Assignment 4: Feature points, matching, homography

Machine perception

2021/2022

Create a folder `assignment4` that you will use during this assignment. Unpack the content of the `assignment4.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as Python scripts to `assignment4` 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 maximum total amount of points for each assignment is 100. Each optional task has the amount of additional points written next to it.

Introduction

This assignment will deal with automatic searching for correspondence points between two images. Correspondences are a key step when dealing with the task of aligning two or more images, for example building a panorama from multiple images. Methods that find correspondences between images usually start by finding feature points in them. Feature 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.

Exercise 1: Feature points detectors

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

(a) The 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:

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

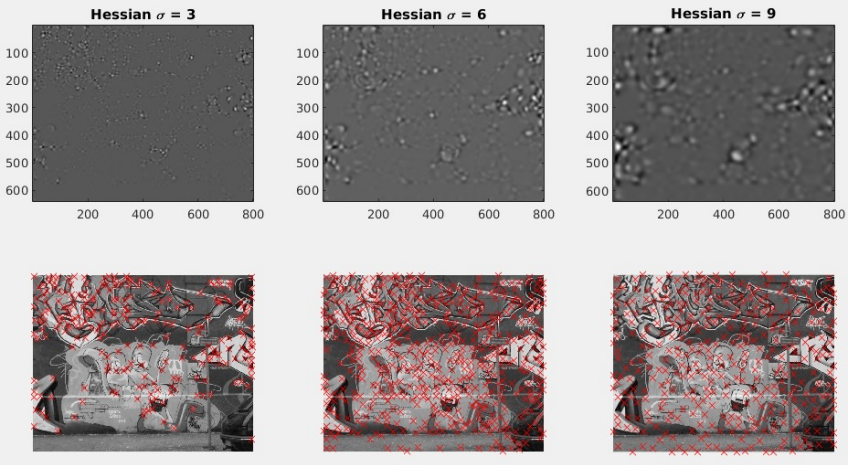
where σ explicitly states that the second derivative is computed on a *smoothed image*. Hessian detector selects point (x, y) as a feature point if the determinant of the Hessian matrix exceeds a given threshold value t. If we want to create a variant of the Hessian detector that is scale independent (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:

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

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.

Extend the function `hessian_points` by implementing a non-maximum suppression post-processing step that only retains responses that are higher than all the neighborhood responses and whose value are higher than a given threshold value `thresh`. The function should return the list of points that satisfy these properties.

Create a function that you will use plot the detected points (use `plt.plot()`) over the input image. Load the 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 a feeling for the effect of the parameters.



Question: What kind of structures in the image are detected by the algorithm? How does the parameter σ affect the result?

(b) Implement the Harris feature point detector. This detector is based on the 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 chooses a point (x, y) for a feature point if both eigenvalues of the auto-correlation matrix for that point are large. 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:

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

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

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

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

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

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

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

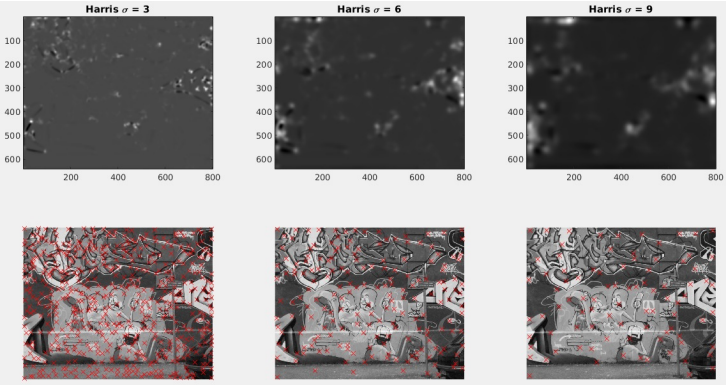
we can express the feature point condition for (x, y) as:

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

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 the feature point detector as function `harris_points` that computes values of equation (7) for all pixels, performs the non-maximum suppression post-processing step as well as thresholding using threshold `thresh`. The function should return the list of points that satisfy these properties.

