Warped Time Series Anomaly Detection

Charlotte Lacoquelle, Xavier Pucel, Louise Travé-Massuyès, Axel Reymonet, Benoît Enaux

▶ **To cite this version:**

Charlotte Lacoquelle, Xavier Pucel, Louise Travé-Massuyès, Axel Reymonet, Benoît Enaux. Warped Time Series Anomaly Detection. 2024. hal-04549643

HAL Id: hal-04549643
https://hal.science/hal-04549643

Preprint submitted on 17 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Warped Time Series Anomaly Detection

Charlotte Lacoquelle[1], Xavier Pucel[2], Louise Travé Massuyès[3], Axel Reymonet[1], and Benoît Enaux[1]

[1]Vitesco Technologies, Toulouse, France
[2]ONERA, DTIS, Université de Toulouse, France
[3]LAAS-CNRS, Université de Toulouse, CNRS, France

April 17, 2024

Abstract

This paper addresses the problem of detecting time series outliers, focusing on systems with repetitive behavior, such as industrial robots operating on production lines. Notable challenges arise from the fact that a task performed multiple times may exhibit different duration in each repetition and that the time series reported by the sensors are irregularly sampled because of data gaps. The anomaly detection approach presented in this paper consists of three stages. The first stage identifies the repetitive cycles in the lengthy time series and segments them into individual time series corresponding to one task cycle, while accounting for possible temporal distortions. The second stage computes a prototype for the cycles using a GPU-based barycenter algorithm, specifically tailored for very large time series. The third stage uses the prototype to detect abnormal cycles by computing an anomaly score for each cycle. The overall approach, named WarpEd Time Series ANomaly Detection (WETSAND), makes use of the Dynamic Time Warping algorithm and its variants because they are suited to the distorted nature of the time series. The experiments show that WETSAND scales to large signals, computes human-friendly prototypes, works with very little data, and outperforms some general purpose anomaly detection approaches such as autoencoders.

1 Introduction

The widespread expansion of connected devices and sensors in any domain (medical, financial, industrial) results in a massive production of data streams - where values are ordered or explicitly timestamped. This creates opportunities for automatically monitoring the execution of tasks in order to detect any deviation from normal execution, and react accordingly.

The quality of the data produced by sensors and data collectors can significantly affect the performance of monitoring approaches. In particular, we are interested in detecting abnormal cycles inside time series collected during the execution of repetitive tasks. In this setting, the considerable variation in cycle time and the need to accommodate missing data pose considerable challenges.

While this work applies to any dataset of cyclostationary time series, it is initially motivated by the need of monitoring robotic arms that operate on production lines. While these are equipped with inner fault detection mechanisms, they fail to detect problems related to their environment (setup, calibration, programming, product defects, etc). Our objective is to detect improper execution of repetitive tasks by the robots through by detecting abnormal execution cycles in their monitoring data. Several factors make it difficult to reuse the state of the art. First, the cycles are very long (in the tens of thousands of time points), and it has been observed that downsampling the time series degrades the detection quality. Second, the cycles may vary in length, because the robot program contains physical tasks of variable duration (eg object gripping, bar code scans), or simply because the operator takes a manual action. Operator intervention can be as limited as pausing and resuming the whole production line, or as intrusive as moving the robot in a different position, skipping part of its task, or re-calibrating it in the middle of a cycle. Ideally, the former interventions should not be detected as abnormal situations, while the latter should. Finally, we aim at being robust to a small but significant amount of data gaps in the acquisition process. This phenomenon produces inter-series and intra-series irregularities [1]. Inter-series irregularities refer to deviations that occur between different time series. In contrast, intra-series irregularities involve deviations within a single time series.

1.1 Scientific problem

This paper addresses the problem of detecting outlier time series inside a set of time series, in the context of intra-series irregularities. It targets systems exhibiting repetitive behavior and generating cyclostationary time series through continuous monitoring. The challenges posed are as follows:

- The time series represent cycles extracted from cyclic operation trajectories. They are ideally periodic, except that the irregularities mentioned above make them quasiperiodic (as defined in [2]). Given that the global shape of abnormal trajectories is not affected, the use of clustering would not be efficient.
- There are no labeled examples of what is considered normal behavior, therefore an unsupervised approach is required.
- The anomalies manifest themselves as gradual deviations in the time series rather than as isolated point outliers.
- The large gaps of missing data defeat window-based approaches, emphasizing the need to study the cycle signal as a whole.

The main contributions of this work are summarized as follows:

- A segmentation algorithm based on the Subsequence Dynamic Time Warming (Subsequence-DTW) algorithm that enables segmentation of a long sequence of repetitive task data in cycle sequences despite temporal warping.
- A novel GPU version of the Soft-DTW barycenter algorithm with chunk-splitting, which makes possible to obtain in reasonable time a representative normal cycle as the barycenter of a training set of cycle sequences.
- A time series outlier detection algorithm, named WarpEd Times Series ANomaly Detection (WETSAND), that allows warping in the time series similarity scoring used to detect anomalies
- A wide range of experiments involving different parameter settings, comparing results with competing algorithms in the field of Deep Learning, and validating the approach on real industrial robot data.

1.2 Motivating use case

The work reported in this paper is motivated by a plant that manufactures printed circuit boards. It features a fleet of 6-axis industrial robots by Universal Robots. Each robot is programmed for a specific task such as palletizing, stacking, or holding a camera for quality control. The repetitive nature of their programmed task causes the output time series to be, in theory, cyclic. However, some physical interactions, such as scanning a bar code or grasping an object, have variable duration, and as each cycle contains several dozens of such interactions, its length varies significantly in practice.

All the robots in the plant are connected to a monitor that collects the timestamp, the angular position of the 6 robot joints, the pose of the robot's end effector, and other variables such as the current and voltage measurements of the joint motors. The monitoring data is sampled at a frequency of 125 Hz. For various design, implementation and deployment reasons, data may be lost during periods of time of varying length, at irregular intervals. In other words, sampling is very irregular, and the time series feature large data gaps. This impacts diagnosis as these gaps contain complex patterns that cannot be reconstructed via interpolation methods. Prediction approaches also fail to impute the missing data, because in addition to the missing data, the physical robot cycles also contain pauses, vary in length, and the interactions with data gaps impair the accuracy of predictors.

While our approach applies to any robot or even any system that performs cyclic tasks, this paper focuses on a robot that fills containers with electronic devices at the end of a production line. Note that the health state of the robot is unknown and it is not possible to reproduce a laboratory experiment that could provide a reference behavior for a cycle as in [3].

The paper is organized as follows. Section 2 provides a state of the art of time-series anomaly detection and elastic distance measures. Section 3 provides background material on Soft-DTW and describes the WETSAND approach with a novel implementation of Soft-DTW. Section 4 describes the use case and the dataset on which the approach is evaluated. Section 5 evaluates the performances of WETSAND on the use case and provides a discussion on the algorithm's fine tuning. Finally, Section 6 concludes the paper.

2 State of the Art

The analysis of systems that perform repetitive tasks has already been addressed in various manners. In [3], the authors exploit the repetitive behavior of a robot - it could be any automated system - to generate fault indicators. They rely on nominal data, the first execution of the robot task, to estimate the joints condition, by monitoring the changes in the distribution of data batches. In [2], a two stage approach is used, where a time series is first segmented into quasi-periods, that are fed to an anomaly detection algorithm based on a deep neural network featuring recurrent and convolutional layers. Time series segmentation can also be performed using Dynamic Time Warping (DTW) [4], and more precisely its variant Subsequence-DTW [5] as in [6], to identify patterns by looking for local minimums that are under a fixed threshold. In [7], segmentation is also achieved through a Subsequence-DTW variant, before anomaly detection is performed with DTW. This approach relies on a reference trajectory produced by a high-fidelity simulator, which is costly to produce.

In time series, anomalies or outliers refer to values or patterns that do not conform with the expected behavior [8]. In [9] the authors present a taxonomy of outlier detection techniques in time series data. They distinguish three types of outliers: point outliers, subsequence outliers and outlier time series. The latter refers to a time series that differs in its entirety from the rest of the multivariate time series population.

The work of this paper falls within the detection of outlier time series. The approach is unsupervised because in our context, labels are rarely available.

AutoEncoder approaches are quite widespread, where a neural network is trained to reconstruct the system's normal behaviour, and poor reconstruction quality is interpreted as an anomaly. [10] uses an autoencoder with dilated convolution and temporal mechanisms to electrocardiogram data. [11] was one of the first papers to introduce a proper theoretical explanation for anomaly detection with a VAE on time series. Since then, the detection accuracy of VAE models has been improved by introducing discriminators to which are fed the raw and reconstructed data [12].

Many of the aforementioned approaches use deep neural networks (DNNs) that suffer from a long training time, laborious hyper-parameter tuning, and require high computational resources. Thus, as the study [13] demonstrates, before choosing complex DNNs, and especially in the context of real-world applications, it is crucial to evaluate whether conventional methods are more ap-

propriate.

