UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA
ETS Ingeniería Informática
Departamento Lenguajes y Ciencias de la Computación
Programa de Doctorado de Tecnologías Informáticas

Measuring the Fidelity of Digital Twins using Trace Alignments

Midiendo la Fidelidad de los Gemelos Digitales usando Alineamiento de Trazas

Tesis Doctoral

PAULA MUÑOZ ARIZA

Directores:
Antonio Vallecillo Moreno
Javier Troya Castilla

Málaga, 2024

UNIVERSIDAD
DE MÁLAGA

AUTORA: Paula Muñoz Ariza

https://orcid.org/0000-0003-2939-5803

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña PAULA MUÑOZ ARIZA
Estudiante del programa de doctorado TECNOLOGÍAS INFORMÁTICAS de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: MEASURING THE FIDELITY OF DIGITAL TWINS USING TRACE ALIGNMENTS

Realizada bajo la tutorización de JAVIER TROYA CASTILLA y dirección de ANTONIO VALLECILLO MORENO Y JAVIER TROYA CASTILLA (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 20 de DICIEMBRE de 2024

Fdo.: PAULA MUÑOZ ARIZA Doctorando/a	Fdo.: JAVIER TROYA CASTILLA Tutor/a
Fdo.: ANTONIO VALLECILLO MORENO Y JAVIER TROYA CASTILLA Director/es de tesis	

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

A todos los que os quedasteis

UNIVERSIDAD
DE MÁLAGA

Universidad de Málaga

Universidad de Málaga
ETS Ingeniería Informática
Departamento Lenguajes y Ciencias de la Computación

El Dr. Antonio Vallecillo Moreno, Catedrático de Universidad, y el Dr. Javier Troya Castilla, Profesor Titular ambos en el Departamento de Lenguages y Ciencias de la Computación de la E.T.S. de Ingeniería Informática de la Universidad de Málaga,

certifican que Dña. Paula Muñoz Ariza, Ingeniera de Software, ha realizado en el Departamento de Lenguages y Ciencias de la Computación de la Universidad de Málaga, bajo su dirección, el trabajo de investigación correspondiente a la Tesis Doctoral titulada:

Measuring the Fidelity of Digital Twins
using Trace Alignments

Revisado el presente trabajo, estimamos que puede ser presentado al tribunal que ha de juzgarlo, y autorizamos la presentación de esta Tesis Doctoral en la Universidad de Málaga.

Del mismo modo certifican que las publicaciones que avalan dicha Tesis Doctoral no han sido utilizadas en tesis anteriores.

En Málaga, Diciembre de 2024

. . . . . .
Dr. Antonio Vallecillo Moreno

. . . . . .
Dr. Javier Troya Castilla

AGRADECIMIENTOS

Writing a thesis. Developing a project. Creating a product.

These are all processes that lead to an artifact. But too often, we fixate on the end result and forget where the real magic happens—buried in everyday mistakes, moments of confusion, and the occasional bolt of clarity that feels like a hard-won treasure. These words are for anyone feeling lost or overwhelmed in their own journey. You're not alone—it's all part of the process.

Let me take you behind the scenes of this thesis. Behind these pages of polished results lies a story, a messy process full of wrong turns, persistence, and the occasional *apparently* accidental breakthrough. Getting better doesn't always feel good. It's uncomfortable. Full of failures. But it's also rewarding and empowering.

And let's be honest: you're reading this section because you want the juicy details—the backstory, the gossip about the people I worked with. You want to know me better. Fair enough. Maybe someday, we'll chat about it in person. But for now, I'll give you a story about taking the hard path and surviving the climb.

Every story needs a beginning. Mine starts at 9 a.m., in a corner of the lab—our name for what's really just an office full of computers. My desk sat near the window, soaking up the brightest spot in the room. That morning, I was absorbed in debugging a code for an article on subjective logic, headphones on, completely oblivious to the world. Suddenly, I nearly jumped out of my chair—Antonio had appeared beside me, brimming with excitement.

"The modeling community is shifting towards digital twins," he announced, dropping this idea like a bombshell. "We should look into it."

And just like that, I was off down a new rabbit hole. Hours turned into weeks as Antonio, Javi, and I debated what a Digital Twin even was. We hashed out ideas, challenged each other, and gradually began to shape some clear guidelines. It wasn't easy, but the best ideas rarely come fully formed. Sometimes, you have to push and collide with

others to learn what works.

Some inspiration came during a conference in a foggy castle in Germany, where the days were filled with presentations, heated discussions, and the aroma of coffee breaks. There's a certain magic in gathering the right people in the right place at the right time—it's how you stumble across the right answers.

Eventually, we felt confident we understood what a Digital Twin truly was. That's when Antonio struck again. One morning, he appeared, paper in hand—"Multi-Fidelity Digital Twins" by Aitor Arrieta. "We need a research question," he said, and I could tell he already had one in mind. The challenge? People were building Digital Twins of everything—human bodies, cars, farms—but how could we systematically validate these complex, heterogeneous systems within a unified framework?

If I'm getting too technical, don't worry. The details are all in the chapters ahead. What mattered then was this: the mountain to climb had been named. Now, it was time to lace up our boots.

This next part? It felt like being left on a boat in the middle of the ocean, handed a paddle, and told to figure it out. No map. No guarantees. Just me, the literature, and the stubborn hope that I could navigate my way to something meaningful.

For weeks, I sat at my desk by the window, reading paper after paper, searching for what made Digital Twins distinct. Nothing clicked—until Antonio slipped a stack of papers into my mailbox. Among them was one titled What Good Are Models by Edward Lee.

I need to pause here to share a personal highlight. In 2023, I had the chance to attend a keynote by Edward Lee in Ciudad Real. Meeting your heroes is always a gamble, but this was one of those moments where reality lived up to the hype. His brilliance was inspiring, and that full-circle moment still stays with me.

Back to the story. Reading Lee's paper, I had a sudden flash of insight: the inherent symmetry of Digital Twins. Two systems, mirroring each other, faithful in behavior like a model and its reality. That was the spark for my first hypothesis.

Months passed as I refined that hypothesis, layering concepts like fidelity, abstraction, and resolution. We even wrote an article about it, but it was rejected. I won't lie—it stung. But rejection isn't failure; it's part of the learning curve. That experience helped us sharpen our ideas and push forward.

Next came a new adventure: Austria. I traded my desk by the window for a new one (still by a window!) in another country. For four months, I worked alongside brilliant colleagues, explored parks filled with ducks, and even learned to eat a plate of pasta at 11:00 a.m.

At first, I floundered. New place, new people, new problems—it was overwhelming. But then, one morning (always in the morning!), my host supervisor, Manuel Wimmer, stopped by and we spent an hour brainstorming on the whiteboard. Another pile of ideas in which one shone over the others: BLAST. That word, that simple nudge, set everything in motion.

As a proof that this is a non-fictional story, I leave the photo of the whiteboard next.

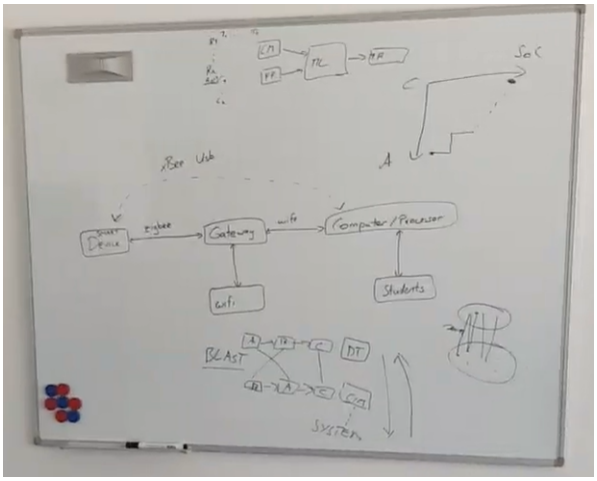


Figure 1: Whiteboard in CDL-Mint Lab in JKU (Austria).

The process wasn't easy from this point onward. We developed an incremental approach, tackling one problem at a time. There were countless meetings huddled around whiteboards and endless brainstorming sessions where we critiqued every proposal, refining them again and again in search of a creative and unique solution. But by then, we could finally see land on the horizon.

I could tell you so many more stories. If protocol allowed, I'd write an extra section for each chapter, spilling every anecdote about how each idea was born. But for now, I'll leave you with this:

This journey wasn't smooth. It was full of uncertainty, stress, but it was also deeply fulfilling. The solution we've developed isn't just the end of this thesis—it's a starting point, and I hope to keep building on it, knowing it could someday make a broader impact.

Needless to say, thank you to everyone who joined me on this journey and helped me run out of whiteboard pens.

First, I want to thank my thesis director, Antonio Vallecillo, who pushed me into this adventure in the first place and has accompanied me every step of the way—from Málaga

to even across the Atlantic Ocean. Thank you for teaching me everything about research, from trusting the *uncertain* process of creating new solutions to objectively criticizing one's work for improvement. You showed me that the shortest path isn't always the best one.

Thank you to my supervisor and co-director, Javier Troya, for walking this entire path with us, offering kind and thoughtful words, and contributing meaningfully at every stage to make this work a success.

Thank you to Manuel Wimmer for being the best possible host during my stay in Linz, always offering encouraging words and creative insights that enriched all our discussions.

Thank you to Aitor Arrieta, not only for writing the work that leads us to the research question that became a key part of this work, but also for inviting me to Mondragon, where I collected the first elevator traces of our proposal carefully analyzed in the following pages. Also, thanks to Cláudio Gomes and his team for providing data from the incubator.

Finally, I want to thank the two external reviewers for taking the time to carefully review the first version of this document. Your feedback greatly helped us improve the thesis.

Now, I want to thank the other people that are part of the stories that I had to leave out of these pages, but are part of the everyday process that amount to the final result. Thank you, Loli, for being one of the reasons I'm defending this thesis—for co-directing my bachelor's thesis and giving me opportunities to collaborate with you. Thank you for still finding ways to help whenever you can.

Thank you to all my colleagues from the research groups Atenea and Scenic for always being there with considerate words and a readiness to help. Special thanks to everyone from the lab—Rafa, Dani, Josemi, Raquel, Inma, and many more who came and went over the years.

Ahora cambio de idioma, porque fuera de la academia hay mucha gente a la que agradecer, y no todo el mundo sabe inglés.

En primer lugar, gracias a mis padres, Victoria y Alonso, por haberme acompañado en cada paso del camino y siempre haber creído en mí y en todo lo que podía lograr. Nunca me ha faltado una palabra de ánimo ni han dudado en ayudarme con cualquier cosa que estuviera en su mano. Tener un hogar seguro es esencial para llegar lejos, y siempre habéis estado ahí para dármelo.

Gracias a mis abuelos, Eugenia y Antonio, porque se dedicaron a cuidarme y a *bregar* conmigo desde que tengo memoria, haciendo de su casa mi segunda casa. Me enseñaron con su ejemplo la importancia del sacrificio y el esfuerzo para conseguir lo que te propones.

Gracias a mi abuela Lola, por todo su cariño y comprensión. Y gracias al resto de mi familia por estar siempre ahí.

Ahora viene la lista interminable de personas maravillosas que me han acompañado entre bastidores para llegar aquí. Muchas gracias a todas mis amigas, que son una bendición para mi vida.

Gracias, Ane, porque llevar 26 años siendo tu amiga es lo más bonito que me ha pasado. No hay nada que se sienta mejor ni más familiar que hablar contigo de cosas que pasaron hace un siglo y reírnos porque ambas las vivimos. No hay mayor confort que saber que cuando no había nada, siempre estabas tú.

Gracias, Mary, por aparecer y tenderme la mano cuando más lo necesitaba. Ya no sé qué sería de todo esto sin ti, sin nuestras reinas, sin el consenso. Gracias, Cris, MJ y Mary, por las mejores aventuras, las risas, el cariño y la comprensión. Porque pensar que he quedado con vosotras esta tarde ya hace que el día sea mucho más divertido.

Gracias a mi Consenso: Lu, Bea, Mary, MJ. Me da igual que le hayáis cambiado el nombre al grupo. Las tardes de chisme y cafelito me reviven, mientras compartimos las incertidumbres de la vida adulta. Y por las aventuras que nos faltan, porque aun tengo que encontrar la rubia que nos guste a las cinco para que nos vayamos de concierto juntas en 2026.

Gracias, Ana, por aparecer, por ser ese punto de inflexión, por siempre hacerme sentir que me entiendes perfectamente cuando algo me pasa, y por ser cómplice de los planes locos que ya hemos vivido y los que nos quedan por vivir.

Gracias, Juan, por haber compartido con nosotros tantas comidas y por ser el compañero de conversación más interesante que puedes tener. Gracias a ti y a Alicia por haber estado incondicionalmente en este viaje.

Por último, tengo que darle las gracias a mi pareja, Fernando. Gracias por haber aparecido hace ya tres años, con comprensión, cariño y sin juicios, para rellenar las grietas. Por no dejarme de lado ni un segundo y por seguir siendo el motor y el apoyo de toda esta ambición que está resurgiendo donde parecía que ya no quedaba nada. Haces de nuestra casa el lugar seguro que necesito.

Paula Muñoz Ariza

ABSTRACT

Digital twins have emerged as a transformative technology for enhancing performance, maintenance, and anomaly detection in cyber-physical systems. However, despite their growing importance, key challenges remain regarding their definition, architecture, and validation. This thesis aims to contribute to two critical aspects of digital twins. First, we propose an architecture for the implementation of digital twins, comparing it with existing approaches in the literature to find a common understanding. Second, we address the validation of digital twins by assessing their fidelity. Specifically, we introduce a method to evaluate whether the physical and digital twins exhibit twinned behaviors through a three-step process. We define a measure of equivalence between system snapshots, a trace alignment algorithm to match states, and quantify fidelity based on the alignment of behavioral traces. Our approach has been validated on four cyber-physical systems, demonstrating its efficacy in identifying faithful and unfaithful behaviors, with a comparative analysis of similar validation methods.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

BLAST	Basic Local Alignment Search Tool
CPS	Cyber-Physical System
DL	Data Lake
DS	Digital Shadow
DT	Digital Twin
DTS	Digital Twin System
DTW	Dynamic Time Warping
LCAW	Low-Complexity Area Weight
MAD	Maximum Acceptable Distance
NDW	Needleman-Wunsch
OME	Observable Manufacturing Element
PT	Physical Twin
SCG	Sequences of Consecutive Gaps

1

INTRODUCTION

1.1 Context

Digital Twins (DTs) have proven to be powerful tools for optimizing the performance of cyber-physical systems, processes, and software services (Dalibor et al. 2022; Digital Twin Consortium 2021). The core idea behind DTs is the creation of a digital replica of an existing physical system, often referred to as the Physical Twin (PT). This concept originated during NASA's Apollo mission, where engineers developed physical replicas of systems used in space to assist astronauts by simulating system behaviors and performing what-if analyses (Grieves and Vickers 2017). These early physical replicas allowed ground engineers to simulate potential scenarios, helping ensure the safety of the mission and the integrity of the spacecraft.

Initially, the replicas were physical, and synchronization was done manually. However, as the potential benefits became evident, these replicas transitioned to digital forms using simulations or analytical models, such as differential equations, and synchronization was automated. Today, DTs play a crucial role in modern systems, representing various aspects of their PTs, including physical constraints, appearance, and behavior, with specific levels of fidelity and synchronization intervals (Dalibor et al. 2022).

The applications of DTs are vast. They can be used to improve the design of new systems by simulating potential behaviors before physical prototypes are built, or they can be developed alongside PTs to enhance operational efficiency. Additionally, DTs are employed post-implementation to monitor and optimize system behavior, allowing real-time decision-making and predictive maintenance (Muñoz, Troya, and Vallecillo 2023).

1.2 Problem Statement

A key advantage of Digital Twins is their ability to replicate the characteristics of their physical counterparts, allowing for advanced simulations, monitoring, and predictive analysis. However, without robust validation, there is no guarantee that DTs will replicate the systems with sufficient accuracy, undermining their value and trustworthiness. In this context, the engineering of DTs becomes critical (Bordeleau et al. 2020). While much of the current work focuses on building and deploying DTs, there is a significant gap in the field of validation (Dalibor et al. 2022). Therefore, it is essential to work on systematic validation approaches that assess the fidelity of the digital model throughout its lifecycle, ensuring that it remains a faithful representation of the real-world system.

In this context, even though the term "Digital Twin" was initially introduced over a decade ago by Grieves in his white paper (Grieves 2014), it has only recently garnered significant attention in both academia and industry (Gartner, Inc. 2018). For this reason, the precise definition of "Digital Twin" is still a work in progress, with over a hundred different definitions in the literature (Wortmann 2023). In this broad ecosystem of definitions, it is becoming hard for the community to convey its ideas. Some researchers view DTs as virtual replicas of physical systems, while others broaden the definition to include connections, databases, or even services like anomaly detection. This lack of alignment hinders effective communication, slows the consolidation of foundational knowledge, and complicates efforts to evaluate and compare new research.

This makes it imperative to establish a clear set of requirements and precise definitions for the various terms and architectural elements of a Digital Twin before attempting to validate it.

1.3 Contributions

This thesis presents two key contributions: first, a modular architecture along with a set of precise definitions to clarify the fundamental terminology of digital twins; second, a trace alignment algorithm accompanied by a set of metrics designed to evaluate the level of fidelity between the behavior of the physical and digital twins. These are described next.

1.3.1 Conceptual Digital Twin Architecture

In this thesis, we introduce a modular architecture based on the concept of a Digital Twin System (DTS). A DTS includes not only the physical system and its digital replica but also the interactions between the two (Muñoz, Troya, and Vallecillo 2023). The DT can take different forms, such as algorithms, simulation models, or analytical models defined by differential equations. It is important to note that a simulation model or algorithm, on

its own, cannot be considered a digital twin. These components only become a DT when they are part of a DTS, where they maintain automatic, bidirectional synchronization with the physical system (the PT) and exchange data and commands to optimize overall performance (Kritzinger et al. 2018).

This thesis aims to provide a precise definition of the key concepts that make up a DTS. The proposed architecture also focuses on enhancing the scalability and composability of digital twins. To support this, the architecture is built around a Data Lake (DL) (Hai, Quix, and Jarke 2021)—a centralized repository designed to store, process, and secure large volumes of structured, semi-structured, and unstructured data. A set of drivers access the DL to manage and coordinate the flow of data and commands between the elements of the DTS. We define the architecture and its main components and compare it to other conceptual frameworks commonly used in the literature.

1.3.2 Trace Alignment Algorithm to Measure Fidelity

In this thesis, we propose an offline method to evaluate the *fidelity* of DTs. Typically, model validation involves comparing the output behavior of the physical system with that of its simulation model (Sargent 2013). The results of this comparison indicate the level of accuracy of the digital model (Lugaresi et al. 2023). More specifically, we measure the degree of fidelity of the DT in relation to its PT.

Fidelity can refer to both the structure and behavior of the system (Gross 1999). In our context, we focus on validating the behavioral fidelity. Our approach evaluates fidelity by comparing the behavioral traces of both twins. In this context, a trace is a sequence of snapshots, each representing the system's state at a specific point in time. To compare the traces, we employ a three-step process. First, we define a comparison function to evaluate whether two individual snapshots (i.e., system states) are sufficiently similar. Second, we use a trace alignment algorithm to align the equivalent states reached by the two twins, using a specified similarity threshold for each property of interest. Finally, we measure the fidelity of the twins' behavior based on the alignment results using two metrics: the level of alignment (percentage of matched snapshots) and the average and maximum distances between the aligned traces. In this process, we also consider the fidelity of the structure to some extent, as one of the prerequisites is that the systems snapshots contain the same set of observable properties of the system. This means that we are also validating the structure of the system states.

1.4 Research Methodology

Design Science (DS) is defined as the scientific study and creation of artifacts developed and used by people to solve practical problems of general interest (Wieringa 2014). This discipline differs from pure sciences in that its focus extends beyond describing the world to actively creating solutions that address human needs. In this work, we applied the DS methodology to develop and evaluate an artifact that improves the state of the art in digital twin systems.

We followed the framework introduced by Johannesson and Perjons (2014), which structures the research process into distinct phases. Each phase was systematically applied in our research as follows:

1. **Problem Explanation:** We began by thoroughly analyzing the challenges associated with validating the behavior fidelity of DTSs. This analysis helped us define the problem precisely and justify its practical relevance in the context of DTS design and development.
2. **Definition of Requirements:** Based on the problem analysis, we established a set of requirements for the artifact, focusing on methods to compare behavioral traces between the physical and digital twins. These requirements guided the design of a solution that ensures accurate trace alignment and validation.
3. **Artifact Design and Development:** We designed and developed a trace alignment tool based on bioinformatics techniques to meet the defined requirements. Specifically, we adapted the Needleman-Wunsch global alignment algorithm and incorporated techniques from the BLAST algorithm, which are traditionally used for sequence alignment in biological research.
4. **Artifact Evaluation:** To demonstrate the artifact's feasibility and potential, we applied it to four different cyber-physical systems.

In the early stages of the project, we used a **case study** to investigate the problem and define the requirements. Case studies are ideal for exploring complex phenomena in their natural context and helped us identify specific issues in the alignment and fidelity measurement of DTSs. For this purpose, we selected an elevator system studied in its ordinary operational environment without manipulation.

For the later stages, including artifact validation and demonstration, we combined case studies with **experiments**. Experiments were essential for empirically testing the cause-and-effect relationships between different configuration parameters and the artifact's performance. By manipulating these parameters in a controlled environment, we could

evaluate the proposed solution's robustness and adaptability.

Our approach exemplifies **Exaptation** in Design Science, where known solutions are extended to address new problems. By leveraging bioinformatics alignment methods, typically used for biological sequences, we successfully adapted these techniques to align behavioral traces in DTSs.

Problem Formulation. In Design Science, the problem formulation must be precise and justified, showing the study's significance for a practice or group (Johannesson and Perjons 2014). To this end, the research question, also called the technical research problem, can be formulated according to the DS template proposed by Wieringa (2014).

Using the DS template (Wieringa 2014), the goal of our study is:

- To improve the validation of Digital Twin Systems
- by designing an algorithm to compare the behavior of two twins of supposedly equivalent behavior
- which provides indicators to reason about the level of fidelity of the two twins
- to determine whether two twins are sufficiently faithful to each other for the purpose they were conceived.

1.5 Context of the Thesis

This thesis was supported by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under the FPI scholarship granted by the CoSCA Project:

- *CoSCA: Digital Avatars: A Framework for Collaborative Social Computing Applications* (PGC2018-094905-B-I00) Ministerio de Ciencia, Investigación y Universidades, 2019-2022. Investigadora predoctoral.

Furthermore, the doctoral candidate also participated in three other research projects:

- *IPSCA: Including people in smart city applications.* (PID2021-125527NB-I00) Ministerio de Ciencia, Investigación y Universidades, 2022-2026. Participación: Investigadora predoctoral.
- *SoCUS: Social Computing for Urban Sustainability* (TED2021-130523B-I00). Ministerio de Ciencia, Investigación y Universidades, 2022 - 2024. Investigadora predoctoral.
- *MBT-I4A: Automated Model-Based Testing of Industry 4.0 Applications.* Junta de Andalucía (PY20_00067). 2021-2023. Investigadora predoctoral.

1.6 Outline of the Thesis

Following this, the thesis is divided into eight more chapters:

- **Chapter 2** is divided into two subsections. The first subsection introduces the main concepts and terminology in the state-of-the-art digital twin conceptualization, including the main definitions and architectures. The second subsection introduces a set of basic definitions about fidelity, as well as the trace alignment algorithms that inspired our validation proposal.
- **Chapter 3** focuses on the concept of the digital twin. It introduces the different components of the architecture, provides definitions for each part, and presents the overall modular architecture.
- **Chapter 4** presents our proposal for measuring the fidelity between the digital and physical twins. It provides a detailed explanation of the algorithm, analyzing its different parts and input parameters through the running example of an elevator.
- **Chapter 5** explores the proposal's application to three additional real-world systems: an incubator, a robotic arm, and a Lego Mindstorms car. Each example aims to showcase specific proposal features that are not presented in the running example.
- **Chapter 6** contains a two-part empirical evaluation. First, we compare our proposal to two other state-of-the-art proposals. Then, we assess the proposal's performance using traces of varying lengths to study its time performance and scalability.
- **Chapter 7** summarizes how to use our proposal with our open-access tool.
- **Chapter 8** outlines various related works in the field of validating DTs, such as similarity measurements and other approaches for analyzing traces.
- **Chapter 9** concludes with our future plans for expanding the validation approach and the conclusions of the thesis.

1.7 Scientific Contribution

1.7.1 Thesis Publications

The outcome of this thesis is published in two journal papers, two international and four national conference papers and several workshop papers:

- International journals
    1. **P. Muñoz,** M. Wimmer, J. Troya and A. Vallecillo, "Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments" in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2024.3462978 (2024)

2. C. J. Fernández-Candel, **P. Muñoz**, J. Troya, A. Vallecillo. "UTypes: A library for uncertain datatypes in Python" SoftwareX 26:101676 (2024)

- International Conferences

1. **P. Muñoz**, J. Troya, and A. Vallecillo. "Towards Measuring Digital Twins Fidelity at Runtime". In Proceedings of the 1st International Conference on Engineering Digital Twins (EDTconf 2024) at ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24). ACM, 507–512. 2024.
2. J. A. Llopis, J. Criado, L. Iribarne, **P. Muñoz**, J. Troya, and A. Vallecillo: "Modeling and Synchronizing Digital Twin Environments". Proceedings of the 2023 Annual Modeling and Simulation Conference (ANNSIM), Hamilton, ON, Canada, 2023 pp. 245-257
3. **P. Muñoz**, P. Karkhanis, M. van der Brand, and A. Vallecillo. "Modeling Objects with Uncertain Behaviors". Proceedings of the 17th European Conference on Modelling Foundations and Applications (ECMFA 2021). J. Object Technol. 20(3): 8:1-16 (2021)

- National Conferences

1. **P. Muñoz**, and A. Vallecillo: "What is a Digital Twin? An Architecture Proposal for its Implementation". Proceedings of the XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023).
2. **P. Muñoz**, A. Arrieta, and A. Vallecillo: "Trace Alignment of Twins. The Elevator Case Study". Proceedings of the XXVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2023). Sistedes (2023).
3. **P. Muñoz**, J. Troya, and A. Vallecillo: "Conformance Analysis of Multi-Fidelity Digital Twins". Proceedings of the XXVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2022). Sistedes (2022).
4. J. Díaz-Gaspar, **P. Muñoz**, and A. Vallecillo: "Automatic Generation of Exhaustive Tests from Models with Contracts". Proceedings of the XXV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2021). Sistedes (2021).

- International workshops

1. **P. Muñoz**, J. Troya, and A. Vallecillo: "A conceptual architecture for building digital twins". Post Proceedings of the STAF 2023 Workshops TTC 2023, MeSS 2023 and AgileMDE 2023, Leicester, United Kingdom, July 18, 2023 and June 21, 2023, CEUR Workshop Proceedings, vol. 3620. CEUR-WS.org (2023)

2. **P. Muñoz,** M. Wimmer, J. Troya, and A. Vallecillo. "Using trace alignments for measuring the similarity between a physical and its digital twin". 2nd International Workshop on Model-Driven Engineering of Digital Twins (ModDit'22). In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '22). ACM, 503-510 (2022).
3. **P. Muñoz.** "How alike are my physical and digital twins?". ACM Student Research Competition (Winner). In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '22). ACM, 201–204 (2022).
4. **P. Muñoz.** "Measuring the fidelity of digital twin systems". Doctoral Symposium. In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '22). ACM, 182-188 (2022).
5. D. Pérez-Porras, **P. Muñoz,** J. Troya, and A. Vallecillo. "Key-Value vs Graph-based data lakes for realizing Digital Twin systems (Poster)". MeSS'22: 2nd International Workshop on MDE for Smart IoT Systems. STAF 2022 Workshop Proceedings (2022).
6. **P. Muñoz,** J. Troya, and A. Vallecillo. "Using UML and OCL Models to Realize High-Level Digital Twins". ModDit'21: 1st International Workshop on Model-Driven Engineering of Digital Twins. ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Fukuoka, Japan, 2021, pp. 212-220 (2021).

1.7.2 Other Publications

During the thesis period we also worked on some orthogonal topics, which resulted in four journal papers:

- International journals

1. J. Deantoni, **P. Muñoz**, C. Gomes, C. Verbrugge, R. Mittal, R. Heinrich, S. Bellis, A.Vallecillo, "Quantifying and combining uncertainty for improving the behavior of Digital Twin Systems". at – Automatisierungstechnik (2024). Accepted.
2. L. Burgueño, **P. Muñoz**, R. Clarisó, J. Cabot, S. Gérard, and A. Vallecillo: "Dealing with Belief Uncertainty in Domain Models". ACM Trans. Softw. Eng. Methodol. 32(2): 31:1-31:34 (2023)
3. **P. Muñoz**, J. Troya, M. Wimmer, and G. Kappel: "Revisiting Fault Localization

Techniques for Model Transformations: Towards A Hybrid Approach". J. Object Technol. 21(4): 4:1-17 (2022)

4. **P. Muñoz**, L. Burgueño, V. Ortiz, and A. Vallecillo: "Extending OCL with Subjective Logic". J. Object Technol. 19(3): 3:1-15 (2020)

- International Conferences

1. **P. Muñoz**, A. Pérez-Vereda, N. Moreno, J. Troya, and A. Vallecillo. "Incorporating Trust into Collaborative Social Computing Applications". IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC 2021): 21-30.

- International workshops

1. **P. Muñoz**, L. Burgueño, A. Vallecillo, and M. Gogolla. "Automatic Generation of Valid Behavioral Scripts from UML Sequence Diagrams". 19th International Workshop in OCL and Textual Modeling (OCL 2019) co-located with IEEE/ACM 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS 2019), Munich, Germany, September 16, 2019: 81-96

2

BACKGROUND

This chapter provides the context and key terminology related to this thesis' contribution. The first section covers the main concepts and terminology in state-of-the-art Digital Twin conceptualizations, focusing on definitions and architectures. The second section presents definitions related to fidelity and introduces the trace alignment algorithms that inspired our validation approach.

2.1 Digital Twins: Concepts and Terminology

In this section, we review various definitions of the term *Digital Twin* as found in the literature. We then highlight some of the most relevant architectures used to realize them.

2.1.1 Digital Twin Definition

The concept of Digital Twin (DT) involves *replicating* an existing system. This idea can be traced back to NASA's Apollo mission (Grieves and Vickers 2017), where engineers created replicas of systems sent into space to assist astronauts from Earth. These *physical* replicas allowed ground engineers to evaluate optimal actions, ensuring astronaut safety and preserving the integrity of the spacecraft. The synchronization between the *Earth* and the *Space* replicas was achieved manually. As the potential value and benefits of these techniques became more evident, efforts were made to digitize the replicas using simulation or analytical models, such as differential equations, and the synchronization between them was automated.

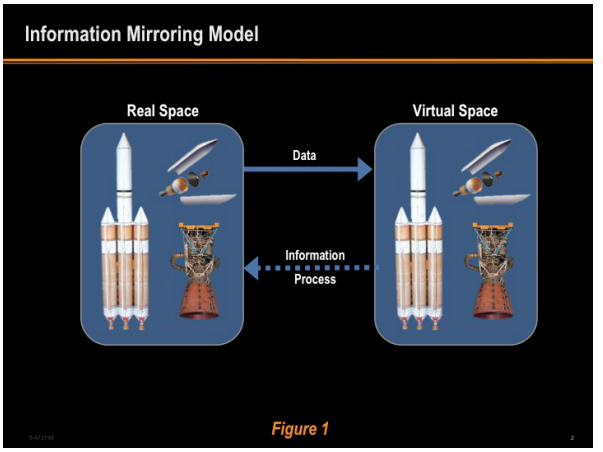


Figure 2.1: Digital Twin Concept Model by Michael Grieves (2014).

The first mention of *Digital Twin* in the literature is typically attributed to Michael Grieves's white paper (Grieves 2014). The concept of the Digital Twin was introduced in 2003 at the University of Michigan in an Executive Course he taught on Product Lifecycle Management (PLM). At that time, the virtual representations of the physical assets were starting to be developed and, therefore, were immature. The information retrieved from the physical systems was collected manually and was primarily paper-based. However, based on his experience, Grieves first defined the *Digital Twin Concept Model*, including Physical Products in Real Space, Virtual Products in Virtual Space, and the Connections of data and information that linked the products, as shown in Figure 2.1.

However, as pointed out by Kritzinger et al. (2018), not every system that includes a digital and physical counterpart can be considered a digital twin. It is essential to distinguish between the terms digital model, digital shadow, and digital twin, often referred to collectively as digital twin. The distinction between these concepts lies in the level of data integration, illustrated in Figure 2.2.

A ***Digital Model*** is a digital representation of an existing or planned physical object. It does not use any form of automated data exchange between the physical and the digital object (Kritzinger et al. 2018). The digital object may include a comprehensive description of different aspects of the physical object, such as appearance or behavior, through mathematical models or simulations. However, any changes occurring in the real world must be updated manually.

A ***Digital Shadow*** (DS) is a *digital model* with an automated one-way data flow from the physical to the digital object (Kritzinger et al. 2018). A Digital Shadow is automatically updated with any changes regarding any change in the state of the relevant

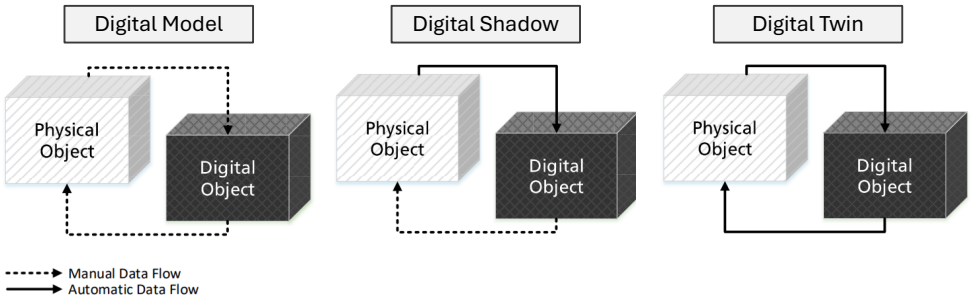


Figure 2.2: Digital Twins according to their level of integration by Kritzinger et al. (2018).

properties of the physical object. This allows for constant monitoring of the system state. However, it does not enable any self-adaptive behavior automatically triggered by the digital system. Instead, a manual action would be required to change the state of the physical asset after receiving any relevant information from the digital object.

Finally, a ***Digital Twin*** (DT) is a digital representation of the physical object that integrates an automatic bidirectional data flow (Kritzinger et al. 2018). In this setup, the digital object is not only influenced by changes in the physical object but also controls the state of its physical counterpart. The bidirectional communication in the DT paradigm distinguishes it from other approaches like system simulation or dashboards. This feature unlocks the main potential of the DT paradigm, which is to create a self-adaptive system continuously monitored by a virtual replica. This allows predictions to prevent potentially unsafe states, perform maintenance, and optimize its behavior (Dalibor et al. 2022).

An issue we observe in most works is the use of vague definitions that do not include all these elements or do not explicitly state the communication mechanism between the different counterparts (Dalibor et al. 2022). In our work, we adhere to the definition provided by the Digital Twin Consortium (2021):

> *A* Digital Twin *is a virtual representation of a real-world entity or process (the so-called* Physical Twin*, PT) synchronized at a specified frequency and fidelity.*

In this definition, all the constraints imposed by Kritzinger et al. (2018) are fulfilled, along with two additional essential points: synchronization and fidelity. First, the counterparts must exchange information to synchronize their state periodically. In the case of real-time requirements, this ***synchronization*** process can be costly, especially with a high sampling frequency (Modoni et al. 2019). Therefore, the synchronization period must be adapted to the system's requirements and the required level of precision for the virtual replica.

For example, consider a digital twin of a robotic car intended to predict maintenance needs. In this case, synchronization occurs daily because wear and tear cannot be detected

at a higher frequency. Conversely, if the purpose of the digital twin is to monitor the car's trajectory, a sampling frequency of under one second would be necessary to enable accurate prediction and monitoring of the car's movements.

Fidelity, on the other hand, describes the accuracy with which the DT represents the PT. The DT is a virtual replica that should mimic the required characteristics of the PT. Since the model cannot perfectly replicate an actual system, the replica should be created with a specific level of fidelity. This will be a central focus in the second part of this document (see chapter 4), as our tool offers a method for measuring the fidelity of the DT.

2.1.2 Digital Twin Architectures

To realize DTs in practice, several authors have proposed various conceptual architectures. Grieves et al. initially defined an architecture of three main components: the virtual object, the physical object, and the connections between them (Grieves 2014; Grieves and Vickers 2017), as discussed in the previous section. Subsequent proposals added two more dimensions to derive further capabilities from DTs (Dalibor et al. 2020; Tao et al. 2018, 2019). The architecture proposed by Tao et al. is depicted in Figure 2.3. The first additional dimension is *Digital Twin Data* (DD), which includes domain knowledge, data from the DT and PT, and their fusion. The second dimension is *Services* (Ss), which includes functionalities such as prediction services, behavior optimization, and dashboards for visualizing the state and health of both systems.

Our work aligns with the Digital Consortium's definition of DT (c.f. Section 2.1.1), which defines DT as solely the virtual replica of its physical counterpart, excluding other elements such as services or connections. The 5-dimension proposal by Tao et al. (2018; 2019) considers the DT as the combination of all five dimensions rather than identifying it as the virtual replica alone. In (Dalibor et al. 2020), the DT is separated from the data and connections but includes the optimization services, which hinders the addition of new services or the modular modification of existing services.

The ISO architecture for DTs in manufacturing (ISO 23247:2021 2021) comprises five main elements, which could be partially mapped to the proposal by Tao et al. with some specific differences. The Architecture consists of the DT Framework, directly connected to the Observable Manufacturing Elements (OMEs), the physical counterpart. The DT Framework includes the DT Entity, encompassing the virtual replica and various analysis and maintenance services. Additionally, there is the Device Communication entity, which manages the communication between the physical and virtual counterparts and stores the data. Lastly, there is the User Entity, which serves as a dashboard for the manufacturing

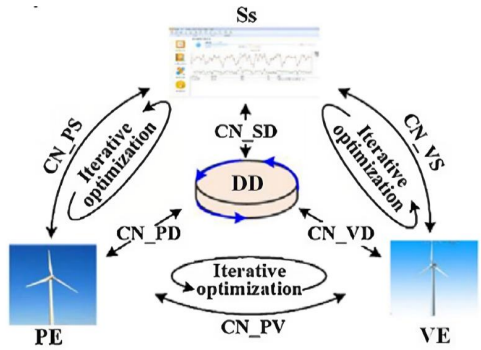


Figure 2.3: Five-dimension architecture by Tao et al. (2018).

operators to visualize the system's state, and the Cross-System Entity, responsible for securing connections and translating data within the DT Framework.

