Exercise 4: Feature points, matching, homography

Machine perception

2018/2019

Create a folder `exercise4` that you will use during this exercise. Unpack the content of the `exercise4.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as the *Matlab/Octave* scripts to `exercise4` folder. In order to complete the exercise you have to present these files to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the presentation as well. The tasks that are marked with ★ are optional. Without completing them you can get at most 75 points for the exercise (the total number of points is 100 and results in grade 10). Each optional exercise has the amout of additional points written next to it.

Introduction

In this exercise we will look at the problem of automatic searching for correspondences between two images. Image correspondences are a key step when we are dealing with the task of aligning two or more images, for example for generating a panorama from multiple images. Methods that find a correspondence between two images usually start by finding feature points in both images. These points denote regions in the image that have a high chance of re-detection in an image where the same scene is captured from a slightly different angle or viewing conditions. In the first assignment you will implement two such methods that will serve as as basis for the rest of the exercise. In the second assignment you will implement a simple stable region descriptor and use it to robustly match the detected feature points between two images. In the last assignment these matches will be used to implement a simple image alignment algorithm.

Assignment 1: Feature points detectors

In this assignment you will implement two frequently used feature point detectors: Hessian algorithm [1](str. 44) and Harris algorithm.

(a) Hessian detector is based on second derivatives matrix $\mathbf{H}(x, y)$ (also named Hessian matrix, hence the name of the algorithm) at point (x, y) in the image:

$$\mathbf{H}(x,y) = \begin{bmatrix} I_{xx}(x,y;\sigma) & I_{xy}(x,y;\sigma) \\ I_{xy}(x,y;\sigma) & I_{yy}(x,y;\sigma) \end{bmatrix}, \tag{1}$$

where σ explicitly states that the second derivative is computed on *smoothed image*. Hessian detector selects point (x, y) as a feature point if the determinant of the

Hessian matrix exceeds a given threshold value t. If we want to create a variant of the Hessian detector that is scale independent (independent of the Gaussian filter that is used to smooth the image) we have to include a normalization factor of σ^4 and thus we get a feature detection rule:

$$det(\mathbf{H}(x,y)) = \sigma^4(I_{xx}(x,y;\sigma)I_{yy}(x,y;\sigma) - I_{xy}(x,y;\sigma)^2) > t. \quad (2)$$

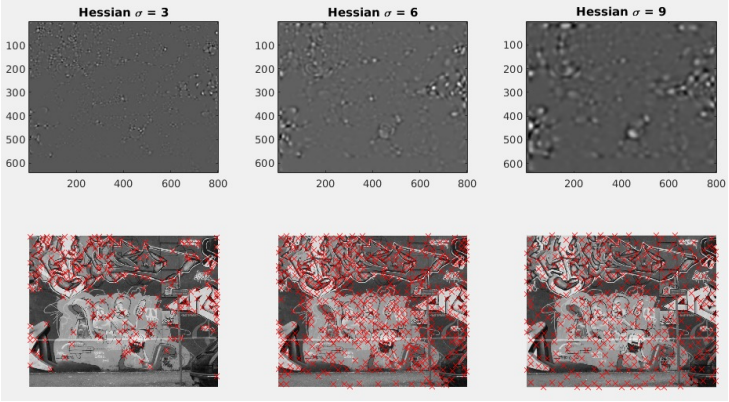
Implement a function `hessian_points`, that computes a Hessian determinant using the equation (2) for each pixel of the input image. As this computation can be very slow if done pixel by pixel, you have to implement it using vector operations (without explicit `for` loops). Test the function using image from `test_points.jpg` as your input (do not forget to convert it to grayscale) and visualize the result.

```
function I_hess = hessian_points( I, sigma )
% TODO
```

Extend the function `hessian_points` by implement a non-maximum suppression post-processing step that only retains responses that are higher than all the neighborhood responses and whose value is higher than a given threshold value `thresh`. In the result vectors `px` and `py` the function returns (x, y) coordinates of detected feature points.

```
function [px, py] = hessian_points( I, sigma, threshold )
% TODO
```

Create a function that you will use plot the detected points (as red plus symbols) over the input image. Load image `test_points.jpg`, process it and visualize the result. Set the input parameters of the Hessian detector to `thresh = 100` and $\sigma = 1$, and then change their values to get some intuition about the effect of the parameters. What kind of structures in the image are detected by the algorithm?



