Computer Vision
Exercise 8: Shape Matching

Viviane Yang 16-944-530

26th November 2019

1 Shape Matching

In this task, we are supposed to find the shape context descriptors, which are descriptors describing the characteristics of a set of points in terms of their shape. With these descriptors, we then compute the thin plate spine model $T(x, y) = (f_x(x, y), f_y(x, y))$ which transforms a set of points X to the set of points Y.

1.1 Shape Context Descriptors

The shape context descriptors count the number of points in buckets around each point. A circle which is divided by radius and angle describes the different buckets. For the radial component, we consider a log scale, which differentiates closer points more (has more buckets) than points that are further away to the point that is being examined.

```
%spacing for radial component
r = linspace(log(smallest_r), log(biggest_r), nbBins_r);
%spacing for angular component
theta = linspace(-pi, pi, nbBins_theta);
```

For each point in the set of points X we need to compute a separate histogram, so the multiple histograms are stored into a cell data structure d. We first compute the distance to all points, then translate the cartesian distance vectors into polar coordinates ($\theta \in (-pi, pi)$) and then using MATLAB's `hist3` function to compute the histogram. For the radial component I am normalizing the value by the mean distance of all pairwise distances in X as mentioned in the task description.

```
for p = 1:n
    px = X(p, :);
    %compute distance to all
    dist_vec = repmat(px, size(X,1), 1) - X;
    dist_vec(p, :) = [];
    [TH, R]= cart2pol(dist_vec(:,1), dist_vec(:,2));
    %save histogram to cell
    d{p} = hist3([TH log(R/meanDistance)], 'Edges', {theta r});
end
```

1.2 Cost Matrix

After the shape context descriptor extraction from the point set X and Y, we now want to compute the cost matrix between the two sets of points. The component of the cost matrix C_{gh} is described in equation (1):

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

1.2.1 Avoiding division by zero

However, when computing C_{gh} for some cases, when the buckets $g(k)$ and $h(k)$ are both empty, we are dividing by zero. To bypass this edge case, I am adding *eps* to the denominator in order to not divide by zero.

```
function c = chi2_cost_vector(g, h)
    c = sum(((g - h).^2)./((g+h)+eps))/2;
end
```

1.2.2 Handling different dimensions

The set of shape context descriptors has one descriptor for each pixel, hence if there is a different number of pixel-samples in X which is n and Y which is m, one also has different dimensions of the sets. The cost matrix is a $n \times m$ matrix, which does not need to be square. However, for the computation of the Hungarian algorithm, a square matrix $\tilde{C}$ is needed. Therefore, I am currently checking which of the dimensions is smaller and just take the submatrix of the smaller dimension.

```
1 if(n < m)
2     C = C(1:n, 1:n);
3 elseif(n > m)
4     C = C(1:m, 1:m);
5 end
```

Afterwards, I am ensuring that X and Y have the same number of points, by sampling n samples from both sets, such that there is no need for the modification anymore.

1.3 Sampling

In the paper [1] it is mentioned in Appendix B, that one should use a subset of the set of points, since the set of points might directly come from a edge detector. With sampling methods, we can control the number of points that are used for the shape matching algorithm. Especially we are interested in having the same number of samples in both sets X and Y for the Hungarian algorithm, since it finds a bipartite matching between both sets and therefore needs X and Y to have the same dimension.

1.3.1 Uniform sampling

At first, I am just randomly sampling points from the set. For this I am using the `datasample` function in MATLAB, which returns random rows from a matrix. When choosing a large number of samples $n = 400$, this methods works well because the original set of points was uniform distributed around the edges such that a large subset still has this characteristic.

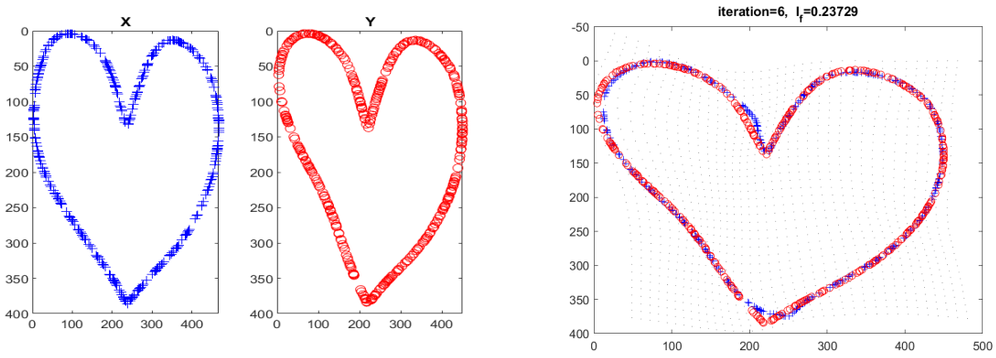

Figure 1: Random sampling with 400 samples

However, if we use a smaller number of samples $n = 100$, for example due to time constraints, we might get points that are very close together and then further apart again. The points are not uniformly distributed anymore and the transformation loses accuracy.

