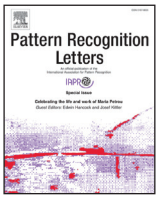
Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec



Self-organizing trajectories

Leonard Johard[a,*], Emanuele Ruffaldi[b]

[a] Innopolis University, 1 Universitetskaya St, Innopolis 420500, Russia
[b] Scuola Superiore Sant'Anna, Piazza Martiri della Libert 33, Pisa 56127, Italy

ARTICLE INFO

Article history:
Received 20 November 2015
Available online 26 September 2016

Keywords:
Shape averaging
Time series
Clustering
Curve averaging
Dynamic time warping

ABSTRACT

Trajectories and parameterized curves are data types of growing importance. Many measures for such data have been proposed in order to provide analogues to the mean and variance of vectors. We identify a counterintuitive oscillating behavior of dynamic time warp-based averages on certain data sets. We present an algorithm that combines ideas from both self-organizing maps and dynamic time warping that avoids these oscillations and hence promises more representative curve averages. These improvements also allow for accurate estimation of the piece-wise variance for a set of general N-dimensional trajectories. The run-time performance is demonstrated on movement data from rowing, where we are able to provide performance feedback in real-time to users in a simulator.

1. Introduction

Parameterized curves are the natural representation for a vast number of data types. Spatial trajectories in particular are a currently expanding research topic, driven mainly by the rapidly increasing performance and lowered cost of navigational and tracking equipment.

Most contemporary statistical and machine learning methods assume fixed input vectors, which is an easier case to treat both theoretically and computationally. In practice, however, curves of varying length may arguably be just as common. Handwritten characters, objects in videos and GPS coordinates are just a few examples of discrete parametrized curves in practice. Hence, there is a growing demand for new statistical methods that are able to effectively treat this data class.

Parameterized curves are often sampled and stored as a time series, but since dynamic time warping is applied the time stamps are irrelevant except the for ordering of the data points. Although the analysis is nominally performed on time series, the data type we are really interested in is the underlying continuous parameterized curve. Analysis of such parameterized curves requires development of new statistical measures.

The extension of measures into a new domain can be realized in several ways. The primary objective of our extended statistical measure is to intuitively transfer information to a human interpreter in describing the data. Although a certain degree of subjectivity is unavoidable, several qualitative and quantitative aspects can be isolated and compared in detail between measures.

For the sake of clarity we will focus our analysis on a particularly common application of statistics for spatial trajectories: human movement analysis. Trajectories here refers to the particular case of parameterized curves in spatial coordinates parameterized by time. Parameterized curve and trajectory will be used interchangeably to describe our underlying data type as we move toward our application example.

1.1. Variance estimation

In the movement analysis applications, especially in sports, the motion commonly consists of repeated movements generated by human subjects in order to achieve an objective. One of the important problems in this application is how to estimate the perfection of movement and rate the performance. Such estimates can be used directly as real-time feedback for learning purposes or, in later analysis, of the subject's skill progression in the task.

Sometimes an easily defined goal, such as success rate or velocity, is available. For cases where such measures are not easily available, it is still possible to estimate user skill using a set of statistical measures that are known to correlate with the performance level in several tasks.

One of the most intuitive movement performance measures that can be applied to a variety of movements is the variance of the set of paths. As a subject improves his skill level, his movements will tend to become progressively more regular and eventually converge towards a locally optimal path with low variance [1].

* Corresponding author.
E-mail address: l.johard@innopolis.ru (L. Johard).

http://dx.doi.org/10.1016/j.patrec.2016.09.012
0167-8655/© 2016 Elsevier B.V. All rights reserved.

Further, in order to give additional information and feedback for training purposes, we are seeking the ability to measure this variability at any specific point over the path. Such feedback promises to enhance the value of variance as feedback, as it allows us to isolate critical areas of the trajectory that have a higher correlation with the performance level.

To arrive at an effective variance estimation for these purposes we will have to create two statistical measures: the mean trajectory and the variance.

1.2. Mean curves

The first objective, to calculate a mean curve, is interchangeably called shape averaging, time series averaging, sequential data averaging or trajectory averaging depending on the application case. Many other topics in trajectory statistics, such as sequence clustering and curve distance measurements, involve the calculation of distance between curves as a central theme in the algorithms.

One approach for developing curve averages is to rely on the median trajectory, as in [2]. This has some advantages, such as not violating space constraints, but are less useful in that its average might be dissimilar in its dynamics. Another related and more recent proposal is the central trajectory by [3].

Both of these averages can produce a representative trajectory in many cases, but do not in any way correspond to the mean in ordinary vector-based statistics. This means that they, while being useful in producing a representative prototype curve, are much less suitable as a foundation for developing other statistical methods relying on the mean, e.g. k-means and variance. We will hence not treat these types of averages further.

There have already been numerous attempts at designing an effective mean trajectory. A common approach is to use dynamic time warping averages between trajectory pairs, as described by [4]. This approach is based on the repeated merger of individual pairs of curves using dynamic time warping until all curves have been processed and a single remaining, hopefully representative, trajectory has been derived.

Unfortunately, as shown by [5], pairwise mergings suffer from undesirable properties for an average. The popular hierarchical averaging method lacks the associative property and can cause cluster centers to drift out of the cluster. These properties are counterintuitive and could result in undesirable behavior in many statistical methods.

