Re-Identification in the Function Space of Feature Warps Supplementary Material

Niki Martinel[1], *Student Member, IEEE,* Abir Das[1], *Student Member, IEEE,* Christian Micheloni, *Member, IEEE,* and Amit K. Roy-Chowdhury[*], *Senior Member, IEEE*

I. Introduction

This supplementary material accompanies the paper "Re-Identification in the Function Space of Feature Warps".

This supplementary material is structured as follows.

- Section II gives an introduction to Dynamic Time Warping (DTW). We describe how the algorithm can be applied to get the warp functions we formulated through Eqn. (1)-(3) in Section IV of the main paper. We also provide the algorithm (Algorithm. 1) for a better comprehension of the concept.
- Section III shows a set of sample images from the RAiD dataset. This shows the change in illumination and pose occurring between several camera views, especially between the indoor and the outdoor cameras.
- Section IV shows the extracted texture features for an example image.
- Section V provides a comparison of re-identification performance in terms of the CMC curves as different classifiers and dense feature patch sizes are used.

To keep the supplementary material as much readable as possible we have tried to design self-explanatory images. Hence, we did not add much text describing each one of them, but we provide a little discussion within the captions.

II. Dynamic Time Warping For Feature Transformation

In the following we introduce the main concepts of Dynamic Time Warping. Then, at the end of the section we provide our formulation where the warp path between two vector functions, computed by DTW, is approximated to obtain a vector of the same length as that of the input. Please note that this is a brief introduction and we direct the interested readers to [1], [2] for further details.

Let $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^n$ be two vector functions. Notice that in our specific case we have that $m = n$. However, in the following, we show a more general DTW framework without considering this assumption.

[1] The first two authors should be considered as joint first authors

[*] Corresponding author

N. Martinel and C. Micheloni are with the Department of Mathematics and Computer Science, Univeristy of Udine, Udine, Italy, 33100.
E-mail: {niki.martinel, christian.micheloni}@uniud.it

A. Das and A.K. Roy-Chowdhury are with the Department of Electrical Engineering, University of California Riverside, Riverside, California, 92521.
E-mail: abir.das@email.ucr.edu, amitrc@ee.ucr.edu

The DTW algorithm constructs a warp path W

$$W = W(1), \ldots, W(K) \tag{1}$$

where $\max(m, n) \le K < m + n$. The k^{th} element of the warp path is defined as

$$W(k) = (a, b) \tag{2}$$

where $a \in \{1, \ldots, m\}$ and $b \in \{1, \ldots, n\}$. The warp path W thus, maps, or aligns, the elements of $\mathbf{x}$ and $\mathbf{y}$ such that the distance between them is minimized.

To formulate a dynamic programming solution of the problem, we must have a distance measure between two elements. Although any valid distances can be used, in general, the magnitude of the difference and the Euclidean distance are used [1]. Let $\delta(x(a), y(b))$ be the distance between the elements $x(a)$ and $y(b)$. Then, a cost matrix $C \in \mathbb{R}^{m \times n}$ is generated where the $(a, b)^{th}$ element (denoted as C_{ab}) of the matrix is given by the distance $\delta(x(a), y(b))$. The DTW algorithm searches for the path W giving the lowest cumulative cost between fixed start point, the $(1, 1)^{th}$ cell and fixed end point, the $(m, n)^{th}$ cell of C. Let $\mathbb{W} = \{W_1, W_2, \cdots\}$ be the set of all possible paths between these two fixed points where W_q denotes the q^{th} path. W_q consists of tuples indicating the indexes of the cells in C.

$$W^* = \underset{W_q \in \mathbb{W}}{\text{argmin}} \left(\sum_{(a,b) \in W_q} C_{ab} \right) \tag{3}$$

For long time series, searching through all possible warp paths leads to high computational cost as the warp path is combinatorially explosive [1]. So, it is sometimes important to restrict the space of all possible warping paths. Some of the constraints commonly used in DTW are described below,

(i) Monotonicity: the a-th and b-th points must be monotonically ordered with respect to time (Fig.1(b)).

(ii) Continuity: the steps in the search grid are confined to neighboring points (Fig.1(c)). That is $W(k+1) - W(k) \in \{(1, 0), (0, 1), (1, 1)\}$ for $k \in \{1, .., K - 1\}$.

(iii) Boundaries: the boundary condition restricts the search space such that the starts and the ends of the two series are always linked: $W(1) = (1, 1)$ and $W(K) = (m, n)$ (Fig.1(d)).

