# Using Trace Alignments for Measuring the Similarity between a Physical and its Digital Twin

Paula Muñoz
paulam@uma.es
ITIS Software. Universidad de Málaga
Málaga, Spain

Manuel Wimmer
manuel.wimmer@jku.at
CDL-MINT, Johannes Kepler University
Linz, Austria

Javier Troya
jtroya@uma.es
ITIS Software. Universidad de Málaga
Málaga, Spain

Antonio Vallecillo
av@uma.es
ITIS Software. Universidad de Málaga
Málaga, Spain

## ABSTRACT

A common problem in the development of digital twin systems is the validation that the behavior of both twins, the physical and the digital, is the same, or at least similar enough given the requirements of the digital twin system. In this paper, we propose a method for the alignment of the traces of both twins. Traces are sequences of snapshots that capture the progressive states of each entity. Our approach is based on a bioinformatic algorithm that we adapt and use for the alignment of snapshots. Additionally, we include a set of measures to evaluate the quality of these alignments and reason about the level of fidelity of the digital twin system. Two case studies are used to demonstrate our proposal and evaluate its accuracy and effectiveness.

## CCS CONCEPTS

• **Software and its engineering → Operational analysis**.

## KEYWORDS

Digital twins, Trace analysis, Trace alignment, Conformance testing

**ACM Reference Format:**
Paula Muñoz, Manuel Wimmer, Javier Troya, and Antonio Vallecillo. 2022. Using Trace Alignments for Measuring the Similarity between a Physical and its Digital Twin. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3550356.3563135

## 1 INTRODUCTION

A *Digital Twin* (DT) is a digital representation of an actual system, service or product (the *Physical Twin* (PT)), synchronized at a specified frequency and fidelity [8]. DTs are usually employed for tasks such as: providing self-adaptive mechanisms for the PT; making predictions to enable calibration; optimizing its performance by adjusting its configuration parameters; or providing what-if analysis for decision support and alerts [15].

To implement these tasks, the DT's behavior must be *faithful* to the PT's, i.e., the behavior of the DT must emulate with sufficient precision the behavior of its PT. In case this requirement is not met, the DT's recommendations might lead to wrong decisions being made or problems during execution time. For example, let us assume we have a DT of a robot arm that implements self-adaptive behavior, enabling the change of trajectories to avoid collisions with unexpected obstacles. If the DT movements are not sufficiently faithful with respect to the PT, the orders provided may be inaccurate and lead to damages to the PT.

Although DTs are convenient to test what-if scenarios or provide optimizations, it is imperative to measure the fidelity between both twins to assess if the results obtained are good enough for its required purpose. It is essential to keep in mind that it is impossible to develop a DT that emulates the physical system with complete accuracy since we will never be able to realize a perfect model [13]. Simulation of high-fidelity DTs is quite costly because it involves the simulation of physical phenomena and the interaction of a great number of components.

To tackle this challenge, some authors, e.g., [1, 2, 5, 10, 24, 25], propose the use of a hierarchy of DTs with different levels of fidelity, some of them being lower fidelity DTs with a very specific goal. These lighter DTs provide abstractions of the PT by selecting the properties of interest and defining them with the required resolution. This reduces the level of fidelity with respect to the original system in order to enable the optimization of resource consumption and reduce response times [25]. However, there are only a few systematic approaches to automatically measure the fidelity of these hierarchies of models at different abstraction and resolution levels [5] to validate whether they are fit for their purpose.

In our proposal, we provide a systematic method to align the execution traces of a DT and a PT, and based on this alignment, we propose a set of similarity measures to reason about the quality of the alignment. These measurements allow reasoning about the level of fidelity of the DT with respect to the PT to assess if it is enough for its given purpose. This approach not only provides a set of similarity measures but also enables the location of discrepancies between the PT and the DT. Our proposal is based on a bioinformatics algorithm named *Needleman-Wunsch* (*NDW*) [20] which is used for global alignment of character sequences in order to find similarities in the amino acid sequences of two proteins. We have adapted this algorithm to enable the comparison between executions traces composed of sequences of *snapshots* [9]. Each snapshot defines the system's state at a given instant in time, specifying the objects, the relationships between them, and their specific values. Our adapted algorithm aligns the traces by matching the snapshots that are considered to be equivalent, and we define measures to analyze the results of the alignment and the fidelity of the system.

This paper is structured as follows. In Section 2, we provide a general background of all the concepts on which we base our proposal, including the NDW algorithm and an introduction to similarity measurement. This section also presents the running example used to illustrate the proposal. In Section 3, we explain our adaptation to the NDW algorithm and present the results of our analysis using the running example and an additional case study of a robot arm. Finally, in Section 4, we present the conclusions and discuss some future work.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Fidelity, Abstraction and Resolution

This work unfolds around the concept of fidelity. Fidelity is defined as "the degree to which a model reproduces the actual state and behavior of a system in a measurable way" [11]. It determines how closely a model realistically represents the actual system. In our case, we aim to determine the fidelity between the DT (the model) and the PT (the actual system). Some works define the interrelationship between the concepts of abstraction, resolution, and fidelity [16].

Abstraction is defined as "the process of selecting the essential aspects of a system to be represented in a model or simulation while ignoring those aspects that are not relevant to its purpose" [11], while resolution has been defined as "the degree of detail used to represent certain aspects of the real world in a model" [16]. The purpose of resolution is to determine *how* precise the elements of the system will be modeled, whereas abstraction determines *which* elements will be included in the model.