Irregularly sampled data can prevent many techniques from performing well, and is an issue that must be addressed specifically. For instance, in [14], a classification task is possible despite very sparsely sampled datasets, enabled by the Gaussian process representation. In [1], the authors propose a time encoding mechanism that models the irregularities in the signal, and train a Recurrent Neural Network (RNN) for a prediction task that usually requires a uniform time distribution

Another approach to handle missing data is to ignore missing data points and compare time series through elastic distances. [15] performs a comprehensive evaluation of 71 time series distance measures but the design of new measures is an active research field. The Dynamic Time Warping (DTW) metric [4] is well known as an appropriate measure of similarity between time series because it handles time deformation and irregular sampling. K-Shape is much faster than Dynamic Time Warping (DTW) [4], however it only accounts for shifting the time series, and not for warping it. The DTW metric has been widely used for analysing time series, including for detecting outlier time series. In [16], DTW is used as the distance of the k-means clustering algorithm. However, anomaly detection itself is then performed inside each cluster using a decision tree approach.

Some elastic distances offer "averaging" capabilities, the average time series being the one that minimizes the value of some elastic distance to a set of time series. This is useful for computing a barycenter (also known as centroïd, pseudo-average, prototype or consensus object) that represents a set of time series. In [17] DTW is extended to a set of N signals, and a genetic algorithm is used to find a signal that minimizes its value. DTW Barycenter Averaging (DBA) [18] is a popular algorithm that provides a prototype through a subgradient iterative method. The more recent Soft-DTW formulation of DTW presented in [19] presents a smoothed DTW that uses the soft-minimum which makes it differentiable with respect to its inputs. Soft-DTW has several applications: clustering, prototype learning for a set of time series, or time series prediction. Similarly, [20] aims at designing the barycenter of spatio-temporal datasets; the temporal shifts are captured by Soft-DTW and the space and size invariances are handled with Unbalanced Optimal Transport.

Such barycenters can naturally be used for anomaly detection, by using the elastic distance between a time series and the barycenter of a given reference set.

3 Warped time series anomaly detection with WETSAND

The method presented in this paper, named WETSAND (from WarpEd Time Series ANomaly Detection), takes as input quasiperiodic time series with missing data. It is organized in three steps that will be called *Step 1*, *Step 2* and *Step*

3 in the paper:

1. *Segmentation into cycles*: starting with a reference cycle (with possibly missing data), we use the Subsequence-DTW (Sub-DTW) algorithm to segment the data stream into quasi periods, or cycles. This stage uses all the signal dimensions together to better accommodate for the data gaps that happen irregularly across the cycles.

2. *Prototype computation*: a set of cycles is used to obtain a prototype cycle with the Soft-DTW barycenter algorithm [19]. As available implementations fail to scale up to tens of thousands of time steps, we describe a novel GPU implementation specifically tailored to long time series.

3. *Outlier detection*: outlier cycles are detected by checking their similarity with the prototype cycle according to the DTW metric. Threshold-based approaches provides satisfactory performance for anomaly detection, more elaborate approaches are left for future work.

After introducing a few notations, this section describes each step in detail.

3.1 Modeling hypotheses and notations

The input of the WETSAND approach is a multivariate time series $\mathbb{Y} = [y_1, \ldots, y_K]$ of length $K \in \mathbb{N}$, where $y_i, i = 1, \ldots, K$ are d-dimensional, $d \in (\mathbb{N} \cup \{nan\})^*$, column vectors acquired with a theoretical sampling rate r. The nan values represent time steps with missing values, however DTW and its variants do not support them. As a consequence, we simply remove them from the regularly sampled time series $\mathbb{Y}$ and obtain an irregularly sampled time series $Y \in \mathbb{R}^{d \times N}$: $Y = [y_1, ..., y_N]$ of length $N \leq K$.

The time series Y can be several hours long with repeated patterns; a repetition is called a *cycle trajectory*. As explained in section 1.2, the length of each cycle trajectory can significantly vary.

We assume that a reference trajectory $X_{ref} \in \mathbb{R}^{d \times M}$ is provided by an expert. X_{ref} describes a single irregularly sampled cycle, and we assume that $M \ll N$. We have deliberately opted for an approach where an expert manually defines the boundaries of the reference cycle. The reference being the initial input of the method, this ensures that the results of the three steps of WETSAND remain easily interpretable for the expert involved.

In a first stage, the data stream is segmented into a set of cycle trajectories that match the reference trajectory. The segments are gathered into a set C_{traj}, and used to compute a prototype B.

In a second stage, the incoming data stream is also segmented into a set of cycle trajectories that match the barycenter B. The cycles are gathered into a set C_{test} upon which anomaly detection is performed.

In this paper, for practical reasons and without loss of representativity, a single data stream $Y \in \mathbb{R}^{d \times N}$ is gathered and segmented into cycles that match a reference trajectory $X \in \mathbb{R}^{d \times M}$. The identified cycles are gathered into a set C that is split into C_{train} and C_{test}.

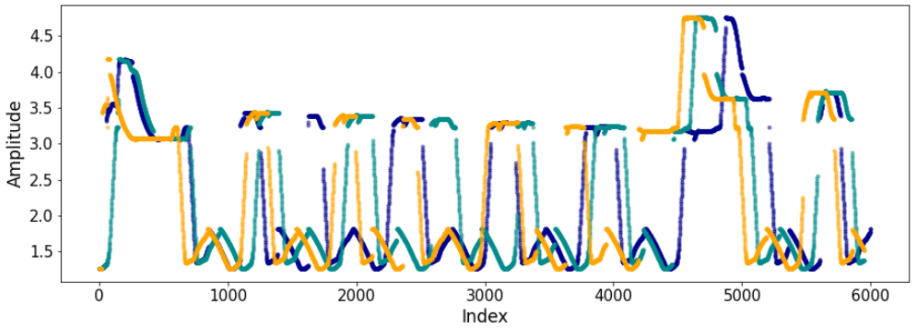

Figure 1: Three trajectories corresponding to three repetitions of the same task, with missing data and time warping that results from the variable duration of physical interactions (handling, sensing).

The 3 core functions of WETSAND are now defined in order: segmentation into cycles, prototype computation, and anomaly detection.

3.2 Segmentation into cycles

The first function of WETSAND uses Subsequence-DTW (Sub-DTW), a pattern-recognition technique allowing temporal deformations [5]. This section introduces definitions for the DTW and the Sub-DTW metrics.

Definition 1 (Alignment path)**.** *Given two multivariate time series of dimension d represented by $X = [x_1, ..., x_M] \in \mathbb{R}^{d \times M}$ and $Y = [y_1, ..., y_N] \in \mathbb{R}^{d \times N}$, an* Alignment path *$\pi$ between X and Y is a sequence of pairs of indices $\pi = [(i_1, j_1), \ldots, (i_k, j_k)]$ such that two consecutive pairs (i, j) and (i', j') satisfy:*

$$(i', j') \in \big\{(i, j+1), (i+1, j), (i+1, j+1)\big\}$$

A subsequence *alignment path is an alignment path such that $i_1 = 1$ and $i_k = M$. A* complete *alignment path is such that $i_1 = j_1 = 1$, $i_k = M$ and $j_k = N$, i.e. it starts at $(1, 1)$ and ends at (M, N).*

Definition 2 (Alignment matrix)**.** *An alignment path between two time series $X = [x_1, ..., x_M] \in \mathbb{R}^{d \times M}$ and $Y = [y_1, ..., y_N] \in \mathbb{R}^{d \times N}$ can be represented as an* Alignment matrix *$A \in \{0, 1\}^{M \times N}$ that contains 1 at the index pairs that belong to the alignment path and 0 elsewhere.*

The set of all alignment matrices for complete *alignment paths of size $M \times N$ is denoted $\mathcal{A}_{M,N}$.*

The set of all alignment matrices for subsequence *alignment paths of size $M \times N$ is denoted $\mathcal{A}^{sub}_{M,N}$.*

Definition 3 (Distance Matrix)**.** *Given two time series $X = [x_1, ..., x_M] \in \mathbb{R}^{d \times M}$ and $Y = [y_1, ..., y_N] \in \mathbb{R}^{d \times N}$, and a distance function $\delta : \mathbb{R}^{2 \times d \times M} \to \mathbb{R}$, the distance matrix $\Delta(X, Y) \in \mathbb{R}^{M \times N}$ such that $\Delta(X, Y)_{i,j} = \delta(x_i, y_j)$.*

The distance function δ is taken as the Euclidean distance, or l^2 norm.

Definition 4 (DTW)**.** *Given two time series $X = [x_1, ...x_M] \in \mathbb{R}^{d\times M}$ and $Y = [y_1, ..., y_N] \in \mathbb{R}^{d\times N}$, the cost of each alignment path π_α is equal to the inner product of its alignment matrix $A_{M,N}$ with the distance matrix $\Delta(X, Y)$.*

The DTW distance between X and Y is defined as the minimal alignment cost among all possible complete alignment paths between X and Y as follows:

$$DTW(X, Y) = \min_{A\in\mathcal{A}_{M,N}} \langle A, \Delta(X, Y)\rangle \tag{1}$$

In practice, the optimal alignment path and its associated alignment cost are obtained through a dynamic programming algorithm [4]. Sub-DTW is a variant of DTW that considers subsequence alignment paths. It is defined by:

$$\begin{aligned} \textit{Sub-DTW}(X, Y) &= \min_{A\in\mathcal{A}^{sub}_{M,N}} \langle A, \Delta(X, Y)\rangle \\ &= \min_{0\leq a\leq b\leq N} DTW(X, Y[a : b]) \end{aligned} \tag{2}$$

Where $Y[a : b] = [y_a, \ldots, y_b]$ is the subsequence of Y between indices a and b. Intuitively, Sub-DTW matches the whole time series X against the best subsequence of Y, usually noted $Y[a^* : b^*]$.

