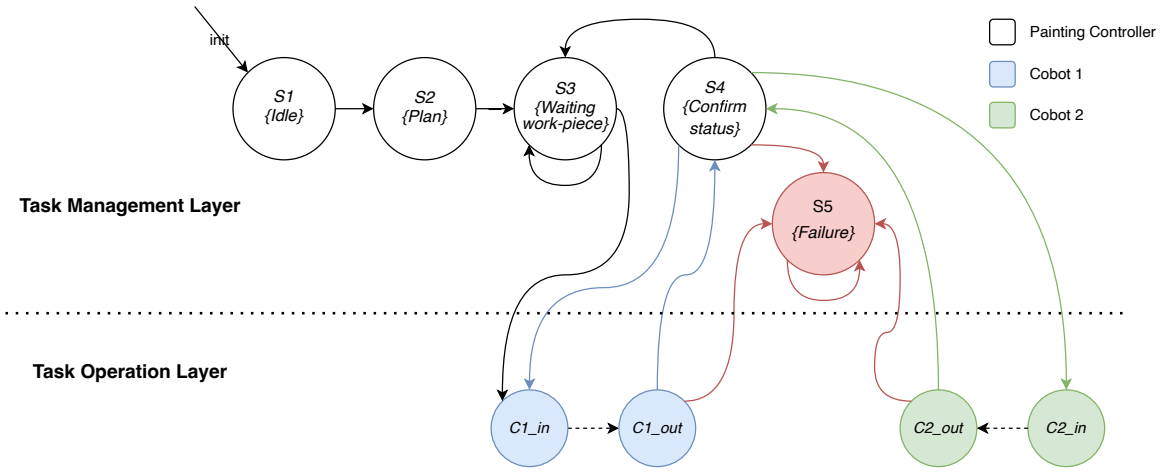

Figure 6.5: The painting system model.

actions and $Dist(S)$ is the set of discrete probability distributions over the set S. $L : S \to 2^{AP}$ is a labelling with atomic propositions. $M_{c1}||M_{c2}$ are two cobot models, and operator $||$ represents the connection using the interface states of these two child models. In this model, the cobot model is considered as a special state machine, in which the S_{in} and S_{out} of the cobot model are the interfaces to integrate with the other four standard machine states S_1, S_2, S_3, S_4 and the *Failure* state S_5.

The initial transition matrix is defined as below, where the probability of Cobot I and II completing the task are represented using $c1_{comp}$ and $c2_{comp}$, respectively. The notation "/" is used to indicate nondeterministic choices in the state transition.

$$P_{painting} = \begin{array}{c} \begin{array}{ccccccc} S_1 & S_2 & S_3 & C_1 & S_4 & C_2 & S_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/0 & 0/1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & c1_{comp} & 1-c1_{comp} \\ 0/0.9875 & 0 & 0 & 0 & 0 & 0.9875/0 & 0.0125 \\ 0 & 0 & 0 & 0 & c2_{comp} & 0 & 1-c2_{comp} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} S_1 \\ S_2 \\ S_3 \\ C_1 \\ S_4 \\ C_2 \\ S_5 \end{array} \tag{6.5}$$

Similar to the cobot child model, the initial transition matrix of the system model is defined according to the system specification and updated at run time based on the sensor confidence score.

6.5.4 Evaluation of System Failure

Experiments were conducted to verify the safety and reliability of the painting process at run time using real sensor readings and working conditions. The probability of system failure is expressed by PCTL as below,

$$P_{failure} =? \left[F^{\leq 20} \left(S_5 \right) \right]$$

where $P_{failure}$ is the probability of eventual system failure state, S_5 : , in the next 20 days, i.e., the service cycle of the cobot.

6.6 Implementation and Results

Two cobots were used to evaluate and validate the proposed compositional model verification framework. One was a UR10e that worked as the workpiece moving cobot C_1 : . The other was a Franka Emika, which was used for workpiece painting cobot C_2 : . The sensor readings and cobot states were retrieved through Modbus protocol for the UR10e and a low-level C++ interface, *libfranka*, for the Franka.

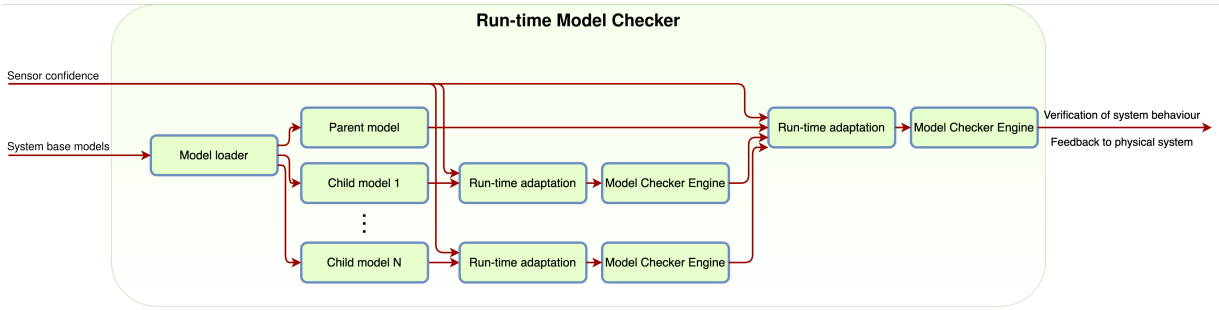

Figure 6.6: Implementation of Run-time model checker for Industry 4.0 applications.

Figure 6.6 shows the compositional run-time model checker implementation. Firstly, this implementation loaded the base model for the painting system along with all the initial child models. Subsequently, the child models were updated with quantified run-time sensors' confidence scores, which used the approach as described in Chapter 5 and derived the run-time model probabilistic transition matrix for interface states. Finally, the updated child models, along with the run-time sensor confidence scores, contributed to the formation of the higher-level painting system model for verification purposes. The child models and painting system model were evaluated using a probabilistic model checker tool, PRISM.

Six test cases were developed to evaluate the proposed approach.

TC_1 This test case adheres to the conventional approach, which assumes that sensor readings are fully reliable and 100% trusted. The results obtained from this test case serve as a

baseline, representing traditional methodologies, and are utilised for comparison with the proposed framework.

TC_2 This test case employs the proposed approach, which incorporates a dynamic *sensor confidence score* to assess the probability of failure of the painting system at run-time. It is designed to evaluate the performance of the proposed framework. By accounting for the run-time trustworthiness of sensors, the system model in this test case evolves continuously in response to the actual operating conditions of the sensors.

TC_3 This test case involves manually increasing sensor readings by 10% to simulate drift events over time, aiming to validate the effectiveness of the proposed framework. Through these manual interventions, the test case evaluates the performance of the proposed approach in detecting and addressing sensor uncertainties.

TC_4 Similar to $TC3$, this test case involves manually decreasing sensor readings by 10% to simulate drift events, with the objective of evaluating the proposed approach and assessing its performance.

TC_5 This test case involves a controlled manual intervention to disrupt the cobot's movement by temporarily obstructing its arm under safety guidance. It is designed to validate the proposed approach in a real-world physical working environment and assess its performance.

TC_6 This test case utilises a defective firmware that returns only signed integers, leading to incorrect decoding of sensor readings originally intended as unsigned integers. The firmware is integral to the sensor module, responsible for converting the sensed analogue signals into digital form and transmitting the digital readings for subsequent processing. While the defective firmware performs adequately in most scenarios, it encounters an overflow error when the variable exceeds 32,768 during the conversion process. This test case aims to validate the proposed approach on a physical machine and assess its performance under these conditions.

A single test result from each case TC_1 and TC_5 was recorded for comparison purposes. To evaluate performance under dynamic changes in sensor trustworthiness over time, nine cycles were executed for TC_2 , TC_3 , and TC_4 . TC_6 was designed to run for one hundred cycles in order to capture the failure events within the physical working environment. Each cycle required approximately three minutes to complete the painting process.