From these definitions, we can naively think that the greater the abstraction, the lower the fidelity since we would be removing elements from the model. Analogously, we can also think that the lower the resolution, the lower the fidelity because we would be reducing the level of detail of its elements. This is not always the case as presented in [16] since it depends on the influence of such changes on the scope of the model.

In our work, we use traces to define the behavior of the system over time. These traces are composed by sequences of *snapshots* [9], i.e., a set of objects, a set of links between them, and the specific values of the attributes of these objects at a specific moment in time. These snapshots are selected sets of elements abstracted from the PT using an abstraction function.

Given two models $A$ and $C$ of a system $S$ at different levels of abstraction (e.g., $A$ is more abstract than $C$), the *abstraction function* defined between $C$ and $A$ is a function that maps the elements of the more concrete model $C$ into elements of the more abstract model $A$. It explains how to interpret each element of the concrete model that is relevant to the user in terms of elements in the abstract model. Abstraction functions are useful for bringing models — or specific elements of models — to the same level of abstraction to enable comparison between them. We use these abstraction functions to compare execution traces defined at different levels of abstraction, in case we are dealing with a hierarchy of DTs.

### 2.2 Similarity Measurement

We record the different states of the DT and the PT reached during execution as sequences of snapshots, which could be interpreted as trajectories, time-series, or even probability distributions if we consider a set of executions. In this section, we analyze some of the proposals in the literature to measure similarity both in simulation executions in general and in some concrete DT use cases.

In recent works, the question of measuring the level of similarity in a DT system arises [2], emphasizing the importance of defining the DT at the minimum required level of fidelity to optimize computational costs. Some existing proposals, such as [1], assess the required level of fidelity using a *validity frame*. This validity frame is defined along with a semi-automated methodology to establish the suitability of a given simulation. In other works, such as [24], they propose to measure the similarity between two groups of simulations using the *Kullback-Liebler Divergence* (*KLD*). This measure calculates the difference between two probability distributions in order to assess the degree of variability between runs. Other proposal [5] uses the *Jensen-Shannon Distance*, which 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.

Other methods for assessing similarity use different measure distances. There are two main groups of measure distances: the *lock-step measures*, which compare the $i$-th point of one time series with the $i$-th point of the other; and the *elastic measures*, which allow one-to-many points or one-to-none points matching [17].

*Lock-step measures.* Let us suppose that we have two traces $A = \{a_1, .., a_n\}$ and $B = \{b_1, .., b_n\}$ that have been already aligned, i.e, they have the same number of points ($n$) and the elements $a_i$ and $b_i$ describe the state of each system at step $i$. Then, assuming that $d(p, q)$ is a distance measure between a pair of points $p$ and $q$ (e.g, the Euclidean or the Manhattan distance), we can compute the average distance between the two traces, $d(A, B)$, as well as its sample standard deviation, $s(A, B)$, as follows:

$$d(A,B) = \frac{1}{n}\sum_{i=1}^{n} d(a_i, b_i) \tag{1}$$

$$s(A,B) = \sqrt{\frac{1}{n-1}\left(\sum_{i=1}^{n} d(a_i,b_i)^2 - n\cdot d(A,B)^2\right)} \tag{2}$$

These two numbers together provide a measure of the distance between the two traces. For example, in [25], they developed two versions of a DT at different levels of fidelity. To measure their similarity, they manually aligned the execution traces and used a lock-step measure based on the *Manhattan*'s distance.

*Elastic measures.* Sometimes, a more flexible alignment between the traces is desirable, especially when one-to-many or one-to-none point matchings need to be considered. One of the most representative measures is the Fréchet distance.

The ***Fréchet distance*** $F(A, B)$ between two given curves $A$ and $B$ is defined as the infimum over all reparametrizations $\alpha$ and $\beta$ of the maximum over all $t \in [0,1]$ of the distance between $A(\alpha(t))$ and $B(\beta(t))$. It is generally explained through this example: Imagine a person who walks from one end of $A$ to the other end, being their position $A(\alpha(t))$; likewise, a dog walks from one end of $B$ to the other end, being its position $B(\beta(t))$, 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 $d(A(\alpha(t)), B(\beta(t)))$ is a distance measure such as the Euclidean or the Manhattan's distance, the Fréchet distance is:

$$F(A,B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \left\{ d(A(\alpha(t)), B(\beta(t))) \right\} \tag{3}$$

## 2.3 Trace Analysis

In our work, we not only aim at measuring the similarity between the traces but also to provide the alignment between the snapshots, in order to check the possible incompatibilities between the executions and to provide a better diagnosis of possible execution errors.

In the literature, there are works that perform trace analysis to obtain higher level information about the system behavior or perform validation by comparing traces of equivalent systems. In [23], they obtain the states through which the system transitions from its state machine. In [3], they infer high-level tasks analyzing the execution of web services to improve the developer's comprehension of such processes. In [6], they propose the use of these techniques to identify the common and frequent behavior to distinguish it from exceptional behavior, potentially helping in validation. In the field of comparison between traces, works such as [12] define *semantic matching rules* which enable us to tell the difference between traces. In [14], they define operations for processing traces. They propose the *Levenshtein* distance to measure similarity. The Levenshtein distance measures the number of operations to transform one string into another and describes the required transformations. They adapt this measurement to use it for trace analysis. In [10], they use Dynamic Time Warping to measure the similarity between a person and a robot arm during real-time imitation, comparing the sequences of traces retrieved from its execution. In [3, 14], they adapt algorithms meant for analyzing sequences of characters to analyze traces, considering each of the instant measurements as a character.

