Article

A Novel Deep Clustering Method and Indicator for Time Series Soft Partitioning

Alexandre Eid [1,2]*, Guy Clerc [1], Badr Mansouri [2] and Stella Roux [3]

[1] Univ Lyon, Université Claude Bernard Lyon 1, INSA Lyon, École Centrale de Lyon, CNRS, Ampère, UMR5005, 69622 Villeurbanne, France; alexandre.eid@univ-lyon1.fr (A.E.); guy.clerc@univ-lyon1.fr (G.C.)

[2] Safran Electronics & Defense, Massy, France; badr.mansouri@safrangroup.com (B.R.)

[3] Grenoble INP - Ensimag, UGA, France; stella.roux@grenoble-inp.org (S.R.)

* Correspondence: alexandre.eid@univ-lyon1.fr

† This paper is an extended version of our paper published in 2021 IEEE International Conference on Prognostics and Health Management (ICPHM 2021).

Abstract: The aerospace industry develops prognosis and health management algorithms to ensure better safety on board. Particularly for in-flight controls where jamming is dreaded. For that, vibration signals are monitored to predict future defect occurrences. However, time series are not labeled according to severity level, and the user can only assess the system health from the data mining procedure. To that extent, we developed a clustering algorithm using a deep neural network core. We encoded the time series into pictures to be feed into an artificially trained neural network: U-NET. From the segmented output, one-dimensional information on cluster frontiers is extracted and filtered without any parameter selection. Then, a kernel density estimation finally transforms the signal into an empirical density. Ultimately, a Gaussian mixture model extracts the latter independent components. The method empowered us to reveal different degrees of severity faults in the studied data, with their respective likelihood without prior knowledge. We compared it to state-of-the-art machine learning algorithms. However, internal clustering results evaluation for time series is an open question. As state-of-the-art indexes were not producing relevant results, we built a new indicator to fulfill this task. We applied the whole method to an actuator consisting of an induction machine linked to a ball screw. This study lays the groundwork for future training of diagnosis and prognosis structures in the health management framework.

Keywords: semantic segmentation; time series; clustering; deep learning; kernel density estimation; electromechanical actuator; data labelling; prognosis and health management; aeronautics

1. Introduction

According to the Maintenance Cost Technical Group (MCTG) from the International Air Transport Association (IATA), US\$69 billion was spent on maintenance repair and overhaul (MRO)in 2018 for a fleet count of $27,535$ aircraft from 54 airlines. The aerospace industry is shifting from hydraulic to electromechanical actuation, which could be prone to seizure. Hence, it is mandatory to prevent actuator jamming at any cost during in-flight control where several ball-screws are used.

The prognosis and Health Management (PHM) framework can be used to reduce the amount of money spent on unscheduled MRO operations by developing methods to forecast structural defects. Further, it can better anticipate this hazardous jamming event by detecting any unusual behavior from either signal of vibration or electrical.

Nowadays, the PHM framework has been moved from conventional–based health monitoring (HM) to a deep learning approach. Therefore, the previous hand-designed features extracted from raw data are automatically determined by the first layers of a deep neural network (DNN). Also, in the extensive review of deep learning by [1], visualizing time series behavior as pictures is an ongoing research topic. Time series dynamics can be extracted by the Gramiam Angular Field representation (GAF) of [2] or the Recurrence Plot (RP) of [3]. These representations can be fed into a DNN for better classification of data. In our work, a similar approach is used for visualizing time series behavior.

In [4], Jain defined *clustering* as "the unsupervised classification of patterns into groups." While this exploratory analysis technique is widely used in spatially organized datasets, it is scarcely used in time series where data is timely organized. Hence the need to create a new method with its quality indicator. Several recent studies have been presented using clustering or pattern recognition to label data for further use in the training of an artificial diagnosis structure. In [5], every deviation from the nominal state of electrical machines is detected from a clustering algorithm. Plus, the operating conditions can be inferred from the data group structure [6]. The association between a group and a fault severity is clearly stated in [7]. We can assume that patterns found in this approach are representative of several fault detection. We followed the same hypothesis: a group of data represents a certain fault severity. For the fussy clustering algorithm, [8] presents another approach.

Further, finding a general indicator is challenging as the clustering quality indexes are specific to the fundamental hypothesis made in the research [9]. This article presents a new method and a new quality indicator for clustering and quantifying the results for time series data. First, the method will be outlined, and then it will be applied to time series. The time series are vibration signals coming from an induction machine attached to a ball screw. From it, we extract and study many precursors of jamming defects. Finally, our method is tested against several state-of-the-art algorithms in the discussion part, and all the results are quantified with a newly created indicator.

2. Materials and Methods

The clustering method has been applied to time series in a broad sense, i.e., indexed signals by a timestamp. More generally, signals, which elements abide by a common precedence relation, are considered. Let $\mathcal{T}$ be such a signal. Practically, it represents the values of a given sensor indexed by time, on a monitored system. n realizations of $\mathcal{T}$ for the system can be gathered: $\{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n\}$. In order to assess system health, an ensemble of p statistical and model-based parameters are extracted from each $\mathcal{T}_{i\in[\![1;n]\!]}$. Let $\mathbb{X}_{j\in[\![1;p]\!]} \in \mathcal{M}_{1n}(\mathbb{R})$ be such parameter. Each one of them can be concatenated to form what is called a *feature matrix* $\mathbb{M} \in \mathcal{M}_{np}(\mathbb{R})$.

In the following work, studied parameters are the chosen columns of the *feature matrix* $\mathbb{M}$. Let choose for $j \in [\![1;p]\!], \mathbb{X}_j = \begin{bmatrix} x_1^j & x_2^j & ... & x_n^j \end{bmatrix}^\top$ the jth descriptor.

A clustering method deals with data classification without prior knowledge. To do so, it has to minimize an intra-class distance and maximize an outer-class one. Doing so, it aggregates objects with similar properties and repels objects with heterogeneous ones. A distance is used to compare its elements $x_{i\in[\![1;n]\!]}$ between each other, allowing the extraction of relevant clustering information from each $\mathbb{X}_j$. It allows the extraction of groups with similar characteristics. For that, the Euclidean distance presented in equation 1 has been chosen.

$$\forall (x, y) \in \mathbb{R}^2, d(x, y) = \sqrt{x^2 - y^2} \tag{1}$$

By applying the distance to all the elements of $\mathbb{X}_i$, a square *distance matrix* $\in \mathcal{M}_n(\mathbb{R})$ can be obtained in equation 2.

$$\mathcal{G}(\mathbb{X}_j) = \begin{bmatrix} d(x_1^j, x_1^j) & d(x_1^j, x_2^j) & \dots & d(x_1^j, x_n^j) \\ d(x_2^j, x_1^j) & d(x_2^j, x_2^j) & \dots & d(x_2^j, x_n^j) \\ \vdots & \vdots & \vdots & \vdots \\ d(x_n^j, x_1^j) & d(x_n^j, x_2^j) & \dots & d(x_n^j, x_n^j) \end{bmatrix} \tag{2}$$

The nearest the points are to each other, the lesser the value of the distance is. Hence, there will be areas in the matrix filled with near zeros values if the original data present some clustering behavior. The symmetry of the *distance matrix* enables groups of data to be visually recognizable, and Euclidean distance will create square shapes of near null values along with the matrix diagonal. Let $\mathcal{G}_{j\in[\![1;p]\!]}$ be the matrix associated with each $\mathbb{X}_j$. Each of them will be processed as one channel RGB image. At this stage, one fundamental hypothesis is made :

Hypothesis 1. *The studied system cannot regenerate itself.*

Thus, every cluster found can be associated with a fault severity degree. As the clustering is done on time series, each new cluster will increase the severity value of the underlying default. The distance matrices defined in equation 2, are the stepping stone to extracting relevant clustering information. As shown in section 3, the processed data in our application is very noisy, and tested traditional signal processing techniques were not entirely effective in improving the signal over noise ratio. Hence, the deep learning approach has been chosen and applied with an image processing technique. The matrices will be considered as images whose contrast will be improved.

In this study, the computed Euclidean distances in the symmetric matrices make square shapes appear on their diagonals. Detecting shapes in images can be done in two steps:

- First, select a value for each class and assign the latter values to every pixel in the image. This process is called semantic segmentation. For example, we chose to assign the value 0 to the wanted square shapes and 1 to the background.
- Secondly, build a structure to reveal the different square shapes and eliminate the noisy background from images automatically, thus conserving only relevant information for clustering. To that extent, deep neural structures have been studied.