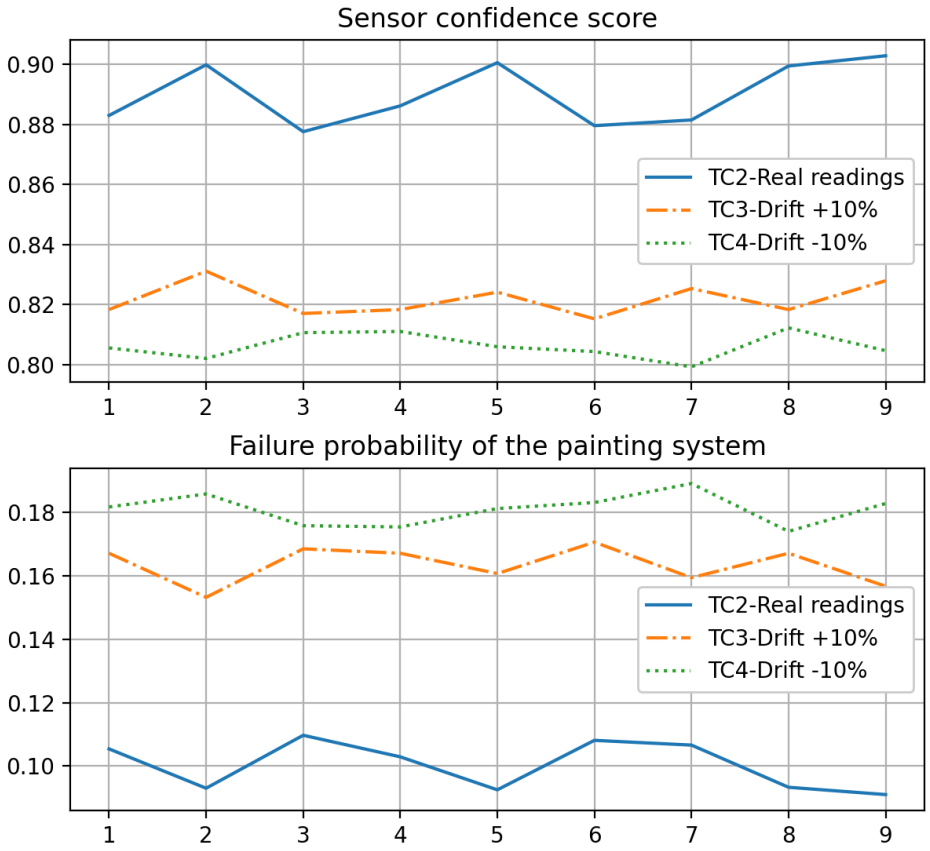

Figure 6.7: Comparison of real sensor readings and drift simulation.

Initially, a dataset of machine states and sensor readings was collected under the supervision of an experienced operator to ensure that the painting system functioned as intended. This dataset was subsequently processed to establish the *sensor normal behaviour* profile of the cobots, providing a baseline for quantifying sensor trustworthiness throughout the experiments conducted in all test cases.

As the first test case TC_1 assumed that the sensor readings are 100% trusted, a system failure probability of 0.0125 was obtained at all times, which is not realistic as it is so close to perfect operation with zero failure. However, in real-world deployment, it is a challenge to get such a perfect result. In TC_2, the run-time sensor readings were monitored and then computed the *sensor confidence score* of 0.88 – 0.90, thus resulting in a system failure probability of 0.09 – 0.11. This effectively reflects the real situation of the painting system, that it was not always working in the perfect condition. As for TC_3 and TC_4, the simulated drift events of $\pm$10% of the real data that was captured, it is observed that the *sensor confidence score* dropped about 10% to 0.80, thus causing an increase in the failure probability to 0.15 and 0.18. Fig 6.7 summarises the comparison of test results of TC_2, TC_3 and TC_4, comparing the real sensor values and the drift simulation results to evaluate their impacts on the failure probability with respect to the sensor confidence score. The horizontal axis indicates the cycle of the test cases, and the vertical

Table 6.1: Sensor confidence score and failure probability.

Test Case	Sensor Confidence Score	Failure Probability
TC1	1.000	0.0125
TC5	0.6735	0.3635
TC6	0.3814	0.7905

axis represents the sensor confidence score and the failure probability.

Test cases TC_5 and TC_6 were utilised to validate the proposed approach under the actual operating conditions of the physical machine. Test case TC_5 was executed by blocking the cobot's action under safety guidance. This intervention triggered the cobot to halt with an exception status. In this case, the resulting *sensor confidence score* dropped to 0.6735 was observed, which was about 25% lower than the normal situation. With this, the 20-day failure probability had thus increased to 0.3635. Lastly, TC_6 , the defective firmware was employed to assess the impact on the system failure probability. The defective firmware caused errors by retrieving incorrect sensor readings due to the improper decoding of unsigned integers. This experiment was prematurely terminated because the control system failed to properly handle these erroneous sensor readings. Consequently, the *sensor confidence score* dropped to its lowest value of 0.3814, leading to a significantly high failure probability of 0.7905, as shown in Table 6.1. These two test cases, TC_5 and TC_6 , are designed to check the system's property using the abstracted system model. Based on the property-checking of the system model, the behaviour of the mock-up physical system was observed and then validated against the run-time system model, thereby closing the Physical $\rightarrow$ Model $\rightarrow$ Physical validation loop. The full results are listed in Appendix C, Table C.1.

Chapter 7

Discussion

This section discusses the implications of this research in the realm of the formal verification of Industry 4.0 applications and highlights the significance of the proposed approach and the processing framework.

The primary objective of this research is to develop a formal verification framework that considers uncertainties of Industry 4.0 applications, for instance, taking into account the run-time sensor failure probability when verifying a manufacturing process. The experiments and evaluation are demonstrated in Chapter 4 and 5. Moreover, this proposed framework is also capable of handling complex scenarios of actual Industry 4.0 applications using the compositional model approach, which is introduced in Chapter 6. The following sections discuss the advantages of the proposed approach compared to existing techniques, possible use cases and deployments of the proposed framework and limitations that should be addressed by the following research work.

Figure 7.1 illustrates an overview of the model verification in the industry with current practices, the contribution of this research work and the gaps to the desired final goal.

Currently, the typical process to rigorously verify a system in the industry is in the following steps:

1. Begin by creating a formal model of the system according to the specification. This model can represent an abstraction of the system that captures the relevant aspects of the system's behaviour.

2. Specify the properties that the system to satisfy. These properties can include safety requirements, functional correctness, or other critical characteristics. The properties are typically expressed in temporal logic or formal language.

3. Choose an appropriate model-checking tool based on the nature of the model and properties. The tools of model checkers may be better suited for hardware verification, software analysis, or specific types of systems according to the verification purpose.

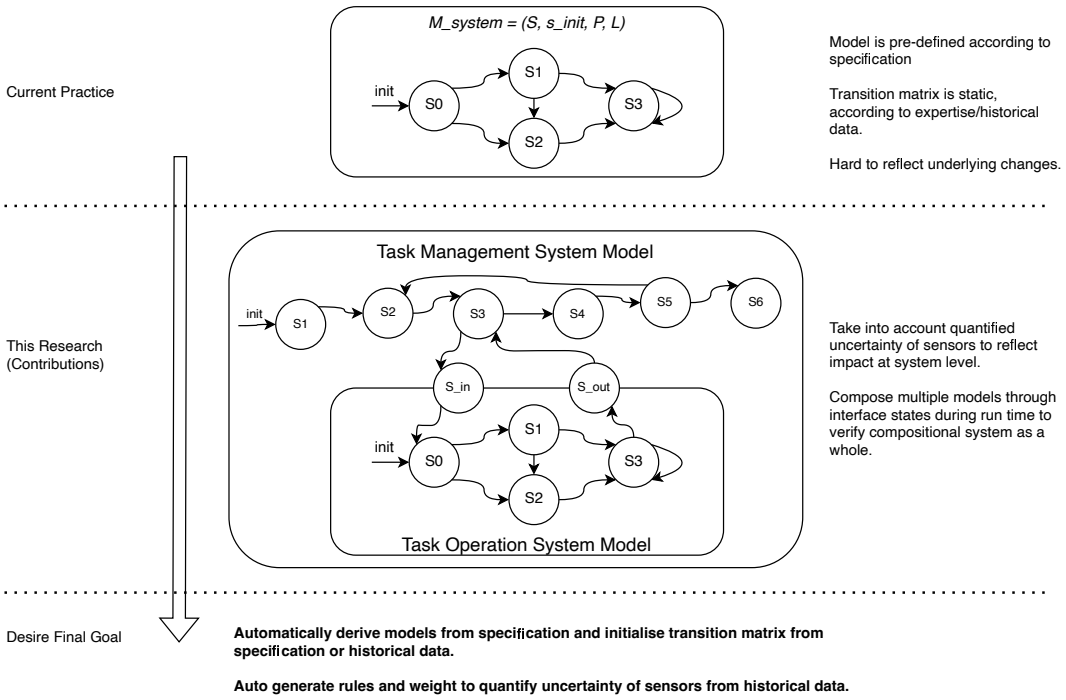

Figure 7.1: Gap analysis.

4. Run the model checking tool on the formal model and provide the specified properties. The tool exhaustively explores the state space of the model, checking if the properties hold for all possible states or scenarios. If the model checker finds a scenario where the properties are not satisfied (i.e. counterexample), the tool provides diagnostic information. This information helps identify the source of the problem, allowing the operator to understand and work out a solution to address the issue.

5. Based on the results of the initial model checking, refine the model and/or properties as needed. This may involve adding more details to the model, adjusting properties, or making other modifications to improve the accuracy of the verification.

According to the current process, most of the steps are manual and highly depend on expertise. Moreover, the existing process is relatively static compared to the requirement of Industry 4.0 applications, which requires the process to be flexible and agile. This presents an obstacle to applying model verification techniques in the industry. The proposed verification framework partially addresses this challenge, as described in the previous chapters, by taking into account uncertainty from underlying sensors and composing multiple models through interface states to assemble system models at run-time to verify as a whole. However, further research work is also required to achieve the final goal of automating model generation and evolution.

In this chapter, the advantages of the proposed framework are discussed in Section 7.1. The possible use cases are shown in Section 7.2. Section 7.3, analysis the gaps and limitation of this research work.

