Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments

Paula Muñoz, Manuel Wimmer, Javier Troya and Antonio Vallecillo

Abstract—Digital twins are gaining relevance in many domains to improve the operation and maintenance of complex systems. Despite their importance, most efforts are currently focused on their design, development, and deployment but do not fully address their validation. In this paper, we are interested in assessing the fidelity of physical and digital twins and, more specifically, whether they exhibit twinned behaviors. This will allow engineers to check the suitability of the digital twin for its intended purpose. Our approach assesses their fidelity by comparing the behavioral traces of the two twins. Our contribution is threefold. First, we define a measure of equivalence between individual snapshots capable of deciding whether two snapshots are sufficiently similar. Second, we use a trace alignment algorithm to align the corresponding equivalent states reached by the two twins. Finally, we measure the fidelity of the behavior of the two twins using the level of alignment achieved in terms of the percentage of matched snapshots and the distance between the aligned traces. Our proposal has been validated with the digital twins of four cyber-physical systems: an elevator, an incubator, a robotic arm, and a programmable robotic car. We were able to determine which systems were sufficiently faithful and which parts of their behavior failed to emulate their counterparts. Finally, we compared our proposal with similar approaches from the literature, highlighting their respective strengths and weaknesses related to our own.

Index Terms—digital twins, fidelity, alignment, traces.

1 Introduction

Digital Twins have emerged as a promising paradigm to enhance the design, operation, and maintenance of complex systems. A *Digital Twin (DT)* is defined as a virtual representation of a real-world entity or process (the so-called *Physical Twin*, PT) synchronized at a specified frequency and fidelity [1]. Depending on the purpose, the replica may represent different properties of the PT, such as physical constraints, appearance, or behavior [2]. The twinned systems (the DT and PT), the connections between them, and the set of system services comprise the so-called *Digital Twin System* (DTS) [3].

A DT may be created before the PT to explore potential product characteristics at higher levels of abstraction with lower costs. They can also be developed after the PT to add advanced functionality to an existing system and enhance its operation. Alternatively, DTs can be developed jointly with the PT, leveraging the more cost-effective design-space exploration with the final goal of enhancing its operation during deployment. During operation, DTSs are commonly used for behavior optimization, monitoring, system validation, and prediction [2].

The essential requirement to employ a DT in any of the previous contexts is that it needs to represent the required properties of the PT accurately. If the DT is not faithful enough, any predictions or conclusions derived from it will be unreliable. In this context, the engineering of DTSs becomes critical [4]. Currently, most works focus on the design, development, and deployment of DTSs, but just a few on validating them [2]. Some of these works apply simulation and testing during the design phase or consistency monitoring during runtime to find discrepancies between the DT's predicted behavior and the PT's actual behavior. [2].

In this work, we propose an offline method to validate the twins' behavior. Here, validation is defined as "the process of determining the degree to which a computational model is an accurate representation of the real world from the perspective of the intended uses of the model" [5]. Usually, model validation consists of comparing the output behavior of the system with the one of the simulation model [6]. The results of the comparison determine the level of accuracy of the digital model [7]. More precisely, we measure the *degree of fidelity* of a DT with respect to its PT. Fidelity is defined as "the degree to which a model reproduces the actual state and behavior of a system in a measurable way" [8].

Our approach assesses behavior fidelity by comparing the two twins' *behavioral traces*. In our context, a trace is a sequence of *snapshots*. Each snapshot represents the state of the system at a given moment in time. The use of traces makes our proposal independent of the DTSs implementation method. We only compare the behavioral outputs of the PT and DT, regardless of their origin.

Trace analysis has already been proposed for validation of complex systems, see, e.g., [7], [9], [10], [11]. However, when we tried to apply these approaches in the context of DTSs, we realized that the application of these proposals presents a number of challenges, as explained next. To illustrate the challenges, let us take an example from [12], in which a DTS is proposed to prevent crane collisions in a limited environment. By considering the PT's state and environment, the DT simulates its movement and warns about

• *P. Muñoz, J. Troya, and A. Vallecillo are with the ITIS Software at Universidad de Málaga, 29010 Málaga, Spain. E-mail: {paulam, jtroya, av}@uma.es*
• *M. Wimmer is with the CDL-MINT at Johannes Kepler University, 4040 Linz, Austria. Email: manuel.wimmer@jku.at*

Manuscript received April 19, 2005; revised August 26, 2015.

collisions. The authors developed higher-level abstraction models to simplify the physics of the crane and reduce computational time. However, the replica must be faithful so as not to give false assurance to the operator. The snapshots will have the x, y, and z coordinates of the crane's tip at each time step. Thus, we need to compare the trajectory of the model with the real system's trajectory to determine if it serves the intended purpose before its deployment.

1) Trace alignments. Typically, traces are compared by considering only the temporal relationships between their elements, assuming that the states must be reached at the same times. However, in some cases, we cannot assume that the two systems are running in unison, and therefore, we need to align the traces using similarity functions and not just timestamps. An *alignment* is defined as a sequence of matching pairs of snapshots from two traces. For the crane problem, we could use a simplified model that does not precisely simulate the movement duration but can still provide the sequences of positions, i.e., the trajectory, to check for collisions. In such a case, a comparison based on temporal relationships would conclude that the model is not faithful. However, it could be good enough for our requirements since we only need to validate the sequence of movements. Thus, we need a general and flexible mechanism to align the traces before calculating their distance, using different options to identify the states that should be considered similar and, therefore, matched. Likewise, we need to define a threshold capable of deciding when two states are too far apart to be considered similar.

2) Affine gaps. Trace alignments often contain gaps and mismatches to signify absent states and discrepancies that the algorithm penalizes based on constant sanctions. However, this approach may not work well when the expected alignments have long gaps, such as when trying to perform anomaly detection [13], because it produces alignments with frequent alternation between gaps, mismatches, and matches. For example, suppose we want to detect a missing step in the crane's trajectory. In that case, we would expect to find one sequence of gaps in the alignment that points to the missing behavior. *Affine gap* consists in assigning a higher penalty to opening a gap than to continuing the current one, grouping inconsistencies, and allowing a better interpretation of the results.

3) Low-complexity regions. Low-complexity regions are trace segments that contain less information about the behavior of interest but produce high-scoring matches, e.g., a system's stationary behavior. This causes alignment algorithms to concentrate on them, to the detriment of searching for alignments in more characteristic parts of the traces. These low-complexity regions must be identified and soft-masked in some way. For example, suppose we performed the alignment of a trace that contains stationary behavior because the crane does not move for long periods. In that case, we may find that the behavior of interest, i.e., the crane's movement, is not included in the alignment. This is because the algorithm will focus on aligning the highest-scoring snapshots, i.e., stationary ones. However, this alignment, as it does not include the movement, will not properly assess the fidelity of the actual trajectory of the crane.

4) Distance metrics. Fidelity measures are usually based on the distances between traces, using averages or maximums. However, we need a combined approach, as each is more appropriate depending on the situation. In addition, before measuring the distance between two traces, we need to ensure a minimum percentage of coincident snapshots. Otherwise, the calculated distances may be meaningless. These distance metrics help to measure the accuracy of the reproduced behavior. In an iterative development of the crane replica, the metrics could determine whether a model's accuracy has improved.

In this work, we aim to address these challenges, providing a suitable offline trace analysis approach for assessing the fidelity of a DT with respect to a PT, or vice-versa, in a quantifiable manner. In particular, to compare the traces, we follow a three-step process. First, we define a *comparison function* between individual snapshots capable of deciding whether two snapshots, i.e., system states, are sufficiently similar or not. The purpose of the DT will dictate the properties of interest that will be considered by this comparison function. For instance, the crane state could include not only its coordinates but also its weight and the load it carries, which can affect its movement [12]. While dynamics are not essential for larger cranes, they are crucial for small mobile cranes since they heavily affect their movement. Therefore, before determining the suitability of a replica, we need to decide which properties need to be replicated and the required accuracy for each of them. Second, we use a *trace alignment algorithm* to align the corresponding equivalent states reached by the two twins, given a certain similarity threshold (the so-called *maximum acceptable distance*, MAD) defined for each property of interest and the identification and masking of the appropriate low-complexity regions. Finally, we measure the fidelity of the two twins' behavior using the *level of alignment* achieved (in terms of the percentage of matched snapshots) and *two distances between the aligned traces* (average and max).

The work has been validated with the DTSs of four cyber-physical systems that exhibit different behaviors: an elevator, an incubator, a robotic arm, and a programmable robotic car. After the analysis, we were able to determine which of these systems were sufficiently faithful and which parts of their behaviors failed to emulate their counterparts. This provides domain experts with precise guidance on how to improve the behaviors in the given scenarios. These examples will also allow us to show three other use cases of our proposal. In the incubator example, we compare the fidelity of two DTs of the same PT [14]. The robotic arm example shows another use case of our proposal, where the objective is to check if the physical system behaves as required by the user, i.e., if the PT is faithful to the DT. Here, the DT acts as the oracle and not as the emulator. Additionally, we consider a fourth system consisting of a case study of a Lego Mindstorms NXT Car to show our approach applied to traces with Boolean and enumerated attributes.

After this introduction, Section 2 briefly describes the background of our work. Then, Section 3 describes in detail our proposal, using one case study to illustrate and motivate it. The configuration of the parameters of the alignment algorithm and their effects on the fidelity metrics are discussed in Section 4. Then, Section 5 presents how the fidelity of a DT can be assessed using our metrics. After that, Section 6 presents the results of further evaluation exercises we have conducted to assess our proposal, as well as its main advantages and limitations. Finally, Section 7 relates our work to other similar approaches and Section 8 concludes with an outline of future work.

2 Background

In this section, we will briefly discuss the terminology and central concepts that form the basis of our proposal. Firstly, in Section 2.1, we define the central concepts: fidelity, digital twin, trace, and snapshot. Section 2.2 presents the formal definitions of the alignment elements and briefly introduces the two sequence alignment algorithms that inspired our proposal. Finally, in Section 2.3, we introduce the distance measure that will be used as a fidelity metric.

2.1 Digital Twins and Fidelity

The core of our contribution revolves around the notion of fidelity between the twinned systems. Although there is no agreed definition of what a Digital Twin is, here we adopt the following one: "A *Digital Twin* is the virtual representation of a real-world entity or process, synchronized at a specified frequency and fidelity" [1]. Here, *Fidelity* is "the degree to which a model reproduces a system's actual state and behavior in a measurable way" [8].

By behavior, we mean the evolution of the system states over time, and therefore, in this work, the behavior of the twinned systems will be represented by *traces*, which we also call *trajectories*. Traces are sequences of *snapshots* captured periodically. Snapshots [15] are used to represent the system states and are defined in terms of objects whose attribute values capture the state of the relevant properties of the system at the time when the snapshot was taken.

Of course, fidelity cannot be assessed in general, but for specific scenarios (e.g., set of inputs and environment constraints) and taking into account the specific purpose of the DTS. Hence, the need to define the *validity frame* of the validation exercise [16], i.e., the experimental context of a model in which that model gives predictable results.

For simplicity, in this work, we will assume all snapshots are expressed at the same level of abstraction, i.e., they contain the same attributes, and these are of the same types. Otherwise, the user may define mappings from one abstraction level to another and implement the needed transformations. For example, if the DT provides the crane's tip coordinates, and the PT provides the servos' angles, we could derive the coordinates from the angles, enabling the comparison.

2.2 Trace Alignment

The first step for comparing the behavioral traces of the two twins is to align them. Aligning the traces requires identifying their common snapshots, the mismatches, and the possible gaps. In order to accurately introduce all concepts related to trace alignments, we have developed a set of definitions in Section 2.2.1. Then, we introduce the two sequence alignment algorithms that form the basis of the proposal (Section 2.2.2), and, finally, we explain the scoring scheme used to improve the quality of the alignments (Section 2.2.3).

2.2.1 *Trace Alignments*

Definition 2.1 (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 $A \subseteq \{0..n\} \times \{0..m\}$, which 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 indexes of the elements that have not been paired. We use the number '0' to represent a gap in the aligment, i.e.,

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

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 (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 2.3 (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):

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

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

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

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 that are matched with gaps: $G_X^A = X - X^A$, and $G_Y^A = Y - Y^A$.

It is also important to pay attention to the *sequences of consecutive gaps* (SCG) produced by the alignment. E.g., 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.

Note that other alternative alignments could have been defined, e.g.:

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

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

$S'_1 = \{(3,0),(4,0)\}$, $S'_2 = \{(6,0)\}$, of lengths 2 and 1.
$S''_1 = \{(2,0)\}$, $S''_2 = \{(4,0)\}$, $S''_3 = \{(6,0)\}$, of length 1.
$S'''_1 = \{(2,0),(3,0)\}$, $S'''_2 = \{(6,0)\}$, of lengths 2 and 1.

Thus, depending on how the alignment decides to pair the elements and determines the corresponding gaps, the length of the SCGs may vary. This is further illustrated in the following example.

Suppose that $X = \{a,a,a,b,b,b,b,b\}$ and $Y = \{a,b\}$. In this case, depending on the strategy used by the alignment algorithm to determine the gaps, we can have alignments with rather different SCGs. 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 one 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 strategy of the alignment algorithm to open a gap or continue exploring the chain. This is an important issue, which is decided by the chosen *affine gap* penalty. This is discussed later in Sect. 2.2.3.

2.2.2 Trace alignment algorithms

To align traces we investigated existing solutions for sequence alignment. This is a well-known problem in other disciplines, such as Bioinformatics, in which efficient algorithms have been defined for aligning DNA and other biological sequences.

There are two types of alignment algorithms: global and local. *Global* alignment algorithms, like Needleman-Wunsch [17] compare two similar sequences and find the highest scoring alignment that includes all elements. In contrast, *local* alignment algorithms, such as the Smith-Waterman algorithm [18], on which BLAST is based, identify a list of the most similar subsequences between two dissimilar sequences.

To develop our algorithm for behavioral trace alignment, we used the classic global alignment algorithm for biological sequences, *Needleman-Wunsch* (NDW) [17], complementing it with a set of techniques borrowed from the *BLAST* algorithm [19], which is broadly used for searching databases of biological sequences for statistically significant similarities. These algorithms are described next.

The Needleman-Wunsch Algorithm (NDW)

The *Needleman-Wunsch algorithm* [17] is a global alignment algorithm that finds the optimal alignment of two sequences of characters. It is implemented using dynamic programming techniques, i.e., it breaks down the problem of comparing sequences into smaller problems (comparing sets of subsequences) to find the optimal solution to the global problem, and 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 having the highest score—there may be no unique optimal alignment. To select the optimal one, the algorithm requires a scoring scheme, which is defined in terms of rewards and penalties assigned to the possible results when comparing two characters a and b, belonging to sequences X and Y, respectively. These outcomes are:

- **Match.** The characters are the same.
- **Mismatch.** The characters are not the same but are aligned, which is considered a mismatch.
- **InsDel** (or **Gap**). The characters are not the same and the best alignment involves one letter paired to a gap (represented by "–") in the other sequence.

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

$$\begin{aligned} H[i,j] = max\{ & H[i-1,j] + g, \\ & H[i,j-1] + g, \\ & H[i-1,j-1] + s(x_i,y_j)\} \end{aligned}$$

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.