One drawback of such architectures is the need for a tremendous amount of training data. In industry, and specifically in aeronautics, relevant training data are scarce and costly to obtain. Consequently, [10] has developed an architecture, U-NET, for cell images segmentation specialized in small data samples to alleviate this limitation. Note that dataset dimension should be compared with usual ensembles used for training state-of-the-art neural network architectures. To give an order of magnitude, one could cite Microsoft's COCO dataset [11] and its 3×10^6 images.

The network U-NET is comprised of two main parts, as seen in Figure 1. The left part of the configuration compresses input and extracts deep features from it. The most profound features of all are processed through a bottleneck set of layers. Its result is fed through the bottom of the right part to be decompressed. Finally, depending on the number of classes to be segmented, either a *sigmoid* or a *softmax* layer is used to transform each processed input pixel value into the likelihood of belonging to a class. Each block has two convolution layers, for the descending part, associated with a *ReLU* activation ending with a *MaxPooling*. For the ascending part, the symmetrical structure is used made of *ConvTranspose2d*, an inverse convolution operation to up-sample its input, as developed in Pytorch [12].

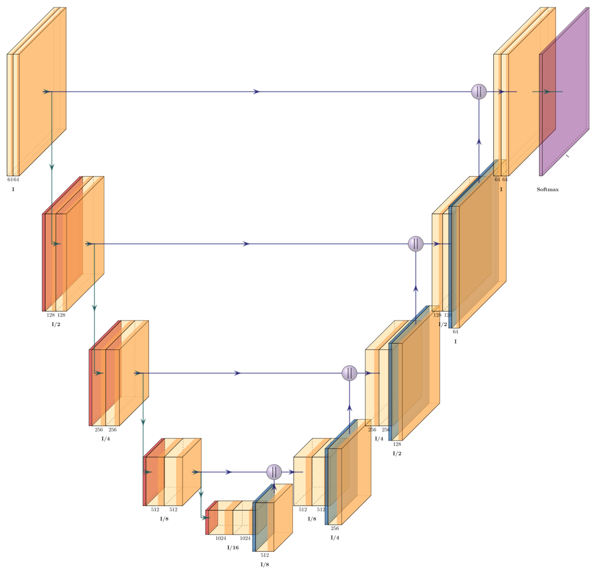


Figure 1. U-NET U-shaped architecture represented using algorithms from [13]

U-NET's goal, is to segment every distance matrix $\mathcal{G}_{j\in[\![1;p]\!]}$. As stated earlier, the network has to learn to recognize square patterns alongside each matrix diagonal. Consequently, it has to be trained to do so.

Like many industrial environments, aeronautics lacks an appropriate amount of high sampled training data. An artificial dataset has been generated to alleviate this problem. Four types of training patterns were produced, all sharing the same underlying structure: a set of squares with black ones along the diagonal.

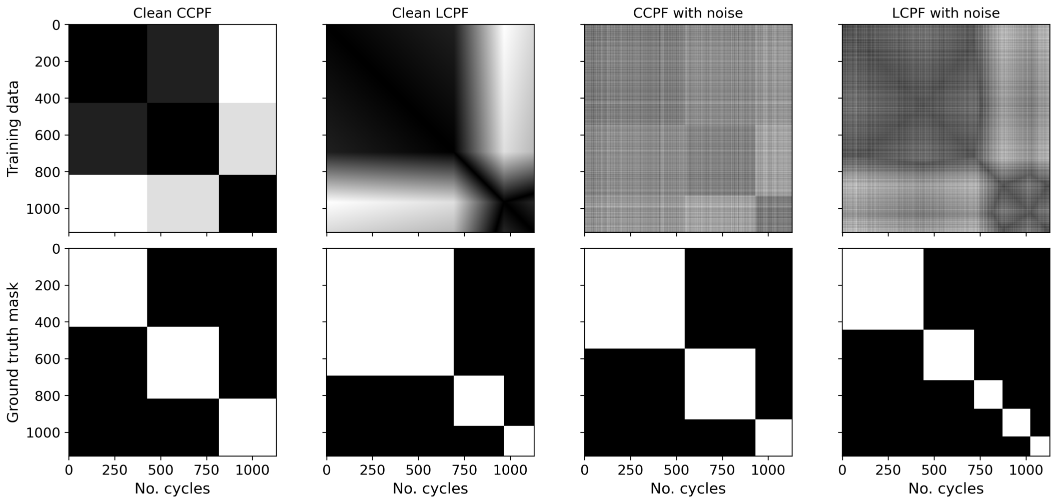


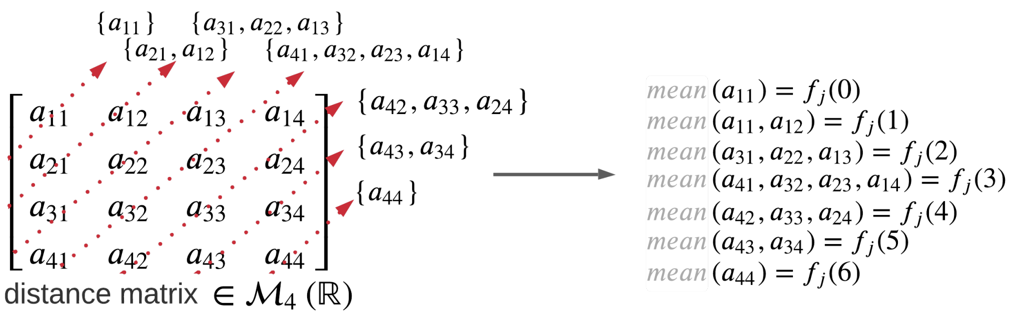
Figure 2. Distance matrices represented as images with several configurations, used for U-NET training with their associated ground truth.

The images from Figure 2 were randomly created by calculating the distance matrices of constant continuous piecewise functions (CCPF), linear continuous piecewise functions (LCPF), and adding Gaussian noise. The first row of Figure 2 represents input training data, whereas the second row represents its associated mask. At the end of each forward pass, the neural network will compare its output to the mask and backpropagate the gradient accordingly. The best training procedure was empirically determined. First, U-NET is trained with clean square images (CCPF). Once the loss function converged, the training of U-NET is reinforced by fuzzy data (LCPF). The operation is then repeated, and Gaussian noise is added to each input to further increase the neural network generalization capacity. Finally, a new dataset is generated with a field-specific bias to help the convergence of the training to the study goal. This bias is coming from another fundamental hypothesis made in this work:

Hypothesis 2. *The aging process is a compounding effect and can only accelerate through time.*

Consequently, the last batch of training data is generated with one constraint: squares alongside the matrix diagonal must have, decreasing dimensions. The latter part of the training reinforcement procedure has subsequently been done.

Once U-NET has been trained, it can produce segmented images from distance matrices. The segmented output is a binary image as all its pixels take either the value 0 or 1. However, as shown in Figure 8 of the results section 3, the neural network is still very noisy. Therefore, further signal processing procedures should be applied to extract a piece of relevant clustering information from it. The first attempt at data clustering can be directly extracted by looping through the output matrices anti-diagonal. It is a first candidate of a cluster frontier signal. The main idea is to recognize the area in each picture where square shapes terminate. Figure 3 visually illustrates the algorithm used to extract a first frontier cluster candidate. The "anti-diagonal" term used previously refers to the direction of each red arrow.



$\{a_{11}\}$
$\{a_{21}, a_{12}\}$
$\{a_{31}, a_{22}, a_{13}\}$
$\{a_{41}, a_{32}, a_{23}, a_{14}\}$
$\{a_{42}, a_{33}, a_{24}\}$
$\{a_{43}, a_{34}\}$
$\{a_{44}\}$
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{31} a_{32} a_{33} a_{34}
a_{41} a_{42} a_{43} a_{44}
distance matrix $\in \mathcal{M}_4(\mathbb{R})$
$mean(a_{11}) = f_j(0)$
$mean(a_{11}, a_{12}) = f_j(1)$
$mean(a_{31}, a_{22}, a_{13}) = f_j(2)$
$mean(a_{41}, a_{32}, a_{23}, a_{14}) = f_j(3)$
$mean(a_{42}, a_{33}, a_{24}) = f_j(4)$
$mean(a_{43}, a_{34}) = f_j(5)$
$mean(a_{44}) = f_j(6)$

Figure 3. Visual explanation of cluster frontier extraction - phase 1

As detailed in Algorithm 1, the matrix indices are fetched and store into a structure. Note that looping through the matrix in such a way produces an object of size $2n-1$ bags of items. Indeed, being of size n, the matrix upper left part will be covered in n iterations, including the maximum anti-diagonal size. It then remains $n-1$ iterations to get through the lower right part of the matrix. The mean is calculated for each ensemble of matrix values as stated in line 16, thus creating a resulting signal of size $2n-1$.