With Sub-DTW defined, the WETSAND segmentation algorithm of a time series $Y = [y_1, \ldots y_N] \in \mathbb{R}^{d\times N}$ with a reference quasi-period $X \in \mathbb{R}^{d\times M}$ is the following:

1. Initialize the set C_{traj} as the empty set and the sliding window indices at $ws = 0, we = 2M$.
2. Compute *Sub-DTW(X, Y[ws:we])* and note a^* and b^* the first and last indices along Y in the subsequence alignment path.
3. Add the time series $Y[a^* : b^*]$ to the set C_{traj}.
4. Update the sliding window indices to $ws = b^* - \alpha$, $we = ws + 2M$.
5. While the sliding window overlaps Y, i.e. $ws < N$, loop back to step 2).

α is a constant set at $0.15M$ in order to ensure that missing data between two cycles does not interfere with the cycle segmentation. When the algorithm terminates, C_{traj} contains the cycles in Y that were matched with X by the Sub-DTW algorithm. The segmentation step of WETSAND is illustrated in the evaluation section, in Figure 7.

In practice, there is a possibility that a fragment of the time series Y (between indices ws and a^* at step 4) is not part of any cycle, and discarded. While we did not encounter this situation in our experiments, it does not discredits our approach: at the prototype computation stage these data can simply be discarded; at the anomaly detection stage, this unassigned data constitutes in itself an anomaly. The anomaly detection stage can focus on identified cycles to detect more subtle anomalies.

3.3 Prototype computation

The second function of WETSAND relies on the Soft-DTW barycenter algorithm. Soft-DTW is another variant of DTW, first described in [19], that achieves a soft-minimum of all alignment costs. The principle behind the Soft-DTW variant is to replace the min operator with a soft $\min^\gamma$ operator defined as follows:

$$\min^\gamma(a_1, \ldots, a_n) = \begin{cases} \min_{i=1}^n a_i & \text{if } \gamma = 0 \\ -\gamma \log \sum_{i=1}^n e^{-a_i \gamma} & \text{if } \gamma > 0 \end{cases} \tag{3}$$

where $a_1, \ldots, a_n$ are the respective costs of the alignment paths $\pi_1, \ldots, \pi_n$ and γ is a smoothing parameter.

Definition 5 (Soft-DTW)**.** *Given two multivariate time series $X = [x_1, ...x_M] \in \mathbb{R}^{d \times M}$ and $Y = [y_1, ..., y_N] \in \mathbb{R}^{d \times N}$,, the Soft-DTW distance between X and Y is computed like $DTW(X, Y)$ in (1), but using the soft* $\min^\gamma$ *operator instead of* min*:*

$$\textit{Soft-DTW}^\gamma(X, Y) = \min_{A \in \mathcal{A}_{M,N}}^\gamma \left(\langle A, \Delta(X, Y) \rangle \right) \tag{4}$$

The dynamic programming algorithm that computing Soft-DTW is similar to that of DTW, and consists in computing the cumulative cost matrix.

Definition 6 (Cumulative alignment cost matrix)**.** *Given two multivariate time series $X = [x_1, ...x_M] \in \mathbb{R}^{d \times M}$ and $Y = [y_1, ..., y_N] \in \mathbb{R}^{d \times N}$, Soft-DTW$^\gamma(X, Y)$ is computed as follows:*

$$r_{i,j} = \begin{cases} 0 & \textit{if } i = j = 0 \\ \infty & \textit{if } i = 0 \textit{ or } j = 0 \\ \delta_{i,j} + min^\gamma \begin{pmatrix} r_{i-1,j-1}, \\ r_{i-1,j}, \\ r_{i,j-1} \end{pmatrix} & \textit{otherwise} \end{cases} \tag{5}$$

$$\textit{Soft-DTW}^\gamma(X, Y) = r_{M,N}$$

Where $\delta_{i,j}$ is shorthand for $\Delta(X, Y)_{i,j}$. The cumulative alignment cost matrix $R(X, Y)$ of size $M \times N$ retains the values $r_{i,j}$ for $i > 0$ and $j > 0$.

Figure 2 depicts the optimal soft alignment obtained with $\gamma = 0$ which is equivalent to the classical DTW alignment, and the alignment obtained with $\gamma = 1$.

The main advantage of Soft-DTW with respect to the standard DTW is that it is differentiable everywhere with respect to its input signal.

The partial derivative of *Soft-DTW*(X, Y) with respect to X is a vector defined as follows:

$$\nabla_X \textit{ Soft-DTW}^\gamma(X, Y) = \left(\frac{\partial \Delta(X, Y)}{\partial X} \right)^T E, \tag{6}$$

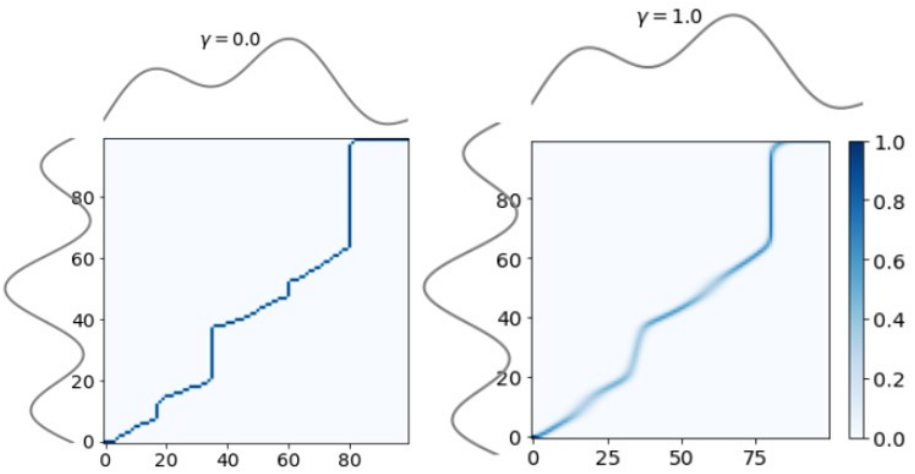

Figure 2: Two time series X and Y and the optimal warping path according to DTW (left) and according to Soft-DTW (right), illustrated through the alignment matrix (graph obtained using Python matplotlib library).

where $\Delta(X, Y)$ is the distance matrix between X and Y, and E is a matrix that can be computed by a dynamic programming algorithm described in [19] that requires $O(MN)$ memory, much like the DTW and Sub-DTW computations.

We use this partial derivative to compute an approximation of the Soft-DTW barycenter of a set of signals by gradient descent in a straightforward manner.

Definition 7 (Soft-DTW barycenter)**.** *Given a family of time series $Y_1, \ldots, Y_T$ of dimension d, of respective lengths $m_1, \ldots, m_T$ and with respective weights $\lambda_1, \ldots, \lambda_T$, a smoothing parameter γ and a barycenter length m_b, the Soft-DTW barycenter B is defined as the time series of length m_b that minimizes the sum of Soft-DTW distances between B and $Y_1, \ldots, Y_T$, normalized by their weights and lengths:*

$$B = \arg \min_{B \in \mathbb{R}^{d * m_b}} \sum_{i=1}^{T} \frac{\lambda_i}{m_i} \textit{Soft-DTW}^{\gamma}(B, Y_i) \tag{7}$$

Unless stated otherwise, we set the barycenter length at $m_b = \max(m_1, \ldots, m_T)$, and the weights at $\lambda_1 = \ldots = \lambda_T = 1$.

A very efficient implementation of Soft-DTW for short time series is found in the TSlearn library [21]. However, this implementation tends to be slow for long signals. For instance, computing the barycenter of 5 signals with 18385 time steps takes up to 4 hours on a cloud instance with 16 CPUs and 125GB of memory.

The barycenter B computation is parallelizable to some extent: the Soft-DTW distance and its partial derivatives are completely independent for each

pair $(B, Y_i), i = 1, \ldots, m_T$. Furthermore, each row in the R and E matrices can be computed in parallel with some synchronization. GPU based implementations of the Soft-DTW algorithm are described in [22] and [23] but they consume too much GPU memory when applied to long signals. The remainder of this section describes an implementation focused on fitting the computation in GPU memory by splitting the R and E matrices into *chunks*.

The proposed implementation consists in computing the matrices R of cumulative costs, and E of partial derivatives as mentioned in [19], by splitting them in chunks as illustrated in Figure 3. More precisely, the T-sized stack of R and E matrices form an R and E tensor of shape $\big(T, m_b, m_Y\big)$, where T is the number of time series, m_b the length of the barycenter, and $m_y = max(m_1, \ldots, m_T)$ is the length of the longest series in Y (m_i being the length of time series i). They are decomposed into chunks of shape (l_T, l_b, l_Y), where $l_T \leq T$ is the number time series evaluated in parallel, $l_b \leq m_b$ corresponds the number of synchronized GPU threads, and $l_Y \leq max(m_1, \ldots, m_T)$ is the number of R values computed by each thread in a chunk.