In particular, the commonly used k-means clustering algorithm may fail to converge. For these reason, pairwise mergers are poor candidate measures for most applications.

1.3. DBA

To solve the various issues with pairwise mergers a new algorithm was suggested by[6]. Their proposed algorithm, Dynamic Time Warping (DTW) Barycenter Averaging (DBA), starts with an initial guess of the average curve from pairwise curve mergers. Starting from this initial guess, it alternates between two different operations that both lower the DTW distance to the curve set:

1. Perform DTW matchings between the average curve and each curve of the data set. Assign each point on each curve to their associated point on the average trajectory.
2. Move each point on the average curve to the center of the assigned points.

The dynamic time warping will converge to a local minimum of the global inertia in parameter space.

In further work by [7] the authors show that their DBA algorithm can be successfully used to apply k-means clustering. Their results imply that the DBA-algorithm establishes state-of-the-art classification performance by significantly reducing the error compared to hierarchical merging-based methods and even outperforms nearest neighborhood methods. At the moment DBA is one of the most promising candidates for more advanced curve statistics and classification.

1.4. Contribution

The DBA algorithm suffers from a weakness that makes it unsuitable in certain cases: it will tend to converge to spatially oscillating curve averages when the dataset contains parallel curves. This is related to a larger problem where the prior does not limit average curve length. We will present this phenomenon in our description of the trajectory average problem and in the experimental results section.

The second statistical issue treated by this paper, local estimation of the variance, is a natural extension of the mean for parameterized curves. As it is a second order momentum, it relies on the assumption that we are already able to effectively estimate the mean.

In this paper we also present a novel and practical algorithm for solving the two problems we have just mentioned. We show that this is a generalization of the DBA algorithm and that we can find a solution within this group of algorithms that avoids the above undesirable behavior in the curve averages. Furthermore, we present an extension of the scalar definition of variance that is valid for parameterized curves. Finally, we demonstrate that we are now able to produce practical, robust software solutions to the trajectory averaging and variance problems in real-time.

2. Algorithms

2.1. Trajectory variance

In order to calculate the piece-wise variance we first need to create the mean curve. We base our curve on two existing algorithms: Self-organizing maps and dynamic time warping. We will combine the advantages of both in order to produce a practical and converging algorithm for parameterized curves.

2.1.1. Self-organizing maps

A self-organizing map is constructed iteratively from two steps:

1. Assign each curve point to the closest node.
2. Move each node towards the weighted means of the points assigned to it and to its neighboring nodes, as specified by the neighborhood function.

By repeating the above process the map will converge to a trajectory. Since curve points are assigned to closest nodes individually and without consideration of the order of points along the whole curve, the map may not be able to accurately represent self-intersecting curves.

2.1.2. Dynamic time warping

Dynamic time warping is often explained as a stretching and contraction of the time coordinate of a trajectory to minimize the distance to another trajectory. In other words, it is the minimum possible Euclidean distances between two sets of points with the restriction that we can only walk forward in each curve to find the next matching pair of points and that each point needs to be matched with at least one other point.

Formally, an (N, M)-warping path is a sequence $\mathbf{p} = (p_1, p_2, \dots p_L)$ with $p_l = (n_l, m_l) \in [1:N] \times [1:M]$ satisfying the following the conditions:

(1) Boundary condition:

$$\begin{cases} p_1 = (1,1) \\ p_L = (N,M) \end{cases} \tag{1}$$

(2) Step size condition:

$$p_{l+1} - p_l \in (1,0), (0,1), (1,1) \tag{2}$$

for $l \in [1 : (L-1)]$

Calculation of this distance and the corresponding matching points is done in one iteration of the algorithm, which uses dynamic programming to arrive at the solution. Several optimization methods for dynamic time warping exist. Many of these are based on removing excessive dilation of the time from the solution space, e.g. constraint to the Sakoe–Chiba [8] or Itakura Bands [9].

2.1.3. Self-organizing trajectory

Our algorithm combines the principles of self-organizing maps with the time dilation of DTW. Specifically, we achieve this by replacing the first step in the SOM algorithm with the matching from dynamic time warping. Note that, in contrast to the case of self-organizing maps, we must match the whole set of points in the curve to the nodes in a batch:

1. For each trajectory, perform DTW matching for each curve with the nodes.
2. Move each node towards the weighted means of the points assigned to it and to its neighboring nodes, as specified by the neighborhood function.

2.1.4. Initialization

Before starting iterations between the steps we need to initialize our nodes. To do this we take a randomly selected curve from our data set, like in DBA. Unlike DBA, we manually define the number of desired nodes and then stretch the curve uniformly by duplicating data points at even intervals so that the resulting curve contains the desired number of nodes. The positions of the initial nodes does not seem to have a noticeable effect on the minimum obtained from our data set. This initialization method can be seen as an heuristic for shorten training time, while giving us the freedom to specify the curve resolution.

2.1.5. Convergence

The SOT algorithm usually converges in practice by simple matching with the nodes and curve points, but for a strict guarantee of convergence we can instead assign nodes based on the neighborhood distance. This is done by summing the squared distance of all nodes in the neighborhood when performing the DTW matching. min and minval is the argument and value of the minimum, respectively.

$$D_{i,j} = \sum_i d_{i,j} N(i,j) \tag{3}$$

Where $d_{i,j}$ is the squared Euclidean distance and N(i, j) the neighborhood function. Since the neighborhood function is usually zero for $|i - j| > k$ for some small integer k, this can usually be computed without significantly increasing the computational requirements of the algorithm.