Our work focuses on this latter aspect: validating the conformance of one process to another; in our case the execution of the DT against that of the PT. We adapt an algorithm for analyzing sequences of characters to analyze traces. These algorithms usually belong to the field of bioinformatics, since they usually need to perform alignments to compare biological sequences. There are two main types of sequence alignment algorithms depending on whether they use local or global alignment.

Local alignment algorithms are used for comparing two dissimilar sequences in which we expect to find some regions of similarities between the two sequences. An example of a local alignment algorithm is the *Smith-Waterman* [22] algorithm based on dynamic programming. In contrast, global alignment algorithms attempt to align all the elements of the sequences, so they are suitable for cases in which the sequences are similar. An example of a global alignment algorithm is the *Needleman-Wunsch* algorithm (*NDW*) [20]. It is also based on dynamic programming and it is one of the most simple but powerful algorithms for sequence alignment and the one that will be adapted to perform alignments in our proposal.

There are further attempts in recent years to improve the performance of the aforementioned algorithms for aligning sequences. One of these algorithms, *BLAST* [4], provides an improved version of the Smith-Waterman algorithm optimized with the use of heuristics. In [7], they provide an optimization of the dynamic programming approach used in NDW algorithm reducing its time and space complexity from $O(n^2)$ to $O(n^2/\log(n))$. Since this is a first attempt at our proposal, we will be using the original version of the NDW algorithm. However, we plan to check its performance on larger data sets and make use of the optimized version if needed.

The ***Needleman-Wunsch algorithm*** [20] is a global alignment algorithm based on dynamic programming to find the optimal alignment between two sequences of characters. It divides the problem of comparing two full sequences into a series of small problems (the comparison of partial sets of the full sequence) to find the optimal solution to the full problem. We enumerate below the different steps followed by this algorithm to reach the optimal solution:

(1) **Set a scoring system.** In general, given two sequences there is no unique alignment between them. Thus, we need a scoring system to evaluate the quality of the alignment depending on our intentions. In the NDW algorithm there are three main outcomes when comparing two characters $C_a$ and $C_b$, belonging to sequence $a$ and $b$ respectively:
   (a) **Match.** The two characters at the current index are the same.
   (b) **Gap.** The characters are different and we decide to leave a gap in the alignment and not choose a character from any of the sequences.
   (c) **InsDel.** The characters are different and we include a character from either sequence $a$ (insertion) or from sequence $b$ (deletion).

   The scoring system will assign a specific score for each of these situations. There are two main types of scoring schemes:
   (a) **Basic scoring scheme.** We assign a score for each of the situations. For example, +1 for a Match, -1 for Mismatch, and 0 for InsDel. This way, we will prioritize alignments that include a lesser number of gaps.
   (b) **Frequency scoring scheme.** We assign a different set of these three scores for each of the possible comparisons between the different available characters. This generates an input matrix that the algorithm takes for processing the alignment. It is common in bioinformatics use cases in which certain proteins are more common than others.

(2) **Fill the similarity matrix.** Once we have selected the scoring scheme, we fill the similarity matrix in which we compare each of the characters and try to optimize the score, choosing between the three main outcomes. The pseudocode for this process is shown in Algorithm 1. In this algorithm, $m$ is the similarity matrix, $t_a$ and $t_b$ are the traces to align and the scores are the ones of the selected scoring system. The *scoreSubstitution* is evaluated after comparing the characters and determining whether it is a match, line 12. If this is

**Algorithm 1** Calculation of the similarity matrix for the Needleman-Wunsch algorithm [20]

```
1: $m[0][0] \leftarrow 0$
2: $c \leftarrow 1$
3: while $c < m[0].size$ do        ▷ Fill the first row of the matrix
4:     $m[0][c] \leftarrow m[0][c-1] + scoreInsertion$
5:     $c \leftarrow c + 1$
6: end while
7: $r \leftarrow 1$
8: while $r < m.size$ do
9:     $m[r][0] \leftarrow m[r-1][0] + scoreDeletion$ ▷ Fill first column
10:    while $c < m[0].size$ do
11:        $ins \leftarrow m[r][c-1] + scoreIns$
12:        $sub \leftarrow m[r-1][c-1] + scoreSub(t_a[r-1], t_b[c-1])$
13:        $del \leftarrow m[r-1][c] + scoreDel$
14:        $m[r][c] \leftarrow max(ins, sub, del)$
15:        $c \leftarrow c + 1$
16:    end while
17: end while
```

the case, it assigns the match score. Otherwise, it assigns a gap score, line 14.

(3) **Build the alignment.** Once the matrix is filled, we need to trace it back from the bottom right, by choosing the optimal scores (highest or lowest depending on whether we are maximizing or minimizing). This leads us to the optimal alignment given our sequences and the selected score system.

## 2.4 Running Example

In previous works [18, 21], we presented the example of the DT of a Lego Mindstorms NXJ Car to demonstrate our proposals. The NXJ is a small car-like vehicle with sensors. It can move, detect obstacles, and interact with its environment in different ways by using its sensors, e.g., following a path defined by a colored line or avoiding obstacles. The NXJ uses Bluetooth to exchange data and commands, e.g., information about its current state (such as position, speed, etc.) as well as other sensor readings about the environment, that will compose the snapshots.