There are different scoring schemas depending on how we would like the alignment to work. For example, we could use a reward of $+1$ for a Match; 0 for a Mismatch; and a penalty of -1 for a Gap. This scheme gives priority to mismatches over gaps, favoring solutions with fewer gaps. Another schema uses a reward of $+1$ for matches and penalties of -1 and -2 for mismatches and gaps, respectively. This schema corresponds to the edit (Levenshtein) distance between the two strings [20]. The lower the alignment score the larger the edit distance.

BLAST

Basic Local Alignment Search Tool (BLAST) [19] 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 a *local* dynamic programming alignment algorithm for biological sequences, the *Smith-Waterman algorithm* [18].

To avoid traversing the entire search space, BLAST uses 3 heuristic phases to refine potential high-scoring pairings.

1) Seeding. The user establishes a specific word length, W. Then, the algorithm looks for alignments of that length whose score is as large as a threshold T. These matches are the seeds for possible alignments. During the seeding phase, the algorithm uses *low-complexity regions*, which are subsequences of little biological interest as they are common in many other sequences, but which produce high scores. To avoid seeding in such regions, they are soft-masked, including them in the alignment in the extension phase but not in the seeding phase.

2) Extension. Once the search space is seeded, the algorithm generates new alignments by extending the individual seeds, increasing the word length progressively. To decide when to stop extending, a threshold T decides how much the alignment score can drop from the last maximum.

3) Evaluation. Once all the seeds are extended, the alignments are evaluated to decide which ones are statistically significant. To determine this significance, the algorithm establishes a score threshold S, which allows sorting alignments into low- and high-scoring.

2.2.3 Affine Gap

The result of an alignment algorithm may include gaps. In the NDW algorithm, the penalty for a gap is set as a constant value for all the alignments. This produces alignments with sequences in which gaps and matches alternate. However, this approach may not work well when the expected alignments have long gaps instead of many small sequences of alternating gaps and matches. For instance, alignments with longer gaps are more meaningful when trying to detect anomaly sequences [13]. To better model this phenomenon, alignment algorithms such as BLAST typically apply an evaluation gap penalty system named *affine gap*, which imposes a higher penalty for opening a new gap than for extending an existing one. If P_{op} is the penalty for opening a gap, and P_{ex} is the penalty for extending one, the total penalty for opening a gap of length L would be $P_{op} + P_{ex} * (L - 1)$.

These penalties are incorporated into the scores of the dynamic programming algorithm so that the optimal alignment is built considering them. The values for the penalties of opening and extending a gap for BLAST are usually obtained empirically. In general, the algorithm works well when the penalty for opening a gap (P_{op}) is between 10 and 15 times the penalty for continuing it (P_{ex}), cf. [19].

2.3 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 [21].

1) 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:

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

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

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

The Fréchet distance $F(X, Y)$ between two already aligned 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 $Y(\psi(t))$. It is generally explained through this example: Imagine a person who walks from one end of a trajectory X to the other end, being their position $X(\chi(t))$; likewise, a dog walks from one end of its trajectory Y to the other end, being its position $Y(\psi(t))$, with the person holding the dog by a leash. The Fréchet distance between both is the minimum leash length needed to walk the dog following their trajectories. Formally, assuming that $d(X(\chi(t)), Y(\psi(t)))$ is a distance measure between individual positions, the Fréchet distance is defined as follows:

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

Note that this measure computes the distance between the two more distant snapshots of the trace. 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 and are less sensitive to outliers, making them appropriate for applications where what really matters is the average distance between the traces.

3 Proposal

Our approach assesses behavior fidelity by comparing the two twins' *behavioral traces*. To compare the traces, we follow a three-step process. First, we represent the system states as snapshots (Section 3.2) and then define a *comparison function* between individual snapshots capable of deciding whether two snapshots are sufficiently similar or not (Section 3.3). Second, we use a *trace alignment algorithm* (Section 3.4) to align the corresponding equivalent states reached by the two twins, given a certain similarity threshold (the so-called *maximum acceptable distance*, *MAD*) defined for each property of interest. Our aligning algorithm adapts the NDW algorithm to work with behavioral traces and then applies two optimization techniques borrowed from BLAST: low-complexity regions and affine gaps. We analyze the impact of these optimizations on the alignments initially produced by the standard NDW algorithm, which did not align relevant parts of the trace (cf. Section 3.4). Finally, we measure the fidelity of the behavior of the two twins using the *level of alignment* achieved (in terms of the percentage of matched snapshots) and the *distance between the aligned traces* (Section 3.5). The proposal is illustrated in this section with the case study of the Mondragon elevator.

To apply our proposal, the problem should fulfill the following requirements: a) the traces are expected to exhibit the same synchronized behavior if the starting times were the same; b) both systems should provide snapshots with the same sample period—we work on the premise that two identical behaviors coincide in 100% of the snapshots; for this to be true, the snapshots must be taken at the same time intervals; c) snapshots must include the behavior of interest for the validation—the similarity between the twin states can be assessed based on the difference between the attribute values of the corresponding snapshots. Under these circumstances, our approach provides a set of metrics that determine the degree of fidelity between the systems, indicating the specific parts of the traces that contain delays, inconsistencies, or differences in behavior. The approach works offline, only pointing to delays or discrepancies in the traces to evaluate if the twins would traverse the same states simultaneously.

3.1 Running example: An elevator

To illustrate the problem and demonstrate our solution we will use the case study of an elevator at the University of Mondragon. The elevator travels between five floors: 0 to 4. It is normally used by students who cannot use the stairs or for transporting heavy loads. Recognizing its frequent usage and its 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 Digital Twin System (DTS) using the commercial simulator *Elevate* [22] as the digital replica. 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 behavior of the simulator is faithful to that of the physical system.

The objective of our proposal, in this case, is 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 acceleration values will be compared with those of the real system for specific operating sequences, which are measured using an accelerometer while performing the same sequences.

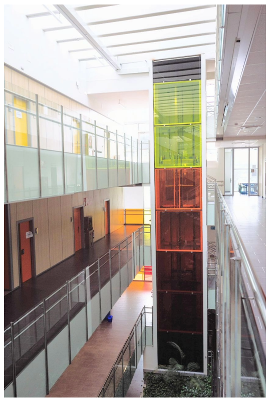
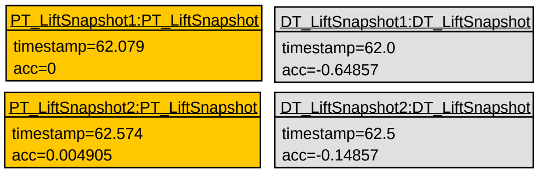


Fig. 1: The elevator.

3.2 System representation

To apply the sequence alignment algorithm to behavioral information, we need a discrete representation of the system's behavior over time. To discretize such information, observations of the system state are captured periodically, often referred to as sampling, as if a film was divided in its photograms. This representation is adapted to the required level of abstraction, only including in the snapshots the information about the properties of interest. For example, Figure 2 shows two snapshots of each twin. In this example, snapshots only include the elevator's acceleration (the property of interest), and the timestamp of the moment the snapshot was taken.

Figure 3 shows the sequences of snapshot values, i.e., the acceleration data, obtained from the real system (the PT, top) and from the Elevate simulator (the DT, bottom) for a particular scenario, which we have called (4-0-4): The elevator starts at floor 4, goes down to floor 0 (from timestamp 5.8 to 24.1 in the DT), stops, and goes up again to floor 4 (from timestamp 44.7 to 63 in the DT). We see four peaks divided into two groups. The first group, with


PT_LiftSnapshot1:PT_LiftSnapshot
timestamp=62.079
acc=0

DT_LiftSnapshot1:DT_LiftSnapshot
timestamp=62.0
acc=-0.64857

PT_LiftSnapshot2:PT_LiftSnapshot
timestamp=62.574
acc=0.004905

DT_LiftSnapshot2:DT_LiftSnapshot
timestamp=62.5
acc=-0.14857

Fig. 2: Sample elevator snapshots from the PT and the DT.

the first two peaks, represents the acceleration during the descent of the elevator from floor 4 to 0. The first peak shows the negative acceleration during descent, while the second represents the positive acceleration during braking upon reaching the floor. Similarly, the next two peaks represent the acceleration during ascent.

These acceleration and deceleration peaks for floor transitions are remarkably similar between the simulation and the real system, disregarding the accelerometer noise near zero when the elevator speed is constant. However, there is one aspect of the real system that is absent in the simulation: Every time the elevator brakes in either direction, an additional and much smaller acceleration is added to smooth out the elevator's stop and improve the user experience. This pattern is specific to this particular elevator model and can be seen as a small peak following the acceleration change due to the braking in the same direction.

The graphics also show a small delay between the DT and the PT. This is because the PT was activated by hand, while the simulation was regularly run. Hence, there is a need to align the traces using similarity functions and not just timestamps since, on many occasions, we cannot assume that the two systems are running in unison, as is the case here.

Finally, note that the accuracy of the discrete representation of the trace depends on the frequency with which snapshots are taken. A shorter snapshot period results in a more accurate trace. However, sampling at a faster speed may produce too many identical snapshots, which provides no new information, increases storage requirements, and slows down analysis algorithms. In this case, we took 696 snapshots, approximately 11 per second.

We conducted ten executions of this scenario to analyze how variations could impact the alignments and account for uncertainties and noise in the behavior. Specifically, we will reference execution 04 and execution 01 in this paper to illustrate some of the algorithm's properties. The analysis of all scenarios and the ten executions for scenario (4-0-4) are described in detail in the technical report of this case study [23].

3.3 Comparison Function

The first step in aligning the traces is the definition of a comparison function capable of comparing two snapshots of the system. Comparison functions in Bioinformatics and other disciplines that align sequences are simple because the sequences they compare are mainly characters. However, snapshots can be complex structures composed of heterogeneous records because they are intended to represent the system states.

Algorithm 1 describes the similarity function "$\mathcal{S}()$" between two snapshots s_A and s_B. We suppose that the two snapshots are composed of the same k typed attributes

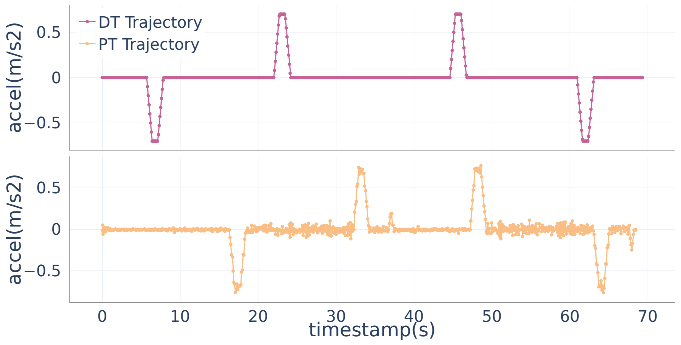


Fig. 3: Elevator traces of the PT and DT from floor 4 to 0, and to 4 again. Execution 04.

$\{a_1, ..., a_k\}$. The function returns a real number in the range [0..1] that indicates the similarity score between the two snapshots, where 1 means they are equal and 0 means that they are different. The difference between each attribute is computed depending 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". If the difference exceeds the *MAD* value, we consider them different and assign a reward of 0. Otherwise, the reward gradually increases as the difference between values decreases, as shown in line 6 of the algorithm. The reward is 1 if they are equal. In the case of Boolean values, a reward of 1 is given if the values are the same and 0 if they are different. For enumerated or string attributes, we could do the same or use the comparison operations defined in [24] (namely, *uEquals()*) to calculate a similarity value between 0 and 1.

Lines 24-28 implement the treatment of the low-complexity regions, as explained later in Section 3.4. We identify those trace elements that are considered to be within a low-complexity region using function *lowComp()*, and then reduce the impact of these elements on the overall score by decreasing their score using the constant *LCAW* (e.g., 0.005). If one of the two snapshots belongs to a low-complexity region, the reward is multiplied by *LCAW* (line 27); and if both belong, the reward is halved (line 25).

This algorithm evaluates the reward for each attribute and computes the average, assuming that all attributes have the same weight. Of course, alternative algorithms could be defined if necessary, either by assigning different weights to the attributes or, for example, by using the minimum of the similarities instead of their average. Furthermore, not all attributes need to be considered in the similarity function, only those relevant to the comparison depending on the purpose of the DT. To restrict the timeframe for matching snapshots, we could consider the timestamp attribute in the similarity function and specify a MAD value. However, we do not impose this restriction in any of the examples presented to enable the detection of similar behavior despite any potential delays. In the Elevator case study, we are especially interested in the acceleration value. Considering the snapshots in Fig. 2, if we compare the first PT_LiftSnaphot with the two DT snapshots using a *MAD* of 0.4 m/s^2, we obtain a similarity value of 0 and 0.63, resp.

Algorithm 1 Similarity function between two snapshots

```
1:  $i \leftarrow 1$; result $\leftarrow 0$
2:  while $i \leq k$ do
3:      if isNumerical($s_A.a_i$) then
4:          $dif \leftarrow abs(s_A.a_i - s_B.a_i)$
5:          if $dif <$ MAD then
6:              result $\leftarrow$ result $+ (1 - \frac{dif}{\text{MAD}})$
7:          else
8:              result $\leftarrow 0$        ▷ Mismatch if MAD exceeded
9:              break
10:         end if
11:     end if
12:     if isBoolean($s_A.a_i$) $\wedge$ $s_A.a_i = s_B.a_i$ then
13:         result $\leftarrow$ result $+ 1$
14:     else
15:         result $\leftarrow 0$
16:         break
17:     end if
18:     if isString($s_A.a_i$)$\wedge$equals($s_A.a_i, s_B.a_i$) $>$ MAD then
19:         result $\leftarrow$ result $+$ equals($s_A.a_i, s_B.a_i$)
20:     else
21:         result $\leftarrow 0$
22:         break
23:     end if
24:     if lowComp($s_A.a_i$) $\wedge$ lowComp($s_B.a_i$) then
25:         result $\leftarrow$ result $*$ LCAW/2
26:     else if lowComp($s_A.a_i$) $\vee$ lowComp($s_B.a_i$) then
27:         result $\leftarrow$ result $*$ LCAW
28:     end if
29:     $i \leftarrow i + 1$
30: end while
31: return result/$k$
```

3.4 Trace alignment algorithm

Our snapshot alignment algorithm adapts the NDW algorithm to work with behavioral traces, and it then applies two of the BLAST optimization techniques: (a) the use of *low-complexity regions* to avoid initiating alignments in those areas and focus on more characteristic regions of the protein sequence; and (b) the *affine gap* techniques for deciding when the algorithm should insert a gap or a mismatch preventing the excessive alternation between gaps and matches and obtaining more relevant alignments.