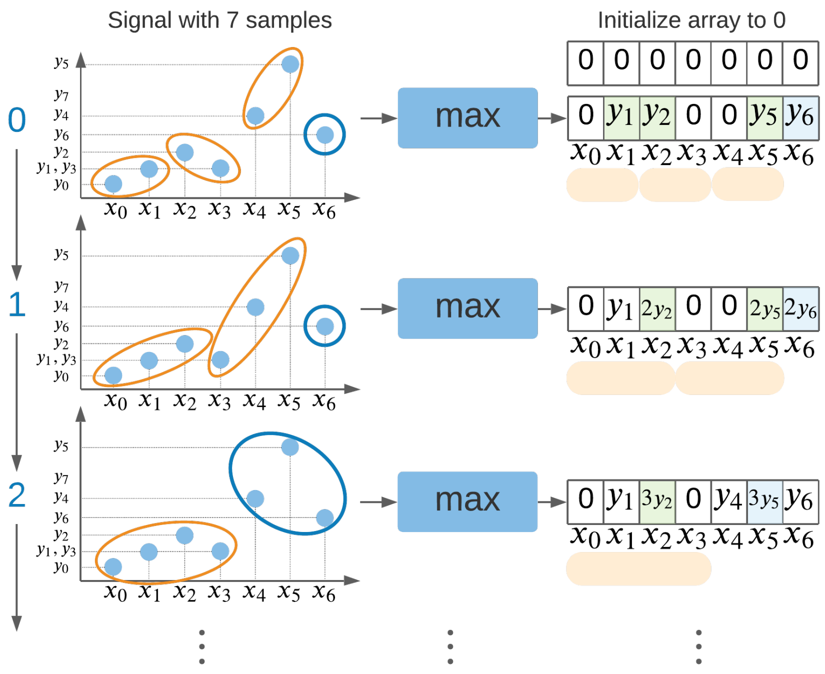
The mean has been chosen here because of the output pixels values. At the neural network output, pixels belonging to square shapes are encoded to 0, whereas pixels depicted in the image background are encoded to 1. By using the mean, a cluster frontier will appear when its value is shifted towards unity. On the opposite, the mean tending towards a null value will indicate the inside of a cluster.

The first phase of the cluster frontier extraction algorithm produces a signal with values lying between 0 and 1. The ones representing the potential cluster frontiers, local extrema have to be found to detect and further refine the clustering boundaries. Each peak in the created function represents a likely square contour in the segmented image.

Algorithm 1 Cluster frontier extraction - phase 1

```
1:  function GET INDICES OF MATRIX($n$)                          ▷ $n$ is matrix size
2:      declare indexArray of $[1; 2n-1]$ arrays                 ▷ Initialize array of size $2n-1$
3:      for $i \leftarrow 1$ to $n$ do                            ▷ get indices for upper left part of the matrix
4:          for $j \leftarrow 1$ to $i$ do
5:              indexArray$[i]$.append($[i-j+1; j]$)
6:      for $i \leftarrow n-1$ to 1 do                            ▷ get indices for other matrix part
7:          for $j \leftarrow i$ to 1 do
8:              indexArray$[2n-i]$.append($[j+1; i-j+2]$)
9:  return indexArray
10:
11: function CALCULATE INTERMEDIATE SIGNAL($M$)                   ▷ Matrix $M$ to process
12:     declare resArray of $[1; 2n-1]$ floats
13:     $n \leftarrow$ size(M)                                    ▷ M is a square matrix
14:     indexArray $\leftarrow$ GET INDICES OF MATRIX($n$)
15:     for $i \leftarrow 1$ to $2n-1$ do
16:         resArray$[i] \leftarrow$ mean(M[indexArray$[i]$])
17: return resArray
```

However, standard filtering methods require the non-trivial setting of parameters, such as the sliding window. The latter represents the number of points to which the filtering algorithm will be applied simultaneously. Hence, it controls the sensitivity of the algorithm to local and global dynamics. An iterative procedure has been created to avoid such shortcomings. It consists of looping through the signal to be filtered with an increased window size for each loop. Figure 4 represents the algorithm visually.



Signal with 7 samples
Initialize array to 0
0
1
2
max
y_5
y_7
y_4
y_6
y_2
y_1, y_3
y_0
x_0 x_1 x_2 x_3 x_4 x_5 x_6
0 0 0 0 0 0 0
0 y_1 y_2 0 0 y_5 y_6
0 y_1 $2y_2$ 0 0 $2y_5$ $2y_6$
0 y_1 $3y_2$ 0 y_4 $3y_5$ y_6

Figure 4. Visual explanation of cluster frontier extraction - phase 2: batch peak detection

The idea of the procedure is to split the input signal into several sets of values of increasing size throughout the loop. The minimum window is of size two, and it reaches a size n at the last iteration to cover the signal as a whole. Since cluster boundaries are detected for values approaching 1, the maximum is taken on each local window. An array of size $2n-1$ is initialized to 0 before the loop beginning to store each maximum found at its respective index in the input signal. Every time a maximum is locally found, it is added to the null array. Consequently, during initial iterations, the algorithm amplifies the local maxima, whereas, in the end, it tends to amplify the more global ones. For further details on how to realize this operation, the Algorithm 2 is given.

Algorithm 2 Cluster frontier extraction - phase 2: Batch Peak Detection

```
1: function BATCH PEAK DETECTION(signalArray)
2:     $n \leftarrow$ size(signalArray)
3:     declare resArray of $n$ floats
4:     for $i \leftarrow 1$ to $n$ do                ▷ fill arrays with zeros and index for initialization
5:         resArray[$i$] $\leftarrow 0$
6:     declare noWindowList of $n$ integers
7:     declare remainingSizeList of $n$ integers
8:     for $i \leftarrow 2$ to $n$ do            ▷ create array with all the windows and remaining sizes
9:         noWindowList[$i$] $\leftarrow i$
10:        remainingSizeList[$i$] $\leftarrow n \,\%\, i$                ▷ % represents the modulus operator
11:    for $i \leftarrow 0$ to $n-1$ do
12:        noWindow $\leftarrow n \,//\, i$
13:        remainElmt $\leftarrow$ remainingSizeList[$i$]
14:        if remainElmt $\neq 0$ then  ▷ suppose that language allows array broadcasting
15:            tmpArray $\leftarrow$ signalArray[: $-$remainElmt].reshape(noWindow, $i$)
16:        else
17:            tmpArray $\leftarrow$ signalArray.reshape(noWindow, $i$)
18:        declare indexStatValueList of unknown size of objects
19:        declare indexMaxArray of unknown size of floats
20:        for $j \leftarrow 0$ to noWindow do                                ▷ get an array
21:            indexMaxArray $\leftarrow$ arg( tmpArray[$i$, :] == max(tmpArray[$i$, :]))
22:            for $k \leftarrow 0$ to $size$(indexMaxArray) do
23:                indexStatValueList.append([$j$, indexMaxArray[$k$]) ▷ track maxima and
their corresponding indexes (like chained list)
24:        declare idxMaskArray of size $m \times l$ integers
25:        for $j \leftarrow 1$ to $m \times l$ do                          ▷ local initialization
26:            idxMaskArray[$j$] $\leftarrow 0$
27:        idxMaskArray.reshape($m, l$)
28:        for $k, p \leftarrow$ [1;noWindow] $\times$ [1; $i$] do                     ▷ create mask
29:            idxMaskArray[$k, p$] $\leftarrow 1$
30:        apply idxMaskArray to tmpArray
31:        unfold tmpArray of size (noWindow, $i$) to $(1, n)$
32:        for $j \leftarrow [1; m]$ do          ▷ Place maximum value at its corresponding index
33:            resArray$+ =$ tmpArray
    return resArray
```

Finally, since the resulting array is still of size $2n-1$, the abscissa scale of the signal is divided by a factor of two. This compensates the dilation generated by the method explained in Algorithm 1. It is assumed that uncertainty of two cycles is tolerated for the health monitoring framework. The resulting signals are noted $f_{j\in[\![1;p]\!]} \in \mathcal{M}_{n1}(\mathbb{R})$. Note that the combination of Algorithm 1 and Algorithm 2 does not require the choice of any threshold or hyperparameter value.