7.1 Advantages of Proposed Approach

The experiments in Chapter 4, Chapter 5 and Chapter 6 demonstrated the effectiveness of the proposed framework and highlighted its ability to identify the run-time uncertainties from underlying sensors. Subsequently, these quantified uncertainties are dynamically adapted to the verification mechanisms to verify the correctness during the operation periods. As a result, the proposed framework is capable of reflecting run-time uncertainty in model verification results. The impact of these uncertainties is hard to quantify using existing formal verification techniques. Compared to traditional model checking approaches, the proposed framework targets Industry 4.0 applications and comes with advantages for the industry as below:

- The proposed framework provides the capability to compose multiple system models during the operation period by extending traditional model interface states. This advantage adopts industry verification for a dynamic manufacturing process, which is common in Industry 4.0 applications. For instance, a robot-based manufacturing process involves conveyors, robot arms, AGVs and a control system. Each component can be verified individually using traditional model-checking techniques. In order to optimise the efficiency of all working components, each component serves multiple manufacturing processes according to its working status. In this scenario, the proposed framework composes the process model using component models with the interface states to verify the process during the operation period. This approach verifies the system efficiently to guarantee its safety and correctness.

- Another advantage of the proposed framework is that it takes into account uncertainties of underlying sensors during the operation period. Leveraging the data-driven approaches, each sensor confidence score is calculated and fed to the system model to verify the system's behaviour. This improves traditional formal verification approaches to reflect the sensor network-based system's behaviour more accurately, which is important for the industry to ensure safety and manage the quality of production lines.

The resulting framework is particularly promising when compared to traditional run-time verification tools. While some tools have shown promise in identifying certain types of errors, our framework goes beyond handling a broader range of industrial-level applications, including process management systems (Chapter 1.1 *Task Management Layer*) and robot operating systems (Chapter 1.1 *Task Operation Layer*). The flexibility of our approach, combining formal methods with lightweight instrumentation, strikes a balance between precision and efficiency, allowing for effective verification even at run time in operation environments. With these advantages, the proposed framework combines data- and model-driven approaches to help advanced manufacturers monitor and verify the systems' efficiency and correctness, especially in the Industry 4.0 context.

7.2 Possible use cases and deployment

Similar to the scenarios in Chapter 4, Chapter 5 and Chapter 6, the proposed verification framework can be easily applied in the industry to verify the behaviour and correctness of the automation equipment, monitor the safety and reliability in the safety-critical production process, and so forth. In the actual industry environment, the following applications can be the possible use cases.

- Run-time monitoring is a technique using a pre-defined model of the expected system and continuously comparing it to the actual behaviour of the system during run time. If any discrepancies or violations are detected, appropriate actions can be taken to handle or mitigate them. Compared to the traditional approaches, the proposed framework considers uncertainties of the system and updates the model during run time to reflect the actual situation of the system. Moreover, to deal with increasingly complex systems, the proposed framework provides compositional model verification and adaptively verifies the system.

- Model-based Testing (MBT) is useful for complex systems and commonly used in industry. Traditional MBT approaches focus on the industry software, generating test cases according to the system model and transitions [129]. The proposed framework extends the existing approaches to the sensor network-based systems and CPSs, considering sensor uncertainties during run time and providing more flexibility to handle compositional structure systems.

- Digital Twin is a virtual representation that mimics the behaviour and characteristics of real-world systems [130, 131]. Applying the proposed framework can verify the digital twins faithfully replicate real-world systems and respond appropriately to various inputs and scenarios, enhancing the reliability of decision-making processes and the value of digital twin applications in fields like manufacturing, IoT, and smart infrastructure.

Testing and verification are crucial steps in the manufacturing industry. As the leading independent Testing, Inspection and Certification (TIC) company, *TÜV SÜD*[1] is from Germany and has more than 150 years of history that provides professional testing and verification services to the manufacturing industry. Adaptive Safety and Security System (AS3) is the system that is developed by TÜV SÜD Product Service (PS) team. This system targets dynamic risk assessment of both virtual environments and real-world applications for Industry 4.0 manufacturers. This research work is also planned to be embedded into AS3 to take into account sensors' uncertainty at run-time to improve system accuracy and efficiency. Additionally, the proposed framework also provides the capability to predict the potential failure probability of the system monitored.

[1] https://www.tuvsud.com/en-sg/about-us

7.3 Limitations and Gaps

Despite the advantages and possible use cases, there are still gaps in achieving the final goal that Industry 4.0 requires. For instance, the proposed framework depends on the expertise, experience and manual work to define the system model and specify the properties that the system satisfies. By leveraging the advantage of AI techniques, such as the Large Language Model (LLM) [132–134], it is possible to be used to assist in initialising system formal models according to the specifications. Another gap is that quantifying the sensor's uncertainty also depends on expertise and experience. As further research work, systematically defining rules and weight for the quantification is an enhancement to the proposed verification framework.

7.3.1 Initialisation of Formal Models

Two approaches were used to initialise a formal model for the target systems in the experiments of this research: expertise-based and data-driven approaches.

- In Chapter 4 and Chapter 6, the initial models of the experiment were built based on expertise. It is common for domain experts to define the initial model according to the system specification and the experience of traditional model verification approaches. However, this is a challenge for the new Industry 4.0 applications that lack the domain knowledge of operators and experts. As Industry 4.0 applications continue to mature and find increased deployment across various fields, this challenge will progressively be overcome.

- The data-driven approach is another popular approach and was used for the experiment in Chapter 5. The initial model was derived from the historical data set that was collected during the passenger lift operation period. From this data set, an EDSM algorithm was employed to derive lift operational states and transition matrix. However, similarly to typical data-driven approaches, having a good data set to derive an accurate system model is a challenge. Moreover, the algorithms only summarise statistics or characteristics of the observed data. Post-processing is needed to provide more meaningful labels to each state and transition according to the domain knowledge. This identifying and labelling work must also depend on the domain experts.

- Learning-based model synthesis is also a powerful approach that can be used to initialise a formal model and parameters of state transitions. This approach helps to reduce dependency on expert knowledge, for instance, learn a model from system observations [135], [136], [137] or learn the specifications that the system meets from system observations [68], [138]. For instance, supervised learning approaches can be used to infer temporal logic formulas with labelled system outputs [139] and derive system normal behaviours. The learning-based approach is an active research field that can be used to tackle the challenge of lack of experience in the Industry 4.0 domain.

To initialise a formal model of an actual system is always a challenge that consumes time and requires deep domain knowledge, especially in a new domain, such as Industry 4.0 manufacturing applications. However, some research work has been established to target to resolve this challenge, which is also a part of this research in future work.

7.3.2 Quantification of Run-time Uncertainties

The quantification algorithm of the proposed framework depends on the assumptions that manufacturer experts provide the knowledge to define the rules and the weight of each rule manually. For example, in Chapter 4, the $weight_i$ in equation 4.1 and the settings in Section 4.4.3 are assumed to be defined by experienced operators. Having domain experts is common in the industry to configure and supervise the manufacturing processes. However, two potential risks may occur:

- The quantification result is only good when experienced operators provide suitable and accurate definitions of the weight for each sensor and each rule. In the traditional industry environment, experienced operators or domain experts optimise and fine-tune each manufacturing process by adjusting configurations and parameters continually over the operation period. However, it may be a challenge in Industry 4.0 manufacturing due to the lack of expertise to define the rules and the weights of each rule for the quantification algorithms. The data-driven approach is another method to derive the weights of each sensor and each rule from historical data sets. Similarly, the quantification result is only good if high-quality data sets are provided. However, providing a fine-grained data set in this new domain is also a challenge.

- The proposed quantification approach is good for the sensors that deliver numerical readings, but it may not be efficient for image sensors that generate binary readings and Near Infrared/Mid-Infrared (NIR/MIR) sensors that produce multivariate readings. As the readings from image and multivariate sensors are more complex than normal numerical readings, typically, pre-processing algorithms are involved before delivery. The concept of quantifying such readings is the same as the proposed quantification approach to first generalise the sensor's *normal behaviour* and quantify run-time readings against the *normal behaviour* subsequently. However, the implementation of the quantification algorithm may need further evaluation to prove its efficiency and accuracy.

The proposed formal verification approach takes into account sensors' and algorithms' uncertainties. However, it is still manual or subjective to define the rules and weight of quantification calculation. Nonetheless, further research will be carried out to address these challenges in the field.

7.4 Summary

This research extends traditional model-checking techniques to take into account the uncertainties of sensors and is capable of composing multiple system models using the proposed interface states. However, due to a lack of domain experts and experience with the latest technologies in the industry, creating the system models is still a challenge. The resulting framework has the ability to verify complex manufacturing systems to make Industry 4.0 successful and for stakeholders to reap maximum value from Industry 4.0.

Chapter 8

Conclusion and Future Work

In this thesis, a novel run-time verification framework is proposed that takes into account uncertainties from sensor networks and decision-making algorithms. This framework is evaluated with three industrial scenarios. The main objective of this research is to verify Industry 4.0 applications rigorously.