(b) Next, you will implement the Harris feature point detector. This detector is based on auto-correlation matrix $\mathbf{C}$ that measures the level of self-similarity for a pixel neighborhood for small deformations. At the lectures you have been told that the Harris detector choses point (x, y) for a feature point if both eigen-values of the auto-correlation matrix for that point are big enough. In a nutshell this means that the neighborhood of (x, y) contains two well defined rectangular structures – i.e. a

corner. Auto-correlation matrix can be computed using the first partial derivatives at (x, y) that are subsequently smoothed using a Gaussian filter:

$$\mathbf{C}(x, y; \sigma, \tilde{\sigma}) = \sigma^2 \begin{bmatrix} G(x, y; \tilde{\sigma}) * I_x^2(x, y; \sigma) & G(x, y; \tilde{\sigma}) * I_x I_y(x, y; \sigma) \\ G(x, y; \tilde{\sigma}) * I_x I_y(x, y; \sigma) & G(x, y; \tilde{\sigma}) * I_y^2(x, y; \sigma) \end{bmatrix}, \tag{3}$$

where $*$ stand for convolution operation. Computing eigen-values λ_1 and λ_2 of matrix $\mathbf{C}(x, y; \sigma, \tilde{\sigma})$ is expensive, therefore we rather use the following relations[1]

$$\det(\mathbf{C}) = \lambda_1 \lambda_2 \tag{4}$$

$$\text{trace}(\mathbf{C}) = \lambda_1 + \lambda_2 \tag{5}$$

to compute the ratio $r = \lambda_1 / \lambda_2$. If we assume that

$$\frac{\text{trace}^2(\mathbf{C})}{\det \mathbf{C}} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2 \lambda_2} = \frac{(r+1)^2}{r}, \tag{6}$$

we can express the feature point condition for (x, y) as:

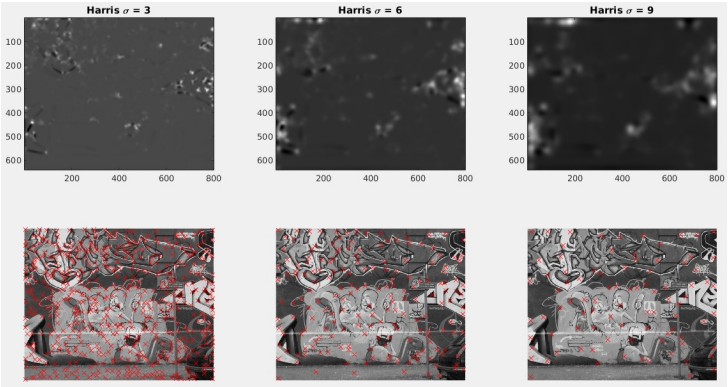
$$\det(\mathbf{C}) - \alpha \text{trace}^2 \mathbf{C} > t. \tag{7}$$

In practice we use $\tilde{\sigma} = 1.6\sigma$, $\alpha = 0.06$ for parameter values. Implement the condition (7) without using explicit computation of matrix $\mathbf{C}$ and without `for` loops, as you did for the core part of the Hessian detector.

Implement function `harris_points` that computes values of equation (7) for all pixels, performs non-maximum suppression post-processing step as well as thresholding using threshold `thresh`, The function returns feature point coordinates in vectors `px` and `py`.

```
function [px, py] = harris_points( I, sigma, thresh )
 ...
```

Load an image `test_points.jpg` and compute Harris feature points. Compare the result with detected feature points by the Hessian detector. Experiment with different parameter values. Do the feature points of both detectors appear on the same structures in the image?



[1] In the following text we will omit the parameters of the auto-correlation matrix $\mathbf{C}(x, y; \sigma, \tilde{\sigma})$ for clarity and write $\mathbf{C}$ instead.

Assignment 2: Matching local regions

One of the uses of feature points is searching for *similar structures* in different images. To do this we need visual descriptors of the regions around these points. In this assignment you will implement some simple descriptors as well as their matching.