(iv) Slope: Allowable warping paths can be constrained by restricting the slope, thus avoiding excessively large movements in a single direction.

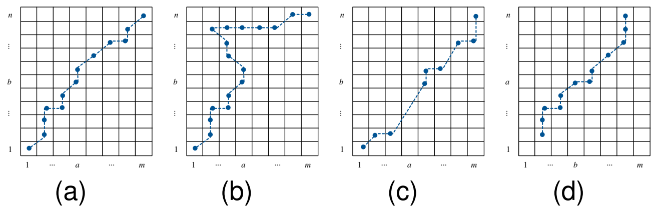


Fig. 1. Four possible warping paths for some sequence $\mathbf{x}$ of length m and sequence $\mathbf{y}$ of length n. (a) Admissible warping path satisfying the conditions (i), (ii), and (iii). (b) Monotonicity condition (i) violated. (c) Continuity condition (ii) violated. (d) Boundaries condition (iii) violated.

(v) Warping window: allowable points can be forced to fall within a constrained window such as the Sakoe-Chiba Band [3] or the Itakura Parallelogram [4] (see Fig. 2).

DTW finds the optimal solution by decomposing it into subproblems and solving them in a recursive way. Lets assume that the continuity constraint holds, then the restriction on the indices makes sure that the total distance D of the best warp path matching $x(a)$ and $y(b)$ is equal to the distance between $x(a)$ and $y(b)$ plus the minimum distance needed to match $x(a)$ and $y(b+1)$, or $x(a+1)$ and $y(b)$, or $x(a+1)$ and $y(b+1)$. In other words

$$D(a,b) = \begin{aligned} \delta(x(a), x(b)) + \min\big(&D(a,b+1),\\ &D(a+1,b),\\ &D(a+1,b+1)\big). \end{aligned} \tag{4}$$

Since the recursive definition has many overlapping subproblems, DTW is suitable for an efficient dynamic programming implementation. It is sufficient to store all the $D(a,b)$ values in a $m \times n$ matrix, which is also commonly referred to as the accumulated cost matrix, and fill it in the proper order. The $D(m,n)$ element of the matrix will hold the total warp path distance starting from the start point to the $(m,n)^{th}$ point. and it is thus, a solution for the minimization problem 3. This value is called the Dynamic Time Warping *distance*, even if it is not a real distance in the mathematical sense of the term, since it does not satisfy the properties of distance measures in a metric space.

The computational complexity of the algorithm is $O(mn)$ both in space and time because the whole matrix needs to be computed. If only the final distance is needed (thus discarding information about the warping path leading to that distance measure), space complexity becomes linear since only two adjacent columns (or rows) of the table are needed at the same time. If an approximate solution is acceptable, the algorithm could be forced not to use the whole table, imposing constraints such as the Sakoe-Chiba Band [3] or the Itakura Parallelogram [4] (Fig.2). Using these constraints, the time complexity becomes linear or quadratic, depending if the width of the allowed band is fixed or proportional to the input size.

We have shown that given two vector functions, the DTW is able to capture the transformation between these. However, as told earlier, in our specific case we have $m = n$. We also need the resulting warp function W^* to be of the same length as that of the input vector functions. This is due to the fact that we have to use it as an input to a classification framework.

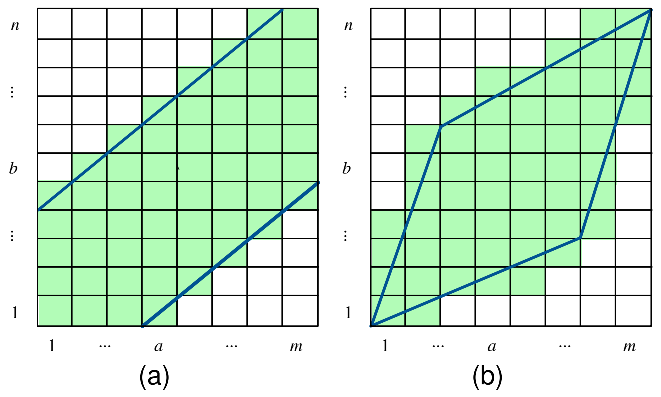


Fig. 2. (a) Sakoe-Chiba Band and (b) Itakura Parallelogram constraints.