The use of affine gaps requires keeping track of the possibilities of opening or continuing a gap in either of the sequences X and Y for every pair of characters. This introduces the need for three additional matrices to the score matrix H of the NDW algorithm: one per snapshot sequence for opening or continuing a gap (H_X and H_Y) and one to evaluate the matches and mismatches (H_M). More precisely, if H is the maximum score matrix of the alignment of sequences X and Y, the Bellman equations for computing this matrix recursively are as follows:

$$\begin{aligned}
H[i,j] &= max\{H_M[i,j], H_X[i,j], H_Y[i,j]\} \\
H_M[i,j] &= H[i-1,j-1] + \mathcal{S}(x_i, y_j) \\
H_X[i,j] &= max\{H[i-1,j] - P_{op} - P_{ex}, H_X[i-1,j] - P_{ex}\} \\
H_Y[i,j] &= max\{H[i-1,j] - P_{op} - P_{ex}, H_Y[i-1,j] - P_{ex}\}
\end{aligned}$$

where $\mathcal{S}()$ is the similarity function described in the previous section, and the initial conditions are as follows:

$$H[i,0] = H[0,j] = 0$$
$$H_M[i,0] = H_M[0,j] = -\infty$$
$$H_X[i,0] = H_Y[0,j] = P_{op} + P_{ex}$$
$$H_X[0,j] = H_Y[i,0] = -\infty$$

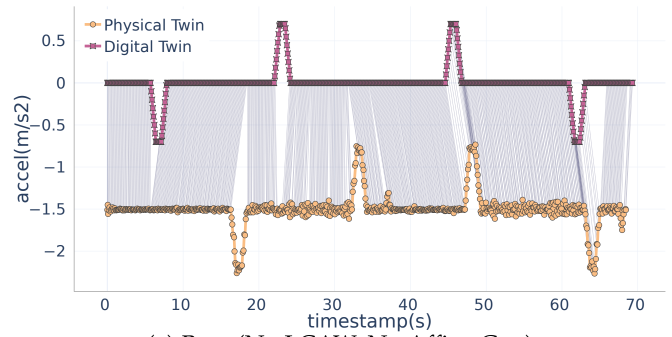
Basically, the algorithm is the same as the NDW algorithm [17], but incorporates two main changes. First, it uses three additional matrices as described above to implement the affine gap mechanism, providing it uses the similarity score function $\mathcal{S}()$ to compare the trace elements (see Algorithm 1) instead of the scoring matrices to compare characters. Second, as described in the previous section, the similarity function deals with the low-complexity regions and adapts their scores accordingly. The extension of the algorithm still guarantees *Bellman's Principle of Optimality* because we respect the *optimal substructure*, meaning that we only incorporate a comparison function between two elements that returns the level of similarity solely based on those elements, as the original approach. This enables the solution composition and the application of dynamic programming, ensuring the optimal solution, which maximizes the similarity between elements. Note that the optimal alignment may not always have the lowest distance between two traces or have the highest fidelity metrics. In this context, the optimal alignment is the one that returns the highest score based on the comparison function between pairs of snapshots and the scoring scheme. A detailed description of the algorithm and the full demonstration are available in the companion technical report [25]

Our algorithm has four parameters that allow customizing its behavior to the particular system of interest: *MAD*, gap opening penalty (P_{op}), gap extension penalty (P_{ex}), and *LCAW*. *MAD* and *LCAW* are used by the similarity function $\mathcal{S}()$, and the other two are used in the algorithm that computes the alignment. These four parameters are described next.

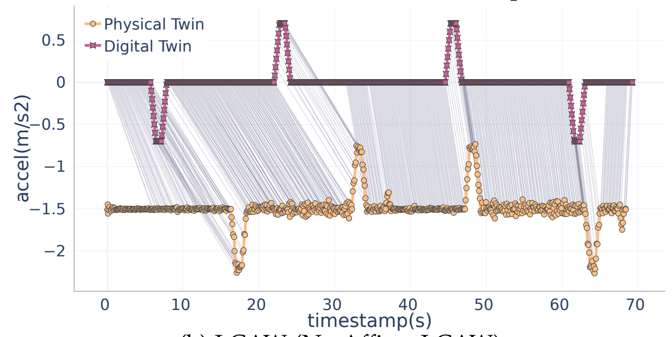
1) *MAD* values. First, we need to define the aforementioned *MAD values* for each attribute of the snapshot. Each *MAD* value determines the maximum distance between two values of the same attribute to be considered similar, i.e., so that the states they represent can be matched because they are "sufficiently similar." Normally, they are defined proportionally (our experiments have shown 2 to 3 times) to the accuracy of the measuring instrument used to obtain the attribute values or to the allowed tolerance of the physical property represented by the attribute.

In the elevator case study, we focus on only one attribute, the acceleration, and then we took a *MAD* value of 0.15 m/s^2 (note that the accuracy of the accelerometer is 0.05 m/s^2, i.e., the *MAD* value is 3 times the accuracy). A very small *MAD* normally produces alignments with a low number of matched pairs but with a small distance between the traces (recall that *MAD* value establishes an upper bound for the Fréchet distance *FD*). On the other hand, alignments with very high *MAD* values can produce meaningless results by matching inconsistent values.

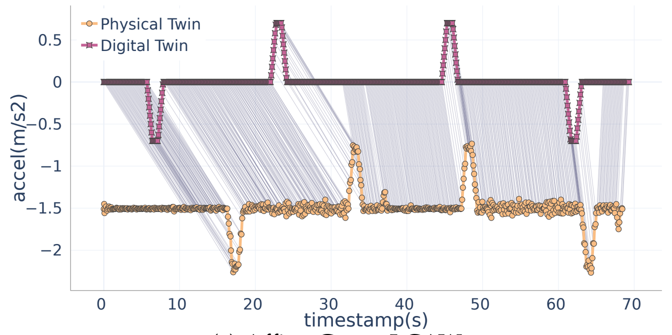
2) Scoring schema values (P_{op}, P_{ex}). We also need to decide the scoring scheme used to select the optimal values of the alignment: **Match**, **Mistmatch**, **Gap opening penalty** (P_{op}), and **Gap extension penalty** (P_{ex}).



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



(b) LCAW (No Affine, LCAW)



(c) Affine Gap + LCAW

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

Instead of being a constant number (as when comparing character sequences with the NDW or BLAST algorithms), we decided that the reward of a *Match* will depend on how similar the two snapshots to compare are. Namely, the reward will be a number in the range $(0, 1]$ that captures the similarity between the attribute values, as computed by the *similarity* function. The score for a *Mismatch* will be 0. This occurs when the two snapshots are farther away from the *MAD* value, but the algorithm considers they should be paired to obtain an optimal alignment. The opening (P_{op}) and extending (P_{ex}) gap scores will be negative numbers, representing penalties. The values of these two parameters are usually set based on tuning experiments. We realized that, in the three systems that we analyzed, the alignment algorithm works well when the opening gap cost P_{op} is in the range $[-0.5, -0.1]$, i.e., between 10% and 50% of the perfect Match reward. Parameter changes within this range of values are not significant, as we shall later see.

3) Low-complexity area weight (*LCAW*). *Information theory* [26] quantifies information using the concept of *entropy*, defined as the amount of uncertainty present in the possible outcomes of a random variable. The entropy is maximum when the probabilities of all possible values of the variable are equal. As the probability of one of the values increases over the others, the entropy decreases. In the elevator example, the regions with lower entropy are the ones at constant speed, i.e., the ones at zero acceleration, which cover most of the trace and contain noisy values around zero. These areas contain less information and do not contribute as much to characterizing the behavior of the elevator compared to the curves. Alignment algorithms such as BLAST aim to maximize the similarity between aligned pairs. However, common low-entropy regions return high similarity scores, shifting the attention from more characteristic subsequences. To address this issue, we mask these snapshots by reducing their reward to refocus the algorithm on aligning snapshots in other regions. In the case of the elevator, these regions can be easily identified because their values are lower than the accelerator accuracy ($0.05\ m/s^2$), i.e., they are indistinguishable from 0. To mask them, we define the *Low-complexity area weight* (*LCAW*). This is the weight that we need to assign to values in the low-complexity areas to reduce their influence. This number is the product of two factors: the percentage of snapshots that are relevant to the behavior of interest (r), and the influence weight (s) we want to assign to them. In the elevator case study, its traces showed that it was changing speed only 10% of the time, so $r = 0.1$. We also considered that $s = 1/20$, meaning that the weight of non-relevant snapshots was $1/20$ of the weight of relevant ones. Therefore, we set *LCAW*$= 0.005$.

To show the effect of taking into account the low-complexity areas, let us consider the alignment of the traces depicted in Figure 3 using *MAD* 0.15 without considering low-complexity areas. The resulting alignment is presented in Figure 4a. All four acceleration peaks are expected to be aligned as they represent the same behavior. However, we observe that the first two acceleration curves, which indicate the movement from the fourth floor to the ground floor, are not aligned. This misalignment, however, is not observed in the other two acceleration curves, which are aligned with their respective counterparts.

The problem is with the algorithm's reward scheme. The similarity function returns a value in the range (0, 1] by comparing two snapshots and returning a higher value for higher similarity. In this example, the reward scheme favors alignments in zero-acceleration regions rather than those in acceleration curves. For instance, consider a DT acceleration value of $0\ m/s^2$ and a PT noised value of $0.005\ m/s^2$ from the zero-acceleration area. Their difference is 0.005, normalized with respect to the MAD at $0.15\ m/s^2$. The reward is $1 - (0.005/0.15) = 0.97$. In contrast, if we examine two potentially 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 consistently prioritizes alignment in the zero-acceleration areas to maximize the alignment final score.

To mitigate this issue and guide the algorithm into considering the values in the acceleration curves over the zero-acceleration regions, we apply the LCAW. Subfigure 4b shows the use of low-complexity areas. This way, we force the algorithm to focus on the areas that characterize the relevant behavior, resulting in a more significant alignment. In this alignment, snapshots whose acceleration is lower than $0.05\ m/s^2$ are considered low-complexity areas, with *LCAW*$= 0.005$.

To see the effect of applying *affine gap* to an alignment, we can compare the Subfigures 4b and 4c. In the latter, we can observe that the gaps in the first 20 timestamps are grouped together instead of alternating. This approach enhances the interpretation of the alignment, making it easier for us to identify the initial delay between the traces. This delay can be measured by estimating the number of consecutive gaps at the beginning of the trace and then multiplying it by the snapshot sampling rate.

Combining *LCAW* and *affine gap* provides two key benefits: a) more accurate alignments by forcing the alignment of the behaviors of interest, and b) longer, more contextually relevant gap groups for a better interpretation of the results.

3.5 Fidelity metrics

After aligning the twins' traces, we need to determine how accurately the DT traces match those of the PT. To do this, we have defined three measures to assess the quality of the alignment: the percentage of matched snapshots and two different distances between the traces.

The first metric is the *Percentage of matched snapshots* (*%MS*) between the PT and DT traces, which is defined as: $\%MS_A^+ = \#X^{A+}/\max(\#X, \#Y) * 100$. The more matched snapshots, the closer the two trajectories are. Furthermore, if the alignment is poor, the results of the distance metrics become meaningless.

To measure the distance between the two trajectories we can use both the *Fréchet* and *Euclidean distances*, as described in Section 2.3. The former calculates the maximum distance of all matched snapshots, while the latter returns the average distance between them. The snapshots in the low-complexity areas are excluded to avoid artificially lowering the average distance.

If the PT and DT traces are identical, all snapshots are matched (*%MS*=1) and the two distances are 0. This means that both twins have gone through the same states.

Note that the *MAD* value has a key influence in these three metrics: the larger the *MAD*, the greater the number of matched snapshots. However, a high *MAD* does not only increase the two distances (since the *MAD* value establishes an upper bound for the Fréchet distance), but it can lead to incorrect matches (cf. Section 3.4). This is why we always have to find the right balance between the *MAD* value, the number of matched snapshots, and the distances between the traces. The selection of the *MAD* value and the rest of the algorithm parameters is discussed below in Section 4.

Other metrics of interest also include the *Percentage of mismatches*, $\%m_A^- = \#X^{A-}/\max(\#X, \#Y) * 100$, and the *Percentage of gaps*, $\%m_A^G = 100 - (\%m_A^+(X) + \%m_A^-(X))$. A larger percentage of gaps than mismatches may indicate a delay between the behaviors, whereas a high percentage of mismatches could mean a temporary deviation of the two behaviors.

Further relevant measures take into account the *Number of sequences of consecutive gaps* and the *Average length of the sequences of consecutive gaps*. These metrics help analyze the distribution of gaps in the trace and provide insights into the potential differences between them.

For example, if there is a small number of very long gaps, it may indicate a delay in the behavior of one of the twins.

Similarly, if there are many clusters of very short gaps, it may be a symptom of stuttering behavior in one of them. The application of *affine gap* increases the length of the gaps when possible and helps in the detection of these symptoms. The precise definitions of all these metrics can be found in the technical report available on the companion website [27], [28].

To analyze the impact of applying the BLAST optimization techniques on the fidelity metrics, we have aggregated the statistics of the ten executions of scenario (4-0-4) as shown in Table 1. We found that the application of LCAW reduces the %MS but does not considerably affect the distance metrics. This is because to match the high-relevant areas, we need to include gaps in the more abundant zero-acceleration areas, which reduces the number of matches but ultimately results in more accurate alignments.

The statistics in Table 1 indicate that the usage of *affine gap* has a minor effect on the fidelity metrics (%MS, FD, ED), but it does have a significant impact on the gap distribution. The average length of gaps increases from 5 to 11 without any increase in the total number of gaps. Additionally, these gaps are now consolidated into 10 groups instead of the previous 20+, which reduces the non-aligned regions in the alignment by half. This results in a clearer and more straightforward interpretation of the alignment. These findings match with the benefits introduced at the end of the previous section. A more detailed statistical analysis of the effects of both optimization techniques is available in the technical report [25].

TABLE 1: Average fidelity metrics for the ten executions scenario (4-0-4).

Algorithm	Fidelity			Gaps			
Variant	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

4 Parameter tuning

We have already mentioned that the *MAD* value significantly influences the percentage of matched snapshots and the distances between the DT and PT traces. Another aspect to analyze is the possible influence of other parameters of the algorithm, namely gap opening and extension penalties, on the three fidelity metrics. This section discusses the effect of these parameters on fidelity and how they can be used to decide whether the DT is faithful to the PT or not.

In order to showcase the use of our algorithm with a more favorable alignment, we will be using execution 01 instead of execution 04 for the remaining part of the Elevator example. This alignment exhibits similar behavior in terms of acceleration curves but does not include the initial delay. By eliminating the initial delay, we can focus on the similarity of the acceleration curves and the synchronization between the movements of both elevators.

4.1 Gap tuning

To understand the effect of the algorithm parameters, we conducted an experiment with different values for *MAD*, P_{op} and P_{ex}, and computed the resulting metrics. *MAD* values ranged between 0.1 and 0.22 with increments of 0.04. P_{op} values ranged between -3.0 and -0.5 with increments of 0.5. Finally, P_{ex} values ranged between -2.0 and 0.0 with increments of 0.1. This results in 504 alignments.