The NXJ has a pose provider for determining its planar coordinates and the angle it is heading to. To move, it has an engine that enables forward movement and allows it to rotate its direction at a certain angle. Furthermore, it is equipped with three types of sensors: (*i*) the ultrasonic sensor detects the distance to any object which is in front of the car, (*ii*) the light sensor reads the color of the ground beneath the car, and (*iii*) two touch sensors are utilized as a bumper to report if the car has collided with an obstacle.

In Figure 1, we can see the attributes that we will take from this system as properties of interest for our analysis. Every snapshot includes a `timestamp` to determine the exact moment at which the system was at such state, as well as an `executionId` to identify it. It coincides with the first timestamp of the execution. The `twinId` attribute is a unique identifier for the twin for storing its snapshots in the data lake, in case there is more than one car, for example. The rest of the attributes describe the state of the car: its position

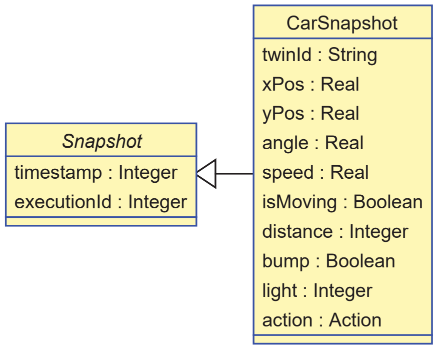

**Figure 1: A UML class model representing the information which is exchanged with the system.**

(`xPos`, `yPos`); the angle the car is heading to (`angle`); its `speed`; whether it is moving or not (`isMoving`); the `distance` in front of it as detected by the ultrasonic sensor; if the car has collided or not, as detected by the touch sensor (`bump`); the color of the floor beneath it, as detected by the light sensor (`light`); and the `action` it is performing: going `#Forward` or `#Rotate`.

Figure 2 shows an excerpt of two of the traces used in our analysis, one from the PT (above) and one from the DT (below). This will be the information used by our approach to match the sequences of snapshots of the DT to the PT. These snapshots are composed using an abstraction function which maps the selected attributes to the corresponding properties of the actual system. This enables the comparison between snapshots defined at different abstraction levels, since we could find the equivalence between their elements by checking the mapping provided by the abstraction function. For example, let us suppose we have an low-fidelity DT which only calculates the position of the car, including only the attributes `xPos`, `yPos` and `angle`, named `x`, `y` and `heading`. If we want to perform an alignment with the snapshots of Figure 2, we would need to check the abstraction function for the equivalence between these given attributes, to know that we need to compare `xPos` with `x`, for example. In the following sections, we will use a set of synthetic traces generated from this example to illustrate our proposal and discuss some initial results.

## 3 APPROACH

In this section, we first show how we adapt the NDW algorithm for trace alignment, and subsequently, we apply the resulting approach for our running example. All the algorithms and materials used for the realization of our proposal are available in our GitHub repository [19].

### 3.1 Leveraging the Needleman-Wunsch Algorithm for DTs Trace Alignment

To perform the global alignment between execution traces, we selected the NDW algorithm and adapted it to assess the equivalence between snapshots. In contrast to original approaches based on comparing single characters, snapshots are composed of both discrete and continuous attributes, which makes it challenging to check this equivalence. Additionally, to assess the equivalence between the

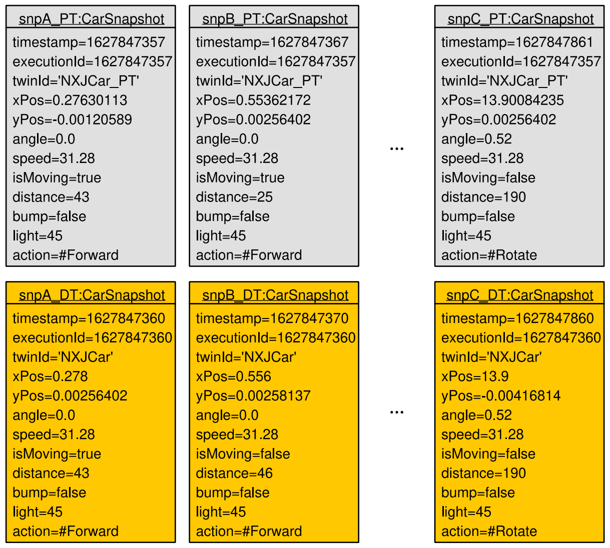

**Figure 2: A UML object model of Lego Car snapshots. PT snapshots (above), DT snapshots (below).**

**Algorithm 2** Equivalence function between two snapshots

```
1: $i \leftarrow 0$
2: while $equals$ and $i < snp_A.size$ do
3:     if $snp_A[i]$ is numerical then
4:         $equals \leftarrow (|snp_A[i] - snp_B[i]| < tolerance)$
5:     else
6:         $equals \leftarrow (snp_A[i] = snp_B[i])$
7:     end if
8:     $i \leftarrow i + 1$
9: end while
```

values of the DT and the PT, we cannot suppose that the values should be identical, for different reasons. First, the snapshots are taken periodically, and we cannot assume that are taken simultaneously for both systems. Second, the PT is usually a cyber-physical system that deals with physical phenomena, and therefore its measurements may include some uncertainty. Third, we may have to deal with computation errors derived from simulation — especially for floating point numbers, which always required a certain level of *tolerance* in their comparison.