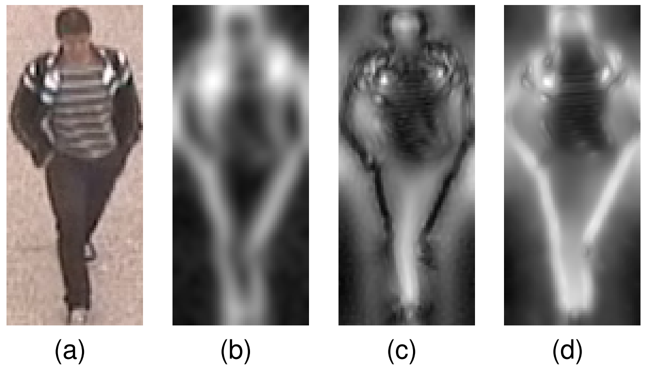


Fig. 4. Response images after convolutions with the Gabor, Schmid and Leung-Malik filter banks. All filter responses are summed and scaled for visualization. (a) Input image. (b) Response after convolution of 40 Gabor filters. (c) Response after convolution of 13 Schmid filters. (d) Response after convolution of 48 Leung-Malik filters.

In the main paper we have provided our approximating rule (see Eq.(3)) to get a m length warp function. Algorithm 1 provides a brief summarization of the steps to compute the warp function considering the approximations and constraints used by us.

III. Sample images from RAiD Dataset

In Fig. 3 we provide a few sample images from the RAiD dataset showing the illumination variation across the cameras.

IV. Response of texture filters

In Fig. 4 responses to the texture filters to an example image is shown. Results has been adapted for visualization.

V. Analysis of Classifiers Performance and Patch Size Parameter

In this section we provide the comparison of re-identification performance in terms of the CMC curves as different classifiers and dense feature patch sizes are used. For this experimentation, all other parameters are kept same as described in Section V-A of the main paper and the experiments are carried out employing a multi-shot strategy with $N = 10$. Following the same convention as used throughout

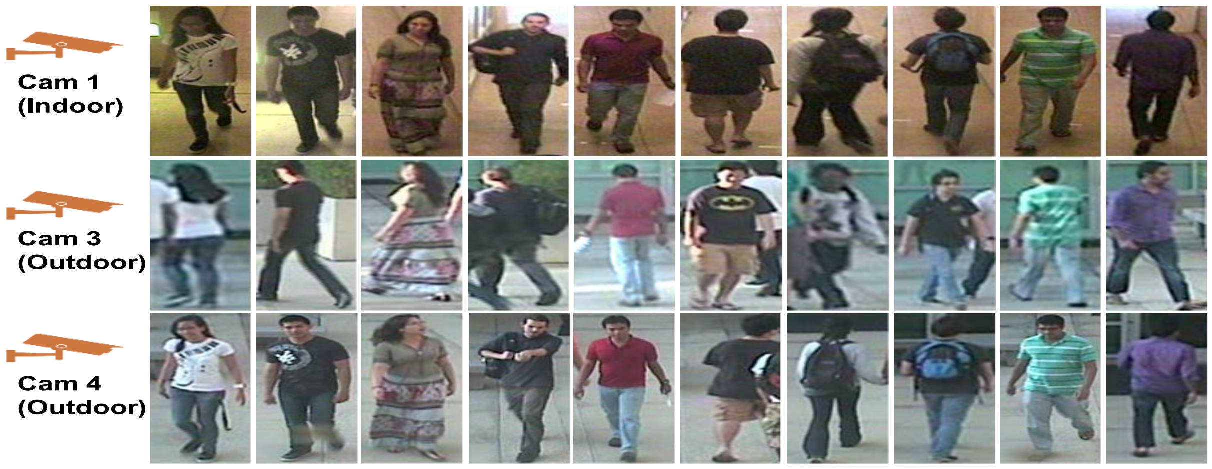


Fig. 3. Sample images of persons from the RAiD dataset showing the variation of appearance between the indoor and the outdoor cameras.

the paper, the patch size used for both the classifiers, is 8×8 and a RF classifier is chosen for the experiments with different patch sizes.

