Computer Vision

Assignment 8: Shape Context

Student: Ivan Alberico

1. Shape Context Descriptors

First step in this task was to implement the `sc_compute.m` function that computes the shape context descriptor for each sampled point of a specific shape. The implementation of the shape context descriptor follows the algorithm proposed in the paper '*Shape Matching and Object Recognition Using Shape Contexts*', as it was indicated in the assignment description. According to this paper, for each point of the shape, it is necessary to derive the histogram of the relative distances from all the other points, expressed in a polar reference frame.

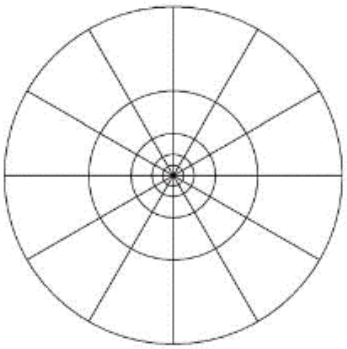


Figure 1 — The image represents the diagram of log-polar histogram bins used in computing the shape contexts. There are 5 bins for $\log(r)$ and 12 bins for θ. The figure is directly taken from the paper "*Shape Matching and Object Recognition Using Shape Contexts*".

Given the number of bins for both θ and r, as well as the max and min values of r, I first compute the vectors defining the edges of the polar reference frame. The vector θ is equally distributed from the values 0 and 2π, according to the number of bins. To obtain the vector r, instead, I take the log of the equally distributed points between `smallest_r` and `biggest_r`, and by doing this, I obtain bins having non-uniform lengths (in the paper mentioned before, the length of the bins along the radial dimension is not equal, as shown in *Figure 1*). These vectors are computed through the `linspace` function in Matlab.

After that, for each point, I compute the relative distances from the other $N-1$ points, I transform these quantities in polar coordinates (through the Matlab function `cart2pol`) and I normalize the resulting radial dimension by the mean distance of the distances between all point pairs in the shape (obtained though the provided function `dist2.m`). The histograms are then computed with the `hist3` function, by setting the edges equal to the vectors θ and $\log(r)$ previously computed, and they are stored in a $N \times 1$ cell array, where N is the number of sampled points from the shape.

2. Cost Matrix and Hungarian Algorithm

Next step was writing the function `chi2_cost.m`, which computes the cost matrix (χ^2 test statistic) between two sets of shape context descriptors s_1 and s_2. Each element of this cost matrix is given by the following:

$$C_{gh} = C(g,h) = \frac{1}{2}\sum_{k=1}^{K}\frac{[\,g(k)-h(k)]^2}{g(k)+h(k)}$$

where $g(k)$ and $h(k)$ are the two shape context histograms and K is the total number of bins, namely K = `nbBins_theta` x `nbBins_r`.

However, while computing the cost matrix, some minor issues were encountered. There were cases in which I had a 0 in the denominator of the cost function, leading to `NaN` values in the matrix. This issue could be overcome by either setting those `NaN` values to 0, or by adding a small quantity ε in the denominator, hence avoiding a division by 0 in the calculation. Both ways were tested and worked well for the task. Another constraint was that the cost matrix needed to be a square matrix, since the `hungarian.m` function worked only with square matrices.

3. Thin Plate Splines

After having computed the shape context descriptors and the cost matrix, last step of the assignment was about implementing in the function `tps_model.m` a transformation that maps the points from one shape to the other. To do so, Thin Plate Splines model was used, having the following form:

$$f(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{n} \omega_i \, U(\|(x_i, y_i) - (x,y)\|)$$

Since this model gives a 1D output and we are interested in the transformation of both the vertical and horizontal components, 2 independent TPS models are needed in order to compute the warping coefficients of our transformation. For each component, we need to solve the following linear system:

$$\begin{pmatrix} K + \lambda I & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \omega \\ a \end{pmatrix} = \begin{pmatrix} v \\ 0 \end{pmatrix} \qquad \Rightarrow \qquad \begin{pmatrix} \omega \\ a \end{pmatrix} = \begin{pmatrix} K + \lambda I & P \\ P^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} v \\ 0 \end{pmatrix}$$