This architecture is at a lower level of abstraction and includes more details about the specific elements needed for a manufacturing DT based on domain knowledge. However, the architecture still follows the concept of having two connected counterparts with bidirectional communications, including a set of services and a data-focused element to enable the usage of historical data.

In a systematic cross-domain mapping study on DTs (Dalibor et al. 2022), over 300 publications in the field were analyzed. One of their research questions focused on the different components that constitute the architecture of a DT. They examined whether DTs included data, hardware components, models (describing structure, behavior, appearance, and constraints), and software components. The study revealed that approximately 77% of the publications considered models as part of the DT, 36% considered data, and 40% included software, primarily for self-adaptive behavior and monitoring. This suggests that 77% of the publications identified the DT with the virtual replica. However, it is also worth noting that around 40% of the publications considered the inclusion of services and data within the DT. Despite this, the tendency to identify the replica as the DT remains more prevalent.

2.2 Fidelity Assessment

In this section, we will briefly discuss the terminology and central concepts that form the basis of our proposal for evaluating the fidelity of DTs. Firstly, in sections 2.2.1 to 2.2.3, we define the central concepts: fidelity, model, trace, and snapshot, among others. Section 2.2.4 presents the formal definitions of the alignment elements. Section 2.2.5 briefly introduces the two sequence alignment algorithms that inspired our proposal. Finally, in Section 2.2.6, we introduce the distance measure that will be used as a fidelity metric.

2.2.1 Fidelity, Abstraction, and Resolution

DTs are applied in different domains including information and communication, mining, human health, and transportation, among others. However, manufacturing stands out as the most common application domain for this type of systems (Dalibor et al. 2022). In all these areas, the physical asset is a complex system with a large set of attributes defining its state. However, depending on the purpose of the DTS, the model should focus on replicating different parts of the PT.

Abstraction is one of the most common mechanisms used to tame complexity by focusing on the important parts of a system and ignoring the rest. Likewise, adjusting the *resolution* of a model, i.e., its level of detail, helps to reduce the costs and complexity significantly (Wang, Kirby, and Johnson 2010). The abstraction and resolution of a model have traditionally been considered to determine its *fidelity*, which is the utility pursued by model users: the ability to emulate complex systems effectively to answer questions with minimal computational costs (Madni, Madni, and Lucero 2019).

The main focus of this thesis is the analysis of a DT behavioral trace at the appropriate level of abstraction to determine whether the behavior showcased is at the required level of fidelity, i.e., whether the abstraction and resolution level of the replica's model are enough to effectively replicate the behavior of interest of its physical counterpart. In the following subsections, we introduce the definitions of these concepts.

2.2.1.1 Model

A definition of model relevant to our context is given in (Rothenberg 1989): "*Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer, or cheaper than reality instead of reality for some purpose. A model represents reality for a given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger, and irreversibility of reality*". Furthermore, the definition by D.T. Ross and M. Minski in 1960 is also appropriate: "M *is a model of* S *if* M *can be used to answer questions about* S" (ISO/IEC 42010 2011).

2.2.1.2 Abstraction

Abstraction is "the process of selecting the essential aspects of a system to be represented in a model or simulation while ignoring those aspects that are not relevant to its purpose" (Gross 1999).

The different definitions of this concept in the literature agree that elements are eliminated or hidden during the abstraction process, resulting in a simpler model (ISO/IEC IS 10746 1998–2010; Manna et al. 1997), which allows a faster analysis or a lighter execution of a simulation, optimizing computational resources (Arrieta 2021). In general, the level of abstraction of a model increases as the amount of information it contains decreases (Moon and Hong 2013). For an abstraction to be *sound*, the properties of interest, i.e., those relevant to the execution of the system, that are true for the abstraction must be true for the original system (Lee and Sirjani 2018).

Although abstraction is a useful process to tame complexity, if we eliminate too many elements, specific system properties may no longer hold, or we may not even be able to check them (Manna et al. 1997). Therefore, it is essential to reach a compromise between the level of abstraction and the soundness of the model. For example, let us suppose we want to build a DT of an autonomous moving robot, predicting its movements to warn the physical robot of possible collisions. In this case, all aspects directly related to the motion of the robot will be selected in the DT models, as these are the properties of interest. Other aspects, e.g., energy consumption, could be abstracted since they do not seem relevant to the problem. While this could help reduce the complexity of the simulations, we may miss important behaviors such as those that occur when the robot is running out of battery and moves much slower or simply stops moving.

Given two models, A and B, of a system S at different levels of abstraction (e.g., A is more abstract than B), an *abstraction function* defined between B and A is a function that maps the elements of the more concrete model B into elements of the more abstract model A. It explains how to interpret each element of the concrete model relevant to the user in terms of elements in the abstract model (Hoare 1972; Lamport 1983). Abstraction functions are helpful in bringing models—or specific elements of models—to the same level of abstraction to enable comparison between them. We will later use abstraction functions to compare the execution traces of two models at different abstraction levels.

2.2.1.3 Resolution

Resolution is defined as "the degree of detail used to represent certain aspects of the real world in a model" (Moon and Hong 2013). The purpose of resolution is to determine *how* these aspects of the system will be modeled—in contrast to abstraction, which consists in determining the elements that will be included in the model.

To illustrate the concept, let us consider the digital twin of a moving robot. One of the parameters we need to decide is the simulation time step. A smaller time step results in more precise simulations and almost instantaneous decisions, while a larger time step

leads to less accurate simulations with slower reactions, resulting in coarser and rougher movements that are less faithful to the actual system.

To compare the resolution of the two models, they must be at the same level of abstraction (Moon and Hong 2013). This means comparing their equivalent elements. Models with more detail have higher resolution. For example, a simulation model with a time step of milliseconds has a higher resolution than one with a time step of seconds. Similarly, decreasing the mesh size in a finite element analysis algorithm increases its resolution.

Higher resolution generally leads to more accurate solutions, and resolution and accuracy are often used as fidelity measures in the simulation domain (Gross 1999). However, increasing resolution also increases computational costs and resource consumption and decreases performance. It is important to find the optimal resolution that maximizes accuracy and model fidelity without incurring unnecessary costs.

2.2.1.4 Fidelity

Fidelity is "the degree to which a model reproduces the real state and behavior of a system in a measurable way" (Gross 1999). It determines how closely a model realistically represents the actual system.

Measuring the level of fidelity of the model with respect to the existing system is essential to determining whether the model fits the purpose for which it was designed. If the level of fidelity is too low, the model will not be helpful because it does not faithfully represent the system. However, if it is too high, we may be wasting time and computational resources.

It is technically impossible to achieve a 100% accurate representation of all aspects of the real-world system. It is not feasible to build a simulation with a level of fidelity that makes it indistinguishable from reality, taking into account measurement errors (Lee and Sirjani 2018). Just like having a map of a city at a 1:1 scale is unfeasible and too expensive (Bézivin 2005). To achieve affordability or performance objectives, almost all models have a lower accuracy and resolution of representation of those aspects of the real world that are less relevant to achieving the simulation objectives.

Therefore, it is important to assist the developer in defining the appropriate level of fidelity to emulate the system with enough realism for our purposes and with minimal computational costs. This level of fidelity will, in turn, influence the required level of abstraction of the model and the resolution of its elements.

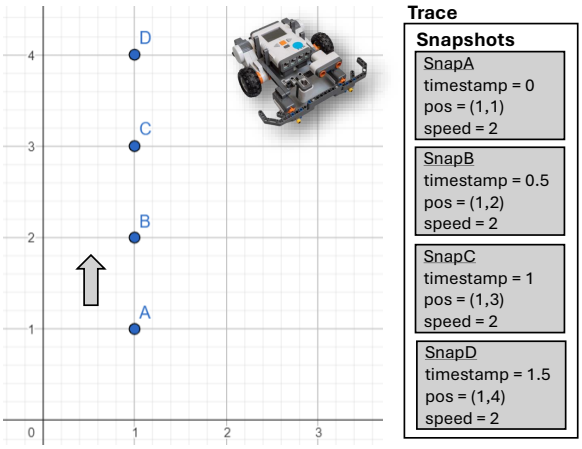


Figure 2.4: Snapshots and the trace of a robotic car.

2.2.2 Engineering and Scientific Models

When discussing DTs, another important consideration is the distinction between engineering and scientific models. **Engineering models** are used to specify the behavior of a system that is being built, guiding its development. In contrast, **scientific models** are designed to replicate the behavior of an existing system (Lee and Sirjani 2018).

DTs for existing systems are essentially scientific models. These models do not need to be completely faithful to the original; they just need to be accurate enough to remain useful. A scientific model is considered accurate if every property that exists in the model is also true in the real system. Similarly, an artifact built from an engineering model will not behave differently than specified but will be similar enough to be useful.

This thesis provides a tool to assess the fidelity of scientific and engineering models' behavior and determine whether they are similar enough to their reference counterparts.

2.2.3 Behavioral Models: Traces and Snapshots

In the definition of *fidelity* (see Section 2.2.1.4), we state that it is the degree to which a model accurately replicates the actual state and behavior of the system in a measurable way. The system's state, at a given point in time, will be defined by the value of its attributes at that time. In turn, the system's behavior refers to the evolution of its states over time. We represent this evolution using traces, which are also called trajectories. Traces consist of snapshots taken at regular intervals. *Snapshots* are defined as a set of objects, a set of links between them, and the specific values of the attributes of these objects at a specific moment in time (Gogolla, Bohling, and Richters 2005).

In Figure 2.4, we have a set of four snapshots describing the position and speed of a robotic car that moves in a straight line. These four snapshots compose a trace that describes the behavior of the robotic car during 1.5 seconds (note how the "timestamp" attribute evolves). These snapshots only include speed and position as the properties of interest since our DT's purpose is optimizing the car's trajectory. The car also includes other sensors to measure distance and detect collisions, but in this case, we focus only on this information.

In our work, we assume that all snapshots are presented at the same level of abstraction. In other words, they have the same attributes and are of the same types. If this is not the case, the user can create mappings from one abstraction level to another and make the necessary transformations. For example, if the car's DT does not directly provide the car's speed but only the position, we could estimate the speed by calculating the difference in position between different snapshots.

2.2.4 Trace Alignment Concepts

To accurately introduce all concepts related to trace alignments, we have developed a set of formal definitions of all elements and metrics involved in the alignments.

Definition 2.2.1 (Comparison Function)**.** Let X and Y be sets of snapshots. A *comparison function* "$\approx$" between two snapshots is defined as:

$$\approx: X \times Y \to \mathbb{B}$$

where $\mathbb{B} = \{\text{true}, \text{false}\}$. The function assigns a boolean value to each pair of elements $(x, y) \in X \times Y$. The value true indicates that the snapshots are sufficiently similar to be considered a match, otherwise the function evaluates to false.

This comparison function is normally user-defined and depends on the application. In Section 4.4, this function is generalized to the so-called *similarity function*, which returns a real value in the range [0,1] that describes the degree of similarity between two snapshots, where 1 means they are identical, and 0 means they are completely different.

Definition 2.2.2 (Aligment)**.** Given two sequences of snapshots $X = \{x_i\}_{i=1}^{n}$ and $Y = \{y_i\}_{i=1}^{m}$, and a comparison function between snapshots "$\approx$", an *alignment* A between X and Y is a set of pairs $A \subseteq \{0..n\} \times \{0..m\}$, that satisfies the following properties (we assume below that i and j will always range between $\{1..n\}$ and $\{1..m\}$, respectively).

- All elements of X are paired with at most one element of Y, i.e., $(i, j_1) \in A \wedge (i, j_2) \in A \Rightarrow j_1 = j_2$.

- All elements of Y are paired with at most one element of X, i.e., $(i_1, j) \in A \wedge (i_2, j) \in A \Rightarrow i_1 = i_2$.
- The set of pairs A must be monotonic, i.e., it always looks ahead: if $(i_1, j_1) \in A \wedge (i_2, j_2) \in A$ then $i_1 \leq i_2 \Rightarrow j_1 \leq j_2$ and vice-versa: $j_1 \leq j_2 \Rightarrow i_1 \leq i_2$.

The alignment may also contain *gaps*, which are the indices of the elements that have not been paired. We use the number '0' to represent a gap in the alignment, i.e.,

- $(i, 0) \in A \Leftrightarrow x_i \in X \wedge \nexists\, j \in \{1..m\} : (i, j) \in A$
- $(0, j) \in A \Leftrightarrow y_j \in Y \wedge \nexists\, i \in \{1..n\} : (i, j) \in A$
- $(0, 0) \notin A$

In the following, G_A will denote the set of gaps of alignment A, i.e., $G_A = \{(i, 0) \in A\} \cup \{(0, j) \in A\}$.

Note that with this definition, A contains one $(i, *)$ and one $(*, j)$ element for each with $1 \leq i \leq n$ and $1 \leq j \leq m$ (where the asterisk denotes an arbitrary value).

Definition 2.2.3 (Match and mismatch)**.** Given an alignment A, we say that a pair $(i, j) \in A$ is a *match* if $x_i \approx y_j$, and a *mismatch* if $x_i \not\approx y_j$.

Note that more than one alignment can be defined between two sequences. For example, let us suppose that $X = \{a, b, c, d\}$, $Y = \{a, b, d\}$, and $Z = \{a, h, c, d\}$. The following alignments can be defined between them:

$A_1(X, Y) = \{(1, 1), (2, 2), (3, 0), (4, 3)\}$

$A_2(X, Z) = \{(1, 1), (2, 2), (3, 3), (4, 4)\}$

$A_3(X, Z) = \{(1, 1), (2, 0), (0, 2), (3, 3), (4, 4)\}$

The first one contains three matches and one gap, the second one contains three matches and one mismatch $(2, 2)$, and the third one contains three matches and two gaps.

Definition 2.2.4 (Paired and matched subsequences)**.** We will denote by X^A and Y^A the subsequences of X and Y that are *paired* by an alignment A with other elements (i.e., the non-gaps):

$X^A = \{x_i \mid x_i \in X \wedge \exists j \in \{1..m\} : (i, j) \in A\}$

$Y^A = \{y_j \mid y_j \in Y \wedge \exists i \in \{1..n\} : (i, j) \in A\}$.

Similarly, X^{A+} and Y^{A+} are the subsequences of X^A and Y^A that are *paired* and *matched* by the alignment:

$X^{A+} = \{x_i \mid x_i \in X \wedge \exists j \in \{1..m\} : (i, j) \in A \wedge x_i \approx y_j\}$

$$Y^{A+} = \{y_j \mid y_j \in Y \wedge \exists i \in \{1..n\} : (i,j) \in A \wedge x_i \approx y_j\}.$$

Then, the *mismatched* subsequences are defined as follows: $X^{A-} = X^A - X^{A+}$, and $Y^{A-} = Y^A - Y^{A+}$. Finally, the *gap* subsequences are those whose elements are matched with gaps: $G_X^A = X - X^A$, and $G_Y^A = Y - Y^A$.

Definition 2.2.5 (Degree of matching)**.** Given an alignment A between two traces X and Y, we can define its *degree of matching* (m_A) with respect to each trace as follows:

$$m_A(X) = \#X^{A+}/\#X$$

$$m_A(Y) = \#Y^{A+}/\#Y$$

where "#" denotes the number of sequence elements. Both degrees are real numbers between 0 and 1.

Definition 2.2.6 (Sequence of consecutive gaps)**.** Given an aligment A, a subset S of G_A is a *sequence of consecutive gaps* (SCG) if the elements of S fulfill the following properties:

- $\forall i \in \{i_b..i_e\} : (i,0) \in S$
- $\forall j \in \{j_b..j_e\} : (0,j) \in S$
- $i_b > 1 \wedge j_b > 1 \Rightarrow (i_b - 1, j_b - 1) \in A$
- $i_e < n \wedge j_e < m \Rightarrow (i_e + 1, j_e + 1) \in A$

where $i_b = \min\{i \mid (i,0) \in S\}$, $i_e = \max\{i \mid (i,0) \in S\}$, $j_b = \min\{j \mid (0,j) \in S\}$, and $j_e = \max\{j \mid (0,j) \in S\}$.

The size of the set S gives the length of a sequence of consecutive gaps S.

Intuitively, $[i_b, i_e]$ defines the range of indices of X in S, and $[j_b, j_e]$ defines the range of indices of Y in S. The first two conditions require that paired elements bound the sequence S, i.e., the pair of A just before the first element of S is either a match or a mismatch, and the pair of A right after the last element of S is either a match or a mismatch in A, too. The last two conditions require that the pairs of S have consecutive indices, i.e., there are no paired elements in all the range of indices of S.

Example 1. Suppose that $X = \{a, b, c, d, e, f, g\}$, $Y = \{a, m, e, g\}$ and that the alignment A is defined as follows:

$$A = \{(1,1), (2,0), (3,0), (0,2), (4,0), (5,3), (6,0), (7,4)\}$$

In this case, only three points have been matched, namely $\{(1,1), (5,3), (7,4)\}$, and the rest have been identified as gaps, i.e., $G_A = \{(2,0), (3,0), (0,2), (4,0), (6,0)\}$.

Then, we can identify two sequences of consecutive gaps: $S_1 = \{(2,0),(3,0),(0,2),(4,0)\}$ and $S_2 = \{(6,0)\}$, with respective lengths of 4 and 1.

Should we prefer to represent the alignment by the elements of X and Y, the sequences of consecutive gaps correspond to $S_1 = \{(b,-),(c,-),(-,m),(d,-)\}$ and $S_2 = \{(f,-)\}$, while the matched sequence is $\{a,e,g\}$.

Note that, in general, representing alignments using the elements of the sequences to be matched is not appropriate, especially since they may contain repeated elements. This would make it impossible to distinguish the specific matches and, thus, the respective gap sequences. This is why the alignments are always specified by the indices of the elements of the respective sequences and not by the elements themselves.

Note as well that three alternative alignments could have been defined:

$A' = \{(1,1),(2,2),(3,0),(4,0),(5,3),(6,0),(7,4)\}$

$A'' = \{(1,1),(2,0),(3,2),(4,0),(5,3),(6,0),(7,4)\}$

$A''' = \{(1,1),(2,0),(3,0),(4,2),(5,3),(6,0),(7,4)\}$

They pair respectively element m in Y with b, c and d in X, identifying them as mismatches. Their corresponding SCGs are the following:

$S'_1 = \{(3,0),(4,0)\}$ and $S'_2 = \{(6,0),(7,0)\}$, both of length 2.

$S''_1 = \{(2,0)\}$, $S''_2 = \{(4,0)\}$ and $S''_3 = \{(6,0),(7,0)\}$, of lengths 1, 1 and 2.

$S'''_1 = \{(2,0),(3,0)\}$ and $S'''_2 = \{(6,0),(7,0)\}$, both of length 2.

Thus, the length of the SCGs may vary depending on how the alignment pairs the elements and determines the corresponding gaps. The next example further illustrates this.

Example 2. Suppose now that $X = \{a,a,a,b,b,b,b,b\}$ and $Y = \{a,b\}$. In this case, depending on the strategy of the alignment algorithm to determine the gaps, we can have alignments with rather different sequences of consecutive gaps. For example, one alignment matches the elements in the extremes of sequence X, obtaining only one SCG of length 6. Another alignment pairs x_4 with y_1 and x_5 with y_2, obtaining two SCGs of lengths 2 and 4. An alternative alignment has three SCGs of lengths 1, 3, and 2. There are 12 possible alignments, each with resulting SCGs of different lengths.

The decision of whether we prefer more SCGs of shorter length or fewer but longer SCGs depends on the alignment algorithm's strategy for opening a gap or continuing to explore the trace.

2.2.5 Trace Alignment Algorithms

To align traces, we explored existing solutions for sequence alignment, a widely studied problem in areas like bioinformatics. In these areas, efficient algorithms have been developed to align DNA and other biological sequences. To align behavioral traces, we adopted a combination of established techniques. Firstly, we used the renowned global alignment algorithm called Needleman-Wunsch (NDW) (Needleman and Wunsch 1970) as the basis for our algorithm. Additionally, we integrated techniques derived from the widely used BLAST algorithm (Altschul et al. 1990), which is used to search databases of biological sequences for statistically significant similarities. These techniques will be elaborated on in the subsequent sections.

2.2.5.1 Global and Local Alignment Algorithms

The literature divides sequence alignment algorithms into two categories: *global* and *local alignment*.

The **global alignment** approach forces an alignment of the entire length of the longest sequence and attempts to align every character in every sequence ("Sequence Alignment (I)" 2021). This approach is usually applied when sequences are expected to be similar and approximately the same length. The Needleman-Wunsch (NDW) algorithm (Needleman and Wunsch 1970) is an example of a general global alignment algorithm.

For example, let us have the sequences:

$$S_1 = \{G, T, C, G, A, C, G, C, A\}$$
$$S_2 = \{G, A, T, T, A, C, A\}$$

A possible global alignment between both, using the NDW algorithm, is:

$$A_0 = \{(G, G), (T, A), (C, T), (G, T), (A, A), (C, C), (G, -), (C, -), (A, A)\}$$

This alignment represents the optimal alignment of all elements in both sequences, considering a scoring schema that penalizes gaps over mismatches. It includes three mismatches, two gaps, and four matches. Further details will be explored in the next section.

The **local alignment** approach identifies regions of similar subsequences within long sequences ("Sequence Alignment (I)" 2021). This approach is typically applied when dealing with dissimilar sequences that are expected to contain similar subsequences. An example of a general local alignment algorithm is the Smith-Waterman algorithm (Smith and Waterman 1981). Unlike the NDW algorithm, the result of the Smith-Waterman algorithm does not produce an alignment covering the entire length of the two sequences. Instead, it lists the most similar subsequences between the sequences.

Let us align sequences S_1 and S_2 using the Smith-Waterman algorithm. The result includes the alignments with the highest score, which are aligned subsequences from S_1 and S_2, respectively.

$$
\begin{aligned}
A_1 &= \{(G,G),(A,T),(T,C),(T,G),(A,A),(C,C)\} \\
A_2 &= \{(G,G),(A,T),(T,C),(T,G),(A,A),(C,C),(A,G)\} \\
A_3 &= \{(G,G),(A,A),(T,-),(T,C),(A,G),(C,C),(A,A)\} \\
A_4 &= \{(G,G),(A,A),(T,C),(T,-),(A,G),(C,C),(A,A)\} \\
A_5 &= \{(G,G),(A,A),(T,C),(T,G),(A,-),(C,C),(A,A)\}
\end{aligned}
$$

2.2.5.2 Needleman-Wunsch Algorithm

Overview. The *Needleman-Wunsch algorithm* (NDW) is a global alignment algorithm that finds *"the largest number of amino acids in one protein that can be matched with those of a second protein allowing for all possible interruptions in either of the sequences"* (Needleman and Wunsch 1970). Sequences of characters represent the two amino acid sequences. It is implemented using dynamic programming techniques, where the problem of comparing sequences is broken down into smaller sub-problems (comparing sets of subsequences) to find the optimal solution to the global problem, and it uses a similarity matrix to re-use calculations. The algorithm assigns a score to every possible alignment and tries to find one of the possible alignments with the highest score—there may be no unique optimal alignment.

Matching outcomes. To select the optimal alignment, the algorithm requires a scoring schema defined in terms of rewards and penalties assigned to the possible outcomes when comparing two characters. These outcomes are:

- **Match.** The characters are the same.
- **Mismatch.** The characters are not the same but are aligned, which is considered a mismatch.

- **Gap**. The characters are different, and the best alignment involves one letter paired to a gap (represented by "–") in the other sequence.

Scoring Schema. It determines the prevalence of the different matching outcomes in the optimal alignment. The two most common scoring schemas are:

- **Constant scoring schema.** Each result receives a specific score, such as a reward of $+1$ for a match, 0 for a mismatch, and a penalty of -1 for a gap. This schema prioritizes mismatches over gaps.
- **Frequency-based scoring schema.** Each potential character comparison includes a specific set of scores for each result. These scores are typically recorded in an input matrix that the algorithm uses for the alignment. Input matrices are the most commonly used scoring system in bioinformatics, as certain nucleotides or amino acids are more prone to mutation or substitution by others (Henikoff and Henikoff 1992). The values of the scores are obtained empirically.

Apart from these basic scoring schemas, we can customize new schemas based on our preferences regarding the prevalence of the different comparison outcomes. For example, following the *constant scoring schema*, we assign a reward of $+1$ for a match; 0 for a mismatch; and a penalty of -1 for a gap. This schema prioritizes mismatches over gaps. If we apply this schema to align the sequences $S_1 = \{G, T, C, G, A, C, G, C, A\}$ and $S_2 = \{G, A, T, T, A, C, A\}$, the resulting alignment is the same as in the previous section:

$$A_0 = \{(G, G), (T, A), (C, T), (G, T), (A, A), (C, C), (G, -), (C, -), (A, A)\}$$

However, if we use a reward of $+1$ for a Match; -1 for a Mismatch; and a penalty of 0 for a Gap, we have a schema that prioritizes gaps over mismatches, and the resulting alignment would be:

$$A'_0 = \{(G, G), (-, A), (T, T), (-, T), (C, -), (G, -), (A, A), (C, C), (G, -), (C, -), (A, A)\}$$

In the alignment A'_0, when the penalty for adding a gap is set to zero, we achieve optimal alignments that involve a gap for each character that cannot be aligned with a counterpart in the other sequence. In this scenario, it is improbable that the benefit of including a mismatch will outweigh the penalty in the overall score. This approach is just as valid as the previous one, but it is less commonly used in bioinformatics because the presence of mismatches could be seen as points of mutations. (Mount 2004).

Algorithm 1 Algorithm to calculate the similarity matrix for the Needleman-Wunsch algorithm (Needleman and Wunsch 1970)

```
1: M[0][0] ← 0
2: c ← 1
3: while c < M[0].size do                          ▷ Fill the first row of the matrix
4:     M[0][c] ← M[0][c − 1] + g
5:     c ← c + 1
6: end while
7: r ← 1
8: while r < M.size do
9:     M[r][0] ← M[r − 1][0] + g                   ▷ Fill first column
10:    while c < M[0].size do
11:        ins ← M[r][c − 1] + g
12:        sub ← M[r − 1][c − 1] + s(A[r − 1], B[c − 1])
13:        del ← M[r − 1][c] + g
14:        M[r][c] ← max{ins, sub, del}
15:        c ← c + 1
16:    end while
17: end while
```

Another schema rewards $+1$ for matches and imposes penalties of -1 and -2 for mismatches and gaps, respectively. This schema is associated with the edit distance, also known as the Levenshtein distance, between the two strings (Levenshtein 1966). The lower the alignment score, the greater the edit distance.

Implementation. The pseudocode to fill the dynamic programming matrix is listed in Algorithm 1, where M is the score matrix, and A and B are the character sequences to align. The function r is evaluated after comparing the characters and determining whether it is a match or a mismatch (line 12). The alignment is performed using a score matrix, which includes sequence A along the vertical axis and sequence B along the horizontal axis. Each cell in the matrix represents a pairing between a character of each sequence. To fill the matrix, the algorithm maximizes the score in each cell by choosing between the three outcomes.

Once the matrix is filled, the algorithm backtracks from the bottom right by selecting the highest scores. The algorithm's result is the alignment that maximizes the scores.

More precisely, if H is the maximum score matrix of the alignment of sequences A and B, the Bellman Equations for computing this matrix recursively are as follows:

$$H[i,j] = max\{H[i-1,j] + g, \\ H[i,j-1] + g, \\ H[i-1,j-1] + s(a_i, b_j)\} \qquad \text{(Eq. 2.2.5.1)}$$

where the initial conditions are $H[i,0] = H[0,j] = 0$; g is the gap penalty; and $s()$ is the function that compares two elements, decides if there is a match or a mismatch, and returns the corresponding reward.

Demonstration of Bellman's Principle of Optimality. The NDW algorithm must fulfill Bellman's Principle of Optimality to employ a dynamic programming approach. The principle can be summarized as follows: *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy concerning the state resulting from the first decision."* (Bellman 1957)

This principle implies that the optimal solution to the problem contains the optimal solutions of all the reduced versions of the same problem, allowing it to be solved in a nested manner. It also implies that, by knowing the optimal solution to these reduced versions, we can deduce the sequence of optimal decisions to reach that solution. The NDW algorithm can formulate the Global Sequence Alignment using subproblems, ensuring an optimal substructure and making it possible to solve the problem using dynamic programming.

(A) Characterize the Structure of an Optimal Solution. Given two sequences of characters $C = \{c_i\}_{i=1}^{n}$ and $D = \{d_j\}_{j=1}^{m}$ and a substitution matrix $m : C \times D \to \mathbf{Z}$ that determines the similarity score between two characters, an alignment A satisfies the properties described in Definition 2.2.2.

Each time a gap is added to the alignment, a constant gap penalty, $g < 0$, is added to the score. Our goal is to obtain the alignment that pairs the most similar characters according to the substitution matrix m. Then, the solution, $A = \{a_1, \ldots, a_p\}$, can be defined as a set of pairs, $a_i = (i_{a_i}, j_{a_i})$.

Assuming these definitions, the function that specifies the score of a given pair is the following:

$$r(a_i) = \begin{cases} m(a_i), & \text{if } i_{a_i} > 0 \wedge j_{a_i} > 0 \\ g & \text{if } i_{a_i} = 0 \vee j_{a_i} = 0 \end{cases} \qquad \text{(Eq. 2.2.5.2)}$$

To solve the complete problem, the algorithm must maximize $R(A) = \sum_{i=1}^{p} r(a_i)$, where p represents the number of pairs in the alignment A, i.e., $p = |A|$. A condition regarding the cardinality of A is that, in the worst case, if all the characters in both sequences were aligned with gaps, the cardinality of A would be, at most, the sum of the lengths of the aligned sequences C and D, i.e., $|A| \leq |C| + |D|$.

(B) Proof of the Optimal Substructure. Let us assume that we have two sequences of characters C and D and an optimal alignment of these two sequences named A^*. If our problem follows an optimal substructure, it must fulfill the condition that ${A^*}_{2 \to p} = \{a_2^*, ..., a_p^*\}$ is the optimal solution to align all the characters except those included in the pair a_1^*.

To arrive at a contradiction and demonstrate the optimality of A^*, let us assume that A^* is not the optimal solution, and there exists another alignment $A' = \{a_2', \ldots, a_p'\}$ that satisfies:

$$|A'| \leq |C \setminus \{i_{a_1^*}\}| + |D \setminus \{j_{a_1^*}\}| \qquad \text{(Eq. 2.2.5.3)}$$

$$\sum_{i=2}^{p} r(a_i') > \sum_{i=2}^{p} r(a_i^*) \qquad \text{(Eq. 2.2.5.4)}$$

If this were true, we could build $A'' = \{a_1^*, a_2', ..., a_p'\}$ that fulfills that:

$$|A'| + |\{a_1^*\}| \leq |C \setminus \{i_{a_1^*}\}| + |\{i_{a_1^*}\}| + |D \setminus \{j_{a_1^*}\}| + |\{j_{a_1^*}\}| \qquad \text{(Eq. 2.2.5.5)}$$

$$\sum_{i=2}^{p} r(a_i'') = r(a_1^*) + \sum_{i=2}^{p} r(a_i') > \sum_{i=1}^{p} r(a_i^*) \qquad \text{(Eq. 2.2.5.6)}$$

We reach a contradiction since we assumed that A^* is the optimal solution to our problem and, therefore, there cannot exist a suboptimal solution with a higher score.

(C) Recursive definition using Bellman's Equation. This proves that the NDW algorithm follows the optimal substructure, and we can express it recursively using Bellman's Equation:

$$N_{i,j} = max \begin{cases} N_{i-1,j-1} + r(c_i, d_j) & \text{Match or Mismatch} \\ N_{i,j-1} + g, & \text{Gap in sequence } D \\ N_{i-1,j} + g, & \text{Gap in sequence } C \end{cases} \qquad \text{(Eq. 2.2.5.7)}$$

Each algorithm's decision to solve the problem will determine if it aligns two characters or adds a gap in either of the sequences C or D. Based on this, we derive the formula in Eq. 2.2.5.7, in which we assume a matrix of dimensions (m, n) with the characters of C indexed by i to the characters of D indexed by j.

2.2.5.3 BLAST

Basic Local Alignment Search Tool (BLAST) (Altschul et al. 1990) is a publicly available algorithm that compares biological sequences against a library of sequences and provides a list of the most similar ones given a similarity threshold.

BLAST is based on the Smith-Waterman algorithm (Smith and Waterman 1981) for *local sequence alignment* (see Section 2.2.5.1), which uses dynamic programming. There are only minor differences between the Smith-Waterman algorithm and NDW. In both cases, the algorithm compares characters individually using a scoring schema or substitution matrix to determine a score for each pair of characters. Aligned subsequences are given a similarity score by adding up these individual scores and applying gap penalties. A higher score indicates a more relevant alignment with more matches and fewer gaps and mismatches. To filter out irrelevant alignments, BLAST allows users to set a numerical threshold for the score, considering only highly relevant alignments during the search.

Smith-Waterman traverses the entire search space to identify the highest-scoring alignments. However, this approach becomes impractical when searching through databases with thousands of sequences. To address this issue, BLAST uses heuristics to identify statistically significant alignments based on a similarity threshold. This method avoids exploring the entire search space between two sequences, which reduces computational costs. Nevertheless, it does not guarantee optimal solutions.

To avoid going through the entire search space, BLAST uses *three heuristic phases* to refine potential high-scoring pairings.

1) **Seeding.** The user specifies a word length, w. Then, the algorithm identifies alignments of that length with a score equal to or greater than a threshold T. These matches are the seeds for possible alignments.

   During the seeding phase, the algorithm masks **Low complexity regions**, which are subsequences of little biological interest as they are common in many other sequences but which produce high scores. They are soft-masked to avoid seeding in such regions, meaning they are included in the alignment in the extension phase but not in the seeding phase.

2) **Extension.** After seeding the search space, the algorithm generates new alignments by extending the individual seeds, gradually increasing the word length. A threshold X decides when to stop extending by limiting how much the alignment score can drop from the previous maximum.
3) **Evaluation.** After extending all the seeds, the alignments are assessed to determine which ones are statistically significant. To establish this significance, the algorithm sets a score threshold S, which categorizes alignments as either low- or high-scoring, and then presents the most similar sequences to the user.

2.2.5.4 Affine Gap

Overview. The result of any sequence alignment algorithm may include gaps, i.e., characters that are not present in the other sequence and, therefore, are not aligned to any element, represented by "–". In the NDW algorithm (see Section 2.2.5.2), the penalty for a gap is set as a constant value for all the alignments. This results in alignments containing alternating sequences of gaps and matches. However, this approach may be ineffective when the expected alignments contain long gaps instead of many small sequences of alternating gaps and matches. Alignments with longer gaps are more meaningful when detecting anomalies in biological sequences (Altschul, Erickson, and Leung 1986).

To better model this phenomenon, alignment algorithms such as BLAST typically use an evaluation gap penalty system known as *affine gap*, which imposes a higher penalty for opening a new gap than for extending an existing gap. If P_{op} is the penalty for opening a gap, and P_{ex} is the penalty for extending one, the total penalty of opening a gap of length L would be $P_{op} + P_{ex} * (L - 1)$.

These penalties are included in the scoring schema of the sequence alignment algorithm so that the optimal alignment prioritizes these longer gaps. When using affine gaps, it is necessary to consider the options of opening or extending a gap in both sequences for every pair of characters. This means adding three extra scoring matrices to the algorithm, which increases the algorithm's time and space complexity.

Implementation. To determine whether to open or continue a gap in either of the sequences for every pair of characters, we require four matrices: one for each sequence to decide on opening or continuing a gap, one to assess matches and mismatches, and one to select the highest value among the three matrices. These matrices are respectively:

$H_A(i,j)$ – max score of the alignment of A and B where $A[i]$ is aligned with a gap
$H_B(i,j)$ – max score of the alignment of A and B where $B[i]$ is aligned with a gap
$H_M(i,j)$ – max score of the alignment of A and B where $A[i]$ and $B[i]$ are aligned

$H(i,j)$ – max score of the alignment of A and B

The initial conditions to fill in the first row and column of the matrix are the following:

$$
\begin{aligned}
H(i,0) &= H(0,j) = 0 \\
H_M(i,0) &= H_M(0,j) = -\infty \\
H_A(i,0) &= H_B(0,j) = P_{op} + P_{ex} \\
H_A(0,j) &= H_B(i,0) = -\infty
\end{aligned}
$$

The Bellman Equations for this extended problem are as follows:

$$
\begin{aligned}
H(i,j) &= max\{H_M(i,j), H_A(i,j), H_B(i,j)\} \\
H_M(i,j) &= H(i-1,j-1) + equals(A[i], B[j]) \\
H_A(i,j) &= max\{H(i-1,j) - (P_{op} + P_{ex}), H_A(i-1,j) - P_{ex}\} \\
H_B(i,j) &= max\{H(i-1,j) - (P_{op} + P_{ex}), H_B(i-1,j) - P_{ex}\}
\end{aligned}
$$

Once the matrix is filled using this schema, the back-tracking process is the same as in the NDW algorithm: Select the maximal score at the bottom-right of the matrix, i.e., $max\{H(i,j)\}$, and backtrack choosing the maximal scores.

Parameters values. The penalties for opening and extending a gap in BLAST are determined empirically and typically rely on the frequency scoring matrix used for the alignment (Korf, Yandell, and Bedell 2003). In general, the algorithm performs well when the penalty for opening a gap (P_{op}) is between 10 and 15 times the penalty of extending it (P_{ex}) (Altschul et al. 1990).

2.2.6 Distance Measures

Once we have aligned two traces, we can measure their *distance* to evaluate how close or far apart they are. There are two main groups of distance metrics: *lock-step measures*, which compare the i-th point of one-time series with the i-th point of the other; and *elastic measures*, able to compare one-to-many points or even one-to-none (Mori, Mendiburu, and Lozano 2016).

Lock-step measures. Let $X = \{x_i\}_{i=1}^n$ and $Y = \{y_i\}_{i=1}^n$ be two already aligned traces, i.e., $A = \{(i, i)\}_{i=1}^n$. Then, assuming that $d(p, q)$ is a distance measure between a pair of snapshots p and q (e.g., the Euclidean or the Manhattan distance), we can compute the average distance between the two traces, $E(X, Y)$, and its sample standard deviation, $s(X, Y)$, as follows:

$$E(X,Y) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, y_i)$$

$$s(X,Y) = \sqrt{\frac{1}{n-1} \left(\sum_{i=1}^{n} d(x_i, y_i)^2 - n \cdot E(X,Y)^2\right)}$$

Elastic measures. This type of measure allows for considering one-to-many or one-to-none element matchings, enabling more flexible alignments. One of the most representative measures in this group is the Fréchet distance.

