Computer Vision Exercise 8: Shape Context

Soomin Lee (leesoo@student.ethz.ch)

December 4, 2020

1. Shape Context Descriptors

To construct a shape context descriptor, we define a log-polar coordinate system with each point as origin and count the number of points inside each bin. One can refer to Fig. 3 in the paper[1].

Given a range of the radius r and number of bins for each dimension r and θ, we can construct a grid, with log-scale in r direction. Then for each point, we regard the point as the origin and convert the Cartesian coordinates to the polar coordinates. We also normalize the radial distances using the mean of the distances between all point pairs in the shape for increased robustness. Lastly, we count the number of points in each bin by constructing a histogram. An example of such a histogram is shown in Figure 1.

Q. Is the shape context descriptor scale-invariant? It is scale-invariant since we normalize all the radial distances, where the scaling factor of the normalization is the mean distance between all point pairs in the shape.

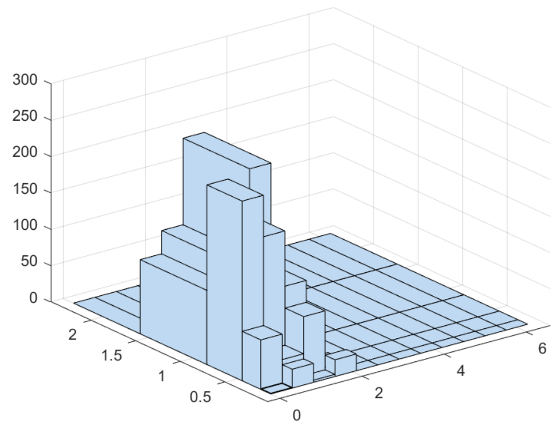


Figure 1: An example shape context. The axis with a uniform grid is θ direction, and the axis with a logarithmic scale grid is r direction.

2. Cost Matrix

Once we have two sets of shape context descriptors from the template and target shapes, we need to compute a cost matrix between them. Here we use the χ^2 test statistic to define the cost of matching two sets of points, using the same equation as in the paper. The resulting cost matrix has a size of $n \times m$, where n, m are the number of points of each shape, respectively. When the denominator of the cost equation is 0 and the value becomes NaN, it was replaced with 0 when summing up the values so that it has the same effect as skipping the pairs as in the code of the paper[2].

[1] Belongie, Serge & Malik, Jitendra & Puzicha, Jan. (2002). Shape Matching and Object Recognition Using Shape Contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence. 24. 10.1109/34.993558.

[2] https://github.com/a554b554/ShapeMatching/blob/master/shapematching.cpp#L10

3. Hungarian Algorithm

Given the cost matrix, we find the matching that minimizes the cost using the Hungarian method. The code is already provided by Niclas Börlin. Yet, since the function only takes a square matrix as the input argument, we need to handle the case when the number of points of each shape is different from each other before running the Hungarian algorithm.

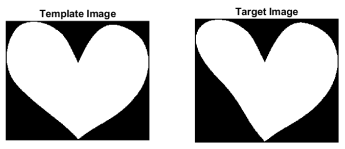
In the code, two different methods are implemented. First, as suggested in the paper, we can add "dummy" nodes to each point set with a constant matching cost of ϵ. The cost ϵ also serves as the threshold parameter for outlier detection, as a point will be matched to a "dummy" when there is no match available at cost smaller than ϵ. Second, instead of using the whole data, we can sample a fixed number of points from each shape so the length is identical from the beginning. The comparison between the two methods are shown in Figure 2.

4. Thin Plate Splines

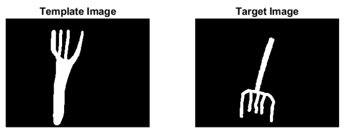
From the correspondence, now we can estimate a transformation from template to target points using thin plate splines. This is done by solving the system described in $Ax = b$ format in the Equation (8) in the paper. A is $[K \mid P;\ P^T \mid 0_{3\times3}]$, x is $[w \mid a]^T$, and b is $[v \mid 0]^T$. Then the total energy is computed as $w_x^T K w_x + w_y^T K w_y$. Figure 3 shows the result of the transformation applied to the shape. The value of lambda that was used for regularization is set as the square of the mean distance between two target points.

5. Results of other datasets

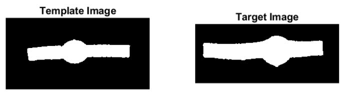
There are three classes in the provided datasets: *heart*, *fork*, and *watch*. So far, the figures that are presented used *heart* dataset, index 1 as the template and index 2 as the target in specific. Same process is repeated on other datasets, namely *fork* and *watch*, and the results are shown in Figure -. Also, the original image pair from each class of the dataset is illustrated in Figure 4.