Being N the number of points for each shape (which is the same for both the template and the target shape, since it is required from the Hungarian algorithm), we have the following:

- $K_{ij} = U(\|(x_i, y_i) - (x,y)\|)$ is a $N \times N$ matrix;
- λ is a scalar, namely the regularization parameter, which is set to be equal to the square of the mean distance between two target points;
- I is the $N \times N$ identity matrix;
- $P = (1 \quad x \quad y)$ is a $N \times 3$ matrix, where x and y are the coordinates of the template points;
- a is a 3×1 column vector containing the coefficients a_1, a_x and a_y of the model;
- ω is the $N \times 1$ column vector of the warping coefficients of the model (we will have respectively ω_x and ω_y for the two TPS models);
- v is the $N \times 1$ column vector of the target point coordinates (we will have respectively v_x and v_y for the two TPS models).

Finally, the bending energy is computed by summing up the two energy terms coming from both TPS models, as in the following:

$$E = {\omega_x}^T K \omega_x + {\omega_y}^T K \omega_y$$

After that all the previous functions have been implemented, I can now run the `shape_matching.m` function, which iteratively computes the shape context descriptors and the cost matrices, runs the Hungarian algorithm and finally obtains the TPS model.

To test the code, I implemented a `main.m` script file, in which I first load the dataset and then I select two contours of the same type ('heart', 'fork' or 'watch') to be matched. The dataset provides a set of points for each contour, and from the two previously selected shapes, one is set to be the template and the other the target. However, in the `shape_matching.m` function we do not input all the points of the selected shapes, but only a number of sampled ones (the sampling is done through the `get_samples.m` function provided in the code).