Tuning the chunk shape essentially depends on the GPU used and aims at enhancing parallelism, which is achieved by higher values for l_T and l_b. Threads in the l_b dimension perform synchronous computation, and current GPUs have a physical limit for this parameter. Threads in the l_T do not need synchronization, so it is desirable to have the highest value for this parameter, the only limitation being the GPU memory size. Furthermore, a very small value for l_Y makes the streaming mechanism (that parallelizes computation and data transfer) unbalanced and less efficient.

The R tensor computation takes as input the B and $Y_i's$ time series, and computes each chunk in increasing time direction, as illustrated in Figure 3. The computation of individual values inside each chunk uses thread synchronization schemes similar to those of [22]. Each chunk of the R tensor is memorized into a stack in the CPU memory. The E tensor computation is also performed by chunks in reverse order, and takes as input the B and $Y_i's$ time series and the corresponding R chunk, and produces the E tensor, from which the gradient vector is computed according to equation (6).

The values of lines, columns and corners between adjacent chunks are transferred between three buffers $hbuf$, $vbuf$ and $dbuf$ of respective shapes $(T, m_b, 1)$, $(T, m_Y, 1)$ and $(T, \lceil m_b/l_b \rceil, \lceil m_Y/l_Y \rceil)$ that are kept in GPU memory, in order to compute the first values of the next chunks, as illustrated in Figure 4. The individual values of the R and E matrices are computed in the classical dynamic programming manner described in [19] as well as [22] and [23].

Finally, the division of matrices in chunks helps leverage the stream mechanism of cuda-enabled GPUs. Two streams are used: one stream is in charge of the data transfer between the main memory and the GPU memory, while the other stream is in charge of kernel invocation, i.e. computation and allocation of shared and local memory.

The overall data flow of the proposed computation is as follows:

1. The B and $Y_i's$ time series are sent to the GPU memory and the buffers

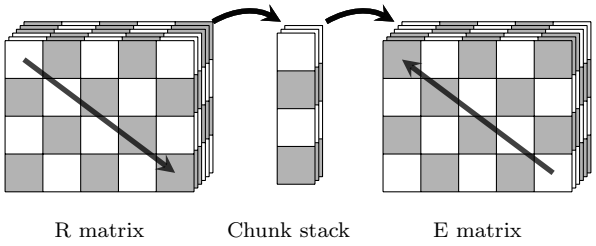

Figure 3: The 3-dimensional alignment and gradient matrices are decomposed into chunks for computation. The R matrix is computed in increasing time step indices, the E in reverse order.

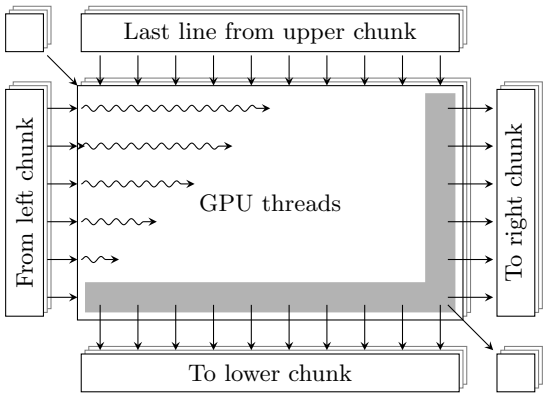

Figure 4: Computation of a chunk of the R matrix with several synchronized GPU threads. Buffers in GPU memory are used to store the initial values of equation (5). For the top left chunk of the R matrix (resp. bottom right of E), the vertical and horizontal buffers are initialized at ∞ (resp. 0) and the diagonal buffer to 0 (resp 1) as detailed in equation (5) (resp. [19]). For the following chunks the horizontal (B-wise) buffer stores the upper chunks' last lines, the vertical buffer (Y-wise) stores the left chunks' last columns and the diagonal buffer stores the bottom right value of the upper-left chunk. The E matrix is computed similarly, but in the opposite direction.

are initialized to hold the initial conditions of equation (5) (i.e. ∞ in $hbuf$ and $vbuf$, and 0 in $dbuf$).

2. Every chunk of the R tensor is computed on GPU, using equation (5) adapted to use buffers for initial values. After each chunk is computed, the buffers are then updated as of Figure 4, and the chunk is sent back to a LIFO stack in main memory.

3. A B-shaped vector is allocated in GPU memory to store the gradient vector $\nabla_B \sum_i \textit{Soft-DTW}^\gamma(B, Y_i)$ and initialized to 0. The buffers are filled with the initial conditions for computing E (i.e. 0 for $hbuf$ and $vbuf$, and 1 for $dbuf$).

4. Fourth, the R memory chunks are sent back in reverse order to GPU memory, and their corresponding E matrix chunk is computed. After each E chunk is computed, the gradient vector is incremented according to equation (6), and both R and E chunks are discarded.

5. After all R and E chunks have been processed, the gradient vector is returned in the main memory.

When computing the barycenter of signals with above 10^4 time steps, our implementation can reduce the computation time by up to 95%. An evaluation of the computation time reduction is done in subsection 5.2.

The WETSAND prototype computation is obtained by applying the GPU-based Soft-DTW barycenter computation to the subset C_{train} of C_{traj} (cf. Section 3.1), C_{traj} being a set of cycle trajectories obtained in the segmentation Step 1. This makes it possible to handle signals of varying length and irregularly sampled due to missing data. Assuming that data gaps are randomly scattered throughout the cycles, a sufficiently large C_{train} set should let the Soft-DTW barycenter B of C_{train} be a realistic prototype of normal cycles.

3.4 Online outlier detection

The third and last function of WETSAND consists in using the barycenter B as a reference to detect outlier time series online. B is used to compute a similarity score, inspired by [24], and defined in Equation 8) for each trajectory in the test set C_{test}, issued from C_{traj} (cf. Section 3.1). A threshold is chosen to decide whether a cycle is normal of abnormal.

Definition 8 (Similarity score)**.** *Given a barycenter B, a cycle C_k, and their optimal alignment path π, the* similarity score d_{score} *of C_k is a normalization of the DTW distance that compensates for the fact that trajectories have different lengths:*

$$d_{score}(B, C_k) = \frac{DTW(B, C_k)}{|\pi|} \tag{8}$$

Here, the DTW score is preferred to the Soft-DTW score to evaluate trajectories because it is deterministic and sometimes more accurate in corner cases. It is also invariant to time shifts while Soft-DTW is not entirely.

There exist many methods to select a threshold, and choosing one heavily depends on the distribution of d_{score} in the population. Section 5.3 evaluates different methods applied to this particular case. Ultimately, choosing an appropriate threshold is out of the scope of WETSAND and left to the user.

4 WETSAND applied to industrial robots

In this section, WETSAND is evaluated on the case study described in Section 1.2. For all our experiments, the Python programming language on a cloud instance with a NVIDIA A10G GPU, an 8-Core AMD EPYC 7R32 CPU and a limit of 31GB of memory has been used. When relevant, the proposed GPU-based Soft-DTW barycenter implementation is compared to that of the TS Learn library in terms of speed and results. It is a cross-platform software package for Python 3.5+ [21] that provides efficient implementations for Soft-DTW and Dynamic Time Warping Barycenter Averaging (DBA) (see [18]) algorithms. This library is itself based on scikit-learn [25].

The datasets are described in Section 4.1, and the assessment of the results, based on both quantitative and qualitative evaluations, is described in Section 4.2.

4.1 Datasets production

The task of a robotic arm is defined by a script, that specifies a number of way points in the robot's task space. The robot's controller computes the corresponding target joint trajectory in order to reach these way points. When the robotic arm moves, its internal encoders read the actual joint trajectory. These values - way points and joint positions - are collected from the controller into a data storage and form $Y \in \mathbb{R}^{d \times N}$, where $d = 9$.

For experimentation purposes, a robot, named "robot A", whose task is to fill containers has been selected. This robot fulfills the working hypotheses, due to the cyclic aspect of its movements on each of its monitored joints (cf. Figure 5). The monitoring data also features numerous and large gaps.

The dataset for the segmentation step defined in subsection 3.2 is composed of the robot movement captured from its first three joints (base is joint 0, elbow is joint 1 and shoulder is joint 2) during 9 hours with no particular incident documented.