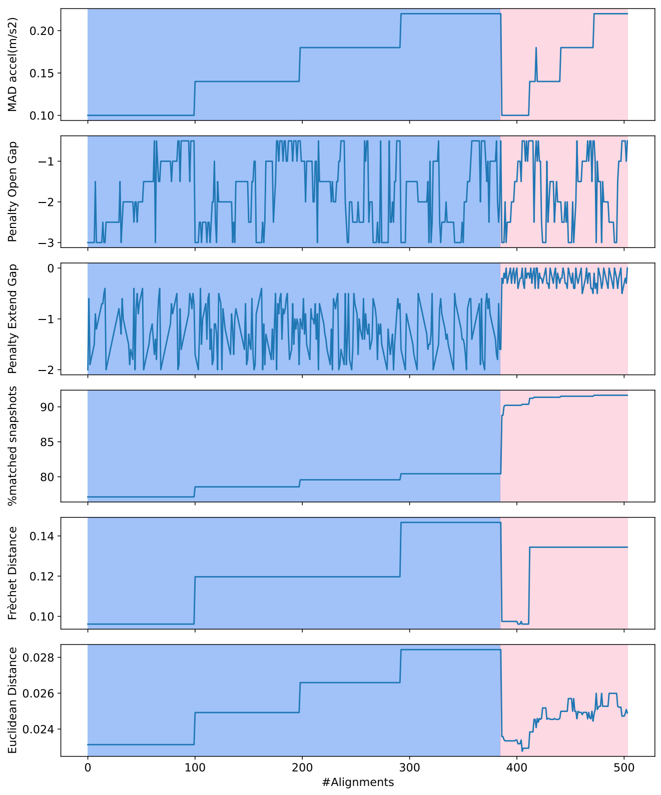


Fig. 5: Results for the 504 alignments of scenario (4-0-4) sorted by increasing *%MS* in the X-axis. From top to bottom: *MAD*; P_{op}; P_{ex}; *%MS*; Fréchet distance *FD* and Euclidean distance *ED*.

Figure 5 shows the results for the 504 alignments of scenario (4-0-4) sorted by increasing percentage of matched snapshots (*%MS*) in the X-axis. The figure shows, from top to bottom, the values of *MAD*, P_{op}, P_{ex}, *%MS*, Fréchet distance (*FD*), and Euclidean distance (*ED*). There is a clear breakpoint where the number of matched snapshots grows, which divides the results into two areas (the right one shaded in pink). These plots show that what really makes the difference is the value of the penalty P_{ex}. As long as P_{ex} is between -0.5 and 0.0, the number of matched points is greater than 89%. Within that range, the *MAD* value is correlated with the two distances. However, the correlation is not linear. This is because small variations of the *MAD* value may cause the alignments to change: an increase in *MAD* may mean that new pairs of snapshots are included in the alignment. We can see that this has a very small effect on the Euclidean distance (that always stays between 0.024 and 0.026), but it can affect the Fréchet distance. Interestingly, the value of penalty P_{op} does not play a significant role when it maintains in the range $[-3, 0]$. Therefore, our assignment $(P_{op}, P_{ex}) = (-1.0, -0.1)$ produces stable alignments on which the distance metrics can be used.

4.2 *MAD* effect

In Section 3.5, we defined three metrics to assess the quality of the alignment: the distances between the traces (Fréchet and Euclidean) and the percentage of matched snapshots (%*MS*). Intuitively, the smaller the distances and the larger the %*MS*, the closer the traces will be. In fact, for a fully faithful DT, the distances are 0.0, and %*MS* is 1.0.

As mentioned above, these three metrics depend on the *MAD* value, which should be chosen in order to (1) maximize the percentage of matched snapshots and (2) minimize Fréchet and Euclidean distances (if not both, at least one of them depending on the particular problem).

Note that the *MAD* value cannot be carelessly chosen: it must be large enough to identify similar snapshots adequately but small enough to produce meaningful alignments.

As discussed in Section 3.4, the *MAD* value normally should be around 2 or 3 times the accuracy of the measuring instrument used to obtain the corresponding attribute value. In the Elevator case study, where we have one single attribute (the acceleration) and the accuracy of the accelerometer is $0.05\ m/s^2$, we obtain a *MAD* value of 0.15. Using this value, the fidelity metrics obtained for the resulting alignment are %*MS*=0.94, *FD*=0.12, and *ED*=0.049.

To summarize, given the traces of two twins, the first step is to set the values for the alignment algorithm parameters, namely P_{op}, P_{ex}, *LCAW*, and *MAD*. We have seen how the first two can be set to -1.0 and -0.1, respectively; this is consistent with the traditional BLAST algorithm recommendations for these values. The *LCAW* parameter determines the weight with which we want to penalize snapshots in low-complexity regions (in our case study, those were the ones whose acceleration was less than the accuracy of the measurement instrument, $0.05\ m/s^2$). *LCAW* was set to 0.005, see Section 3.4. Finally, the *MAD* value was set to three times the accuracy of the measurement instrument used to measure the attribute of interest: 0.15 in this case.

5 Assessing fidelity

This section presents how the above metrics can be used to define indicators of the fidelity of a DT with respect to its PT. In addition, we show how these metrics can be employed to compare the fidelity of various DTs and discuss the impact of the stochastic variability of different twin executions on the fidelity metrics and indicators.

5.1 Fidelity indicators

The degree of fidelity of a DT with respect to a PT can be decided depending on the values of the three metrics. But first of all, it is essential to decide whether the alignment is sufficient to make the measurements meaningful.

- For example, if the percentage of matched snapshots (%*MS*) is below 90% ($\pm 2\%$), we could conclude that the traces could not be properly aligned, and therefore no faithful behavior can be expected.
- If %*MS* is between 90% and 95% ($\pm 2\%$), the alignment is low, but the distance metrics will make sense and can be considered. In general, the acceptable distance between the traces is application-dependent, and whether it is the Fréchet or the Euclidean distance that matters.
- An alignment with %*MS* above 95% ($\pm 2\%$) could be generally considered good enough, and the degree of fidelity depends on the distance between the traces.

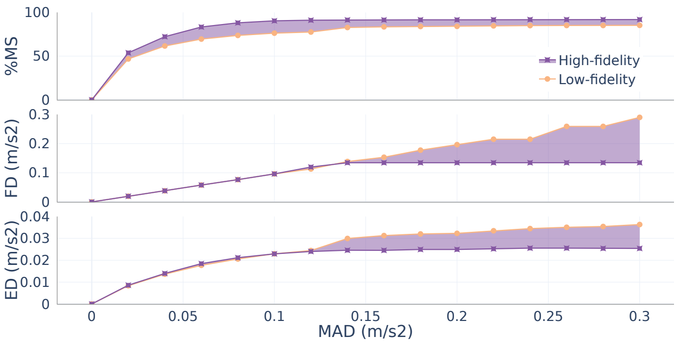


Fig. 6: Fidelity comparison of two DTs for scenario (4-0-4). Execution 01.

Of course, these thresholds depend on the application and must be determined by the end user. In our examples we have used those commonly used in general engineering environments [29]. Note as well that we are assuming a maximum permissible error (MPE) of 2% [29] for the assessment of %*MS*, since most times thresholds are not completely accurate.

In execution 01, we obtain an alignment able to match 94% of the snapshots (%*MS*=0.94), a Fréchet distance *FD*=0.12, and a Euclidean distance of *ED*=0.05. The percentage of matched snapshots is sufficient to consider the distance metrics. Then, the Euclidean distance, which captures the average difference between matched snapshots, is below the accuracy of the measuring instrument, so we can conclude that it is acceptable. However, the decision on whether the Fréchet distance ($0.12\ m/s^2$) is acceptable or not also depends on the application.

5.2 Multi-fidelity digital twins

Using more than one DT of the same PT is an emergent technique called *multi-fidelity digital twins* [14], borrowed from the modeling and simulation community [30]. It addresses the problem that occurs when very accurate simulations are computationally too expensive, and a lower resolution model can provide “accurate enough” simulations. We can then have several digital twins, each of a different resolution, and use the most appropriate results when the very accurate simulations are computationally prohibitive. Deciding whether the degree of fidelity of a model is good enough or not becomes a critical issue in this context, and this is precisely the motivation of our work.

To illustrate how our approach can be used to compare the fidelity of two DTs of the same PT, we developed a more abstract and lower-resolution model for the elevator case study in UML. We employed the USE tool [31], which allows executing the UML specifications [32]. The USE model of the Elevator is available from [27], [28] and employs a simple algorithm to compute the acceleration, in contrast to the more elaborated algorithm used by the *Elevate* simulator.

Figure 6 displays the values of our three metrics (%*MS*, *FD*, and *ED*) for different values of *MAD* in the range $[0.02, 0.3]$. We can see how the commercial simulator, included in the legend as “High-fidelity", obtains larger %*MS* and lower distance values than the USE model. In this case, given that the alignment of the low-resolution USE model does not even reach 90% of matched snapshots, we can conclude that it does not represent a faithful twin of the physical system.

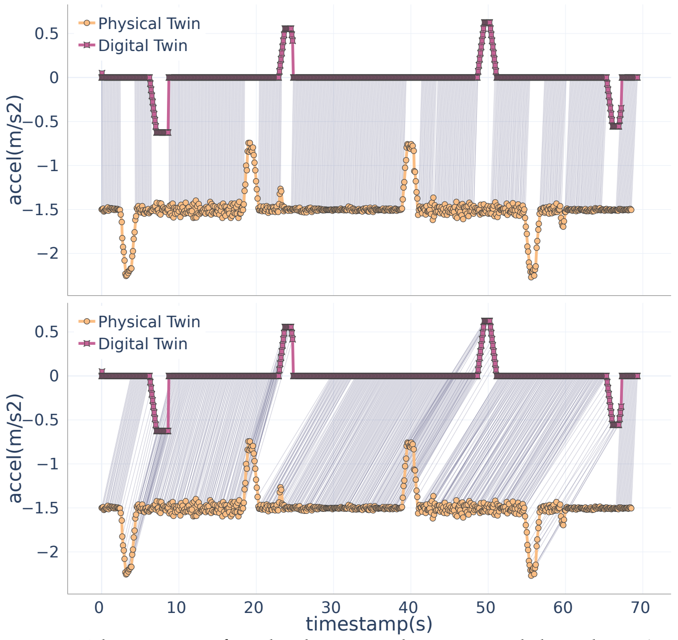


Fig. 7: Alignments for the low-resolution model with *MAD* values 0.12 (top) and 0.15 (bottom).

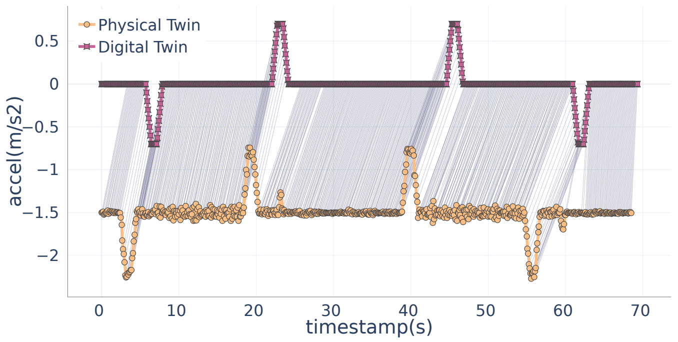


Fig. 8: Alignment of the high-resolution model for *MAD*=0.12.

These types of plots are also very useful because they clearly allow us to decide the *MAD* value that obtains the lower distance with better *%MS*. As the *MAD* value increases, both *%MS* and the two distances also increase. For the high-resolution model, these three values reach a plateau when *MAD* is $0.15\ m/s^2$. After this point, if we increase the *MAD*, the *%MS* does not change, i.e., the alignment stabilizes. All three values reach a plateau in the high-resolution model because the algorithm is able to align almost 90% of the points and, as we increase the *MAD*, there are no more snapshots that can be included in the alignments—mostly because these are gaps needed to perform the alignment, due to a delay between the traces. Therefore, the values of the distances will not change. This is not the case for the low-resolution model, whose values keep worsening as the *MAD* value increases.

Figure 6 also shows that there is no difference between the distances of the low- and high-resolution models when the *MAD* is below 0.12. However, the *%MS* of the USE model is lower. In fact, in order to have a meaningful fidelity indicator, it is essential to ensure a minimum *%MS*. To graphically illustrate this issue, Figure 7 shows the alignments obtained for the low-resolution model corresponding to *MAD* values 0.12 and 0.15. Their *%MS* values are, respectively, 0.78 and 0.83. In the first alignment, the acceleration curves were not included, as they were not emulated with the required precision. In the second figure, we were finally able to align points of the behavior of interest. However, the %MS still did not reach 0.90. Based on the alignments, to improve this model, we need to improve the precision of the acceleration curves.

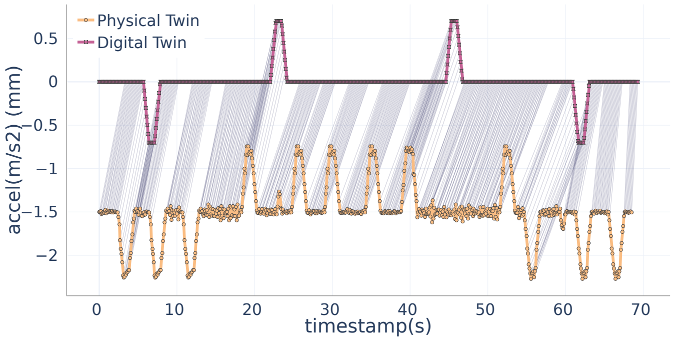


Fig. 9: Alignment of the DT scenario (4-0-4) with an anomalous PT trace *MAD*=0.15.

Something that Figure 6 also shows is that the three fidelity metrics already obtain acceptable values in the case of the High-fidelity model: for a *MAD* of 0.12, we obtain *%MS*=0.91, *FD*=0.10, and *ED*=0.024. So, in this case, a *MAD* value about 2 times the accuracy of the measurement instrument can be “good enough.” Figure 8 shows the alignment obtained for *MAD*=0.12, with 91% of the snapshots matched (excluding low-complexity areas).

The additional metrics, namely the number of individual gaps and the mean length of the gaps, are also useful. For instance, in the alignment displayed in Figure 8, there are 14 sequences of gaps, with a total of 114 gaps. The mean length of these sequences is 8. Since the sampling period is 0.1, this means that the average delay is almost a second in the trace execution. This is consistent with the behavior of DT, which lags behind PT, as shown in the figures.

5.2.1 *Additional synthetic scenarios*

To showcase the potential of the approach, we performed analysis for two additional synthetic scenarios created by manipulating the actual measurements of the elevator scenario (4-0-4). These examples demonstrate how our algorithm identifies anomalous and erratic behavior and how it impacts the performance metrics.

Acceleration Anomalies in the PT. We have incorporated a set of acceleration patterns into the PT trace, which the algorithm classifies as mismatched and/or gaps, effectively identifying and locating the anomalous behavior within the trace, presented in Figure 9. The original traces' fidelity metrics were %MS 94.2, FD 0.12, and ED 0.05. However, after including the anomalies, the %MS reduced to 75.07, while the distances remained the same. This is because the newly added snapshots are not aligned, and only matched snapshots are considered in the distance metrics.