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



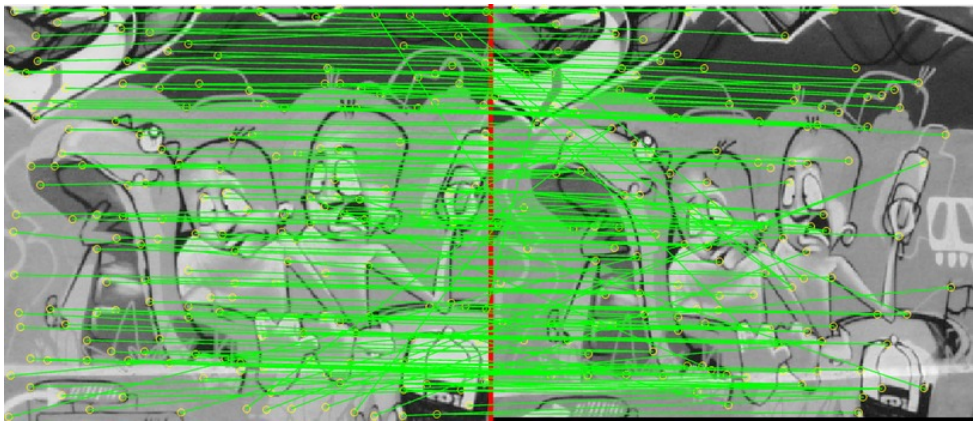
Exercise 2: Matching local regions

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

(a) Use the function `simple_descriptors` to calculate descriptors for a list of feature points. Then, write a function `find_correspondences` that, given two lists of descriptors, calculates the similarities between all descriptors in both lists. Use the Hellinger distance. Finally, for each descriptor from the first list, find the most similar descriptor from the second list. A good idea might be to use the same list (perhaps shuffled) for both inputs to sanity check your code.

Note: Use your own functions `gauss` and `gaussdx` from previous assignments in order for the descriptor function to work correctly.

(b) Write a script that loads images `graf/graf1_small.jpg` and `graf/graf2_small.jpg`, runs the function `find_correspondences` and visualizes the result. Use the function `display_matches` from the supplementary material for visualization. Experiment with different parameters for descriptor calculation and report on the changes that occur.



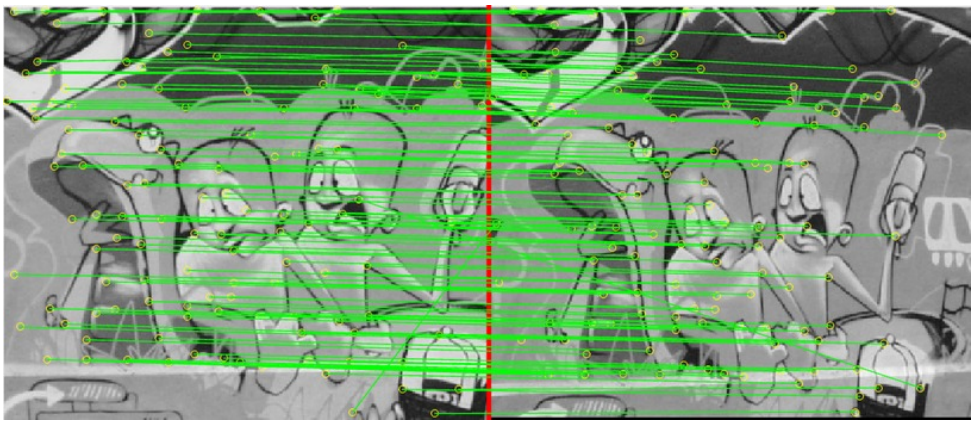
(c) Implement a simple feature point matching algorithm. Write a function `find_matches` that is given two images as an input and returns a list of matched feature points from image 1 to image 2. The function should return a list of index pairs, where the first element is the index for feature points from the first image and the second element is the index for feature points from the second image.

Follow the algorithm below:

- Execute a feature point detector to get stable points for both images (you can experiment with both presented detectors),
- Compute simple descriptors for all detected feature points
- Find best matches between descriptors 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.