The segmentation results in a pool of 217 cycles $C_{train_1}, ..., C_{train_{217}}$ represented by three matrices, one per joint, of 1 row and respectively $m_1, ..., m_{217}$ columns. Several cycles - the number varies across experiments - are randomly selected to create a training set C_{train} for the prototype computation step. Then, the evaluation of the results and the outlier detection step is done on a test set C_{test} of 190 randomly selected cycles among the rest. As explained in

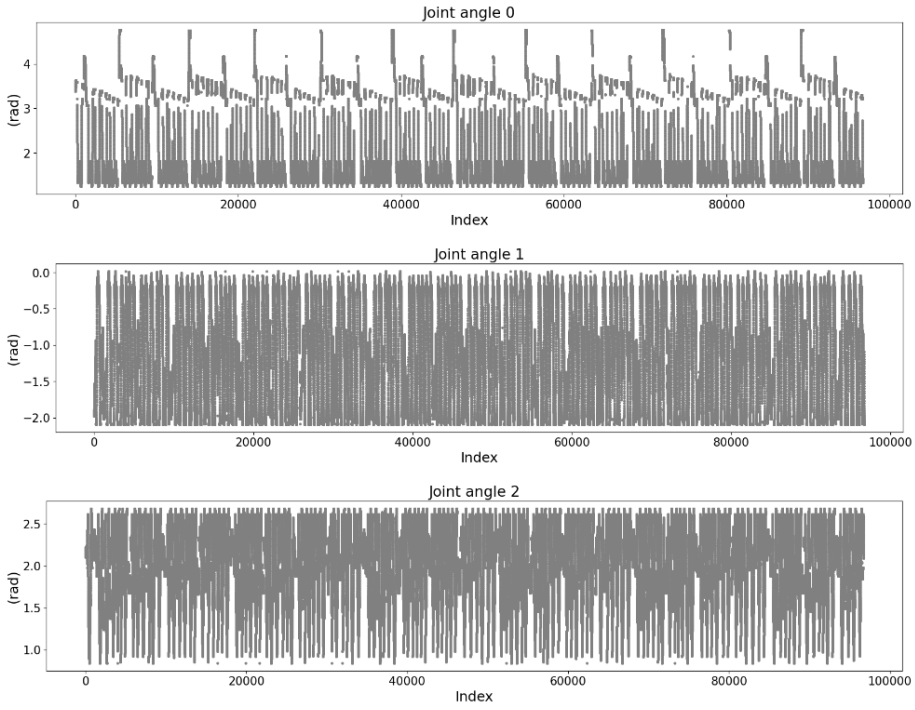

Figure 5: Robot angular position monitored during several task cycles, by internal sensors in joint 0 (base), joint 1 (elbow) and joint 2 (shoulder).

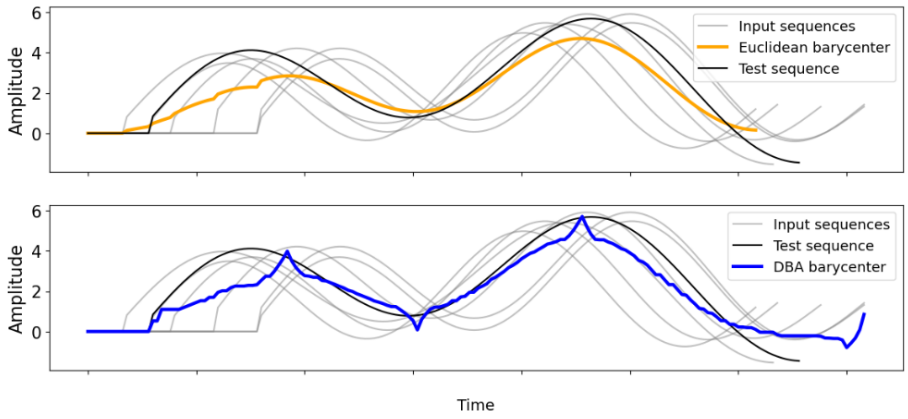

Figure 6: Illustration of the computation of Euclidean mean (above) and DBA barycenter (below) from an input set in grey. DBA tends to optimise better the sum of DTW distances, but is less interpretable.

subsection 3.1, each cycle has different length; to respect the requirements of Soft-DTW's implementation, all the cycle are padded with their respective last values to the length of the longest.

Another monitored robot, named "robot B", is used to provide the dataset for the experiments of 5.1. The task of robot B is to depalletize devices and place them into a machine. Its movement is much more complex and less smooth that that of robot A; due to the industrial setup of the data collection, this behavior increases the amount of missing data. The dataset acquired on robot B represents the worst case scenario to test the limitations of our approach.

4.2 Evaluation means and metrics

Abnormal cycles detected by WETSAND are meant to be analysed by a maintenance operator. This is why strong emphasis is placed on *visual* inspection of the results for the segmentation and the prototype computation steps. For instance, Figure 6 illustrates that a barycenter can easily be assessed visually. It is clear that the DBA barycenter, despite being excellent at minimizing elastic distances, looses the shape of the input signals, thus making its inner working confusing to a non-expert user.

The intraclass inertia with respect to the DTW distance is also used (the lowest value, the best). In Figure 6 the intraclass inertia for the Euclidean mean is 354.5 and the one for the DBA barycenter is 97.2, which shows that DBA is better suited to compare warped trajectories than Euclidean distance.

For computation times, comparison is done with the equivalent algorithms in the TS Learn library, as they provides high quality, efficient implementations.

For evaluating the overall anomaly detection performance, the whole C_{test} set has been manually annotated into normal and abnormal cycles. While the

whole WETSAND approach is completely unsupervised, this allows us to compare the results of WETSAND and the other approaches with the annotated ground truth. Confusion matrices and F_1 score are used to evaluate various means to set the detection threshold, as well as the global WETSAND approach.

5 Experiments and results

This section provides a detailed analysis of the three steps of WETSAND and evaluates their performances via the metrics described in subsection 4.2.

5.1 Evaluation of Step 1 – Segmentation into cycles

The aim of the first experiment is to partition a long dataset of monitored robot movement into cycle trajectories by applying Sub-DTW through a sliding-window. While [7] demonstrates a similar function, their reference cycle X is extracted from a simulator, which is better to avoid, given that a simulator is difficult to build or obtain from the robot manufacturer. Instead, an operator is asked to limit one cycle even if it is distorted and has data gaps, and use this cycle as the reference.

Figure 7 reports the result of such segmentation in which each limited cycle has different color. Qualitative visual evaluation was conducted on each signal and tends to shows that the segmentation algorithm works well and identifies the real bounds for the task cycles in the data stream.

The computation time is also satisfactory. The algorithm scales up to Y streams of around 6.10^5 time steps. A single segmentation step takes around 3 seconds in the case of robot A that has a reference cycle of 16 300 time steps.

As described in section 4.1 the dataset consists of 9 synchronous time series for each robot joint and 3 Cartesian coordinates, with synchronous data gaps. We found that using all of the robot's joint and task space variables all together as input to Sub-DTW enhances the accuracy of the segmentation. Thus, cycles are successfully characterized despite the poor quality of both the query and the search sequence.

5.2 Evaluation of Step 2 – Prototype computation

The second experiment aims at assessing the quality of the GPU-based Soft-DTW barycenter implementation presented in Section 3.3. In this experiment, each joint is analyzed separately. The reason for this is that an anomaly due to an internal failure may only be visible on one joint. Hence, there is a potential for the anomaly to become less noticeable when blended with the surrounding normal joints.

The desired qualitative properties of the barycenter are:

- to accurately represent the true movements of the robot during one cycle;

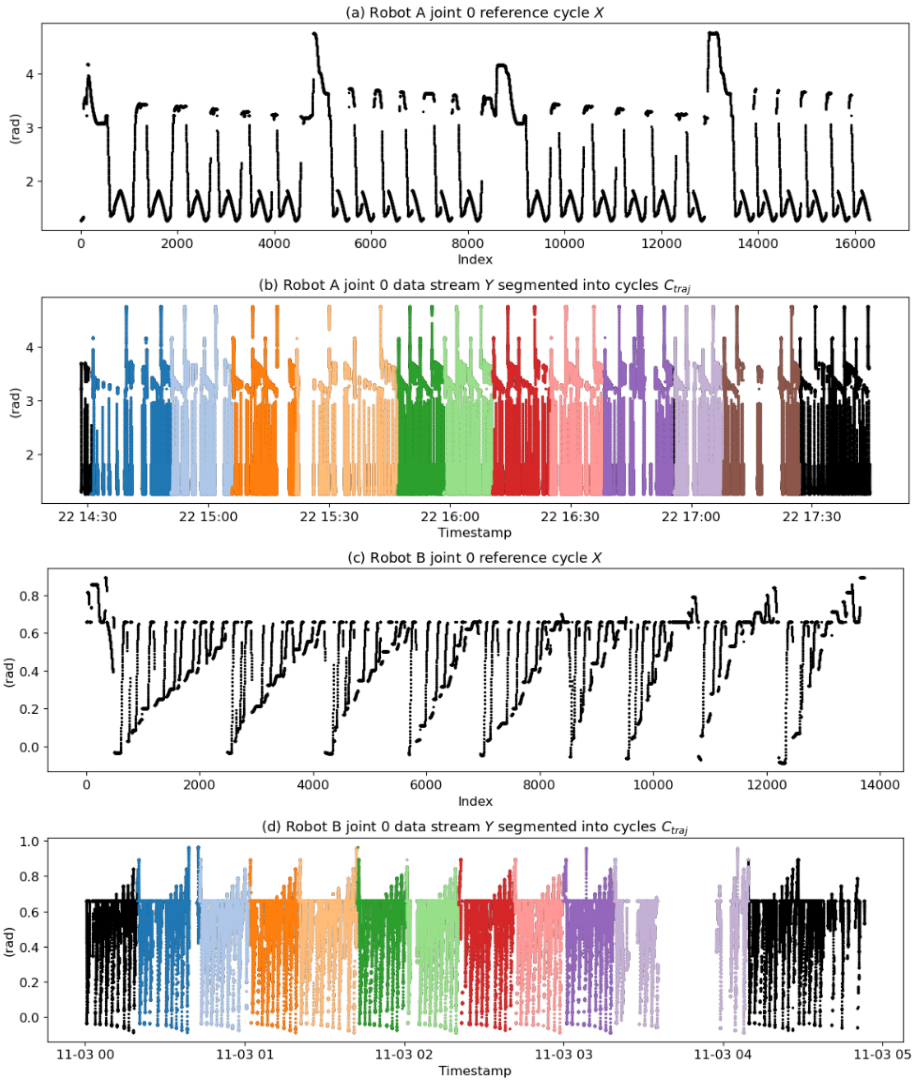

Figure 7: Segmentation of the movement data of two robot datasets into cycles. Above, the reference cycle (a) and segmented data stream (b) for robot A. Below, the reference cycle (c) and the segmented data stream (d) for robot B. In (b) and (d) black represents the samples that have not been associated with a complete cycle.