Convergence is then guaranteed by the fact that both steps in the algorithm maintain or lower the total sum of squared distances across all curves, which is clearly bounded from below. In order to distinguish this sum from the pairwise inertia used in the DBA algorithm we denote this new sum being minimized by SOT the neighborhood inertia.

The original SOM algorithm performs matching based on distances to individual nodes. If only distances between individual nodes are used in the DTW matching, as per the original SOM algorithm, convergence is not guaranteed. Still, such simplified matching might be a better choice in practice as it is faster to compute. Both SOM and SOT tend to converge on real data with this simplified matching procedure, but convergence proofs are more problematic. This alternative matching method could be preferable in applications where performance is essential.

2.1.6. Pseudocode

The pseudocode of the algorithm is presented below in a form resembling the pseudocode in [6] for easy comparison. We calculate the average sequence C by applying the SOT algorithm a number of times. It in turn uses the neighborhood distance between a set of coordinates δ_N and the node matching function DTW_N.

Algorithm 1 δ_N

```
Require: $N(x)$: Neighborhood function
Require: $a$
Require: $B = \langle b_1, \dots, b_T \rangle$
Require: $b_i$
    $dist_N \leftarrow 0$
    for $i \leftarrow 1$ to $T$ do
        $dist_N \leftarrow dist_N + N(i - b_i)\delta(a, b_i)$
    return $dist_N$
```

Algorithm 2 DTW_N

```
Require: $A = \langle a_1, \dots, a_S \rangle$
Require: $B = \langle b_1, \dots, b_T \rangle$
    $m[1,1] \leftarrow (\delta_N(a_1, B, 1), (0,0))$
    for $i \leftarrow 2$ to $S$ do
        $m[1,j] \leftarrow (m[i-1,1,1] + \delta_N(a_i, B, 1), (i-1, 1))$
    for $j \leftarrow 2$ to $T$ do
        $m[1,j] \leftarrow (m[i1, j-1, 1] + \delta_N(a_1, B, j), (1, j-1))$
    for $i \leftarrow 2$ to $S$ do
        for $j \leftarrow 2$ to $T$ do
            $minimum \leftarrow minVal(m[i-1,j], m[i,j-1], m[i-1,j-1]$
            $m[i,j] \leftarrow (first(minimum) + \delta_N(a_i, B, j), second(minimum))$
    return $m[S,T]$
```

2.1.7. Relation with DTW barycenter averaging

A special case of the SOT algorithm is a variant where it updates the node centers in batches and moves the average with a step size equal to the distance to the curve average. If we then use

Algorithm 3 SOT

```
Require: $\alpha$ Step size monotonically decreasing for each iteration of
    SOT
Require: $C = \langle C_1, \dots, C'_T \rangle$ the initial average sequence
Require: $\mathbb{S}_1 = \langle s_{1,1}, \dots, s_{1,T} \rangle$ the 1
    $C' \leftarrow C$
    for seq in $\mathbb{S}$ do
        $m \leftarrow DTW_N(seq, C)$
        $i \leftarrow T$
        $j \leftarrow T'$
        while $i \geq 1 and j \geq 1$ do
            $C'_j = C_j + \alpha(seq_i - C'_j)$
            $(i,j) \leftarrow first(m[i,j])$
    return $C'$
```

the Kronecker delta as the neighborhood function,

$$N(i,j) = d_{i,j} \tag{4}$$

this is equivalent to the DBA algorithm.

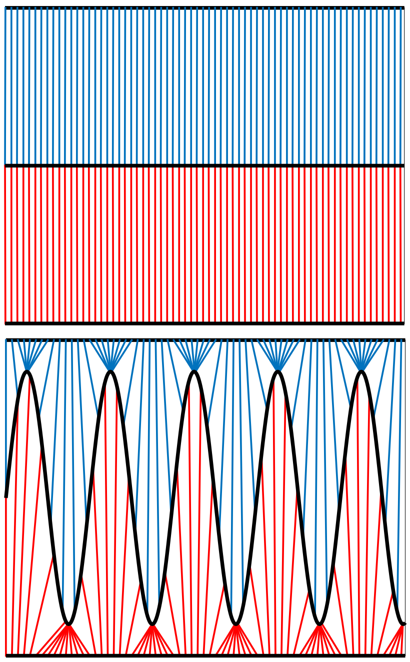


Fig. 1. Two different curve averages (black thick line) between two parallel lines (thin black lines). DTW matchings between top and bottom lines with the curve average are illustrated by red and blue lines. A straight curve is a better average candidate, but a densely oscillating curve is on average closer to the top and bottom curve. In this example the inertia produced by the bottom oscillating curve is 0.64 of the inertia of the straight top curve. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Note that the choice of neighborhood function affects the distance measure being minimized. In particular, an average trajectory minimizing the DTW distance with a Kronecker delta neighborhood will likely result in an oscillating average. The reason for this is that a spatial oscillation between different parallel curves will always have points close to any given curve on either paths and in most cases, consequently, a smaller summed DTW distance than any given path parallel to these curves. A conceptual description of this can be seen in Fig. 1.