A. *Performance comparison for different choices of classifiers*

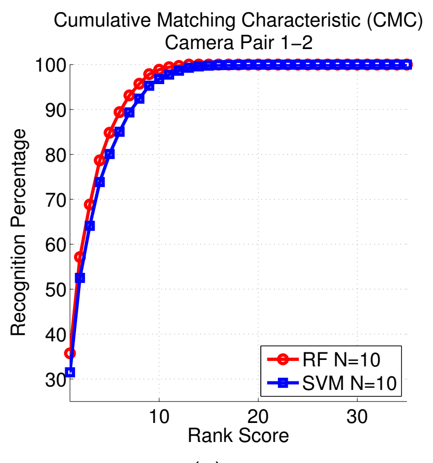
Fig. 5 and 6 show the CMC curves showing the comparison of re-identification performance with two different classifiers (RF and SVM) for WARD and RAiD dataset respectively. As seen in Table V of the main paper, Fig. 5 and 6 also show that the variation of performance is very little for all the camera pairs for both the two datasets.

B. *Performance comparison for different choices of patch sizes*

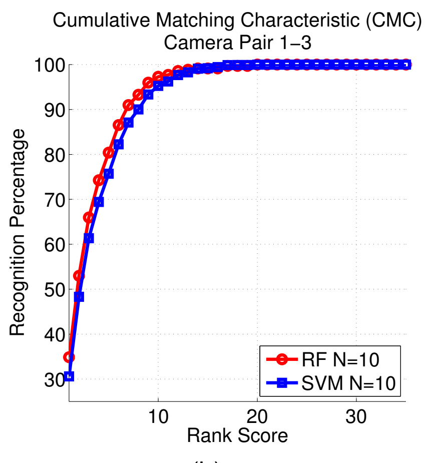
Fig. 7 and 8 show the CMC curves showing the comparison of re-identification performance with three different dense feature patch sizes ($4 \times 4, 8 \times 8$ and 16×16) for WARD and RAiD dataset respectively. As seen in Table V of the main paper, Fig. 7 and 8 also show that the change in performance becomes little as the patch size grows for all the camera pairs for both the two datasets.

References

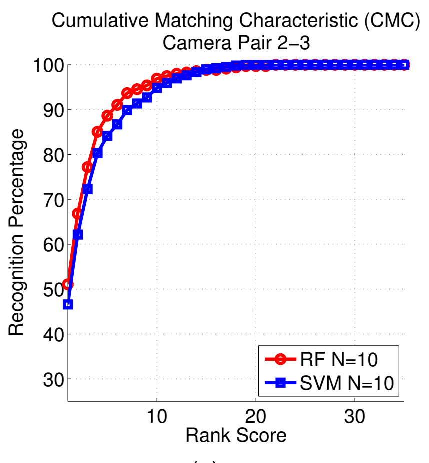
[1] D. J. Bemdt and J. Clifford, “Using Dynamic Time Warping to Find Patterns in Time Series,” in *Working Notes of the Knowledge Discovery in Databases Workshop*, 1994, pp. 359–370.
[2] M. Müller, “Dynamic Time Warping,” in *Information Retrieval for Music and Motion*. Berlin: Springer Publications, 2007, vol. 2, ch. 4, pp. 69–84.
[3] H. Sakoe and S. Chiba, “Dynamic programming algorithm, optimization for spoken word recognition,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.
[4] F. Itakura, “Minimum prediction residual principle applied to speech recognition,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 52–72, 1975.



(a)

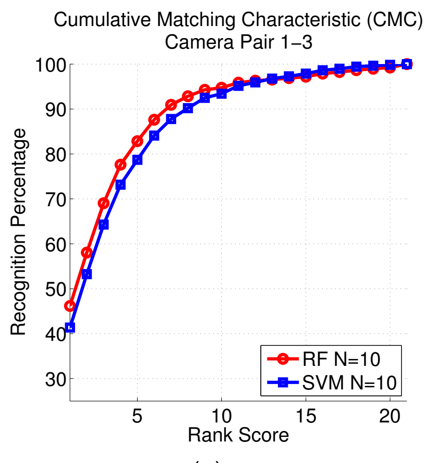


(b)

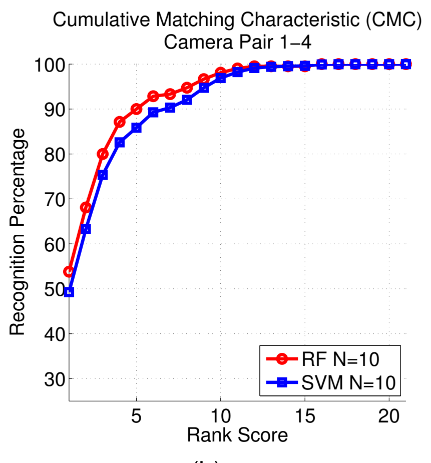


(c)