(a) Write a function, `descriptors_maglap`, that computes a histogram of magnitude of partial image derivatives and Laplacian of a Gaussian around each point defined by vectory `px` and `py`. To compute the histograms use the function `features_maglap` that you can find in the supplementary material. Histograms are returned as rows in a result matrix `D`.

```
function D = descriptors_maglap(I, px, py, m, sigma, bins)
    rad = round((m - 1) / 2);
    [h, w] = size(I);
    D = zeros(length(px), bins ^ 2);
    [Imag, Ilap] = features_maglap(I, sigma, bins);
    for i = 1 : length(px)
        minx = max([px(i) - rad, 1]);
        maxx = min([px(i) + rad, w]);
        miny = max([py(i) - rad, 1]);
        maxy = min([py(i) + rad, h]);
        Rmag = Imag(miny:maxy, minx:maxx);
        Rlap = Ilap(miny:maxy, minx:maxx);
        d = accumarray(cat(2, Rmag(:), Rlap(:)), 1, [bins, bins]);
        D(i,:) = d(:) / sum(d(:));
    end
end
```

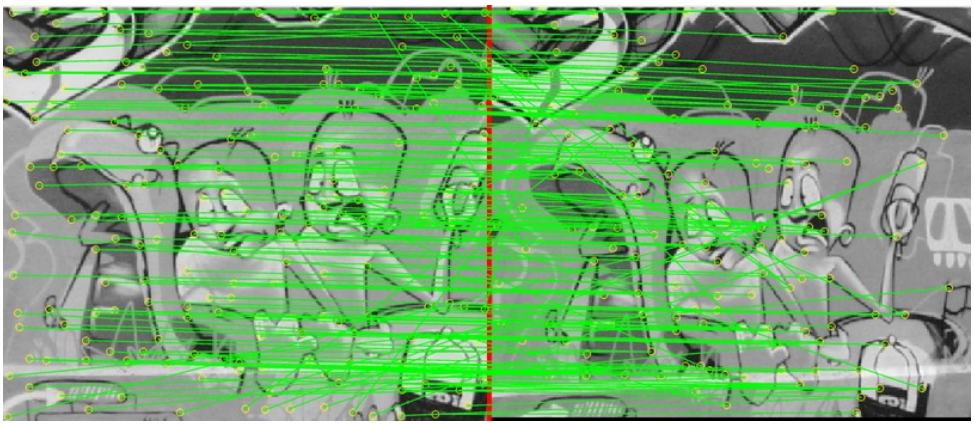
(b) Write a function `find_correspondences` that is given two sets of descriptors of the same type, D_1 and D_2 and finds for each descriptor in D_1 the most similar descriptor in D_2 using the Hellinger distance:

```
function [indices, distances] = find_correspondences(D1, D2)
 ...
```

In the function above, let `indices` be a vector of match mappings from D_1 to D_2, and `distances` be a vector of similarity scores for these matches. For example, if `indices(5) = 7`, this means that the fifth element in D_1 is most similar to the seventh element in D_2 and the actual distance between them is stored in `distances(5)`. To test and debug the implementation it is best to use the same image, e.g. `test_points.jpg`, for both inputs. Detect all feature points in image and store their descriptors in D_1, as well as D_2. This way you will quickly know if the matching function is returning the correct results[2].

Use the function `displaymatches` from the supplementary material to display all the matches of points from the first to the second image. Write a script that loads images `graf/graf1_small.jpg` and `graf/graf2_small.jpg`, runs the function `find_correspondences` and visualizes the result.