While this might appear as an insignificant effect on small data sets, this is not always the case. A particularly problematic aspect of this phenomenon is that the magnitude of this oscillating effect grows as sample sizes and precision grow, i.e. when we have curve variance that is large in comparison to the distance between the nodes of the curve average.

We propose a simple solution to this through our SOT algorithm: by utilizing the more general self-organizing trajectories and apply a different neighborhood function we will increase the prior likelihood of non-oscillating curves.

The step size is largely an implementation detail and does not affect the optimum nor the convergence property if we base the node matching on the neighborhood inertia. If we instead match based on distance to individual nodes, a smaller step size might offer a more stable behavior as there is no strict guarantee that the neighborhood inertia decreases. A well-balanced step size also allows for exponentially weighted moving averages. On the other hand, the DBA solution leads to the greatest possible reduction in neighborhood inertia in each node movement step of the algorithm.

2.2. Variance estimation

We seek to arrive at pointwise variance estimate along a curve in order to estimate user performance and provide valuable feedback to the user. In order to do this we will first extend the concept of variance into the domain of trajectories. We would like this measure to be intuitive and have similar properties as the variance of vectors. We begin with a definition for continuous trajectories in order to guarantee that our estimate converges to some value as we increase the sampling frequency and the size of the data set. After establishing our continuous measure we proceed with a method for estimating it given a set of sampled trajectories.

2.2.1. Defining continuous trajectory variance

We define trajectories as a set of spatial coordinates $\mathbf{x}_i$ with corresponding time coordinates t_i. An intuitive concept of local trajectory variance of these is not easily defined. First we have to define a local concept for curves similar to the mean. Intuitively, this suggests a curve around which the average displacement of all curves is 0 on a certain plane locally orthogonal to the curve.

For a more formal definition in the continuous case, we define an orthogonal hyperplane $\mathbf{H_i}(t)$ of a regular parameterized curve $\mathbf{c_i}(t)$ as the set of points described by

$$\mathbf{H_i}(t) = \mathbf{c}_i(t) + \mathbf{v}, \quad \mathbf{v} \cdot \frac{\partial}{\partial t}\mathbf{c}(t) = 0 \tag{5}$$

We define the projection $\mathbf{proj}_{\mathbf{c}_i,\mathbf{c}_j} : \mathbb{R} \to \mathbb{R}^{n-1} \times \mathbb{R}$ of a curve $\mathbf{c}_i$ on the set of orthogonal hyperplanes of a another curve $\mathbf{c}_j$ as the following set:

$$\mathbf{proj}_{\mathbf{c}_i,\mathbf{c}_j}(t) = \mathbf{c}_i(s) \cup \mathbf{H}_j(t) \tag{6}$$

To restrict our definition to trajectories that run in the same direction and within the focus of the curvature we also require two additional conditions: that s is a monotonic function of t and that $\frac{\partial}{\partial t}\mathbf{c}_i(s) \cdot \frac{\partial}{\partial t}\mathbf{c}(t) > 0$.

Using the notation we can define the trajectory mean $\mathbf{m}(s)$ of a set of curves $\mathbf{c}_i(t)$ as follows:

$$\sum_i \mathbf{proj}_{\mathbf{c}_i,\mathbf{m}}(t) = \mathbf{m}(t) \tag{7}$$

If such a mean is well-defined for a curve in some neighborhood of t_0, it at each point closely agrees with the corresponding definition of mean for vector. This definition allows us to estimate the variance as the squared distance of our variance estimation in the same hyperplanes.

2.2.2. Estimating trajectory mean and variance

After establishing our piecewise linear curve average we calculate a variance estimate for each curve segment. DTW assigns points to nodes, which is used as a starting point for curve segment assignment. To make an assignment to curve segments, we assign each point to one of the mean curve segments connected to the node. First we check if the point belongs to the orthogonal hyperplanes spanned by any of the two adjoining curve segments:

$$0 < \frac{(\mathbf{p} - \mathbf{n}_i) \cdot (\mathbf{n}_{i+1} - \mathbf{n}_i)}{\|\mathbf{n}_{i+1} - \mathbf{n}_i\|} < 1 \tag{8}$$

where p is the point of the trajectory, while n_i and n_{i+1} are the ending points of the curve segment in question.

All trajectory means assigned to a curve segment are used to calculate mean and variance. This is done as the regular case around a point, except that we substract the component parallel to the curve segment.

$$\mathbf{p}_{proj} = \mathbf{p} - \mathbf{n}_i - (\mathbf{p} - \mathbf{n}_i) \cdot \frac{(\mathbf{n}_{i+1} - \mathbf{n}_i)}{\|\mathbf{n}_{i+1} - \mathbf{n}_i\|} \tag{9}$$

If the point lies in the set of orthogonal hyperplanes to both line segments, it is assigned to the closest line segment.

3. Results in human motion analysis

In this section we evaluate the performance of our algorithm in practice. First we demonstrate curve averaging in real-time while retaining high quality solutions. We also illustrate the implications of oscillation on real data, as discussed in relation to the DBA algorithm, and compare with the curves produced by self-organizing trajectories. An example of our code can be found in [10].

