

Stefan Breuers <breuers@vision.rwth-aachen.de>
Wolfgang Mehner <mehner@vision.rwth-aachen.de>

Exercise 5: Matching, Homographies

due **before** 2017-01-09

Important information regarding the exercises:

- The exercise is not mandatory.
- There will be no corrections.
- Nevertheless, we encourage you to work on the exercises and present your solutions in the exercise class. For this regard the submission rules.
- In the archive for this exercise you will find the functions `apply.m` that should be used for displaying your results. You should also use it to test your implementation and see if the results make sense. Answers are to be submitted within `answers.m` Do **not** modify the `apply` files in any way.
- Please do **not** include the data files in your submission!
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (mn).
- Submit your solutions via the L²P system.

Question 1: Region Descriptors ($\Sigma = 0$)

In order to find correspondences between interest points, we need to design region descriptors. In this question, we will implement several **simple descriptors** based on the **histogram** representations from exercise sheet 4.

You can test your implementations either using the points you detected in the previous exercise or using the points from the file `ip_graff.mat`. This file contains points detected with the **Harris detector** using the parameter settings `sigma = 2.0, thresh = 100000`. The vectors `px1` and `py1` are point coordinates from the image `graff5/img1.png` and `px2` and `py2` are point coordinates from the image `graff5/img2.png`.

The functions `histr_g` and `histr_dxdy` are modified versions of `myhist3` and `myhist4` from a previous exercise sheet, which take the input image as a first parameter.

- (a) As an example, we provide the function `descriptors_rg` which takes an input image and a list of interest points and computes **an r, g color histogram over** the $m \times m$ sub-windows **around each interest point** (using the function `histr_g`).

Write a similar function `descriptors_dxdy` which computes **a dx, dy histogram** around each interest point (using the function `histr_dxdy`).

Hint: For our application here, suitable descriptor parameters are a window size of `m=41`, a histogram resolution of `bins=16` and `sigma=2.0`.

```
1 function D = descriptors_dxdy(img, px, py, size, sigma, bins)
```

Visualize the histograms for some interest points. Do they look useful?

- (b) Write a another function `histr_maglap` and `descriptors_maglap` which compute a *mag, lap* (**gradient magnitude, laplacian**) histogram around each interest point. For this descriptor, you can also use the parameters from question a.

```

1 function h = histmaglap(img, sigma, bins)
2 function D = descriptors_maglap(img, px, py, size, sigma, bins)

```

Visualize the histograms for some interest points. Do they look useful?

- (c) Given two histograms h_1 and h_2 with D bins, the χ^2 distance is defined as following:

$$\text{dist_chi2}(h_1, h_2) = \sum_{i=1}^D \frac{(h_1(i) - h_2(i))^2}{h_1(i) + h_2(i)}$$

In the nominator it matches the **squared Euclidean distance**, and in the denominator each cell is **normalized by the total sum**. This means that the cells are not weighted equally anymore, but bins with less data have a higher impact on the distance. This has a statistical background. The χ^2 distance tests if the underlying distributions of the histograms are different. In general the χ^2 distance is more robust to outliers, given the histograms contain enough observations.

Write a function `findnn` which takes two sets of region descriptors `D1` and `D2` and tries to find for each descriptor in `D1` the nearest neighbor in `D2` using the Euclidean distance (Matlab command **norm**). The function should return the indices `Idx` and distances `Dist` of the nearest neighbors. Create a second function `findnn_chi2` which does the same using the χ^2 distance.

```

1 function [Idx, Dist] = findnn(D1, D2)
2 function [Idx, Dist] = findnn_chi2(D1, D2)

```

Find the best match using `descriptors_rg` and display the matching histograms. Do they look similar enough?

Question 2: Matching ($\Sigma = 0$)

Now we have all the components for a small matching application. In the exercise archive you will find the folders `graff5` and `NewYork` that contain test scenes with controlled image-plane rotations and viewpoint changes, for which we will try to find point correspondences.

- (a) First we want to try out the color descriptors. Load the two example images `graff5/img1.png` and `graff5/img2.png` and perform the following steps.
 1. Compute Harris interest points for both images. (Use a high threshold to limit the number of interest point to less than 1000)
 2. Compute r/g color histogram descriptors for all interest points.
 3. Find the best matches using the functions `findnn` and `findnn_chi2`.
 4. Use the function `displaymatches` to visualize the N best matches. What do you observe?
- (b) Now, try the same with Hessian interest points and the dx/dy histogram descriptors. What do you observe? Which combination gives the better results? Can you think of an explanation?
- (c) Next, we will try to find matches under image plane rotations. The folder `NewYork` contains a series of test images for which the true homographies are known. In the exercise archive, you can find a small Matlab program `hom_gui_H` which lets you visualize the corresponding point locations. Load the two images `NewYork/im1.png` and `NewYork/im5.png` and start the program as follows. This will open a window showing the two images side by side. Click on one image and describe what happens.

```

1 img1 = double(imread('NewYork/im1.png'));
2 img2 = double(imread('NewYork/im5.png'));
3 H = load('NewYork/H1to5');
4 hom_gui_H(uint8(img1), uint8(img2), H);

```

- (d) Try to find matches between the two images NewYork/im1.png and NewYork/im5.png using Hessian interest points and the dx/dy histogram descriptors. What do you observe?