The Fréchet distance $F(X, Y)$ between two traces X and Y is defined as the infimum over all reparametrizations χ and ψ of the maximum over all $t \in [0, 1]$ of the distance between $X(\chi(t))$ and $Y(\psi(t))$ (Ball 1984). It is generally explained through this example: Picture a person walking from one end of a path represented as X to the other end, with their position being $X(\chi(t))$. Similarly, imagine a dog walking from one end of its path, represented as Y to the other end, with its position being $Y(\psi(t))$. The person holds the dog by a leash. The Fréchet distance between both is the minimum leash length needed to walk the dog following their paths. To define this formally, if we assume that $d(X(\chi(t)), Y(\psi(t)))$ represents the distance between individual positions, the Fréchet distance can be defined as follows:

$$F(X,Y) = \inf_{\chi,\psi} \max_{t \in [0,1]} \left\{ d(X(\chi(t)), Y(\psi(t))) \right\}$$

The reparametrizations χ and ψ encompass all potential alignments of the elements within the traces X and Y. One-to-many alignments are intuitively considered, meaning the dog and the owner can wait for each other. The ultimate objective is to determine which of the various routes requires the shortest leash, and this is precisely what the Fréchet distance measures.

The Fréchet distance cannot detect slight changes between the alignments because it only provides a global measure of the similarity of the aligned snapshots. In contrast, the Euclidean measures compute the average distance between all the paired snapshots. Therefore, the Fréchet distance is usually more appropriate in applications where a maximum distance cannot be exceeded, e.g., a moving object that cannot collide with surrounding obstacles. In contrast, lock-step measures provide an overview of the alignment quality. They are less sensitive to outliers, making them appropriate for applications where the average distance between the traces matters.

3

DIGITAL TWIN CONCEPTUALIZATION

The thesis revolves around the concept of a *Digital Twin* to provide a solution for its validation, specifically for measuring the fidelity of its behavior. Initially, our efforts focused on defining the concept and proposing a conceptual architecture. This chapter presents our proposal based on the findings from our experiments, the existing literature, and interactions with other members of the DT community.

Specifically, we propose a conceptual architecture for defining and deploying digital twin systems in a decoupled manner, illustrating it with high-level models using UML and OCL (Muñoz, Troya, and Vallecillo 2021; Pérez-Porras et al. 2022). Our second proposal involves refining the initial conceptual architecture into a framework by considering additional challenges and elements, following discussions with the DT community and further experiences with real case studies (Muñoz, Troya, and Vallecillo 2023).

As part of these works, we introduced the concept of *Digital Twin System* (DTS), which encompasses the DT, the PT, the services, and the connections between them. This term differentiates our proposal from others by identifying the DT solely as the virtual replica, distinguishing it from other typically included elements like connections or services.

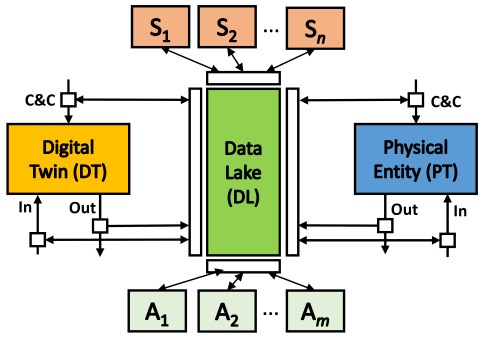


Figure 3.1: Digital twin system architecture by Muñoz et al. (2021) (S_n *and* A_n *are the service and analysis components respectively).*

3.1 A Conceptual Architecture for Defining and Deploying Digital Twin Systems

In our initial works (Muñoz, Troya, and Vallecillo 2021, 2023), we introduced an architecture for defining and deploying DTs. The main goal of these proposals was to present a modular architecture that facilitates prototyping with high-level DTs developed through standard UML (Object Management Group 2015) models with OCL (Object Management Group 2014) constraints. We employed the modeling tool USE (Gogolla, Büttner, and Richters 2007) because it enabled the definition of system behavior using its executable language, SOIL (Büttner and Gogolla 2014).

The reference architecture, depicted in Figure 3.1, uses high-level models of different DTs but can also be applied to final replicas by replacing UML models with lower-level implementations. After completing the design exploration phase with these lightweight models, the final implementations could be derived through model transformation or by using these models as blueprints. Our papers demonstrate the architecture through the development of DTSs for a Lego Mindstorms NXT Car (Muñoz, Troya, and Vallecillo 2021) and an Arduino Braccio (Muñoz, Troya, and Vallecillo 2023).

3.1.1 Physical and Digital Twin Interface

Our architecture and digital twins, in general, are focused on enhancing the capabilities of the replicated physical system. Let us start by introducing the relevant elements of the interface of the physical entity, i.e., the potential physical twin, as depicted in Figure 3.2.

Typically, a physical entity interacts with its environment through inputs (`In`) and outputs (`Out`). Inputs are typically generated by sensors that detect events or changes in the environment. Outputs are the reactions of the Physical Entity to input events from

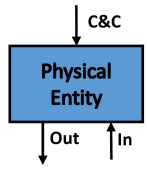


Figure 3.2: Simplified view of the Physical Entity.

the environment, commands issued to it, or interval changes. Physical Entities can also accept external orders in the form of commands that control their behavior or modify the values of their configuration parameters. This interaction is represented by the Command and Control (`C&C`) incoming arrow.

The DT will have the same input and output interface as the PT, acting as a specular image. The inputs for the DT could be generated in a virtual environment or be signals forwarded from the sensors of the physical entity. This could allow us to study whether the DT reacts the same way as the PT when given the same output signals. Given the same inputs, the outputs of both twins should be equivalent, with a certain degree of fidelity, since the DT is a model.

3.1.2 Digital Twin System Architecture

Our proposed architecture, depicted in Figure 3.1, is inspired by the 5-dimensional architecture by Tao et al. (Tao et al. 2018, 2019) introduced in Section 2.1.2. It consists of the physical and digital parts, the system's data, the connections between them, and a set of services. The Physical Entity from Figure 3.2 appears as the PT on the right, while the DT, its digital replica, is on the left, mirroring it.

A **Data Lake (DL)** facilitates communication between the twins. According to (Hai, Quix, and Jarke 2021), a data lake is "a flexible, scalable data storage and management system that ingests and stores raw data from heterogeneous sources in their original format and provides query processing and data analytics in an on-the-fly manner." The remaining architecture components use the DL to write data and gather information in a loosely-coupled and asynchronous manner. The DL follows the Blackboard architectural pattern (Buschmann et al. 1996) but with the property that the information is stored in raw format (i.e., as produced by the sources).

The different components access the DL through ***drivers***, which convert the data into formats that each component can understand. In Figure 3.1, the drivers are represented by white rectangles around the DL. The drivers attached to the physical entity connections `In`, `Out`, and `C&C` can intercept the information flowing through them and store it in the DL. Similarly, they can query the DL for commands or inputs stored by other components and

send them to the physical entity, enabling, e.g., the emulation of the physical environment. Furthermore, the drivers can modify parameters during run-time or implement self-adaptive behaviors. From the DT's perspective, the drivers can be used, e.g., to simulate the inputs and commands received by the physical entity and record its outputs in the DL.

The ***Service*** **components** ($\mathtt{S}_1$,$\mathtt{S}_2$,..., $\mathtt{S}_n$) are responsible for implementing the additional functionality in a DTS. Examples of this functionality include dynamic data visualization dashboards or algorithms that learn from the past movements of a car, draw floor maps, and avoid obstacles. Note that in the Blackboard architectural pattern, any permitted component can write in the DL, allowing services to issue commands as if they were external users.

Finally, the ***Analysis*** **components** ($\mathtt{A}_1$,$\mathtt{A}_2$,..., $\mathtt{A}_n$) implement various tests on the twins, their connection, behaviors, or synchronization. For instance, a monitoring component can check that the traces produced by the DT and the PT show faithful behavior (Khan et al. 2018), as our validation tool does. Another analysis component can test a single twin by sending commands and verifying the responses. Analysis components can also validate Service components, ensuring that their algorithms for self-adaptive behaviors or machine learning predictions function correctly.

As a final remark, the framework's architecture supports multiple digital twins, each one dedicated to specific aspects of the physical entity, and can integrate more than one physical twin. Additionally, service or analysis components can be dynamically added to the system during execution, as all communications between components occur through the DL.

3.2 Framework for Digital Twin Systems

After discussions with the DT community[1] and our experience with different implementations, we proposed refining the initial architecture (Muñoz, Troya, and Vallecillo 2023). The underlying idea of the architecture remains: a symmetric, modular architecture for the twinned systems. The other components of this framework will focus on ensuring the proper interaction between the twins and enhancing the performance of both counterparts. It is also centered around a DL with a set of drivers for accessing it. Still, in this new version, a dedicated component orchestrates the exchange of data and commands between the elements of the DTSs.

[1] In particular, the discussions at the 2022 Dagstuhl Seminar 22362 "Model-Driven Engineering of Digital Twins" (Cleophas et al. 2023) and the 2023 Cargèse CAMPaM workshop on "Model-Based Systems Engineering for and by Digital Twins", which allowed us to refine and improve our proposals.

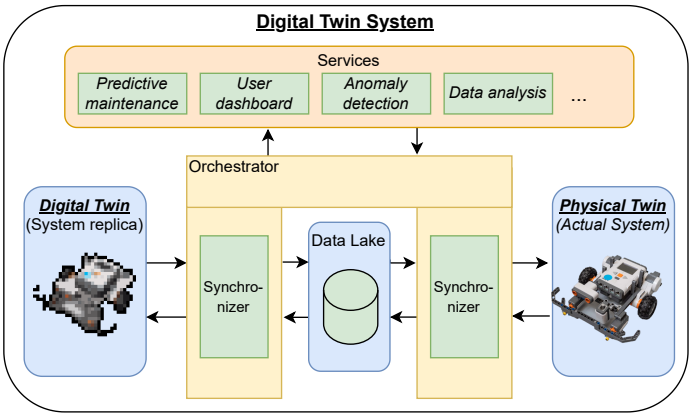


Figure 3.3: Conceptual framework for Digital Twin Systems by Muñoz et al. (2023).

Figure 3.3 depicts our proposed framework for implementing Digital Twin Systems. Before providing a detailed description, it is essential to explicitly define the key terms:

- **Physical Twin** (also **Actual System** or **Observable Object**). This is the system, service, or product whose behavior we want to optimize using a digital replica. The state of the PT is continuously monitored, and the values of its properties of interest are periodically stored. The PT may already exist when the DT is created, or it may not exist, for example, if we intend to use the DT during the design phase of the physical twin.
- **Digital Twin**. It is the synchronized replica of the PT, updated at regular intervals and modeled with the required fidelity level to capture the PT's properties of interest at the necessary level of detail. The DT represents the system's properties of interest and emulates its behavior using data and (analytical or simulation) models, which are continuously updated and stored in the DL throughout the system's lifespan.

Building upon the above definitions, we define a **Digital Twin System (DTS)** as a system engineered for a specific purpose, composed of:

- The *Physical Twin* (PT).
- The virtual replica, i.e., the *Digital Twin* (DT).
- The synchronization mechanisms that facilitate the exchange of data and commands between the twins.
- A set of *Services* that enable the exploitation of the data produced and exchanged between the twins to support achieving the DTS objectives, e.g., performance improvement, predictive maintenance, or anomaly detection.

As in our previous proposal, the framework connects the elements through a Data Lake, using the Blackboard architectural pattern (Buschmann et al. 1996), allowing asynchronous exchange of information. However, if critical information requires minimal latency, elements can also connect directly through the ***Orchestration*** component. This component enables the direct exchange of information between elements and connection to the DL. Additionally, the Orchestration component includes the necessary synchronization services to ensure the correct operation of the system. We removed the analysis elements and included them as services, as we found that the previous distinction was not very useful. In most of the existing literature, there is no such distinction.

The new framework maintains the flexibility in connecting the elements and the highly independent connection between them from the previous proposal. Adding a new element would only require adapting the information input to the Orchestration component without affecting any other system elements. Similarly, if multiple DTs need to be interconnected and their behavior synchronized within the DTS, it can also be achieved through the Orchestration component. We can also have several DTs with different levels of fidelity organized to choose the best one to use at each moment, depending on the state of the PT, the concrete performance requirements, or the degree of faithfulness needed. Additionally, we can replace the PT with another virtual model for comprehensive system testing. All these changes can be accomplished by simply modifying the specific interface implementation for the Orchestration component.

In this work (Muñoz, Troya, and Vallecillo 2023), our primary objective was to precisely define the different concepts that comprise any DTS. Currently, there is a lack of clarity in the terminology used to distinguish between the system's replica (referred to as DT) and the entire system (referred to as DTS), which involves the exchange of information and services. The term "Digital Twin" is often inconsistently used in both cases. Our work advocates for a symmetric definition in which we have two twinned systems whose operations are synchronized. The other elements of the framework ensure the proper interaction between them and optimize the operation of the target system based on its DT. This whole system is what we call the DTS and is clearly distinguished from the digital replica.

These definitions and considerations will remain consistent throughout the following sections when we refer to the terms DT, PT, and DTS.

4

DIGITAL TWIN FIDELITY ASSESSMENT

During operation, Digital Twin Systems (DTSs) can be used for different purposes, such as behavior optimization, monitoring, system validation, and prediction. An essential requirement in any of these contexts is the accurate representation of the PT's properties of interest. If the DT is not sufficiently faithful, any predictions or conclusions based on it will be unreliable. Therefore, the engineering of DTSs is critical (Bordeleau et al. 2020). Currently, most research focuses on the design, development, and deployment of DTSs, with little attention given to their validation (Dalibor et al. 2022).

This chapter presents our proposal for validating DTs by assessing their level of fidelity. The contents are structured as follows: Firstly, we provide some fundamental definitions of key concepts of our approach. Then, we introduce a practical example—the DT of an elevator—and a detailed discussion of our algorithm.

4.1 *SnapAligner* - Measuring the Fidelity of Digital Twins

Our approach assesses behavior fidelity by comparing the behavioral traces of two twins. This is done through a three-step process. First, we represent the system's states as snapshots and define a comparison function to determine whether two snapshots are sufficiently similar based on a threshold (Section 4.3), known as the maximum acceptable distance (MAD), for each property of interest.

Second, we align the equivalent states reached by the twins using a trace alignment algorithm (Section 4.5). This algorithm adapts the NDW algorithm, modified to work

with behavioral traces. It also incorporates two optimizations from the BLAST algorithm: handling low-complexity regions and the use of affine gaps. These optimizations improve the initial alignments produced by NDW, ensuring more relevant parts of the trace are properly aligned.

Finally, we measure the fidelity of the two systems by evaluating the level of alignment, expressed as the percentage of matched snapshots, and calculating the distance between the aligned traces, which reflects how close the behaviors of the twins are (Section 4.6).

The approach is demonstrated through the case study of the Mondragon elevator, showcasing its effectiveness in this context.

To apply this method, certain conditions must be met.

- The traces must exhibit **synchronized behavior** when starting at the same time.
- The systems should provide **snapshots at identical sample intervals** we work on the assumption that two identical behaviors coincide in 100% of the snapshots; for this to hold true, the snapshots must be sampled at the same time intervals.
- The snapshots should capture the **behavior of interest** for validation, allowing us to assess the similarity between the states based on the differences in their attributes of the corresponding snapshots.

When these conditions are fulfilled, our approach offers a set of metrics that indicate the degree of fidelity between the systems, identifying parts of the traces that contain delays, inconsistencies, or behavioral differences. This process works offline, detecting discrepancies in the traces to determine if the twins would traverse the same states simultaneously.

4.2 Running Example - An elevator

To illustrate the problem and present our solution, we will use the case study of an *elevator* at the University of Mondragon in one of its buildings on the Arrasate Campus. The elevator serves five floors, numbered 0 to 4, and is primarily used by students who are unable to use the stairs or need staff to transport heavy loads. Recognizing the elevator's frequent usage and critical role, the University decided to optimize its operation and maintenance to save energy and prevent breakdowns.

To achieve this, the University decided to implement a DTS using the commercial simulator *Elevate* (Peters Research 2023) as DT. This simulator can be configured with the different physical parameters of the specific elevator to emulate its behavior, in particular, the acceleration during floor transitions.

From this acceleration, it is possible to deduce the speed and descent times to estimate the degradation of the equipment and verify if the configuration is optimal. However, they wonder whether and to what extent the simulator's behavior is faithful to the physical system's.

In this case, our proposal aims to assess whether the acceleration sequences of the simulator are sufficiently faithful to undertake the role of a DT, especially in terms of user comfort and the absence of abrupt movements. The DT's accelerations will be compared with those of the real system, which have been measured by traveling in the elevator using a *WITMOTION WT901BLECL* accelerometer.

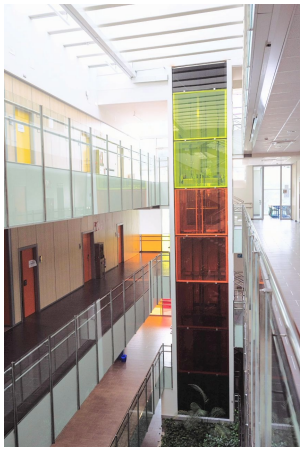


Figure 4.1: The elevator.

4.3 System Representation

To apply the sequence alignment algorithm to behavioral traces, we need a discrete representation of the system's behavior over time. This is achieved by periodically capturing observations of the system's state, a process commonly referred to as sampling, much like dividing a film into individual frames. The resulting representation is adjusted to the required level of abstraction, including only the information related to the properties of interest. For instance, Figure 4.2 shows two snapshots of each twin. In this example, snapshots only include the elevator's acceleration (the property of interest) and the timestamp at which the snapshot was taken.

Figure 4.3 displays the sequences of snapshot values—specifically the acceleration data—captured from both the real system (the PT, top) and the Elevate simulator (the DT, bottom) for a specific scenario we refer to as (4-0-4). In this scenario, the elevator starts at floor 4, descends to floor 0 (from timestamp 5.8 to 24.1 in the DT), stops, and then ascends back to floor 4 (from timestamp 44.7 to 63 in the DT). The graph reveals four peaks divided into two groups. The first two peaks represent the acceleration during the descent, with the first peak showing negative acceleration as the elevator moves down and the second peak showing positive acceleration as the elevator brakes upon reaching floor 0. Similarly, the following two peaks correspond to the elevator's ascent, following the same pattern.

The acceleration and deceleration peaks for floor transitions are remarkably similar between the simulation and the real system, aside from minor accelerometer noise near zero when the elevator moves at a constant speed. However, one aspect present in the real system that is absent in the simulation is the subtle additional acceleration that occurs

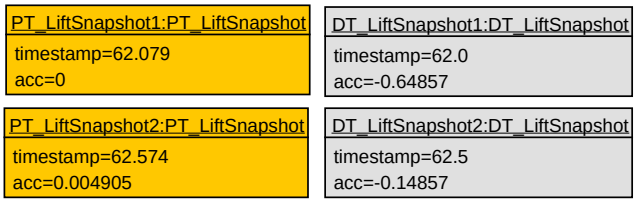


Figure 4.2: Sample elevator snapshots from the PT and the DT.

whenever the elevator brakes in either direction. This slight acceleration, which smooths out the elevator's stop to enhance user experience, appears as a minor peak following the primary braking acceleration in the same direction. This pattern is specific to this particular elevator model.

The graphs also reveal a slight delay between the DT and the PT. This discrepancy arises because the PT was manually activated while the simulation ran regularly. Therefore, aligning the traces requires using similarity functions rather than relying solely on timestamps. This is particularly important in cases like this, where we cannot assume that both systems are operating in perfect synchrony.

Finally, the accuracy of the discrete trace representation relies on how often the snapshots are taken. Sampling snapshots at shorter intervals leads to a more accurate trace. However, taking snapshots too frequently can result in too many identical snapshots, which do not provide new information, increase storage requirements, and slow down analysis algorithms. In our study, we captured 696 snapshots, averaging approximately 11 per second.

We conducted ten executions of this scenario to analyze how variations could affect the alignments and to account for uncertainties and noise in the behavior. In the algorithm's introduction, we reference execution 04 and execution 01 to illustrate some of its properties. The analysis of all scenarios and the ten executions for the (4-0-4) scenario is provided in the technical report of this case study (Muñoz et al. 2023b).

4.4 Similarity Function

The first step in aligning the traces is to define a similarity function that can effectively compare two system snapshots. In fields like bioinformatics, comparison functions are relatively straightforward because they often deal with sequences composed mainly of characters. In contrast, snapshots can be complex structures with heterogeneous records, as they represent the system states

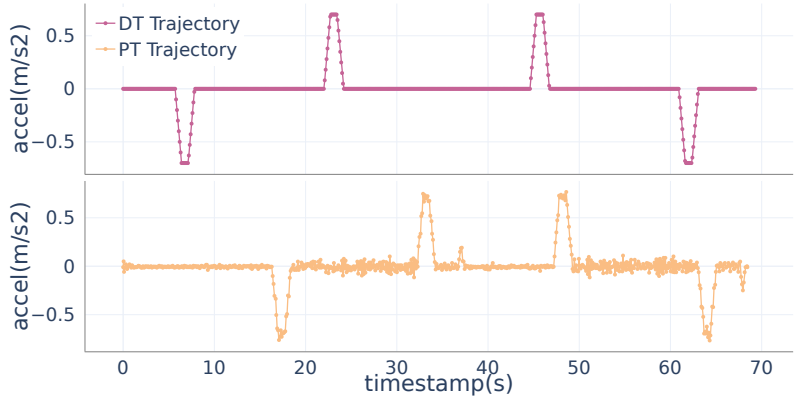


Figure 4.3: Elevator traces of the PT and DT. Scenario (4-0-4), execution 04.

Algorithm 2 describes the similarity function "$\mathcal{S}()$" between two snapshots s_A and s_B. We assume that the two snapshots are composed of the same k typed attributes $\{a_1, ..., a_k\}$. The function returns a real number in the range [0..1], where a score of 1 indicates that the snapshots are identical, and a score of 0 indicates that they are different.

The difference between each attribute is computed based on its type:

- For **numerical values**, we use a real number that represents the *Maximum Acceptable Distance* (*MAD*) between two values of one attribute to be considered "similar". Each attribute will have a different threshold depending on its nature; therefore, we use an array of MAD values. If the difference exceeds the *MAD*, we assign a similarity score of 0. If the difference is within the *MAD*, the similarity score increases gradually as the difference decreases, as detailed in line 6 of the algorithm. A similarity score of 1 is assigned if the values are equal.
- In the case of **Boolean values**, a similarity score of 1 is given if the values are the same and a score of 0 if they are different.
- For **enumerated or string attributes**, we can either use a similar approach or apply the comparison operations defined by Fernández-Candel et al. (2024) (namely, *uEquals()*) to calculate a confidence value. This value indicates the confidence that the two strings are equal, with 1 meaning complete certainty and 0 indicating no confidence. To obtain a similarity value, we would adjust it to be 1-*uEquals()*).

We could also account for uncertainty in the comparisons by considering the potential measurement errors of the instruments. In this case, a numerical value with associated uncertainty could be used, and comparisons could be performed using a confidence threshold (Fernández-Candel et al. 2024). This approach would enable more robust comparisons in the presence of noise, as is typical in cyber-physical systems such as in the elevator case study.

Algorithm 2 Similarity function between two snapshots

```
1: i ← 1; result ← 0
2: while i ≤ k do
3:     if isNumerical(s_A.a_i) then
4:         dif ← abs(s_A.a_i − s_B.a_i)
5:         if dif < MAD_i then
6:             result ← result + (1 − dif / MAD_i)
7:         else
8:             result ← 0                          ▷ Mismatch if MAD_i exceeded
9:             break
10:        end if
11:    end if
12:    if isBoolean(s_A.a_i) then
13:        if s_A.a_i = s_B.a_i then
14:            result ← result + 1
15:        else
16:            result ← 0
17:            break
18:        end if
19:    end if
20:    if isString(s_A.a_i) then
21:        if equals(s_A.a_i, s_B.a_i) > MAD_i then
22:            result ← result + (1 − equals(s_A.a_i, s_B.a_i))
23:        else
24:            result ← 0
25:            break
26:        end if
27:    end if
28:    if lowComp(s_A.a_i) ∧ lowComp(s_B.a_i) then
29:        result ← result ∗ LCAW/2
30:    else if lowComp(s_A.a_i) ∨ lowComp(s_B.a_i) then
31:        result ← result ∗ LCAW
32:    end if
33:    i ← i + 1
34: end while
35: return result/k
```

Lines 24-28 address the treatment of the low-complexity regions, as detailed in Section 4.5. We identify trace elements within these low-complexity regions using the function *lowComp()*. To mitigate the impact of these elements on the overall similarity score, we adjust their score by a constant *LCAW* (e.g., 0.005). If one of the two snapshots falls within a low-complexity region, the reward is multiplied by *LCAW* (line 27). If both snapshots are in low-complexity regions, the reward is halved (line 25). We provide a detailed explanation in the next section. Additionally, the values of the different parameters were calculated based on experiments presented later.

This algorithm evaluates the reward for each attribute and computes the average, assuming equal weight for all attributes. Alternative approaches could be defined if needed, such as assigning different weights to attributes or using the minimum similarity instead of the average. Additionally, not all attributes need to be included in the similarity function; only those relevant to the comparison need to be included, depending on the purpose of the DT. To restrict the timeframe for matching snapshots, we could incorporate the timestamp attribute into the similarity function and specify a MAD value. However, this restriction is not imposed in the examples provided, allowing for the detection of similar behavior despite potential delays.

In the Elevator case study, we are especially interested in the acceleration value. For instance, considering the snapshots in Fig. 4.2, if we compare the first `PT_LiftSnaphot` with the two DT snapshots using a MAD of 0.4 m/s^2, it yields similarity scores of 0 and 0.63, respectively.

4.5 Trace Alignment Algorithm

Overview. The *SnapAligner* algorithm is an adaptation of the NDW algorithm (Section 2.2.5.2) to work with behavioral traces. It incorporates two key optimization techniques from BLAST (Section 2.2.5.3):

a) it avoids initiating alignments in ***low-complexity regions***, focusing instead on more distinctive parts of the trace.

b) it uses the ***affine gap*** technique to manage gaps and mismatches, ensuring more meaningful alignments by preventing excessive alternation between gaps and matches.

The SnapAligner algorithm shares a core structure with the NDW algorithm, but it is specifically designed for aligning behavioral traces. It uses a **similarity score function** $\mathcal{S}()$ to compare the elements of the traces (refer to Algorithm 2). Additionally, the similarity function considers **low-complexity regions** and adjusts the score accordingly. Applying this similarity function, SnapAligner introduces three additional matrices to implement the **affine gap** mechanism, as outlined in Section 2.2.5.4.

Bellman's Principle of Optimality. The extension of the algorithm still adheres to *Bellman's Principle of Optimality* by preserving the *optimal substructure*. Specifically, the similarity function between two elements assesses their similarity based solely on those elements, as in the original approach, maintaining the independence between subproblems—a key requirement for Bellman's Principle of Optimality. Instead of using a substitution matrix for amino acids, the algorithm uses the similarity function between snapshots. Thus, the optimality proof remains analogous to the one presented in Section 2.2.5.2. Note that the optimal alignment may not always have the lowest distance between two traces or the highest fidelity metrics. In this case, the optimal alignment is defined as the one that returns the highest score based on the comparison function between snapshot pairs and the scoring schema.

Parameters. *SnapAligner* has four parameters that allow customizing its behavior to any particular system: Maximum Acceptable Distance (*MAD*), Low-Complexity Area Weight (*LCAW*), gap opening penalty (P_{op}), and gap extension penalty (P_{ex}). The similarity function $\mathcal{S}()$ utilizes MAD and LCAW, while the other two parameters are applied within the alignment algorithm to configure *affine gap*. These four parameters are described next.

1) *MAD* values. To define the *MAD values* for each snapshot attribute, we determine the maximum distance between two values of the same attribute to be considered similar, i.e., so that the states they represent can be matched because they are "sufficiently similar." Typically, MAD values are set proportionally—according to our experiments, around 2 to 3 times—to either the accuracy of the measuring instrument used to obtain the attribute values or the tolerance of the physical property represented by the attribute.

In the elevator case study, we focus on one attribute: acceleration. The *MAD* value chosen for acceleration is 0.15 m/s^2, which is three times the accelerometer's accuracy of 0.05 m/s^2. A smaller *MAD* tends to produce alignments with fewer matched pairs but results in a smaller distance between the traces, as the MAD value sets an upper bound on the distance metrics (FD, ED). Conversely, alignments with very high *MAD* values can lead to inconsistent and meaningless alignments by matching incompatible values.

2) Low-complexity area weight (*LCAW*). *Information theory* (Shannon 1948) quantifies information through *entropy*, which measures the amount of uncertainty in the outcomes of a random variable. Entropy is maximized when all possible values have equal probability, and it decreases as the probability of one outcome prevails over the others. In the elevator case study, regions with lower entropy correspond to periods of constant speed, where acceleration is close to zero. These regions contain less information due to noise and contribute less to characterizing the elevator's behavior than the acceleration curves.

Alignment algorithms like BLAST aim to maximize the similarity between aligned pairs. Still, common low-entropy regions, such as the constant speed areas, return high similarity scores, shifting the attention from more characteristic subsequences. To countereffect this, we mask these low-entropy snapshots by reducing their reward, thus redirecting the alignment algorithm toward more meaningful regions. In the elevator example, constant-speed regions are identified by acceleration values below the accelerometer's accuracy ($0.05\ m/s^2$), making them indistinguishable from zero.

To mask them, we introduce the LCAW, which reduces the influence of low-complexity areas. The LCAW is the product of two factors: the proportion of snapshots relevant to the behavior of interest (r) and the weight (s) assigned to non-relevant snapshots. In the elevator case study, we found that the elevator changed speed only 10% of the time, so $r = 0.1$. Additionally, we set $s = 1/20$, meaning that the weight of non-relevant snapshots was 1/20 of the weight of relevant ones. Thus, LCAW was set to 0.005.

To illustrate the effect of considering low-complexity areas, we align the traces in Figure 4.3 using *MAD* 0.15 without accounting for low-complexity areas. As shown in Subfigure 4.4a, although all four acceleration peaks represent the same behavior, the first two acceleration curves are not aligned, while the last two are. This misalignment results from the algorithm's reward schema, prioritizing high-similarity regions, such as the zero-acceleration areas, over the acceleration curves. For example, comparing a DT acceleration value of $0\ m/s^2$ with a PT noised value of $0.005\ m/s^2$ from the zero-acceleration area results in a reward of 0.97 while comparing two similar snapshots from the acceleration curves, $0.7\ m/s^2$, and $0.76\ m/s^2$, their difference is 0.06, so we obtain the following reward: $1 - (0.06/0.15) = 0.6$. The algorithm thus tends to align zero-acceleration areas to maximize the score.

By applying *LCAW*, as shown in Subfigure 4.4b, we refocus the alignment on the acceleration curves, leading to more meaningful results. Snapshots with acceleration lower than $0.05\ m/s^2$ are treated as low-complexity areas with *LCAW*$= 0.005$, helping the algorithm prioritize the behaviorally relevant regions.

3) Scoring schema values (P_{op}, P_{ex}). The scoring schema for selecting the optimal alignment includes four key parameters: **match**, **mismatch**, **gap opening penalty** (P_{op}), and **gap extension penalty** (P_{ex}).

Unlike traditional algorithms like NDW or BLAST, where constant scores are assigned, the reward for a *match* in our approach depends on the similarity between the two snapshots being compared. Specifically, the reward is a value in the range $(0, 1]$, which reflects how similar the attribute values are, as computed by the *similarity* function. A *mismatch* is assigned a score of 0 when the attribute difference exceeds the *MAD* value, but the snapshots are still paired to optimize the alignment.

Both the *gap opening penalty* (P_{op}) and the *gap extension penalty* (P_{ex}) are negative values, representing penalties. These parameters are typically fine-tuned based on experimental results. In our analysis, we found that the algorithm performs best when the opening gap cost P_{op} is set between $[-0.5, -0.1]$, i.e., 10% and 50% of the perfect match reward. Changes within this range do not significantly affect the results.

To see the effect of applying *affine gap* to an alignment, we can compare the Subfigures 4.4b and 4.4c. We see that gaps in the first 20 timestamps are groups rather than alternating, making it easier to interpret the initial delay between the traces. This delay can be measured by counting the consecutive gaps and multiplying by the snapshots sampling rate.

Combining *LCAW* and *affine gap* provides two key advantages:

a) more accurate alignment of behaviorally relevant regions

b) more contextually meaningful gap groupings, facilitating better interpretation of the results.

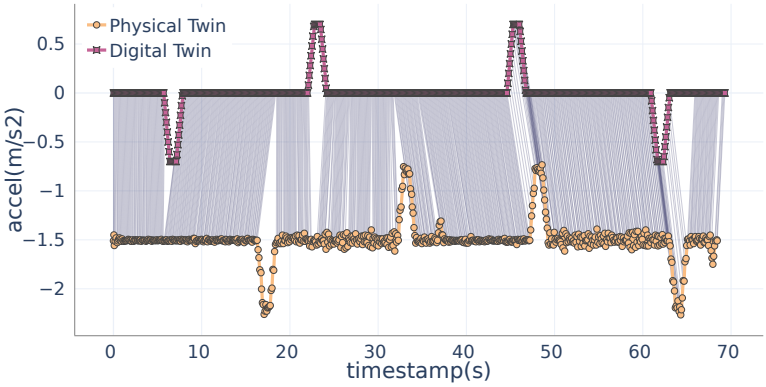
4.6 Fidelity Metrics

After aligning the traces, to asses how accurately the DT traces match those of the PT, we define three key metrics: the percentage of matched snapshots (%MS), the Frèchet Distance (FD), and the Euclidean Distance (ED).

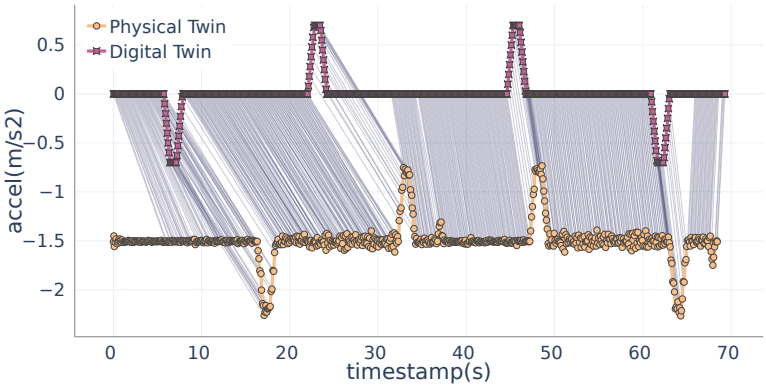
1) Percentage of matched snapshots (%MS). This metric quantifies the proportion of snapshots in the PT and DT traces that are successfully aligned. It is defined as:

$$\%MS_A^+ = \#X^{A+} / \max(\#X, \#Y) * 100 \qquad \text{(Eq. 4.6.0.1)}$$

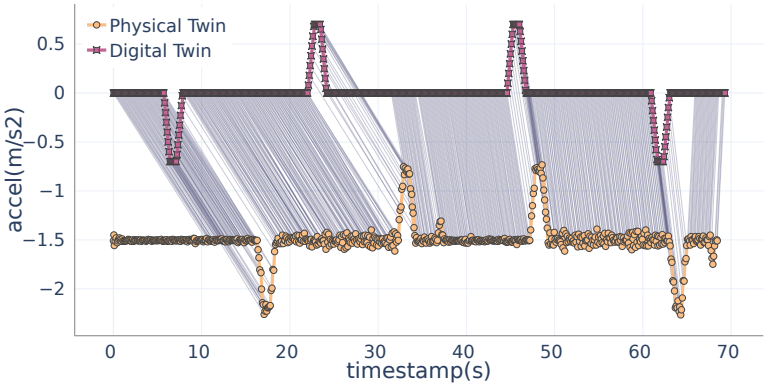
A higher %MS indicates a closer alignment between two trajectories, and poor alignment renders the distance metrics less meaningful.



(a) Base (No LCAW, No Affine Gap)



(b) LCAW (No Affine, LCAW)



(c) Affine Gap + LCAW

Figure 4.4: Alignments for Scenario (4-0-4). Execution 04. *MAD:* $0.15m/s^2$.

2) Distance measures. To evaluate the distance between the two trajectories, we use two distance metrics described in Section 2.2.6:

- **Frèchet distance (FD).** It captures the maximum distance between all matched snapshots, focusing on the largest discrepancies.
- **Euclidean distance (ED).** It provides the average distance between matched snapshots.

Note that snapshots from low-complexity areas are excluded from the distance calculations to prevent artificially lowering the average distance.

If the PT and DT traces are identical, all snapshots are matched (*%MS*=1) and both distance metrics are 0, meaning that both twins went through the same states. The *MAD* value plays a critical role in these metrics: a larger *MAD* increases the number of matched snapshots but also raises the distance values (since the *MAD* value establishes an upper bound for the Fréchet distance) and risk incorrect matches (cf. Section 4.5). Therefore, it is essential to balance *MAD*, matched snapshots, and distance to achieve a meaningful alignment. The value selection of these parameters is discussed below in Section 4.7.

Other metrics of interest could be:

- **Percentage of mismatches.** This metric indicates the proportion of snapshots that could not be aligned, which could indicate temporary deviations of the twins' behavior.

$$\%MS_A^- = \#X^{A-} / \max(\#X, \#Y) * 100 \qquad \text{(Eq. 4.6.0.2)}$$

- **Percentage of gaps.** This measures the proportion of gaps in the alignment, which can indicate delays between the twins' behaviors.

$$\%MS_A^G = 100 - (\%MS_A^+(X) + \%MS_A^-(X)) \qquad \text{(Eq. 4.6.0.3)}$$

We also examine **the number and the average length of sequences of consecutive gaps**, which provide insights into how gaps are distributed across the traces. Long gap sequences may signal a delay in one twin's behavior, while frequent alternation of short gaps may indicate stuttering behavior in one of them. Applying *affine gap* tends to increase the lengths of gaps while consolidating them into fewer groups, enhancing the clarity of the alignment by grouping related gaps and helping detect these symptoms.

To assess the impact of BLAST optimization techniques on fidelity metrics, we have compiled statistics from ten runs of scenario (4-0-4) as outlined in Table 4.1.

Table 4.1: Average fidelity metrics for the ten executions scenario (4-0-4).

Algorithm / *Variant*	Fidelity			Gaps			
	FD	ED	%MS	%	length	#	#groups
Base	0.126	0.022	93.43	6.11	3.59	82	24
LCAW	0.126	0.023	91.79	7.78	4.79	105	22
Affine	0.127	0.023	93.48	5.58	7.93	75	10
LCAW + Affine	0.129	0.024	91.88	7.69	11.74	104	9

Using *LCAW* reduced the %MS but did not significantly affect the distance metrics. This is because matching high-relevance areas requires the introduction of gaps in the more abundant zero-acceleration areas, reducing the number of matches but ultimately resulting in more accurate alignments.