3.1. Real-time application

First we would like to evaluate the ability to perform the analysis in real-time with immediate feedback to the user, opening the doors to many other real-life applications. We test our system on data from users on a real rowing simulator collected by [11]. Our algorithm will run in the background in parallel with a physical simulator of the boat and a stereoscopic 3D graphics environment

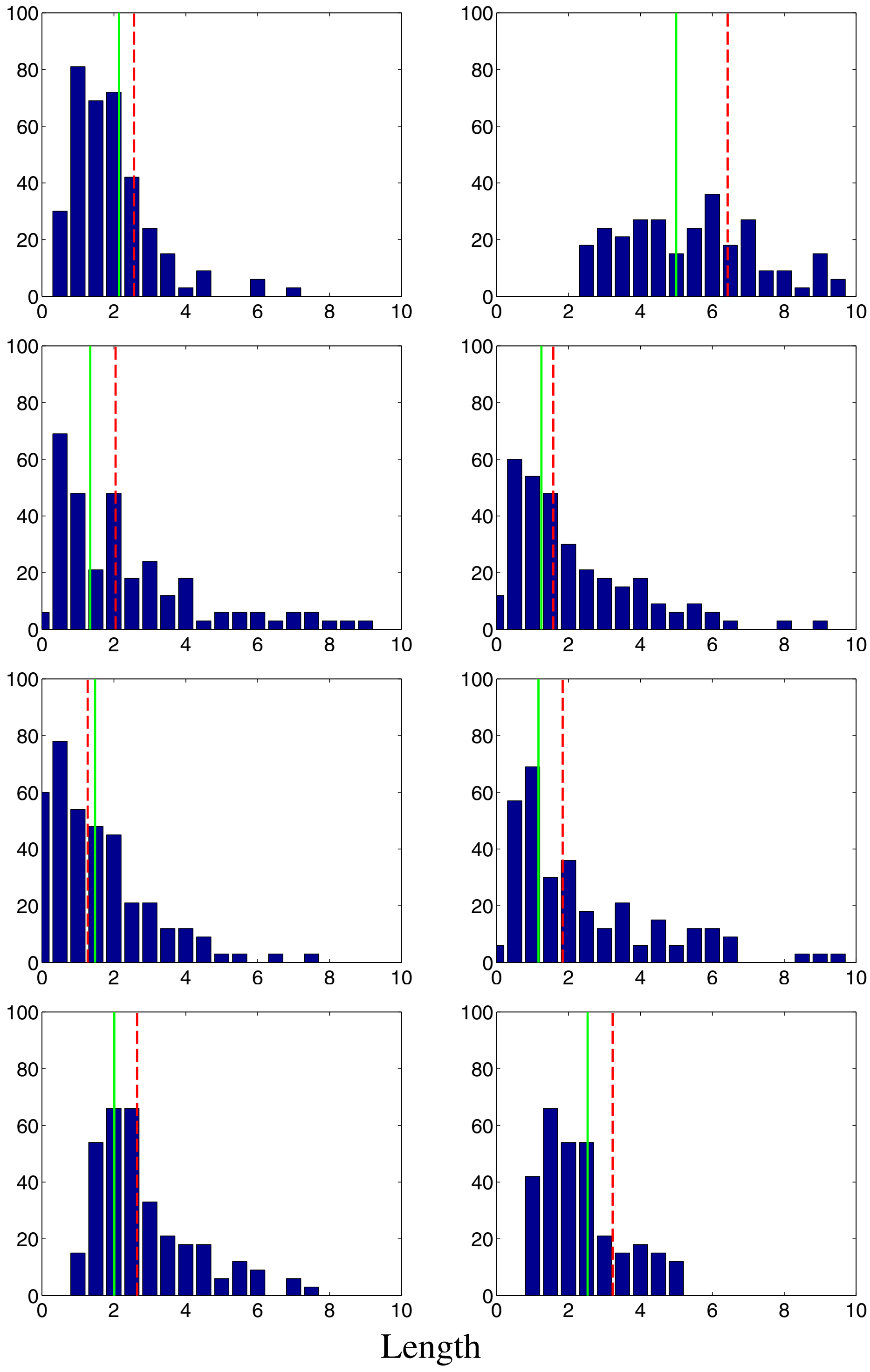


Fig. 2. Histogram of curve lengths compared to SOT mean curve length (green line) and DBA mean curve length (dotted red line). Curve length is here defined as the Euclidean length of the path created by connecting the nodes with lines in consecutive order. The SOT mean curves tend to be shorter and closer to the mode. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

providing realistic input to the rower. With the simulator running at 100 Hz, this leaves a small and highly variable fraction of the CPU available to our algorithm.

Rowing is a periodic movement generally divided into four phases and with a frequency in the order of 2 seconds. We trained a neural network to identify the start and end of each rowing cycle as in [12] and add the resulting cycle trajectory into our SOT. To deal with a periodic trajectory we create a SOT with a circular neighborhood function, i.e. a neighborhood function where the first and the last nodes of the map are considered to be adjacent. This guarantees that the mean trajectory forms a periodic path. For clarity and in order to avoid overfitting of hyperparameters, we have used a simple triangular neighborhood function, with

$$N(i,j) = \begin{cases} 1 \text{ if } i = j \\ 0.5 \text{ if } |mod(i-j)| = 1 \\ 0 \text{ in all other cases} \end{cases} \tag{10}$$

Since a calculation of the full average is too expensive to be performed in a single rowing cycle, we store the last 30 curves and weigh them by an exponential decay function $e^{-\lambda i}$ where i is an index of all 30 curves starting from the most recent one counting backwards in time. λ is the decay rate parameter. This approximately transforms our average trajectory estimate into the trajectory equivalent of an exponentially weighted moving average. Whenever free computing power is available, we then alternate between two steps:

(1) Sampling a stored curve for updating their respective DTW matching against the current set of nodes
(2) Sampling a curve and moving the mean trajectory nodes toward their matched curve points

This alternation performs expectation-maximization stochastic gradient descent on the total DTW distance function if we use neighborhood matching. Since the input trajectories are constantly changing and processing power limited we may not expect the algorithm to be able to converge despite a decaying step size. Instead we chose a fixed step size that has been optimized manually for both accuracy and adaptability to changes in the rowing style of the rower.