Under these assumptions, it is clear that the equivalence comparison must consider some kind of accuracy in order to address some of these situations. The specific algorithm to assess this equivalence is shown in Algorithm 2, where $snp_A$ and $snp_B$ are the snapshots to compare. They include a list of attributes. We also introduce a parameter called `tolerance`, line 4, that determines the maximum difference between numerical values that we accept for considering two snapshots equivalent. In this first approach, we are considering the same level of tolerance for all the attributes. To make this comparison, we normalize the attributes rescaling them to the range between 0 and 1 using their maximum value. Concerning the discrete attributes, line 6, we consider that they all need to be equal in order to consider them equivalent.

It is important to note that our proposal assumes that the snapshots are taken at the same time steps in both the DT and the PT. The question about how to compare traces with different periods is still open for future work. Since time steps are the same in both systems, we do not take into account this attribute in the equivalence check. This way, we can detect missing states, stuttering or delays between traces, since the snapshots of two identical processes should match given some degree of tolerance if they reach the same states. If we included the timestamp in the equivalence check, we would be measuring synchronization of the traces. However, we would be unable to detect behavior equivalence in case of delays for example, since the algorithm would consider that case as a full mismatch, because the timestamps would not correspond. Additionally, we assume that the traces of the PT and the DT are at the same level of abstraction, i.e., they contain the same attributes. Otherwise, we could use the *abstraction function* mentioned in Section 2.1.

Figure 3 shows an example of an alignment of two sequences of snapshots of the Lego Car using a tolerance of 0.1. To simplify this example, we have only considered the x coordinate of the car (`xPos`). The snapshots presented for both the DT and PT simulate a car moving forward along the x-axis. However, the starting point of the DT is located before the one of the PT, and the DT slows down and then suddenly accelerates again. This generates a difference in both twins' trajectories that is reflected in the alignment shown on the right part of the figure. If we carefully analyze the alignment, we can see how the first snapshot of the PT was marked as deleted as a result of the alignment, because the algorithm is unable to match it with any of the PT's snapshots, since the DT's starting position was different. Second, our algorithm detects a missing snapshot in the DT on the third position since it reaches the location faster than the PT. The remaining snapshots are matched.

This alignment can help us not only to assess the possible differences between the traces but also to identify where these differences are exactly located. In the following sections, we analyze how to use the tolerance value and some distance measures in order to reason about the level of fidelity of the DT with respect to its PT.

## 3.2 Fidelity Measures

During the following sections, we will use an additional scenario to the car example to evaluate our proposal. Let us assume that we have a DT system of the Lego Mindstorms NXJ car. The DT's purpose is to monitor the trajectory of the PT to provide recommendations to avoid collisions. In this illustrative scenario, the PT describes a square trajectory and the DT tries to simulate the same movement. However, the DT describes the trajectory with a constant speed, while the PT goes slower and then accelerates at some point, catching up with the DT. Figure 4 shows the two trajectories aligned using a tolerance of 0.01 and 0.1, respectively. In the top chart, our algorithm only aligned the overlapped parts of the trajectory, considering the non-overlapping as gaps between the sequences. However, in the chart below, which uses a higher value of the tolerance (0.1), the algorithm aligns most of both trajectories.

To assess the level of fidelity with which the DT emulates the PT's behavior, we use different distance measures, which are shown in Table 1. If we check the columns from left to right, the first one is the tolerance used to align the traces. Columns 2 and 3 show the

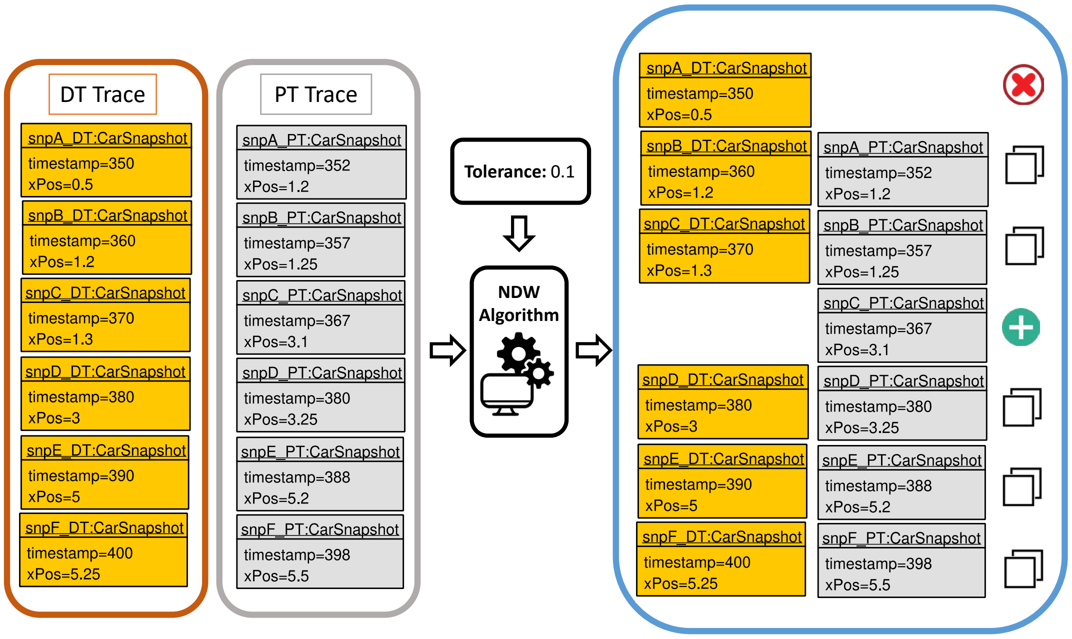

**Figure 3: Alignment of a set of traces using our approach with a tolerance of 0.1.**

**Table 1: Mapping scores and percentages of matched snapshots for the alignments, and results of several distances between the two trajectories, for different tolerance values.**