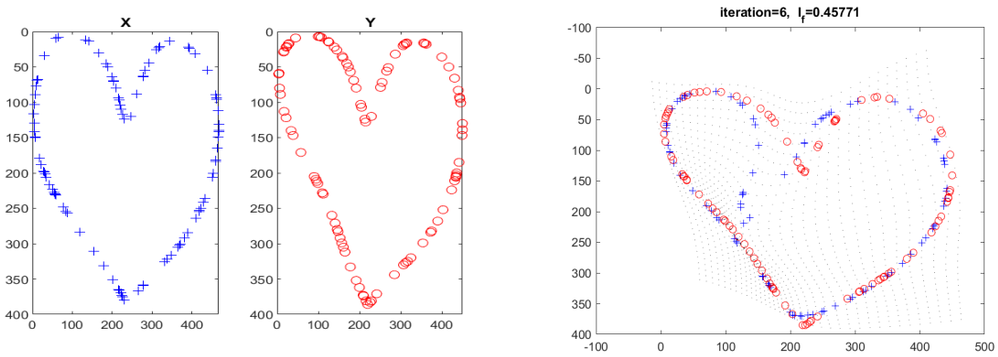

Figure 2: Random sampling with 100 samples

1.3.2 Minimum distance sampling

To tackle this problem, it was also mentioned in the paper that one should sample with a minimum distance requirement. Using a hard threshold however does not work, since we don't now how dense the contour points are and how many points to sample we can't set the minimum distance. However

we can start with all points and remove the point that has the smallest pairwise distance to one of the other points. We stop removing points when the number of points we want to sample is reached. The algorithm looks like the following:

```
function X_sampled = get_sample(X, n)
N=size(X, 1);
if(N < n)
    X_sampled = X;
else
    index = randperm(N);
    %ensure the points are distant to each other by choosing the n ones
    %with max distance
    dist = dist2(X, X);
    dist = dist + diag(Inf*ones(N,1) - diag(dist)); %set distances to self high

    while length(index) > n
        %remove the point that has the smallest distance in distance
        %matrix.
        [col_min, idx] = min(dist);
        [~, col] = min(col_min);
        row = idx(col);
        %set this to high, dist is symmetric
        dist(row, :) = ones(1,N)*Inf;
        dist(:, row) = ones(N,1)*Inf;
        %remove one of the indexes ( here its the first one )
        index(find(index == row)) = [];
    end
    X_sampled = X(index, :);
end
```

The resulting sampled points look more uniformly distributed and we get a better result for the transformation. We can reduce the complexity of the algorithm by using less points.

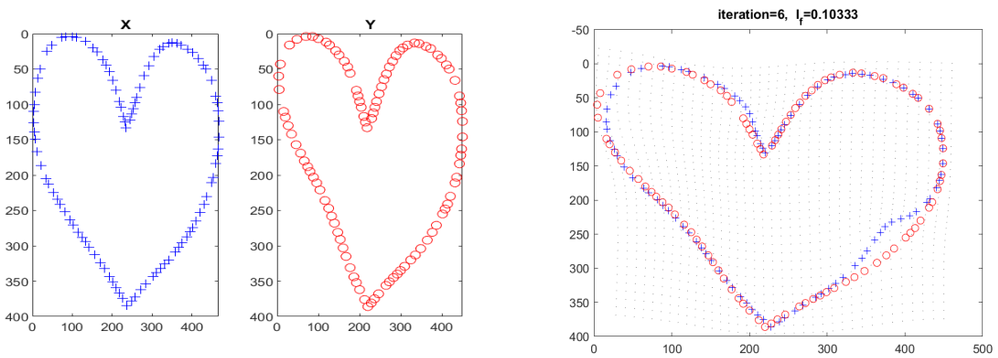

Figure 3: Minimum Distance Sampling with 100 samples

1.4 Thin Plate Spines

In this part, we want to find a thin plate spine model (TPS) or a plane transformation using the point correspondences. The math is described extensively in the exercise sheet. We need to solve 2 linear equation systems for the 2 coordinates x and y. The energy can then be computed as the sum of both energies E_x and E_y. With the parameters a and w the transformations f_x and f_y can be applied to the set of points X which is iteratively used to better match Y.

```
function [w_x w_y E] = tps_model(X,Y,lambda)
    n = size(X, 1);
    K = U(sqrt(dist2(X,X)));
    P = [ones(n,1) X];
    A = [K+lambda*eye(n) P; P' zeros(3)];
    vx = Y(:, 1);
    vy = Y(:, 2);
    w_x = A \ [vx; zeros(3,1)];
    w_y = A \ [vy; zeros(3,1)];

    ax = w_x(end-2:end);
    ay = w_y(end-2:end);

    E_x = w_x(1:n)'*K*w_x(1:n);
    E_y = w_y(1:n)'*K*w_y(1:n);
    E = E_x + E_y;
end


function u = U(t)
```

```
21        u = t.^2 .* log(t.^2);
22        u(isnan(u)) = 0;
23    end
```

2 Results

Here are some results of the finished shape matching algorithm.