When a new curve is added, we prioritize the calculation of its DTW distance and also calculate the distance to each line segment to provide feedback to the users. This could be accomplished within the first 10 ms frame, providing practically latency-free feedback of the previous rowing cycle and allowing the rower to adapt immediately.

In our example application written in Matlab we utilized an average of 4% of the Intel Core 2 Q6600 CPU across a 10-minute session and were able to provide excellent feedback to users of the simulator. Due to common artifacts in VICON motion capture, such as reflection from the machinery, we have several outliers in the input data. These tended to fall outside the searched volumes for the mean calculation, resulting in a very robust mean and variance estimation even in presence of such severe outliers.

Even when starting from an initial trajectory severely distorted by noise, the algorithm quickly reestablished the proper shape. The average can be seen in Fig. 3, where some of the more extreme outliers are also clearly visible. We used no other filters except those inherent in the algorithm described.

As subjects changed rowing style within a single trial in response to fatigue and desired velocity, the algorithm quickly adapted and reestablished a new mean curve as well as updated variance estimates per segment within a few strokes.

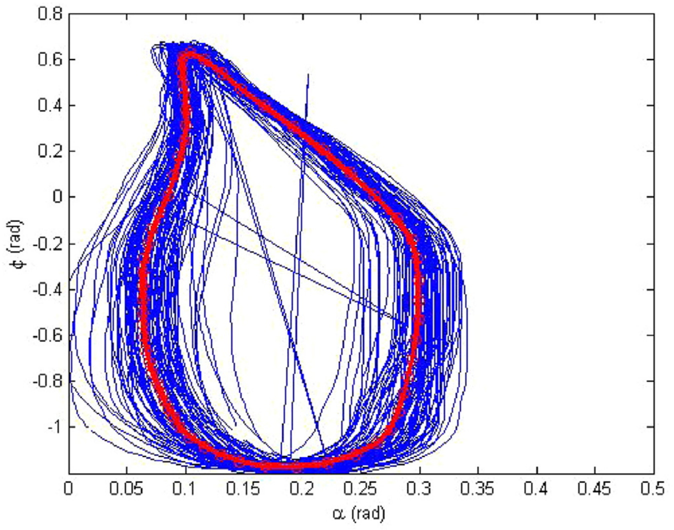


Fig. 3. Mean curve (thick red line) and sample curves (thin blue lines) calculated in real-time. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.2. Curve comparison with DBA

We illustrate how the SOT algorithm compares against DBA in practice using two examples of human movement analysis: our own dataset from the rowing simulator and the accelerometer recordings of 8 different hand signs by [13]. We use 100 nodes in all examples in order to illustrate effects on sampling rates similar to those required for our variance calculations. The number of data points of each trajectory in our data set varies between 150 and 400 data points, except for a few outliers. To isolate the relevant aspects of our comparison we performed all tests in this comparison offline, i.e. using the regular SOT rather than the weighted averages used for real-time estimates described above. We also picked a number of iterations of SOT at least twice of what seemed to be required for convergence in early trials.

In these experiments the DBA produces the expected oscillating behavior, while the SOT resulted in a much smoother mean curve. The same respective behavior of both algorithms is reproduced on the other data set, as seen in Figs. 4 and 5.

Converged mean curves on our own dataset can be seen in Fig. 6. Being of higher resolution than the hand sign data set, we see a high degree of oscillation using the DBA on these data. The resulting curve average is practically useless for defining the orthogonal planes and calculating variance. The effect of the neighborhood function in SOT is dramatic on the structure of the curve and the direction of its curve segments.

The qualitative difference seems obvious, but below we will also show the difference using quantitative measures in order to better understand how the curve averages behave on our data sets.

One way to tackle the comparison quantitatively is by investigating curve lengths, here defined as the Euclidean length of the path formed by connecting the nodes in consecutive order with lines. The average should preferably eliminate sensor noise, while preserving other qualities of the curve. Hence we expect a good mean to be of equal or shorter length than typical curves in the data set. The distribution of curve lengths can be seen in the histogram in Fig. 2. While the SOT produces the expected mean curve lengths, DBA sometimes produces mean curves that are much longer than the distribution supports. SOT also produces lengths closer to the mode of the example curves in almost every case.