| Tolerance | Score | % matched | Fréchet (E) | Fréchet (M) | Avg±Std (E) | Avg±Std (M) |
|---|---|---|---|---|---|---|
| 0.01 | 86 | 41.04% | 0.19 | 0.19 | 0.76 ± 0.06 | 0.78 ± 0.06 |
| 0.025 | 102 | 48.58% | 3.02 | 3.38 | 0.78 ± 1.10 | 0.82 ± 1.17 |
| 0.05 | 137 | 65.09% | 9.05 | 9.94 | 3.64 ± 4.12 | 3.79 ± 4.33 |
| 0.075 | 163 | 77.36% | 12.08 | 13.41 | 5.71 ± 5.66 | 5.98 ± 5.97 |
| 0.1 | 180 | 85.38% | 9.02 | 9.55 | 7.68 ± 7.15 | 8.03 ± 7.52 |
| 0.25 | 206 | 97.64% | 3.12 | 3.13 | 1.54 ± 1.12 | 1.54 ± 1.12 |
| 0.5 | 211 | 100.00% | 3.12 | 3.13 | 1.78 ± 1.39 | 1.78 ± 1.39 |

results of the mapping, expressed in terms of the score obtained in the NDW algorithm and the percentage of snapshots matched by that algorithm. These two measures are important to assess the degree of matching between the traces. The rest of the columns in Table 1 show the distances between the traces using different measures, both elastic (i.e., the Fréchet distance) and lock-step (i.e., the Average distance), see Section 2.2. First, the Fréchet distance calculated using the Euclidean (E) and the Manhattan (M) distances, respectively; and then the average distance (Avg) between each matched pair of points using the Euclidean and the Manhattan distances, respectively, together with their corresponding sample standard deviations (Std).

Analyzing the values in these columns, we can see some interesting insights. Firstly, and as expected, the use of Euclidean or Manhattan distances is not really relevant, and the results obtained are practically the same — both for the elastic and the lock-step distances. Second, the scores increase as more points are included since the size of the NDW matrix also increases. In the rest of the columns, we can see how the distance increases as the tolerance decreases since we are reducing the requirement for equivalence and therefore we would expect the alignments to keep getting worse. However, this happens until we reach the 0.075 tolerance value, where the distance starts to decrease again until, for the 0.25 and 0.5 tolerance values, we get a value similar to that obtained for a tolerance of 0.025.

In view of these results, it is clear that we cannot base our fidelity measure only on the results of the distance measures: we should take into account both the tolerance and the percentage of points matched. To illustrate this claim, let us suppose that we have two almost identical trajectories; in this case, regardless of the value of the tolerance, the percentage of points matched would be close to 100% and the distances really close to zero. In our case, this happens in the first row, when we get distances really close to zero, which means that the alignment is almost perfect — and therefore the fidelity is very high. However, the alignment only includes 40% of the points with a tolerance of 0.01.

With these results, we would need to assess if the deviation detected in the non-overlapping parts is acceptable for our purpose, e.g., whether differences in the behaviors of the two twins represent a problem or not.

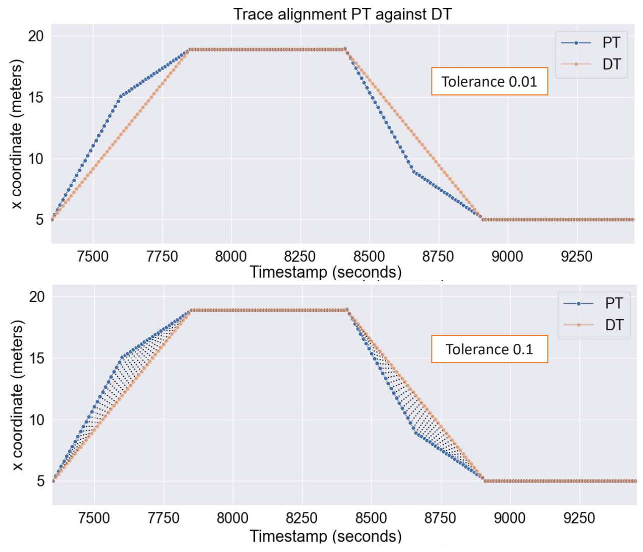

**Figure 4: Computed mappings between the trajectories of the PT (orange) against the DT (blue) showing its x coordinate along time, for two different values of tolerance: 0.01 (top) and 0.1 (bottom).**