(a) *heart*.



(b) *fork*.

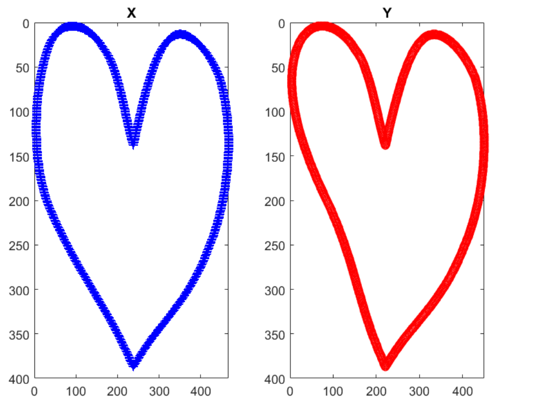


(c) *watch*.

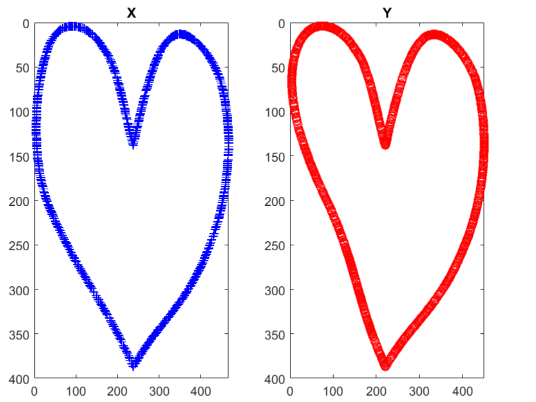
Figure 4: Images from each class that are used.

For *fork*, index 9 is used as the template and index 7 is used as the target. The size is matched by randomly sampling 1000 points from each shape. The result after 3 iterations is shown in Figure 5. The template has been rotated to match the target, but the result is not as good as *heart* or *watch* example. One could try to increase robustness to rotation by using a relative frame instead of the absolute frame for computing the shape context at each point, as proposed in the paper.

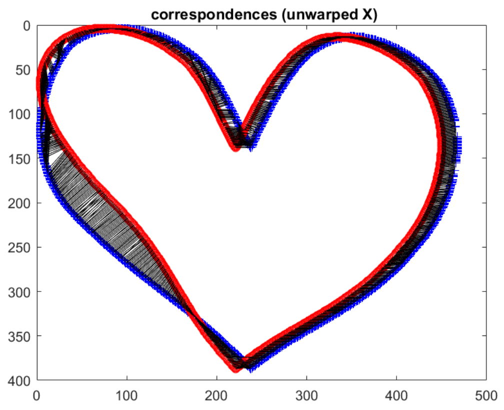
For *watch*, index 12 is used as the template and index 11 is used as the target. The size is matched by adding the dummy nodes. The result after 4 iterations is shown in Figure 6. When using the dummy nodes method, note that the correspondences that go to dummy nodes will get discarded and thus it might be beneficial to have the point set with a smaller number of points as template. Also, since false matches significantly influence the transformation parameters, it seems like there are certainly some cases where having dummy nodes as outlier detectors help improving the results. Nonetheless, the Hungarian algorithm runs in polynomial time so it is also worth considering the sampling method as well. One could also use both methods together.



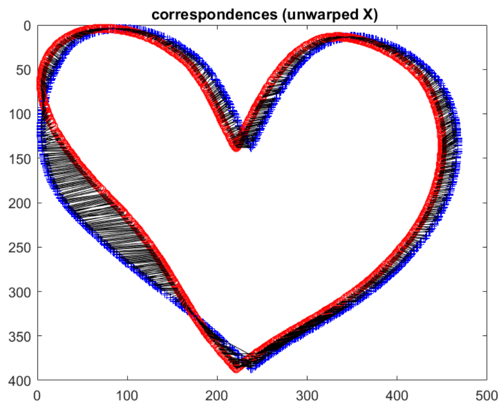
(a) Original template (X, left) and target (Y, right) shape.



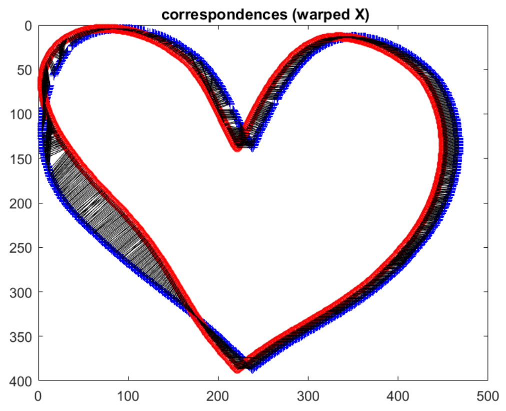
(b) Sampled template (X, left) and target (Y, right) shape.