Use the function `display_matches` from the supplementary material to display all the symmetric matches. Write a script that loads images `graf/graf1_small.jpg` and `graf/graf2_small.jpg`, runs the function `find_matches` and visualizes the result.

Question: What do you notice when visualizing the correspondences? How robust are the matches?



(d) ★ (5 points) Incorrect matches can occur when matching descriptors. Suggest and implement a simple method for eliminating at least some of these incorrect matches. You can use the ideas that were presented at the lectures or test your own ideas. Either way, you need to explain the idea behind the method as well as well as demonstrate that the number of incorrect matches is lowered when using the proposed method.

(e) ★ (25 points) Implement a local feature detector of your choice (e.g. SIFT, SURF, BRIEF, HOG). Test it on other assignments and report on the changes (increased robustness etc.).

(f) ★ (10 points) Record a video with your phone or webcam. Use OpenCV to detect keypoints of your choice, display them using `cv2.drawKeypoints` and save the video with the displayed keypoints. The video must demonstrate the keypoint detector's robustness to rotation and scale change. Make the video at least 15s long.

Exercise 3: Homography estimation

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

We will start with a short overview of the minimization of mean square error on a simpler case of *similarity transform* estimation. The similarity transform is a linear transform that accounts for translation, rotation, and scale. The transformation of a 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:

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

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

where points $\mathbf{x}_r$ and $\mathbf{x}_t$ are written in *homogeneous coordinates*[2]. 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 an *overdetermined 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 eigenvectors and eigenvalues. A solution of such a system is the unit eigenvector of $\mathbf{A}^T\mathbf{A}$ that corresponds to the lowest eigenvalue (using function `eig`). The same solution can be obtained more efficiently using the singular-value

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

decomposition (SVD) as an eigenvector that corresponds to the lowest eigenvalue of $\mathbf{A}$ (using function `svd`).

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

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:

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

(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`).

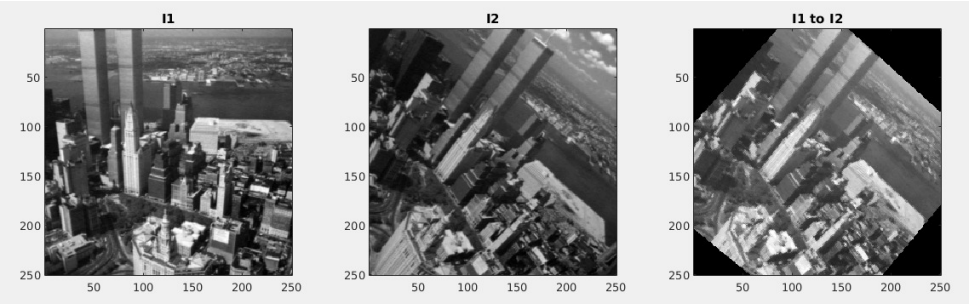
(b) Load the two New York cityscape images (`newyork/image1.jpg` and `newyork/image2.jpg`), and four hand-annotated correspondence pairs in the file `newyork.txt`[3]. Use the function `display_matches` to display the pairs of points. Using these pairs estimate the homography matrix H from the first image to the second image and use function `cv2.warpPerspective` 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`.

Note: You can check the correctness of your implementation by comparing with the file `newyork/H.txt`

(c) When you get the correct result for the given points, repeat the procedure by generating a set of matching points 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 matching 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.

Note: This does not need to produce perfect results every time because DLT is not robust to outliers.

[3]Use `read_data` function to read the points. Reshape the data into four columns. The first two columns contain x and y coordinates of the points in the first image and the second two columns contain the corresponding points for the second image.



(d) ★ (15 points) Implement and test a more robust homography estimation algorithm using iterative reweighed least squares approach that you have heard about at lectures. Test the robustness of the algorithm on the image pairs from the previous task. Experiment with different numbers of correspondences used as an input to the estimation.

(e) ★ (10 points) Write your own function for mapping points using the homography matrix. It should work like OpenCV's `warpPerspective()`. It should accept an image and a homography matrix and return the input image as remapped by the homography matrix. The size of the output should match the size of the input image.

Hint: You will need to use homogeneous coordinates.

References

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