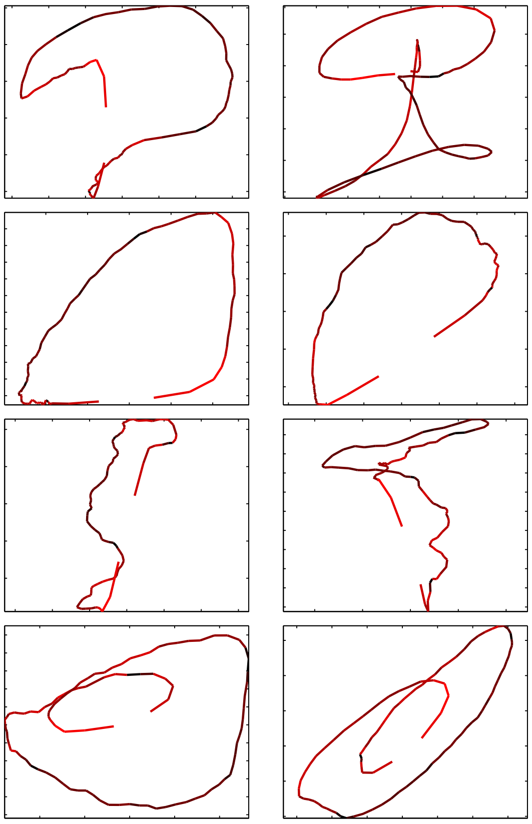


Fig. 4. Curves produced by the SOT algorithm projected on the y-z plane. The mean curves display a low level of oscillation even for a high number of nodes.

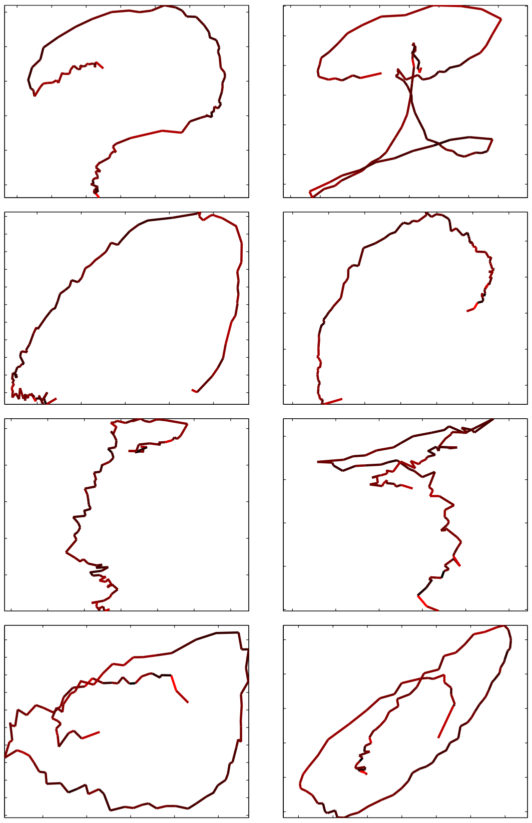


Fig. 5. Curves produced by the DBA algorithm projected on the y-z plane. All mean curves display increasing oscillations which do not correspond to oscillations in the original data set as the number of nodes increase.

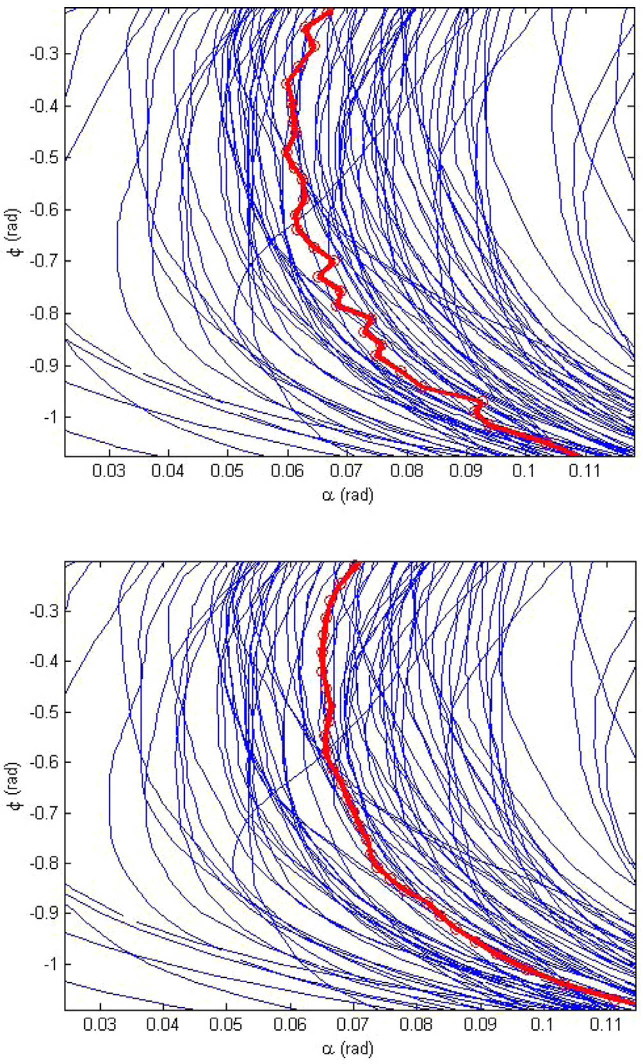


Fig. 6. Enlarged view after convergence. Mean curve (thick red line) and sample curves (thin blue lines). Top: aliasing effects from the DBA algorithm. Bottom: solution from the SOT algorithm. All parameters except the neighborhood function are kept constant between the images. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Second, how does this influence the inertia of the solution? The SOT algorithm minimize the neighborhood inertia rather than the inertia, as explained in Section 2.1.5. It could naively be expected that DBA has a significant advantage in this area, as there are fewer restrictions for its curve shape and since the oscillation patterns allow a much greater flexibility in the fine-grained curve structure that lowers the inertia, as we illustrated in Fig. 1.