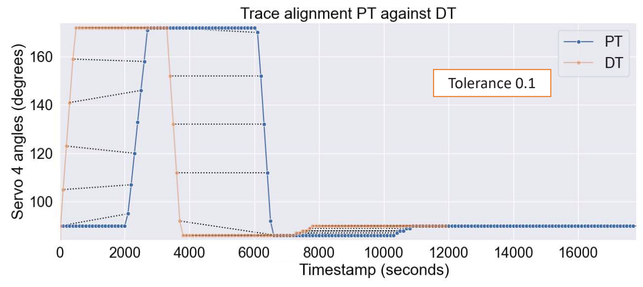

**Figure 5: Trajectory of the PT (orange) against the DT (blue) showing servo4 positions against time. Tolerance 0.1.**

## 3.3 The DT of a Robotic Arm

Another example that we have used for evaluating our proposal and validating our results is the DT of an Arduino Braccio, presented in [21]. The TinkerKit Braccio[1] is a fully operational robotic arm, controlled using an Arduino Board. It has 6 servos: one of them rotates the arm around its base, three servos control the arm's movements and position, and the last two allow rotating and controlling the gripper. Figure 5 shows a scenario in which the execution of `servo4` of the DT is delayed with respect to the same servo of the PT. The DT holds a certain position and then moves, but a bit later than the PT has moved. Our alignment is able to detect this delay when comparing the traces and also detects the parts of the trajectory which are equivalent after this point. This helps to better assess the fidelity of the DT since we only have to compare the parts of the executions that are equivalent to assessing their fidelity.

[1]https://store.arduino.cc/products/tinkerkit-braccio-robot

## 3.4 Discussion

After analyzing the results obtained in our example, we can discuss how these measures may point us to an assessment of the level of fidelity of the DT against the PT based on a given set of execution traces.

First of all, it is essential to consider the relationship between the *percentage of matched points*, the *tolerance* used to perform this matching, and the *distance* between the matched snapshots. The higher the percentage, and the lower the tolerance and the distance, the higher the quality of the alignment. This means, that the ideal match (which would lead us to assume that our DT's behavior is a replica of that of the PT) should include all the points, with tolerance and distance values close to zero. If we get higher distance values and a lower percentage of matched snapshots, we would be dealing with a lower fidelity DT, i.e, its behavior would deviate from that of the PT.

Once we have defined the matching algorithm and the measures for calculating the distance, our next step is to further study the interrelationship of these three parameters (required tolerance, percentage of matched points, and distance between the traces) to define concrete measures to assess the level of fidelity of DT systems from their performance traces, as well as indicators to make decisions on whether the degree of fidelity achieved is acceptable or not.

## 4 CONCLUSIONS AND FUTURE WORK

In this work, we have introduced a method and a set of measures to assess the fidelity of the DT against the PT. We introduced an adaptation of the NDW algorithm by providing an equivalence function to compare snapshots. We also defined a set of measures based on distances between snapshots that enable reasoning about the quality of the alignments. From these measures, we can reason about the level of fidelity based on the results of the alignment and the interrelationship between tolerance, percentage of matched snapshots, and the distance between them. We have validated our proposal with synthetic traces derived from two initial case studies. We have also discussed the importance of considering the interrelationship between abstraction, resolution, and fidelity when defining the alignments and measuring the fidelity level.

As part of our future work, we plan to check the performance of the trace alignment algorithm with a higher workload. We would also like to further analyze and interpret the results obtained for the traces, and their impact on the degree of fidelity of the system, deriving a specific measure and appropriate indicators to assess it. We plan to compare our algorithm with the optimized version of the NDW algorithm presented in [7], to see whether it really pays off in our context (it was originally devised to align DNA sequences). In our efforts to optimize the alignment algorithm, we also plan on define our own *Frequency scoring scheme*, cf. Section 2.3, to assign different scores to snapshots' elements according to their impact in the matching process. For instance, an attribute which is always the same may have less impact than attributes having very different values. Another interesting line of work is to explicitly take into account the possible uncertainties involved in the process while assessing the fidelity level, due to imprecision in the sensors' readings, delays in the communication channels, unreliable networks, or rounding errors. Finally, we want to contrast our results with traces taken from industrial DT systems to assess the applicability of our proposal in further scenarios. We would like to understand the factors that contribute to the complexity: the size of the system's snapshots, the size of the traces, etc.

## ACKNOWLEDGMENTS

We would like to thank the reviewers for their useful comments and suggestions. This work is partially funded by the Austrian Science Fund (P 30525-N31) and by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG). It is also funded by the Spanish Research projects PGC2018-094905-B-I00 (COSCA) and P20-00067-FR (MBT4IA).

## REFERENCES

[1] Bert Van Acker, Paul De Meulenaere, Joachim Denil, Yuri Durodie, Alexander Van Bellinghen, and Kris Vanstechelman. 2019. Valid (Re-)Use of Models-of-the-Physics in Cyber-Physical Systems Using Validity Frames. In *2019 Spring Simulation Conference, SpringSim 2019, Tucson, AZ, USA, April 29 - May 2, 2019*. IEEE, 1–12. https://doi.org/10.23919/SpringSim.2019.8732858

[2] John Ahlgren, Kinga Bojarczuk, Sophia Drossopoulou, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Simon M. M. Lucas, Erik Meijer, Steve Omohundro, Rubmary Rojas, Silvia Sapora, and Norm Zhou. 2021. Facebook's Cyber-Cyber and Cyber-Physical Digital Twins. In *EASE 2021: Evaluation and Assessment in Software Engineering, Trondheim, Norway, June 21-24, 2021*. ACM, 1–9. https://doi.org/10.1145/3463274.3463275