At this stage of the method, p time series have been transformed into p distance matrices $\mathcal{G}_j$, and the neural network has segmented each matrix. From each output picture, p first candidates to cluster boundaries signal are obtained, every one of them is filtered through batch peak detection algorithm 2. Hence, p new time series $\mathcal{F}_j$ have been created. These could be concatenated into a matrix $\mathcal{F}$ as represented in equation 3.

$$\mathcal{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & \cdots & f_{1p} \\ f_{21} & f_{22} & f_{23} & \cdots & f_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{n1} & f_{n2} & f_{n3} & \cdots & f_{np} \end{bmatrix} \in \mathcal{M}_{np}(\mathbb{R}) \tag{3}$$

The developed method assumes that each descriptor contributes to the clustering result. Therefore, this information has to be fused to get a global final frontier signal.

The principle of kernel density estimation (KDE) [14] is used to do so. By considering each column of $\mathcal{F}$ to realize a random process, the KDE can be applied to the entire matrix. Consequently, the initial p descriptors are themselves considered as random variables. Once these hypotheses are made, the distribution of each $f_{.j}$ gives insight into their respective theoretical probability density. With assumptions made about kernel functions, as presented by [15, Theorem 6.7], the KDE will construct an empirical density that will converge to its theoretical result. By considering the p temporal segments as realizations of several independent and identically distributed random processes, it is coherent to sum all columns of $\mathcal{F}$ defined in equation 3 to obtain a vector:

$$(y_i)_{1\leq i\leq n} = \sum_{j=1}^{p} f_{ij} \tag{4}$$

From [15, Chapter 6, Equation 6.1], the empirical density can be expressed in equation 5.

$$\hat{p}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-y_i}{h}\right) \tag{5}$$

K is a kernel function in equation 5, applied to the centered variable of the sample time series and scaled by a factor h. Even if the Gaussian kernel defined in equation 6 is not a computationally optimized one, it is chosen for its theoretical properties.

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp^{-\frac{1}{2}x^2} \tag{6}$$

The best property in this context is the ability to chose a simple bandwidth parameter from state-of-the-art. As detailed in [16, Equation 3.31], the bandwidth h of a Gaussian kernel can be expressed as $h = 0.9An^{-1/5}$, with the constant $A = \min(\sigma, \text{inter-quartile range}/1.34)$.

Finally, gathering all the steps in this section 2, a function estimating the empirical likelihood of cluster frontiers has been created for the studied dataset.

3. Results

In this second part, the clustering method developed in section 2 is applied to monitoring vibration signals generating by an induction motor driving a ball screw. After presenting the dataset used, several signals will be selected to test the previous theoretical part. Finally, our method will be compared to other state-of-the-art clustering methods.

3.1. Dataset and studied system

The dataset came from four different electromechanical actuators, each made of an asynchronous electrical machine and a ball screw. A controlled test bench was constructed to monitor the vibration behavior of each actuator. The goal of the procedure was to detect any precursor of a jamming defect in each ball screw of the different actuators. That is why an accelerometer was placed on every monitored structure, as shown in Figure 5. This configuration allowed the collection of vibration signals coming from the longitudinal axis of the actuation device.

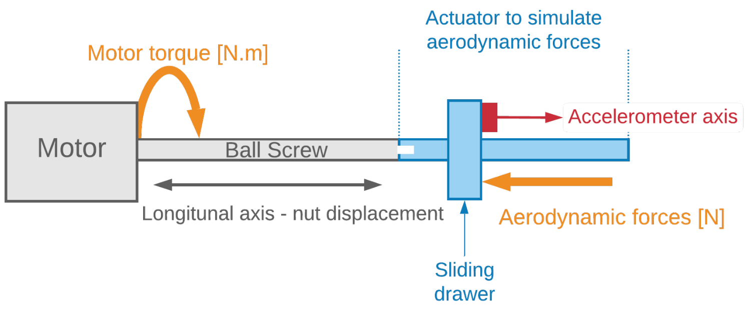


Figure 5. Schematics of the instrumented actuator bench

Every actuator was stressed through 1130 cycles specially designed to emulate a realistic environment. With this protocol, everyone has reached the end of life at the last cycle. The signal measured was sampled at 1 kHz, standardized, and normalized between 0 and 1.

3.2. Time series encoding

As stated in section 2, p statistical and model-based descriptors are computed from the raw data measured on the monitored actuator. Every descriptor $\mathbb{X}_j$ has 1130 samples. Among all p features, four were chosen. They will be referred to as $\{\mathbb{X}_1, \mathbb{X}_2, \mathbb{X}_3, \mathbb{X}_4\}$. Each has been standardized and normalized between 0 and 1. They are represented in Figure 6. The four descriptors were selected to evaluate the method on signals that exhibit various dynamics but are still coherent for extracting a health monitoring information extraction. Obviously, from Figure 6, the features are rather noisy and not strictly monotonic. However, a general ascending or descending trend is recognizable.

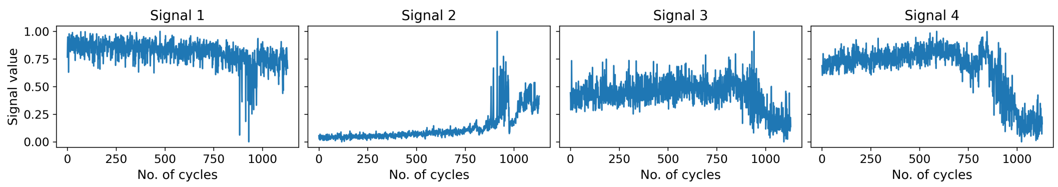


Figure 6. $\{\mathbb{X}_1, \mathbb{X}_2, \mathbb{X}_3, \mathbb{X}_4\}$ collected on our actuator.

Using the Euclidean distance, four distance matrices $\{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4\}$ are calculated from the previous four statistical features. They are presented in Figure 7.

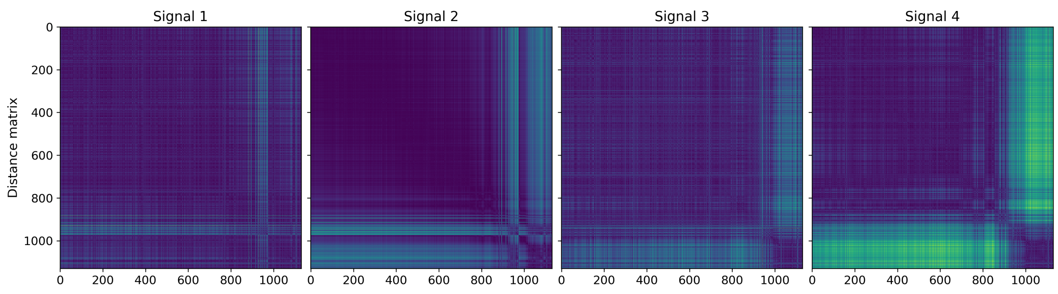


Figure 7. Distance matrices $\{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4\}$ extracted from $\{\mathbb{X}_1, \mathbb{X}_2, \mathbb{X}_3, \mathbb{X}_4\}$.

In Figure 7, each pixel corresponds to one value of the distance matrix. Note that pictures have been colorized to better highlight symmetries, but they are in grayscale with original values between 0 and 1. Here, the darker the pixel is, the lower the represented distance is. Thereby, as stated in section 2, potential clusters are represented as dark symmetrical shapes along with each picture diagonal. Indeed, those are the areas where the distance is minimal between points. Outside of it, the distance increases; hence a potential cluster limit can be found. Moreover, horizontal and vertical stripes in the images came from the noise of input data. To further illustrate the concept, at least two probable clusters in signal four of Figure 7 can be seen.

3.3. Semantic segmentation

U-NET was trained with an artificial dataset containing 3000 images. A glimpse of its content can be seen in Figure 2. The data were randomly split for the learning phase and shuffled into 2250 training pictures and 750 testing ones. As presented in section 2, four types of artificial signals are equally present in the data set: signals with successive constant stages, signals with successive linear stages, and an adaptation of those signals with a white noise of ten percent. A study was carried out to determine the best learning hyper-parameters. For the learning rate, the adaptive scheduler *ReduceLROnPlateau* from package *optim.lr_scheduler* [12] with an initial value of 1×10^{-4} was selected. After monitoring the learning phase of U-NET with *tensorboard* [17], 200 epochs were sufficient to reach a stable accuracy. Because of a limited GPU memory bandwidth, we had to limit ourselves to a batch size, a hyperparameter, of 4 images for training. The input is an image of size 1130×1130; consequently, it rapidly saturates the VRAM during training. Finally the loss function used was *BCEWithLogitsLoss* from the *torch.nn* package [12]. Note that the learning phase is done only once.