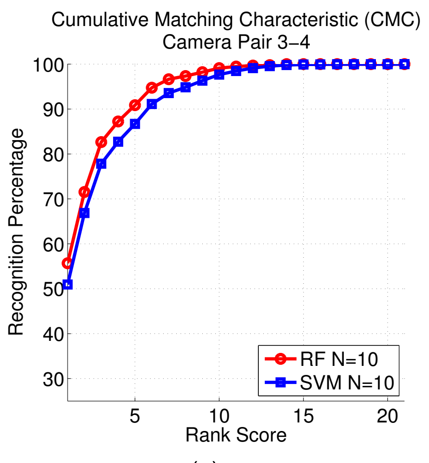
Fig. 5. CMC curves showing the comparison of re-identification performance with two different classifiers (RF and SVM) for WARD dataset. In (a), (b) and (c) comparisons are shown for the camera pairs 1-2, 1-3 and 2-3 respectively.



(a)

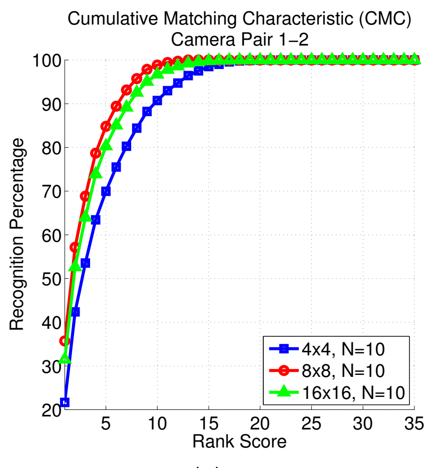


(b)

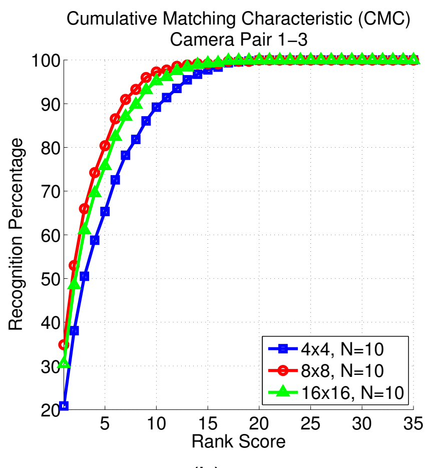


(c)

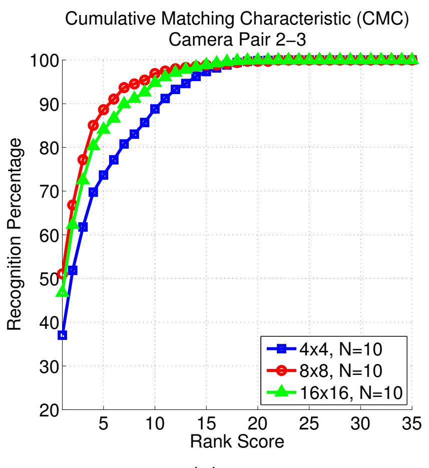
Fig. 6. CMC curves showing the comparison of re-identification performance with two different classifiers (RF and SVM) for RAiD dataset. In (a), (b) and (c) comparisons are shown for the camera pairs 1-3, 1-4 and 3-4 respectively.



(a)

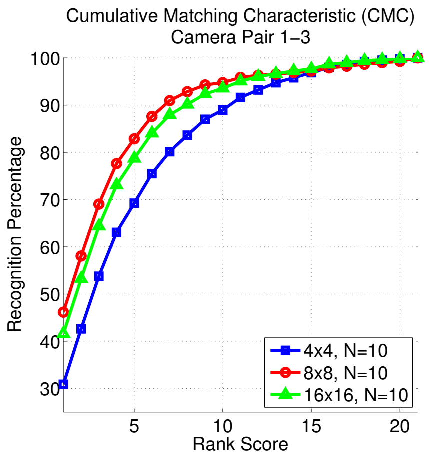


(b)

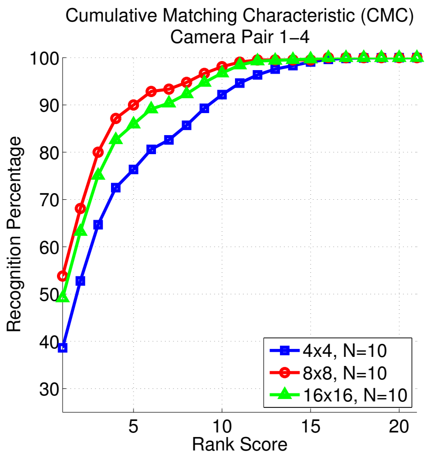


(c)