8.1 Conclusion

A comprehensive analysis of the research question has been conducted through the proposed verification framework and experiments with the industrial scenarios and environments.

RQ1. How to quantify the trustworthiness of the sensor and sensor network at run-time?

This question is addressed by developing a sensor behaviour analyser to profile a sensor's normal behaviour and quantify the reverse of sensor readings against its normal behaviour at run time. In Chapter 4, a sensor reading processing framework was provided to profile the sensors and quantify sensors' trustworthiness, namely *Sensor Confidence Score*. This architecture was also evaluated using an industrial CNC turn-mill machine. The results indicated the effectiveness of the proposed approach and framework.

RQ2. How to model a sensor network-based system that reflects the impact of the changes on the sensor's trustworthiness?

As an answer to this question, a traditional formal model is extended with run-time *Sensor Confidence Scores*. As the uncertainty of sensors affects the transition of system states only during the operation period, these *Sensor Confidence Scores* are used to reflect the run-time system behaviour in order to analyse the impact of the changes on the sensors' trustworthiness. In Chapter 4 and 5, two actual sensor network-based systems, CNC turn-mill machine and a passenger lift, are used to evaluate the proposed approach.

RQ3. How can an Industry 4.0 application be verified by detecting the reachability of error states to ensure safety during the operation period, for example, a dynamic manufacturing process using multiple CPSs?

Reaching an error state signifies a disruption in the system's behaviour, rendering it non-operational. If the system enters an error state, it indicates that safe operation is no longer assured. Leveraging probabilistic model checking techniques, a higher probability of transitioning to an error state indicates a greater likelihood of system failure, necessitating the cessation of operation. Therefore, assessing the reachability of an error state serves as a critical safety measure, as it provides an indication of potential hazards. Moreover, the proposed approach and framework are designed for generalisability in Industry 4.0 applications by integrating multiple sub-modules within complex industrial systems. In Chapter 6, a two-cobot-based testbed was introduced to simulate an industrial use case, demonstrating the effectiveness of the proposed approach in evaluating system safety by quantifying the probability of reaching error states.

To summarise, this research employs a data-driven approach to derive the sensor's normal behaviour and quantify the trustworthiness of sensor network readings. Additionally, a model-driven approach is utilised to verify run-time system behaviours. The evaluation results presented in Chapters 4, 5, and 6 demonstrate the efficiency of the proposed framework. These findings offer valuable insights into run-time verification for Industry 4.0 applications and contribute to the existing body of knowledge in the field.

8.2 Future Work

One of the most noteworthy findings of this thesis is a novel approach proposed to verify Industry 4.0 applications using a compositional model checker and take into account the uncertainties of the operation period. This approach opens up new avenues for future exploration and has the potential to impact modern manufacturers to guarantee operation safety and maintain systems more efficiently.

The research questions posed at the beginning of this thesis have been addressed, and the conclusions are in alignment with the initial hypotheses. The data supports our claims, and statistical analysis confirms the validity of our results.

Additionally, certain limitations in the research design that may have influenced the outcomes are acknowledged. These limitations include:

1. Initial model creation highly depends on the expertise or a good quality historical dataset.

2. The sensor profiling process is not generic enough to be practical in the industry.

Despite these challenges, we believe the results remain robust and insightful.

As a conclusion of this thesis, there will always be new questions and unexplored areas waiting for future researchers. They are encouraged to build upon this work and take it to new

heights for future research in this evolving field. The possible future work includes the following three directions:

1. The rules for deriving sensor network confidence scores are manually defined according to expertise on a specific system basis. This should be generalised so that the proposed approach can be applied to a wider range of systems.

2. A context-awareness adaptive algorithm is needed to reflect more general scenarios, e.g., the lift moves with and without passengers.

3. A hierarchical or structural probabilistic model is required to model more sophisticated sensor network-based systems. For instance, to verify a modern building with smart lifts and intelligent power control systems.

This study would not have been possible without the cooperation and participation of support from industry partners, and we express our gratitude for their invaluable contributions. Furthermore, we acknowledge the support received from the Singapore EDB IPP grant and TÜV SÜD Digital Service that made this research financially feasible.

In conclusion, this thesis has achieved its objectives by providing a comprehensive understanding and a novel formal verification approach to verify the Industry 4.0 applications' runtime behaviours and predict potential failures. The results have shed light on various aspects and implications, contributing to the broader knowledge in the field, such as predicting a CNC turn-mill machine failure rate in a certain period or providing predictive maintenance for an operating lift.

I hope that this research will inspire and stimulate further investigations in this area. The insights gained from this study can be applied in modern manufacturers, leading to advancements and improvements in Industry 4.0 applications.

Once again, I extend my sincere gratitude to all those who have supported and encouraged me throughout this academic endeavour.

Appendix A

PRISM code

Following is the source code of individual cobot. It aims to verify the cobot's behaviour according to the requirements and specifications.

```
// Collaborative Robot
dtmc

// Parameterised rates by sensor confidence input
// assume sensor failure rate is failed one second per day

const double zone_confidence = 0.9893;
const double qr_confidence = 0.9999;
const double sensornetwork_confidence = 1;

//****** Above parameters will be input dynamically *******/
// Cobot states of cobot:
// standby -- 0.
// idle -- 1.
// zone detection -- 2.
// workpiece detection -- 3.
// execution -- 4.
// parking -- 5.
// stop -- 6.
// out of barrier -- 7
// e-stop -- 8
// error -- 9
// collision -- 10

module cobot
```

```
26    cobot_status: [0..10] init 0;
27
28    [cob_start] cobot_status=0 -> 1:(cobot_status' = 1);
29    [] cobot_status=1 -> 1:(cobot_status' = 2);
30    [] cobot_status=2 -> (zone_confidence):(cobot_status' = 3)
    ↪  + (1-(zone_confidence)):(cobot_status' = 9);
31    [] cobot_status=3 -> (qr_confidence):(cobot_status' = 4) +
    ↪  (1-(qr_confidence)):(cobot_status' = 9);
32    [] cobot_status=4 ->
    ↪  (sensornetwork_confidence):(cobot_status' = 5) +
    ↪  (1-(sensornetwork_confidence)):(cobot_status' = 7);
33    [] cobot_status=5 ->
    ↪  (sensornetwork_confidence):(cobot_status' = 6) +
    ↪  (1-(sensornetwork_confidence)):(cobot_status' = 7);
34    [cob_end] cobot_status=6 -> 1:(cobot_status' = 0);
35    [] cobot_status=7 -> 1:(cobot_status' = 8);
36    [] cobot_status=8 ->
    ↪  (sensornetwork_confidence):(cobot_status' = 6) +
    ↪  (1-(sensornetwork_confidence)):(cobot_status' = 10);
37    [cob_err] cobot_status=9 -> 1:(cobot_status' = 0);
38    [cob_err] cobot_status=10 -> 1:(cobot_status' = 0);
39  endmodule
40
41  label "complete" = cobot_status = 6;
42  label "detection_error" = cobot_status = 9;
43  label "collision" = cobot_status = 10;
44
```

Below is the process program to verify the behaviour of the painting process including two cobots.

```
1  // Collaborative Robot
2  mdp
3
4  // Parameterised rates by sensor confidence input
5
6  const double sensornetwork_confidence = 0.3479727139406982;
7  const double lidar_confidence =  sensornetwork_confidence;
8  //****** Above paremeters will be input dynamiclly *******/
9
```

```
// Process states of painting system:
// idle -- 0.
// plan -- 1.
// workpiece detection -- 2.
// C1 -- 3.
// confirm state -- 4.
// C2 -- 5.
// failure -- 6

module process
  process_status: [0..6] init 0;
  move_done: bool init false;
        painting_done: bool init false;

  [] process_status=0 -> 1:(process_status' = 1);
  [] process_status=1 -> 1:(process_status' = 2);
  [] process_status=2 -> 1:(process_status' = 2);
//  [c1_start] cobot_status=2 ->
  ↪ (idar_confidence):(cobot_status' = 3) +
  ↪ (1-lidar_confidence):(cobot_status' = 6);
  [c1_start] process_status=2 -> 1:(process_status' = 3);
  [c1_end] process_status=3 -> 1:(process_status' = 4) &
  ↪ (move_done' = true);
  [c1_err] process_status=3 -> (process_status' = 6);
  // confirm status of all devices
  [c2_start] process_status=4 -> 1:(process_status' = 5);
  [] process_status=4 & move_done & painting_done ->
  ↪ (0.9875):(process_status' = 2) +
  ↪ (1-0.9875):(process_status' = 6);
  // [] process_status=4 ->
  ↪ (sensornetwork_confidence):(process_status' = 2) +
  ↪ (1-sensornetwork_confidence):(process_status' = 6);

  [c2_end] process_status=5 -> 1:(process_status' = 4) &
  ↪ (painting_done' = true);
  [c2_err] process_status=5 -> (process_status' = 6);
endmodule