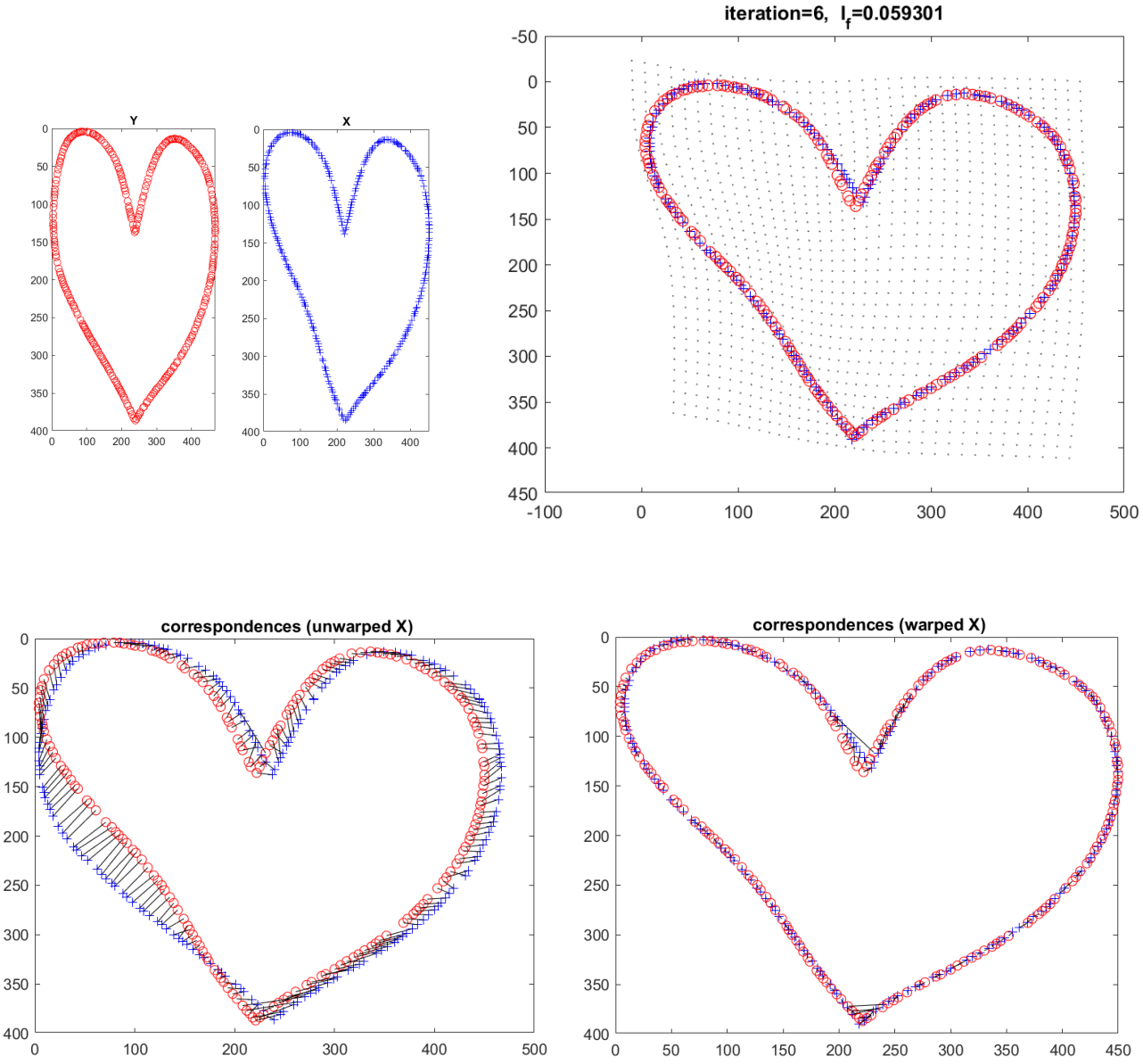


Figure 2 — Shape matching between template shape #1 and target shape #2. The matching is performed by taking 200 samples from the original shape points and 6 iterations were executed. The final bending energy is equal to 0.059301.