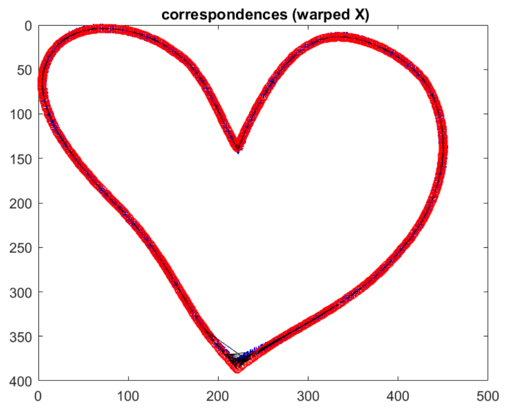
(c) Correspondences with unwarped X and Y.



(d) Correspondences with unwarped X and Y.

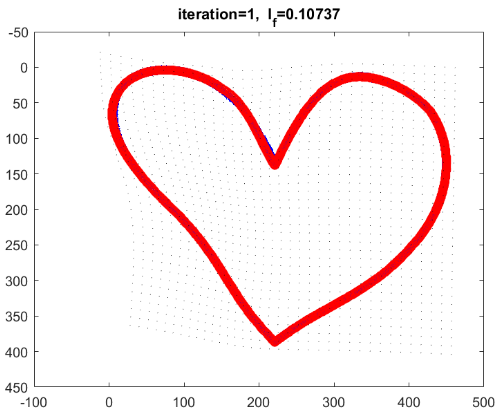


(e) Correspondences with warped X and Y.

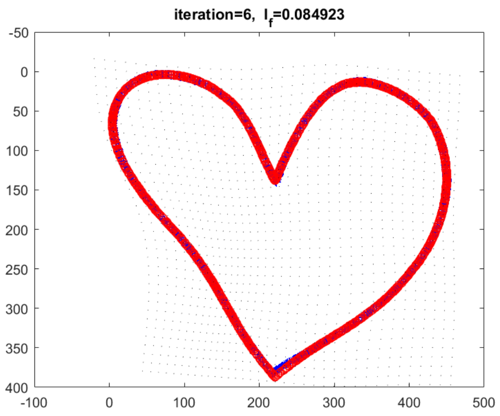


(f) Correspondences with warped X and Y.

Figure 2: Introducing dummy nodes (left) and downsampling (right) to get a square cost matrix. For downsampling method, 1000 points are sampled from each shape for this example.

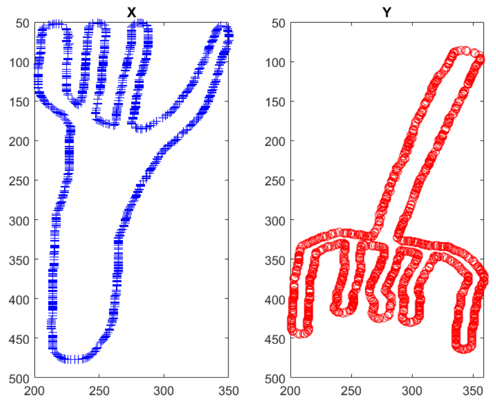


(a) Transformed template after 1 iteration (dummy nodes method).

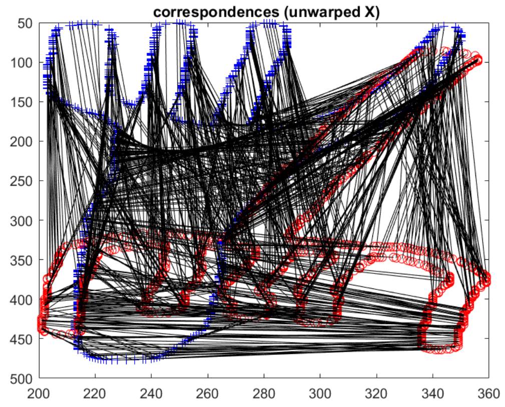


(b) Transformed template after 6 iterations (downsampling method).