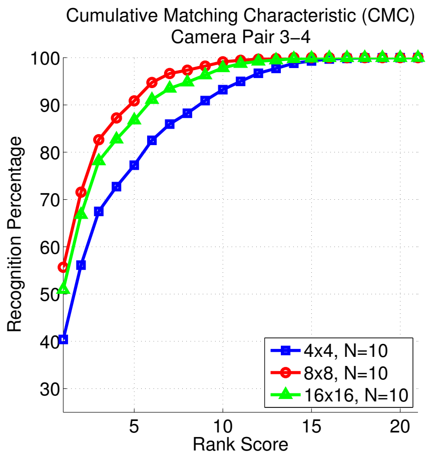
Fig. 7. CMC curves showing the comparison of re-identification performance with three different dense patch sizes ($4 \times 4, 8 \times 8$ and 16×16) for WARD dataset. In (a), (b) and (c) comparisons are shown for the camera pairs 1-2, 1-3 and 2-3 respectively.



(a)



(b)



(c)

Fig. 8. CMC curves showing the comparison of re-identification performance with three different dense patch sizes ($4 \times 4, 8 \times 8$ and 16×16) for RAiD dataset. In (a), (b) and (c) comparisons are shown for the camera pairs 1-3, 1-4 and 3-4 respectively.

Algorithm 1: DTW algorithm to obtain an m length warp function

```
input  : Two vector functions $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^m$ extracted from two images acquired by disjoint cameras
output: The warp function $\mathbf{f}_{(\mathbf{x},\mathbf{y})} \in \mathbb{R}^m$, as defined in the main paper
// Given the two vector functions x and y, ComputeCostMatrix computes the
   corresponding cost matrix $C$
1  Function ComputeCostMatrix(x, y)
       // To compute the cost matrix we have to loop through all possible
          combinations of $a$ and $b$
2      for $a \leftarrow 1 : |\mathbf{x}|$ do
3          for $b \leftarrow : |\mathbf{y}|$ do
               // The function $\delta(x(a), y(b))$ can be any distance function. In our work,
                  as suggested in [1], we have used the Euclidean distance
4              $C_{\ a,b} \leftarrow \delta(x(a), y(b))$
5          end
6      end
7      return $C$
// Once the cost matrix $C$ is computed, ComputeWarpPath is used to find the optimal
   warp path between x and y. In this function, we impose the monotonicity,
   continuity and boundary constraints.
8  Function ComputeWarpPath($C$)
       // This imposes the start boundary constraint
9      $W^*\ (1) \leftarrow (1, 1)$
10     $i \leftarrow 1$
11     $j \leftarrow 1$
12     $k \leftarrow 1$
13     while $W^*(k) \neq ((|\mathbf{x}|, |\mathbf{y}|))$ do
14         $k \leftarrow k + 1$
           // The function ComputeIndexOfClosestTuple gives the index of the closest
              tuple in the input set. In this case we provide only 1-step nearest
              cells to impose the monotonicity and continuity constraints
15         $(\hat{i}, \hat{j}) \leftarrow$ ComputeIndexOfClosestTuple$(C_{i,j}, \{C_{i+1,j}, C_{i+1,j+1}, C_{i,j+1}\})$
           // Add closest tuple indexes to the warping path
16         $W^*(k) \leftarrow (\hat{i}, \hat{j})$
17         $i \leftarrow \hat{i}$
18         $j \leftarrow \hat{j}$
19     end
20     return $W^*$
// To use the warp function for classification it must be the same length for each
   possible input vector functions x and y. To allow this, we proposed to compute a
   warp function approximation via ComputeWarpPathApproximation
21 Function ComputeWarpPathApproximation($W^*$)
22     for $a \leftarrow 1 : |\mathbf{x}|$ do
           // The first and last values of the approximated warp path must be always
              1 and the length of the input vector functions, respectively
23         if $a == 1$ or $a == |\mathbf{x}|$ then
24             $f(a) = a$
25         else
               // The ComputeMinIndexWarpPath function is Eq.(3) of the main paper
26             $minb$ = ComputeMinIndexWarpPath$(W^*, a)$
27             $f(a) = minb$
28         end
29     end
30     return $\mathbf{f}_{(\mathbf{x},\mathbf{y})}$
```