[3] Saba Alimadadi, Ali Mesbah, and Karthik Pattabiraman. 2018. Inferring hierarchical motifs from execution traces. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. ACM, 776–787. https://doi.org/10.1145/3180155.3180216

[4] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215, 3 (1990), 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

[5] K. Bojarczuk, N. Gucevska, S. Lucas, I. Dvortsova, M. Harman, E. Meijer, S. Sapora, J. George, M. Lomeli, and R. Rojas. 2021. 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)* (Bari, Italy) *(ESEM'21)*. ACM, 10 pages. https://doi.org/10.1145/3475716.3484196

[6] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. 2012. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Inf. Syst.* 37, 2 (2012), 117–141. https://doi.org/10.1016/j.is.2011.08.003

[7] Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. 2002. A Sub-Quadratic Sequence Alignment Algorithm for Unrestricted Cost Matrices *(SODA'02)*. Society for Industrial and Applied Mathematics, 679–688.

[8] Digital Twin Consortium. 2021. Glossary of Digital Twins. https://www.digitaltwinconsortium.org/glossary/index.htm.

[9] Martin Gogolla, Jörn Bohling, and Mark Richters. 2005. Validating UML and OCL Models in USE by Automatic Snapshot Generation. *SoSyM* 4, 4 (2005), 386–398.

[10] Liang Gong, Binhao Chen, Wenbin Xu, Chengliang Liu, Xudong Li, Zelin Zhao, and Lujie Zhao. 2022. Motion Similarity Evaluation between Human and a Tri-Co Robot during Real-Time Imitation with a Trajectory Dynamic Time Warping Model. *Sensors* 22, 5 (2022), 1968. https://doi.org/10.3390/s22051968

[11] David C. Gross. 1999. Report from the fidelity implementation study group. In *Proc. of the Fall Simulation Interoperability Workshops*.

[12] Philip Langer, Tanja Mayerhofer, and Gerti Kappel. 2014. Semantic Model Differencing Utilizing Behavioral Semantics Specifications. In *Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8767)*. Springer, 116–132. https://doi.org/10.1007/978-3-319-11653-2_8

[13] Edward A. Lee and Marjan Sirjani. 2018. What Good are Models?. In *Formal Aspects of Component Software - 15th International Conference, FACS 2018, Pohang, South Korea, October 10-12, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11222)*. Springer, 3–31. https://doi.org/10.1007/978-3-030-02146-7_1

[14] Dorian Leroy, Erwan Bousse, Anaël Megna, Benoît Combemale, and Manuel Wimmer. 2018. Trace Comprehension Operators for Executable DSLs. In *Modelling Foundations and Applications - 14th European Conference, ECMFA@STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10890)*. Springer, 293–310. https://doi.org/10.1007/978-3-319-92997-2_19

[15] Azad M. Madni, Carla C. Madni, and Scott D. Lucero. 2019. Leveraging Digital Twin Technology in Model-Based Systems Engineering. *Systems* 7, 1 (2019), 7. https://doi.org/10.3390/systems7010007

[16] Il-Chul Moon and Jeong-Hee Hong. 2013. Theoretic interplay between abstraction, resolution, and fidelity in model information. In *Proc. of WSC'13*. IEEE, 1283–1291. https://doi.org/10.1109/WSC.2013.6721515

[17] Usue Mori, Alexander Mendiburu, and José Antonio Lozano. 2016. Distance Measures for Time Series in R: The TSdist Package. *R Journal* 8, 2 (2016), 451. https://doi.org/10.32614/rj-2016-058

[18] Paula Muñoz, Javier Troya, and Antonio Vallecillo. 2021. Using UML and OCL Models to Realize High-Level Digital Twins. In *Proc. of ModDiT2021@MODELS'21*. IEEE, 212–220. https://doi.org/10.1109/MODELS-C53483.2021.00037

[19] Paula Muñoz. 2022. GitHub repository: Trace alignment for DTs. (2022). https://github.com/atenearesearchgroup/trace-alignment-for-dts

[20] Saul B. Needleman and Christian D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48, 3 (1970), 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

[21] Daniel Pérez-Porras, Paula Muñoz, Javier Troya, and Antonio Vallecillo. 2022. Key-Value vs Graph-based data lakes for realizing Digital Twin systems. In *Proc. of MeSS@STAF'22*.

[22] T.F. Smith and M.S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

[23] Sabine Wolny, Alexandra Mazak, Manuel Wimmer, and Christian Huemer. 2019. Model-driven Runtime State Identification. In *40 Years EMISA 2019, May 15-17, 2019, Tutzing, Germany (LNI, Vol. P-304)*. Gesellschaft für Informatik e.V., 29–44. https://dl.gi.de/20.500.12116/33137

[24] K. Worden, E. J. Cross, R. J. Barthorpe, D. J. Wagg, and P. Gardner. 2020. "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* 6, 3 (2020). https://doi.org/10.1115/1.4046740

[25] V Zhidchenko, I Malysheva, H Handroos, and A Kovartsev. 2018. Faster than real-time simulation of mobile crane dynamics using digital twin concept. *Journal of Physics: Conference Series* 1096 (2018), 012071. https://doi.org/10.1088/1742-6596/1096/1/012071