The results of Figure 8 were obtained after 38 h of training on HPC resources with a bi-Intel Xeon Silver 4215R,a bi-NVIDIA Quadro RTX 6000 with 24 GB of GDDR6 each and 192 GB of RAM. U-NET outputs binary images. Purple corresponds to null values and yellow corresponds to ones.

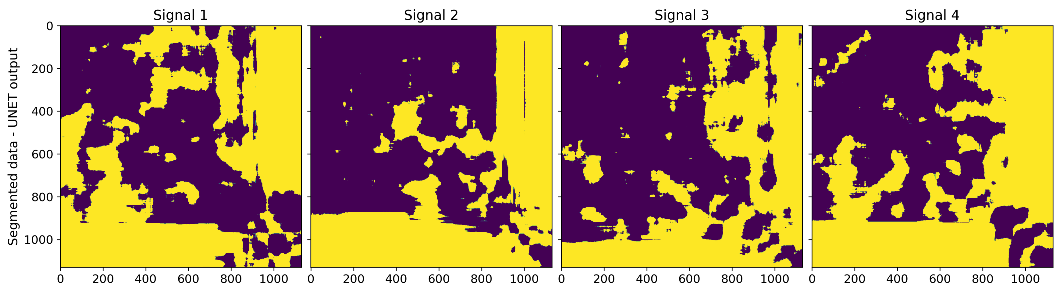


Figure 8. Segmented $\{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4\}$ after U-NET inference phase.

From Figure 8 it is clear that U-NET's training phase could be improved. Despite the noisy results, the latter signal processing algorithms were designed to alleviate this problem.

3.4. Cluster frontier extraction - first phase

By applying the algorithm 2 of section 2 to each $\mathcal{G}_\rangle$ of $\{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4\}$, the signals in Figure 9 are extracted.

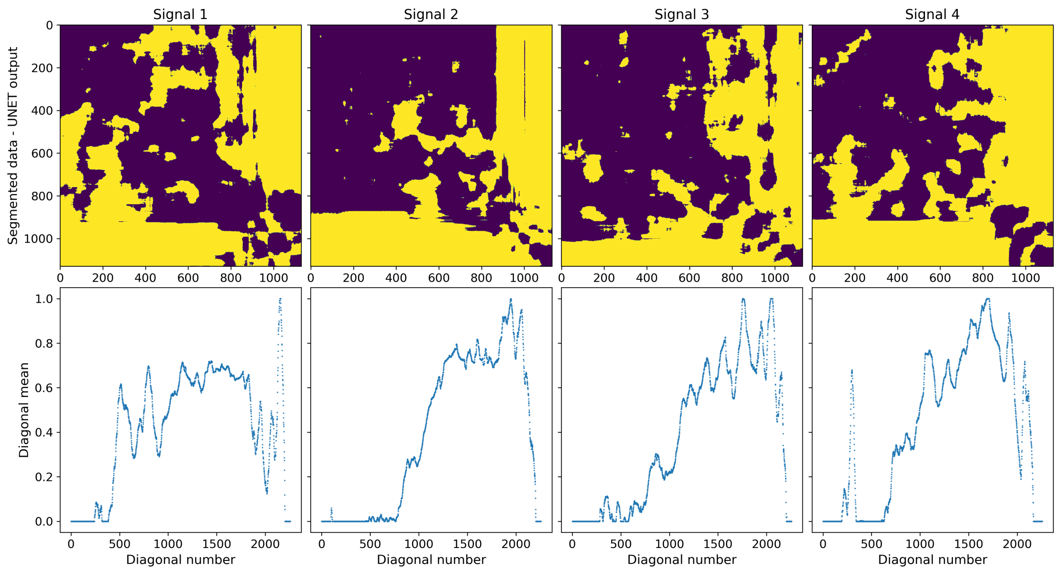


Figure 9. Mean of the values (second row) of the segmented distance matrices (first row) on each anti-diagonal.

The segmented images being of size 1130×1130, the temporary frontier signal is of size $2 \times 1130 - 1$, thus of size 2259. As stated in section 2, the local maxima of these signals have to be found. If an anti-diagonal contains only yellow pixels, the signal will reach 1.

3.5. Cluster frontier extraction - second phase

The cluster frontier extraction, the *batch peak detection* algorithm 2 of section 2, is applied to each signal obtained at the previous phase. This current phase consists of local magnifying maxima in the signal to get the first discrete potential cluster boundaries. The dimension of *batch peak detection* results is then shrunk by a factor of 2 to obtain the final signal of Figure 10. It now contains only 1130 values: one for each actuator cycle.

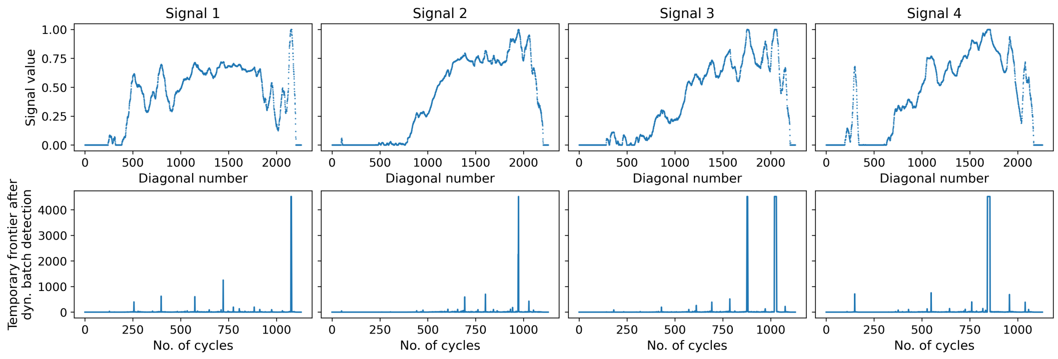


Figure 10. Frontier extraction (second row) from signal computed in phase 1.

From Figure 10, cluster frontiers are mainly concentrated after the 750th cycle. This behavior is consistent with motifs from Figure 7, where the discontinuity in signal essentially occurs at the bottom right of each image.

3.6. Cluster frontier extraction - third phase

In order to obtain a continuous signal out of the frontier extraction from phase two, we use the Gaussian kernel density estimation of *scipy.stats.gaussian_kde* with Silverman [16] bandwidth method. Then, we sum the four frontier extraction signals from Figure 10, and we compute the Gaussian Kernel Density Estimation on this global frontier extraction to obtain Figure 11.

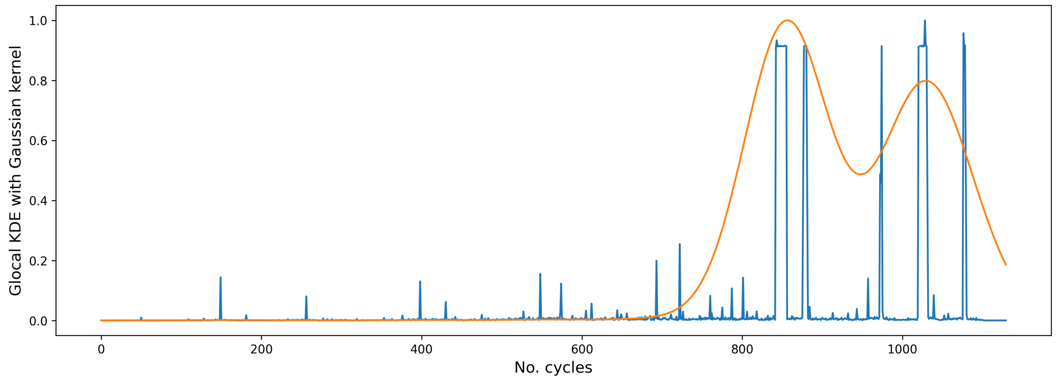


Figure 11. Gaussian kernel density estimation computed with the frontier extraction results of all four statistical parameters.

Each local maximum of the density function is likely to be a cluster frontier. Hence, two cluster frontiers are detected in Figure 11: a first one at cycle 856, a second at cycle 1029. It means we are likely to have three clusters. The first cluster from cycles 0 to 855 which corresponds to the first degree of severity where the system is healthy. The second cluster from cycle 856 to cycle 1028 is where the system is in the second stage of fault severity. The last cluster starting at cycle 1029 is the third and worst stage of fault severity.