Table 1: Hyper-parameters used in WETSAND.

Parameter	**Description**	**Value**
Gamma	The DTW softness parameter (used in softmin)	1 (default)
Bandwidth	The Sakoe-Chiba bandwidth, as a fraction of the longest signal length	0.6
Init	Initial barycenter to start from for the optimization process	Euclidean barycenter (default)
Method	Optimization method	L-BFGS (default)
Max iter	(scikit-learn) maximum number of gradient descent updates	40
Max fun	(scikit-learn) maximum number of times soft-dtw barycenter is computed	200
G tol	(scikit-learn) gradient-based termination parameter for the L-BFGS algorithm	$1.e^{-8}$
Tolerance	(scikit-learn) distance-based termination parameter for the L-BFGS algorithm	$1.e^{-5}$
Chunk shape	Tuple (nx, ny, tx, ty) where nx (resp ny) is the number of signals from X (resp Y) and tx (resp ty) is the number of timesteps from x (resp y)	$(1, \lvert C_{traj}\rvert, 2^9, 2^{10})$ (default)

- to ensure the absence of any data gaps, despite the presence of numerous and sizable gaps in the original cycles of the training set C_{train}.

The barycenter computed by the GPU-based Soft-DTW implementation is also visually compared with the Euclidean mean, and the DBA barycenter provided by TS Learn. This allows one to assess the accuracy with which each barycenter faithfully represents a genuine cycle.

As illustrated in Figure 8, the Soft-DTW barycenter has the qualitative properties, and it obviously outperforms the Euclidean mean and the DBA barycenter in terms of representation faithfulness. Interestingly, this result has been obtained with only 5 trajectories in the training set. The hyper parameters used for the Soft-DTW computation are listed in Table 1.

The same conclusion is obtained with quantitative metrics, as provided in Table 2. On each of the three joints, Soft-DTW has the lowest *intraclass inertia* and mean d_{score} compared to the Euclidean mean and the DBA barycenter.

A final comparison to the TS Learn package is done regarding the computation time. The barycenter computation is achieved on the three joints of robot A with different input training set sizes via both implementations. The results are displayed in Table 3. The relevance of this comparison is guaranteed by the fact that the barycenters computed by each implementation are alike - indeed

Table 2: Comparison of Euclidean mean, DBA and the Soft-DTW barycenters w.r.t their ability to synthesize a proper robot trajectory

Algorithm	**Joint 0**		**Joint 1**		**Joint 2**	
	Intraclass inertia	Mean d_{score}	Intraclass inertia	Mean d_{score}	Intraclass inertia	Mean d_{score}
Euclidean mean	30440	0.197	25725	0.159	15979	0.103
DBA	3808	0.028	5411	0.032	3413	**0.023**
GPU-based Soft-DTW	**1516**	**0.017**	**2653**	**0.025**	**2484**	**0.023**

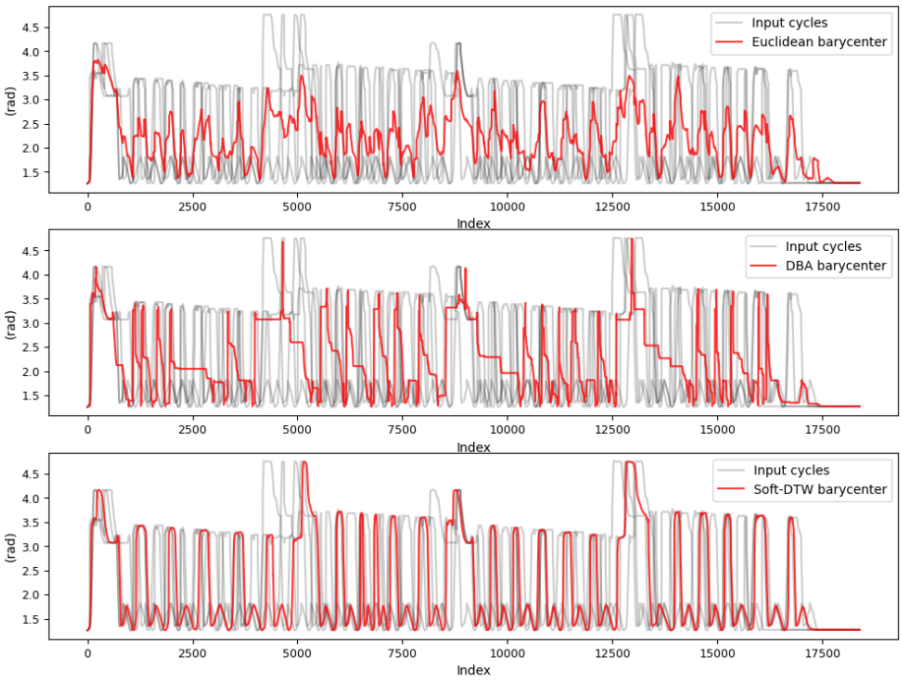

Figure 8: Comparison of barycenters obtained by Euclidean mean (top), by DBA (middle) and by Soft-DTW (bottom) on joint 0's angular position. In grey, the 5 input time series and in red the barycenter.

Table 3: Soft-DTW barycenter computation time with TS Learn and WETSAND implementations on signals with 18385 time steps.

Number of cycles in the training set C_{train}	5	10	25
TSLearn Soft-DTW computation time (minutes)	777	1217	3933
GPU Soft-DTW computation time (minutes)	**40.3**	**64**	**128**

the d_{score}'s order of magnitude between both of them is 10^{-3} which is extremely low for normalized signals of length 18385.

5.3 Evaluation of Step 3 – Online outlier detection

The ultimate target of WETSAND is the anomaly detection step that aims at labeling cycles as "normal" and "abnormal" according to their similarity with their respective joint's barycenter. Step 3 takes as input the barycenter computed in the previous experiment (cf. Section 5.2), and the test set C_{test}. It computes the d_{score} as of equation (8) for each sample of C_{test}. For the same reasons as for Step 2, the d_{score} is computed independently for each joint.

The evaluation of Step 3 is divided into three parts. First, the distribution of the d_{score} values is inspected with box plots to assess the feasibility to set a detection threshold. Second, the expert annotations on the test dataset (described earlier in Section 4.2) are used to establish the confusion matrix for different threshold values. Third, the results are compared with deep-learning approaches using several variants of autoencoders.

5.3.1 Setting the detection threshold

The boxplots of the distributions of d_{score} are depicted in Figure 9 for different sizes of the training set C_{train}: S_5 for 5 cycles, S_{10} for 10 cycles, and S_{25} for 25 cycles. The illustrated result is the mean of 10 randomly sampled training sets. The d_{score} clearly shows a sparse tail distribution for all joints and configurations, which makes a threshold based approach suitable.

Two possible ways to set the detection threshold are considered. First, the threshold is set at two standard deviations of the mean - but one could also set the threshold at three standard deviations. It is then called the 2σ *threshold*. Second, the threshold is based on the boxplot. It is then called the *boxplot threshold* and set to $Q_{max} = q_3 + 1.5 * |q_3 - q_1|$, where q_1 and q_3 are the first and the third quartiles of the distribution respectively (cf. Figure 9).

5.3.2 Evaluation against the expert annotations

The remainder of this section only considers the scores computed with a barycenter obtained with a training test of 5 cycles for conciseness. The relationship

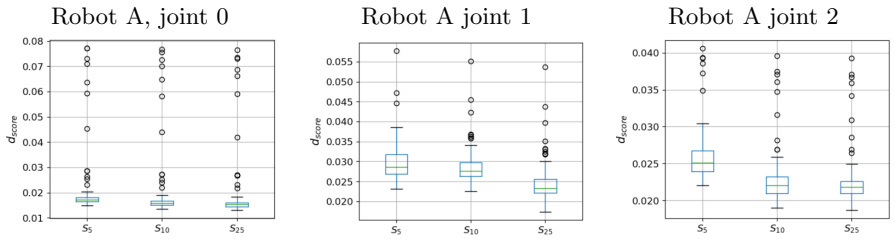

Figure 9: d_{score} distribution boxplots for 3 joints of robot A with 5, 10 and 25 cycles in the training set C_{train}, i.e. S_5, S_{10}, and S_{25} sizes for C_{train} respectively.

Table 4: Confusion matrix between ground truth and WETSAND prediction on joint 0 of robot A using a 2σ and a boxplot threshold.