Using *affine gap* had a minor effect on the fidelity metrics (%MS, FD, ED), but it has a pronounced effect on the gap distribution. It increased the average gap length from 5 to 11 without raising the total number of gaps. Additionally, gaps were consolidated from 20+ groups to just 10, reducing non-aligned regions and simplifying the interpretation of the alignment. These findings align with the benefits discussed in the previous section.

4.7 Parameter Tuning

The *MAD* value plays a significant role in determining the percentage of matched snapshots and the distances between DT and PT traces. However, another crucial aspect is the influence of gap penalties—specifically, the gap opening and extension penalties—on the three fidelity metrics (%MS, FD, ED). These parameters directly affect how the algorithm balances matches, mismatches, and gaps in the alignment process, impacting the assessment of whether the DT is faithful to the PT.

To better illustrate the behavior of our algorithm, we shift our focus from Execution 04 to Execution 01 (see Figure 4.8) in the elevator case study. Execution 01 presents a more favorable alignment as it exhibits similar acceleration curves without the initial delay observed in Execution 04. By excluding the initial delay, we can concentrate on the alignment of the acceleration curves and the synchronization between the movements of both elevators. This allows for a more straightforward analysis of how gap penalties influence fidelity and the overall quality of the alignment.

4.7.1 Gap tuning

To analyze the effect of algorithm parameters on trace alignment, we conducted an experiment varying the values of *MAD*, gap opening penalty (P_{op}), and gap extension penalty (P_{ex}). Specifically, *MAD* values ranged from 0.1 to 0.22 (increments of 0.04),

P_{op} from -3.0 to -0.5 (increments of 0.5), and P_{ex} from -2.0 to 0.0 (increments of 0.1), resulting in 504 alignments.

Figure 4.5 presents the results of these 504 alignments for scenario (4-0-4), sorted by the %MS along the X-axis. The plots show the values of *MAD*, P_{op}, P_{ex}, %MS, FD, and ED. A noticeable breakpoint divides the results into two regions, with the right side shaded in pink, showing where the number of matched snapshots grows significantly.

The results indicate that P_{ex} is the most influential parameter. When P_{ex} is between -0.5 and 0.0, the number of matched points exceeds 89%. Within this range, the *MAD* value correlates with the distance metrics, though not linearly. Small changes in *MAD* may alter alignments by including additional snapshot pairs, which has little effect on the Euclidean distance (which remains between 0.024 and 0.026) but can impact the Fréchet distance. Conversely, the P_{op} value does not significantly affect the results within the range $[-3, 0]$.

Our chosen parameter values, $(P_{op}, P_{ex}) = (-1.0, -0.1)$, yield stable alignments that allow for consistent use of the distance metrics.

4.7.2 *MAD* Effect

In Section 4.6, we defined three key metrics to evaluate the quality of an alignment: the Fréchet and Euclidean distances between traces and the percentage of matched snapshots (%MS). Intuitively, an alignment with smaller distances and a larger %MS indicates a closer match between the PT and DT. For a fully faithful DT, these metrics would result in distances of 0.0 and a %MS of 1.0.

As mentioned earlier, these metrics depend heavily on the choice of the *MAD* value. The *MAD* should be set to maximize %MS while minimizing the Fréchet and Euclidean distances. However, this value must be chosen carefully: a *MAD* that is too small may exclude valid matches, while one too large may lead to meaningless alignments.

As discussed in Section 4.5, the *MAD* value is typically set to 2 or 3 times the accuracy of the instrument used to measure the relevant attribute. In the elevator case study, we only consider the acceleration attribute, and since the accelerometer has an accuracy of $0.05\ m/s^2$, we select a *MAD* value of 0.15. With this value, the fidelity metrics for the alignment are: %MS = 0.94, Fréchet Distance (FD) = 0.12, and Euclidean Distance (ED) = 0.049.

4.7.3 Summary

To summarize, the alignment algorithm's parameters—P_{op}, P_{ex}, *LCAW*, and *MAD*—must be carefully set to achieve optimal fidelity. For this case study:

- The gap opening penalty (P_{op}) and extension penalty (P_{ex}) can be set to -1.0 and -0.1, respectively, following BLAST alignment recommendations.
- The *LCAW*, which penalizes snapshots in low-entropy areas (e.g., acceleration values below $0.05\ m/s^2$), is set to 0.005 (as described in Section 4.5).
- The *MAD* value is set to three times the accelerometer's accuracy, i.e., 0.15.

4.8 Assessing Fidelity

This section introduces how the fidelity metrics defined in previous sections—%MS, FD, and ED—can be used to establish indicators of the fidelity of a DT with respect to its PT. Furthermore, we show how these metrics enable the comparisons between different DTs and the impact of anomalous and erratic behavior on the fidelity metrics and indicators.

4.8.1 Fidelity indicators

The degree of fidelity of a DT relative to its PT can be determined based on the values of the three metrics. However, before drawing any conclusions on the level of fidelity, it is essential to ensure that the alignment is sufficient to make these measurements meaningful.

Guidelines for Interpreting Fidelity.

- **Low Alignment** (%MS < 90% ($\pm 2\%$)). If the %MS is below 90%, the traces could not be properly aligned, making the metrics unreliable. Consequently, no faithful behavior between the DT and PT can be expected in such cases.
- **Moderate Alignment** (90% $\leq$ %MS $\leq$ 95% ($\pm 2\%$)). In this range, the alignment is considered low but still adequate for assessing the distance metrics. Depending on the application, FD or ED may be more relevant in evaluating the DT's fidelity. For instance, FD would be critical when capturing peak deviations, while ED is more useful when the average decision over time is of interest.
- **High Alignment** (%MS > 95% ($\pm 2\%$)). An alignment above 95% is generally considered strong, and the distance between traces determines the fidelity. In such cases, FD and ED should be examined closely to assess the DT's faithfulness to replicating the PT.

It is essential to notice that these thresholds depend on the specific application and must be defined by the end user. In our examples, we have used the thresholds commonly found in general engineering environments (Snow 2003). It is important to note that we are assuming a maximum permissible error (MPE) of 2% (Snow 2003) to assess the percentage of MS since thresholds are often not entirely accurate.

Example: elevator - Execution 01. In the elevator example, Execution 01 yields a %MS of 94%, FD of 0.12, and ED of 0.05:

- **Alignment Sufficiency:** The %MS of 94% is within the acceptable range, making the distance metrics meaningful.
- **Euclidean Distance:** At 0.05, this value is below the accuracy of the measuring instrument (0.05 m/s²), meaning the average deviation between matched snapshots is negligible and acceptable.
- **Fréchet Distance:** The FD of 0.12 m/s^2, representing the maximum deviation between the two traces, must be evaluated based on the application's tolerance for deviation. For some applications, this may be acceptable, while for others, it may be too large depending on how critical peak deviations are.

Ultimately, these decisions rely on the specific requirements of the system being modeled and the level of fidelity needed for a given application.

4.8.2 Multi-fidelity Digital Twins

The concept of *Multi-fidelity Digital Twins* (Arrieta 2021) has gained atraction as a strategy to balance computational cost and model accuracy. In this approach, various DTs, each with a different level of fidelity, are used to simulate the same PT. This technique is particularly valuable when many high-fidelity models are too resource-intensive to be used consistently, and lower-fidelity models can provide "good-enough" simulations at a fraction of the cost. The challenge, however, lies in determining which DTs are sufficiently faithful to the PT for a given task, and this is where our approach is useful.

To illustrate how our approach can be used to compare the fidelity of two DTs of the same PT, we developed an additional lower-resolution model for the elevator case study using UML. We employed the tool (Gogolla, Büttner, and Richters 2007), which allows executing the UML specifications (Büttner and Gogolla 2014). The USE model of the elevator is available from (Muñoz et al. 2023a) and employs a simple algorithm to compute the acceleration, as opposed to the more complex algorithm used by the *Elevate* simulator.

Figure 4.6 provides a comparison of the high- and low-resolution models across varying *MAD* values in the range [0.02, 0.3]. The high-fidelity model consistently performs better in terms of alignment (%MS) and distance metrics (FD and ED). For instance, the high-fidelity model stabilizes at a %MS of 91% when the MAD value reaches $0.15\ m/s^2$, meaning that the alignment is solid and stable at this threshold. Additionally, both FD and ED reach a plateau, indicating that no further improvement in alignment is possible by increasing the *MAD* value. This stability suggests that the high-fidelity DT successfully captures the behavior of the PT with minimal deviations.

In contrast, the lower-resolution USE model continues deteriorating as the *MAD* value increases. Even at its best, the low-resolution model's %MS does not exceed 90%, meaning that it struggles to achieve an alignment good enough to make its distance metrics reliable.

In this case, the high-fidelity DT is considered faithful, as it achieves acceptable values across the metrics: %MS = 0.91, FD = 0.10, ED = 0.024. These values indicate that the high-fidelity DT accurately reproduces the PT's behavior, with an acceptable average deviation (ED) and peak deviation (FD). For a *MAD* value of 0.12 (about twice the accuracy of the accelerometer), the alignment covers 91% of the snapshots, a high percentage, and the distances indicate the deviations are within acceptable bounds.

In contrast, the low-resolution DT fails to achieve a meaningful alignment for *MAD* values over 0.12, and even then, the percentage of matched snapshots remains below 90%. For example, Figure 4.7 illustrates the poor alignment at *MAD* values of 0.12 and 0.15, showing that the USE model fails to capture the nuances of the PT's acceleration curves. This further highlights the need for improvements in the lower-resolution model, specifically by increasing the precision of the simulated acceleration.

Gap-related metrics—such as the number of individual gaps and the mean length of the gap sequences—provide further insights. For example, in the alignment of the high-fidelity DT at *MAD* = 0.12 (shown in Figure 4.8), 91% of the snapshots are matched, and there are 14 gap sequences, totaling 114 gaps. The mean length of the gap sequences is 8, corresponding to an average delay of nearly one second. This delay is consistent with the observed behavior of the DT, which slightly lags behind the PT.

4.8.3 Additional Synthetic Scenarios

To further illustrate the robustness and potential of our approach, we conducted experiments on two synthetic scenarios derived by modifying the actual measurements from the elevator case study (4-0-4). These scenarios demonstrate the algorithm's ability to detect anomalies and erratic behavior and how this behavior impacts the fidelity metrics.

Acceleration Anomalies in the PT. In this scenario, we introduced a set of anomalous acceleration patterns into the PT trace. The algorithm successfully identified these patterns as mismatches or gaps, highlighting the locations of the anomalies within the trace. Figure 4.9 displays the alignment with the anomalies.

Originally, the fidelity metrics for the unaltered traces were %MS = 94.2, FD = 0.12, and ED = 0.05. However, after incorporating the anomalies, the %MS dropped to 75.07, indicating a significant reduction in alignment quality. Interestingly, the distance metrics (FD and ED) remained unchanged because the added snapshots with anomalies were not aligned, and our algorithm only calculates the distances between matched snapshots. This shows that while the alignment percentage suffered, the actual differences between the aligned sections did not increase.

Random Noise in the Working Range In this experiment, we replaced the PT trace with random noise within the expected working range of the acceleration values. When attempting to align this random noise trace with the original sequence, the algorithm struggled to find meaningful matches, resulting in poor alignments. Figure 4.10 illustrates this misalignment.

For the original PT trace, the fidelity metrics at the chosen *MAD* were %MS = 94.2, FD = 0.12, and ED = 0.05. In contrast, the random noise trace yielded drastically different results, with the %MS plummeting to 22.04 and 76.2% of the snapshots marked as mismatches. Additionally, the distance metrics increased, with FD rising to 0.15 and ED to 0.07. These values clearly show the algorithm's effectiveness in distinguishing a meaningful trace from one with random noise, as evidenced by the marked deterioration in fidelity metrics.

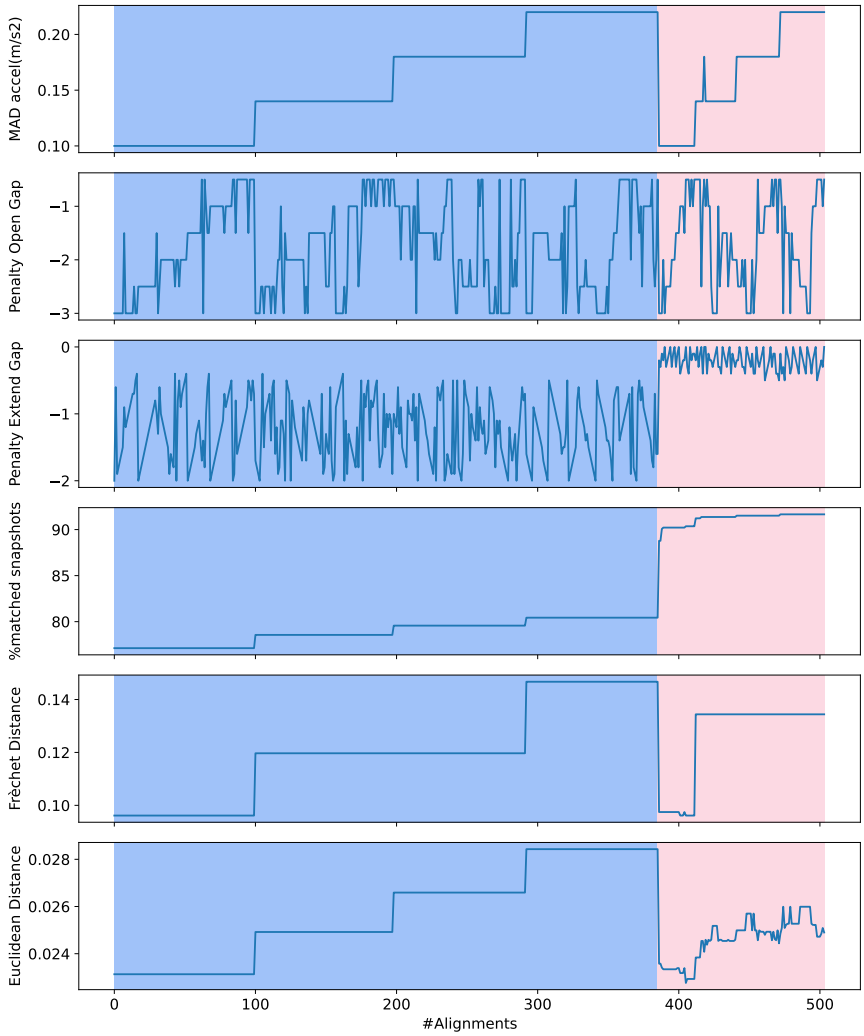


Figure 4.5: Results for the 504 alignments of scenario (4-0-4) sorted by increasing *%MS* in the X-axis. From top to bottom: *MAD*; P_{op}; P_{ex}; *%MS*; Fréchet distance *FD* and Euclidean distance *ED*.

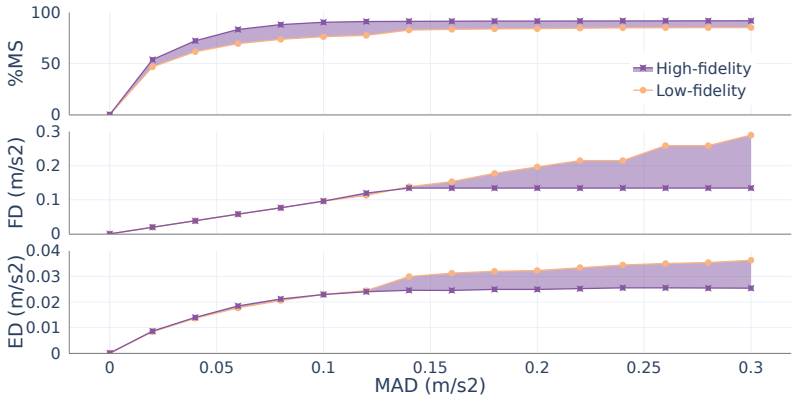


Figure 4.6: Fidelity comparison of two DTs for scenario (4-0-4). Execution 01.

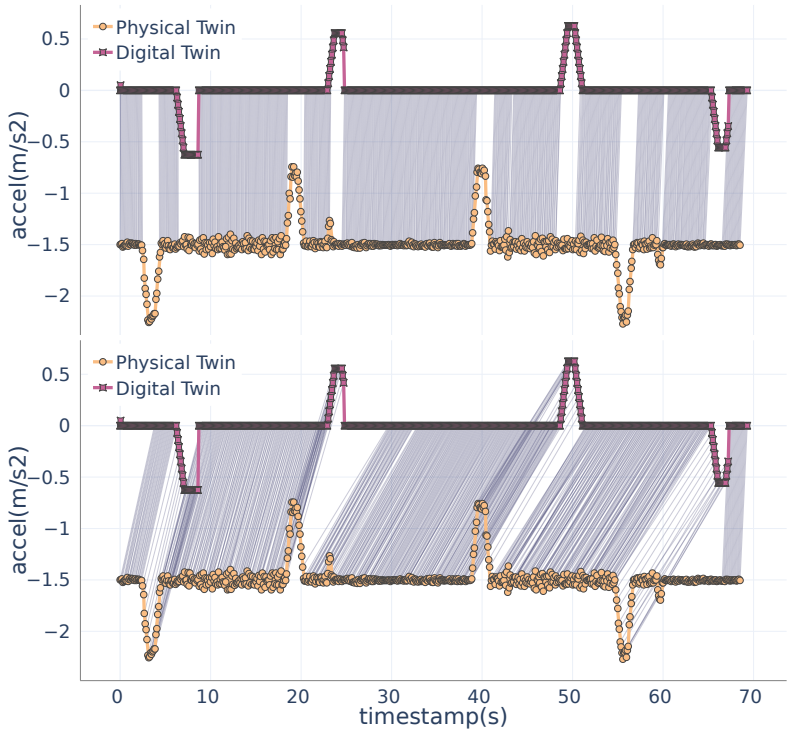


Figure 4.7: Alignments for the low-resolution model with *MAD* values 0.12 (top) and 0.15 (bottom).

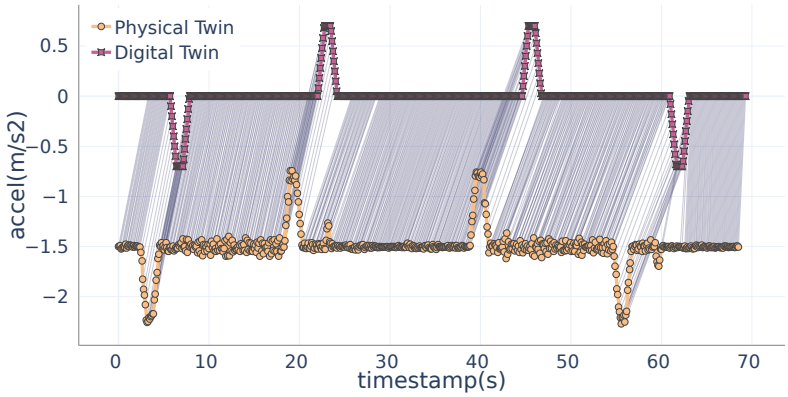


Figure 4.8: Alignment of the high-resolution model for *MAD*=0.12. Execution 01.

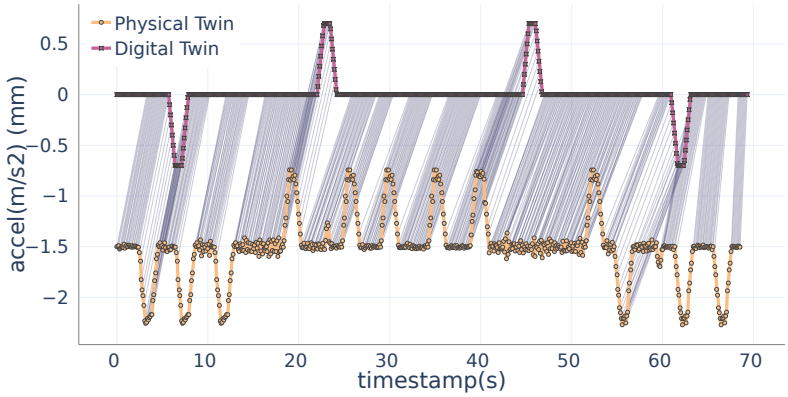


Figure 4.9: Alignment of the DT scenario (4-0-4) with an anomalous PT trace *MAD*=0.15.

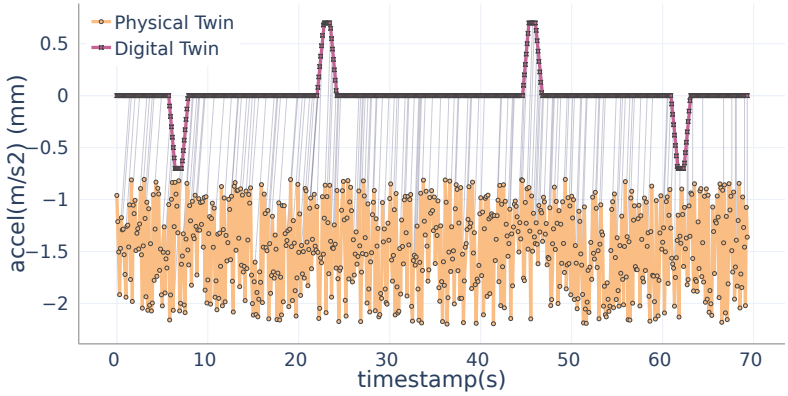


Figure 4.10: Alignment of the DT scenario (4-0-4) with random noise in the working range. *MAD*=0.15.

5

DEMONSTRATION CASES

In this chapter, we apply our algorithm to three additional CPSs to further explore our proposal, complementing the previous example. First, we evaluate an **incubator** system with two models, differing in accuracy and computational cost. This helps us determine which model is optimal for specific operational scenarios. Next, we analyze a **robotic arm**, showing how the approach handles snapshots with multiple parameters, such as servo data, and test how accurately the robot follows a model of desired behavior. Finally, we demonstrate the method's ability to handle traces involving Boolean and enumerated data types in a third case study using a **Lego car**.

5.1 Incubator: Model Comparison

Description. At Aarhus University in Denmark, researchers are developing a DTS using an incubator as a foundational case study (*Example Digital Twin: The Incubator* 2023; Feng et al. 2021). The incubator system, as depicted in Figure 5.1, is a simple yet representative thermal system. The primary goal of this DT is to maintain the incubator at a specific temperature, regardless of its contents.

The **physical twin** of the incubator is an insulated box with a heatbed controlled by a Raspberry Pi. The system includes communication software, a controller, and simulation models. Two DHT22 sensors[1] monitor the internal temperature, while an additional sensor tracks the room temperature. The sensors provide readings every two seconds, while the

[1] `https://components101.com/sensors/dht22-pinout-specs-datasheet`

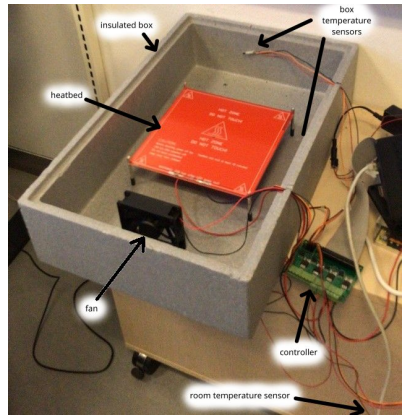


Figure 5.1: Incubator in the University of Aarhus (Denmark).

controller operates every three seconds. The controller averages the internal readings to assess the incubator's temperature and regulate it accordingly.

Two simulation models were created to serve as the **digital twin**, capable of predicting the temperature inside the incubator. One model has two free parameters, while the other has four:

- The **2-parameter model** for the incubator is a straightforward linear model that predicts the internal temperature using two key parameters: the *heat transfer coefficient* and *thermal resistance*. These parameters are assumed to remain constant throughout the operation. To calibrate the model, experimental data from the physical incubator system was used, and the differences between predicted and actual temperatures were minimized using a least-squares method. While this model is generally accurate in predicting the incubator's temperature, it has limitations in capturing the system's non-linear dynamics and transient behavior.
- The **4-parameter model**, on the other hand, is a more advanced, non-linear model. It considers four key parameters: this is the *heat transfer coefficient*, *thermal resistance*, *thermal capacitance*, and a *time constant*. As the 2-parameter model, it was calibrated using experimental data and a least-squares method. However, this model offers better accuracy, particularly when capturing the system's transient behavior. The trade-off is that it requires more computational resources and is more complex than the simpler 2-parameter model.

For further details on the incubator system and its implementation, refer to (Feng et al. 2021) and (*Example Digital Twin: The Incubator* 2023).

Scenarios. The incubator's goal is to maintain the temperature within specific boundaries. This means that the temperature inside the incubator will fluctuate within a range depending on the heating and cooling cycles:

- **Heating time 3 s - Heating gap 2 s (Ht3Hg2).** Figure 5.2 shows the heating and cooling patterns for the first scenario, labeled as Ht3Hg2. In this scenario, the heater is activated for 3 seconds, then turned off for 2 seconds. If the target temperature has not been reached, the cycle repeats. This short heating and cooling cycle helps keep the temperature within the 30–35°C range, both in the PT and the two DTs. The frequent, short heating periods allow for tight control over the temperature, ensuring that it remains within the desired limits.
- **Heating time 30 s - Heating gap 20 s (Ht30Hg20).** Figure 5.3 displays the heating pattern for the second scenario, Ht30Hg20. Here, the heater is turned on for 30 seconds if the temperature falls below 35°C and turned off for 20 seconds. If the temperature remains below the upper limit after this period, the heater reactivates. The longer intervals between adjustments lead to slower decision-making and can cause the temperature to drift outside the desired range. This results in less precise temperature control, and in the two-parameter model, a sawtooth pattern emerges, highlighting the model's inability to smooth out the temperature changes caused by the longer control intervals.

Fidelity Assessment. The incubator uses two DHT22 sensors (Components101 2018) to measure the internal temperature, with an accuracy of $\pm 0.5°C$. To ensure a sufficient level of fidelity, the *MAD* should be within the range of 1 to 1.5°C (cf. Section 4.7.2).

In the **Ht3Hg2 scenario**, the heater activates and deactivates at very short intervals, making the two-second cooling period almost imperceptible. This results in a challenging transition between heating and cooling states, where non-linear behaviors occur. Both the 2-P and 4-P models struggle with this transition. The alignment statistics show that neither model can faithfully reproduce the incubator's behavior with an *MAD* in the range [1, 1.5]°C, as shown in Table 5.1 and Fig. 5.4. Specifically:

- **2-P Model:** *%MS* ranges from 0.86 to 0.89, *FD* ranges from 0.433 to 0.712, and *ED* ranges from 0.122 to 0.180.
- **4-P Model:** *%MS* ranges from 0.89 to 0.92, *FD* ranges from 0.591 to 0.734, and *ED* ranges from 0.135 to 0.178.

Both models fail to accurately capture the transitional behavior, as evidenced by the alignment being below 90%.

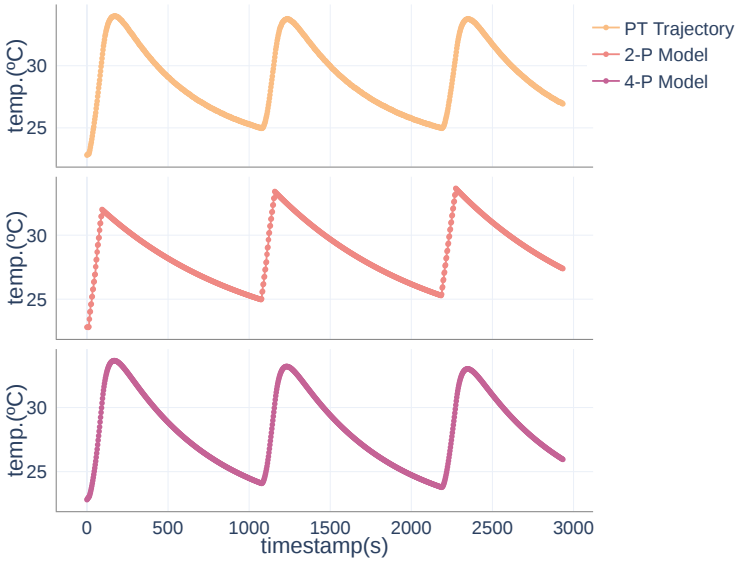


Figure 5.2: Traces for scenario Ht3Hg2 (In order: Physical Twin, 2-P model, 4-P model).

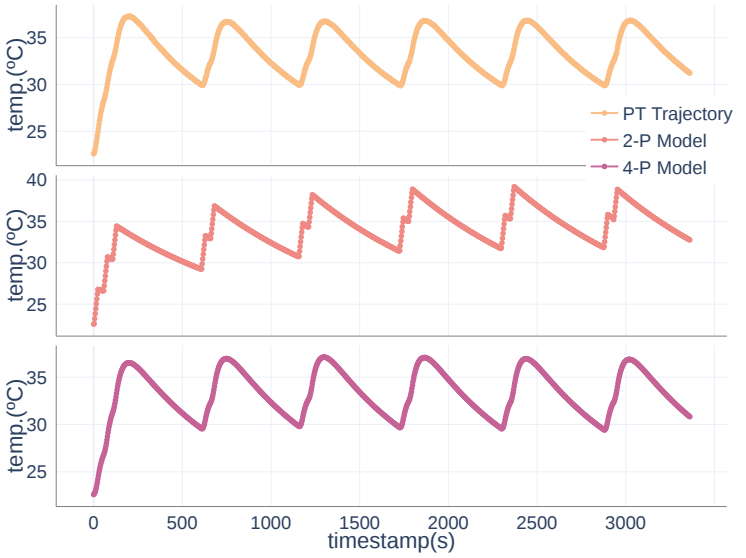


Figure 5.3: Traces for scenario Ht30Hg20 (In order: Physical Twin, 2-P model, 4-P model).

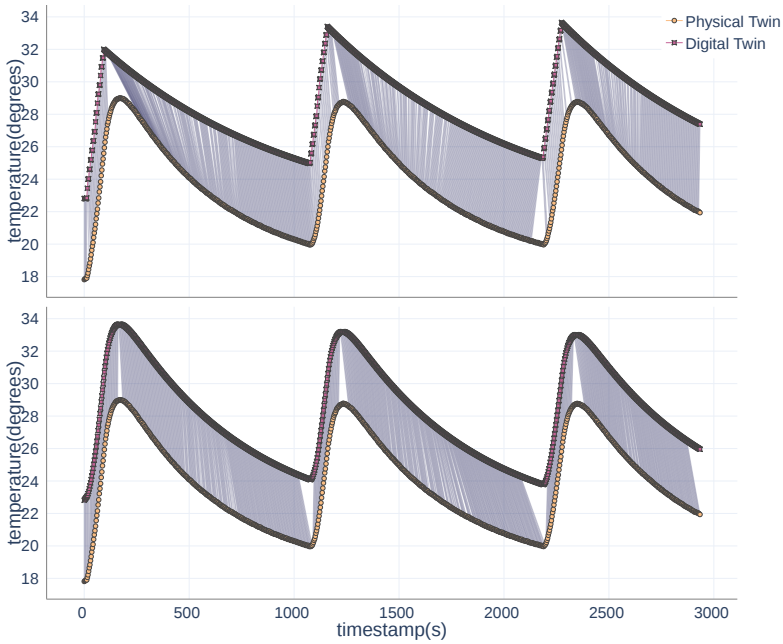


Figure 5.4: Alignments for the 2-P model (top) and the 4-P model (bottom) in scenario Ht3-Hg2 with *MAD=1.20°C (Note: the PT trajectory is displaced 5 degrees for improved visualization).*

Table 5.1: Fidelity results for scenario Ht3Hg2.

	2-P Model			4-P Model		
MAD ($^{\circ}C$)	*%MS*	*FD*	*ED*	*%MS*	*FD*	*ED*
0.2	79.97	0.19	0.03	78.54	0.19	0.03
0.4	82.53	0.25	0.05	83.96	0.39	0.06
0.6	83.75	0.31	0.07	87.12	0.54	0.09
0.8	85.29	0.43	0.10	88.55	0.51	0.11
1	86.41	0.49	0.12	89.78	0.59	0.13
1.2	88.04	0.61	0.15	90.90	0.67	0.15
1.4	89.17	0.71	0.18	91.82	0.73	0.17
1.6	90.50	0.77	0.21	92.44	0.77	0.19
1.8	91.50	0.90	0.24	93.36	0.82	0.21
2	92.54	0.96	0.27	94.17	0.88	0.24

Table 5.2: Fidelity results for scenario Ht30Hg20.

	2-P Model			4-P Model		
MAD ($^{\circ}C$)	*%MS*	*FD*	*ED*	*%MS*	*FD*	*ED*
0.2	54.14	0.17	0.04	89.65	0.19	0.03
0.4	57.80	0.39	0.08	93.84	0.34	0.05
0.6	62.97	0.59	0.13	95.71	0.56	0.07
0.8	67.43	0.78	0.17	96.52	0.65	0.08
1	72.25	0.89	0.22	97.05	0.65	0.10
1.2	75.37	0.91	0.28	97.59	0.65	0.11
1.4	77.25	0.97	0.31	98.03	0.65	0.12
1.6	80.19	1.06	0.37	98.21	0.65	0.13
1.8	81.89	1.22	0.42	98.39	0.65	0.14
2	83.76	1.44	0.47	98.66	0.65	0.15

In the **Ht30Hg20 scenario**, the longer activation and deactivation periods give both models a better opportunity to predict the cooling phase.

- **2-P Model:** The 2-P model still struggles with transitions and displays a sawtooth pattern that does not match the actual behavior. The statistics for this scenario indicate less than 80% alignment, with the following ranges for *MAD* values in [1, 1.5]°C: *%MS* ranges from 0.72 to 0.77, *FD* ranges from 0.890 to 0.974, and *ED* ranges from 0.228 to 0.319.
- **4-P Model:** The 4-P model demonstrates better performance in the Ht30Hg20 scenario due to its reduced sensitivity to long cooling gaps and better transition reproduction. It achieves a remarkable 98% alignment of snapshots, with metrics as follows: *%MS* ranging from 0.97 to 0.98, *FD* with a value of 0.65, and *ED* from 0.100 to 0.127,as shown in Table 5.2. This indicates that the 4-P model closely emulates the incubator's behavior, deviating on average by only 0.1°C, with peak differences of just 0.65°C, and successfully matching over 97% of the snapshots.

Figure 5.6 illustrates the comparative trends between the two models for scenario Ht30-Hg20. The 4-P model reaches a plateau in all three metrics near 1°C, while the 2-P model continues to increase the percentage of matched snapshots but at the cost of greater average and peak distances.

In summary, the 2-P model struggles to accurately replicate the incubator's behavior, particularly in scenarios with rapid transitions. Conversely, the 4-P model excels in scenarios with longer transitions but falters with shorter heating and cooling intervals. Evaluating

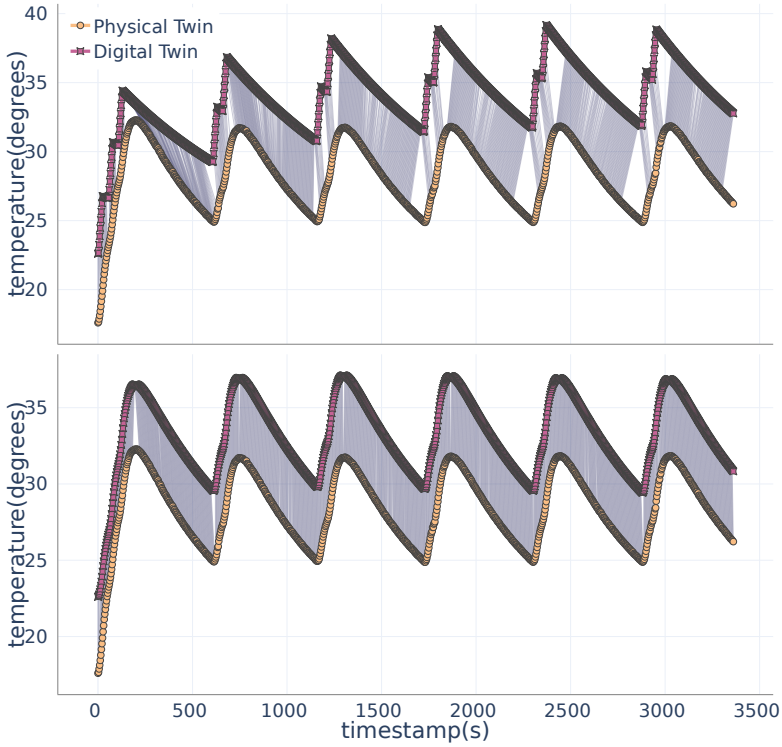


Figure 5.5: Alignments for the 2-P model (top) and the 4-P model (bottom) in scenario Ht30-Hg20 with *MAD=1.20°C (Note: the PT trajectory is displaced 5 degrees for improved visualization).*

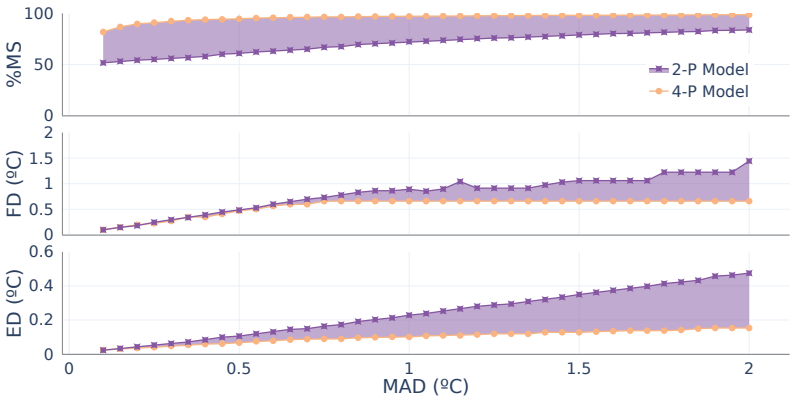


Figure 5.6: Fidelity metrics comparison of the incubator for scenario Ht30-Hg20.

model fidelity involves analyzing alignment and metric values, with higher %MS and lower ED and FD indicating better fidelity. However, determining the most suitable model for a specific problem requires considering whether the model accurately reproduces relevant behavior sequences, even if it does not achieve the highest overall fidelity.

Our tool provides both local and global assessments of different characteristics that influence fidelity. However, the domain expert must make the final decision to determine the most appropriate model for their specific problem.

5.2 Robotic Arm: Multiple Numerical Attributes

Description. This case study explores the use of our proposal in a scenario where the **DT serves as the oracle**, helping to determine whether the PT faithfully replicates the behavior specified by the DT. Specifically, we aim to evaluate whether a robotic arm can meet the operational requirements set by the user. These requirements are defined through engineering models (Lee and Sirjani 2018) that outline the expected behavior in different scenarios. The primary goal is to assess whether a particular robotic arm is suitable for transporting small, lightweight objects, similar to the crane example discussed in (Zhidchenko et al. 2018).