```

```
40 label "failure" = process_status = 6;
41
42 // Parameterised rates by sensor confidence input
43 // assume sensor failure rate is failed one second per day
44
45 const double zone_confidence = 0.9893;
46 const double qr_confidence = 0.9999;
47
48 //****** Above paremeters will be input dynamiclly *******/
49
50 // Cobot states of lift:
51 // standby -- 0.
52 // idle -- 1.
53 // zone detection -- 2.
54 // workpiece detection -- 3.
55 // execution -- 4.
56 // parking -- 5.
57 // stop -- 6.
58 // out of barrier -- 7
59 // e-stop -- 8
60 // error -- 9
61 // collision -- 10
62
63 module cobot1
64   cobot1_status: [0..10] init 0;
65
66   [c1_start] cobot1_status=0 -> 1:(cobot1_status' = 1);
67   [] cobot1_status=1 -> 1:(cobot1_status' = 2);
68   [] cobot1_status=2 -> (zone_confidence):(cobot1_status' =
    ↪  3) + (1-(zone_confidence)):(cobot1_status' = 9);
69   [] cobot1_status=3 -> (qr_confidence):(cobot1_status' = 4)
    ↪  + (1-(qr_confidence)):(cobot1_status' = 9);
70   [] cobot1_status=4 ->
    ↪  (sensornetwork_confidence):(cobot1_status' = 5) +
    ↪  (1-(sensornetwork_confidence)):(cobot1_status' = 7);
71   [] cobot1_status=5 ->
    ↪  (sensornetwork_confidence):(cobot1_status' = 6) +
    ↪  (1-(sensornetwork_confidence)):(cobot1_status' = 7);
```

```
72    [c1_end] cobot1_status=6 -> 1:(cobot1_status' = 0);
73    [] cobot1_status=7 -> 1:(cobot1_status' = 8);
74    [] cobot1_status=8 ->
      ↪  (sensornetwork_confidence):(cobot1_status' = 6) +
      ↪  (1-(sensornetwork_confidence)):(cobot1_status' = 10);
75    [c1_err] cobot1_status=9 -> 1:(cobot1_status' = 0);
76    [c1_err] cobot1_status=10 -> 1:(cobot1_status' = 0);
77  endmodule
78
79  module cobot2 = cobot1[cobot1_status=cobot2_status,
      ↪  c1_start=c2_start, c1_end=c2_end, c1_err=c2_err]
80  endmodule
81
82  label "detection_error" = (cobot1_status = 9|cobot2_status =
      ↪  9);
83  label "collision" = (cobot1_status = 10|cobot2_status = 10);
84
```

Appendix B

Normal Behaviour of Sensors

State	Sensor	Readings	Estimate Range
Open	Pressure	101112.7222 ± 49.3098	[101127.7463, 101070.2508]
	Accelerometer (x)	0.2471 ± 0.0493	[0.2492, 0.2421]
	Accelerometer (y)	0.1291 ± 0.0411	[0.1358, 0.1278]
	Accelerometer (z)	9.8495 ± 0.0476	[9.8531, 9.8458]
	Door Accelerometer (x)	0.2284 ± 0.0507	[0.2339, 0.2230]
	Door Accelerometer (y)	-0.1401 ± 0.0573	[-0.1355, -0.1456]
	Door Accelerometer (z)	9.8909 ± 0.0610	[9.8961, 9.8847]
	Door Magnetometer (x)	361.3239 ± 60.7937	[426.4904, 358.8167]
	Door Magnetometer (y)	-255.4435 ± 41.0223	[-203.3242, -248.9581]
	Door Magnetometer (z)	-1012.8542 ± 49.6644	[-979.2021, -1036.7485]
Opening	Pressure	101111.6576 ± 50.7321	[101133.1830, 101080.1530]
	Accelerometer (x)	0.2455 ± 0.0569	[0.2482, 0.2420]
	Accelerometer (y)	0.1293 ± 0.0680	[0.1346, 0.1271]
	Accelerometer (z)	9.8497 ± 0.0520	[9.8539, 9.8465]
	Door Accelerometer (x)	0.2293 ± 0.1694	[0.2414, 0.2195]
	Door Accelerometer (y)	-0.1312 ± 0.4005	[-0.0964, -0.1596]
	Door Accelerometer (z)	9.8884 ± 0.4878	[9.9223, 9.8525]
	Door Magnetometer (x)	407.1049 ± 302.7329	[463.3033, 333.9414]
	Door Magnetometer (y)	-286.1824 ± 679.9266	[-277.3579, -579.8120]
	Door Magnetometer (z)	-1400.7753 ± 854.3010	[-1204.0743, -1712.1645]
Closing	Pressure	101113.1610 ± 50.6397	[101131.2448, 101072.1170]
	Accelerometer (x)	0.2446 ± 0.0606	[0.2481, 0.2401]
	Accelerometer (y)	0.1286 ± 0.0785	[0.1356, 0.1256]
	Accelerometer (z)	9.8498 ± 0.0447	[9.8527, 9.8471]
	Door Accelerometer (x)	0.2309 ± 0.1465	[0.2429, 0.2224]
	Door Accelerometer (y)	-0.1423 ± 0.3177	[-0.1293, -0.1557]

	Door Accelerometer (z)	9.8939 ± 0.3541	[9.9126, 9.8706]
	Door Magnetometer (x)	357.0492 ± 290.4521	[380.1966, 231.0807]
	Door Magnetometer (y)	-401.9241 ± 703.2059	[-302.0236, -701.4637]
	Door Magnetometer (z)	-1643.3087 ± 769.1128	[-1259.9963, -1755.2282]
Idle	Pressure	101096.9727 ± 49.8653	[101127.5304, 101057.3916]
	Accelerometer (x)	0.2465 ± 0.0376	[0.2498, 0.2414]
	Accelerometer (y)	0.1299 ± 0.0352	[0.1354, 0.1246]
	Accelerometer (z)	9.8497 ± 0.0396	[9.8533, 9.8452]
	Door Accelerometer (x)	0.2332 ± 0.0376	[0.2385, 0.2277]
	Door Accelerometer (y)	-0.1610 ± 0.0368	[-0.1531, -0.1699]
	Door Accelerometer (z)	9.8905 ± 0.0390	[9.8984, 9.8846]
	Door Magnetometer (x)	291.4004 ± 56.5600	[351.4418, 259.9940]
	Door Magnetometer (y)	-641.2680 ± 30.8843	[-578.3093, -685.0291]
	Door Magnetometer (z)	-1617.2318 ± 87.5608	[-1475.7214, -1665.2511]
Moving	Pressure	101121.6097 ± 43.8236	[101134.5944, 101071.3317]
	Accelerometer (x)	0.2405 ± 0.2321	[0.2540, 0.2293]
	Accelerometer (y)	0.1277 ± 0.0946	[0.1338, 0.1213]
	Accelerometer (z)	9.8502 ± 0.5205	[10.2128, 9.6135]
	Door Accelerometer (x)	0.2316 ± 0.0550	[0.2365, 0.2263]
	Door Accelerometer (y)	-0.1562 ± 0.0687	[-0.1422, -0.1685]
	Door Accelerometer (z)	9.8908 ± 0.4942	[10.3320, 9.7330]
	Door Magnetometer (x)	259.6374 ± 93.3942	[352.2076, 231.3671]
	Door Magnetometer (y)	-717.7221 ± 56.9472	[-670.6700, -738.6191]
	Door Magnetometer (z)	-1448.2481 ± 152.1316	[-1366.2226, -1591.3664]

Appendix C

Full experiment result of six scenarios

Table C.1: Sensor confidence score and failure probability.

Test Case	Sensor Confidence Score	Failure Probability
TC_1	1.000	0.0125
TC_2 dataset#1	0.883	0.1054
TC_2 dataset#2	0.8999	0.093
TC_2 dataset#3	0.8776	0.1097
TC_2 dataset#4	0.8862	0.1029
TC_2 dataset#5	0.9006	0.0925
TC_2 dataset#6	0.8796	0.1081
TC_2 dataset#7	0.8815	0.1066
TC_2 dataset#8	0.8995	0.0933
TC_2 dataset#9	0.9029	0.091
TC_3 dataset#1	0.8183	0.1671
TC_3 dataset#2	0.8311	0.1532
TC_3 dataset#3	0.817	0.1685
TC_3 dataset#4	0.8183	0.1671
TC_3 dataset#5	0.8241	0.1607
TC_3 dataset#6	0.8152	0.1706
TC_3 dataset#7	0.8253	0.1594
TC_3 dataset#8	0.8183	0.1671
TC_3 dataset#9	0.8279	0.1567
TC_4 dataset#1	0.8055	0.1817
TC_4 dataset#2	0.802	0.1858
TC_4 dataset#3	0.8106	0.1758
TC_4 dataset#4	0.811	0.1754
TC_4 dataset#5	0.8059	0.1812
TC_4 dataset#6	0.8043	0.1831
TC_4 dataset#7	0.7992	0.1891
TC_4 dataset#8	0.8122	0.174
TC_4 dataset#9	0.8046	0.1828
TC_5	0.6735	0.3635
TC_6	0.3814	0.7905

Bibliography

[1] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, August 2014.