Random noise in the working range. In this experiment, we generated a new trace consisting of random noise in the acceleration working range. However, when we tried to align this trace with the original sequence, we observed an alteration in the alignments, making it impossible to properly match any sequences. Figure 10 shows the misalignment of the random noise trace. In the original scenario, the fidelity

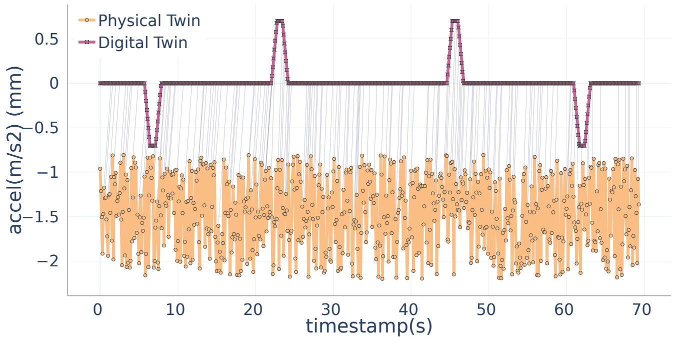


Fig. 10: Alignment of the DT scenario (4-0-4) with random noise in the working range. *MAD*=0.15.

metrics for this MAD were as follows: %MS 94.2, FD 0.12, and ED 0.05. However, when we tried aligning this random noise trace, the %MS dropped to 22.04, with a mismatch percentage of 76.2%. Additionally, the distances between the aligned sequences increased, with FD rising to 0.15 and ED to 0.07. This experiment shows that our algorithm and metrics can effectively distinguish a proper alignment from a random sequence, as demonstrated by the differences in the fidelity metrics.

All the experiments conducted and the statistical results of their analysis are available in the companion technical report for the elevator [23].

6 Evaluation

The evaluation is divided into two subsections. In the first subsection, we consider a set of three additional CPSs to showcase and discuss further aspects of our proposal. This includes comparing models of the same system or considering multiple attributes in the comparison function. This part of the evaluation aims to demonstrate our proposal's applicability and feasibility in other examples and illustrates how to adapt it to other systems. The second subsection focuses on quantitative analysis. We will start by analyzing the proposal against related proposals in the literature and discussing the differences. Finally, we will measure the time complexity and scalability of our proposal and the others under consideration.

6.1 Demonstration Cases

In this subsection, we showcase the application of the algorithm for three additional CPSs to illustrate and discuss further aspects of our proposal, which complement those shown in the running example. First, we introduce the example of an incubator system for which we have two models with different computational costs and accuracy. We want to evaluate these models' fidelity to decide which is the best to use in two different working scenarios. Next, we analyze a robotic arm that illustrates the use of our proposal when the snapshots contain several parameters (the robot servos). Furthermore, in this case, our starting point is the model of the system with the required behavior, and we want to test how faithfully a particular robot behaves according to it. Finally, we developed an additional case study demonstrating how our approach supports traces with Boolean and enumerated types.

6.1.1 Incubator: Compare models

Description

A research team from the Aarhus University in Denmark designed and built the DTS for an incubator [33], [34]. The incubator consists of an insulated box with a heating device and a fan. The controller reads the values from the temperature sensors and actuates on the fan and the heating device, turning them *on* and *off* in order to maintain a stable temperature inside the box between 30 and 35 degrees Celsius. For this purpose, the controller uses a heating pattern in which it first activates the heating for a certain number of seconds (Heating time, Ht) and stops it for a further time (Heating gap, Hg). The controller then checks whether the temperature has reached the upper limit. If not, it reactivates the heating pattern. Two scenarios following different strategies are considered: one uses a short pattern (Ht: 3 s, Hg: 2 s), and another a long one (Ht: 30 s, Hg: 20 s).

Two simulation models were developed to serve as DTs for the incubator, one with two parameters (*2-P model*) and one with four (*4-P model*). Their purpose was to accurately capture the non-linear behavior and transient changes during temperature fluctuations. The 2-P model is a simple linear model that is computationally efficient and provides approximate predictions for the incubator temperature. The 4-P model is a non-linear model that captures more accurately the system's transient behavior but at a higher computational cost. Our goal is to measure the fidelity of these two DTs and to check how accurate they really are.

Fidelity assessment

The incubator utilizes two DHT22 sensors [35] for measuring the temperature inside the box. According to their datasheets, these sensors have an accuracy of $\pm 0.5°C$. Therefore, to ensure a sufficient level of fidelity, the *MAD* should fall within the range of 1 to 1.5 ºC (cf. Section 4.2).

In the first scenario, Ht3-Hg2, the controller activates and deactivates the heater at very short intervals. This results in a heating pattern where the two-second cooling interval is hardly noticeable. The challenge for the models lies in the transition from the heating state to the cooling state, where non-linear behaviors happen. Both models fail to accurately replicate this transitional behavior, and our statistics show an alignment below 90%. Namely, the ranges of the 3 fidelity metrics for *MAD* values in the range $[1, 1.5]$ that the 2-P model obtains are *%MS*=$[0.86, 0.89]$, *FD*=$[0.433, 0.712]$, and *ED*=$[0.122, 0.180]$. In turn, the ranges of values of these metrics for the 4-P model are *%MS*=$[0.89, 0.92]$ *FD*=$[0.591, 0.734]$, and *ED*=$[0.135, 0.178]$. This implies that, given that *MAD*, neither model is able to faithfully reproduce the behavior of the incubator.

In the second scenario, the activation and deactivation periods of the heater are longer, providing both models with an opportunity to better predict the 20-second cooling part. The 2-P model, which again struggles to model transitions effectively, incorporates a small cooling effect after each heating period. The result is a sawtooth pattern that does not resemble the actual behavior of the system, see Fig. 11 (top): statistics indicate that less than 80% of points are aligned, with value ranges of *%MS*=$[0.72, 0.77]$, *FD*=$[0.890, 0.974]$, and *ED*=$[0.228, 0.319]$ for *MAD* values in the range $[1, 1.5]$.

TABLE 2: Fidelity results for scenario Ht30Hg20.

	2-P Model			4-P Model		
MAD (ºC)	*%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

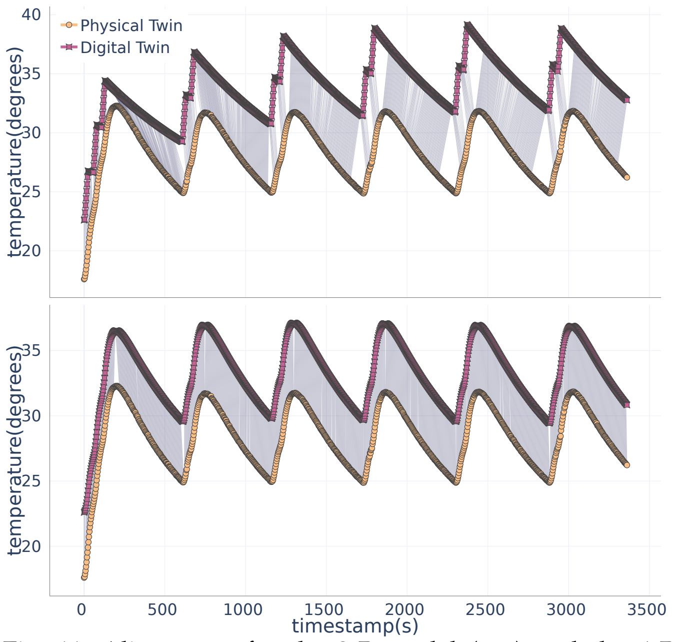


Fig. 11: Alignments for the 2-P model (top) and the 4-P model (bottom) in scenario Ht30-Hg20 with *MAD=1.20ºC (Note: the PT trajectory is displaced 5 degrees for improved visualization).*

The 4-P model performs better in this scenario because it is less sensitive to these long cooling gaps and can accurately capture transitions in these conditions, resulting in a total of 98% aligned snapshots, with $\%MS$=$[0.97, 0.98]$ FD=$[0.65, 0.65]$, and ED=$[0.100, 0.127]$. Thus, the 4-P model, as shown in Table 2 and Fig. 11 (bottom), faithfully emulates the incubator behavior. Namely, it only deviates on average 0.1°C from the PT, with short peaks of just 0.65°C difference, being able to match more than 97% of the snapshots.

Figure 12 shows the comparative trend of these two models. The 4-P model reaches a plateau in all three metrics near 1ºC, while the other model continues to increase the percentage of matched snapshots at the expense of increasing the average and peak distances.

After performing the analysis, we can conclude that the 2-P model does not accurately replicate the behavior of the incubator. It fails to approximate precisely the temperature and transitions of the system. In turn, the 4-P model is not reliable when the heating time and gap are too short. However, it accurately simulates the system behavior when the transition is slower, as observed in the second scenario.

To compare the fidelity of different behavioral models of the same system, we analyze the alignments and fidelity metrics values. The model that returns a higher %MS and

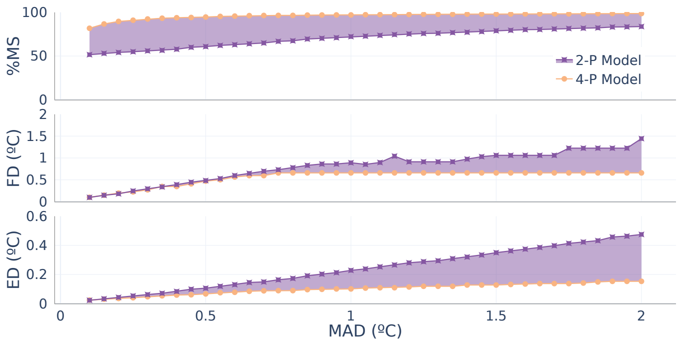


Fig. 12: Fidelity metrics comparison of the incubator for scenario Ht30-Hg20. (*Note that the scale and units are different from Fig. 6*)

lower ED and FD is considered more faithful. However, determining if the more faithful model is the most suitable for a specific problem is not straightforward. Other factors, such as whether the model accurately reproduces behavior sequences relevant to the given application, even if they are fewer in number of snapshots, need to be taken into account. Similarly, if a model has a lower %MS than another but reproduces certain behaviors of interest with greater precision, it could be more suitable, even if it is not the most faithful overall. Our tool provides both local and global assessments of different characteristics that influence fidelity. Still, the domain expert must make the final decision to determine the most appropriate model for their specific problem.

6.1.2 Robotic arm: Multiple numerical attributes

Description

This case study shows another use case of our proposal, where the DT acts as the oracle, and we want to check whether the PT is faithful enough to the DT. More precisely, in this case study, we are interested in evaluating whether a given physical device, namely a robotic arm, can behave according to the requirements demanded by the user. These requirements are defined by models that specify the expected behavior in various usage scenarios. That is, our starting point is a set of engineering models [36], and we need to measure the fidelity of the robot we plan to use in our plant, and whether it fits its purpose: transport small and light objects, similar to the crane example presented in the introduction section of this paper [12]. There are several robotic arms on the market capable of doing this job, each with different features and costs, e.g., the ABB GoFa CRB 15000, the Nyrio Ned, or the Arduino Tinkerkit Braccio robot. These types of robots have six servos that control their joints: the orientation of the base (s1), the position of the shoulder (s2), the elbow (s3), the vertical position of the wrist (s4), its rotation (s5), and the gripper (s6).

In this section, we will analyze the last one, the Braccio robot [37], a low-cost robotic arm that has been successfully used in different applications [38]. Given its reduced price, we are interested in testing whether it could serve our purposes in two usage scenarios, as they are not too demanding. The first scenario (named *Simple Moves*) corresponds to a set of four commands involving 2 or 3 servos at a slow pace. The second represents a more complex movement (named *Pick&Drop*) in which the arm has to grasp a distant object,

TABLE 3: Fidelity results for scenario *SimpleMoves*.

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	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

TABLE 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	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

raise it 30 cm, move it 50 cm, lower the arm, release the object, and return to the initial position. We developed some models with the required behavior of the servos (they are available on the paper companion website [27], [28]). For simplicity, we only considered the movements of the robot assuming very light loads, so they do not affect the speed or the trajectory of the arm.

Fidelity assessment

In the *SimpleMoves* scenario, the arm's movements involve a maximum of 3 servos at a slow pace. This ensures stationary servo positions and generates smooth transitions similar to those produced by the engineering model. We achieve adequate results in this scenario, matching more than 90% of the snapshots. For a *MAD* between 2 and 3 (the accuracy of the servos is 1 degree), the ranges of the three fidelity metrics are as follows: %MS=[90.98, 91.59], FD=2.28, ED=0.23 (see Table 3). These findings indicate that the robot is well-suited to emulate the behavior described in this scenario.

However, when executing the *Pick&Drop* scenario, which involves faster movements and concurrent activation of multiple servos, the results are less satisfactory. The fidelity metrics values decrease because the robot struggles to maintain fidelity in a more complex scenario: %MS=[78.07, 78.78], FD=[2.72, 2.79], ED=[1.63, 1.92] (see Table 4).

To further investigate the causes behind this decline, we calculated the coordinates of the gripper and performed the alignments for both scenarios. The results of the metrics for the alignments using the coordinates are also presented in Tables 3 and 4. Similar to the servo values, the data indicates that the arm performs well in the first scenario but deteriorates in the second. For a clear understanding of why the arm fails in the second scenario, we analyzed the alignment for a *MAD* value of 15 mm, depicted in Figure 13 (the accuracy of the robot coordinates is 5 mm [38], and therefore we considered *MAD* values between 10 and 15). The figure illustrates how the arm remains stationary for extended periods, unable to react within the required response time, resulting in a 20% of gaps in the alignment. These gaps are visible in the alignment plot when the PT stays still for longer periods. Additionally, we observe that Braccio's transition movements are slower compared to the engineering model. However, this discrepancy falls within

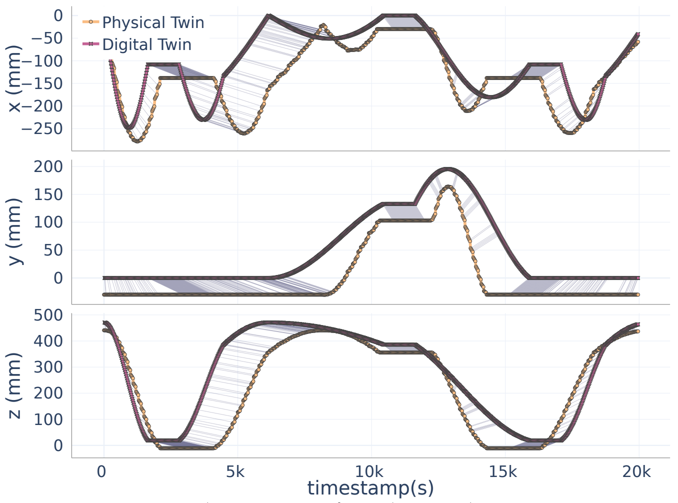


Fig. 13: X-Y-Z alignments for the Pick&Drop scenario, *MAD*=15 mm (Note: the PT trajectory is displaced 30 mm for improved visualization).

the *MAD* range, enabling the algorithm to align most of the snapshots. In conclusion, the Braccio robot would require a faster response time to accurately emulate the behavior established by our engineering model.

In this assessment, the snapshot comparison function is similar to that of a single attribute. It evaluates if the difference between each property of interest is within the corresponding MAD range to categorize it as a match.