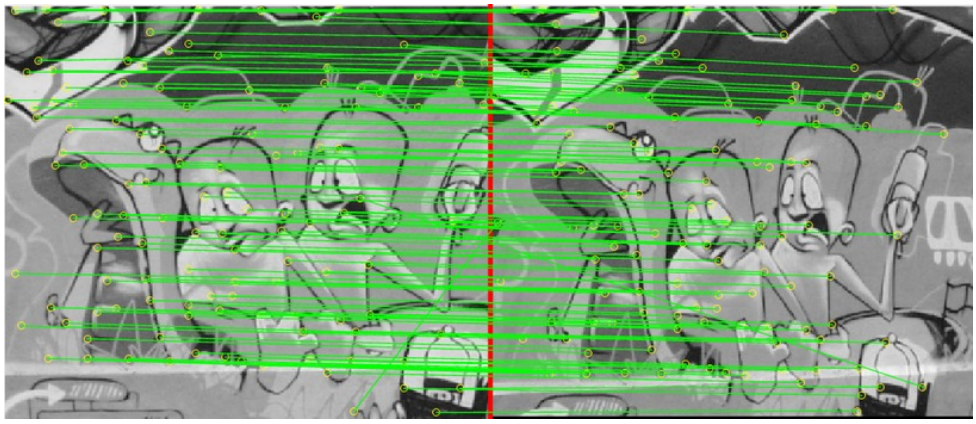
[2]Since all the points will match to themselves this is a good sanity check for your code.



(c) Implement a simple feature point matching algorithm. Write a function `find_matches` that is given two images as an input and returns a matrix M that contains pairs of the matched feature points from image 1 to image 2 and for each pair includes a similarity score. The matrix M should therefore contain five columns: first two for the coordinates of the feature point in the first image, second two for the coordinates of the feature point in the second image, and the last one for the similarity value. Follow the algorithm below for the implementation:

- Use a feature point detector to get stable points for both images (you can experiment with both presented detectors),
- Compute *mag/lap* histograms for all detected feature points,
- Find best matches between histograms in left and right images using the Hellinger distance, i.e. compute the best matches from the left to right image and then the other way around. In a post-processing step only select *symmetric* matches. A symmetric match is a match where a feature point P_1^i in the left image is matched to point P_2^j in the right image and at the same time point P_2^j in the right image is matched to the point P_1^i in left image. This way we get a set of point pairs where each point from the left image is matched to exactly one point in the right image as well as the other way around.
- Construct matrix `M` with the resulting matches.

Use the function `displaymatches` from the supplementary material to display all the symmetric matches (`N = inf`). Write a script that loads images `graf/graf1_small.jpg` and `graf/graf2_small.jpg`, runs the function `find_matches` and visualizes the result. What do you notice when visualizing the correspondences? How robust are the matches? For the initial values of the input parameters set the size of the local regions for descriptor to `m = 41` pixels and the number of histogram bins to `bins = 16`, but you should also try different values.



(d) ★ (5 points) As you can see in the test images, there are several incorrect matches that can be find among the resulting set. Suggest and implement a simple technique that can eliminate at least some of these incorrect matches from the final set of matches. You can use the solutions that were presented at the course lectures, but you can also test your own ideas. It is also best to first solve the basic tasks in the next assignment and then move back to this task if you still have time because it is an open-ended one.

Assignment 3: Homography estimation

As we are dealing with planar images in this exercise, we can try to estimate a homography $\mathbf{H}$ that transforms one image in the space of the other one. In this assignment you will revisit the algorithm that is used to compute such transformations using minimization of mean square error. For additional information about the method, consult the lecture notes as well as the course literature [1] (in literature the term *direct linear transform (DLT)* is frequently used to describe the idea).

You will start with a short revisit about the minimization of mean square error on a simpler case of *similarity transform* estimation. Similarity transform is a linear transform that accounts for translation, rotation, and scale. The transformation of point $\mathbf{x}$ can be written as $\mathbf{x}' = f(\mathbf{x}, \mathbf{p})$, where $\mathbf{p} = [p_1, p_2, p_3, p_4]$ is a vector of four parameters that define the transform such that

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} xp_1 - yp_2 + p_3 \\ xp_2 + yp_1 + p_4 \end{bmatrix}.$$

Question: Looking at the equation above, which parameters account for translation and which for rotation and scale?

Question: Write down a sketch of an algorithm to determine similarity transform from a set of point correspondences $P = [(\mathbf{x}_1, \mathbf{x}'_1), (\mathbf{x}_2, \mathbf{x}'_2), \dots (\mathbf{x}_n, \mathbf{x}'_n)]$. For more details consult the lecture notes.

