Image Processing - Exercise Sheet 2

Teaching assistants: Binbin Xiang, Shengyu Huang, Rodrigo Caye Daudt
Main contact: shenhuan@ethz.ch
Date: April 2023

In the second exercise sheet of the lecture on Image Processing, it is required to implement a set of image processing tasks. Please submit both your code and a brief report that includes the results for each subtask, along with a concise comment, if deemed necessary. The comment for each subtask should be limited to no more than four sentences.
This exercise sheet contributes 80% to the overall grade for the course. Please submit your solutions to the assignment section no later than May 25, 2023. Each student will discuss his/her answers individually with the teaching assistants. These discussions will take place on June 1st, 2023 between 15:45 and 17:30.
The prepared code base can be found here: https://github.com/prs-eth/Bildverarbeitung. **It is not allowed to use other libraries than those already imported there!**

1 Fourier transform

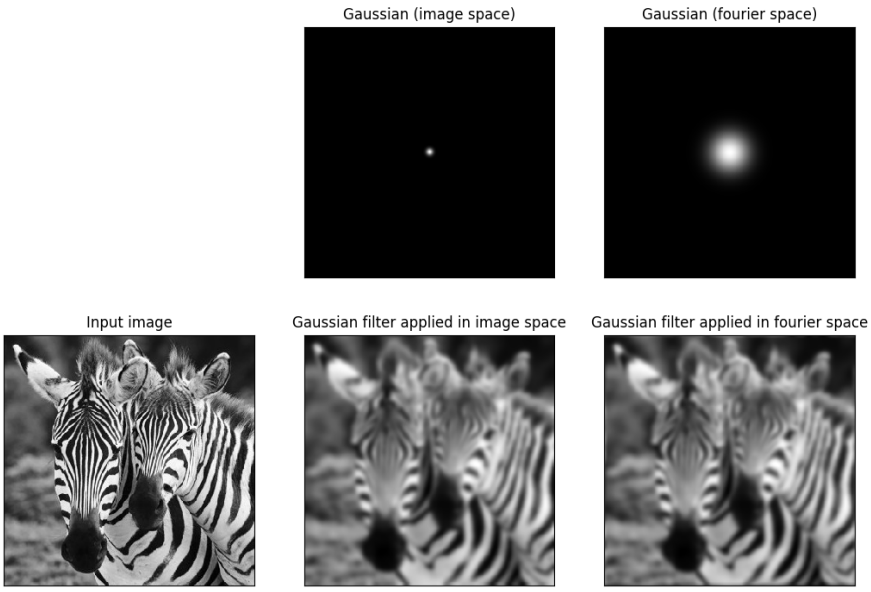
In the following task, you will work with the *Fast Fourier Transform (FFT)*, an efficient algorithm for computing the *Discrete Fourier Transform (DFT)*. Please fill in *fourier.ipynb*.

1.1 Transformation

Please load the input image *tuebingen.jpg*, transform it with the FFT (*scipy.fft.fft2*) and plot the spectrum. In this and all following plots, low frequencies should be plotted in the center of the image and high frequencies at the edge of the image. Please use *scipy.fft.fftshift* and *scipy.fft.ifftshift* to do this. Notes: (i) plot only the magnitude of the resulting spectrum, not the phase. (ii) logarithmic scaling of magnitudes could improve the contrast of the visualization.

1.2 High-pass-Filter

Now please filter the spectrum so that low frequencies are suppressed. Please implement a function *highpass_filter* that returns the ideal high-pass filter (hard



(a) Bild 1

Figure 1: Fourier transform: Example result

circular aperture, radius 30 pixels), then apply the filter in frequency space, and transform the filtered spectrum back into spatial space and plot the resulting image as well as the filtered spectrum.

1.3 Gaussian filter

The goal of this sub-task is to apply a Gaussian low-pass filter in both spatial and frequency space. You should get identical results except for numerical inaccuracies.

1.3.1 Construct a Gaussian filter

Please define the function *g_core* that returns a Gaussian kernel. To do this, please use the density function *scipy.stats.norm.cdf* to calculate the correct filter weight for each interval (pixel).

1.3.2 Convolution with the Gaussian filter

Construct a Gaussian filter with standard deviation 5. First, plaese set the filter size to the image size. Since the values far from the filter center hardly differ from 0, you can cut out a smaller window (about 6× standard deviation, odd

side length in pixels) for the convolution and use this for the convolution with the input image. To do this, please use *scipy.ndimage.convolve*.