When considering more than one attribute (in this example, six), the comparison is more restrictive, including gaps and mismatches, even when only one attribute is outside the MAD range. For example, in the case of the robotic arm, all servo values are crucial in the comparison, and all must be taken into account. If one of the servos is delayed or at a different angle, even though the rest are functioning as expected, this could denote that the robotic arm is performing a completely different task. However, in cases where some attributes are not as crucial in the evaluation, the comparison function could be adapted to reduce the level of similarity to some extent rather than considering the difference as a mismatch. Another option could be to analyze the different properties of interest separately to detect individual inconsistencies.

Finally, note that in this example and in the previous one, there was no need to consider low-complexity regions and the associated *LCAW*. They are only required when the system exhibits large periods of stationary behavior. These segments can affect the alignment because they produce high matching rates that divert the algorithm from searching for alignments in the most relevant parts of the traces.

6.1.3 Lego Car: Boolean and enumerations

Description

The *Lego NXT Car* is the example that we used for the prototype of our approach introduced in [39]. The Lego NXT Car is an example of a robot that comes with several built-in sensors.

Figure 14 shows a snapshot of the robot with its attributes. The robot has a pose-provider that returns its current planar coordinates and the angle of its direction, represented by xPos and yPos. It also has an ultrasonic

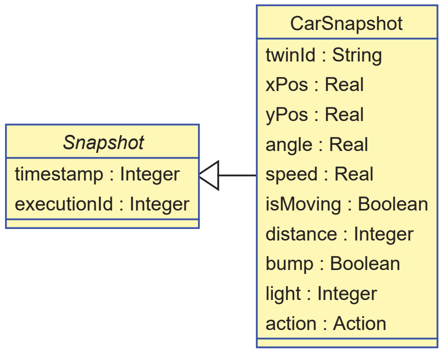


Fig. 14: UML Class of the NXT Lego Car Snaphot

TABLE 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	-	-

sensor that measures the distance to an object in front of it, a light sensor that determines the color of the ground beneath it, and two touch sensors that act as a bumper to detect collisions. Additionally, the robot can detect its current speed and whether it isMoving. The programming of this robot allows it to set states depending on the action it is executing, so it also provides the action that it is currently performing.

To simulate the car's actions, we developed a Digital Twin employing the USE modeling tool [31]. We used it to demonstrate that the behavior of the real car was not totally reliable due to the inaccuracy of its sensors, the slowness of its controllers, and the slack in its parts, as it could not faithfully replicate the same trajectories. However, this is a very illustrative example of when synthetic traces are used as PT, which contain variations synthetically generated.

Fidelity assessment

To align the snapshots, the Boolean attributes such as bump and isMoving and the enumeration types such as action must have identical values. Please note that the MAD for each sensor, as listed in Table 5, is three times the corresponding sensor accuracy. If there is a discrepancy in non-numerical values, or the numerical values differ 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 by increasing its speed by 1.5 during a specific period. However, since this increase in speed is within the MAD, it does not affect the alignment, as we are not as strict. If we reduced the MAD, we would be able to pinpoint this difference. In contrast, the algorithm detects inconsistencies in the bump attribute, identifying two collisions by the end of the trace, and in the action attribute, performing two of the rotations later than the DT predicted. In both cases, gaps are introduced in the alignment since the enumerated

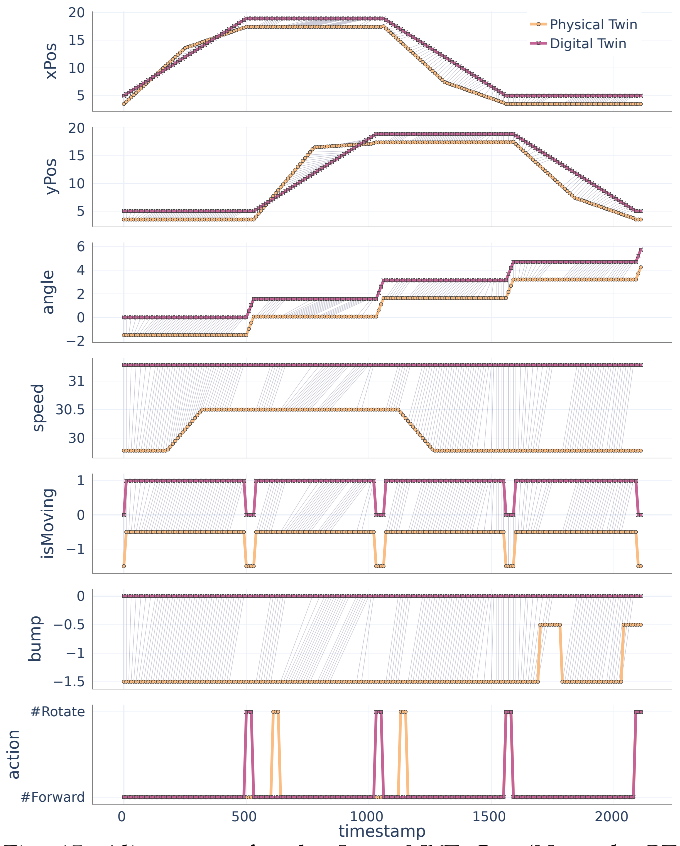


Fig. 15: Alignments for the Lego NXT Car (*Note: the PT trajectory is displaced 1.5 units and the attributes* distance *and* light *were not included for improved visualization*).

and boolean types must strictly match. Considering this, the %MS is 70%, FD is 1.58, and ED is 0.7. This means that although at least 30% of the trace is not matched due to inconsistencies in action and bump detection, the rest of the behavior was reproduced within our requirements. We could measure the distances for every attribute to find which would contribute to better replicating the behavior. This would conclude that attributes such as xPos and yPos include the most significant difference, as opposed to speed or angle, whose values are similar.

Apart from this example, we included a set of different executions in the corresponding Technical Report in [40]. Some of these examples showcase partial inconsistencies in one of the attributes and how they affect the fidelity metrics.

When evaluating the fidelity of snapshots with Boolean or enumerated attributes, the approach is similar to the one with multiple numerical attributes, ensuring that the differences between the attributes of paired snapshots are within the MAD. However, the alignments become even more restrictive. Suppose there are any differences in non-numerical attributes. In that case, the snapshot will be considered a gap or mismatch, which helps to identify inconsistencies, even if they exist in just one of the attributes.

In the Lego car example, these constraints are essential because we want to determine whether Boolean sensors, such as the bumper, are working correctly. If the bumper provides an incorrect value in either the DT or the PT, it could denote that the virtual environment is incorrectly configured or that the sensor is not working. Similarly, if the position is accurate but the enumerated value of the action includes a different value, it could mean that the logic of the DT differs from that of the PT.

Nevertheless, if Boolean or enumerated attributes do not represent properties of interest and should not be considered, they could be excluded from the alignment. Another possibility would be to modify the comparison function to reduce the level of similarity without causing a mismatch. However, in cases where attributes are crucial to the state of the system, the match should be avoided if they have different values, even in only one of them.

6.2 Empirical Evaluation

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

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

RQ.2 What is the time complexity of the different configurations of our algorithm?

RQ.3 How does the execution time of our proposal compare to that of other related approaches?

To answer RQ.1, we performed a comparison study with two of the closest proposals [41], [42]. We analyze the expressiveness of their results and their ability to measure fidelity. We also compare their performance. RQ.2 and RQ.3 are answered by conducting experiments with varying trace lengths and performing regression analysis to estimate time complexity. All detailed results of the fidelity assessments presented in this section are available in the companion technical reports [43], [44].

6.2.1 *Comparison with other proposals (RQ.1)*

To illustrate the differences between our algorithm and the state of the art, we compared our algorithm with two of the closest proposals using the Elevator Case Study. The full comparison, including figures and statistical results, is available in the technical report [25].

Dynamic Time Warping

Dynamic Time Warping (DTW) [42] is a dynamic programming algorithm used to measure the similarity between two temporal sequences, which may vary in speed. It computes an optimal alignment between the two sequences, minimizing the distance between the aligned points. Unlike NDW, it allows one-to-many and many-to-one matches because DTW assumes that one sequence is a time-warped version of the other, i.e., the target sequence is stretched (one-to-many), condensed (many-to-one) or unwarped (one-to-one) with respect to the source sequence. Therefore, DTW does not support the notion of gaps, while NDW takes gaps explicitly into account and assigns a penalty per gap. To show the differences with our proposal, we conducted the same alignment as the one in Subfigure 4c using DTW. The results of this alignment are presented in Figure 16. During our analysis, we discovered two main limitations of the algorithm. First, as it does not support gaps, it aligns all elements, even those with low similarity. For example, it aligns the smoothing braking patterns of the PT (seconds 38 and 68) with zero-acceleration snapshots of the DT trace. Thus, the percentage of matched snapshots is always 100%. Therefore, the alignments produced are not very informative, as they lose information on gaps and mismatches. This is essential to determine differences between traces and to pinpoint anomalies.

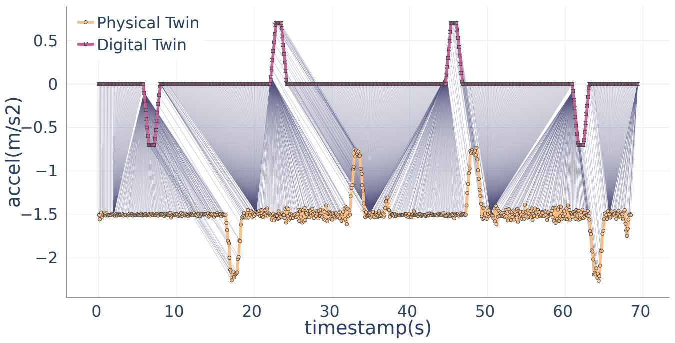


Fig. 16: DTW for Scenario (4-0-4). Execution 04.

Second, and more importantly, one-to-one or many-to-one alignments are not well suited for comparing traces or accurately calculating their distances. For example, in the alignment in Figure 16, the DTW algorithm aligns one snapshot with 28% of the snapshots of the other trace. This leads to unreliable results and does not accurately represent the actual similarity between the traces. Some studies have highlighted the inadequacy of this approach, and various proposals include methods to mitigate this issue, e.g., [45]. This is also why our alignments are one-to-one and may contain gaps and mismatches.

Several alternative implementations of DTW exist, each aimed at addressing specific challenges. For instance, some versions focus on reducing time complexity to speed up computations [46], while others aim to mitigate issues such as overstretching and overcondensing [47], [48]. However, these approaches have their own limitations. They cannot guarantee the optimal solution and typically require fine-tuning parameters. If these parameters are not properly tuned, there is a risk of overfitting to specific characteristics of the training data.

To avoid these potential drawbacks, we restrict our comparison to the original implementation of DTW [42]. This allows for a more general and straightforward comparison to other proposals, as the original algorithm guarantees optimal solutions without the need for parameter tuning.

Online Validation of DTs by Lugaresi et al.

The second proposal is the approach from Lugaressi et al. [41], which introduces three metrics to validate the DT at run-time using trace alignment algorithms based on dynamic programming. The evaluation is performed at two different levels of abstraction: events and KPIs. Even though our proposal analyzes the traces offline, this is not critical for comparing the behavior of the algorithms applied.

Event-level validation. To evaluate the sequence of events, they used the Longest Common Subsequence (LCSS) algorithm, which aligns two sequences of characters and finds the longest sequences of equivalent elements between them. A parameter δ is established to represent the time window in which two events could be aligned. The authors propose a metric that calculates the length of the LCSS divided by the longest trace. This provides the percentage of snapshots that are included in this subsequence.

To compare this approach, which we call LCSS-Events, we added events to the raw trace of Scenario (4-3-2-1-0-

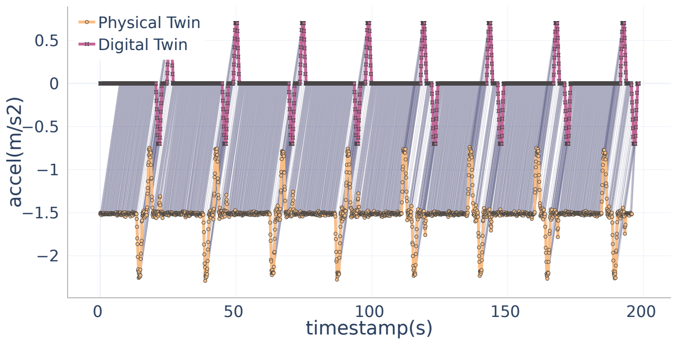


Fig. 17: Affine Gap + LCAW for Scenario (4-3-2-1-0-1-2-3-4).

TABLE 6: 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

1-2-3-4) in Figure 17, in which the elevator goes down, stopping at every floor, and then goes up doing the same. We represented the events with four types: "Down," "Up," "Brake," and "Arrival." Each event was manually assigned to a specific snapshot, corresponding to an acceleration change during the elevator's operation. As a result, the trace included 24 events.

One problem with raising the level of abstraction to events is that their identification may not be trivial. Our approach enables the alignment of snapshots at any level of abstraction, from events to raw traces.

By gradually increasing the value of δ, we obtained the results presented in Table 6. Starting from 6.9 seconds, where no subsequence is returned, we reach the maximum score of 1 at 7.8 seconds. These results indicate that all the events are included in the correct order, but they are delayed by 7 to 7.8 seconds. Visualizing the traces aligned by our algorithm made this interpretation easier. However, this result alone does not identify specific reasons for the delay. It only shows that a certain similarity value is achieved with a certain δ.

This approach does not provide the resulting subsequences, which prevents reasoning about the results when they are bad, based on a single metric. Even if subsequences were provided, stuttering in the trace could result in short subsequences. This is something that could occur even if the traces have a high similarity due to alternating behavior.

KPIs-level validation. For the KPIs-level validation, we defined the KPI *average time between floors* as the time between the events ("Up," "Down") and the stopping of the elevator ("Arrival"). We applied this metric to the scenario (4-3-2-1-0-1-2-3-4) in Figure 17. To evaluate KPIs, the authors propose two metrics.

The first one also uses the LCSS algorithm, which we call LCSS-KPIs, with an additional parameter ϵ that determines the maximum difference between two KPI values to be considered similar. It then divides the resulting LCSS by the length of the longest trace. By gradually increasing the value of ϵ in the adapted trace, we obtain the results in Table 7. Based on these results, we can only conclude

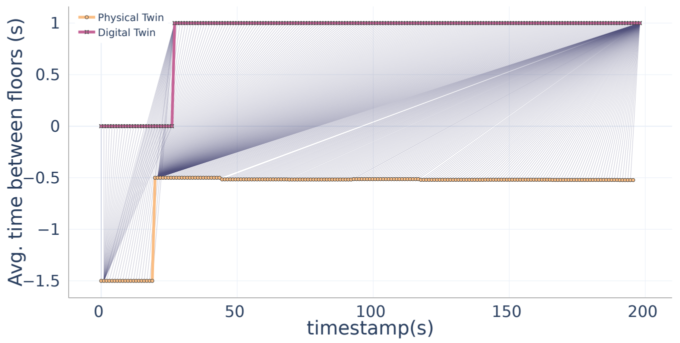


Fig. 18: DTW alignment for Scenario (4-3-2-1-0-1-2-3-4) using KPI *avg. time between floors*. (*Note that it includes only 10% of the values for improved visualization*)

that 0.5 seconds is the maximum difference between the furthest two average values. It should be noted that, as in the previous algorithm, the resulting subsequences are not provided, which makes it difficult to identify and analyze any discrepancies. In addition, there may be instances where partial matches are found in certain areas of the sequence, but a sequence of unaligned values separates them. Such cases would result in a low score, as with stuttering, since this metric suffers from the same limitations as the previous one.