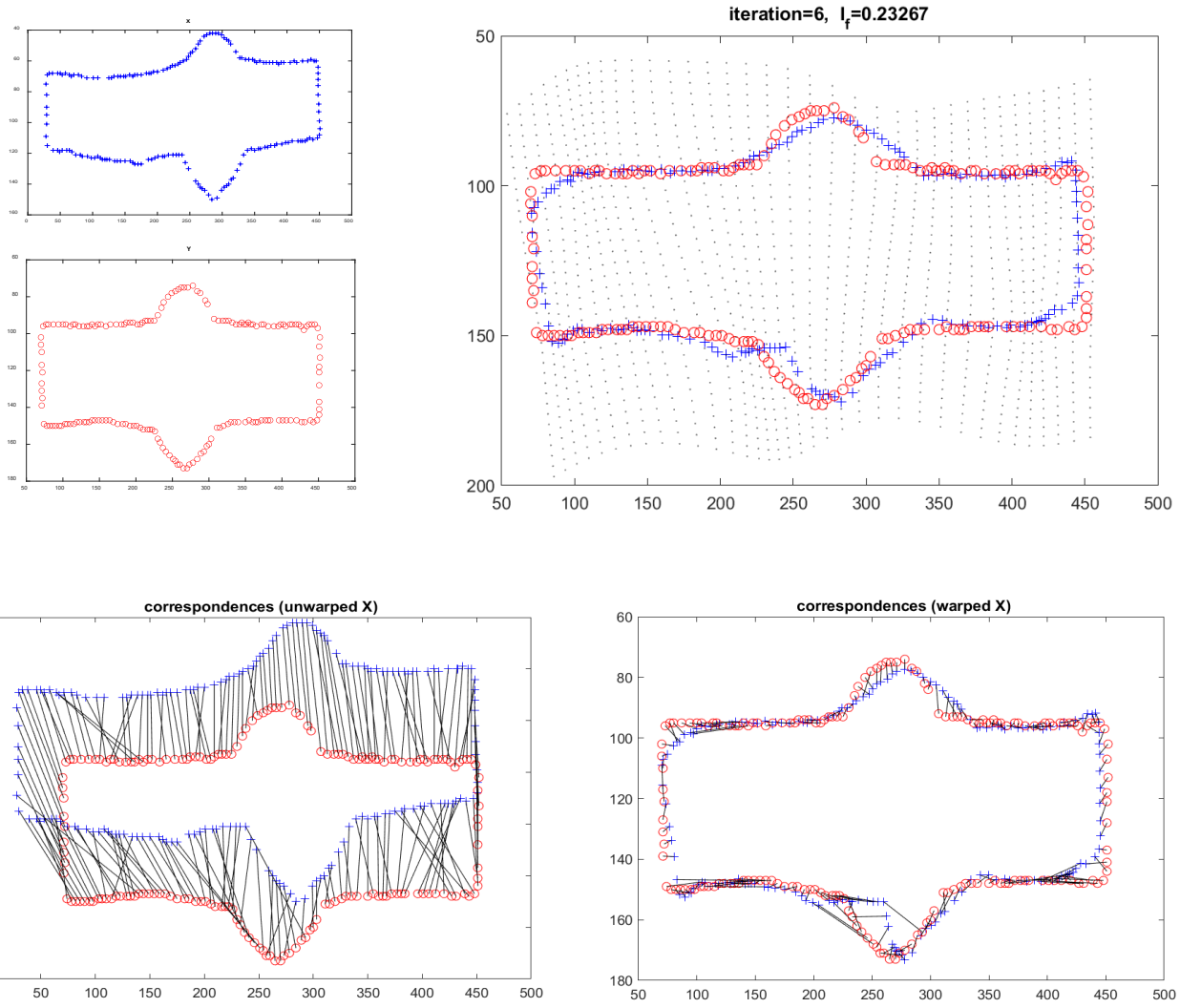
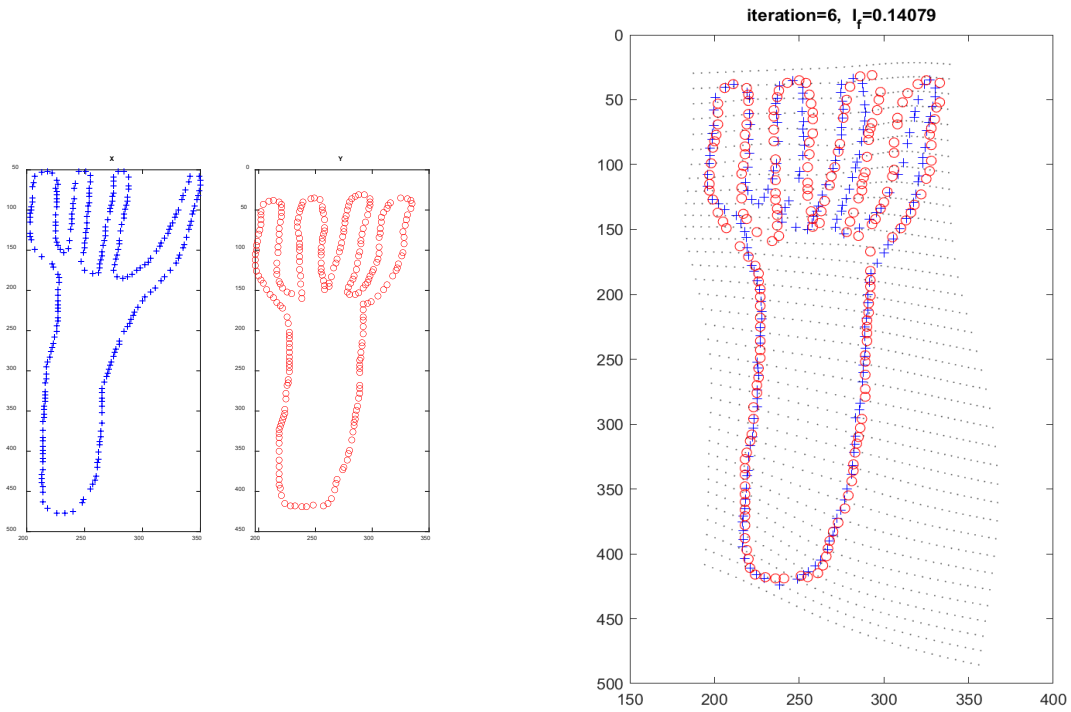


Figure 3 — Shape matching between template shape #11 and target shape #15. The matching is performed by taking 180 samples from the original shape points and 6 iterations were executed. The final bending energy is equal to 0.23267.