Figure 5.7: Braccio robotic arm.

There are several robotic arms on the market capable of fulfilling this role, including options like the ABB GoFa CRB 15000 (ABB 2024), the Niryo Ned (Niryo 2024), and the Arduino Tinkerkit Braccio (Tinkerkit 2021). These robots typically feature six servos that control their movements: the base orientation (s1), shoulder position (s2), elbow (s3), vertical wrist position (s4), wrist rotation (s5), and the gripper (s6). For this case study, we focus on the **Arduino Braccio robotic arm** (Tinkerkit 2021), a low-cost option successfully applied in various fields (Masood and Haghshenas-Jaryani 2021). Due to its affordability, we are testing whether it can meet the needs of two specific scenarios with relatively undemanding requirements.

Scenarios. Both the PT and DT of the robotic arm operate using sequences of servo positions as input. By calibrating these sequences to meet specific user goals, the robotic arm can perform various tasks. To test whether this low-cost robotic arm could meet the required operational standards, we evaluated it in two relatively simple scenarios.

- **Simple Moves.** In this first scenario, the robotic arm executes a basic sequence of four commands at a low speed. The arm starts with all servos fully extended and the gripper facing upward. This sequence involves simple, coordinated movements of two or three servos simultaneously. The following sequence is performed:

```
Position positions[] = {
    Position(90, 90, 180, 172, 90, 10),
    Position(90, 55, 170,  86, 90, 10),
    Position(0,  90,  90,  90, 90, 43),
    Position(90, 90,  90,  90, 90, 73)
};
```

- **Pick & Drop.** In the second, more complex scenario, the arm performs a series of movements at a higher speed. Starting with all servos extended and the gripper facing upward, the arm is tasked with grasping a distant object, raising it 30 cm, moving it 50 cm to the right, lowering it, releasing the object, and returning to its initial position. The commands for this sequence are as follows:

```
#define GRIPPER_OPEN 10
#define GRIPPER_CLOSED 73

Position positions[] = {
    Position(90, 90, 180, 172, 90, GRIPPER_OPEN),
    Position(90, 90, 180, 172, 90, GRIPPER_CLOSED),
    Position(90, 55, 170,  86, 90, GRIPPER_CLOSED),
    Position(90, 90,  90,  90, 90, GRIPPER_CLOSED),
    Position( 0, 55, 170,  86, 90, GRIPPER_CLOSED),
    Position( 0, 55, 170,  86, 90, GRIPPER_OPEN),
};
```

  This scenario tests the arm's ability to perform more dynamic actions, including lifting and moving objects. Both scenarios assume very light loads, simplifying the task by minimizing their impact on the arm's speed and trajectory. For more detailed behavior models of these servo movements, see the companion repository (Muñoz et al. 2023a).

Fidelity Assessment. In the *Simple Moves* scenario, the robotic arm moves slowly, using no more than three servos at a time. This allows for smooth transitions and stationary servo positions, closely mirroring the behavior outlined in the engineering model. As a result, the system achieves high fidelity, with over 90% of the snapshots matching. For a MAD range between 2 and 3 degrees (since the servo accuracy is 1 degree), the fidelity metrics are as follows: %MS=[90.98, 91.59], FD=2.28, and ED=0.23 (see Table 5.3). These metrics demonstrate that the robotic arm is well-suited for replicating the specified behavior in this scenario.

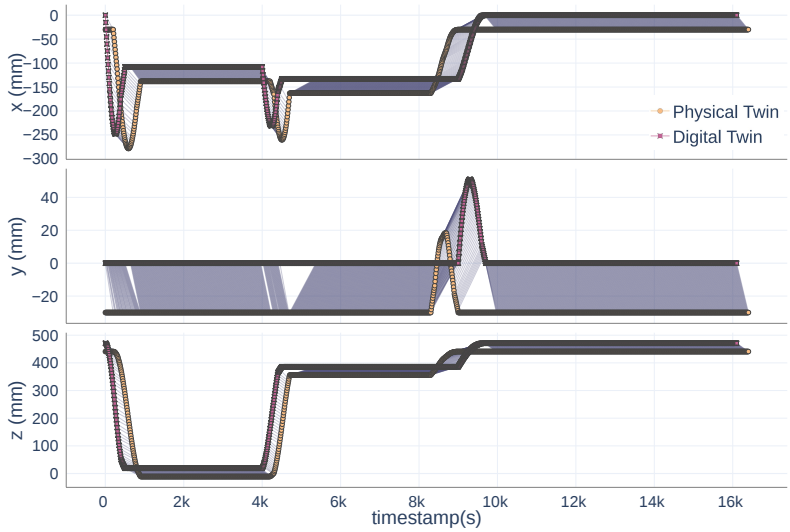


Figure 5.8: X-Y-Z alignments for the *Simple Moves* scenario, *MAD*=15 mm (Note: the PT trajectory is displaced 30 mm for improved visualization).

Table 5.3: Fidelity results for scenario *Simple Moves*.

Axis position (degrees)				Grip's Coordinates (mm)			
MAD	%MS	FD	ED	*MAD*	%MS	FD	ED
0.2	77.39	0.22	0.01	3	84.76	3.93	0.09
0.8	84.52	1.32	0.06	6	90.37	7.88	0.41
1.4	90.49	2.03	0.15	9	91.10	9.64	0.47
2.0	90.98	2.27	0.17	12	91.22	11.47	0.51
2.6	91.40	2.28	0.22	15	91.46	11.47	0.57
3.2	91.59	2.28	0.23	18	91.59	13.19	0.65

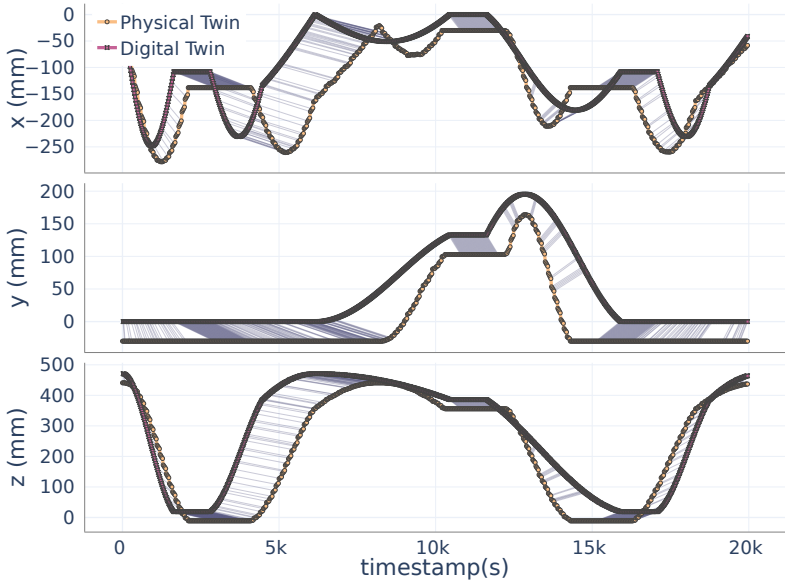


Figure 5.9: X-Y-Z alignments for the Pick&Drop scenario, *MAD*=15 mm (Note: the PT trajectory is displaced 30 mm for improved visualization).

However, in the *Pick & Drop* scenario, which involves faster movements and concurrent activation of multiple servos, the performance decreases. The robot struggles to maintain fidelity under these more complex conditions, leading to reduced metric values: %MS=[78.07, 78.78], FD=[2.72, 2.79], and ED=[1.63, 1.92] (see Table 5.4).

To better understand this decline, we computed the coordinates of the gripper and analyzed the alignments in both scenarios. The results, which are consistent with the servo values, show that the robot performs well in the first scenario, as illustrated in Figure 5.8, but performs less effectively in the second. Specifically, for a MAD value of 15 mm

Table 5.4: Fidelity results for scenario *Pick&Drop*.

Axis position (degrees)				Grip's Coordinates (mm)			
MAD	%MS	FD	ED	*MAD*	%MS	FD	ED
0.2	0.20	0.2	0.15	3	59.35	4.13	1.44
0.8	51.95	1.36	0.66	6	73.47	8.44	2.47
1.4	75.07	1.91	0.98	9	78.17	12.21	3.31
2.0	78.07	2.69	1.31	12	79.47	12.12	3.72
2.6	78.47	2.72	1.63	15	81.08	14.85	4.35
3.2	78.78	2.79	1.92	18	82.08	14.85	5.13

(since the robot's coordinate accuracy is 5 mm (Masood and Haghshenas-Jaryani 2021)), Figure 5.9 illustrates how the robot remains stationary for longer than required, resulting in approximately 20% of gaps in the alignment. These gaps occur when the physical twin (PT) fails to respond quickly enough, leading to alignment issues. Furthermore, the Braccio's transition movements are slower than those of the engineering model. Despite this, the movements remain within the acceptable MAD range, enabling the algorithm to align most of the snapshots.

In summary, the Braccio robotic arm would require a faster response time to accurately emulate the behavior established by our engineering model.

The snapshot comparison process is similar to evaluating a single attribute, where differences within the MAD range are considered a match. However, when comparing multiple attributes (in this case, six servos), the process becomes stricter. If even one servo is out of alignment, it could suggest that the robot is performing an entirely different task. However, in cases where specific attributes are less critical, the comparison function could be adjusted to allow for some flexibility. We could reduce the level of similarity to some extent instead of considering it a mismatch. Alternatively, each attribute could be analyzed individually to detect isolated inconsistencies.

In both this and the previous example, low-complexity regions and associated LCAW were not considered. These scenarios do not include long periods of inactivity, which could affect the alignment by increasing the match rate in less relevant areas. This could distract the algorithm from finding alignments in the most important parts of the traces.

5.3 Lego Car: Boolean and enumerations

Description. The ***Lego NXT Car*** served as a key example in the prototype of our approach, first introduced in (Muñoz et al. 2022). This robot features multiple built-in sensors, making it a valuable tool for demonstrating the capabilities of DT technology.

Figure 5.10: The Lego car.

Figure 5.11 depicts a snapshot of the robot with its properties of interest. It has a pose-provider that returns its current planar coordinates and directional angle, represented by `xPos` and `yPos`. In addition, it includes an ultrasonic sensor to measure `distance` to the closest object in front of it, a `light` sensor that identifies the color of the ground below, and two touch sensors that function as `bumpers` to detect collisions. The robot also records its current `speed` and whether it `isMoving`. Its programming allows it to set states based on the action it is executing, providing the current `action` being performed.

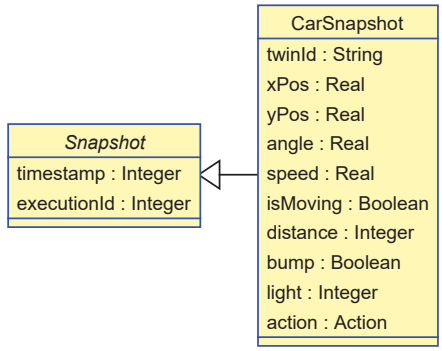


Figure 5.11: UML Class of the NXT Lego Car Snaphot.

To simulate the car's behavior, we developed a **DT using the USE modeling tool** (Gogolla, Büttner, and Richters 2007). This simulation revealed that the real car's behavior was unreliable, mainly due to sensor inaccuracies, controller delays, and mechanical slack, preventing it from faithfully replicating consistent trajectories. Despite these issues, this example is highly illustrative of cases where synthetic traces—containing artificially generated variations—are used as the PT, as depicted in Figure 5.12.

Scenario. Since these traces are artificially created, the scenario was designed to demonstrate how parameters of various datatypes (such as enumerations, booleans, and numeric values) interact. In this scenario, the car follows a circuit with four turns, creating a rectangular trajectory. Both twins follow the same path, although we found that the physical twin increases its speed during some parts, slightly altering its theoretical trajectory therefore affecting some of its attribute values.

Fidelity Assessment. To align the snapshots, Boolean attributes like `bump` and `isMoving`, as well as enumerated types such as `action`, must have identical values. For numerical attributes, the MAD for each sensor is set at three times the sensor's accuracy, as outlined in Table 5.5. If there is any discrepancy in non-numerical values or numerical values beyond their respective MADs, the similarity function will return 0, indicating that the snapshots do not match.

In this scenario, the PT follows a slightly different path, increasing its speed by 1.5 at a certain point. However, since this increase remains within the MAD, it does not affect the alignment. A stricter MAD could pinpoint this variation, but the current threshold considers the difference acceptable. In contrast, inconsistencies were detected in the `bump` attribute, with two collisions recorded by the end of the trace, and in the `action` attribute, where two rotations were performed later than the DT predicted. These discrepancies introduce gaps in the alignment, as Boolean and enumerated types require exact matches.

Table 5.5: MAD values for Lego NXT Car.

Parameter	Accuracy	MAD
xPos, yPos	$0.50\ cm$	$1.50\ cm$
angle	$0.05\ degrees$	$0.15\ degrees$
speed	$0.50\ cm/s^2$	$1.50\ cm/s^2$
isMoving	-	-
distance	$0.50\ cm$	$1.50\ cm$
bump	-	-
light	0.05	0.15
action	-	-

As a result, the alignment yielded a %MS of 70%, a FD of 1.58, and an ED of 0.7. This indicates that 30% of the trace did not match due to inconsistencies in the `action` and `bump` attributes. However, the remaining behavior was replicated satisfactorily. Further analysis of individual attributes could measure the fidelity of each one. This would conclude that `xPos` and `yPos` contribute the most to the increase in distance metrics, while attributes such as `speed` and `angle` are more faithful.

In the corresponding technical report (Muñoz et al. 2023c), additional examples of different executions highlight partial inconsistencies in individual attributes and how they affect fidelity metrics.

When evaluating the fidelity of snapshots involving Boolean or enumerated attributes, the approach mirrors that used for multiple numerical attributes—differences must remain within the MAD for matches. However, the presence of Boolean and enumerated attributes makes the alignment process more restrictive. A single discrepancy in non-numerical attributes results in a gap or mismatch, helping to pinpoint inconsistencies.

In the case of the Lego car, these strict constraints are crucial for determining whether sensors, such as the bumper, are functioning correctly. If a Boolean sensor like the bumper provides a different value in the DT or PT, this could indicate either a misconfiguration in the virtual environment or a malfunctioning sensor. Similarly, if the robot's position is accurate but the `action` value differs, it could reflect a logic mismatch between the DT and PT. This is why it is so important to require exact matches for non-numerical values.

Nevertheless, if Boolean or enumerated attributes were not critical to the evaluation, they could be excluded from the alignment process. Alternatively, the comparison function could be adjusted to reduce the level of similarity without causing a complete mismatch. However, if these attributes were essential to the system's state, any difference—even in one attribute—should prevent a match.

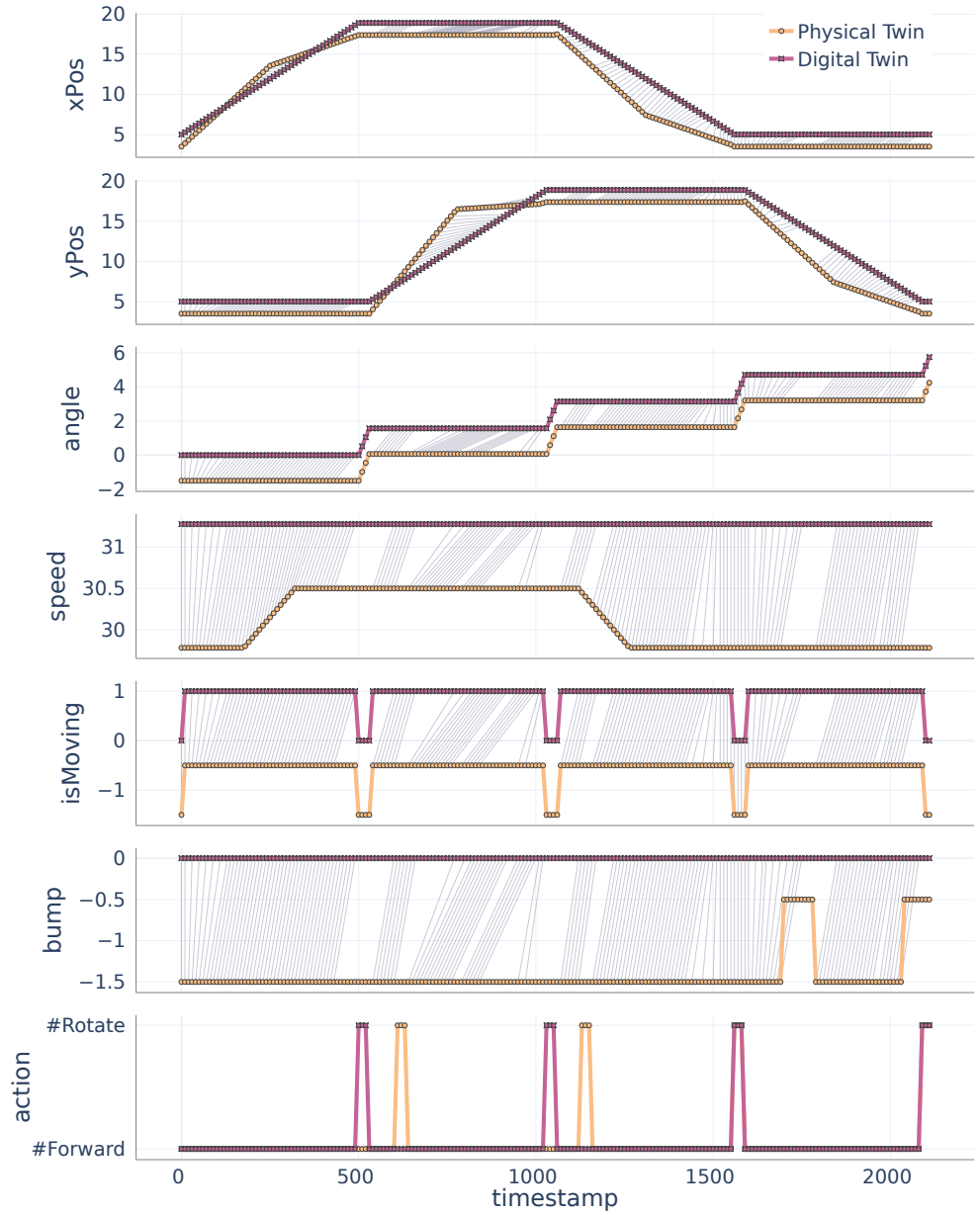


Figure 5.12: Alignments for the Lego NXT Car (*Note: the PT trajectory is displaced 1.5 units and the attributes* `distance` *and* `light` *were not included for improved visualization*).

6

EMPIRICAL EVALUATION

In this section, we further assess our proposal by addressing the following research questions (RQs):

RQ.1 How effective is our proposal in evaluating fidelity compared to existing approaches?

RQ.2 What are the time and space complexities of the different configurations of our algorithm?

RQ.3 How do the execution time and memory usage of our proposal compare to that of other related approaches?

To address RQ.1, we conducted a comparison study with two of the closest related works: (Sakoe and Chiba 1971) and (Lugaresi et al. 2022). We evaluated their expressiveness, ability to measure fidelity, and overall performance compared to our approach.

For RQ.2 and RQ.3, we performed experiments with varying trace lengths and applied regression analysis to estimate the time complexity of our algorithm. We also compared the execution time of our approach with that of the closest related works.

6.1 Comparison with other Proposals (RQ.1)

To demonstrate the differences between our algorithm and existing state-of-the-art approaches, we conducted a comparison using the elevator case study. We evaluated our algorithm alongside two of the closest related proposals. The first one (Sakoe and Chiba 1971) uses Dynamic Time Warping to align the traces, while the second one (Lugaresi et al. 2022) uses event-driven data analysis to align the traces and defines KPI-based fidelity metrics.

6.1.1 Dynamic Time Warping

Description. Dynamic Time Warping (DTW) (Sakoe and Chiba 1971) is a dynamic programming algorithm used to measure the similarity between two temporal sequences that may have varying speeds. Similar to our *SnapAligner* algorithm, DTW can be applied to any sequence alignment problem where a similarity distance between elements can be defined. Its goal is to compute an optimal alignment that minimizes the pairwise distances between corresponding elements in the sequences.

The Bellman Equation for DTW is:

$$D(i,j) = \mathsf{dist}(s_i, t_j) + \min\{D(i-1,j), D(i,j-1), D(i-1,j-1)\} \quad \text{(Eq. 6.1.1.1)}$$

In this equation, $D(i,j)$ represents the accumulated cost at the position (i,j) in the alignment matrix. The function dist(s_i, t_j) calculates the similarity distance between the i-th element of sequence s, and the j-th element of sequence t. The minimum function chooses the minimum cost path from three possible positions to reach (i,j), ensuring the optimal alignment.

DTW differs from NDW because it allows for one-to-many and many-to-one matches, assuming one sequence is a time-warped version of the other. This flexibility accounts for stretching (one-to-many), condensing (many-to-one), or maintaining equal speed (one-to-one) between sequences. Therefore, DTW does not support the notion of gaps, while NDW explicitly takes gaps into account and assigns a penalty per gap.

Ultimately, DTW aims to minimize the pairwise distances, and a smaller total distance indicates higher similarity between the sequences.

Comparison. To demonstrate the differences between DTW and our approach, we applied DTW to the same alignment presented in Subfigure 6.1b. The results, shown in Subfigure 6.1a, reveal two significant limitations of DTW.

First, DTW does not support gaps, meaning it forces the alignment of all elements, even those with very low similarity. For instance, it matches the PT's smooth braking patterns (at seconds 38 and 68) with zero-acceleration snapshots in the DT trace. This results in 100% matched snapshots, but the alignment becomes less informative because gaps and mismatches—critical for identifying differences and pinpointing anomalies—are ignored.

Second, DTW's one-to-many and many-to-one alignments are problematic for accurately comparing trace distances. For example, in Figure 6.1a, DTW aligns one snapshot 201 times, i.e., with 28.7% of the snapshots from the other trace, as included in Table 6.1.

Table 6.1: Dynamic Time Warping statistics for scenario (4-0-4). Execution 4.

Metric	DT	PT
Max times a snapshot is aligned	148	201
# Snapshots aligned more than once	16	9
Avg times a snapshot is aligned more than once	36.44	64.11
Std times a snapshot is aligned more than once	57.42	74.27
Total one-to-many alignments	567	567
Trace length	694	
Similarity score	17.00 m/s2	

This leads to unreliable results and fails to represent the actual similarity between the traces accurately. Some studies have highlighted the shortcomings of this approach, and various proposals include methods to mitigate this issue, e.g., (Li et al. 2020). This is also why our alignments are one-to-one and may contain gaps and mismatches.

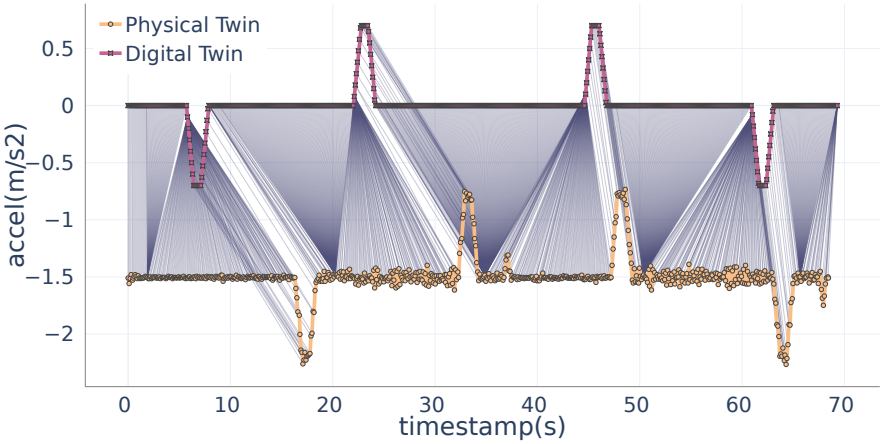
Moreover, several DTW variants have been developed to handle specific issues. Some aim to improve time complexity, speeding up computations (Salvador and Chan 2007), while others attempt to counteract overstretching and overcondensing issues (Fu et al. 2008; Sakoe and Chiba 1978). Despite their improvements, these solutions typically require fine-tuning parameters, and when not properly configured, they may overfit to particular data sets.

To avoid these issues in our study, we limit our comparison to the original DTW algorithm (Sakoe and Chiba 1971). This ensures we use an implementation that guarantees optimal solutions without requiring parameter adjustments, providing a clearer, more direct comparison to our method.

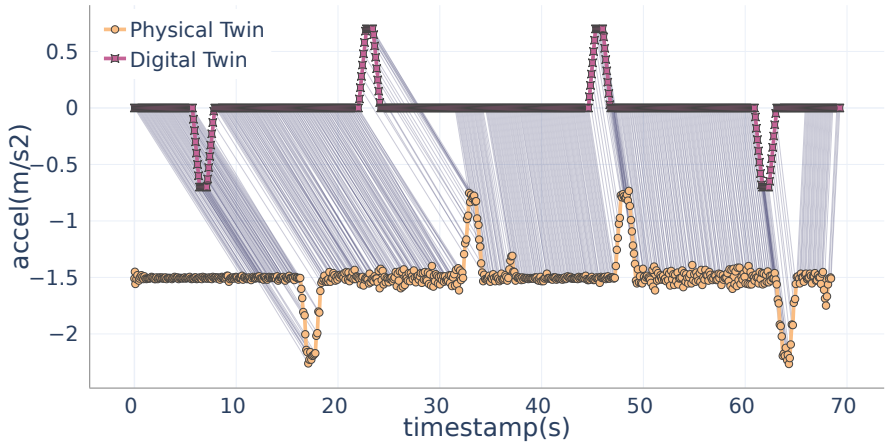
6.1.2 *Online Validation of DTs* by Lugaresi et al.

In 2023, Lugaresi et al. (Lugaresi et al. 2023) published a paper titled *Online Validation of Digital Twins for Manufacturing Systems*. The algorithm introduced in their work shares key principles with our proposal: both aim to compare the behavior of DTs by evaluating the sequence of discrete behavioral traces using dynamic programming techniques. Furthermore, both approaches define fidelity metrics to assess the accuracy of their results.

Despite these similarities, there are notable differences between the two solutions. First, while Lugaresi et al. (Lugaresi et al. 2023) focus on real-time, online trace comparison, our work operates offline. This distinction in timing is not critical for comparing the core behavior of the two algorithms, as both ultimately seek to assess fidelity between DTs.



(a) Dynamic Time Warping



(b) *SnapAligner* (LCAW, Affine Gap)

Figure 6.1: Alignments for Scenario (4-0-4). Execution 04.

A second significant difference lies in the type of data being analyzed. Our proposal is versatile, allowing for the comparison of any kind of trace, from raw sensor data to event-based analysis, provided a suitable distance measure (ranging between 0 and 1) is defined. This flexibility enables us to generate alignments between traces, facilitating the identification of gaps and mismatches. Our fidelity metrics, such as %MS, FD, and ED, offer further insights into the alignment quality.

In contrast, Lugaresi et al.'s work is more specialized, focusing on event-driven data and the analysis of Key Performance Indicators (KPIs). Their validation process occurs at two levels: event-level validation and KPI/performance-level validation. Event-level validation involves assessing the sequence of discrete events within the system, while the higher-level KPI validation analyzes performance indicators, which provide insight into the overall system efficiency. These levels are discussed below:

Event-level Validation. At the event-level validation stage, Lugaresi et al. employ the *Longest Common Subsequence (LCSS)* algorithm to determine whether the events generated by the twins are comparable. This dynamic programming algorithm identifies the longest sequence of matching events between two traces, maintaining the same order of appearance.

The Bellman equation for this algorithm is the following:

$$L(a_i, b_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ L(a_{i-1}, b_{j-1}) + 1 & \text{if } d(a_i, b_j) = 0 \wedge d(t_{a,[i]}, t_{b,[j]}) < \delta \\ \max\{L(a_{i-1}, b_j), L(a_i, b_{j-1})\} & \text{otherwise} \end{cases}$$

(Eq. 6.1.2.1)

Here, $L(a_i, b_j)$ is the accumulated score for the subsequence, representing how many matching events have been found up to the given point. If the events (represented by a_i, b_j) are equivalent, that is $d(a_i, b_j) = 0$, the score is incremented by 1. Furthermore, the authors added a time constraint, $d(t_{a,[i]}, t_{b,[j]}) < \delta$, where δ defines the maximum allowable time difference between two events for them to be considered equivalent. This time window helps ensure that events are aligned only if they occur within a similar timeframe, preventing the alignment of events that are too far apart temporally.

The algorithm's result is a numerical value representing the length of the longest common subsequence between the two traces. This value is then normalized to generate a fidelity metric ranging from 0 to 1, where 1 indicates that the sequence of events is perfectly faithful to the other trace. The fidelity metric is calculated by dividing the LCSS score by the length of the shorter trace.

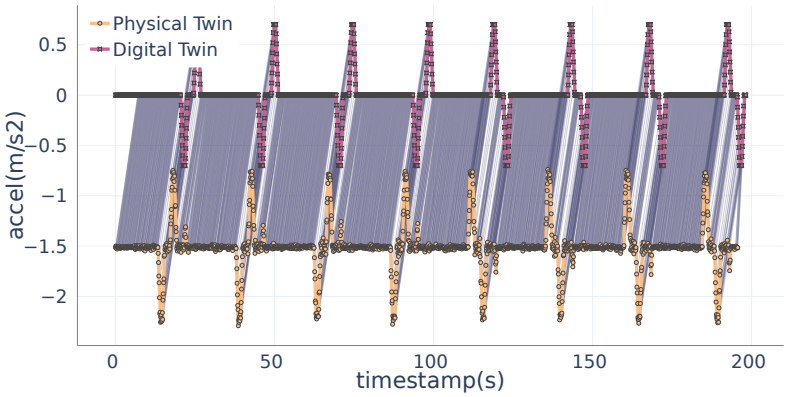


Figure 6.2: Affine Gap + LCAW for Scenario (4-3-2-1-0-1-2-3-4).

One limitation of Lugaresi et al.'s implementation is that they do not perform backtracking to reconstruct the actual optimal subsequence that the LCSS algorithm identifies. While they obtain the LCSS score, backtracking could provide valuable insights by revealing which specific events were excluded from the optimal subsequence, offering a more detailed understanding of where and why the traces diverge. By focusing solely on the score, they miss the opportunity to analyze misalignments more thoroughly.

To **empirically compare this approach**, which we call LCSS-Events, we added events to the raw trace from Scenario (4-3-2-1-0-1-2-3-4), as shown in Figure 6.2, by introducing events. In this scenario, the elevator moves down, stopping at every floor, and then ascends while doing the same. The four types of events added were *Down*, *Up*, *Brake*, and *Arrival*. These events were manually assigned to specific snapshots, corresponding to moments of acceleration change during the elevator's operation. Consequently, the trace included a total of 24 events.

One key issue when transitioning to an event-based analysis is the complexity involved in event identification. Identifying the precise events in real-world systems can be challenging and may require manual intervention or sophisticated algorithms to detect these events correctly. In contrast, our approach allows the alignment of snapshots at any level of abstraction, from raw sensor data to higher-level events. This flexibility offers a significant advantage by enabling detailed analysis across different levels, whether the focus is on low-level, granular data or more abstract event-based traces.

By gradually increasing the value of δ, we obtained the results presented in Table 6.2. Starting at 6.9 seconds, no subsequence was returned, and as δ, approached 7.8 seconds, the algorithm achieved the maximum score of 1. This means that while all events were included in the correct order, they were delayed by 7 to 7.8 seconds. The visualization

Table 6.2: Event-validation using LCSS statistics for scenario (4-3-2-1-0-1-2-3-4). Trace length of 24 events.

δ	normalized score	score / LCSS length
6.9	0.00	0
7.2	0.42	10
7.5	0.92	22
7.8	1.00	24

of the traces aligned by our algorithm (Figure 6.2) made it easier to interpret this result. However, the LCSS-Events method does not explain why the delay occurred—it merely shows that a particular similarity score was reached with a specific δ, value.

This limitation prevents more detailed reasoning when results are poor. The LCSS method only provides a single metric, and even if the subsequences were made available, there could still be issues. For instance, *stuttering* or alternating behavior in the trace could result in short subsequences. This phenomenon could occur even if the traces are highly similar due to alternating behavior.

KPIs / Performance-level validation. At this level, the system's performance is evaluated using KPIs to ensure that it matches the DT's results. These KPIs are sampled and concatenated as numerical sequences. Lugaresi et al. propose two algorithms for this comparison: **Modified Longest Common Subsequence (mLCSS)** and **Dynamic Time Warping (DTW)**.

The first algorithm, **Modified Longest Common Subsequence (mLCSS)**, follows a similar Bellman Equation as presented earlier but introduces a threshold value ϵ, which determines the maximum allowable distance between two KPI values for them to be considered equivalent:

$$L(a_i, b_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ L(a_{i-1}, b_{j-1}) + 1 & \text{if } d(a_i, b_j) < \epsilon \\ \max\{L(a_{i-1}, b_j), L(a_i, b_{j-1})\} & \text{otherwise} \end{cases} \quad \text{(Eq. 6.1.2.2)}$$

The algorithm then computes the length of the Longest Common Subsequence, normalized by dividing it by the length of the shortest sequence. The result is a fidelity metric ranging from 0 to 1, which reflects how well the sequences align. However, like the LCSS for event-level validation, mLCSS does not provide the aligned subsequences for further analysis, limiting the ability to examine the quality of the match or identify specific issues with the trace alignment.

Table 6.3: Performance-validation using LCSS statistics for scenario (4-3-2-1-0-1-2-3-4). Trace length of 1983 values of the average time between events.

ϵ	normalized score	score / LCSS length
0.1	0.099	197
0.2	0.099	197
0.3	0.224	444
0.4	0.722	1431
0.5	0.964	1911

To **empirically compare this approach**, which we call LCSS-KPIs, we introduced a specific KPI, *average time between floors*, which measures the time between the elevator's movement events ("Up," "Down") and the stopping event ("Arrival"). This was applied to the elevator Scenario (4-3-2-1-0-1-2-3-4) from Figure 6.2.

By gradually increasing ϵ, Table 6.3 results show that a 0.5-second difference is the largest discrepancy between any two average KPI values. However, there are several limitations with this approach:

- **Lack of subsequence output:** As with LCSS-Events, no resulting subsequences are provided, which prevents the identification and analysis of any discrepancies
- **Partial matches:** The presence of gaps between matching subsequences could lower the overall score, even when there is strong similarity between many segments.
- **Masked anomalies:** Since KPIs aggregate data, outliers or brief deviations may go unnoticed, which makes this method less effective for real-time anomaly detection.

The second algorithm used in the comparison is **Dynamic Time Warping (DTW)**, already described in previous sections. The authors adapt the DTW metric to produce a fidelity score between 0 and 1. The steps include:

1. **Normalization of the input sequences:** Each sequence is normalized by dividing the values by the maximum values of both sequences.
2. **DTW alignment:** The standard DTW algorithm is used to compute the alignment between the two normalized sequences.
3. **Metric calculation:** After obtaining the DTW distance $D_{[m,n]}(a,b)$, the final fidelity metric is calculated as:

$$\Phi_{DTW}(a,b) = 1 - \frac{D_{[m,n]}(a,b)}{max(m,n)} \qquad \text{(Eq. 6.1.2.3)}$$

This ensures that a perfect match between two identical sequences results in a fidelity score of 1, while two completely dissimilar sequences result in a score of 0.

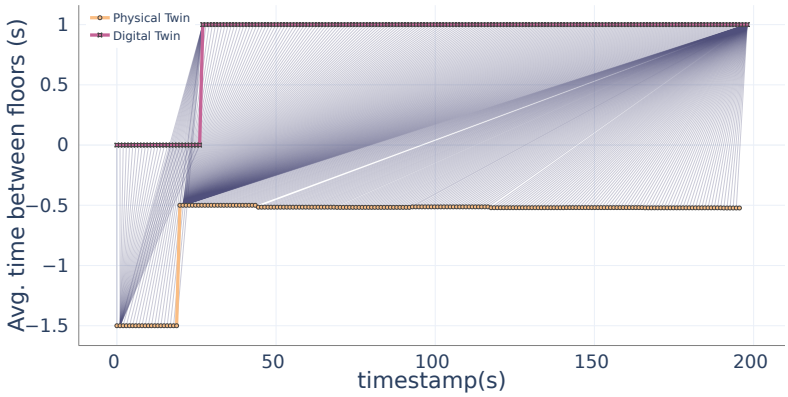


Figure 6.3: DTW alignment for Scenario (4-3-2-1-0-1-2-3-4) using KPI *avg. time between floors*. (*Note that it includes only 10% of the values for improved visualization*).

To **empirically compare this second approach**, we apply the same example. In Figure 6.3, the progression of KPI values is visualized as the scenario unfolds. However, this method suffers from the same issues discussed in the previous sections (c.f. Section 6.1.1). For instance, over-stretching and over-condensing distort the alignment. In the specific case of this alignment, one KPI value from the PT was aligned more than 1700 times, significantly skewing the results. In addition, it only depends on the output score without analyzing the output alignment. As a result, this method might overlook results influenced by over-stretching or over-compression.

An additional limitation may arise during the normalization process. Consider two sets of five elements each for example:

$$A = \{0.05, 0.05, 0.05, 0.05, 0.05\} \qquad B_1 = \{0.01, 0.02, 0.03, 0.04, 0.05\}$$

We normalized the traces by dividing both of them by their maximum value (0.05).

$$\overline{A} = \{1, 1, 1, 1, 1\} \qquad \overline{B_1} = \{0.2, 0.4, 0.6, 0.8, 1\}$$

If we assume that the returned alignment consists in aligning the elements at the same position, we obtain the following set of distances:

$$D(A, B_1) = D(\{0.8, 0.6, 0.4, 0.2, 0\}) = 2$$

Applying the metric to this output distance, we obtain:

$$\Phi_{DTW}(A, B_1) = 1 - \frac{2}{5} = 0.6$$

Now, let us compare this situation with a similar alignment, including a trace B_2 that has an outlier. The normalized traces would be the same since the maximum value is 1.

$$A = \overline{A} = \{0.05, 0.05, 0.05, 0.05, 0.05\} \qquad B_2 = \overline{B_2} = \{0.01, 0.02, 0.03, 0.04, \mathbf{1}\}$$

The alignment was expected to perform worse due to the presence of an outlier rather than a perfect match. However, the actual results show a better alignment, primarily due to the impact of the normalization process in the DTW algorithm. Instead of applying a normalization factor below 1, the factor remained at 1, resulting in smaller distance values being assigned less weight. This ultimately masked the effect of the outlier in the sequence.

$$D(A, B_2) = D(\{0.04, 0.03, 0.02, 0.01, 0.95\}) = 1.05$$

When calculating the distance between the sequences, the algorithm returned a distance of 1.05 instead of the anticipated 2. This discrepancy implies that the DTW algorithm misjudges the similarity between the traces, interpreting them as more closely aligned than they are in reality. Consequently, the fidelity metric scored 0.79, which incorrectly suggests that the traces are more similar than a previous comparison, which had a score of 0.6.

$$\Phi_{DTW}(A, B_2) = 1 - \frac{1.05}{5} = 0.79$$

However, a closer look reveals this assessment to be inaccurate. The two traces are less similar, as the final value of the second trace is significantly more distant than the corresponding value in the first trace, B_1. Due to normalization, the other distance values are artificially reduced, and the overall distance score (1.05) becomes lower than expected. This results in the algorithm falsely concluding that the trace B_2 is more similar to A than B_1, which is not the case. The outlier is effectively hidden by the normalization, which prevents the algorithm from accurately assessing the degree of mismatch.