[2] Florian Biesinger, Davis Meike, Benedikt Kraß, and Michael Weyrich. A digital twin for production planning based on cyber-physical systems: A Case Study for a Cyber-Physical System-Based Creation of a Digital Twin. *Procedia CIRP*, 79:355–360, January 2019.

[3] Elisa Negri, Luca Fumagalli, and Marco Macchi. A Review of the Roles of Digital Twin in CPS-based Production Systems. *Procedia Manufacturing*, 11:939–948, January 2017.

[4] Fei Tao, Qinglin Qi, Lihui Wang, and A.Y.C. Nee. Digital Twins and Cyber-Physical Systems toward Smart Manufacturing and Industry 4.0: Correlation and Comparison. *Engineering*, 5(4):653–661, August 2019.

[5] George Lăzăroiu, Mihai Andronie, Mariana Iatagan, Marinela Geamănu, Roxana Ștefănescu, and Irina Dijmărescu. Deep Learning-Assisted Smart Process Planning, Robotic Wireless Sensor Networks, and Geospatial Big Data Management Algorithms in the Internet of Manufacturing Things. *ISPRS International Journal of Geo-Information*, 11(5):277, May 2022.

[6] Sumayah Almuntasheri and Mohammed J. F. Alenazi. Software-Defined Network-Based Energy-Aware Routing Method for Wireless Sensor Networks in Industry 4.0. *Applied Sciences*, 12(19):10073, January 2022.

[7] Edward R Griffor, Chris Greer, David A Wollman, and Martin J Burns. Framework for cyber-physical systems: volume 1, overview. Technical Report NIST SP 1500-201, National Institute of Standards and Technology, Gaithersburg, MD, June 2017.

[8] N. Jazdi. Cyber physical systems in the context of Industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4, Cluj-Napoca, Romania, May 2014. IEEE.

[9] Bernd Dworschak and Helmut Zaiser. Competences for Cyber-physical Systems in Manufacturing – First Findings and Scenarios. *Procedia CIRP*, 25:345–350, 2014.

[10] Amy J. C. Trappey, Charles V. Trappey, Usharani Hareesh Govindarajan, John J. Sun, and Allen C. Chuang. A Review of Technology Standards and Patent Portfolios for Enabling Cyber-Physical Systems in Advanced Manufacturing. *IEEE Access*, 4:7356–7382, 2016.

[11] Yong Ming Wang, Hong Li Yin, Nan Feng Xiao, and Yan Rong Jiang. Internet-based remote manipulation and monitoring of an industry robot in advanced manufacturing systems. *The International Journal of Advanced Manufacturing Technology*, 43(9):907–913, August 2009.

[12] Jian Qin, Ying Liu, and Roger Grosvenor. A Categorical Framework of Manufacturing for Industry 4.0 and Beyond. *Procedia CIRP*, 52:173–178, 2016.

[13] Shen Yin, Steven X. Ding, Xiaochen Xie, and Hao Luo. A Review on Basic Data-Driven Approaches for Industrial Process Monitoring. *IEEE Transactions on Industrial Electronics*, 61(11):6418–6428, November 2014. Conference Name: IEEE Transactions on Industrial Electronics.

[14] Enzo M. Frazzon, Mirko Kück, and Michael Freitag. Data-driven production control for complex and dynamic manufacturing systems. *CIRP Annals*, 67(1):515–518, January 2018.

[15] Junfeng Wang, Qing Chang, Guoxian Xiao, Nan Wang, and Shiqi Li. Data driven production modeling and simulation of complex automobile general assembly plant. *Computers in Industry*, 62(7):765–775, September 2011.

[16] Lin Li, Qing Chang, and Jun Ni. Data driven bottleneck detection of manufacturing systems. *International Journal of Production Research*, 47(18):5019–5036, September 2009. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207540701881860.

[17] Jan Peleska. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. *Electronic Proceedings in Theoretical Computer Science*, 111:3–28, March 2013. arXiv:1303.1006 [cs].

[18] Muthukumar N., Seshadhri Srinivasan, K. Ramkumar, Deepak Pal, Juri Vain, and Srini Ramaswamy. A model-based approach for design and verification of Industrial Internet of Things. *Future Generation Computer Systems*, 95:354–363, June 2019.

[19] Muhammad Waseem Anwar, Muhammad Rashid, Farooque Azam, Aamir Naeem, Muhammad Kashif, and Wasi Haider Butt. A Unified Model-Based Framework for the Simplified Execution of Static and Dynamic Assertion-Based Verification. *IEEE Access*, 8:104407–104431, 2020.

[20] Xin Xin, Sye Loong Keoh, Michele Sevegnani, and Martin Saerbeck. Dynamic Probabilistic Model Checking for Sensor Validation in Industry 4.0 Applications. In *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 43–50, Beijing, China, August 2020. IEEE.

[21] Xin Xin, Sye Loong Keoh, Michele Sevegnani, and Martin Saerbeck. Run-Time Probabilistic Model Checking for Failure Prediction: A Smart Lift Case Study. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, pages 1–7, October 2022.

[22] Xin Xin, Sye Loong Keoh, Michele Sevegnani, Martin Saerbeck, and Teck Ping Khoo. Adaptive Model Verification for Modularized Industry 4.0 Applications. *IEEE Access*, 10:125353–125364, 2022.

[23] Yong Zhi Lim, Xin Xin, and Teck Ping Khoo. Enhancing UAV Flight Safety through Sensor-based Runtime Risk Assessment. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, pages 1–5, October 2022.

[24] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, July 2008. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[25] Angela Pappagallo, Annalisa Massini, and Enrico Tronci. Monte Carlo Based Statistical Model Checking of Cyber-Physical Systems: A Review. *Information*, 11(12):588, December 2020.

[26] Yu Wang, Mojtaba Zarei, Borzoo Bonakdarpoor, and Miroslav Pajic. Probabilistic conformance for cyber-physical systems. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, ICCPS '21, pages 55–66, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Nashville, Tennessee.

[27] Edmund M. Clarke and Jeannette M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[28] IBM CICS Introduction. https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-introduction-cics. [Accessed Jan. 21, 2024].

[29] C-130J Super Hercules. https://www.lockheedmartin.com/en-us/products/c130.html. [Accessed Jan. 21, 2024].

[30] Mehdi Mahmoodjanloo, Reza Tavakkoli-Moghaddam, Armand Baboli, and Ali Bozorgi-Amiri. Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm. *Applied Soft Computing*, 94:106416, September 2020.

[31] Selcuk Cebi, Fatma Kutlu Gündoğdu, and Cengiz Kahraman. Consideration of reciprocal judgments through Decomposed Fuzzy Analytical Hierarchy Process: A case study in the pharmaceutical industry. *Applied Soft Computing*, 134:110000, February 2023.

[32] Hsuan-An Kuo, Chen-Fu Chien, Hans Ehm, and Thomas Ponsignon. A semantic web-based risk assessment framework for collaborative planning to enhance overall supply chain effectiveness for semiconductor industry. *Applied Soft Computing*, 149:110976, December 2023.

[33] Kuldip Singh Sangwan, Rishi Kumar, Christoph Herrmann, Dev Kartik Sharma, and Rushil Patel. Development of a cyber physical production system framework for 3D printing analytics. *Applied Soft Computing*, 146:110719, October 2023.

[34] Runliang Dou, Yanchao Hou, Yixin Wei, and Jing Liu. Dual carbon oriented optimization method for manufacturing industry chain based on BP neural network and clonal selection algorithm. *Applied Soft Computing*, 148:110887, November 2023.

[35] Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar. Chapter 1 - Formal verification: From dreams to reality. In Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar, editors, *Formal Verification*, pages 1–22. Morgan Kaufmann, Boston, January 2015.

[36] Antti Valmari. The state explosion problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491, pages 429–528. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. Series Title: Lecture Notes in Computer Science.

[37] Faranak Nejati, Abdul Azim Abd Ghani, Ng Keng Yap, and Azmi Bin Jafaar. Handling State Space Explosion in Component-Based Software Verification: A Review. *IEEE Access*, 9:77526–77544, 2021.

[38] Wolfgang Bibel. Early History and Perspectives of Automated Deduction. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence*, volume 4667, pages 2–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science.

[39] W. Bibel. Transition Logic Revisited. *Logic Journal of IGPL*, 16(4):317–334, June 2008.

[40] B Meenakshi., Kuntal Das Barman, K. Ganesh Babu, and Karan Sehgal. Formal safety analysis of mode transitions in aircraft flight control system. In *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pages 2.C.1–1–2.C.1–11, Dallas, TX, USA, October 2007. IEEE.

[41] William Denman, Mohamed H. Zaki, Sofiène Tahar, and Luis Rodrigues. Towards Flight Control Verification Using Automated Theorem Proving. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 6617, pages 89–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[42] Dejanira Araiza-Illan, Kerstin Eder, and Arthur Richards. Formal verification of control systems' properties with theorem proving. In *2014 UKACC International Conference on Control (CONTROL)*, pages 244–249, Loughborough, UK, July 2014. IEEE.

[43] Omar A. Jasim and Sandor M. Veres. Formal Verification of Quadcopter Flight Envelop Using Theorem Prover. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1502–1507, Copenhagen, August 2018. IEEE.