		WETSAND 2σ threshold		WETSAND boxplot threshold	
		Anomaly	Normal	Anomaly	Normal
Ground truth	Anomaly	7	4	11	1
	Normal	0	179	1	177

between training set size and d_{score} distribution is further discussed in Section 5.4. The following evaluations of the anomaly detection step rely on expert annotations of the test dataset, as illustrated in Figure 10, which depicts the Soft-DTW barycenter for joint 0 of robot A, a cycle labeled "normal", and a cycle correctly labeled "abnormal". Similarity can be assessed by counting the number of peaks - which illustrate the joint going back and forth - and judging their amplitude.

The resulting confusion matrices are shown in Table 4. These are further analyzed through the F_2 score, a popular metric in Machine Learning. In the considered industrial context, the drawbacks of having false negatives (undetected anomalies) outweighs those of having false positives (spurious anomalies). In other words, *recall* is favored against *precision*, which means obtaining the highest F_2 score. Given that the 2σ threshold results in $F_2 = 0.69$ and the boxplot threshold results in $F_2 = 0.92$, the threshold is set at the boxplot threshold for the experiments that follow.

5.3.3 Evaluation against deep autoencoders

The performance of WETSAND is compared against that of a Convolutional AutoEncoder (CAE). CAE are often used for anomaly detection in time series through the use of 1D convolutional layers. Autoencoder-based anomaly detection consists in training the autoencoder on a set that contains only (or mostly) normal individuals. Under this training protocol, it is assumed that the CAE reconstructs new normal individuals - which are similar to those used for training - better than abnormal ones. Then, the reconstruction error that is the loss

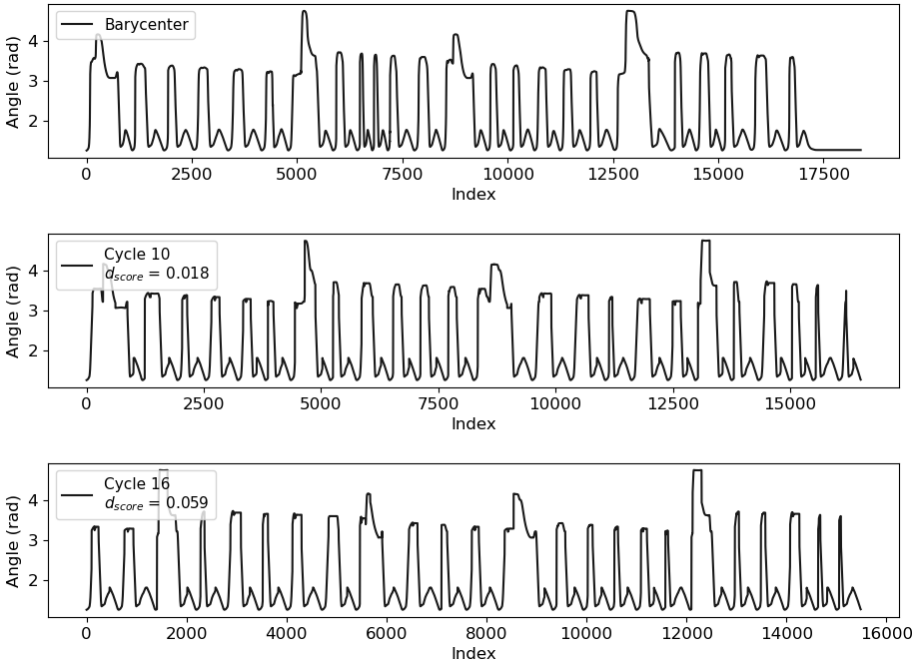

Figure 10: The Soft-DTW barycenter for joint 0 of robot A (top), a normal (middle), and an abnormal cycle (bottom).

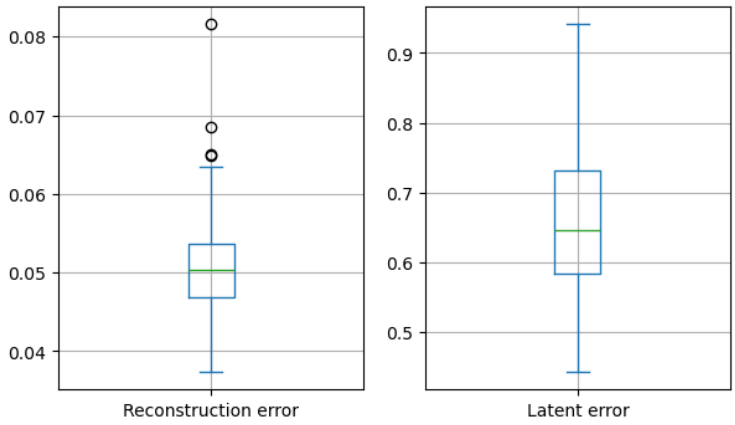

Figure 11: Reconstruction error (left) and error computed on the latent space (right) of the baseline CAE on a test set of 190 trajectories.

function used to train the autoencoder naturally becomes a d_{score} for anomaly detection.

While autoencoders can learn extremely elaborate datasets, they usually require massive amounts of training data. This is incompatible with the industrial setting at hand, where robots are daily recalibrated or reassigned to a different task. For a fair comparison, autoencoders are trained with the same number of cycles as the Soft-DTW barycenter, i.e. on a training set C_{train} of 5 trajectories. As a consequence, architectures that can compact signals of 18 000 time steps into a small latent space with few parameters are aimed for.

To that end, investigations were conducted to find the best CAE architecture. The time series are first last-value-padded to 24576 time steps, which is the maximal number of time steps of the cycles in the training set. A first convolutional layer with layer kernel size 32, stride 32, and 16 output channels is followed by a rectified linear unit (ReLU) layer used for non linearity. A second convolutional layer uses kernel size 32, stride 32, and 16 output channels, thus yielding a latent space of dimension $5 * 16 * 24 = 1\,920$, and a total of 17 457 trainable parameters. Decoding is performed with two 1D transpose convolutional layers with the same parameters, separated by a ReLU layer. Fine tuning is computed with the adaptative moment estimation (ADAM) algorithm with a learning rate of 10^{-3}. The parameters were chosen to minimize the loss score during the training stage, while minimizing the number of parameters and therefore, the training duration. The reconstruction error distribution is depicted on the left of Figure 11. Setting the boxplot anomaly threshold yields the confusion matrix depicted in Table 5, and an F_2 score of 0.10, which is a quite poor result.

Variants of the CAE model have been tested to improve the anomaly detection performance, such as adding Attention layers, which reduces the total

Table 5: Confusion matrix for anomaly detection using a Convolutional Autoencoder

		Convolutional autoencoder	
		Anomaly	Normal
Ground truth	Anomaly	1	11
	Normal	3	175

number of parameters of the model. In particular, a test was done with a CAE whose encoder consists of a first layer of kernel size 32, stride 32, 8 output channels, feeding second layer of 8 channel multihead attention (in all query, key and value inputs) followed by a third convolutional layer of kernel size 16, stride 16 and 8 output channels, with two ReLU layers in between. The decoder is symmetric, with the same attention layer and convolutions being replaced with transpose convolutions. This CAE has a latent space dimension of $5 * 8 * 48 = 1\,920$, but only 3 161 trainable parameters. It yields an F_2 score of 0.23, which is better that the previous CAE but is still much lower than the anomaly detection performance with WETSAND. Other autoencoder variants, such as variational autoencoders, recurrent layers, or using the latent space distance as a detection threshold (an idea presented in [26]), did not yield better results.

An explanation for the poor performance of autoencoders is illustrated in Figure 12. Even with a low number of parameters, the autoencoders reconstruct normal and abnormal cycles equally well. Increasing the number of network trainable parameters only worsens the problem. Conversely, decreasing the number of trainable parameters to the limit of underfitting only makes the autoencoder equally bad at reconstructing normal and abnormal cycles. Using recurrent layers or larger training sets also yields similar results.

An interpretation is that the nature of targeted anomalies does not bode well with autoencoders. Targeted anomalies rarely cause out of bounds signal points, but are rather characterised by a missing or an undesired pattern, e.g. longer or shorter than usual. An autoencoder that has learned the nominal robot pattern can easily reconstruct a cycle with an additional (or missing) pattern with the same shape. This explains why autoencoders generalize too well to abnormal cycles in the presented use case, and fail at detecting them. In contrast, elastic distances such as DTW used in WETSAND must match each pattern to a reference, exhibiting high sensitivity to the absence or inclusion of undesired patterns, which perfectly fits the need of targeted anomalies. Another argument against CAE in this application is that the tuning of depth and hyperparameters is long, and it is specific to each type of time series. It is unlikely that in the industrial context, one would fine tune a model for each robot and each robots' joints.