In practice, as can be seen in Fig. 7, the DBA inertia advantage is counteracted by the limited number of nodes of our curve average. Given the same number of nodes the oscillation patterns provide just a slight improvement in the converged inertia. In many cases DBA also seems to get stuck in local minima and/or plateaus and we find it quite surprising that DBA results in a higher inertia than the SOT algorithm on certain data.

Overall, DBA does not seem to offer a notable advantage in terms of inertia, despite the algorithm optimizing it directly. This means that the quality advantage of DBA over SOT is small, while DBA also suffers from longer curve lengths and inconsistent curve derivatives due to oscillation. An advantage of DBA is that its simpler node matching is computationally less expensive by a constant factor.

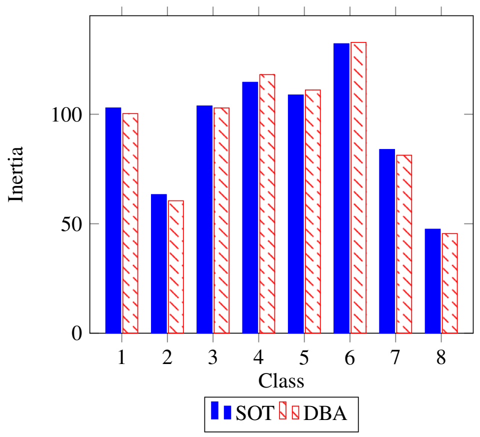


Fig. 7. Comparison between the inertia produced by the averages from SOT and DBA algorithms. The trajectories are qualitatively different, but the difference in inertia is small.

It can be noted that the oscillation produced by the DBA is less evident if we lower node numbers; on the other hand, this reduction will limit the maximum resolution of the curves and is for this reason an unsuitable method for high-quality means on large data sets.

4. Conclusion

Curve oscillation can be an important artifact in mean curve generation, especially for purposes such as variance calculations. Our algorithm replaces the inertia with a neighborhood inertia that proves an efficient heuristic to solve oscillation problems. However, in future work we are looking to develop an explicit prior over all possible trajectories.

We have shown that SOT offer solutions with significant reductions in oscillation without a large loss in inertia in exchange for a small computational overhead.

We have also proposed a practical way to estimate the variance for curves and demonstrated that both measures can be calculated efficiently enough to be used in real-time applications.

Sound mean and variance measures are at the core of many more advanced statistical methods for vector-based statistics, e.g. clustering and classification algorithms, that could be extended into the trajectory domain. Our hope is that these measures can help researchers to effectively create useful methods in these data sets.

References

[1] C.M. Harris, D.M. Wolpert, Signal-dependent noise determines motor planning., Nature 394 (6695) (1998) 780–784.
[2] K. Buchin, M. Buchin, M. Van Kreveld, M. Löffler, R.I. Silveira, C. Wenk, L. Wiratma, Median trajectories, Algorithmica 66 (3) (2013) 595–614.
[3] M.J. van Kreveld, M. Löffler, F. Staals, Central trajectories, CoRR abs/1501.0 (2015).
[4] L. Gupta, D.L. Molfese, R. Tammana, P.G. Simos, Nonlinear alignment and averaging for estimating the evoked potential, IEEE Trans. Biomed. Eng. 43 (4) (1996) 348–356.
[5] V. Niennattrakul, C. Ratanamahatana, Inaccuracies of shape averaging method using dynamic time warping for time series data, in: Int Conf Comput Sci, in: ICCS '07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 513–520.
[6] F. Petitjean, A. Ketterlin, P. Gançarski, A global averaging method for dynamic time warping, with applications to clustering, Pattern Recognit. 44 (3) (2011) 678–693.
[7] F. Petitjean, G. Forestier, G.I. Webb, A.E. Nicholson, Y. Chen, E. Keogh, Dynamic time warping averaging of time series allows faster and more accurate classification, in: IEEE Int Conf Data Min, IEEE, 2014, pp. 470–479.
[8] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, IEEE Trans. Acoust. Speech Signal Process. 26 (1) (1978) 43–49.
[9] F. Itakura, Minimum prediction residual ap- plied to speech recognition, IEEE Trans. Acoust. Speech Signal Process. 23 (1) (1975) 67–72.
[10] L. Johard, Self-organizing Trajectories v1.0, 2016, 10.5281/zenodo.55925
[11] A. Filippeschi, E. Ruffaldi, Boat dynamics and force rendering models for the SPRINT system, IEEE Trans. Hum.Mach. Syst. 43 (6) (2013) 631–642.
[12] L. Johard, A. Filippeschi, E. Ruffaldi, Real-time error detection for a rowing training system, in: BIO Web of Conf, 1, EDP Sciences, 2011, p. 44.
[13] J. Liu, L. Zhong, J. Wickramasuriya, V. Vasudevan, uWave: Accelerometer-based personalized gesture recognition and its applications, Pervasive Mob. Comput. 5 (6) (2009) 657–675.