The two independent components of the density estimation of Figure 11 are computed.

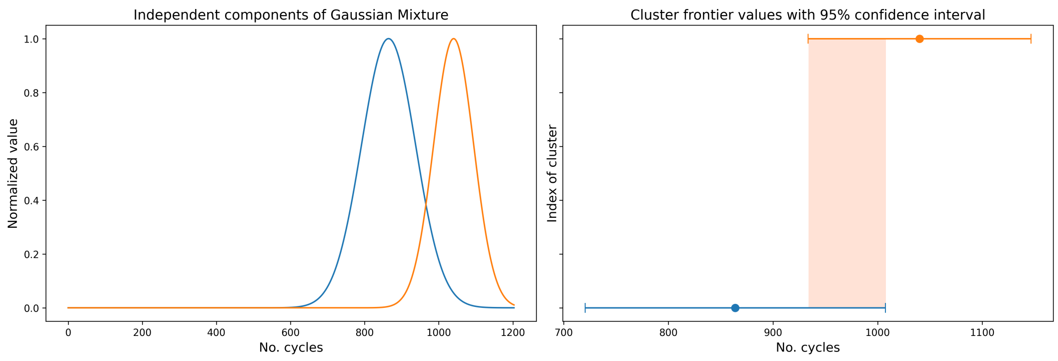


Figure 12. Cluster frontiers with 95% confidence interval

3.7. Clustering results

In order to represent the clustering results on the four descriptors, we consider that each local maximum of the density function is a cluster frontier. Figure 13 is obtained.

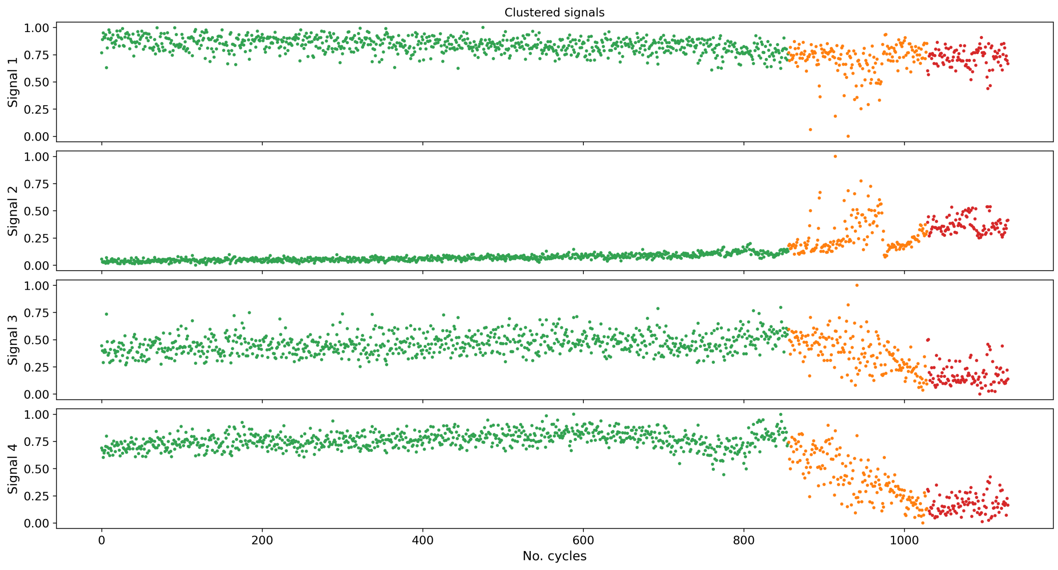


Figure 13. Clustering results of initial signals from Figure 6

In Figure 13, each cluster has a different color corresponding to the health stage of the actuator. The first cluster is green, the second cluster is orange, and the last cluster is red. One can notice that the clusters obtained are coherent with the shapes of the signals.

4. Discussion

In this subsection, we compare our clustering method to other previous state-of-the-art algorithms. The latter are summarized in Table 1. They are taken from the Python library *scikit-learn* [18] except *Kmeans* which comes from the *tslearn* [19] package.

Table 1. Clustering methods considered in our study

Clustering Method	Category	Metric	FunctionName	Parameters
K-Means	Partitioned	metric='euclidean'	KMeans	max_iter=200 n_jobs=-1 init='k-means++' n_init=10
		metric='dtw'		
K-medoids	Paritional	metric='euclidean'	KMedoids	max_iter=200 init='k-medoids++'
Mean Shift	Partitional	/	MeanShift	max_iter=200 n_jobs=-1
OPTICS	Density-based	metric='euclidean'	OPTICS	n_jobs=-1 min_samples=3
		metric='minkowski'		
Agglomerative	Hierarchical	metric='euclidean'	Agglomerative Clustering	min_cluster_size=50
		metric='manhattan'		
Gaussian Mixture Model	Model-based	cov='spherical'	GaussianMixture	max_iter=1000 n_init=100
		cov='diag'		
		cov='full'		
		cov='tied'		

We have selected these methods according to two criteria. The first being the ease of implementation to our benchmark scope, as they are present in standard machine learning libraries in Python. The second is the representativeness of the diversity of the clustering algorithms. Indeed, according to [20, Fig 6.], they can be separated into six different approaches. We tried to cover at least three of them: partitional, hierarchical, and model-based. Each method in Table 1 was fed a data structure with five columns and 1130 lines. Four of the columns were the concatenation of $\mathbb{X}_{i\in[\![1;4]\!]}$, and the fifth one was an index array ranging from 0 to 1129 representing time. The same $\mathbb{X}_i$ of Figure 6 were used. Thus, it allows each algorithm to be aware of the time precedence relation between points. Each clustering method fits the data and then predicts the clustering labels. The labels are then stored in a matrix to represent clustering results.

Since the clustering methods use the four $\mathbb{X}_i$ to produce one clustering result, we can only represent the algorithm outcome for one signal. Figure 14 plots the clustered first signal. We chose the number of groups in data according to a new quality indicator.

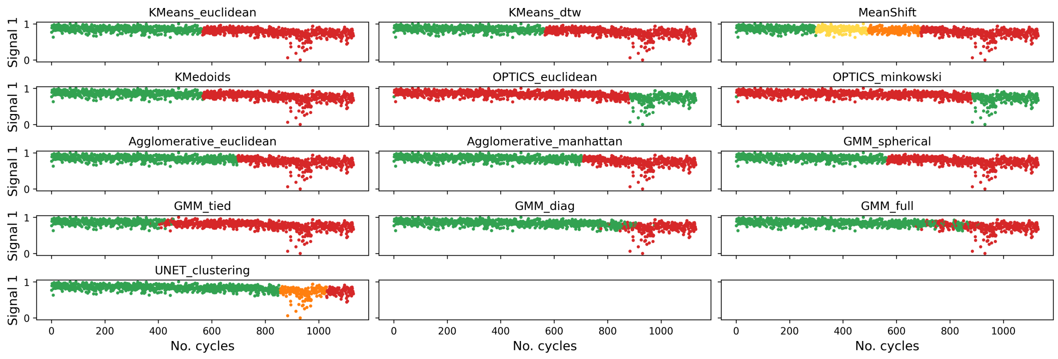


Figure 14. Clustering results with state-of-the-art algorithms for $\mathbb{X}_1$

In Figure 14, the majority of the methods have detected two clusters except for *MeanShift* and our clustering method *UNET clustering*. One detects four groups, whereas ours provides three groups in data, as seen previously in Figure 13. Regarding the shape of signal $\mathbb{X}_1$, three clusters seem to be the optimal number embedded in the data.

To evaluate the accuracy of our method regarding the state-of-the-art, we realized a new clustering indicator. Indeed, whereas indicators for spatial clustering are well known, specific existing indicators to infer clustering quality for internal time series clustering are scarce. We can cite the *S_Dbw* [21] to measure the compactness and the good separation of clusters or the exhaustive review of [20] on time series clustering. Also, recently, an invariance guided criterion has been created by [22]. Nevertheless, these indexes can be separated into two main groups: the external and the internal ones [23]. Since we do not have class labels for our data, we only must count on information contained in the studied data as stated in [23] study. Eleven widely used indicators are compared. In our work, we tested *Silhouette* [24], the Davies-Bouldin separation measure [25], the Calinski Harabasz [26] indicator, the conjunction of an inertia and temporal consistency measure from [27], and finally the SD [28] and *S_dbw* [21] validity indexes. None of them could manage to select the right clustering result in Figure 14. As those are not fitted to the task, our measure should quantify relevant information according to the physics of the system. On the first hand, as stated in fundamental hypothesis 1, the studied system cannot be repaired during its life cycle. Once the severity degree has increased, i.e., once a cluster frontier had been crossed, the system can not go back to a previous healthier stage. In other words, each class must be time continuous. Temporal jumps in clusters are forbidden. Consequently, The indicator must penalize any temporal inconsistency. At the second end, we can see in Figure 14 that good clustering is made when two consecutive groups do not share the same density properties. This shape information also has to be captured.