This highlights a significant limitation of DTW-Lugaresi when applied to normalized sequences: It can produce misleading similarity metrics when outliers are present and fail to provide an accurate representation of alignment quality.

Answer to RQ.1: Our approach proves to be more effective than the existing methods discussed for detecting and diagnosing delays and anomalies, and measuring the behavior similarity between traces. Unlike DTW, our method is robust against overstretching or overcondensing, allowing it to effectively highlight discrepancies between traces. Additionally, our proposal offers greater versatility compared to the work of Lugaresi et al. (2022), as it can align both sequences of events and raw traces. Moreover, our approach provides a detailed alignment of the trace, facilitating the detection of inconsistencies and offering valuable support in their diagnosis.

6.2 Time and Space Complexity. Scalability Analysis (RQ.2-3)

In this section, we evaluate the performance of the NDW variants by comparing them with the state-of-the-art algorithms introduced earlier. To carry out this comparison, we aligned traces from the (4-3-2-1-0-1-2-3-4) scenario, incrementally increasing the number of snapshots in the dataset. If the scenario did not contain enough snapshots for meaningful comparison, we simulated a system restart, looping through the scenario multiple times.

For each alignment, we recorded the execution time and memory usage of the process and estimated the algorithm's time and space complexities using regression analysis. The goal was to assess whether the NDW variants exhibit more efficient scaling as the number of snapshots increases relative to the baseline algorithms. Each alignment with a particular trace length was repeated five times to ensure accuracy and account for variability. We then computed the average execution time and the standard deviation for each alignment.

Evaluation Settings. The performance analysis of the algorithms should remain independent of the system configuration used for execution. However, to facilitate the reproducibility of the results, we have documented the configurations for each algorithm that requires specific parameters to generate results. The table below outlines the key configuration settings applied during the evaluation of these algorithms. If an algorithm is not listed in the table, it indicates that it does not necessitate any configuration parameters for its execution.

Parameter	Value
Needleman-Wunsch Variants	
Maximum Acceptable Distance (MAD)	0.15
Penalty opening a gap (P_{op})	-1
Penalty extending a gap (P_{ex})	-0.1
Low Complexity Areas Weight (LCAW)	0.005
LCSS Events	
Max time between matched snaps (δ)	6.0
LCSS KPIs	
Max distance between matched snaps (ϵ)	0.5

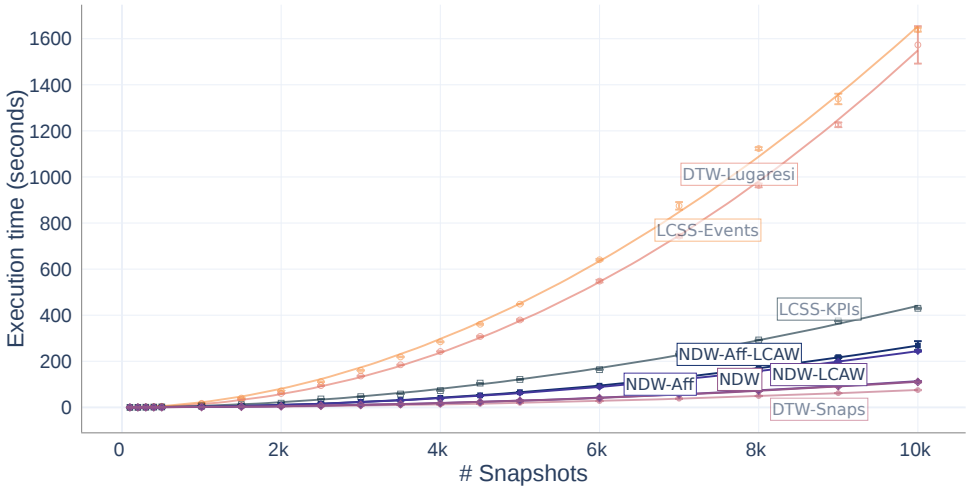
The algorithms were executed in the following computational environment:

CPU	Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
RAM	64 GB
OS	Ubuntu (5.15.0-78-generic #85-Ubuntu SMP x86_64 GNU/Linux)

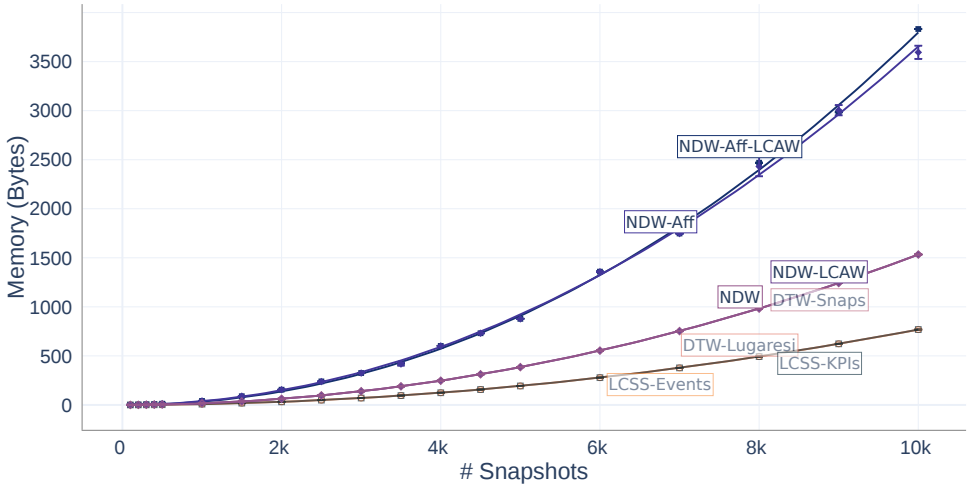
Results. The performance results for each algorithm with varying sequence lengths are presented in Figure 6.4 and Table 6.5a for execution time, and in Table 6.5b for memory usage. To assess the time complexity of these algorithms, we used polynomial regression analysis, which provided insights into their computational behavior. The results of this analysis are summarized in Table 6.5.

As indicated in Table 6.5, all the algorithms exhibit approximately quadratic complexity, aligning with the expected theoretical complexity of dynamic programming algorithms in time and space. These algorithms typically fill a dynamic programming matrix, where each cell represents a computational operation. Although the dimensions of this matrix can vary, for simplicity, we modeled them as square matrices. The polynomial regression model we used is $y = a \cdot x^b$, where x is the sequence length, and y is the execution time in seconds or the memory consumption in bytes.

The coefficient 'a' represents the base time to process each snapshot, influenced by the specific processor used. Although 'a' may vary across different hardware, snapshot processing on our processor took approximately 100 nanoseconds. Comparing the algorithms based on this coefficient, a higher 'a' value corresponds to a steeper slope, indicating poorer scalability. Similarly, in the memory consumption analysis, 'a' is the base memory needed to store and process a trace of one snapshot.



(a) Execution time comparison.



(b) Memory consumption comparison.

Figure 6.4: Execution time and memory consuption by the number of snapshots in Scenario (4-3-2-1-0-1-2-3-4).

The algorithms, ranked from the most computationally expensive to least expensive with respect to time consumption, as shown in Figure 6.4a, are the following:

- **DTW-Lugaresi.** The most computationally expensive algorithm, as reflected by the highest 'a' value. This is primarily due to the use of the np.min function from numpy in each cell calculation, which incurs additional overhead compared to the standard min function in Python. By switching to the latter, this algorithm could be optimized to be closer in execution time to the standard DTW implementation.
- **LCSS-Events.** This algorithm also demonstrates relatively high execution time, second only to DTW-Lugaresi. The main reason for this is the string comparison overhead, which is more costly than the number comparison used in LCSS-KPIs. A potential optimization would be to replace string identifiers with unique numeric values to improve efficiency.
- **LCSS-KPIs.** Although this version also employs the LCSS algorithm, it is more efficient than LCSS-Events, owing to the simpler numeric comparisons rather than string comparisons.
- **SnapAligner Variants.** Among the NDW variants, those that do not incorporate affine gap achieve the best performance. This is because their decision-making process involves only a single matrix. The inclusion of the affine gap variant increases computational complexity, leading to a performance reduction of 30.7%. Similarly, the addition of the LCAW mechanism increases execution time but only marginally—by less than 1% in the non-affine gap case and by 6% when affine gap is included.
- **Original DTW.** Despite being the most efficient algorithm in terms of execution time, outperforming even the base NDW variant by 30%, DTW produces lower-quality alignments compared to SnapAligner, as explained in the preceding section.

With respect to memory consumption, we categorize the algorithms into three groups:

- **Three-matrix calculation**. This includes variants of our algorithm that use Affine Gap. As explained in the previous section, deciding whether to include a gap in either of the traces requires two additional matrices to record such decisions, effectively doubling the memory cost with respect to the next group.
- **One-matrix calculation**. This group includes NDW, NDW-LCAW, and DTW-Snaps, which perform the calculations using only one matrix that records the alignment rewards between snapshots.
- **No backtracking**. This group includes all proposals by Lugaresi et al. These algorithms also use one matrix for calculations but do not perform backtracking to obtain the optimal alignment. The backtracking process requires additional memory to store the results but provides valuable insights for identifying discrepancies.

While DTW algorithms, particularly the optimized versions, outperform other methods in terms of raw execution time, their alignment quality often falls short. In contrast, NDW variants offer a more balanced approach. If smaller memory consumption and faster response times are required, sacrificing the extra precision provided by the affine gap may be acceptable. In that case, the base NDW algorithm delivers reliable insights with the same memory consumption and similar time performance as DTW.

Answer to RQ.2: The time and space complexity of our algorithm, along with all its configurations, is quadratic with respect to the number of snapshots. The affine gap optimization introduces significant overhead, as it requires the algorithm to consult and modify three matrices in each step. This results in a 30% performance decrease and doubles memory consumption. On the other hand, the LCAW optimization has a minimal impact, reducing performance by less than 1% when the affine gap is not used. However, when combined with the affine gap, the performance impact increases to 6%.

Table 6.4: Regression analysis results for the algorithms

(a) Execution Time

Algorithm name	Regression result in 10^{-6}	Time Complexity
NDW-Aff-LCAW	$y = 1.83 \cdot x^{2.04}$	$O(n^{2.04}) \approx O(n^2)$
NDW-Aff	$y = 2.82 \cdot x^{1.98}$	$O(n^{1.98}) \approx O(n^2)$
NDW-LCAW	$y = 1.40 \cdot x^{1.98}$	$O(n^{1.98}) \approx O(n^2)$
NDW	$y = 1.40 \cdot x^{1.97}$	$O(n^{1.97}) \approx O(n^2)$
DTW-Snaps	$y = 1.07 \cdot x^{1.96}$	$O(n^{1.96}) \approx O(n^2)$
DTW-Lugaresi	$y = 9.92 \cdot x^{2.05}$	$O(n^{2.05}) \approx O(n^2)$
LCSS-Events	$y = 51.9 \cdot x^{1.88}$	$O(n^{1.88}) \approx O(n^2)$
LCSS-KPIs	$y = 15.7 \cdot x^{1.86}$	$O(n^{1.86}) \approx O(n^2)$

(b) Memory Usage

Algorithm name	Regression result in 10^{-6}	Space Complexity
NDW-Aff-LCAW	$y = 21.2 \cdot x^{2.06}$	$O(n^{2.06}) \approx O(n^2)$
NDW-Aff	$y = 40.9 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$
NDW-LCAW	$y = 16.5 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$
NDW	$y = 16.4 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$
DTW-Snaps	$y = 16.5 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$
DTW-Lugaresi	$y = 8.67 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$
LCSS-Events	$y = 8.49 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$
LCSS-KPIs	$y = 8.53 \cdot x^{1.99}$	$O(n^{1.99}) \approx O(n^2)$

Answer to RQ.3: The time and space complexity of all dynamic programming algorithms considered here is quadratic, as expected. Although the implementation by Lugaresi et al. has minimal memory consumption, its average computational time is higher than that of any configuration of our proposed algorithm. In contrast, the original DTW (Sakoe and Chiba 1971) outperforms our algorithm in terms of computation time and consumes the same amount of memory as our algorithm without the affine gap, as it imposes fewer alignment restrictions. However, as demonstrated earlier, this lack of restrictions leads to imprecise measurements, especially when handling over-stretching or over-condensing, which can significantly impact result accuracy.

Table 6.5: Raw results for the algorithms

(a) Execution time in seconds.

# snaps	NDW	NDW-LCAW	NDW-Aff	NDW-Aff-LCAW	DTW-Snaps	DTW-Lugaresi	LCSS-Events	LCSS-KPIs
100	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.15 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
200	0.04 ± 0.00	0.04 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.03 ± 0.00	0.60 ± 0.01	0.70 ± 0.00	0.06 ± 0.00
300	0.10 ± 0.00	0.10 ± 0.00	0.20 ± 0.00	0.21 ± 0.00	0.06 ± 0.00	1.34 ± 0.01	1.58 ± 0.02	0.62 ± 0.01
400	0.16 ± 0.00	0.16 ± 0.00	0.33 ± 0.00	0.35 ± 0.00	0.11 ± 0.00	2.40 ± 0.02	2.84 ± 0.02	1.37 ± 0.01
500	0.28 ± 0.00	0.28 ± 0.00	0.60 ± 0.01	0.62 ± 0.00	0.19 ± 0.00	3.81 ± 0.07	4.45 ± 0.06	2.17 ± 0.01
1000	1.19 ± 0.01	1.21 ± 0.01	2.60 ± 0.01	2.72 ± 0.02	0.81 ± 0.01	15.09 ± 0.12	17.59 ± 0.12	6.54 ± 0.04
1500	2.59 ± 0.02	2.62 ± 0.02	5.64 ± 0.03	5.92 ± 0.02	1.77 ± 0.02	33.95 ± 0.27	40.43 ± 0.44	11.61 ± 0.07
2000	4.43 ± 0.04	4.51 ± 0.03	9.68 ± 0.03	10.24 ± 0.06	3.06 ± 0.03	60.32 ± 0.43	71.72 ± 0.40	18.10 ± 0.16
2500	6.98 ± 0.05	7.04 ± 0.05	15.27 ± 0.13	15.94 ± 0.07	4.84 ± 0.04	94.49 ± 0.77	111.42 ± 0.93	36.40 ± 0.26
3000	10.25 ± 0.09	10.53 ± 0.08	22.39 ± 0.06	23.57 ± 0.08	7.13 ± 0.07	136.13 ± 1.62	160.39 ± 1.37	46.63 ± 0.26
3500	13.78 ± 0.03	14.19 ± 0.13	30.27 ± 0.13	31.84 ± 0.09	9.62 ± 0.08	184.77 ± 1.30	218.98 ± 1.46	57.96 ± 0.33
4000	17.82 ± 0.07	18.16 ± 0.19	38.89 ± 0.30	41.24 ± 0.18	12.35 ± 0.04	242.43 ± 0.49	284.50 ± 1.86	72.84 ± 0.36
4500	22.79 ± 0.21	23.29 ± 0.21	49.40 ± 0.23	52.29 ± 0.35	15.57 ± 0.12	307.70 ± 2.79	360.56 ± 2.17	105.27 ± 0.62
5000	28.45 ± 0.23	28.88 ± 0.12	62.25 ± 0.31	65.98 ± 0.11	19.53 ± 0.12	379.34 ± 3.11	448.00 ± 0.94	121.08 ± 0.36
6000	40.23 ± 0.22	40.86 ± 0.23	88.32 ± 0.48	93.98 ± 0.55	27.31 ± 0.27	547.94 ± 5.47	640.56 ± 3.63	163.72 ± 0.72
7000	55.48 ± 0.27	57.30 ± 0.30	121.48 ± 0.33	129.60 ± 0.66	37.80 ± 0.41	743.34 ± 6.89	874.74 ± 16.51	231.62 ± 0.98
8000	71.38 ± 0.15	73.12 ± 0.31	156.51 ± 1.22	167.91 ± 0.40	49.24 ± 0.72	963.42 ± 8.18	1123.23 ± 6.44	292.53 ± 1.44
9000	91.56 ± 0.23	93.91 ± 0.31	199.60 ± 1.12	218.06 ± 3.92	61.71 ± 0.60	1226.47 ± 10.15	1338.50 ± 22.83	375.12 ± 1.46
10000	110.16 ± 2.12	112.75 ± 1.00	243.10 ± 1.86	268.03 ± 19.51	75.11 ± 0.83	1573.30 ± 81.39	1638.76 ± 8.28	429.95 ± 2.65

(b) Memory consumption in bytes

# snaps	NDW	NDW-LCAW	NDW-Aff	NDW-Aff-LCAW	DTW-Snaps	DTW-Lugaresi	LCSS-Events	LCSS-KPIs
100	0.28 ± 0.14	0.31 ± 0.21	0.90 ± 0.09	1.18 ± 0.57	0.27 ± 0.19	0.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.00
200	0.97 ± 0.12	0.96 ± 0.06	3.39 ± 0.14	3.33 ± 0.06	0.91 ± 0.07	0.32 ± 0.07	0.26 ± 0.10	0.43 ± 0.11
300	1.93 ± 0.16	1.98 ± 0.12	5.91 ± 0.20	6.13 ± 0.38	1.83 ± 0.21	0.97 ± 0.10	1.23 ± 0.08	1.04 ± 0.22
400	3.20 ± 0.44	2.79 ± 0.39	9.06 ± 0.53	8.25 ± 0.08	2.95 ± 0.34	1.57 ± 0.32	2.27 ± 0.32	1.59 ± 0.34
500	4.30 ± 0.34	4.24 ± 0.33	13.15 ± 1.01	12.63 ± 0.01	3.87 ± 0.00	2.24 ± 0.35	3.12 ± 0.41	2.45 ± 0.41
1000	15.98 ± 0.00	15.98 ± 0.00	40.83 ± 0.41	41.09 ± 0.68	16.02 ± 0.09	8.25 ± 0.00	8.23 ± 0.62	8.26 ± 0.01
1500	35.54 ± 0.10	35.58 ± 0.00	89.32 ± 0.03	88.79 ± 0.73	35.58 ± 0.00	18.05 ± 0.00	18.18 ± 0.01	18.15 ± 0.01
2000	62.65 ± 0.00	62.65 ± 0.00	155.39 ± 0.29	155.17 ± 0.28	62.87 ± 0.09	31.71 ± 0.00	31.62 ± 0.00	31.70 ± 0.00
2500	97.20 ± 0.00	97.20 ± 0.00	237.96 ± 2.00	239.83 ± 3.71	97.45 ± 0.00	49.17 ± 0.13	49.09 ± 0.14	49.06 ± 0.00
3000	139.47 ± 0.00	139.51 ± 0.09	324.67 ± 0.73	325.22 ± 0.72	139.73 ± 0.00	70.38 ± 0.00	70.25 ± 0.09	70.30 ± 0.00
3500	189.49 ± 0.00	189.49 ± 0.00	416.94 ± 15.56	423.13 ± 0.72	189.62 ± 0.14	95.39 ± 0.00	95.38 ± 0.03	95.45 ± 0.10
4000	247.02 ± 0.10	247.09 ± 0.14	601.29 ± 0.69	601.45 ± 0.07	247.24 ± 0.00	124.26 ± 0.00	124.08 ± 0.02	124.28 ± 0.10
4500	312.21 ± 0.00	312.21 ± 0.00	730.84 ± 1.04	730.88 ± 0.75	312.66 ± 0.12	157.00 ± 0.00	$156.74 \pm nan$	157.00 ± 0.00
5000	385.17 ± 0.00	385.17 ± 0.00	877.47 ± 1.09	877.73 ± 0.11	385.43 ± 0.00	193.61 ± 0.00	193.44 ± 0.04	193.45 ± 0.00
6000	553.78 ± 0.00	553.78 ± 0.00	1357.70 ± 0.98	1358.75 ± 4.61	554.04 ± 0.00	278.18 ± 0.00	277.64 ± 0.11	278.15 ± 0.00
7000	752.81 ± 0.00	752.81 ± 0.00	1746.63 ± 1.39	1747.89 ± 2.82	753.07 ± 0.00	377.95 ± 0.00	377.43 ± 0.65	377.78 ± 0.12
8000	982.42 ± 0.40	982.48 ± 0.10	2430.57 ± 99.30	2469.05 ± 54.89	982.81 ± 0.09	492.93 ± 0.00	492.31 ± 0.71	492.90 ± 0.00
9000	1242.65 ± 0.00	1242.65 ± 0.00	3005.07 ± 52.28	2983.77 ± 84.25	1243.17 ± 0.00	623.23 ± 0.42	622.77 ± 0.12	623.07 ± 0.00
10000	1533.32 ± 0.14	1533.24 ± 0.10	3594.38 ± 67.32	3832.29 ± 58.45	1533.98 ± 0.00	767.90 ± 0.39	768.13 ± 0.25	768.75 ± 0.00

6.3 Limitations and Threats To validity

The proposal is subject to several limitations and threats to its validity.

First, we have validated the proposed approach with four DTSs, chosen from different application domains and with diverse characteristics. Our approach has performed consistently across these cases. However, additional validation is needed with other types of systems to assess the **generalizability** of our results. Specifically, the threshold and parameter values used by our algorithm and fidelity indicators must be further confirmed and validated in more extensive industrial case studies.

Second, we assume that both digital and physical twins can be represented as discrete sequences of states (snapshots) at a suitable level of abstraction and resolution. While this

is a common assumption for many systems, we must further investigate its **applicability to highly dynamic systems**, such as those involving fluids, complex interactions between parts with different natures, or systems involving human participation. In such cases, this representation may not always be feasible or sufficiently faithful.

Third, our proposal relies on several parameters that need to be customized for each digital twin system's fidelity measurement: *MAD*, affine gap values (P_{op}, P_{ex}), *LCAW*, and the fidelity indicator thresholds. Although we have provided default values, these parameters are system-dependent and may require **fine-tuning** by end users to ensure optimal results.

Additionally, while our fidelity indicators have shown robustness in handling the inherent stochastic nature and **random uncertainties of cyber-physical systems**, as well as the variability across different runs of the same system, more extensive testing and evidence from diverse systems are needed to confirm this resilience.

One limitation of our current approach is its reliance on comparing complete traces to determine the systems' fidelity. This represents an initial step toward a more dynamic method that can assess the fidelity of both twins in **real-time**. Developing a more dynamic, runtime-based solution is part of our future research. Similarly, our current proposal does not yet address real-time synchronization between the two twins, an aspect we plan to explore further in upcoming work.

7

TOOL SUPPORT

The proposal presented in this thesis for assessing the fidelity of DTs has been implemented as a software tool. The implementation and its evaluation with the different demonstration cases is openly available (Muñoz et al. 2023a).

The implementation, developed in Python 3.10, accepts a ".yaml" file as input, which contains the alignment configuration hyperparameters (e.g., MAD, Affine Gap, LCAW, etc.) along with the paths to the ".csv" files containing the traces. The algorithm's output includes an image depicting the alignment and two ".csv" files: one detailing the alignment and the other containing the corresponding fidelity metrics values.

This section outlines how to use the tool for performing alignments. We begin by specifying the required execution environment and instructions for installing the necessary packages. Next, we provide usage examples to illustrate how to perform one of the alignments discussed in the thesis. Finally, we detail all the parameters of the configuration file, explaining how users can customize it for their specific needs.

7.1 Execution Environment

The algorithm was developed using **Python 3.10**, so we recommend installing this version of Python for executing the alignments and analysis. We strongly recommend using **Linux or the Linux Subsystem for Windows** for the alignments, as the required library for generating the graphics (kaleido) may encounter some issues when running on Windows.

7.2 Required Packages

To use the tool, clone the repository using:

```
git clone -b sliding-window https://github.com/atenearesearchgroup/
    ↪ fidelity-measure-for-dts.git
```

To install all the required packages:

1. Open the command line in the *blast* folder.
2. Install the required packages with pip:

```
python -m pip install -r requirements.txt
```

7.3 Usage

1. Open a *Linux/Windows Subsystem Linux* command line
2. Navigate to the /src directory
3. Using Python 3.10, you can execute the following command to display the help information of our alignment script.

```
python align_traces.py -h
```

```
options:
-h, --help      show this help message and exit
--figures       It processes the alignment and generates figures
    ↪ as image files
--engine ENGINE Engine to process output pdf figures  (orca or
    ↪ kaleido). By default, kaleido.
--config CONFIG Config file name stored in the /src/config folder
```

4. To execute alignments for the Incubator case study using one of the predefined YAML configuration files:

```
python align_traces.py --figures --engine kaleido --config
    ↪ incubator_comparison.yaml
```

7.4 Configuration

This is the configuration file for the variability study of the elevator, evaluating all possible scenarios. The comments describe the content of the different attributes.

```yaml
# Specify the system type (e.g., RoboticArm, Lift, Incubator).
system: "Lift"

# Define the directory paths for input and output files.
paths:
  input:
    # Parent directory under /src
    main: "resources/input/lift/"
    # Subdirectory for the DT files.
    dt: "digital_twin_high_fid_elevate/"
    # List of the DT traces files
    dt_files:
      - "Bajada_4_0_4.csv"
      - "Bajada_4_2_0_2_4.csv"
      - "Bajada_4_3_2_1_0_1_2_3_4.csv"
    # Subdirectory for PT files.
    pt: "physical_twin"
    # List of the PT traces that correspond to the DT's
    pt_files:
      - "Bajada_4_0_4"
      - "Bajada_4_2_0_2_4"
      - "Bajada_4_3_2_1_0_1_2_3_4"
    # Output directory to store alignments and metrics
    output: "resources/output/lift/variability/"

# Specify the labels of the properties of interest
labels:
  timestamp_label: "timestamp(s)"
  params:
    - "accel(m/s2)"
  # Set the main parameter to plot the alignments
  param_interest: "accel(m/s2)"


# Define the ranges for the input hyperparameters
ranges:
  # MAD - Maximum Acceptable Distance
  mad:
    start: 0.02
    step: 0.02
    end: 0.30
  # LCAW - Low Complexity Area Weight
```


```
  low:
    start: 200
    step: 2
    end: 201
  # p_op - Penalty for opening a Gap
  init_gap:
    start: -1.0
    step: 0.5
    end: -0.6
  # p_ex - Penalty for extending a Gap
  cont_gap:
    start: -0.1
    step: 0.1
    end: -0.09
```

8

RELATED WORK

In this chapter, we introduce some of the main state-of-the-art proposals related to our approach for validating DTs. We first discuss other proposals for validating DTs in general, followed by a review of similarity measurements for models, trace analysis algorithms, and finally, online validation techniques that are closely related to our proposal.

8.1 Validation of DTs

Traditional validation techniques often rely on statistical methods, requiring large data sets and multiple independent replications due to the stochastic nature of physical systems (Balci 1994). However, the effectiveness of these techniques often depends on the availability and quality of the data used for validation.

Typically, data for validation purposes is gathered using two main types of simulations (Banks 1998). The first is *self-driven simulations*, where input data is generated by sampling from probabilistic models. These models are usually derived by fitting probability distributions to observed data, such as real-world measurements or historical records. Self-driven simulations are particularly useful in scenarios where it is difficult to obtain real-time or comprehensive datasets, allowing the model to explore a wide range of possible system behaviors under different conditions. However, the accuracy of the outcomes is directly dependent on the quality of the probabilistic models and the assumptions made when fitting the data, which can introduce biases or fail to capture rare events effectively.

The second approach is Trace-Driven Simulations (TDS) (Marquardt, Cleophas, and Morgan 2021), where the model is executed using real trace data collected from the system.

TDS has the advantage of grounding the simulation in actual observed behaviors, providing a more accurate representation of the system under study. This method is particularly valuable in cases where the system operates in environments with high variability or unpredictability, as it allows the model to directly reflect real-world inputs. However, TDS can be limited by the quality, completeness, and representativeness of the trace data. If the traces do not cover all relevant operational scenarios or fail to capture the full range of system behaviors, the validation might be incomplete or skewed towards specific conditions.

Our trace alignment algorithm allows for a more precise comparison of traces expected to represent analogous behaviors. This enhances the accuracy of TDS and improves the analysis of any two traces that describe synchronized behaviors within our proposal.

8.2 Similarity Measurements

Measuring similarity in a DT system is crucial for defining the DT at the minimum required level of fidelity to optimize computational costs (Ahlgren et al. 2021). Ensuring a balance between accuracy and efficiency is crucial, especially in real-time applications. By measuring fidelity, we can guarantee that the DT closely represents the physical system without unnecessary computational overhead.

Some proposals, such as (Acker et al. 2019), introduce a *validity frame*, which provides a semi-automated methodology to determine the suitability of a given simulation. This *validity frame* enables the reuse of models with different levels of accuracy, also considering requirements such as performance or computational constraints. It defines an iterative process in which the suitability of the representation is checked, and parameters are adjusted if it does not meet the requirements, continuing this loop until it does.

However, most existing methods for assessing similarity between simulations use specific measures. For instance, Kullback-Liebler Divergence (KLD) is used by Worden et al. (2020) to assess the difference between probability distributions and thus evaluate the variability between runs. The Jensen-Shannon Distance, a normalized, symmetrical version of KLD, is applied by Bojarczuk et al. (2021) to account for the uncertainty in non-deterministic executions.

Another widely used method for assessing similarity in simulations is Hotelling's T^2 (Hotelling 1931), particularly valuable for comparing multivariate data. It evaluates the difference between mean vectors of datasets while accounting for the covariance between variables, making it suitable for complex systems with interdependent parameters

These approaches share similarities with ours but primarily focus on comparing probability distributions, without accounting for alignment gaps or mismatches, which are key to our method.

8.3 Trace Analysis

Recent work focuses on comparing data traces (or sequences) for real-time validation of digital models (Lugaresi et al. 2023). For example, semantic matching rules are employed by Langer, Mayerhofer, and Kappel (2014) to identify differences between traces. Another study by Leroy et al. (2018) uses operators to compare execution traces of state machine models, measuring their similarity with the Levenshtein distance (Levenshtein 1966), which quantifies the minimum number of edits needed to transform one sequence into another. This method is particularly effective in scenarios where execution order matters.

In these approaches, system traces are often synthesized from executions (Alimadadi, Mesbah, and Pattabiraman 2018; Bose and Aalst 2012) or inferred from system specifications (e.g., from state machines (Wolny et al. 2019)). Process mining techniques (Aalst 2016) can also infer trace models from event logs, using similarity measures to assess conformance with discovered models. For example, a variant of the Damerau-Levenshtein distance (Bard 2007) is used by Camargo, Dumas, and González (2020) to compare event log traces, with the Hungarian algorithm (Kuhn 2010) pairing traces to minimize their distance.

These algorithms typically aim for global trace alignment, which aligns complete sequences. Local alignments, like those used in BLAST, focus on specific trace patterns (e.g., identifying a floor change pattern in one of the elevator traces). In contrast, we aim to compare full traces, making global alignment algorithms like NDW more suitable. Nonetheless, we incorporate key optimizations from BLAST to enhance our approach.

Unlike these methods, which focus on deterministic systems, our approach is tailored to cyber-physical systems with inherent uncertainties. We introduce the *MAD* parameter to account for a certain tolerance when aligning snapshots and allow for gaps, enabling the detection of missing behaviors and deviations in trace comparisons.

8.4 Online Validation Techniques

Online validation techniques are widely used in Digital Twin engineering (Lugaresi et al. 2022). Unlike traditional offline methods, which typically rely on post-simulation analysis, these online techniques validate the digital twin in real-time by comparing the real system's traces against those generated by the DT through TDS (Marquardt, Cleophas, and Morgan 2021). Rather than returning binary validation outcomes, these approaches often express the validity of the model as a percentage of credibility.

Initial works in this area focus on continuous traces, using techniques such as harmonic analysis (Lugaresi et al. 2019), which, compared to statistical techniques, can deliver

reliable results with smaller data sets. DTW is employed by Gong et al. (2022) to measure the similarity between human and robot arm movements, comparing trace sequences in real-time. DTW is valuable because it aligns sequences of varying lengths or different timings. Other works (Alimadadi, Mesbah, and Pattabiraman 2018; Leroy et al. 2018) apply sequence analysis algorithms from text processing to trace analysis, where each measurement is treated as a character. A similar method is proposed by Lugaresi et al. (2022) and applied to a simple single-server system. Furthermore, our earlier work (Muñoz et al. 2022) introduced a version of the Needleman-Wunsch algorithm for trace alignment and distance calculation.

A recent work (Lugaresi et al. 2023), described in Section 6.1, is closely related to our approach. It uses a DTW variant for aligning traces, enhanced by a comparison function to identify similar events. While both methods align traces before defining distance measures, they do not utilize key BLAST optimizations, such as handling sequences of gaps or masking low-complexity regions. Moreover, they lack the *MAD* threshold and our fidelity indicators, which have proven critical in assessing digital-physical twin fidelity effectively. Our experiments highlight the importance of these optimizations to avoid alignment errors and inaccuracies in measuring trace distances.

9

CONCLUSIONS AND FUTURE WORK

9.1 Summary and Contributions

This thesis emerged from the recent revival of the Digital Twin concept within the Industry 4.0. At the start of this thesis, the topic was still in its early stages within the research community, and discussions primarily focused on defining its main concepts and exploring their interrelations. Engaging in these discussions allowed us to gain a clearer understanding of these concepts, leading us to realize that the key to effective system twinning lies in the fidelity between the elements being twinned.

In this context, the contributions of this thesis are twofold. First, by engaging in discussions within the community, we present our vision of this paradigm and propose a conceptual architecture, a framework, and a set of definitions that contextualize all their elements. Our proposals stand out from much of the existing literature as we introduce the idea of a symmetric architecture consisting of two components—the digital and the physical—which are embedded in a system we refer to as the Digital Twin System.

As we analyze this symmetric architecture with two synchronized twinned systems, we recognize the crucial importance of fidelity in the validation. Since it is impossible to create truly identical twins—because we cannot replicate the exact physical characteristics of a real twin—our goal is to develop faithful twins that accurately reflect the behavior of the original system with an adequate level of fidelity. The main challenge is measuring this level of fidelity between the two twins to ensure it meets the requirements for their intended purpose.

We propose a method and a set of metrics to assess the fidelity of the two twins' behavior based on comparing the sequences of states they reach during execution. To achieve this, we define a discrete representation of the twins' behavior as a sequence of snapshots. We then adapt a character alignment algorithm from bioinformatics and introduce a comparison function to evaluate the similarity between pairs of snapshots. After aligning the two traces, our method computes a set of metrics (%MS, FD, and ED) that evaluate the fidelity between them. Additionally, we offer a guide to interpret these metrics and determine the degree of fidelity.

This approach is broadly applicable for comparing the behavior of two systems, even beyond the context of DTSs, and assessing behavioral similarity in various scenarios. It can be used to compare physical systems (e.g., identifying deviations between two instances of the same machine), digital systems (e.g., comparing two simulations with different levels of detail), or between a physical and digital system, as in the case of DTSs. The algorithm is suitable in any context where a twinned behavior is expected.

9.2 Future Work

This thesis contributes to the field of DTs, which faces emerging challenges as its industrial applications spread. Our method can be further improved and extended to address additional challenges and overcome some limitations.

First, our algorithm can be used to diagnose sequences of misaligned traces automatically. While the current method identifies deviations, we aim to offer a more comprehensive diagnosis. Our preliminary work classified these deviations into anomalies and delays (Muñoz, Troya, and Vallecillo 2024). Our early experiments suggest that a series of gaps in one of the traces often indicates a delay in the behavior of one of the twins. In contrast, a sequence of mismatches typically points to a deviation in the behavior of one or both twins. We believe this classification can be refined to provide more informative feedback, such as alerts about excessive noise in some sensors or instances of stuttering.

Moreover, in cases where snapshots contain multiple attributes, we can leverage domain-specific knowledge to deliver an even more precise diagnosis based on the attributes associated with the misalignments. For instance, in the Lego Car example, if the misalignments are caused by discrepancies linked to a specific sensor value, we could provide targeted feedback regarding the malfunction of that sensor. This line of future work includes improving our current classification system and automatically diagnosing and classifying various issues.

Our proposal aims to analyze specific scenarios for a given system and assess the fidelity of those scenarios. However, for complex CPSs that interact with stochastic environments, it is impractical to consider every possible scenario the system may encounter during the validation process. To address this, we have conducted preliminary work that involves applying the algorithm in real time and analyzing the data snapshots using sliding windows (Muñoz, Troya, and Vallecillo 2024). Therefore, this second line of future work aims to implement this approach within an architecture that allows for distributed processing of alignments and the configuration of alerts based on the existing diagnosis system and applied thresholds for fidelity metrics. The development of this architecture, along with establishing fidelity metric thresholds for alerts, will be part of our future work. Ultimately, we aim to create a comprehensive real-time tool for monitoring and validating digital twins.

Finally, we plan to incorporate **uncertainty** to apply our recently published tool (Fernández-Candel et al. 2024) to represent attribute values as random variables instead of the traditional crisp data types. This could help us represent the stochastic behavior of physical systems more realistically and deal with their inherent uncertainty, resulting in more precise measurements of the system's fidelity (Bertoa et al. 2020; Fernández-Candel et al. 2024; Jézéquel and Vallecillo 2023).

9.3 Conclusions

In conclusion, this thesis aims to contribute to defining key terms related to DTs and address a recognized gap in both industry and academia regarding the need for tools that enable the V&V of DTSs (Muctadir et al. 2023). Specifically, the thesis tackles the challenge of assessing the consistency of behavior between the DT and the PT, which we refer to as *fidelity*. Ensuring a sufficient degree of fidelity between the twins is crucial for the effectiveness of DTSs, which require identical or similar behavior. Our method provides a solution that supports the analysis of these aspects and facilitates reasoning about the underlying causes of discrepancies.

BIBLIOGRAPHY

Aalst, Wil M. P. van der (2016). *Process Mining – Data Science in Action*. 2nd edition. Springer. DOI: 10.1007/978-3-662-49851-4 (cited on page 103).

ABB (2024). *GoFa CRB 15000*. URL: https://new.abb.com/products/robotics/es/robots/robots-colaborativos/gofa-crb-15000 (cited on page 70).

Acker, Bert Van et al. (2019). "Valid (Re-)Use of Models-of-the-Physics in Cyber-Physical Systems Using Validity Frames". In: *Proc. of SpringSim'19*. IEEE, pages 1–12. DOI: 10.23919/SpringSim.2019.8732858 (cited on page 102).