2.1 Heart

Figure 4: Shapes of hearts, X is left and Y is right

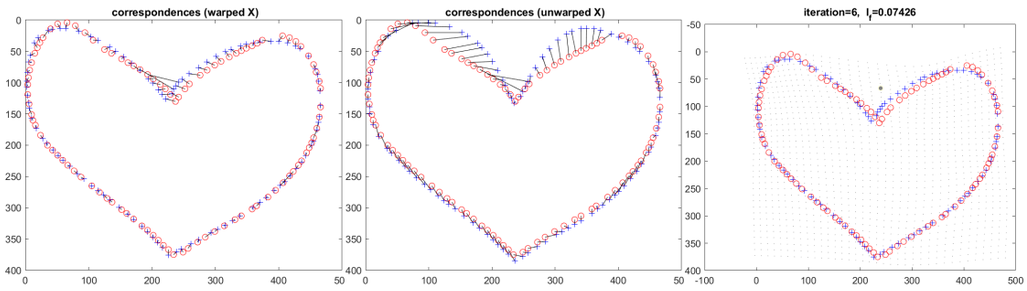

Figure 5: Correspondences after 6 iterations as well as the final transformation mapping from X to Y

2.2 Fork

Figure 6: Shapes of forks, X is left and Y is right

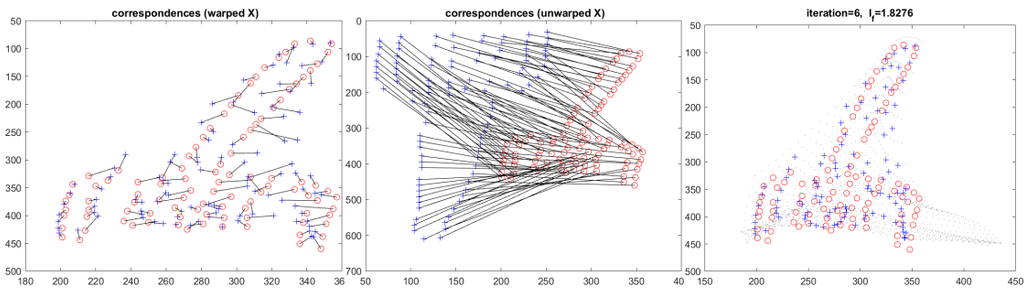

Figure 7: Correspondences after 6 iterations as well as the final transformation mapping from X to Y. As you can see, the shape context descriptors are invariant to rotation or translation, since only the distances between the points matter.

2.3 Watch

Figure 8: Shapes of watches, X is left and Y is right

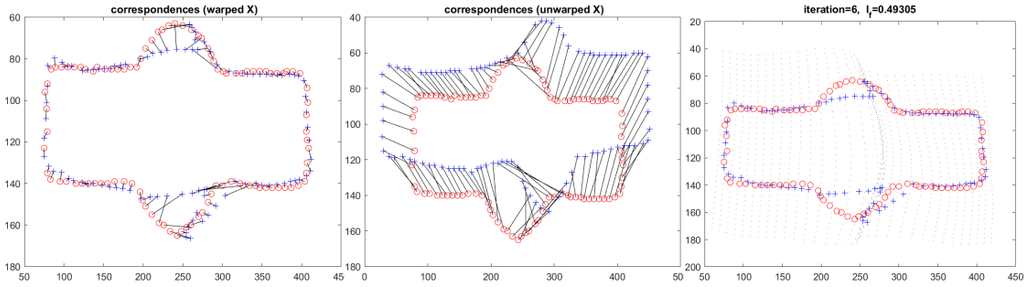

Figure 9: Correspondences after 6 iterations as well as the final transformation mapping from X to Y

3 Questions

3.1 Is the shape context descriptor scale-invariant?

The shape context descriptor of a point uses the distance vector to every other point. This distance vector has different lengths for the same shape at different scales, so that the shape context descriptor is not scale invariant without normalization. However, if one normalizes the distance by the mean of all distances as we do in this exercise, it can be made scale invariant. Now, the relative distance of every set of points corresponding to the same shape should be the same.

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, April 2002.