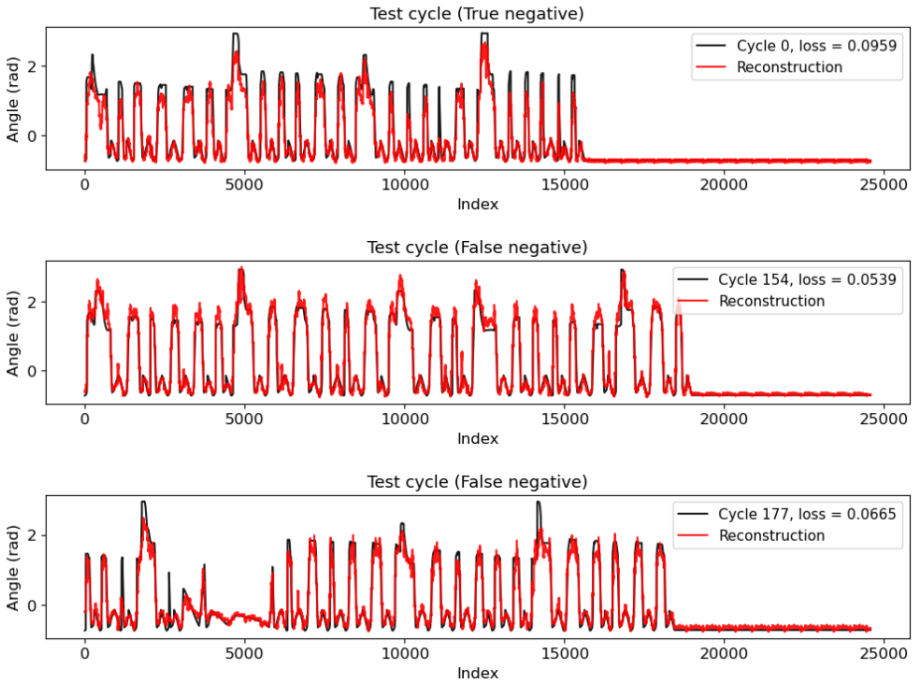

Figure 12: Examples of reconstructed cycles (in red) with the Attention-CAE compared to the input cycle (in black). Every cycle has a low d_{score}, even though the middle and lower ones bear anomalies.

5.4 Discussion on hyperparameter tuning

In this subsection, the impact of several hyperparameters of the Soft-DTW algorithm on the barycenter computation are evaluated. We report on the tuning of 2 hyperparameters:

- barycenter initial value;
- training set size;

5.4.1 Barycenter initial value tuning experiment

3 initialization methods for the gradient descent phase of the Soft-DTW barycenter are compared: zero, random, and euclidean mean. It has been observed that initializing the algorithm with an array of zeros makes gradient descent 10% slower than when randomly initialized. However, the produced barycenter is more similar to the normal trajectory. Initializing with the euclidean mean of the training set makes gradient descent 8% faster than zero-initialization, and yields an even better barycenter.

5.4.2 Training set size tuning experiment

This experiment reports on the influence of the size of the training set C_{train} on the barycenter computation. The intuitive idea driving this experiment is that a higher number of cycles in the training set will lead to a more representative barycenter. Consequently, this is expected to result in a higher similarity score between the barycenter and the normal test cycles. Here, the GPU-based implementation is run with the parameters described in Table 1. Three training sets are tested: the subsets S_5, S_{10}, S_{25}, which are randomly selected from the cycle pool, such that $|S_5| = 5, |S_{10}| = 10, |S_{25}| = 25$, and $S_5 \subset S_{10} \subset S_{25}$. Besides, the experiment is done independently on the three joints of robot A and each of them is repeated ten times for statistical accuracy; then, the mean result for each sample is recorded. The d_{score} between the computed barycenters and the test set according to the training set size on each joint are presented on Figures 9. A lower median value for S_{25} on each experiment demonstrates the validity of the intuition. It is also noticeable in Table 6 that, depending on the signal's shape, the training set size has an impact on the sensitivity of the d_{score} to outliers. While joint 0's barycenter is satisfying with S_5 as a training set, the anomaly detection is improved with a barycenter computed with a larger training set for joints 1 and 2.

6 Conclusion

In this paper, a frugal and efficient anomaly detection approach named WETSAND is introduced. It is designed to handle cyclostationary time series with irregular sampling and distortion in the time dimension. WETSAND segments

Table 6: F_2 scores on the test set of 190 cycles, with WETSAND implementation on Robot A's three joints.

Number of cycles in the training set C_{train}	5	10	25
F_2 score Robot A Joint 0	**0.92**	**0.92**	**0.92**
F_2 score Robot A Joint 1	0.42	0.80	**0.95**
F_2 score Robot A Joint 2	0.79	**0.97**	**0.97**

a large quasiperiodic time series into quasiperiodic cycles, using as input a (possibly corrupted) cycle. It then automatically computes a representative prototype of the normal cycles using an efficient and new GPU-based implementation of the Soft-DTW barycenter, specifically tailored for long time series. Outlier time series detection is then performed using the DTW distance between the computed barycenter and upcoming data.

WETSAND's efficiency is evaluated with several experiments performed on real data from industrial robots. It is demonstrated that WETSAND scales up to times series in the tens of thousands of time steps, and outperforms autoencoder based approaches on the data set of repetitive robotic arm motions. This is another argument in favor of the assertion developed in [13] that says that deep neural network methods do not necessarily outperform conventional approaches and should not be used systematically.

Acknowledgments

The authors would like to thank Christophe Merle, Head of the AI Group at Vitesco Technologies and ANITI Industrial Coordinator, for sharing his expertise in robotized production lines and for his sound advice. This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement no. ANR-19-PI3A-0004.

References

[1] C. Sun, S. Hong, M. Song, Y. Zhou, Y. Sun, D. Cai, and H. Li, "Te-esn: time encoding echo state network for prediction based on irregularly sampled time series data," *arXiv preprint arXiv:2105.00412*, 2021.

[2] F. Liu, X. Zhou, J. Cao, Z. Wang, T. Wang, H. Wang, and Y. Zhang, "Anomaly detection in quasi-periodic time series based on automatic data segmentation and attentional lstm-cnn," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 6, pp. 2626–2640, 2020.

[3] A. C. Bittencourt, K. Saarinen, and S. Sander-Tavallaey, "A data-driven method for monitoring systems that operate repetitively-applications to

wear monitoring in an industrial robot joint," *IFAC Proceedings Volumes*, vol. 45, no. 20, pp. 198–203, 2012.

[4] 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.

[5] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," in *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 2006, pp. 1046–1055.

[6] J. Barth, C. Oberndorfer, P. Kugler, D. Schuldhaus, J. Winkler, J. Klucken, and B. Eskofier, "Subsequence dynamic time warping as a method for robust step segmentation using gyroscope signals of daily life activities," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2013, pp. 6744–6747.

[7] C. Lacoquelle, L. Travé-Massuyès, X. Pucel, N. B. Roa, and C. Merle, "Deviation tracking with incomplete and distorted data-application to motion trajectories of industrial robots," in *33rd International Workshop on Principle of Diagnosis–DX 2022*, 2022.

[8] C. C. Aggarwal and C. C. Aggarwal, *An introduction to outlier analysis*. Springer, 2017.

[9] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[10] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Outlier detection for time series with recurrent autoencoder ensembles." in *IJCAI*, 2019, pp. 2725–2732.

[11] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 187–196.

[12] Z. Fan, Y. Wang, L. Meng, G. Zhang, Y. Qin, and B. Tang, "Unsupervised anomaly detection method for bearing based on vae-gan and time-series data correlation enhancement (june 2023)," *IEEE Sensors Journal*, vol. 23, no. 23, pp. 29 345–29 356, 2023.

[13] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Do deep neural networks contribute to multivariate time series anomaly detection?" *Pattern Recognition*, vol. 132, p. 108945, 2022.

[14] S. C.-X. Li and B. M. Marlin, "Classification of sparse and irregularly sampled time series with mixtures of expected gaussian kernels and random features." in *UAI*, 2015, pp. 484–493.

[15] J. Paparrizos, C. Liu, A. J. Elmore, and M. J. Franklin, "Debunking four long-standing misconceptions of time-series distance measures," in *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, 2020, pp. 1887–1905.

[16] E. Zhang, N. Masoud, M. Bandegi, and R. K. Malhan, "Predicting risky driving in a connected vehicle environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 17 177–17 188, 2022.

[17] F. Petitjean and P. Gançarski, "Summarizing a set of time series by averaging: From steiner sequence to compact multiple alignment," *Theoretical Computer Science*, vol. 414, no. 1, pp. 76–91, 2012.

[18] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[19] M. Cuturi and M. Blondel, "Soft-dtw: a differentiable loss function for time-series," in *International conference on machine learning*. PMLR, 2017, pp. 894–903.

[20] H. Janati, M. Cuturi, and A. Gramfort, "Averaging spatio-temporal signals using optimal transport and soft alignments," *arXiv preprint arXiv:2203.05813*, 2022.

[21] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods, "Tslearn, a machine learning toolkit for time series data," *Journal of Machine Learning Research*, vol. 21, no. 118, pp. 1–6, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-091.html

[22] H. Zhu, Z. Gu, H. Zhao, K. Chen, C.-T. Li, and L. He, "Developing a pattern discovery method in time series data and its gpu acceleration," *Big Data Mining and Analytics*, vol. 1, no. 4, pp. 266–283, 2018.

[23] B. Schmidt and C. Hundt, "cuDTW++: Ultra-Fast Dynamic Time Warping on CUDA-Enabled GPUs," in *Euro-Par 2020: Parallel Processing*, ser. Lecture Notes in Computer Science, M. Malawski and K. Rzadca, Eds. Cham: Springer International Publishing, 2020, pp. 597–612.

[24] J. Jiang, S. Lai, L. Jin, and Y. Zhu, "Dsdtw: Local representation learning with deep soft-dtw for dynamic signature verification," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2198–2212, 2022.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*. Springer, 2019, pp. 622–637.

7 Biography Section

To be completed after anonymous review