Ahlgren, John et al. (2021). "Facebook's Cyber-Cyber and Cyber-Physical Digital Twins". In: *Proc. of EASE'21*. ACM, pages 1–9. DOI: 10.1145/3463274.3463275 (cited on page 102).

Alimadadi, Saba, Ali Mesbah, and Karthik Pattabiraman (2018). "Inferring hierarchical motifs from execution traces". In: *Proc. of ICSE'18*. ACM, pages 776–787. DOI: 10.1145/3180155.3180216 (cited on pages 103–104).

Altschul, Stephen F., Bruce W. Erickson, and Henry Leung (1986). "Local Alignment (with Affine Gap Weights)". In: *Encyclopedia of Algorithms*. Edited by Ming-Yang Kao. Boston, MA: Springer US, pages 459–461. ISBN: 978-0-387-30162-4. DOI: 10.1007/978-0-387-30162-4_207 (cited on page 31).

Altschul, Stephen F. et al. (1990). "Basic local alignment search tool". In: *Journal of Molecular Biology* 215.3, pages 403–410. ISSN: 0022-2836. DOI: 10.1016/S0022-2836(05)80360-2. URL: https://blast.ncbi.nlm.nih.gov/ (cited on pages 24, 30, 32, 127, 131).

Arrieta, Aitor (2021). "Multi-Fidelity Digital Twins: a Means for Better Cyber-Physical Systems Testing?" In: *CoRR* abs/2101.05697. arXiv: 2101.05697 (cited on pages 17, 56).

Balci, Osman (1994). "Validation, verification, and testing techniques throughout the life cycle of a simulation study". In: *Ann. Oper. Res.* 53.1, pages 121–173. DOI: 10.1007/BF02136828 (cited on page 101).

Ball, Alexander A. (1984). "Reparametrization and its application in computer-aided geometric design". In: *International Journal for Numerical Methods in Engineering* 20, pages 197–216. DOI: 10.1002/nme.1620200202 (cited on page 33).

Banks, Jerry (1998). *Handbook of simulation - principles, methodology, advances, applications, and practice*. Wiley. ISBN: 9780471134039 (cited on page 101).

Bard, Gregory V. (2007). "Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric". In: *Proc. of ACSW'07*. Volume 68. CRPIT. Australian Computer Society, pages 117–124 (cited on page 103).

Bellman, Richard (1957). *Dynamic Programming*. Dover Publications. ISBN: 9780486428093 (cited on page 28).

Bertoa, Manuel F. et al. (2020). "Incorporating measurement uncertainty into OCL/UML primitive datatypes". In: *Softw. Syst. Model.* 19.5, pages 1163–1189. DOI: 10.1007/s10270-019-00741-0 (cited on pages 107, 142).

Bézivin, Jean (2005). "On the Unification Power of Models". In: *Software and Systems Modeling* 4.2, pages 171–188. DOI: 10.1007/s10270-005-0079-0 (cited on page 18).

Bojarczuk, K. et al. (2021). "Measurement Challenges for Cyber Cyber Digital Twins: Experiences from the Deployment of Facebook's WW Simulation System". In: *Proc. of ESEM'21*. Bari, Italy: ACM. ISBN: 9781450386654. DOI: 10.1145/3475716.3484196 (cited on page 102).

Bordeleau, F. et al. (2020). "Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges". In: *Proc. of ICSMM'20*. Springer, pages 43–54. DOI: 10.1007/978-3-030-58167-1_4 (cited on pages 2, 41, 120).

Bose, R. P. Jagadeesh Chandra and Wil M. P. van der Aalst (2012). "Process diagnostics using trace alignment: Opportunities, issues, and challenges". In: *Inf. Syst.* 37.2, pages 117–141. DOI: 10.1016/j.is.2011.08.003 (cited on page 103).

Buschmann, F. et al. (1996). *Pattern–Oriented Software Architecture: A System of Patterns*. John Wiley & Sons. ISBN: 978-047-195-869-7 (cited on pages 37, 40).

Büttner, Fabian and Martin Gogolla (2014). "On OCL-based imperative languages". In: *Sci. Comput. Program.* 92, pages 162–178. DOI: 10.1016/j.scico.2013.10.003 (cited on pages 36, 56).

Camargo, Manuel, Marlon Dumas, and Oscar González (2020). "Automated discovery of business process simulation models from event logs". In: *Decis. Support Syst.* 134, page 113284. DOI: 10.1016/j.dss.2020.113284 (cited on page 103).

Cleophas, Loek et al. (2023). "Model-Driven Engineering of Digital Twins (Dagstuhl Seminar 22362)". In: *Dagstuhl Reports* 12.9. Edited by Benoit Combemale, Bernhard

Rumpe, and Steffen Zschaler, pages 20–40. ISSN: 2192-5283. URL: https://drops.dagstuhl.de/opus/volltexte/2023/17808 (cited on page 38).

Components101 (2018). *DHT22 Temperature and Humidity Sensors specification*. URL: https://components101.com/sensors/dht22-pinout-specs-datasheet (cited on page 65).

Dalibor, Manuela et al. (2020). "Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits". In: *Proc. of ER'20*. Volume 12400. LNCS. Springer, pages 377–387 (cited on page 14).

Dalibor, Manuela et al. (2022). "A Cross-Domain Systematic Mapping Study on Software Engineering for Digital Twins". In: *J. Syst. Softw.* 193, page 111361. DOI: 10.1016/j.jss.2022.111361 (cited on pages 1–2, 13, 15–16, 41, 119–120).

Digital Twin Consortium (2021). *Glossary of Digital Twins*. https://www.digitaltwinconsortium.org/glossary/index.htm (cited on pages 1, 13, 119).

Efrat, Alon et al. (2002). "New Similarity Measures between Polylines with Applications to Morphing and Polygon Sweeping". In: *Discret. Comput. Geom.* 28.4, pages 535–569. DOI: 10.1007/s00454-002-2886-1 (cited on page 135).

Example Digital Twin: The Incubator (2023). URL: https://github.com/INTO-CPS-Association/example_digital-twin_incubator (cited on pages 63–64).

Feng, Hao et al. (2021). *The Incubator Case Study for Digital Twin Engineering*. DOI: 10.48550/arXiv.2102.10390. arXiv: 2102.10390 [eess.SY] (cited on pages 63–64).

Fernández-Candel, Carlos Javier et al. (2024). "UTypes: A library for uncertain datatypes in Python". In: *SoftwareX* 26, page 101676. ISSN: 2352-7110. DOI: 10.1016/j.softx.2024.101676 (cited on pages 45, 107, 129, 131, 142).

Fu, Ada Wai-Chee et al. (2008). "Scaling and time warping in time series querying". In: *The VLDB Journal* 17, pages 899–921 (cited on page 81).

Gartner, Inc. (2018). *Gartner Identifies Five Emerging Technology Trends That Will Blur the Lines Between Human and Machine*. Accessed: 2024-09-27. URL: https://www.gartner.com/en/newsroom/press-releases/2018-08-20-gartner-identifies-five-emerging-technology-trends-that-will-blur-the-lines-between-human-and-machine (cited on pages 2, 120).

Gogolla, Martin, Jörn Bohling, and Mark Richters (2005). "Validating UML and OCL Models in USE by Automatic Snapshot Generation". In: *SoSyM* 4.4, pages 386–398 (cited on pages 19, 128).

Gogolla, Martin, Fabian Büttner, and Mark Richters (2007). "USE: A UML-Based Specification Environment for Validating UML and OCL". In: *Sci. Comput. Program.* 69, pages 27–34. ISSN: 0167-6423 (cited on pages 36, 56, 75).

Gong, Liang et al. (2022). "Motion Similarity Evaluation between Human and a Tri-Co Robot during Real-Time Imitation with a Trajectory Dynamic Time Warping Model". In: *Sensors* 22.5, page 1968. DOI: 10.3390/s22051968 (cited on page 104).

Grieves, Michael (2014). *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. White Paper. Florida,US: Florida Institute of Technology (cited on pages 2, 12, 14, 120).

Grieves, Michael and John Vickers (2017). "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems". In: *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. Springer, pages 85–113. ISBN: 978-3-319-38756-7. DOI: 10.1007/978-3-319-38756-7_4 (cited on pages 1, 11, 14, 119).

Gross, David C. (1999). "Report from the fidelity implementation study group". In: *Proc. of the Fall Simulation Interoperability Workshops* (cited on pages 3, 16, 18, 121).

Hai, Rihan, Christoph Quix, and Matthias Jarke (2021). "Data lake concept and systems: a survey". In: *CoRR* abs/2106.09592. arXiv: 2106.09592 (cited on pages 3, 37, 121).

Henikoff, Steven and Jorja G Henikoff (1992). "Amino acid substitution matrices from protein blocks." In: *Proceedings of the National Academy of Sciences* 89.22, pages 10915–10919. DOI: 10.1073/pnas.89.22.10915 (cited on page 26).

Hoare, C. A. R. (1972). "Proof of Correctness of Data Representations". In: *Acta Informatica* 1, pages 271–281. DOI: 10.1007/BF00289507 (cited on page 17).

Hotelling, Harold (1931). "The Generalization of Student's Ratio". In: *The Annals of Mathematical Statistics* 2.3, pages 360 –378. DOI: 10.1214/aoms/1177732979 (cited on page 102).

ISO 23247:2021 (Oct. 2021). *Automation systems and integration – Digital Twin Framework for manufacturing*. International Organization for Standardization (ISO) (cited on page 14).

ISO/IEC 42010 (2011). *Systems and software engineering – Architecture description*. ISO/IEC (cited on page 16).

ISO/IEC IS 10746 (1998–2010). *Information Technology – Open Distributed Processing – Reference Model. Parts 1 to 4* (cited on page 17).

Johannesson, Paul and Erik Perjons (2014). *An Introduction to Design Science*. Springer. ISBN: 978-3-319-10631-1. DOI: 10.1007/978-3-319-10632-8 (cited on pages 4–5, 123).

Jézéquel, Jean-Marc and Antonio Vallecillo (2023). "Uncertainty-aware Simulation of Adaptive Systems". In: *ACM Trans. Model. Comput. Simul.* 33.3, 8:1–8:19. DOI: 10.1145/3589517 (cited on pages 107, 142).

Khan, Adnan et al. (2018). "Digital Twin for Legacy Systems: Simulation Model Testing and Validation". In: *Proc. of CASE'18*. IEEE, pages 421–426. DOI: 10.1109/COASE.2018.8560338 (cited on page 38).

Korf, Ian, Mark Yandell, and Joseph A. Bedell (2003). *BLAST - an essential guide to the basic local alignment search tool*. O'Reilly. ISBN: 978-0-596-00299-2 (cited on page 32).

Kritzinger, Werner et al. (2018). "Digital Twin in manufacturing: A categorical literature review and classification". In: *IFAC-PapersOnLine* 51.11. Proc. of INCOM'18, pages 1016–1022. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.08.474 (cited on pages 3, 12–13).

Kuhn, Harold W. (2010). "The Hungarian Method for the Assignment Problem". In: *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, pages 29–47. DOI: 10.1007/978-3-540-68279-0_2 (cited on page 103).

Lamport, Leslie (Apr. 1983). "Specifying Concurrent Program Modules". In: *ACM Trans. Program. Lang. Syst.* 5.2, 190–222. ISSN: 0164-0925. DOI: 10.1145/69624.357207 (cited on page 17).

Langer, Philip, Tanja Mayerhofer, and Gerti Kappel (2014). "Semantic Model Differencing Utilizing Behavioral Semantics Specifications". In: *Proc. of MODELS'14*. Volume 8767. LNCS. Springer, pages 116–132. DOI: 10.1007/978-3-319-11653-2_8 (cited on page 103).

Lee, Edward A. and Marjan Sirjani (2018). "What Good are Models?" In: *Proc. of FACS'18*. Volume 11222. LNCS. Springer, pages 3–31. DOI: 10.1007/978-3-030-02146-7_1 (cited on pages 17–19, 70).

Leroy, Dorian et al. (2018). "Trace Comprehension Operators for Executable DSLs". In: *Proc. of ECMFA'18*. Volume 10890. LNCS. Springer, pages 293–310. DOI: 10.1007/978-3-319-92997-2_19 (cited on pages 103–104).

Levenshtein, VI (1966). "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". In: *Soviet Physics Doklady* 10, page 707 (cited on pages 27, 103).

Li, Huanhuan et al. (2020). "Adaptively constrained dynamic time warping for time series classification and clustering". In: *Information Sciences* 534, pages 97–116. ISSN: 0020-0255. DOI: 10.1016/j.ins.2020.04.009 (cited on page 81).

Lugaresi, Giovanni et al. (2019). "Real-time Validation of Digital Models for Manufacturing Systems: a Novel Signal-processing-based Approach". In: *Proc. of CASE'19*. IEEE, pages 450–455. DOI: 10.1109/COASE.2019.8843082 (cited on pages 103, 142).

Lugaresi, Giovanni et al. (2022). "Online Validation of Simulation-Based Digital Twins Exploiting Time Series Analysis". In: *Proc. of WSC'22*. IEEE, pages 2912–2923. DOI: 10.1109/WSC57314.2022.10015346 (cited on pages 79, 89, 103–104).

— (2023). "Online validation of digital twins for manufacturing systems". In: *Comput. Ind.* 150, page 103942. DOI: 10.1016/j.compind.2023.103942 (cited on pages 3, 81, 103–104, 121, 138, 142).

Madni, Azad M., Carla C. Madni, and Scott D. Lucero (2019). "Leveraging Digital Twin Technology in Model-Based Systems Engineering". In: *Systems* 7.1, page 7. DOI: 10.3390/systems7010007 (cited on page 16).

Manna, Zohar et al. (1997). "Abstraction and Modular Verification of Infinite-State Reactive Systems". In: *Proc. of RTSE'97*. Volume 1526. LNCS. Springer, pages 273–292. DOI: 10.1007/10692867_13 (cited on page 17).

Marquardt, Teresa, Catherine Cleophas, and Lucy E. Morgan (2021). "Indolence is Fatal: Research Opportunities in Designing Digital Shadows and Twins for Decision Support". In: *Proc. of WSC'21*. IEEE, pages 1–11. DOI: 10.1109/WSC52266.2021.9715332 (cited on pages 101, 103).

Masood, Muhammad Umar and Mahdi Haghshenas-Jaryani (2021). "A Study on the Feasibility of Robotic Harvesting for Chile Pepper". In: *Robotics* 10.3, page 94. DOI: 10.3390/robotics10030094 (cited on pages 70, 74).

Modoni, Gianfranco E et al. (2019). "Synchronizing physical and digital factory: benefits and technical challenges". In: *Procedia Cirp* 79, pages 472–477 (cited on page 13).

Moon, Il-Chul and Jeong-Hee Hong (2013). "Theoretic interplay between abstraction, resolution, and fidelity in model information". In: *Proc. of WSC'13*. IEEE, pages 1283–1291. DOI: 10.1109/WSC.2013.6721515 (cited on pages 17–18).

Mori, Usue, Alexander Mendiburu, and José Antonio Lozano (2016). "Distance Measures for Time Series in R: The TSdist Package". In: *R Journal* 8.2, page 451. DOI: 10.32614/rj-2016-058 (cited on page 32).

Mount, D.W. (2004). *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Series. Cold Spring Harbor Laboratory Press. ISBN: 9780879697129. URL: https://books.google.es/books?id=bvY21DGa1OwC (cited on page 26).

Muñoz, Paula, Javier Troya, and Antonio Vallecillo (2024). "Towards Measuring Digital Twins Fidelity at Runtime". In: *Proceedings of the 1st International Conference on Engineering Digital Twins (EDTconf 2024) at ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. MODELS Companion '24. Linz, Austria: Association for Computing Machinery, 507–512. DOI: 10.1145/3652620.3688267 (cited on pages 106–107).

Muctadir, Hossain Muhammad et al. (June 2023). "Current Trends in Digital Twin Development, Maintenance, and Operation: An Interview Study". In: *CoRR* abs/2306.10085. arXiv: 2306.10085 [cs.SE] (cited on pages 107, 142).

Muñoz, Paula, Javier Troya, and Antonio Vallecillo (2021). "Using UML and OCL Models to Realize High-Level Digital Twins". In: *Proc. of ModDiT2021@MODELS'21*. IEEE, pages 212–220. DOI: 10.1109/MODELS-C53483.2021.00037 (cited on pages 35–36, 124).

— (2023). "A Conceptual Architecture for Building Digital Twins". In: *Post Proceedings of the STAF 2023 Workshops TTC 2023, MeSS 2023 and AgileMDE 2023, Leicester, United Kingdom, July 18, 2023 and June 21, 2023*. Volume 3620. CEUR Workshop Proceedings. URL: https://hdl.handle.net/10630/27428 (cited on pages 1–2, 35–36, 38–40, 124–125).

Muñoz, Paula et al. (2022). "Using trace alignments for measuring the similarity between a physical and its digital twin". In: *Proc. of ModDiT@MODELS'22*. ACM, pages 503–510. DOI: 10.1145/3550356.3563135 (cited on pages 74, 104).

Muñoz, Paula et al. (2023a). *Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments – Git repository*. URL: https://github.com/atenearesearchgroup/fidelity-measure-for-dts (cited on pages 56, 71, 97).

— (2023b). *Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments: Elevator Technical Report*. URL: https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_Elevator.pdf (cited on page 44).

— (2023c). *Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments: NXT Lego Mindstorms Car*. URL: https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_NXT_Car.pdf (cited on page 76).

Needleman, Saul B. and Christian D. Wunsch (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *J. Molecular Biology* 48.3, pages 443–453. ISSN: 0022-2836. DOI: 10.1016/0022-2836(70)90057-4 (cited on pages 24–25, 27, 127, 131).

Niryo (2024). *Ned2*. URL: https://niryo.com/product/6-axis-robotic-arm/ (cited on page 70).

Object Management Group (Feb. 2014). *Object Constraint Language (OCL) Specification. Version 2.4*. OMG Document formal/2014-02-03 (cited on page 36).

— (Mar. 2015). *Unified Modeling Language (UML) Specification. Version 2.5*. OMG document formal/2015-03-01 (cited on page 36).

Peters Research (2023). *Elevate Software*. URL: https://peters-research.com/index.php/elevate/ (cited on pages 42, 127).

Pérez-Porras, Daniel et al. (2022). "Key-Value vs Graph-based data lakes for realizing Digital Twin systems". In: *Proc. of MeSS@STAF'22*. URL: https://hdl.handle.net/10630/24677 (cited on page 35).

Rothenberg, J. (1989). "The Nature of Modeling". In: *Artificial Intelligence, Simulation, and Modeling*. Edited by L.E. William, K.A. Loparo, and N.R. Nelson. John Wiley and Sons, 75—92 (cited on page 16).

Sakoe, Hiroaki and Seibi Chiba (1971). "A Dynamic Programming Approach to Continuous Speech Recognition". In: *Proc. of the 7th International Congress on Acoustics*. Volume 3. Akadémiai Kiadó, pages 65–69 (cited on pages 79–81, 94).

— (1978). "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE transactions on acoustics, speech, and signal processing* 26.1, pages 43–49 (cited on pages 81, 138).

Salvador, Stan and Philip Chan (2007). "Toward accurate dynamic time warping in linear time and space". In: *Intelligent Data Analysis* 11.5, pages 561–580 (cited on page 81).

Sargent, Robert G. (2013). "Verification and validation of simulation models". In: *J. Simulation* 7.1, pages 12–24. DOI: 10.1057/jos.2012.20 (cited on pages 3, 121).

"Sequence Alignment (I)" (2021). In: *Algorithms in Bioinformatics*. John Wiley & Sons, Ltd. Chapter 3, pages 51–97. ISBN: 9781119698005. DOI: 10.1002/9781119698005.ch3 (cited on pages 24–25).

Shannon, Claude Elwood (1948). "A mathematical theory of communication". In: *The Bell system technical journal* 27.3, pages 379–423 (cited on pages 49, 132).

Smith, T.F. and M.S. Waterman (1981). "Identification of common molecular subsequences". In: *Journal of Molecular Biology* 147.1, pages 195–197. ISSN: 0022-2836. DOI: 10.1016/0022-2836(81)90087-5 (cited on pages 25, 30).

Snow, Dennis A., editor (2003). *Plant Engineer's Reference Book*. 2nd edition. Elsevier. ISBN: 978-0-7506-4452-5. DOI: 10.1016/B978-0-7506-4452-5.X5052-4 (cited on pages 56, 136).

Tao, Fei et al. (2018). "Digital twin driven prognostics and health management for complex equipment". In: *CIRP Annals* 67.1, pages 169–172. ISSN: 0007-8506. DOI: 10.1016/j.cirp.2018.04.055 (cited on pages 14–15, 37).

Tao, Fei et al. (2019). "Digital Twin in Industry: State-of-the-Art". In: *IEEE Trans. Ind. Informatics* 15.4, pages 2405–2415 (cited on pages 14, 37).

Tinkerkit (2021). *Arduino Tinkerkit Braccio Robot*. URL: https://store.arduino.cc/products/tinkerkit-braccio-robot (cited on page 70).

Wang, Dafang, Robert M. Kirby, and Chris R. Johnson (2010). "Resolution Strategies for the Finite-Element-Based Solution of the ECG Inverse Problem". In: *IEEE Transactions on Biomedical Engineering* 57.2, pages 220–237. DOI: 10.1109/TBME.2009.2024928 (cited on page 16).

Wieringa, Roel J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer. ISBN: 978-3-662-43838-1. DOI: 10.1007/978-3-662-43839-8 (cited on pages 4–5, 122–123).

Wolny, Sabine et al. (2019). "Model-driven Runtime State Identification". In: *Proc. of EMISA'19*. Volume P-304. LNI. Gesellschaft für Informatik e.V., pages 29–44 (cited on page 103).

Worden, K. et al. (2020). "On Digital Twins, Mirrors, and Virtualizations: Frameworks for Model Verification and Validation". In: *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering* 6.3. ISSN: 2332-9017. DOI: 10.1115/1.4046740 (cited on page 102).

Wortmann, Andreas (2023). *Digital Twin Definitions*. URL: https://awortmann.github.io/research/digital_twin_definitions (cited on pages 2, 120).

Zhidchenko, V et al. (2018). "Faster than real-time simulation of mobile crane dynamics using digital twin concept". In: *Journal of Physics: Conference Series* 1096, page 012071. DOI: 10.1088/1742-6596/1096/1/012071 (cited on page 70).

A

RESUMEN

Los *Gemelos Digitales* (por sus siglas en inglés, DTs) han demostrado ser herramientas eficaces para la optimización del rendimiento de sistemas ciberfísicos, procesos o servicios de software (Dalibor et al. 2022; Digital Twin Consortium 2021). La idea central de los DTs consiste en la replicación digitalizada de un sistema físico existente, que se denomina *Gemelo Físico* (por sus siglas en inglés, PT).

Este concepto surgió durante la misión Apollo de la NASA, donde los ingenieros desarrollaron réplicas físicas de los sistemas utilizados. Su objetivo era proporcionar apoyo a los astronautas, proveyendo recomendaciones lo más precisas posible reproduciendo el estado de la nave con las réplicas (Grieves y Vickers 2017). Esto permitía garantizar la seguridad de los astronautas y la integridad de la nave, dando recomendaciones muy precisas que habían sido probadas y contrastadas, simulando distintos escenarios en el laboratorio de Tierra.

Durante estas misiones, las réplicas eran dispositivos físicos y la sincronización se realizaba manualmente. Sin embargo, una vez que los beneficios del enfoque fueron patentes, las réplicas pasaron a ser digitales, mediante simulaciones o modelos analíticos, y la sincronización se automatizó. Actualmente, los DTs juegan un rol esencial en los sistemas modernos, representando distintos aspectos de sus PTs, como restricciones físicas, apariencia y comportamiento, a distintos niveles de fidelidad y distintas frecuencias de sincronización (Dalibor et al. 2022).

Los campos de aplicación de los gemelos digitales son muy diversos. Se pueden emplear durante la fase de diseño de nuevos sistemas, simulando comportamientos potenciales que ayuden a construir prototipos, o se pueden desarrollar a la vez que los PTs para aumentar la eficiencia operativa. Una vez el sistema se ha construido, también podemos usar los DTs para tareas de monitorización y optimización del comportamiento del sistema, permitiendo la toma de decisiones en tiempo real y el mantenimiento predictivo.

A.1 Planteamiento del Problema

Una de las principales ventajas de los Gemelos Digitales es la replicación de las características del sistema físico, que les permite realizar simulaciones avanzadas, monitorización y análisis predictivo. Sin embargo, si estos sistemas no son sometidos a validación, no hay garantía de que los DTs repliquen los sistemas con suficiente precisión, afectando a su valor y fiabilidad. En este contexto, la ingeniería de los DTs se convierte en un elemento esencial (Bordeleau et al. 2020). Actualmente, una gran parte del trabajo de la literatura se concentra en construir y desplegar DTs, pero encontramos escasos trabajos en el área de la validación (Dalibor et al. 2022). Por lo tanto, es esencial trabajar en enfoques sistemáticos que evalúen la fidelidad del DT a lo largo de su ciclo de vida, asegurando que siga siendo una réplica fiel del sistema real.

En este contexto, aunque el término "Gemelo Digital" fue introducido hace más de una década por Grieves en su informe técnico (Grieves 2014), ha sido recientemente cuando el término ha captado la atención de la academia y la industria (Gartner, Inc. 2018). Por esta razón, la definición precisa de DT continúa siendo una labor en curso, con más de cien definiciones diferentes en la literatura (Wortmann 2023). Este amplio ecosistema de definiciones dificulta el entendimiento de la comunidad. Algunos investigadores ven los DTs como réplicas virtuales de los sistemas físicos, mientras que otros amplían la definición para incluir conexiones, bases de datos o incluso servicios como la detección de anomalías. Esta falta de alineamiento dificulta la comunicación efectiva, ralentiza la consolidación del conocimiento y pone trabas en la evaluación y comparación de nuevas investigaciones.

Esto hace imperativo establecer un conjunto claro de requisitos y definiciones precisas para los distintos términos y elementos arquitectónicos de un DT antes de validarlo.

A.2 Contribuciones

Esta tesis presenta dos contribuciones clave: primero, una arquitectura modular y un conjunto de definiciones precisas que aclaran la terminología fundamental de los DTs; segundo, un algoritmo para el alineamiento de trazas de comportamiento acompañado de un conjunto de métricas que evalúan el nivel de fidelidad de los gemelos físico y digital.

Estas contribuciones se describen a continuación:

A.2.1 Arquitectura Conceptual para Gemelos Digitales

En esta tesis se introduce una arquitectura modular basada en el concepto de *Sistema de Gemelo Digital* (DTS, por sus siglas en inglés). Un DTS incluye no sólo el sistema físico y su réplica digital, sino que también incluye las interacciones entra ambos. El gemelo digital puede tomar diferentes formas: algoritmos, modelos de simulación o modelos analíticos con ecuaciones diferenciales, entre otros. Sin embargo, es importante señalar que estos elementos, por sí solos no pueden considerarse un DT. Sólo pueden considerarse así cuando forman parte de un DTS, donde mantienen una sincronización bidireccional automática con el PT e intercambian datos y comandos para optimizar su rendimiento.

Uno de los objetivos de esta tesis es proporcionar una definición precisa de los conceptos clave que conforman un DTS. La arquitectura propuesta también se centra en facilitar la escalabilidad y la composición de los DTs. Para respaldar esto, la arquitectura se construye alrededor de un *Data Lake* (Hai, Quix y Jarke 2021), es decir, un repositorio centralizado diseñado para almacenar, procesar y asegurar grandes volúmenes de datos estructurados, semiestructurados y no estructurados. Un conjunto de controladores accede al DL para gestionar y coordinar el flujo de datos y comandos entre los elementos del DTS.

En esta contribución, definimos la arquitectura y sus componentes principales y la comparamos con otras propuestas de la literatura.

A.2.2 Algoritmo de Alineamiento de Trazas para Medir la Fidelidad

En esta tesis, proponemos un método offline para evaluar la fidelidad de los gemelos digitales. Típicamente, la validación de modelos de comportamiento supone comparar su salida con la del sistema físico un sistema físico al que representa (Sargent 2013). Los resultados de esta comparación indican el nivel de precisión del modelo (Lugaresi et al. 2023). Más específicamente, nuestra propuesta mide el nivel de fidelidad del gemelo digital con respecto al gemelo físico.

La *fidelidad* puede referirse tanto a la estructura como al comportamiento del sistema (Gross 1999). En nuestro contexto, nos centraremos en validar la fidelidad del comportamiento. Nuestra contribución evalúa la fidelidad comparando las trazas de comportamiento de ambos gemelos. En este contexto, una traza es una secuencia de *snapshots*, en los que cada uno representa el estado del sistema en un momento específico.

Para comparar las trazas, empleamos tres pasos. Primero, definimos una función de comparación para evaluar si dos *snapshots* son lo suficientemente similares. Segundo, usamos un algoritmo de alineamiento de trazas para emparejar los estados equivalentes

alcanzados por ambos gemelos. Para comparar dichos estados, empleamos un umbral para cada propiedad de interés, que es la diferencia máxima que puede haber entre que el algoritmo los empareje. Finalmente, medimos la fidelidad en base a los resultados del alineamiento, empleando dos métricas: el porcentaje de *snapshots* alineados y las distancias media y máxima entre las trazas alineadas.

A.3 Metodología de Investigación

Design Science (DS) se define como el estudio científico y la creación de artefactos desarrollados y utilizados por la gente par resolver problemas prácticos y de interés general (Wieringa 2014). Esta disciplina se diferencia de las ciencia puras en que su enfoque va más allá de describir el mundo, ya que busca crear soluciones que respondan a las necesidades humanas. En esta tesis, aplicamos la metodología de DS para desarrollar un evaluar un artefacto que mejora el estado del arte en el campo de los gemelos digitales.

Seguimos el marco introducido por Johannesson y Perjons (Wieringa 2014) que estructura el proceso de investigación en fases distintas. Cada fase fue aplicada sistemáticamente en nuestra investigación de la siguiente forma:

1. **Explicación del Problema:** Comenzamos analizando a fondo los desafíos que implican la validación de la fidelidad del comportamiento en los DTSs. Este análisis nos ayudó a definir el problema con precisión y justificar su relevancia práctica en el contexto del diseño y desarrollo de DTSs.
2. **Definición de Requisitos:** Basándonos en el análisis del problema, establecimos un conjunto de requisitos para el artefacto, centrándonos en métodos de comparación de trazas de comportamiento. Estos requisitos guiaron el diseño de una solución que garantiza un alineamiento precisa de trazas.
3. **Diseño y Desarrollo del Artefacto:** Diseñamos y desarrollamos una herramienta de alineamiento de trazas basada en técnicas de bioinformática para cumplir con los requisitos definidos. Específicamente, adaptamos el algoritmo de alineamiento global de Needleman-Wunsch e incorporamos técnicas del algoritmo BLAST, que tradicionalmente se utilizan para el alineamiento de secuencias en investigación biológica.
4. **Evaluación del Artefacto:** Para demostrar la viabilidad y el potencial del artefacto, lo aplicamos a cuatro sistemas ciberfísicos diferentes.

En las primeras etapas del proyecto, utilizamos un **caso de estudio** para investigar el problema y definir los requisitos. Los casos de estudio son ideales para explorar fenómenos complejos en su contexto natural y nos ayudaron a identificar problemas específicos en el

alineamiento y la medición de la fidelidad de los DTSs. Para este propósito, seleccionamos el caso de estudio de un ascensor estudiado en su entorno operativo normal sin manipulación.

En las etapas posteriores, incluyendo la validación y demostración del artefacto, combinamos casos de estudio con **experimentos**. Los experimentos fueron esenciales para probar empíricamente las relaciones de causa y efecto entre diferentes parámetros de configuración y el rendimiento del artefacto. Al manipular estos parámetros en un entorno controlado, pudimos evaluar la robustez y adaptabilidad de la solución propuesta.

Nuestro enfoque ejemplifica la **Exaptación** en DS, donde soluciones conocidas se extienden para abordar nuevos problemas. Adaptamos los métodos de alineamiento de la bioinformática, típicamente usados para secuencias biológicas, con éxito para alinear trazas de comportamiento en los DTSs.

Formulación del Problema. En DS, la formulación del problema debe ser precisa, mostrando la relevancia del estudio para una práctica o grupo (Johannesson y Perjons 2014). Con este fin, la pregunta de investigación, también llamada *problema técnico de investigación*, puede formularse según el esquema DS propuesto por Wieringa (2014). Usando el esquema de DS (Wieringa 2014), el objetivo de nuestro estudio es:

- Mejorar la validación de los Sistemas de Gemelos Digitales
- mediante el diseño de un algoritmo para comparar el comportamiento de dos gemelos de comportamiento supuestamente equivalente
- que proporcione indicadores para razonar sobre el grado de fidelidad de los dos gemelos
- para determinar si dos gemelos son suficientemente fieles entre sí para el propósito para el cual fueron concebidos.

A.4 Conceptualización del Gemelo Digital

Esta sección desarrolla la primera contribución de la tesis. Inicialmente, propusimos una *arquitectura conceptual* para definir y desplegar DTSs modularmente, que después fue refinada y convertida en un *framework* donde considerábamos elementos y retos adicionales.

Como parte de estos trabajos, introdujimos el concepto de DTS, que incluiría el DT, el PT, los servicios del sistema y las conexiones entre ellos. Al contrario que algunos trabajos de la literatura, nuestra propuesta identifica el gemelo digital como la réplica virtual del sistema, distinguiéndolo de otros elementos incluidos típicamente, como las conexiones y los servicios.

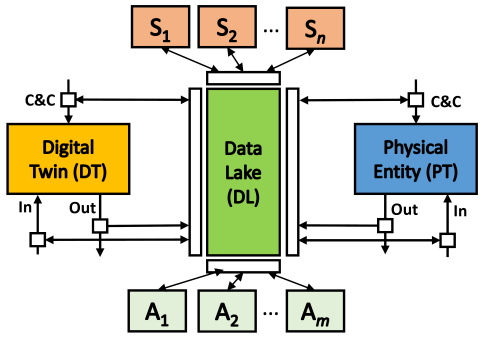


Figura A.1: Arquitectura Concpetual para un Sistema de Gemelo Digital (Muñoz, Troya y Vallecillo 2021).

A.4.1 Arquitectura Conceptual

En nuestros trabajos iniciales (Muñoz, Troya y Vallecillo 2021, 2023), presentamos una arquitectura para definir y desplegar DTs. El principal objetivo de esta propuesta era facilitar el prototipado de gemelos digitales de alto nivel, desarrollados usando UML y OCL.

La arquitectura conceptual está ilustrada en la Figura A.1. La entidad física interactúa con su entorno a través de *entradas* (`In`) y *salidas* (`Out`). Las entradas suelen estar generadas por sensores que detectan cambios en el entorno, mientras que las salidas son las reacciones del PT estos cambios en el entorno. Los PT también pueden recibir órdenes externas en forma de comandos que controlan su comporamiento y modifican el valor de sus parámetros. Esta interacción está representada por la flecha *Command & Control* (`C&C`).

El DT tendrá la misma interfaz de entrada y salida que el PT, ya que es su réplica digital. Las entradas del DT serán generadas por un entorno virtualizado o serán las mismas entradas de los sensores del PT. Esto último puede ayudarnos a estudiar si el DT reacciona de la misma forma que el PT cuando se le da la misma información de entrada. Dadas las mismas entradas, las salidas de ambos gemelos deberán ser equivalentes, dado un cierto nivel de fidelidad.

Un *Data Lake* (DL) facilitará la comunicación entre los gemelos, almacenando todos los datos del sistema. El resto de los componentes de la arquitectura usarán en DL para escribir y recopilar información de forma asíncrona. Estos componentes acceden al DL a través de *drivers* que transforman los datos a formatos que cada uno de los componentes es capaz de entender. En la Figura A.1, estos drivers están representados por cuadrados blancos alrededor del DL.

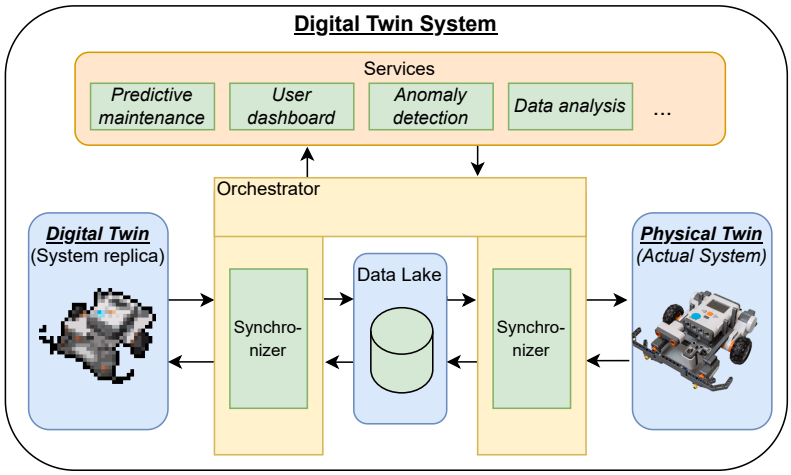


Figura A.2: Conceptual framework para Sistemas de Gemelos Digitales (Muñoz, Troya y Vallecillo 2023).

Los componentes de *Servicios* se encargan de implementar funcionalidades adicionales, como dashboards o algoritmos de detección de anomalías, predicción, optimización del comportamiento etc. Finalmente, los componentes de *Análisis* implementan varios tipos de pruebas sobre el sistema. Por ejemplo, un componente de este tipo podría monitorizar que las trazas de ambos gemelos sean fieles, tal y como lo hace nuestra propuesta.

A.4.2 Framework para Sistemas de Gemelos Digitales

Tras experimentar con distintas implementaciones, propusimos un refinamiento de la arquitectura inicial (Muñoz, Troya y Vallecillo 2023). La idea base de dicha arquitectura, sigue vigente en el framework propuesto: una estructura simétrica y modular que permite la comunicación entre los dos gemelos. Los otros componentes se centrarán en asegurar la interacción entre los gemelos y en mejorar el rendimiento.

Al igual que la propuesta anterior, esta arquitectura se construye entorno al *Data Lake*, al que se accede a través de un conjunto de *drivers*. La diferencia principal es que contamos con un *Orchestrator*, el elemento encargado de organizar el intercambio de datos y comandos en el DTS. En la Figura A.2, encontramos la propuesta del framework.

Las definiciones de los gemelos dentro del contexto del DTS son las siguientes:

1. **Gemelo Físico**. Es el sistema, servicio o producto cuyo comportamiento queremos optimizar usando la réplica digital. El estado del PT se monitoriza y los valores de sus propiedades de interés se almacenan periódicamente.