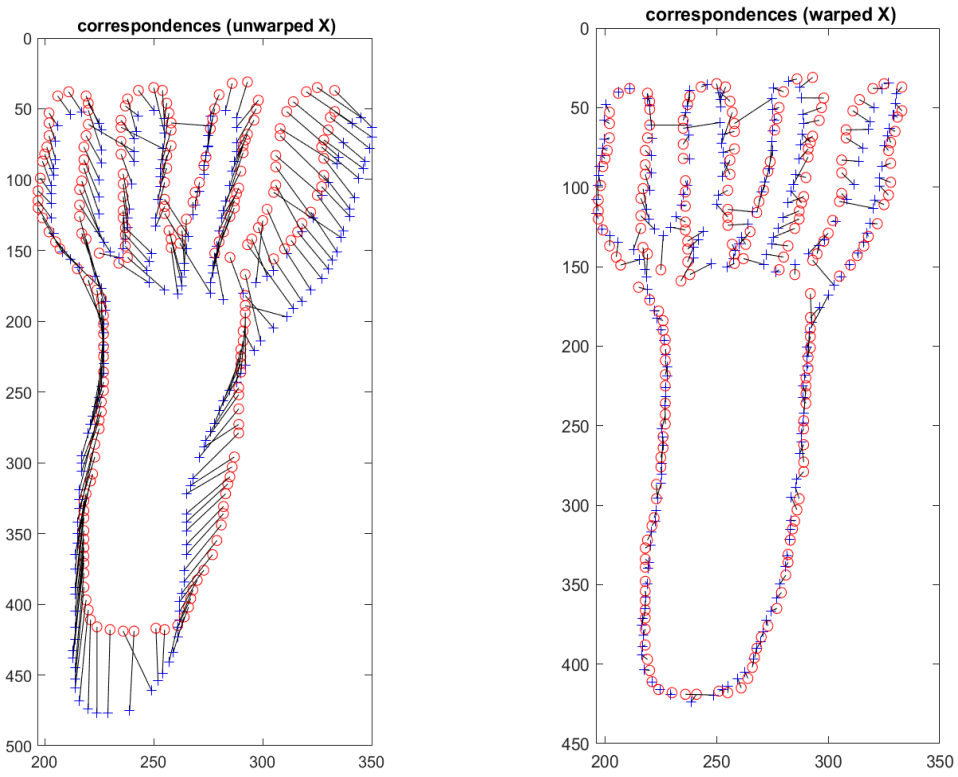


Figure 4 — Shape matching between template shape #9 and target shape #8. The matching is performed by taking 200 samples from the original shape points and 6 iterations were executed. The final bending energy is equal to 0.14079.

The implemented algorithm seems to work correctly in all the 3 cases, as it can be seen from the previous images. Of course, the more complex the shape is the less accurate result we are going to obtain. In fact, in the case of the 'heart' shapes, the two contours seem to almost overlap in the end and the bending energy is very close to zero (it is around 0.05). However, in the case of the 'fork' or the 'watch', which are shapes that are indeed slightly more complex, even though the matching is good, we still do not have a very precise overlapping, with a slightly increased bending energy. The result also depends a lot on the number of initial samples we consider for the shapes. For a larger number of samples, besides the fact that the algorithm is computationally much more expensive, it is also more likely to get a bigger number of wrong matches, and in fact in that case I obtained slightly higher values of the bending energy.

QUESTION: Is the shape context descriptor scale-invariant? Explain why or why not.

Shape context descriptors are scale-invariant. This is the result of normalizing the radial distance r by the mean distance between all the point pairs in the shape. In this way, even if we scale the shape by some factor, this would not influence the distances we use to generate the shape context (they are shaped by the same factor).

We could achieve scale-invariance also in other ways. One, for example, could be normalizing the radial distances by the median distance between all the point pairs in the shape, instead of the mean. This solution would be more robust to outliers in the matchings.