Now try the *mag/lap* histogram descriptors on the same image pair. What performance do you get? Which descriptor performs better? Why do you think that is?

Question 3: Homography Estimation ($\Sigma = 0$)

Since the test images show only planar scenes, we can try to estimate the homography \mathbf{H} which transforms one of the images into the other. In the following, we briefly repeat the procedure introduced in the lecture. In the literature, this procedure is known as the *Direct Linear Transformation* (DLT) algorithm.

For a point \mathbf{x}_r in the reference image and a corresponding point \mathbf{x}_t in the transformed image, the transformation can be written as follows:

$$\mathbf{H}\mathbf{x}_r = \mathbf{x}'_t \quad (1)$$

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = \begin{bmatrix} x'_t \\ y'_t \\ z'_t \end{bmatrix} \quad \text{with} \quad \frac{1}{z'_t} \begin{bmatrix} x'_t \\ y'_t \\ z'_t \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix}, \quad (2)$$

and we obtain a system of linear equations with 8 unknowns h_{11} , h_{12} , h_{13} , h_{21} , h_{22} , h_{23} , h_{31} , and h_{32} :

$$\frac{h_{11}x_r + h_{12}y_r + h_{13}}{h_{31}x_r + h_{32}y_r + 1} = x_t \quad (3)$$

$$\frac{h_{21}x_r + h_{22}y_r + h_{23}}{h_{31}x_r + h_{32}y_r + 1} = y_t \quad (4)$$

which can be rewritten as follows

$$h_{11}x_r + h_{12}y_r + h_{13} - x_th_{31}x_r - x_th_{32}y_r - x_t = 0 \quad (5)$$

$$h_{21}x_r + h_{22}y_r + h_{23} - y_th_{31}x_r - y_th_{32}y_r - y_t = 0 \quad (6)$$

In order to estimate these 8 parameters, we need at least 4 corresponding point pairs.

However, we can enhance the accuracy of the estimated homography by using more than 4 point pairs $(\mathbf{x}_1, \mathbf{x}'_1), \dots, (\mathbf{x}_n, \mathbf{x}'_n)$. This leads to an overdetermined system of equations

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y_1y'_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -x'_2y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y_2y'_2 & -y'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx'_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y_ny'_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (7)$$

$$\mathbf{A}\mathbf{h} = \mathbf{0} \quad (8)$$

which we want to solve by minimizing the least-squares error. If \mathbf{A} is square, we can directly obtain an exact solution. However, if the system is overdetermined (i.e. $n > 4$), the matrix \mathbf{A} is not square. This problem can be solved by building the so-called *pseudo-inverse* $\mathbf{A}^T \mathbf{A}$, which is square and can therefore be decomposed by eigenvalue decomposition. The solution is the (unit) eigenvector of $\mathbf{A}^T \mathbf{A}$ with least eigenvalue (Matlab command **eig**). Equivalently (and computationally more efficiently), the solution can be obtained by SVD as the unit singular vector corresponding to the smallest singular value of \mathbf{A} (Matlab command **svd**).

$$\mathbf{A} \stackrel{\text{svd}}{=} \mathbf{U} \mathbf{D} \mathbf{V}^T = \mathbf{U} \begin{bmatrix} d_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_{99} \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \cdots & v_{99} \end{bmatrix}^T \quad (9)$$

In this formulation, the eigenvector of $\mathbf{A}^T \mathbf{A}$ corresponding to the smallest eigenvalue minimizes the least-squares error to the solution for \mathbf{h} . We can therefore obtain the homography \mathbf{h} from the last column of \mathbf{V} (since we require $h_{33} = 1$, we normalize with v_{99}):

$$\mathbf{h} = \frac{[v_{19}, \dots, v_{99}]}{v_{99}} \quad (10)$$

- (a) Write a function `estimate_homography` which approximates the homography between two images from a set of point correspondences according to the procedure described above.
 1. Build up the matrix \mathbf{A} according to equation (7).
 2. Apply SVD to decompose the matrix using the Matlab command `[U, S, V] = svd(A)`. (*Caution:* Maybe you noted that the matrix \mathbf{V} is transposed (see Equation (9)). Matlab also returns a matrix that is transposed in the same way.)
 3. Compute \mathbf{h} according to Equation (10).
 4. Transform \mathbf{h} into a 3×3 matrix \mathbf{H} using the **reshape** command and then transpose it so it is row-major.

```
1 function H = estimate_homography(px1,py1,px2,py2)
```

- (b) Compute the best 10 matches between the two images `NewYork/im1.png` and `NewYork/im5.png` using Hessian interest points and the *mag/lap* histogram descriptors (Matlab command **sort**). Estimate the homography \mathbf{H} from the matches and compare it to the ground truth matrix from the file `NewYork/H1to5`. How accurate is your estimate? Does it improve when you take the first 20 or 50 matches instead? Why/why not?
- (c) Try the accuracy of your estimated homography using the demo program from Question 2. What do you observe?