[44] Omar A. Jasim and Sandor M. Veres. Verification Framework for Control System Functionality of Unmanned Aerial Vehicles, June 2020. arXiv:2006.10860 [cs, eess].

[45] O. A. Jasim and S. M. Veres. Verification framework for control theory of aircraft. *The Aeronautical Journal*, 127(1307):41–56, January 2023.

[46] Adnan Rashid, Umair Siddique, and Sofiène Tahar. Formal Verification of Cyber-Physical Systems Using Theorem Proving. In Osman Hasan and Frédéric Mallet, editors, *Formal Techniques for Safety-Critical Systems*, volume 1165, pages 3–18. Springer International Publishing, Cham, 2020. Series Title: Communications in Computer and Information Science.

[47] Adnan Rashid and Osman Hasan. Formal analysis of the continuous dynamics of cyber–physical systems using theorem proving. *Journal of Systems Architecture*, 112:101850, January 2021.

[48] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer International Publishing, Cham, 2018.

[49] Marta Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, July 2007.

[50] Edmund Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Completeness and Complexity of Bounded Model Checking. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937, pages 85–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. Series Title: Lecture Notes in Computer Science.

[51] Shaz Qadeer and Jakob Rehof. Context-Bounded Model Checking of Concurrent Software. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440, pages 93–107, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

[52] Lucas Cordeiro, Bernd Fischer, and Joao Marques-Silva. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. *IEEE Transactions on Software Engineering*, 38(4):957–974, July 2012.

[53] Gerard J. Holzmann. Explicit-State Model Checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 153–171. Springer International Publishing, Cham, 2018.

[54] Abhishek Udupa, Ankush Desai, and Sriram Rajamani. Depth Bounded Explicit-State Model Checking. In Alex Groce and Madanlal Musuvathi, editors, *Model Checking Software*, volume 6823, pages 57–74. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[55] Brad Bingham, Jesse Bingham, Flavio M. de Paula, John Erickson, Gaurav Singh, and Mark Reitblatt. Industrial Strength Distributed Explicit State Model Checking. In *2010 Ninth International Workshop on Parallel and Distributed Methods in Verification, and Second International Workshop on High Performance Computational Systems Biology*, pages 28–36, September 2010.

[56] Dirk Beyer and Stefan Löwe. Explicit-State Software Model Checking Based on CEGAR and Interpolation. In Vittorio Cortellessa and Dániel Varró, editors, *Fundamental Approaches to Software Engineering*, volume 7793, pages 146–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. Series Title: Lecture Notes in Computer Science.

[57] Leonardo de Moura and Nikolaj Bjørner. Satisfiability Modulo Theories: An Appetizer. In Marcel Vinícius Medeiros Oliveira and Jim Woodcock, editors, *Formal Methods: Foundations and Applications*, Lecture Notes in Computer Science, pages 23–36, Berlin, Heidelberg, 2009. Springer.

[58] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, September 2011.

[59] Clark Barrett and Cesare Tinelli. Satisfiability Modulo Theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 305–343. Springer International Publishing, Cham, 2018.

[60] Marta Kwiatkowska, Gethin Norman, and David Parker. Quantitative Analysis With the Probabilistic Model Checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, May 2006.

[61] Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 263–272, San Juan, PR, USA, December 2012. IEEE.

[62] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Quantitative analysis of real-time systems using priced timed automata. *Communications of the ACM*, 54(9):78–87, September 2011.

[63] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model Checking and the State Explosion Problem. In Bertrand Meyer and Martin Nordio, editors, *Tools for Practical Software Verification*, volume 7682, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. Series Title: Lecture Notes in Computer Science.

[64] Dirk Beyer and Andreas Podelski. Software Model Checking: 20 Years and Beyond. In Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Rupak Majumdar, editors, *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 554–582. Springer Nature Switzerland, Cham, 2022.

[65] Muffy Calder, Simon Dobson, Michael Fisher, and Julie McCann. Making Sense of the World: Framing Models for Trustworthy Sensor-Driven Systems. *Computers*, 7(4):62, November 2018.

[66] Radu Calinescu and Shinji Kikuchi. Formal Methods @ Runtime. In Radu Calinescu and Ethan Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662, pages 122–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[67] César Sánchez, Gerardo Schneider, Wolfgang Ahrendt, Ezio Bartocci, Domenico Bianculli, Christian Colombo, Yliès Falcone, Adrian Francalanza, Srđan Krstić, Joa o M. Lourenço, Dejan Nickovic, Gordon J. Pace, Jose Rufino, Julien Signoles, Dmitriy Traytel, and Alexander Weiss. A survey of challenges for runtime verification from advanced application domains (beyond software). *Formal Methods in System Design*, 54(3):279–335, November 2019.

[68] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In Ezio Bartocci and Yliès Falcone, editors,

Lectures on Runtime Verification, volume 10457, pages 135–175. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[69] Sebastian Junges, Hazem Torfah, and Sanjit A. Seshia. Runtime Monitors for Markov Decision Processes. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 553–576, Cham, 2021. Springer International Publishing.

[70] Vojtěch Forejt, Marta Kwiatkowska, David Parker, Hongyang Qu, and Mateusz Ujma. Incremental Runtime Verification of Probabilistic Systems. In Shaz Qadeer and Serdar Tasiran, editors, *Runtime Verification*, volume 7687, pages 314–319. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. Series Title: Lecture Notes in Computer Science.

[71] Peter Faymonville, Bernd Finkbeiner, Malte Schledjewski, Maximilian Schwenger, Marvin Stenger, Leander Tentrup, and Hazem Torfah. StreamLAB: Stream-based Monitoring of Cyber-Physical Systems. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 421–431, Cham, 2019. Springer International Publishing.

[72] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[73] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[74] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806, pages 585–591. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[75] Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098, pages 87–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. Series Title: Lecture Notes in Computer Science.

[76] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Verifier. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, volume 1633, pages 495–499. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. Series Title: Lecture Notes in Computer Science.

[77] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification*, volume 2404, pages 359–364. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. Series Title: Lecture Notes in Computer Science.

[78] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, volume 10427, pages 592–600. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[79] André Platzer. A Complete Uniform Substitution Calculus for Differential Dynamic Logic. *Journal of Automated Reasoning*, 59(2):219–265, August 2017.

[80] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 432–442, Cham, 2019. Springer International Publishing.

[81] Muffy Calder and Michele Sevegnani. Stochastic Model Checking for Predicting Component Failures and Service Availability. *IEEE Transactions on Dependable and Secure Computing*, 16(1):174–187, January 2019.

[82] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, March 2009.

[83] Michele Sevegnani, Milan Kabac, Muffy Calder, and Julie McCann. Modelling and Verification of Large-Scale Sensor Network Infrastructures. In *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 71–81, Melbourne, VIC, December 2018. IEEE.

[84] Michele Sevegnani and Muffy Calder. Bigraphs with sharing. *Theoretical Computer Science*, 577:43–73, April 2015.

[85] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 341–350, Waikiki, Honolulu HI USA, May 2011. ACM.

[86] Nianyu Li, Sridhar Adepu, Eunsuk Kang, and David Garlan. Explanations for human-on-the-loop: a probabilistic model checking approach. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '20, pages 181–187, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Seoul, Republic of Korea.

[87] Richard Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

[88] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Model evolution by run-time parameter adaptation. In *2009 IEEE 31st International Conference on Software Engineering*, pages 111–121, Vancouver, BC, Canada, 2009. IEEE.

[89] Shaukat Ali and Tao Yue. U-Test: Evolving, Modelling and Testing Realistic Uncertain Behaviours of Cyber-Physical Systems. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–2, Graz, Austria, April 2015. IEEE.

[90] Mah Noor Asmat, Saif Ur Rehman Khan, and Shahid Hussain. Uncertainty handling in cyber–physical systems: State-of-the-art approaches, tools, causes, and future directions. *Journal of Software: Evolution and Process*, 35(7):e2428, July 2023.

[91] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. Sensor network data fault types. *ACM Transactions on Sensor Networks*, 5(3):1–29, May 2009.

[92] Abhishek B. Sharma, Leana Golubchik, and Ramesh Govindan. Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks*, 6(3):1–39, June 2010.

[93] N. Ramanathan, L. Balzano, M. Burt, D. Estrin, T. Harmon, C. Harvey, J. Jay, E. Kohler, S. Rothenberg, and M.Srivastava. Rapid Deployment with Confidence:Calibration and Fault Detection in Environmental Sensor Networks. Technical report, UCLA: Center for Embedded Network Sensing, 2006. UCLA: Center for Embedded Network Sensing. Retrieved from https://escholarship.org/uc/item/8v26b5qh.

[94] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2):105–112, March 2009.

[95] Zhen Hu and Sankaran Mahadevan. Uncertainty quantification and management in additive manufacturing: current status, needs, and opportunities. *The International Journal of Advanced Manufacturing Technology*, 93(5):2855–2874, November 2017.

[96] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 63–78, New York, NY, USA, 2019. Association for Computing Machinery. event-place: Phoenix, AZ, USA.