Equation 7 presents the newly developed quality indicator for internal clustering of time series.

$$\chi = \frac{1}{m}\sum_{j=0}^{m-1}\left|a_{j+1} - a_j\right|, \text{ with} \tag{7}$$

$$a_j = \frac{\sigma_j(\max y_j - \min y_j)}{\dfrac{1}{n_j - 1}\sum\limits_{\substack{k=0\\ k\in\mathcal{J}}}^{n_j-1} t_{k+1} - t_k} \tag{8}$$

With m the number of clusters found in the data, σ_j the standard deviation of the jth cluster, $\max y_j - \min y_j$ the range of the jth cluster, y_j its elements and t_k its respective indexes and n_j its number of points. Note that $\mathcal{J}$ is the ensemble of the point indexes in cluster j. While the denominator of a_j measures the temporal consistency, the numerator measure the shape of the clusters.

Finally, the quality index χ deals with the variation between two consecutive groups and not a_j in itself. Note that the denominator in a_j can never be null for time series. Indeed, we can see in Figure 14 for our algorithm that the first and third data partitions exhibit similar dynamics. However, since the system cannot rejuvenate itself, those should be considered as two different groups. To recognize the better partitioning result, the value $\mathcal{I}$ defined in Equation 7 has to be maximized.

The ideal number of clusters in Figure 14 was selected according to the results of the Table 2. Note that our algorithm gets the global maximum of the array. From these results, we can extract two main trends. There are some indicator values in the neighborhood of the global maximum 0.083: *KMeans_euclidean*, *KMeans_dtw*, *KMedoids*, *Agglomerative_euclidean*, *Agglomerative_manhattan* and *GMM_spherical*. This result is coherent with the presented data as these algorithms tend to cluster data similarly, thus all the partitioning algorithms have very similar indicator values. Moreover it seems that the chosen distance does not have a significant impact on the results.

Table 2. Global indicator χ value depending on the clustering method and the number of clusters

	Number of clusters						
	2	**3**	**4**	**5**	**6**	**7**	**8**
KMeans_euclidean	0.074	0.051	0.034	0.025	0.026	0.035	0.032
KMeans_dtw	0.074	0.051	0.036	0.025	0.026	0.035	0.031
MeanShift			0.040				
KMedoids	0.074	0.051	0.034	0.026	0.027	0.037	0.032
OPTICS_euclidean	0.008						
OPTICS_minkowski	0.008						
Agglomerative_euclidean	0.073	0.050	0.024	0.020	0.018	0.019	0.017
Agglomerative_manhattan	0.074	0.051	0.038	0.032	0.027	0.030	0.027
GMM_spherical	0.074	0.051	0.034	0.025	0.026	0.034	0.032
GMM_tied	0.068	0.021	0.025	0.021	0.012	0.011	0.009
GMM_diag	0.041	0.027	0.027	0.023	0.013	0.011	0.011
GMM_full	0.049	0.034	0.026	0.023	0.028	0.023	0.021
UNET_clustering		0.083					

To better grasp the idea of the indicator, we have separately given Table 3 which represents the contribution of the *shape* measure to χ and Table 4 which represents the temporal consistency one. Practically, Table 3 represents the values of $\sigma_j(\max y_j - \min y_j)$ and Table 4 represents the values of $\frac{1}{n_j-1}\sum\limits_{k=0}^{n_j-1} t_{k+1} - t_k$ for each clustering results. From that we can see that the *shape* part of the indicator is coherent with the results found in Figure 14 as different clustering behavior gets different values. From Equation 7, one could infer that there is no temporal inconsistency. This proved by the unit value of the denominator. The problem with the methods like *GMM_tied*, *GMM_diag*, *GMM_full*, *OPTICS_euclidean* and *OPTICS_minkowski* is that they clusters are not time continuous. Hence the value in Table 4 is greater than one. Thus the numerator of a_j is divided by an amount greater than one which minimize its value. This penalization is then propagated in the total calculus of χ.

Table 3. Shape score $\sigma_j(\max y_j - \min y_j)$ depending on the clustering method and the number of clusters

	Number of clusters						
	2	**3**	**4**	**5**	**6**	**7**	**8**
KMeans_euclidean	0.074	0.051	0.034	0.025	0.026	0.035	0.032
KMeans_dtw	0.074	0.051	0.036	0.025	0.026	0.035	0.031
MeanShift			0.040				
KMedoids	0.074	0.051	0.034	0.026	0.027	0.037	0.032
OPTICS_euclidean	0.055						
OPTICS_minkowski	0.055						
Agglomerative_euclidean	0.073	0.050	0.024	0.020	0.018	0.019	0.017
Agglomerative_manhattan	0.074	0.051	0.038	0.032	0.027	0.030	0.027
GMM_spherical	0.074	0.051	0.034	0.025	0.026	0.034	0.032
GMM_tied	0.072	0.037	0.028	0.023	0.018	0.016	0.015
GMM_diag	0.061	0.042	0.030	0.026	0.039	0.035	0.031
GMM_full	0.068	0.036	0.028	0.025	0.040	0.037	0.033
UNET_clustering		0.083					

Table 4. Temporal consistency score depending on the clustering method and the number of clusters

	Number of clusters						
	2	**3**	**4**	**5**	**6**	**7**	**8**
KMeans_euclidean	1.000	1.000	1.000	1.000	1.000	1.000	1.000
KMeans_dtw	1.000	1.000	1.000	1.000	1.000	1.000	1.000
MeanShift			1.000				
KMedoids	1.000	1.000	1.000	1.000	1.000	1.000	1.000
OPTICS_euclidean	2.785						
OPTICS_minkowski	2.785						
Agglomerative_euclidean	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Agglomerative_manhattan	1.000	1.000	1.000	1.000	1.000	1.000	1.000
GMM_spherical	1.000	1.000	1.000	1.000	1.000	1.000	1.000
GMM_tied	1.075	1.310	1.197	1.268	1.919	1.804	1.789
GMM_diag	1.259	1.758	1.903	2.503	2.700	3.002	2.980
GMM_full	1.247	1.472	1.633	1.746	1.609	1.889	1.761
UNET_clustering		1.000					

5. Conclusions

This paper developed a new clustering method for time series with a deep neural network core. The whole method consists of three main steps. First, temporal data are encoded into pictures in order to be fed to the DNN. Then, the segmented images are processed with consistent signal processing algorithms. Finally, the information from all the descriptors is fused to obtain the general data partitioning profile. The whole algorithm empowers us to find cluster frontiers with a likelihood measure from an ensemble of time series. We designed it specifically to consider the precedence relation of elements and to avoid temporal consistency incoherence. Note that the whole algorithm could be applied to multivariate time series as data nature is irrelevant to its results.

However, it requires more significant computing resources than its counterpart to train the deep structure. But it is worth the cost as the neural network and the signal processing operations following the image segmentation can cluster very noisy data. Besides, the better the deep structure training is, the better the clustering results will be.

Furthermore, to quantify the results of this new method against state-of-the-art machine learning, we have created a new indicator. The latter can evaluate two properties: the temporal consistency of clustering and the shape of the cluster. We designed it with our physical assumptions. Hence it exhibits better results from its state-of-the-art counterpart.

Finally, this whole method is a stepping stone to a more broad framework aiming to determine the remaining useful life of actuators in a PHM framework.

Author Contributions: Conceptualization, A.E., G.C. and B.M.; methodology, A.E., G.C. and B.M.; software, A.E. and S.R.; validation, A.E.; formal analysis, A.E.; investigation, A.E.; resources, B.M.; data curation, A.E.; writing—original draft preparation, A.E, G.C., B.M. and S.R.; writing—review and editing, A.E., G.C. and B.M.; visualization, A.E. and S.R.; supervision, G.C. and B.M.; project administration, G.C. and B.M.; funding acquisition, B.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the French agency for research and technologye (ANRT) through the CIFRE grant 2018/0649, by Safran Electronics & Defense (Safran Group) and by laboratoire Ampère CNRS - UMR5005own funds.