1.3.3 Transformation of the filter into the frequency space

Transform the filter into the frequency space. As described in task 1.1, we want to have the low frequencies in the center of the image, for this purpose please rearrange both the frequency representation of the input image and the frequency representation of the Gaussian filter using *scipy.fft.fftshift*.

1.3.4 Convolution with Gaussian filter in frequency domain

Multiply the Fourier transform of the image by the transformed Gaussian filter (magnitude only), then transform the result back into the spatial space.

1.3.5 Visualisation

Please visualise:

- The Gaussian filter in spatial space
- The Gaussian filter in frequency space
- The output image after convolution with the Gaussian filter in spatial space
- The output image after multiplication with the Gaussian filter in frequency space

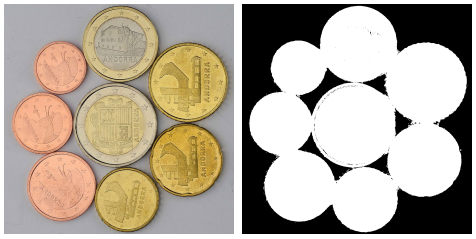
Figure 1 shows an example (different image, different filter width) of how your results should be displayed. In addition, please calculate and visualize the deviation between the two results.

1.3.6 Separable filtering

You used a 2D Gaussian filter in subtask 1.3.1. Now repeat this subtask, but filter separately in x and $y-$ directions, and compare the resulting images and the computation time. Please note that to get a reliable measurement of the computation time it is necessary to compute the filtering for a large image or repeat several times.

2 Morphology

In this task, the graphical user interface (GUI) is used, and you will implement the functions in *libs/morphology.py*.



(a) Coins (b) Segmentation

Figure 2: Coins task

2.1 Coins

We provide an input image *coins.jpg* and a simple approximate segmentation *coins_seg.png*.

2.1.1 Segmentation

Please segment the input coin image *coins.jpg* into coins and background. How to solve this task is up to you. Please experiment with different segmentation/clustering methods and color spaces, and try to achieve at least the quality of the mask as is shown in *coins_seg.png*.

2.1.2 Close the holes inside the coins

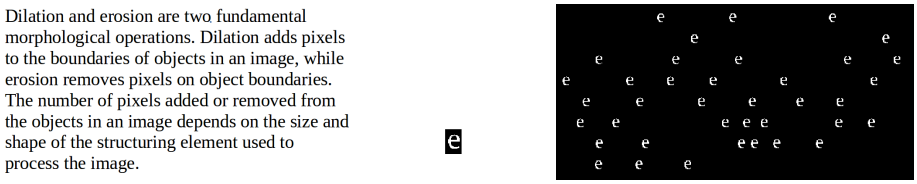
Please start with the provided segmentation image *coins_seg.png* and use *cv2.dilate* and *cv2.erode* to close the holes inside each coin, and visualize the results.

2.1.3 Coin instance segmentation

Please start with the result of the last subtask and extend it so that the coins are segmented individually. Note: You can use marker-based segmentation using the watershed algorithm. First, use morphological operations and the function *cv.connected Components* to find connected components as markers for the watershed algorithm. It is sufficient to find one component per coin, the shape does not have to exactly match the shape of the coin. In a second step, read the initial image *coins.jpg* and apply the watershed algorithm (*cv.watershed*). In the final step, generate random colors and colorize the individual coins. The background should also be colored and stand out from the coins.

2.2 Letter recognition

In this task, the input consists of a rasterized text (Fig. 3a) and a template for the letter "m" (Fig. 3b). Please use morphological operations (dilation, erosion, others if necessary) to extract all letters "m" from the text and create a result image like the one shown in Fig. 3c.



(a) Input-Text (b) Letter (c) Result

Figure 3: Letter recognition

3 Corner detector

Please implement the Harris and Shi-Tomasi corner detectors for black and white images. To do this, follow the instructions in *corner.ipynb* and fill in the *TODO* part. Please visualize all intermediate results using the provided script. The desired intermediate results are for each of the two methods:

- Harris / Shi-Tomasi response per pixel
- pixels above the selected corner threshold.
- corners after non-maximum suppression

Please also apply the same detector to rotated versions of the image, and compare the results in the report.

4 Canny edge detection

Please implement an edge pixel detector for gray scale images. For this, follow the instructions in *canny_edge.ipynb* and fill in the *TODO* part. Please visualize intermediate results using the provided script which should cover:

- input
- image after application of Gaussian filter
- magnitudes and directions of gradients
- Canny edge pixels after non-maximum suppression

Please comment on similarities and differences to corner detection in the report.

5 Bonus question

Integrate one of the three tasks from the Jupyter files into the graphical user interface (GUI). You can customize it according to your needs.