[97] Hazem Torfah, Sebastian Junges, Daniel J. Fremont, and Sanjit A. Seshia. Formal Analysis of AI-Based Autonomy: From Modeling to Runtime Assurance. In Lu Feng and Dana Fisman, editors, *Runtime Verification*, Lecture Notes in Computer Science, pages 311–330, Cham, 2021. Springer International Publishing.

[98] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 215–224, Cape Town South Africa, May 2010. ACM.

[99] Susmit Jha and Sanjit A. Seshia. A theory of formal synthesis via inductive learning. *Acta Informatica*, 54(7):693–726, November 2017.

[100] Sanjit A. Seshia. Introspective Environment Modeling. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification*, volume 11757, pages 15–26. Springer International Publishing, Cham, 2019. Series Title: Lecture Notes in Computer Science.

[101] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Toward verified artificial intelligence. *Commun. ACM*, 65(7):46–55, June 2022. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[102] Håkan L. S. Younes and Reid G. Simmons. Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification*, volume 2404, pages 223–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. Series Title: Lecture Notes in Computer Science.

[103] Edmund M. Clarke and Paolo Zuliani. Statistical Model Checking for Cyber-Physical Systems. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis*, volume 6996, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[104] Håkan L. S. Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, June 2006.

[105] Mojtaba Zarei, Yu Wang, and Miroslav Pajic. Statistical verification of learning-based cyber-physical systems. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–7, Sydney New South Wales Australia, April 2020. ACM.

[106] Stefan Mitsch and André Platzer. ModelPlex: verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design*, 49(1-2):33–74, October 2016.

[107] André Platzer. Differential Dynamic Logics: Automated Theorem Proving for Hybrid Systems. *KI - Künstliche Intelligenz*, 24(1):75–77, April 2010.

[108] Rose Bohrer, Yong Kiam Tan, Stefan Mitsch, Andrew Sogokon, and André Platzer. A Formal Safety Net for Waypoint-Following in Ground Robots. *IEEE Robotics and Automation Letters*, 4(3):2910–2917, July 2019. Conference Name: IEEE Robotics and Automation Letters.

[109] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *The International Journal of Robotics Research*, 36(12):1312–1340, October 2017.

[110] Brandon Bohrer, Adriel Luo, Xue An Chuang, and André Platzer. CoasterX: A Case Study in Component-Driven Hybrid Systems Proof Automation. *IFAC-PapersOnLine*, 51(16):55–60, 2018.

[111] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195, pages 527–538. Springer International Publishing, Cham, 2015. Series Title: Lecture Notes in Computer Science.

[112] Jan-David Quesel, Stefan Mitsch, Sarah Loos, Nikos Aréchiga, and André Platzer. How to model and prove hybrid systems with KeYmaera: a tutorial on safety. *International Journal on Software Tools for Technology Transfer*, 18(1):67–91, February 2016.

[113] Norihiro Kamide and Ryu Yano. Logics and translations for hierarchical model checking. *Procedia Computer Science*, 112:31–40, 2017.

[114] Ankush Desai, Tommaso Dreossi, and Sanjit A. Seshia. Combining Model Checking and Runtime Verification for Safe Robotics. In Shuvendu Lahiri and Giles Reger, editors, *Runtime Verification*, volume 10548, pages 172–189. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[115] Jeff Huang, Cansu Erdogan, Yi Zhang, Brandon Moore, Qingzhou Luo, Aravind Sundaresan, and Grigore Rosu. ROSRV: Runtime Verification for Robots. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, volume 8734, pages 247–254. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science.

[116] Angelo Ferrando, Rafael C. Cardoso, Michael Fisher, Davide Ancona, Luca Franceschini, and Viviana Mascardi. ROSMonitoring: A Runtime Verification Framework for ROS. In Abdelkhalick Mohammad, Xin Dong, and Matteo Russo, editors, *Towards Autonomous Robotic Systems*, volume 12228, pages 387–399. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[117] Attie Paul C. and Lynch Nancy A. Dynamic input/output automata: A formal and compositional model for dynamic systems. *Elsevier {BV}*, 249:28–75, August 2016.

[118] Pierre Civit and Maria Potop-Butucaru. Dynamic Probabilistic Input Output Automata. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969.

[119] Mihai Andronie, George Lăzăroiu, Mariana Iatagan, Cristian Uță, Roxana Ștefănescu, and Mădălina Cocoșatu. Artificial Intelligence-Based Decision-Making Algorithms, Internet of Things Sensing Networks, and Deep Learning-Assisted Smart Process Management in Cyber-Physical Production Systems. *Electronics*, 10(20):2497, January 2021. Number: 20 Publisher: Multidisciplinary Digital Publishing Institute.

[120] Alberto Villalonga, Elisa Negri, Giacomo Biscardo, Fernando Castano, Rodolfo E. Haber, Luca Fumagalli, and Marco Macchi. A decision-making framework for dynamic scheduling of cyber-physical production systems based on digital twins. *Annual Reviews in Control*, 51:357–373, January 2021.

[121] Mihai Andronie, George Lăzăroiu, Roxana Ștefănescu, Cristian Uță, and Irina Dijmărescu. Sustainable, Smart, and Sensing Technologies for Cyber-Physical Manufacturing Systems: A Systematic Literature Review. *Sustainability*, 13(10):5495, January 2021.

[122] Dima Alberg and Mark Last. Short-term load forecasting in smart meters with sliding window-based ARIMA algorithms. *Vietnam Journal of Computer Science*, 5(3):241–249, September 2018.

[123] Wootae Jeong and Shimon Y. Nof. A collaborative sensor network middleware for automated production systems. *Computers & Industrial Engineering*, 57(1):106–113, August 2009.

[124] L. Q. Zhuang, D. H. Zhang, and M. M. Wong. Wireless Sensor Networks for Networked Manufacturing Systems. In Javier Silvestre-Blanes, editor, *Factory Automation*. IntechOpen, Rijeka, 2010. Section: 7.

[125] Kh.E Khujamatov and T K Toshtemirov. Wireless sensor networks based Agriculture 4.0: challenges and apportions. In *2020 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–5, November 2020.

[126] Teck Ping Khoo and Jun Sun. The Miles Before Formal Methods - A Case Study on Modeling and Analyzing a Passenger Lift System. In Jing Sun and Meng Sun, editors, *Formal Methods and Software Engineering*, volume 11232, pages 54–69. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[127] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning*, 86(3):295–333, March 2012.

[128] James Buckner, Barry L. Chin, and Jon Henri. Prometrix RS35e Gauge Study in Five Two-Level Factors and One Three-Level Factor. In *Statistical Case Studies for Industrial Process Improvement*, pages 9–17. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1997. _eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719765.ch2.

[129] Mark Utting, Bruno Legeard, Fabrice Bouquet, Elizabeta Fourneret, Fabien Peureux, and Alexandre Vernotte. Recent Advances in Model-Based Testing. In *Advances in Computers*, volume 101, pages 53–120. Elsevier, 2016.

[130] Jan Vachalek, Lukas Bartalsky, Oliver Rovny, Dana Sismisova, Martin Morhac, and Milan Loksik. The digital twin of an industrial production line within the industry 4.0 concept. In *2017 21st International Conference on Process Control (PC)*, pages 258–262, Strbske Pleso, Slovakia, June 2017. IEEE.

[131] Adam Thelen, Xiaoge Zhang, Olga Fink, Yan Lu, Sayan Ghosh, Byeng D. Youn, Michael D. Todd, Sankaran Mahadevan, Chao Hu, and Zhen Hu. A comprehensive review of digital twin — part 1: modeling and twinning enabling technologies. *Structural and Multidisciplinary Optimization*, 65(12):354, November 2022.

[132] Sudipta Paria, Aritra Dasgupta, and Swarup Bhunia. DIVAS: An LLM-based End-to-End Framework for SoC Security Analysis and Policy-based Protection, August 2023. arXiv:2308.06932 [cs].

[133] Maxwell Crouse, Ibrahim Abdelaziz, Kinjal Basu, Soham Dan, Sadhana Kumaravel, Achille Fokoue, Pavan Kapanipathi, and Luis Lastras. Formally Specifying the High-Level Behavior of LLM-Based Agents, October 2023. arXiv:2310.08535 [cs].

[134] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. LLM-assisted Generation of Hardware Assertions, June 2023. arXiv:2306.14027 [cs].

[135] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to Active Automata Learning from a Practical Perspective. In Marco Bernardo and Valérie Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659, pages 256–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[136] Amel Bennaceur and Karl Meinke. Machine Learning for Software Analysis: Models, Methods, and Applications. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits*, volume 11026, pages 3–49. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[137] Falk Howar and Bernhard Steffen. Active Automata Learning in Practice: An Annotated Bibliography of the Years 2011 to 2016. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits*, volume 11026, pages 123–148. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[138] Fujun Wang, Zining Cao, Lixing Tan, and Hui Zong. Survey on Learning-Based Formal Methods: Taxonomy, Applications and Possible Future Directions. *IEEE Access*, 8:108561–108578, 2020.

[139] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. Temporal logic inference for classification and prediction from data. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 273–282, Berlin Germany, April 2014. ACM.