Data Availability Statement: 3rd Party Data. Restrictions apply to the availability of these data. Data was obtained from Safran Electronics & Defense and are not publicly available.

Acknowledgments: This work was supported by the PMCS2I from École Centrale de Lyon, France

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CCPF	Constant Continuous Piecewise Function
DL	Deep Learning
DNN	Deep Neural Network
GDDR	Graphics Double Data Rate
GPU	Graphics Processing Unit
HPC	High Performance Computing
IATA	International Air Transport Association
MCTG	Maintenance Cost Technical Group
MRO	Maintenance Repair and Overhaul
KDE	Kernel Density Estimation
LCPF	Linear Continuous Piecewise Function
PHM	Prognosis and Health Management
RAM	Random Access Memory
VRAM	Video Random Access Memory

References

1. Fink, O.; Wang, Q.; Svensén, M.; Dersin, P.; Lee, W.J.; Ducoffe, M. Potential, challenges and future directions for deep learning in prognostics and health management applications. *Engineering Applications of Artificial Intelligence* **2020**, *92*, 103678. doi:10.1016/j.engappai.2020.103678.
2. Wang, Z.; Oates, T. Spatially Encoding Temporal Correlations to Classify Temporal Data Using Convolutional Neural Networks. *arXiv:1509.07481 [cs]* **2015**. arXiv: 1509.07481.
3. Hatami, N.; Gavet, Y.; Debayle, J. Classification of Time-Series Images Using Deep Convolutional Neural Networks. *Proceedings of SPIE, the International Society for Optical Engineering* **2018**, *10696*, UNSP 106960Y. Publisher: SPIE, The International Society for Optical Engineering, doi:10.1117/12.2309486.
4. Jain, A.K.; Murty, M.N.; Flynn, P.J. Data clustering: a review. *ACM Computing Surveys* **1999**, *31*, 264–323. doi:10.1145/331499.331504.

5. Hendrickx, K.; Meert, W.; Mollet, Y.; Gyselinck, J.; Cornelis, B.; Gryllias, K.; Davis, J. A general anomaly detection framework for fleet-based condition monitoring of machines. *Mechanical Systems and Signal Processing* **2020**, *139*, 106585. doi:10.1016/j.ymssp.2019.106585.
6. Perafán-lópez, J.C.; Sierra-pérez, J. An unsupervised pattern recognition methodology based on factor analysis and a genetic-DBSCAN algorithm to infer operational conditions from strain measurements in structural applications. *Chinese Journal of Aeronautics* **2020**. doi:10.1016/j.cja.2020.09.035.
7. Zaporowska, A.; Liu, H.; Zakwan, S.; Yifan, Z. A clustering approach to detect faults with multi-component degradations in aircraft fuel systems. *IFAC-PapersOnLine* **2020**, *53*, 113–118. Publisher: Elsevier, doi:10.1016/j.ifacol.2020.11.018.
8. Chen, J.; S. Chen.; Z. Liu.; C. Luo.; Z. Jing.; Q. Xu. Health Monitoring of Landing Gear Retraction/Extension System Based on Optimized Fuzzy C-Means Algorithm. *IEEE Access* **2020**, *8*, 219611–219621. doi:10.1109/ACCESS.2020.3042888.
9. Javed, A.; Lee, B.S.; Rizzo, D.M. A benchmark study on time series clustering. *Machine Learning with Applications* **2020**, *1*, 100001. doi:10.1016/j.mlwa.2020.100001.
10. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]* **2015**. arXiv: 1505.04597.
11. Lin, T.Y.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramanan, D.; Zitnick, C.L.; Dollár, P. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]* **2015**. arXiv: 1405.0312.
12. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32*; Wallach, H.; Larochelle, H.; Beygelzimer, A.; dAlché Buc, F.; Fox, E.; Garnett, R., Eds.; Curran Associates, Inc., 2019; pp. 8024–8035.
13. Iqbal, H. HarisIqbal88/PlotNeuralNet, 2020. Programmers: _:n2500 original-date: 2018-07-24T16:51:34Z.
14. Parzen, E. On Estimation of a Probability Density Function and Mode. *Annals of Mathematical Statistics* **1962**, *33*, 1065–1076. Publisher: Institute of Mathematical Statistics, doi:10.1214/aoms/1177704472.
15. Scott, D.W. *Multivariate Density Estimation: Theory, Practice, and Visualization, Second Edition*; Wiley Series in Probability and Statistics, John Wiley & Sons, Ltd, 2015. Section: 6 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118575574.ch6, doi:10.1002/9781118575574.ch6.
16. Silverman, B.W. *Density Estimation for Statistics and Data Analysis*; CRC Press, 1986.
17. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.
18. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
19. Tavenard, R.; Faouzi, J.; Vandewiele, G.; Divo, F.; Androz, G.; Holtz, C.; Payne, M.; Yurchak, R.; Rußwurm, M.; Kolar, K.; Woods, E. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research* **2020**, *21*, 1–6.
20. Aghabozorgi, S.; Seyed Shirkhorshidi, A.; Ying Wah, T. Time-series clustering – A decade review. *Information Systems* **2015**, *53*, 16–38. doi:10.1016/j.is.2015.04.007.
21. Halkidi, M.; Vazirgiannis, M. Clustering validity assessment: finding the optimal partitioning of a data set. Proceedings 2001 IEEE International Conference on Data Mining, 2001, pp. 187–194. doi:10.1109/ICDM.2001.989517.
22. Forest, F.; Mourer, A.; Lebbah, M.; Azzag, H.; Lacaille, J. An Invariance-guided Stability Criterion for Time Series Clustering Validation. 2020 25th International Conference on Pattern Recognition (ICPR), 2021, pp. 9296–9303. ISSN: 1051-4651, doi:10.1109/ICPR48806.2021.9412020.
23. Liu, Y.; Li, Z.; Xiong, H.; Gao, X.; Wu, J. Understanding of Internal Clustering Validation Measures. 2010 IEEE International Conference on Data Mining, 2010, pp. 911–916. ISSN: 2374-8486, doi:10.1109/ICDM.2010.35.
24. Rousseeuw, P.J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* **1987**, *20*, 53–65. doi:10.1016/0377-0427(87)90125-7.
25. Davies, D.L.; Bouldin, D.W. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1979**, *PAMI-1*, 224–227. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, doi:10.1109/TPAMI.1979.4766909.
26. Caliński, T.; Harabasz, J. A dendrite method for cluster analysis. *Communications in Statistics* **1974**, *3*, 1–27. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/03610927408827101, doi:10.1080/03610927408827101.
27. Breuneval, R. Surveillance de l'état de santé des actionneurs électromécaniques : application à l'aéronautique. Thèse de doctorat, Université de Lyon, Lyon, 2017.
28. Halkidi, M.; Vazirgiannis, M.; Batistakis, Y. Quality Scheme Assessment in the Clustering Process. PKDD, 2000. doi:10.1007/3-540-45372-5_26.

Short Biography of Authors

Alexandre Eid is born in 1993 in France. He gets his engineering degree in computer science/IT and control from Grenoble INP - Esisar in 2018. Its final year project in the aeronautic industry is a stepping stone to begin a Ph.D. thesis in computer science applied to prognosis and health management. He currently works at the University Claude Bernard Lyon 1 and Safran aeronautic group. He focuses his research on data science and artificial intelligence algorithm applied to time series.

Guy Clerc is Professor and senior member of the Institute of Electrical and Electronics Engineers (IEEE). He has obtained the engineer grade of École Centrale de Lyon in 1984 and the PHD of École Centrale de Lyon in 1989. He has been Head of the Ampère Laboratory (UMR 5005) since April 2013 until January 2016 and Delegate Vice president for research in University Claude Bernard Lyon 1 since April 2016 until April 2020. He teaches Electrical Engineering at the Claude Bernard University of Lyon-I (design, control and modelling of electrical machines). He carries out research on the diagnosis of AC machines and on electrical urban transport at the Ampère Laboratory, a joint research unit to École Centrale de Lyon, the Claude Bernard University of Lyon-I, INSA de Lyon and the French CNRS.

Badr Mansouri received the Ph.D. degree in automatic control from the University of Reims Champagne Ardenne, in 2005. He is currently a flight control system engineer at Safran Electronics & Defense. His main works relate to the modeling of dynamic systems, automatic control and prognostics and health management of electromechanical systems.

Stella Roux is a second year student at engineering school Grenoble INP - Ensimag of applied mathematics and computing. She specializes in Mathematical Modelling, Image and Simulation. She worked at the University Claude Bernard Lyon 1 for a 2 months internship.