Additionally, when dealing with KPIs, the aggregation of values may mask anomalous events. This is because outliers may take some time to affect certain metrics, which results in an inefficient monitoring system.

TABLE 7: 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

The second metric applied to KPIs uses DTW to align a normalized version of the input sequence values. We refer to their implementation of DTW as DTW-Lugaressi. The results are included in Figure 18, in which we see how the KPI value changes as the execution progresses. This alignment is greatly affected by over-stretching and over-condensing, introduced in the previous section, which leads to unreliable results. For instance, in this particular alignment, one KPI value of the PT is aligned over 1700 times. Therefore, it is difficult to draw any further conclusions from these results due to the strong influence of this phenomenon.

In addition to the aforementioned limitations of the DTW algorithm for aligning traces, the normalization process can also hinder the comparison of traces with large differences between their values (cf. [25]). Let us assume we normalize the following traces to measure their similarity using the proposed metric:

$$A = \{0.05, 0.05, 0.05, 0.05, 0.05\} \quad \overline{A} = \{1, 1, 1, 1, 1\}$$
$$B_1 = \{0.01, 0.02, 0.03, 0.04, 0.05\} \quad \overline{B_1} = \{0.2, 0.4, 0.6, 0.8\}$$

If we apply DTW to align them, the resulting distance is 2. Now, let us align a new trace B_2, which contains the same

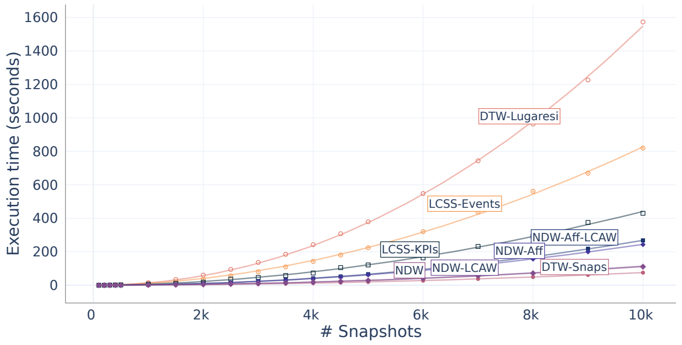


Fig. 19: Performance of the compared algorithms.

elements as B_1 but with one outlier value.

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

These two traces are less similar because the last value of the second trace is further apart than the last value of the previous trace, B_1. However, when normalization is applied, the distances between the other elements become smaller in comparison. Thus, the algorithm returns a distance of 1.05, which is smaller than the previous distance, meaning that B_2 is more similar to A than B_1, which is not the case.

Answer to RQ.1: Our approach is more effective than other considered methods of the existing literature in detecting and diagnosing delays and anomalies and for measuring the behavior similarity between traces of the same scenario. Unlike DTW, our proposal is unaffected by overstretching or overcondensing, effectively identifying discrepancies between traces. Furthermore, our approach is more general than the proposal by Lugaressi et al. [41], as it can align both sequences of events and raw traces. Finally, our approach also provides a concrete alignment of the trace, enabling the detection of inconsistencies and support in its diagnosis.

6.2.2 *Time complexity and scalability analysis (RQ.2-3)*

In this section, we will compare the time performance of different configurations of our algorithm with the state-of-the-art algorithms introduced in the previous section. To determine the time complexity, we conducted alignments using the same scenario, gradually increasing the number of snapshots from 100 to 10 000. If the scenario did not include enough snapshots, we simulated a restart by running through it in a loop. Next, we measured the execution time for each alignment and attempted to perform a regression analysis to estimate the time complexity and compare it across the algorithms. Each alignment with a set length was repeated five times. The complete analysis with all the measurements is available in [25].

To generate the graphics in Figure 19, we used polynomial regression analysis to determine each algorithm's time complexity. The regression results are available in Table 8.

Needleman-Wunsch configurations (RQ.2). We analyze the performance of the proposed algorithm, with and without the inclusion of LCAW and Affine Gap, i.e., four versions: NDW, NDW-LCAW, NDW-Aff, and NDW-Aff-LCAW. The regression results of the execution time of all configurations are shown in Table 8 and exhibit approximately quadratic complexity, consistent with their theoretical complexity as dynamic programming algorithms.

It is worth noting that including the Affine Gap in our algorithm results in a 30% decrease in performance since each matrix cell's computation requires the algorithm to consult and modify three matrices to decide whether to include a gap in either sequence. This means that if the user is not concerned about preventing gap alternation, they could choose not to use the Affine Gap to reduce computational costs.

Answer to RQ.2: The time complexity of our algorithm and all its configurations is quadratic with respect to the number of snapshots. The Affine Gap optimization requires the algorithm to consult and modify three matrices in each step, causing a 30% decrease in performance. The LCAW optimization worsens performance by less than 1% without the Affine Gap, but this impact increases to 6% when the Affine Gap is included.

TABLE 8: Regression analysis results for the alignment algorithms.

Algorithm name	Regression result	Time Complexity
NDW-Aff-LCAW	$y = 1.83 \cdot 10^{-6}x^{2.04}$	$O(n^{2.04}) \approx O(n^2)$
NDW-Aff	$y = 2.82 \cdot 10^{-6}x^{1.98}$	$O(n^{1.98}) \approx O(n^2)$
NDW-LCAW	$y = 1.40 \cdot 10^{-6}x^{1.98}$	$O(n^{1.98}) \approx O(n^2)$
NDW	$y = 1.40 \cdot 10^{-6}x^{1.97}$	$O(n^{1.97}) \approx O(n^2)$
DTW-Snaps	$y = 1.07 \cdot 10^{-6}x^{1.96}$	$O(n^{1.96}) \approx O(n^2)$
DTW-Lugaresi	$y = 9.92 \cdot 10^{-6}x^{2.05}$	$O(n^{2.05}) \approx O(n^2)$
LCSS-Events	$y = 26.0 \cdot 10^{-6}x^{1.88}$	$O(n^{1.88}) \approx O(n^2)$
LCSS-KPIs	$y = 15.7 \cdot 10^{-6}x^{1.86}$	$O(n^{1.86}) \approx O(n^2)$

Comparison with other proposals (RQ.3) We also compare the performance of our algorithm with the other approaches mentioned in the previous section: LCSS-KPIs, LCSS-Events, DTW-Snaps, and DTW-Lugaresi. The first two use the LCSS algorithm. DTW-Snaps applies the original DTW algorithm [42] to the traces, and DTW-Lugaresi is the original implementation from [41]. The latter uses the numpy.min function to find the minimum value for each cell, whose performance is poor for small sets of numbers. This is the main reason for this algorithm's overall performance.

All algorithms also demonstrate approximately quadratic complexity, as shown in Table 8. All versions of our algorithm perform better than the others by at least a 60% except for DTW-Snaps. However, we have seen that DTW may not be suitable for aligning execution traces because it aligns all events, ignores gaps and mismatches, and generates inaccurate distance metrics.

Answer to RQ.3: The time complexity of all dynamic programming algorithms considered here is quadratic, as expected. However, the average computational time for the implementations by Lugaresi et al. is greater than that of any of the possible configurations for our proposal. In turn, the original version of DTW [42] outperforms the different configurations of our algorithm in computation time, as it does not consider as many restrictions in the alignments as we do. However, as we have shown above, this leads to imprecise measurements when dealing with over-stretching or over-condensing.

6.3 Limitations and threats to validity

The proposal is subject to several limitations and other threats that may question its validity.

First, we have validated the proposed approach with four DTSs. They were selected from different application domains and with different characteristics, and our proposal has demonstrated to work similarly well with all three. However, we still need to perform further validation exercises with other types of systems, to check the generalizability of our results. For example, the threshold and parameter values used by our algorithm and fidelity indicators should be confirmed and validated with more industrial case studies.

Second, we assume that we can capture the behavior of digital and physical twins as discrete sequences of states (snapshots) at the appropriate level of abstraction and resolution. Although this is a common assumption for most systems, we need to investigate further whether this representation is always possible and sufficiently faithful, as in the case of highly dynamic systems, e.g., those involving fluids, complex interactions between parts of different natures, or human beings.

Third, our proposal has several parameters that must be tailored to each digital twin system fidelity measurement: *MAD*, affine gap values (P_{op}, P_{ex}), *LCAW*, and the fidelity indicator thresholds. Although we have proposed some default values for them, they depend on the actual system and, therefore, might require some tuning by end users.

We have also seen how our fidelity indicators were robust enough to absorb the inherently stochastic nature and random uncertainty of cyber-physical systems, as well as the variability of different runs of the same system. Or, at least to keep this variability under control. However, we need to validate this with further tests and more evidence from different systems.

Another limitation of our current proposal is that it requires comparing the complete traces to determine the degree of fidelity of the systems. In this sense, it represents a first step towards a more dynamic approach capable of assessing the fidelity of the two twins lively at runtime. This is part of our upcoming research work. Similarly, our proposal currently does not handle real-time synchronization of the two twins, something we also plan to investigate in future work.

7 Related work

7.1 Validation of DTs

Traditional validation techniques rely on statistical methods, which require large data sets and multiple independent replications due to the stochastic nature of the physical system [49]. Data for validation purposes can be obtained using two main types of simulations [50]. In a self-driven simulation, the model is executed by generating input data through sampling from probabilistic models. Typically, input data is represented by fitting probability distributions to the observed data. In a Trace-Driven Simulation (TDS) [51], the model is executed using as input the same trace data collected from the system.

The use of the trace alignment algorithm allows a more accurate comparison of traces that are expected to represent analogous behaviors, thus improving not only trace-based simulations but also the analysis with our proposal of any two traces describing the same synchronized behavior.

7.2 Similarity measurements

Measuring the level of similarity in a DT system is becoming relevant because of the importance of defining the DT at the minimum required level of fidelity to optimize computational costs [52]. Some existing proposals, such as [16], assess the required level of fidelity using a *validity frame*, which is defined along with a semi-automated methodology to establish the suitability of a given simulation. However, the majority of the proposals for assessing the similarity between two groups of simulations use dedicated measures. For example, the *Kullback-Liebler Divergence* (*KLD*) measure is used in [53] to calculate the difference between two probability distributions in order to assess the degree of variability between runs. The *Jensen-Shannon Distance* is used in [54]. This is a normalized symmetrical version of KLD that takes into account the uncertainty involved in non-deterministic executions, checking for a certain level of variability.

These approaches are similar to ours, although they focus on comparing probability distributions. Moreover, they do not consider alignments that may contain gaps and mismatches.

7.3 Trace analysis

Recent approaches compare data traces (or sequences) for validating in real-time digital models [7]. For example, *semantic matching rules* are defined in [9] to identify the differences between traces. The work [10] defines a set of operators for comparing execution traces of state machine models, and uses the Levenshtein distance [20] to measure their similarity.

In these works, the system traces are either synthesized from executions [11], [55] or inferred from the system specifications, e.g., from state machines [56]. Process mining techniques [57] are also used to infer trace models from event logs. They use similarity measures as part of their conformance-checking algorithms to assess the accuracy of the discovered models. For example, a variant of the Damerau–Levenshtein distance [58] is used in [59] to compare traces in the event log. Moreover, traces are paired using the Hungarian algorithm [60] to minimize their distance. The similarity between event logs is then assessed as the overall distance between paired traces.

These algorithms aim at global alignment of the traces, as opposed to those that look for local alignments, such as BLAST. We would use local alignments if we were looking for a specific behavior in the traces. For example, a floor change pattern in one of the elevator traces. However, we aim to compare the complete traces and therefore need a global alignment algorithm like NDW. Nevertheless, we have made use of some of the key optimizations used in BLAST to improve ours.

Similar to these works, we utilize traces obtained from model or system executions and apply a pairing algorithm. However, unlike these approaches, we focus on cyber-physical systems that inherently include uncertainties in behavior. Hence, we introduce the *MAD* parameter to allow for a certain tolerance when aligning snapshots. Furthermore, our algorithm accommodates the presence of gaps, enabling not only the detection of equivalent behavior but also the identification of absent behaviors and deviations.

7.4 Online validation techniques

Online validation techniques are also used in the domain of Digital Twin engineering [41]. The main difference with the previously mentioned works is that they compare the traces of the real system and of the digital twin to assess their validity, using Trace-Driven Simulation (TDS) [51]. The outcome of the validation process is expressed as a percentage of credibility rather than a binary variable assessing the correctness in absolute terms.

Initial works focus on continuous traces, using techniques such as harmonic analysis [61]. Compared to statistical techniques, these methods are able to achieve reliable results even when applied to relatively small data sets.

Dynamic Time Warping (DTW) is used in [62] to measure the similarity between a person and a robot arm during real-time imitation, comparing the sequences of traces retrieved from its execution. Similarly, works such as [10], [55] adapt algorithms for analyzing sequences of characters to analyze traces, considering each of the individual measurements as a character. A similar algorithm is proposed in [41] and successfully applied on a simple single server system. Likewise, in [39] we proposed the first version of our algorithm, using a modified version of the Needleman–Wunsch algorithm to align the traces and compute some distances. A recent work [7] is probably the closest to our proposal, as presented in section 6.2.1. They use a variant of the Dynamic Time Warping (DTW) algorithm for aligning the traces, enriched with a comparison function to identify similar events. Although these works [7], [39] align the traces before defining distance measures, they do not make use of some of the key optimizations provided by the BLAST algorithm, such as the possibility of defining strategies for deciding how to deal with sequences of gaps or for masking low-complexity regions, nor use the *MAD* threshold or define any of our fidelity indicators, which have proven instrumental for the effective assessment of the fidelity of the two twins. Our experiments have demonstrated the importance of taking these optimizations into account to avoid incorrect adjustments in the alignments or wrong measurements of the distance between the traces.

8 Conclusions and future work

In this work, we propose both a method and a set of metrics to assess the fidelity of the behavior of two twins, based on the comparison of the sequences of the states they reach during execution. To achieve this, we propose a discrete definition of the twins' behavior as a sequence of snapshots. We adapted a character alignment algorithm from Bioinformatics and defined a comparison function to assess the similarity between pairs of snapshots. After aligning the two traces using the algorithm, we obtain a set of metrics (%MS, FD, and ED) that are able to evaluate the fidelity between them. Additionally, we propose a guide to interpret these metrics and determine the degree of fidelity of the twin.

This approach can be applied not only to compare the behavior of two systems in the context of DTSs, but also to assess the degree of similarity of the behavior of any two systems. These systems can be both physical (such as two instances of the same machine, in which we want to identify behavioral deviations), digital (evaluating the fidelity of two simulations designed with different levels of detail, comparing a validated one with one under development), or one physical and one digital, as in the case of DTSs. The algorithm is applicable in any context where two systems are expected to exhibit twinned behavior.