You will now implement a similar, but more complex algorithm for homography estimation. For a reference point $\mathbf{x}_r$ in the first image we compute a corresponding point $\mathbf{x}_t$

in the second image as:

$$\mathbf{H}\mathbf{x}_r = \mathbf{x}_t \tag{8}$$

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} \; ; \; \text{where} \; \frac{1}{z'_t} \begin{bmatrix} x'_t \\ y'_t \\ z'_t \end{bmatrix} = \begin{bmatrix} \frac{x'_t}{z'_t} \\ \frac{y'_t}{z'_t} \\ 1 \end{bmatrix},$$

where points $\mathbf{x}_r$ and $\mathbf{x}_t$ are written in *homogeneous coordinates*[3]. Using the equations (8) we get a system of linear equations with eight unknowns $h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}$:

$$\frac{h_{11}x_r + h_{12}y_r + h_{13}}{h_{31}x_r + h_{32}y_r + 1} = x_t \tag{9}$$

$$\frac{h_{21}x_r + h_{22}y_r + h_{23}}{h_{31}x_r + h_{32}y_r + 1} = y_t, \tag{10}$$

that can be transformed to

$$h_{11}x_r + h_{12}y_r + h_{13} - x_t h_{31}x_r - x_t h_{32}y_r - x_t = 0 \tag{11}$$

$$h_{21}x_r + h_{22}y_r + h_{23} - y_t h_{31}x_r - y_t h_{32}y_r - y_t = 0. \tag{12}$$

If we want to estimate the eight parameters that determine a homography, we need at least four pairs of matched feature points. As some matches can be imprecise, we can increase the accuracy of our estimate by using a larger number of matches $(\mathbf{x}_1, \mathbf{x}'_1), \ldots, (\mathbf{x}_n, \mathbf{x}'_n)$. This way we get a *over-determined system* of equations:

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{13}$$

$$\begin{bmatrix} x_{r1} & y_{r1} & 1 & 0 & 0 & 0 & -x_{t1}x_{r1} & -x_{t1}y_{r1} & -x_{t1} \\ 0 & 0 & 0 & x_{r1} & y_{r1} & 1 & -y_{t1}x_{r1} & -y_{t1}y_{r1} & -y_{t1} \\ x_{r2} & y_{r2} & 1 & 0 & 0 & 0 & -x_{t2}x_{r2} & -x_{t2}y_{r2} & -x_{t2} \\ 0 & 0 & 0 & x_{r2} & y_{r2} & 1 & -y_{t2}x_{r2} & -y_{t2}y_{r2} & -y_{t2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{rn} & y_{rn} & 1 & 0 & 0 & 0 & -x_{tn}x_{rn} & -x_{tn}y_{rn} & -x_{tn} \\ 0 & 0 & 0 & x_{rn} & y_{rn} & 1 & -y_{tn}x_{rn} & -y_{tn}y_{rn} & -y_{tn} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \tag{14}$$

that can be (similarly to estimation of similarity transform) solved as a minimization of mean square error. If the matrix $\mathbf{A}$ is a square matrix then we get an exact solution of the system. In case of an over-determined system (e.g., $n > 4$) the matrix $\mathbf{A}$ is not square. This problem is usually [1] solved using a matrix pseudo-inverse $\mathbf{A}^T\mathbf{A}$, that is square and can be therefore be split to eigen-vectors and eigen-values. A solution of such system is the unit eigen-vector of $\mathbf{A}^T\mathbf{A}$ that corresponds to the lowest eigen-value (using function `eig`). The same solution can be obtained more efficiently using the singular-value decomposition (SVD) as an eigen-vector that corresponds to the lowest eigen-value of $\mathbf{A}$ (using function `svd`).

$$\mathbf{A} \overset{svd}{=} \mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{U} \begin{bmatrix} \lambda_{11} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{99} \end{bmatrix} \begin{bmatrix} v_{11} & \ldots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \cdots & v_{99} \end{bmatrix}^T \tag{15}$$

[3]Homogeneous coordinates for a point in 2D space are obtained by adding a third coordinate and setting it to 1