2. **Gemelo Digital**. Es la réplica sincronizada del PT, que se actualiza a intervalos regulares y está modelada a un nivel adecuado de fidelidad, capturando las propiedades de interés el PT. El DT representa las propiedades de interés y emula elcomportamiento utilizando datos y modelos.

Teniendo en cuenta las dos definiciones anteriores, definimos el Digital Twin System (DTS) como un sistema construido para un propósito concreto, compuesto por:

1. El **Gemelo Físico**.
2. Su réplica digital, es decir, el **Gemelo Digital**.
3. Las **conexiones y mecanismos de sincronización** que facilitan el intercambio de datos y comandos entre los gemelos.
4. El conjunto de **servicios** que permiten la explotación de los datos producidos e intercambiados por los gemelos y que contribuyen al cumplimiento de los objetivos del DTS.

Este nuevo framework mantiene la flexibilidad en la conexión de los elementos, teniendo únicamente que actualizar el componente de orquestración. De la misma forma, si múltiples DTs se tuvieran que interconectar y sincronizar su comportamiento, también se encargaría de ello el Orquestrador. Este framework también es compatible con tener diferentes gemelos digitales definidos a distintos niveles de fidelidad, y que se emplean dependiendo del propósito del sistema en un instante determinado. Además, también podemos sustituir el gemelo físico por otro modelo virtual para probar el sistema. Todos estos cambios son posibles modificando la interfaz del elemento de orquestración.

Todas estas definiciones y conceptos, se mantendrán consistentes a través de las próximas secciones, cada vez que hagamos referencia a los términos DT, PT y DTS.

A.5 Cuantificación de la Fidelidad de un Gemelo Digital

Esta Sección introduce nuestra propuesta para la validación de DTs, analizando su nivel de fidelidad. Nuestra propuesta analiza la fidelidad del comportamiento comparando las trazas de los dos gemelos. Esto se realiza a partir de tres pasos: Primero, representamos los estados del sistema como *snapshots* y definimos una función de comparación que determina cuando dos snapshots son suficientemente similares, a partir de un umbral, que llamaremos MAD (del inglés *Maximum Acceptable Distance*) para cada propiedad de interés del sistema.

En segundo lugar, alineamos los estados equivalentes que transitan los gemelos usando un algoritmo de alineamiento de trazas. Este algoritmo es una adaptación del

algoritmo *Needleman-Wunsch* (Needleman y Wunsch 1970) incluyendo optimizaciones de *BLAST* (Altschul et al. 1990). Ambos algoritmos son comúnmente usados en bioinformática para alinear secuencias de ADN.

Finalmente, medimos la fidelidad de ambos sistemas evaluando el nivel de alineamiento, expresado en términos del porcentaje de *snapshots* alineados, y calculando la distancia entre las trazas alineadas, que reflejan cómo de cercanos son los comportamientos de ambos gemelos. Presentaremos la propuesta a partir del caso de estudio de un ascensor localizado en uno de los edificios de la Universidad de Mondragón.

Para emplear nuestra propuesta, el sistema tiene que cumplir ciertas condiciones:

1. Las trazas deben exhibir un **comportamiento sincronizado** si ambas comenzasen su ejecución a la vez.
2. Ambos gemelos deben registrar los snapshots con el **mismo periodo de muestre**o, ya que trabajamos con el supuesto de que dos comportamientos idénticos deben coincidir en un 100 % de los snapshots. Para que esto se cumpla, los snapshots tienen que muestrearse con el mismo periodo.
3. Los snapshots deben **capturar el comportamiento del sistema** que se desea validar, permitiendo que analicemos la similitud a través de estados basándonos en las diferencias en los atributos de sus snapshots.

Cuando estas condiciones se cumplen, nuestra propuesta ofrece un conjunto de métricas que indican el nivel de fidelidad entre los sistemas. Identificando las partes de las trazas que contienen retrasos, inconsistencia o discrepancias.

A.5.1 Caso de Estudio: Ascensor en la Universidad de Mondragón

Para ilustrar y presentar nuestra solución, usaremos el caso de estudio de un ascensor localizado en la Universidad de Mondragón, en uno de sus edificios del Campus de Arrasate. El ascensor se traslada entre cinco plantas, numeradas del 0 al 4. Generalmente lo usan los estudiantes que no puedan usar las escaleras o el personal de mantenimiento para transportar cargas pesadas. Dado el rol crítico de esta infraestructura, la universidad ha decidido optimizar su operación y mantenimiento para ahorrar energía y evitar roturas.

Para conseguir esto, la Universidad ha implementado un DTS usando el simulador comercial Elevate (Peters Research 2023) como DT. Este simulador se puede configurar con diferentes parámetros físicos para emular el comportamiento de un ascensor en particular, concretamente, la aceleración entre plantas.

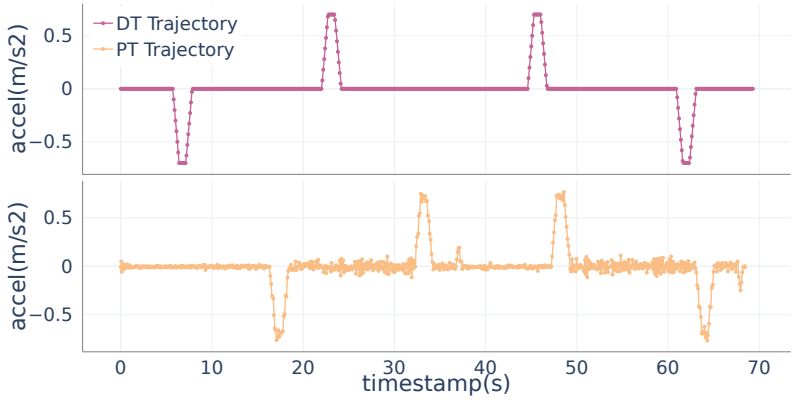


Figura A.3: Elevator traces of the PT and DT. Scenario (4-0-4), execution 04.

A partir de esta aceleración, somos capaces de deducir su velocidad y su posición para estimar la degradación de sus partes y verificar que la configuración es la óptima. Sin embargo, la Universidad desea averiguar hasta qué punto este simulador es capaz de replicar el comportamiento del ascensor real.

En este caso, nuestra propuesta será útil para comprobar si las secuencias de aceleración generadas por el simulador son lo suficientemente fieles para tomar el rol de DT. Compararemos la aceleración del DT con la del sistema real, que ha sido medida usando un acelerómetro.

A.5.2 Representación del Sistema

Para aplicar el algoritmo de alineamiento de trazas de comportamiento, necesitamos una representación discreta del comportamiento del sistema a lo largo del tiempo. Esto se consigue muestreando con un periodo constante las observaciones del estado del sistema.

En la Figura A.3, podemos ver el comportamiento del ascensor definido como una secuencia de *snapshots* que almacenan el valor de su aceleración, que es la propiedad de interés. Los snapshots son un conjunto de objetos, sus relaciones y sus atributos recopilados en un momento específico del tiempo (Gogolla, Bohling y Richters 2005). Las trazas son secuencias de snapshots ordenados cronológicamente.

La traza de la figura corresponde al escenario (4-0-4) en el que el ascensor comienza su recorrido en la planta 4, baja hasta la 0, para y vuelve a subir a la planta 4. En la figura, podemos ver 4 picos que podremos dividir en dos grupos. Los primeros dos picos representan la aceleración durante la bajada a la planta 0. El primer pico es una aceleración negativa que hace que el ascensor se desplace hacia abajo, mientras que el segundo pico es una aceleración de la misma magnitud en sentido contrario que se usa para detener

el ascensor cuando ha llegado a la planta baja. De la misma forma, los otros dos picos corresponden a la subida a la planta 4.

Estas aceleraciones son similares entre el gemelo físico y el digital. Sin embargo, hay un aspecto del sistema real que no se reproduce en el DT. En el sistema real, cada vez que el ascensor alcanza una planta concreta, hay una sutil aceleración que suaviza el frenado del ascensor y mejora la experiencia del usuario. Esta es una característica específica de este modelo de ascensor.

A.6 Función de Similitud

El primer paso para alinear estas las trazas es definir la función de similitud que puede comparar de forma efectiva los snapshots de los dos sistemas. En el contexto original de ambos algoritmos bioinformáticos, esta comparación es bastante directa, porque lidian con secuencias compuestas por caracteres. Sin embargo, los *snapshots* pueden ser estructuras complejas, ya que deben representar el estado del sistema.

El Algoritmo 3 describe la función de similitud "$\mathcal{S}()$" entre dos snapshots s_A y s_B. Asumimos que los dos snapshots están compuestos por los mismos k atributos de tipo $\{a_1, ..., a_k\}$. La función devuelve un número real en el rango [0..1] donde una puntuación de 1 significa que los snapshots son idénticos, y una de 0, que son completamente distintos.

La diferencia entre cada uno de los atributos se calcula en base a su tipo:

- Para **valores numéricos**, usamos como umbral la *Máxima Diferencia Aceptable* (en inglés MAD). Cada atributo tendrá un valor de MAD distinto dependiendo de su naturaleza, por ello, usamos un array de valores. Si la diferencia entre ambos valores supera el MAD, asignamos una puntuación de similitud de 0. En cambio, si la diferencia se encuentra dentro del rango del MAD, la puntuación iría incrementándose gradualmente conforme la diferencia entre los dos valores se reduce, tal y como se ve en la línea 6 del algoritmo. Una puntuación de 1 significa que los valores son iguales.
- En el caso de **valores Booleanos**, una puntuación de 1 significa que los valores son iguales, y 0, en caso de que sean diferentes.
- En el caso de **atributos enumerados o cadenas de caracteres**, podemos usar un enfoque similar o aplicar las operaciones de comparación definidas en (Fernández-Candel et al. 2024) (denominadas *uEquals()*) para calcular un valor de confianza entre 0 y 1. Para obtener un valor que nos indique la similaridad, bastaría con ajustarlo a 1-*uEquals()*.

Algorithm 3 Función de similitud entre dos snapshots.

```
1: i ← 1; result ← 0
2: while i ≤ k do
3:     if isNumerical(s_A.a_i) then
4:         dif ← abs(s_A.a_i − s_B.a_i)
5:         if dif < MAD_i then
6:             result ← result + (1 − dif / MAD_i)
7:         else
8:             result ← 0                          ▷ Mismatch if MAD_i exceeded
9:             break
10:        end if
11:    end if
12:    if isBoolean(s_A.a_i) then
13:        if s_A.a_i = s_B.a_i then
14:            result ← result + 1
15:        else
16:            result ← 0
17:            break
18:        end if
19:    end if
20:    if isString(s_A.a_i) then
21:        if equals(s_A.a_i, s_B.a_i) > MAD_i then
22:            result ← result + (1 − equals(s_A.a_i, s_B.a_i))
23:        else
24:            result ← 0
25:            break
26:        end if
27:    end if
28:    if lowComp(s_A.a_i) ∧ lowComp(s_B.a_i) then
29:        result ← result ∗ LCAW/2
30:    else if lowComp(s_A.a_i) ∨ lowComp(s_B.a_i) then
31:        result ← result ∗ LCAW
32:    end if
33:    i ← i + 1
34: end while
35: return result/k
```

También podríamos considerar la incertidumbre de los atributos durante estas comparaciones, teniendo en cuenta los potenciales errores de medida causados por los instrumentos. En este caso, un valor numérico con incertidumbre asociada podría emplearse, y las comparaciones se harían en base a un umbral de confianza (Fernández-Candel et al. 2024). Esta propuesta daría lugar a comparaciones más robustas en presencia de ruido, que es una circunstancia bastante común en sistemas ciberfísicos.

A.7 Algoritmo para el Alineamiento de Trazas

Resumen. El algoritmo de nuestra propuesta es una adaptación del algoritmo Needleman-Wunsch (Needleman y Wunsch 1970) que permite el alineamiento de no sólo secuencias de caracteres sino también de trazas de comportamiento. Esta adaptación incorpora dos optimizaciones de BLAST (Altschul et al. 1990):

a) evita iniciar alineamiento en **regiones de baja complejidad**, centrándose en las partes más características de la traza

b) emplea la técnica ***affine gap*** para agrupar los *gaps* de forma que se evite la excesiva alternancia, dando lugar a alineamientos más comprensibles.

Nuestro algoritmo mantiene la misma estructura básica de Needleman-Wunsch, pero está especialmente diseñado para alinear trazas de comportamiento. En primer lugar, emplea una función de similitud $\mathcal{S}()$ para comparar los elementos. Adicionalmente, la función de similitud se centra en las regiones más características enmascarando las de baja complejidad, ajustando la puntuación del algoritmo acorde. Para apliar esta función de similitud, el algoritmo introduce tres matrices adicionales que implementa el mecanismo de *affine gap*.

Parámetros. Hay cuatro parámetros principales que permiten adaptar el comportamiento del algoritmo a un sistema en particular: Máxima Distancia Aceptable (MAD, por sus siglas en inglés), Peso de Áreas de Baja Complejidad (LCAW, por sus siglas en inglés) y Penalización por Abrir un *Gap* (P_{op}) y Penalización por Extender un *Gap* (P_{ex}). La función de similitud $\mathcal{S}()$ emplea MAD y LCAW, mientras que los otros dos parámetros son usados por el algoritmo para configurar *affine gap*.

1) *MAD*. Para definir el valor de MAD para cada uno de los atributos del snapshot, necesitamos determinar la máxima distancia entre valores que permite seguir considerándolos suficientemente similares. Normalmente, el valor de MAD suele ser proporcional, de acuerdo a nuestros experimentos entre el doble o el triple, a la precisión del instrumento de medida o a la tolerancia de la propiedad física representada por el atributo.

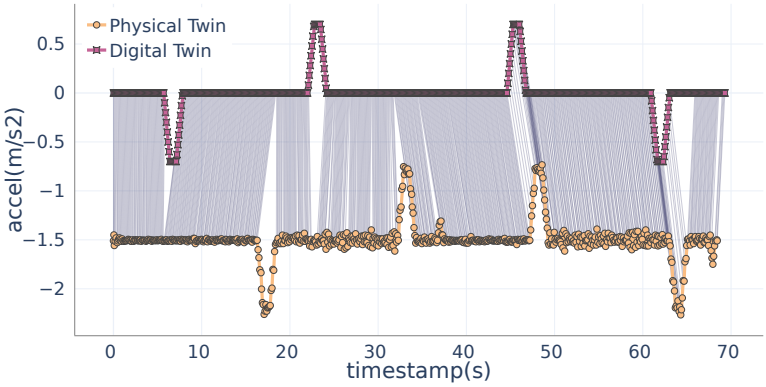
En el caso de estudio del ascensor, nos centramos en su aceleración. El MAD en este caso es 0.15 m/s^2 que coincide con el tripe del valor de la precisión del acelerómetro usado para tomar las medidas (0.05 m/s^2). Un *MAD* mayor tiende a producir alineamientos con pocos snapshots alineados. Por el contrario, valores de *MAD* muy bajos dan lugar a alineamientos inconsistentes donde se emparejan valores incompatibles.

2) Peso de áreas de baja complejidad (*LCAW*). La *teoría de la información* (Shannon 1948) cuantifica la información a través de la *entropía*, que mide la cantidad de incertidumbre en los resultados de una variable aleatoria. La entropía se maximiza cuando todos los valores posibles tienen la misma probabilidad, y disminuye a medida que la probabilidad de un resultado prevalece sobre los demás. En el estudio de caso del ascensor, las regiones con menor entropía corresponden a períodos de velocidad constante, donde la aceleración es cercana a cero. Estas regiones contienen menos información y contribuyen menos a caracterizar el comportamiento del ascensor en comparación con las curvas de aceleración.

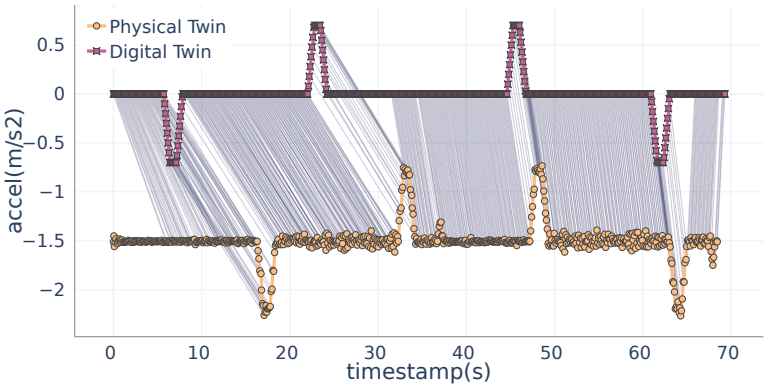
Los algoritmos de alineamiento como BLAST buscan maximizar la similitud entre pares alineados. Sin embargo, las regiones comunes de baja entropía, como las áreas de velocidad constante, devuelven puntuaciones de similitud altas, desviando la atención de subsecuencias más características. Para contrarrestar este efecto, enmascaramos estos momentos de baja entropía reduciendo su recompensa, redirigiendo así el algoritmo de alineamiento hacia regiones más significativas. En el caso del ascensor, las áreas de velocidad constante se identifican por valores de aceleración inferiores a la precisión del acelerómetro (0.05 m/s^2), haciéndolas indistinguibles de cero.

Para enmascararlas, introducimos el *Peso de Áreas de Baja Complejidad* (LCAW), que reduce la influencia de las áreas de baja complejidad. El LCAW es el producto de dos factores: la proporción de momentos relevantes para el comportamiento de interés (r) y el peso (s) asignado a los momentos no relevantes. En el estudio de caso del ascensor, encontramos que el ascensor cambiaba de velocidad solo el 10 % del tiempo, por lo que $r = 0{,}1$. Además, establecimos $s = 1/20$, lo que significa que el peso de los momentos no relevantes era 1/20 del peso de los relevantes. Así, el LCAW se fijó en 0,005.

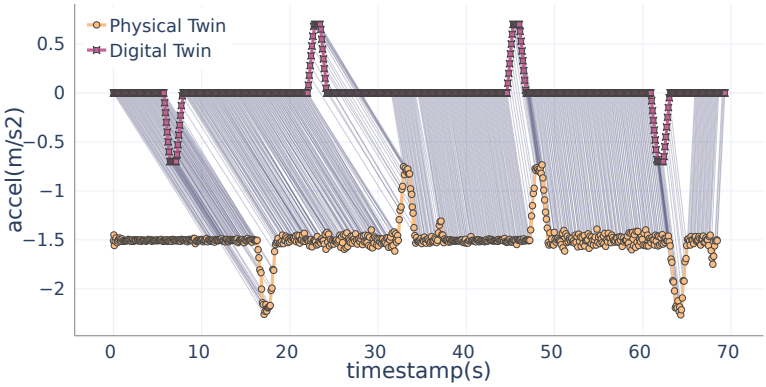
Para ilustrar el efecto de considerar áreas de baja complejidad, alineamos las trazas en la Figura A.3 utilizando un *MAD* de 0.15 sin tener en cuenta las áreas de baja complejidad. Como se muestra en la Subfigura A.4a, se observa que, aunque los cuatro picos de aceleración representan el mismo comportamiento, solo las dos últimas curvas están alineadas. Esto ocurre porque el algoritmo prioriza las áreas de aceleración cero sobre las curvas de aceleración. Sin embargo, al aplicar el *LCAW*, como se muestra en la Subfigura A.4b, el algoritmo se enfoca en el alineamiento en las curvas de aceleración,



(a) Base (No LCAW, No Affine Gap)



(b) LCAW (No Affine, LCAW)



(c) Affine Gap + LCAW

Figura A.4: Alignments for Scenario (4-0-4). Execution 04.
MAD = 0.15 m/s^2

obteniendo resultados más relevantes al reducir el peso de los alineamientos en las áreas de baja complejidad con *LCAW* = 0,005.

3) Penalizadores de *affine gap* (P_{op}, P_{ex}). El esquema de puntuación para seleccionar el alineamiento óptimo incluye cuatro parámetros clave: ***Match*/Alineamiento**, ***Mismatch***, **Penalización por Apertura de *Gap*** (P_{op}) y **Penalización por Extensión de *Gap*** (P_{ex}).

A diferencia de los algoritmos tradicionales como NDW o BLAST, donde se asignan puntuaciones constantes, la recompensa por un *Match* varía según la similitud de los snapshots, con una puntuación entre 0 y 1. Un *Mismatch* recibe una puntuación de 0 si la diferencia excede el *MAD*.

Tanto la *Penalización por Apertura de Gap* (P_{op}) como la *Penalización por Extensión de Gap* (P_{ex}) son valores negativos y ajustables. En nuestro análisis, encontramos que el algoritmo funciona mejor cuando el costo de apertura de hueco P_{op} se establece entre [-0.5,-0.1], es decir, entre el 10 % y el 50 % de la recompensa por un *Match* perfecto. Los cambios dentro de este rango no afectan significativamente los resultados.

Para ver el efecto de aplicar *affine gap* a una alineamiento, podemos comparar las Subfiguras A.4b y A.4c. Observamos que los huecos en los primeros 20 instantes están agrupados en lugar de alternarse, lo que facilita la interpretación del retraso inicial entre las trazas.

Combinar *LCAW* y *affine gap* ofrece dos ventajas clave:

a) alineamiento más preciso de las regiones relevantes para el comportamiento

b) agrupaciones de huecos más significativas en su contexto, facilitando una mejor interpretación de los resultados.

A.8 Métricas de Fidelidad

Después de alinear las trazas, para evaluar la precisión con que las trazas del DT replican el comportamiento del PT, definimos tres métricas clave: el porcentaje de *snapshots alineados* (%MS, por sus siglas en inglés), la distancia de Frèchet (FD, por sus siglas en inglés) y la distancia euclídea (ED, por sus siglas en inglés).

1) Porcentaje de snapshots coincidentes *snapshots alineados*. Esta métrica cuantifica la proporción de snapshots en las trazas de PT y DT que se alinean correctamente. Se define como:

$$\%MS_A^+ = \#X^{A+} / \max(\#X, \#Y) * 100$$

Un mayor %MS indica un alineamiento más cercano entre las dos trayectorias, mientras que un alineamiento deficiente hace que las métricas de distancia pierdan significado.

2) Medidas de distancia. Para evaluar la distancia entre las dos trayectorias, utilizamos dos métricas de distancia:

- **Distancia de Frèchet (FD)** (Efrat et al. 2002). Captura la distancia máxima entre todos los snapshots alineados, enfocándose en las discrepancias más grandes.
- **Distancia euclídea (ED).** Proporciona la distancia media entre los snapshots alineados.

Si las trazas de PT y DT son idénticas, todos los snapshots están alineados ($\%MS{=}1$) y ambas métricas de distancia son 0, lo que significa que ambos gemelos pasaron por los mismos estados.

A.9 Indicadores de Fidelidad

El grado de fidelidad de un DT respecto a su PT se puede determinar basándose en los valores de las tres métricas introducidas en el apartado anterior. Sin embargo, antes de determinar el nivel de fidelidad, es fundamental asegurarse de que el alineamiento sea suficientemente bueno para que estas mediciones sean significativas.

Guía para Interpretar la fidelidad.

- **Bajo nivel de alineamiento** (%MS <90 % ($\pm 2\,\%$)). Si el %MS es inferior al 90 %, las trazas no se han alineado correctamente, lo que hace que las métricas de distancia sean poco fiables. En estos casos, no se puede esperar un comportamiento fiel entre el DT y el PT.
- **Nivel de alineamiento moderado** (90 % $\leq$ %MS $\leq$ 95 % ($\pm 2\,\%$)). En este rango, el alineamiento se considera bajo pero suficiente para evaluar las métricas de distancia. Dependiendo de la aplicación, la FD o la ED pueden ser más relevantes para evaluar la fidelidad del DT. Por ejemplo, la FD sería clave para capturar desviaciones máximas, mientras que la ED es más útil cuando interesa el promedio a lo largo del tiempo.
- **Nivel de alineamiento alto** (%MS >95 % ($\pm 2\,\%$)). Una alineamiento superior al 95 % se considera generalmente alto, y la distancia entre las trazas determinará la fidelidad. En estos casos, se deben examinar detenidamente la FD y la ED para evaluar la capacidad del DT de replicar fielmente al PT.

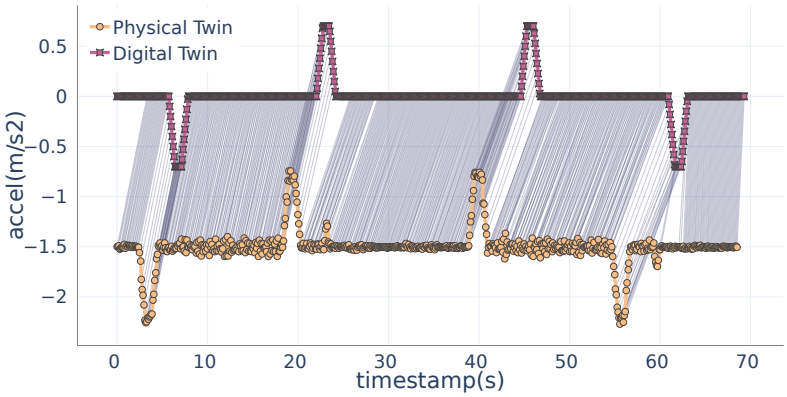


Figura A.5: Alineamiento de la Ejecución 01. *MAD* = 0.12 m/s^2.

Es importante destacar que estos umbrales dependen de la aplicación específica y deben ser definidos por el usuario final. En nuestros ejemplos, hemos utilizado los umbrales comúnmente usados en entornos de ingeniería general (Snow 2003). Cabe señalar que asumimos un error máximo del 2 % (Snow 2003) para evaluar el porcentaje de MS, ya que los umbrales a menudo no son completamente precisos.

Ejemplo: Ascensor - Ejecución 01. Figura A.5. En el ejemplo del ascensor, la Ejecución 01 obtiene unas métricas: %MS del 94 %, FD de 0,12 y ED de 0,05:

- **Nivel de alineamiento:** El %MS del 94 % está dentro del rango aceptable, lo que hace que las métricas de distancia sean significativas.
- **Distancia euclídea:** Con un valor de 0,05, esta cifra está por debajo de la precisión del instrumento de medición (0,05 m/s^2), lo que significa que la desviación promedio entre los momentos alineados es insignificante y aceptable.
- **Distancia Fréchet:** La FD de 0,12 m/s^2, que representa la desviación máxima entre las dos trazas, debe evaluarse según la tolerancia de la aplicación a las desviaciones. Para algunas aplicaciones, esto puede ser aceptable, mientras que para otras puede ser demasiado, depende de la importancia de las desviaciones máximas en el contexto de la solución.

A.10 Evaluación

La evaluación de la propuesta se divide en tres partes. Primero, aplicamos el algoritmo a tres sistemas ciberfísicos adicionales con el objetivo de probar su eficacia, complementando el ejemplo anterior. Luego, comparamos nuestra propuesta con dos soluciones de la literatura, que también se basan en el alineamiento de trazas, destacando sus ventajas

e inconvenientes en relación con la nuestra. Finalmente, evaluamos el rendimiento del algoritmo calculando el tiempo de ejecución en función del número de snapshots de la traza.

A.10.1 Aplicación del Algoritmo en Otros Sistemas

Los tres sistemas analizados se diferencian principalmente por sus snapshots, que pueden incluir uno o varios atributos, ya sea numéricos o cadenas de caracteres, y también por el contexto de análisis del sistema. En algunos casos, el gemelo digital actúa como el oráculo, mientras que en otros es el elemento que debe ser validado.

A.10.1.1 Incubadora - Comparar Dos Modelos del Mismo Sistema

El primer sistema es un gemelo digital de una incubadora, desarrollado en la Universidad de Aarhus. El objetivo de la incubadora es mantener la temperatura entre 22 y 30 grados. Para replicar el sistema físico se crearon dos modelos con diferente número de parámetros de parámetros: uno con 2 parámetros y otro con 4, siendo este último más costoso computacionalmente. La tarea era evaluar si ambos modelos son lo suficientemente fieles al comportamiento real de la incubadora, y si por tanto eran adecuados como gemelo digital.

Para evaluar ambos modelos, se establecieron dos escenarios, con distintos periodos de enfriamiento y calentamiento. En el primer escenario, con periodos de calentamiento y enfriamiento cortos, ninguno de los modelos alcanzó más del 90 % de snapshots alineados, lo que indica que ninguno es suficientemente fiel en este contexto. En el segundo escenario, con periodos de calentamiento y enfriamiento más largos, el modelo de 2 parámetros solo alcanzó el 70 % de snapshots alineados, mientras que el de 4 parámetros logró un 97 %. Esto nos lleva a concluir que el modelo de 4 parámetros suficientemente fiel en escenarios con patrones de calentamiento más lentos.

A.10.1.2 Brazo Robótico - *Snapshots* con más de un Atributo Numérico

El siguiente ejemplo es el caso de un brazo robótico. En este caso, tenemos un conjunto de modelos de ingeniería y queremos verificar si nuestro Arduino Braccio puede cumplir con los requisitos de movimiento, para decidir si es necesario comprar un robot más costoso. Se evaluaron dos escenarios. En el primero, el robot desciende, recoge un objeto, se mueve y luego lo suelta. En este escenario el robot tiene dificultades para mantener la velocidad descrita por el modelo. La mayoría de las discrepancias y desajustes se producen en las transiciones más rápidas, resutando en solo un 78 % de snapshots alineados, lo que no es suficiente para considerar que el robot sea lo suficientemente fiel al modelo.

En el segundo escenario, los movimientos son mucho más lentos, con menos transiciones que involucran múltiples servos. En este caso, el brazo robótico puede replicar los movimientos, alcanzando más del 90 % de coincidencias en los snapshots. Solo se observan algunos desajustes debido a pequeñas desincronizaciones. Con esto, concluimos que el brazo solo es útil para escenarios en los que los movimientos sean lentos y no involucren varios servos.

A.10.1.3 Lego Car Mindstorms - *Snapshots* con Atributos de Varios Tipos

Finalmente, presentamos cómo este enfoque puede medir la fidelidad de trazas con snapshots más complejos. En este caso, se trata de trazas sintéticas para un coche Lego Mindstorms, que incluye diversos sensores para detectar colisiones y opera con una máquina de estados interna que determina la acción que está ejecutando. Por lo tanto, trabajamos con valores booleanos (como *isMoving* o *bump*) y enumerados (como *action* que representa sus diferentes estados.)

Lo que hace interesante este ejemplo es que los alineamientos se vuelven mucho más estrictos: cualquier diferencia en uno de los atributos provoca un desajuste. Esto nos ayuda a identificar discrepancias, como que el gemelo digital no detecte colisiones o no reproduzca la misma secuencia de estados.

A.10.2 Comparación con Otras Propuestas de la Literatura

Para evaluar nuestra propuesta, la comparamos con dos propuestas de la literatura, en el ámbito del alineamiento de trazas: *Dynamic Time Warping* (DTW) (Sakoe y Chiba 1978) y la propuesta de Lugaresi et al. (2023).

A.10.2.1 Dynamic Time Warping

Dynamic Time Warping (DTW) (Sakoe y Chiba 1978) es un método utilizado para medir la similitud entre dos series temporales que pueden variar en velocidad o tiempo. Este algoritmo alinea las secuencias minimizando la distancia entre ellas, aunque una de las series esté extendida o comprimida en el tiempo en comparación con la otra. Para lograr esto, el algoritmo puede alinear un snapshot con más de uno en la otra traza. Sin embargo, DTW no es capaz de lidiar con posibles huecos, ya que alinea cada snapshot con el más cercano de la otra traza sin considerar umbrales ni apuntar a posibles secuencias anómalas. El resultado de DTW es una distancia que representa el nivel de similitud entre las trazas, pero no incluye un reporte con áreas anormalmente alejadas qu podrían ayudar en la detección de errores.

Otro problema de este algoritmo está relacionado con su capacidad de alinear un *snapshot* con más de uno de la otra traza, haciendo que a veces sobreextienda y sobrecomprima el alineamiento. El algoritmo puede alinear un gran número de snapshots de una traza con un solo snapshot de la otra si eso ayuda a reducir la distancia total y optimizar el resultado. En el caso del ascensor, esto genera un resultado de similitud que no es realista, ya que en realidad no está alineando snapshots similares, sino que está reutilizando excesivamente ciertos snapshots para obtener una mejor puntuación. De hecho, encontramos que en este alineamiento, DTW alineó un solo snapshot con el 28 % de los snapshots de la otra traza. Por lo tanto, en casos como este, DTW no es útil para diagnosticar discrepancias.

A.10.2.2 Validación en Tiempo Real de Lugaresi et al.

El enfoque propuesto por Lugaresi et al. introduce tres métricas para validar los DTs en tiempo real, utilizando algoritmos de alineamiento de trazas basados en programación dinámica, y se evalúa en dos niveles: eventos y KPIs.

Validación a nivel de eventos. Utilizan el algoritmo de la Subsecuencia Común Más Larga (LCSS, por sus siglas en inglés) para alinear secuencias de eventos, con una ventana de tiempo δ que define la ventana de tiempo en que dos eventos se consideran equivalentes. La métrica propuesta divide la longitud de la LCSS entre la traza más larga, proporcionando un porcentaje de eventos alineados correctamente. Para comparar nuestra propuesta, adaptamos uno de los escenarios del ascensor. Al ajustar δ, obtuvimos un alineamiento completo con un retraso de entre 7 y 7.8 segundos. Sin embargo, el método no provee el alineamiento concreto, sólo la métrica, lo que nos impide estudiar las causas exactas del retraso y sus subsecuencias específicas.

Validación a nivel de KPIs. Otra de las métricas de la propuesta se basa en el análisis de KPIs del sistema y su alineamiento usando el algoritmo de la Subsecuencia Común Más Larga (LCSS, por sus siglas en inglés). Para comparar este enfoque, adaptamos otro escenario del ascensor, incluyendo el cálculo del "tiempo promedio entre pisos". El algoritmo cuenta con un parámetro ϵ que define la diferencia máxima admitida entre valores de KPI para considerarse similares. Aunque esta métrica puede mostrar correspondencias, al igual que el enfoque anterior, no identifica discrepancias. Lo mismo que ocurría en el vaso del alineamiento de eventos. La segunda métrica emplea DTW para alinear los valores de KPI. Sin embargo, sufre de los mismos problemas que al algoritmo original, generando distorsiones en el alineamiento. Además, la normalización de los valores hace que el resultado no sea comparable entre escenarios con rangos de valores distintos, lo que puede producir resultados poco fiables y conclusiones erróneas.

A.10.3 Evaluación del Tiempo de Ejecución y Memoria. Análisis de Escalabilidad

En esta evaluación, comparamos el rendimiento de los algoritmos introducidos en el apartado anterior, con las variantes de nuestro algoritmo. Los resultados muestran que todos tienen una complejidad cuadrática con respecto a tiempo y memoria, acorde con la naturaleza de los algoritmos de programación dinámica. Entre ellos, el más costoso en términos de tiempo de ejecución es la versión de DTW implementada por Lugaresi, debido a una sobrecarga adicional en sus cálculos. La implementación de LCSS para eventos también presenta un alto tiempo de ejecución, principalmente por la complejidad de comparar cadenas en lugar de números. Su variante para KPIs resulta más eficiente. Sin embargo, estos tres algoritmos son los que presentan un menor consumo de memoria en comparación con los demás, debido a que utilizan una única matriz para calcular el alineamiento. Además, su enfoque no incluye el proceso de backtracking para devolver el alineamiento óptimo.

Las variantes de nuestro algoritmo, especialmente aquellas que no consideran el uso de *affine gap*, son las más rápidas, ya que emplean una única matriz para los cálculos de alineamiento. La inclusión de *affine gap* reduce el rendimiento en un 30,7 %, mientras que el mecanismo LCAW lo afecta de manera marginal. El algoritmo DTW original es el más rápido, superando incluso a la variante básica de nuestro algoritmo, aunque genera alineamientos de menor calidad, como se ha mostrado en secciones anteriores.

En cuanto a memoria, las variantes de nuestro algoritmo que incluyen *affine gap* tienen un mayor consumo en comparación con aquellas que no lo incluyen y con DTW. Esto se debe a la necesidad de utilizar tres matrices para decidir la inclusión de un hueco en cada una de las trazas.

Por lo tanto, al seleccionar una variante del algoritmo, es importante considerar que *affine gap* incrementa tanto el tiempo de cómputo como el consumo de memoria, pero produce resultados más precisos. Si la eficiencia es prioritaria, podría ser recomendable prescindir de *affine gap*.

B

CONCLUSIONES

B.1 Resumen y Trabajos Futuros

Esta tesis se centra en la validación de Gemelos Digitales. Primero, se introduce una arquitectura conceptual y un marco para definir con precisión los términos clave de este paradigma. Con una comprensión clara del concepto de DT y sus elementos, se define el término "fidelidad" y se propone un método y un conjunto de métricas para evaluarla. El método compara el comportamiento de los gemelos digitales, empleando las secuencias de estados que alcanzan durante su ejecución.

Este enfoque es aplicable no sólo en el contexto de los gemelos digitales, sino que también sirve para evaluar la similitud del comportamiento de cualquier par de sistemas. Por ejemplo, puede usarse para comparar dos sistemas físicos (por ejemplo, identificar desviaciones entre dos instancias de la misma máquina), o dos sistemas digitales (por ejemplo, comparar dos simulaciones con diferentes niveles de detalle), o entre un sistema físico y uno digital, como en el caso de los DTS.

Esta tesis contribuye al campo de los Gemelos Digitales, un área que ha ganado atención recientemente y enfrenta desafíos emergentes. De hecho, nuestro método para determinar la fidelidad abre la posibilidad de continuar diversas líneas de investigación. Primero, nuestro algoritmo podría extenderse para el diagnóstico automático de discrepancias. Si bien el método actual identifica desviaciones entre trazas, podría automatizarse para identificar las causas exactas del comportamiento anómalo sin necesidad de análisis humano.

También planeamos incorporar incertidumbre en las comparaciones, aprovechando nuestros resultados sobre la representación de incertidumbre en atributos usando variables aleatorias, en lugar de los tipos de datos tradicionales (Bertoa et al. 2020; Fernández-Candel et al. 2024; Jézéquel y Vallecillo 2023). Finalmente, otra posible mejora podría ser la validación de trazas en tiempo de ejecución utilizando *ventanas corredizas* (Lugaresi et al. 2019, 2023), lo que permitiría una validación dinámica a lo largo del ciclo de vida de un DTS.

B.2 Conclusiones

En conclusión, nuestra propuesta se enfoca en resolver un problema relevante para la industria y la academia ya que se necesitan herramientas que permitan la verificación y validación de los DTs (Muctadir et al. 2023). Nuestro enfoque aborda específicamente el desafío de evaluar la similitud del comportamiento entre el DT y su contraparte física, lo que denominamos *fidelidad*. Garantizar un nivel de fidelidad adecuado es crucial para la efectividad de los DTSs, que requieren la réplica del comportamiento. Nuestro método ofrece una solución que apoya el análisis de estos aspectos y facilita la interpretación de las causas subyacentes de las discrepancias.