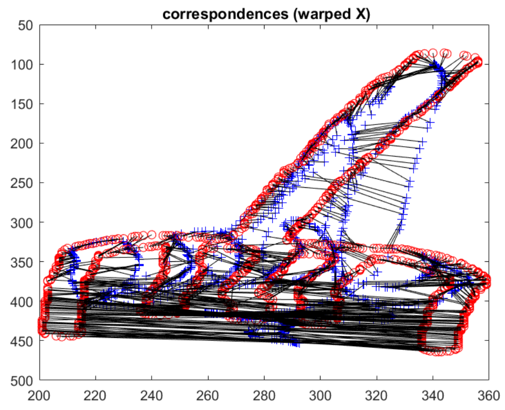
Figure 3: Transformed template with the parameters found with thin plate splines. Left is the result of dummy nodes method and right is the result of downsampling method.



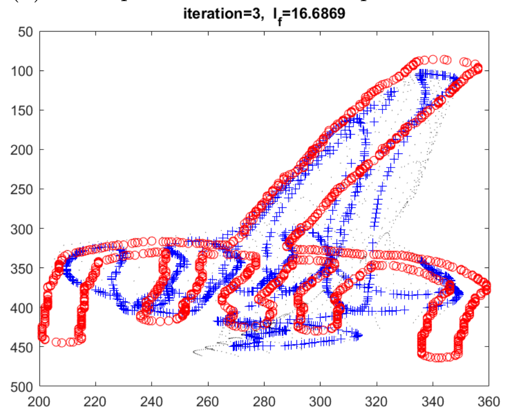
(a) Original template (X, left) and target (Y, right) shape.



(b) Correspondences with unwarped X and Y.

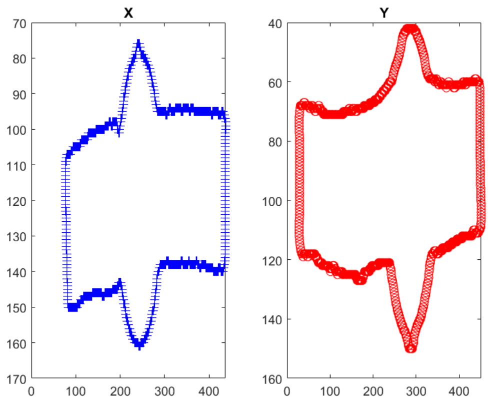


(c) Correspondences with warped X and Y.

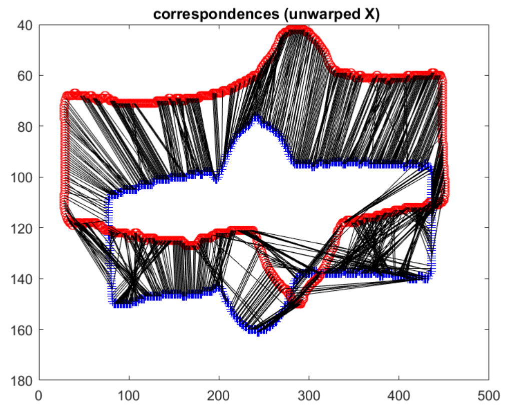


(d) Transformed template after 3 iterations (downsampling method).

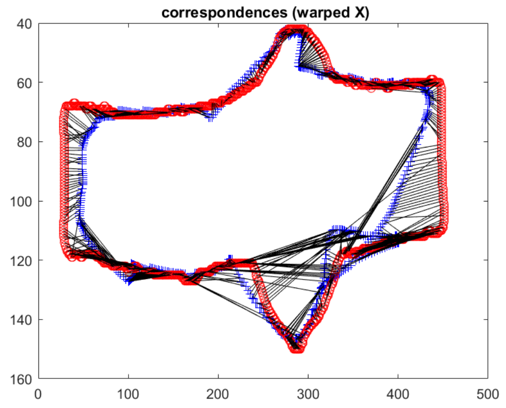
Figure 5: Plots for dataset *fork*.



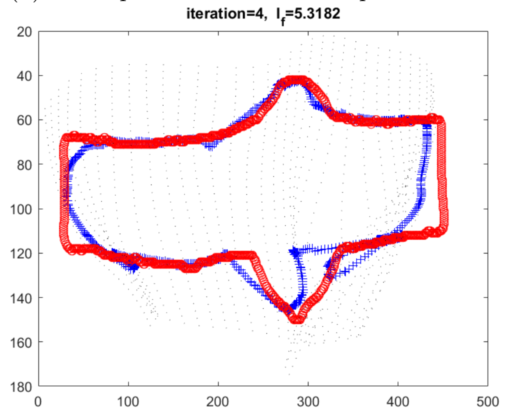
(a) Original template (X, left) and target (Y, right) shape.



(b) Correspondences with unwarped X and Y.



(c) Correspondences with warped X and Y.



(d) Transformed template after 4 iterations (dummy nodes method).

Figure 6: Plots for dataset *watch*.