A vector $\mathbf{h}$ that contains the parameters of the homography matrix is obtained from the last column of matrix $\mathbf{V}$ and by normalizing the vector with the value of v_{99} to set the last element in h to 1:

$$\mathbf{h} = \frac{[v_{19}, \ldots, v_{99}]^T}{v_{99}}. \tag{16}$$

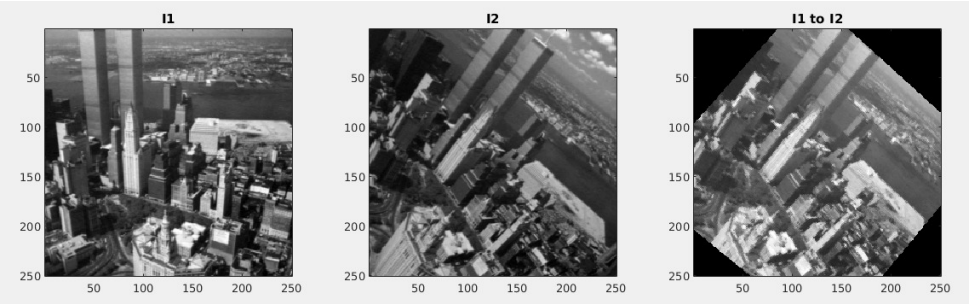
(a) Write function `estimate_homography`, that approximates a homography between two images using a given set of matched feature points following the algorithm below.

- Construct a matrix $\mathbf{A}$ using the equation (13).
- Perform a matrix decomposition using the SVD algorithm: `[U, S, V] = svd(A)`.
- Compute vector $\mathbf{h}$ using equation (16).
- Reorder the elements of $\mathbf{h}$ to a 3×3 matrix $\mathbf{H}$ (e.g. using the function `reshape`).

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

(b) Load the two New York city-scape images (`newyork/image1.jpg` and `newyork/image2.jpg`), and four hand-annotated correspondence pairs in the file `newyork.txt`[4]. Use the function `displaymatches` to display the pairs of points. Using these pairs estimate the homography matrix H from the first image to the second image and use function `transform_homography` to transform the first image to the plane of the second image using the given homography, then display the result. Also test your algorithm with images `graf/graf1.jpg`, `graf/graf2.jpg` and points from `graf.txt`.

(c) When you get the correct result for the given points, repeat the procedure by generating a match set automatically using the `find_matches` function that you have implemented in the previous assignment. Select the first 10 best matches and use them to compute a homography. Visualize the result by transforming the first image to the space of the second image as well as displaying the matches between the images. How well have you managed to estimate the homography? Does the estimate improve if you use more/less image pairs? What happens if you include imperfect matches in the set? Set the parameters (feature point detector, parameters, number of selected points, etc.) to get the best performance of the method.



[4]Use `load` function to read the points. The first two columns contain x and y coordinates of the points in the first image (first line contains x, second y) and the second two columns contain the corresponding points for the second image.

(d) ★ (10 points) Implement and test a more robust homography estimation algorithm using iterative reweighed least squares approach that is described in the notes published on the e-classroom. Test the robustness of the algorithm on the image pairs from the previous task. Experiment with the number of correspondences used as an input to the estimation.

(e) ★ (10 points) In the directory `panorama` you can find an ordered series of images that should be combined into a panorama[5]. As the sequence is already sorted, your task is to align and merge consecutive images to form a panorama. To combine two images into one using a given homography, you can use function `merge_homography` that you can find in the supplementary material. If you are not able to create the entire panorama, try to create at least its subset. A sequence of at least three images (it does not matter where you start or in which direction you process the images) has to be merged in order to complete this task partially (max. 5 points), however, the highest number of points will only be given for a complete panorama where you will not use separate fine-tuned parameters for merging every individual pair of images (i.e. a generally applicable solution).

(f) ★ (10 points) In the directory `puzzle` you can find a series of images that together represent a larger image. Develop an algorithm that determines which images are neighbors in the puzzle and connects them together using the concepts that you have acquired during this exercise. You can assume that each piece is only connected to at most four other pieces, with each of them it shares a 50% overlap, that each neighbor is only shifted to one side of the piece (up, down, left, right), and that the transformations are only translative.

References

[1] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2002.

[5]Note that these images were extracted from an existing panorama, so the process of re-merging them is easier than doing it with raw images. In that case some extra steps are needed that are not the subject of this exercise.