The proposal is subject to several limitations that highlight the need for further research. For example, the algorithm, its parameters and the threshold values defined here should be confirmed and validated with more industrial case studies. Additionally, we intend to compare our algorithm with other state-of-the-art methods, including various implementations of DTW. This will help us identify potential limitations in our approach and explore techniques to mitigate them. Furthermore, right now the proposal is able to identify deviations between traces, but is not able to identify the specific cause of the anomalous behavior of the model (or the system). A significant advance would be to study methods and tools to automatically detect the reason why the model is invalid. In addition, incorporating uncertainty into our algorithms is a matter of future research, for which we plan to use our recent results of representing attribute values as random variables instead of the traditional crisp values of standard datatypes [24], [63]. Another line of research will focus on runtime validation of traces by defining time windows or event batches [7], [61]. This is essential to enable the dynamic and continuous validation of the two twins throughout the entire lifetime of the DTS.

In conclusion, our proposal aims to address a gap in the current state of the art, recognized by both industry and academia, which calls for the need for tools that enable the Verification and Validation (V&V) of DTSs [64]. Specifically, our proposal focuses on assessing the consistency of behavior between the DT and PT. In this work, we refer to this consistency as *Fidelity*, which is a crucial requirement for ensuring the effectiveness of DTSs, and that involves achieving identical (or *sufficiently similar*) behavior. Our approach presents a solution that facilitates the analysis of all these aspects and enables reasoning about the precise causes behind discrepancies.

Verifiability

For the sake of verifiability, our prototype as well as all artifacts of the experiments are available online [27], [28].

Acknowledgments

This work was partially supported by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under projects SoCUS [TED2021-130523B-I00] and IPSCA [PID2021-125527NB-I00], and by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and by the Austrian Science Fund (P28519-N31, P 30525-N31).

References

[1] Digital Twin Consortium, "Glossary of digital twins," https://www.digitaltwinconsortium.org/glossary/index.htm, 2021, accessed: October 13, 2024.

[2] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, and A. Wortmann, "A cross-domain systematic mapping study on software engineering for digital twins," *J. Syst. Softw.*, vol. 193, p. 111361, 2022.

[3] P. Muñoz, J. Troya, and A. Vallecillo, "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*, ser. CEUR Workshop Proceedings, vol. 3620. CEUR-WS.org, 2023. [Online]. Available: https://ceur-ws.org/Vol-3620/mess23_paper01.pdf

[4] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *Systems Modelling and Management*. Cham: Springer International Publishing, 2020, pp. 43–54.

[5] ASME V&V 20-2009, *Standard for Verification and Validation in Computational Solid Mechanics*, 2009, American Society for Mechanical Engineers (New York, NY).

[6] R. G. Sargent, "Verification and validation of simulation models," *J. Simulation*, vol. 7, no. 1, pp. 12–24, 2013.

[7] G. Lugaresi, S. Gangemi, G. Gazzoni, and A. Matta, "Online validation of digital twins for manufacturing systems," *Comput. Ind.*, vol. 150, p. 103942, 2023.

[8] D. C. Gross, "Report from the fidelity implementation study group," in *Simulation Interoperability Workshop*. Orlando, FL, USA: Simulation Interoperability and Standards Organization, 1999, paper 99S-SIW-167.

[9] P. Langer, T. Mayerhofer, and G. Kappel, "Semantic model differencing utilizing behavioral semantics specifications," in *Model-Driven Engineering Languages and Systems*. Cham: Springer International Publishing, 2014, pp. 116–132.

[10] D. Leroy, E. Bousse, A. Megna, B. Combemale, and M. Wimmer, "Trace comprehension operators for executable DSLs," in *Modelling Foundations and Applications - 14th European Conference, ECMFA@STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings*, ser. Lecture Notes in Computer Science, vol. 10890. Cham: Springer, 2018, pp. 293–310.

[11] R. P. J. C. Bose and W. M. P. van der Aalst, "Process diagnostics using trace alignment: Opportunities, issues, and challenges," *Inf. Syst.*, vol. 37, no. 2, pp. 117–141, 2012.

[12] V. Zhidchenko, I. Malysheva, H. Handroos, and A. Kovartsev, "Faster than real-time simulation of mobile crane dynamics using digital twin concept," *Journal of Physics: Conference Series*, vol. 1096, p. 012071, 2018.

[13] S. F. Altschul, B. W. Erickson, and H. Leung, *Local Alignment (with Affine Gap Weights)*. Boston, MA: Springer US, 2008, pp. 459–461.

[14] A. Arrieta, "Multi-fidelity digital twins: a means for better cyber-physical systems testing?" *CoRR*, vol. abs/2101.05697, 2021.

[15] M. Gogolla, J. Bohling, and M. Richters, "Validating UML and OCL Models in USE by Automatic Snapshot Generation," *SoSyM*, vol. 4, no. 4, pp. 386–398, 2005.

[16] B. V. Acker, P. D. Meulenaere, J. Denil, Y. Durodie, A. V. Bellinghen, and K. Vanstechelman, "Valid (re-)use of models-of-the-physics in cyber-physical systems using validity frames," in *2019 Spring Simulation Conference (SpringSim)*, 2019, pp. 1–12.

[17] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[18] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[19] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990. [Online]. Available: https://blast.ncbi.nlm.nih.gov/

[20] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[21] U. Mori, A. Mendiburu, and J. A. Lozano, "Distance measures for time series in R: the tsdist package," *R Journal*, vol. 8, no. 2, p. 451, 2016.

[22] Peters Research, "Elevate software," 2023. [Online]. Available: https://peters-research.com/index.php/elevate/

[23] P. Muñoz, J. Troya, M. Wimmer, and A. Vallecillo, "Measuring the fidelity of a physical and a digital twin using trace alignments: Elevator technical report," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/ateneasearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_Elevator.pdf

[24] M. F. Bertoa, L. Burgueño, N. Moreno, and A. Vallecillo, "Incorporating measurement uncertainty into OCL/UML primitive datatypes," *Softw. Syst. Model.*, vol. 19, no. 5, pp. 1163–1189, 2020.

[25] P. Muñoz, J. Troya, M. Wimmer, and A. Vallecillo, "Measuring the fidelity of a physical and a digital twin using trace alignments: General concepts," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/ateneasearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_General_Concepts.pdf

[26] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[27] P. Muñoz, J. Troya, M. Wimmer, and A. Vallecillo, "Measuring the fidelity of a physical and a digital twin using trace alignments – Git repository," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/ateneasearchgroup/fidelity-measure-for-dts

[28] ——, "Measuring the fidelity of a physical and a digital twin using trace alignments – Zenodo permanent link," 2023, accessed: October 13, 2024. [Online]. Available: https://doi.org/10.5281/zenodo.12527797

[29] D. A. Snow, Ed., *Plant Engineer's Reference Book*, 2nd ed. Oxford: Elsevier, 2003.

[30] S. H. Choi, S. J. Lee, and T. G. Kim, "Multi-fidelity modeling & simulation methodology for simulation speed up," in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 139–150.

[31] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Sci. Comput. Program.*, vol. 69, no. 1, pp. 27–34, 2007, Special issue on Experimental Software and Toolkits.

[32] F. Büttner and M. Gogolla, "On OCL-based imperative languages," *Sci. Comput. Program.*, vol. 92, pp. 162–178, 2014.

[33] H. Feng, C. Gomes, C. Thule, K. Lausdahl, M. Sandberg, and P. G. Larsen, "The incubator case study for digital twin engineering," 2021. [Online]. Available: https://arxiv.org/abs/2102.10390

[34] "Example digital twin: The incubator," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/INTO-CPS-Association/example_digital-twin_incubator

[35] Components101, "DHT22 Temperature and Humidity Sensors specification," 2018, accessed: October 13, 2024. [Online]. Available: https://components101.com/sensors/dht22-pinout-specs-datasheet

[36] E. A. Lee and M. Sirjani, "What good are models?" in *Formal Aspects of Component Software*. Cham: Springer International Publishing, 2018, pp. 3–31.

[37] Tinkerkit, "Arduino Tinkerkit Braccio Robot," 2021, accessed: October 13, 2024. [Online]. Available: https://store.arduino.cc/products/tinkerkit-braccio-robot

[38] M. U. Masood and M. Haghshenas-Jaryani, "A study on the feasibility of robotic harvesting for chile pepper," *Robotics*, vol. 10, no. 3, p. 94, 2021.

[39] P. Muñoz, M. Wimmer, J. Troya, and A. Vallecillo, "Using trace alignments for measuring the similarity between a physical and its digital twin," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022*. ACM, 2022, pp. 503–510.

[40] P. Muñoz, J. Troya, M. Wimmer, and A. Vallecillo, "Measuring the fidelity of a physical and a digital twin using trace alignments: Nxt lego mindstorms car," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/ateneasearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_NXT_Car.pdf

[41] G. Lugaresi, S. Gangemi, G. Gazzoni, and A. Matta, "Online validation of simulation-based digital twins exploiting time series analysis," in *2022 Winter Simulation Conference (WSC)*, 2022, pp. 2912–2923.

[42] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Proc. of the 7th International*

Congress on Acoustics, vol. 3. Budapest: Akadémiai Kiadó, 1971, pp. 65–69.

[43] P. Muñoz, J. Troya, M. Wimmer, and A. Vallecillo, "Measuring the fidelity of a physical and a digital twin using trace alignments: Incubator technical report," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_Incubator.pdf

[44] ——, "Measuring the fidelity of a physical and a digital twin using trace alignments: Robotic arm technical report," 2023, accessed: October 13, 2024. [Online]. Available: https://github.com/atenearesearchgroup/fidelity-measure-for-dts/blob/main/docs/Technical_Report_Robotic_Arm.pdf

[45] H. Li, J. Liu, Z. Yang, R. W. Liu, K. Wu, and Y. Wan, "Adaptively constrained dynamic time warping for time series classification and clustering," *Information Sciences*, vol. 534, pp. 97–116, September 2020.

[46] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[47] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[48] A. W.-C. Fu, E. Keogh, L. Y. H. Lau, C. A. Ratanamahatana, and R. C.-W. Wong, "Scaling and time warping in time series querying," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, p. 649–660.

[49] O. Balci, "Validation, verification, and testing techniques throughout the life cycle of a simulation study," *Ann. Oper. Res.*, vol. 53, no. 1, pp. 121–173, 1994.

[50] J. Banks, *Handbook of simulation - principles, methodology, advances, applications, and practice*. Wiley, 1998.

[51] T. Marquardt, C. Cleophas, and L. Morgan, "Indolence is fatal: Research opportunities in designing digital shadows and twins for decision support," in *2021 Winter Simulation Conference (WSC)*, 2021, pp. 1–11.

[52] J. Ahlgren, K. Bojarczuk, S. Drossopoulou, I. Dvortsova, J. George, N. Gucevska, M. Harman, M. Lomeli, S. M. M. Lucas, E. Meijer, S. Omohundro, R. Rojas, S. Sapora, and N. Zhou, "Facebook's cyber–cyber and cyber–physical digital twins," in *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–9. [Online]. Available: https://doi.org/10.1145/3463274.3463275

[53] K. Worden, E. J. Cross, R. J. Barthorpe, D. J. Wagg, and P. Gardner, "On digital twins, mirrors, and virtualizations: Frameworks for model verification and validation," *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering*, vol. 6, no. 3, p. 030902, 2020.

[54] K. Bojarczuk, N. Gucevska, S. Lucas, I. Dvortsova, M. Harman, E. Meijer, S. Sapora, J. George, M. Lomeli, and R. Rojas, "Measurement challenges for cyber cyber digital twins: Experiences from the deployment of facebook's ww simulation system," in *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '21. New York, NY, USA: Association for Computing Machinery, 2021.

[55] S. Alimadadi, A. Mesbah, and K. Pattabiraman, "Inferring hierarchical motifs from execution traces," ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 776–787.

[56] S. Wolny, A. Mazak, M. Wimmer, and C. Huemer, "Model-driven runtime state identification," in *Proc. of EMISA'19*, ser. LNI, vol. P-304. Bonn: Gesellschaft für Informatik e.V., 2019, pp. 29–44.

[57] W. M. P. van der Aalst, *Process Mining – Data Science in Action*, 2nd ed. Berlin: Springer, 2016.

[58] G. V. Bard, "Spelling-error tolerant, order-independent passphrases via the damerau-levenshtein string-edit distance metric," in *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers - Volume 68*, ser. ACSW '07. AUS: Australian Computer Society, Inc., 2007, p. 117–124.

[59] M. Camargo, M. Dumas, and O. González-Rojas, "Automated discovery of business process simulation models from event logs," *Decision Support Systems*, vol. 134, p. 113284, 2020.

[60] H. W. Kuhn, "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*. Berlin, Heidelberg: Springer, 2010, pp. 29–47.

[61] G. Lugaresi, G. Aglio, F. Folgheraiter, and A. Matta, "Real-time validation of digital models for manufacturing systems: a novel signal-processing-based approach," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019, pp. 450–455.

[62] L. Gong, B. Chen, W. Xu, C. Liu, X. Li, Z. Zhao, and L. Zhao, "Motion similarity evaluation between human and a tri-co robot during real-time imitation with a trajectory dynamic time warping model," *Sensors*, vol. 22, no. 5, p. 1968, 2022.

[63] J.-M. Jézéquel and A. Vallecillo, "Uncertainty-aware simulation of adaptive systems," *ACM Trans. Model. Comput. Simul.*, vol. 33, no. 3, pp. 8:1–8:19, 2023.

[64] H. M. Muctadir, D. A. M. Negrin, R. Gunasekaran, L. Cleophas, M. van den Brand, and B. R. Haverkort, "Current trends in digital twin development, maintenance, and operation: An interview study," *CoRR*, vol. abs/2306.10085, Jun. 2023.

Paula Muñoz is a PhD candidate at the University of Málaga. She graduated in Software Engineering from the University of Málaga in June 2019. Her research focuses on precisely specifying and testing software systems using models, as well as the validation of Digital Twins. You can contact her at paulam@uma.es.

Manuel Wimmer is a full professor leading the Department of Business Informatics - Software Engineering at the Johannes Kepler University Linz. His research interests comprise foundations of software engineering techniques and their application in domains such as tool interoperability, software modernization, as well as cyber-physical systems. For more information, please visit https://www.se.jku.at/manuel-wimmer/.

Javier Troya is Associate Professor at the Universidad de Málaga, Spain. Before, he was Assistant Professor at the Universidad de Sevilla, Spain (2016-2020), and a post-doctoral researcher in the TU Wien, Austria (2013-2015). He obtained his International PhD with honors at the Universidad de Málaga, Spain (2013). His current research interests include MDE, Software Testing and Digital Twins. For more information, please visit https://javiertroyauma.github.io/.

Antonio Vallecillo is a retired full professor of software engineering at the University of Málaga, working on systems modeling and analysis. His research interests include open-distributed processing, model-based software engineering, and software quality. For more information about his publications, research projects, and activities, please visit http://www.lcc.uma.es/~av.