VU 183.585

Computer Vision

Exercise Course: Assignments I

WS 2017/18

Sebastian Zambanini

Computer Vision Lab
Institute of Computer Aided Automation
http://www.caa.tuwien.ac.at/cvl/teaching/wintersemester/cv/index.html

zamba@caa.tuwien.ac.at

Assignment 1: Colorizing Images (4 Points)

The goal of this assignment is to learn to work with images in MATLAB by taking color channel images and assemble them to RGB color images. For this purpose, we use digitized glass plate images from the Prokudin-Gorskii collection (see Figure 1 for an example).

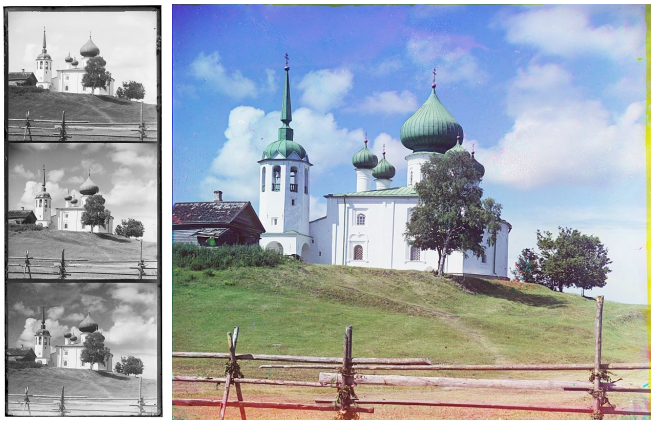

Figure 1: Glass plate scans of photographs with blue, green and red color filter (left) and colorized digital image (right).

Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer who, between the years 1909-1915, traveled the Russian empire and took thousands of photos of everything he saw. He used an early color technology that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. Back then, there was no way to print such photos, and they had to be displayed using a special projector. Prokudin-Gorskii left Russia in 1918. His glass plate negatives survived and were purchased by the Library of Congress in 1948. Today, a digitized version of the Prokudin-Gorskii collection is available online.

Detailed Instructions

There are six sets of glass plate images available on TUWEL, where the R, G and B channels are labelled by the suffixes `'_R'`, `'_G'` and `'_B'`, respectively. Write a program that reads in the three channel images and returns a RGB color image. The three channels are not perfectly aligned, thus your program has to align them automatically. In order to do so, exhaustively search over a window of possible displacements (say [-15,15] pixels), score each one using some image matching metric, and take the displacement with the best score. Although the images do not actually have the same brightness values (they are different color channels), the normalized cross-correlation (NCC) is a sufficient choice for the image matching metric. The NCC between two images I_1 and I_2 is defined as

$$NCC(I_1, I_2) = \frac{\sum_{x,y}(I_1(x,y) - \overline{I}_1) \cdot (I_2(x,y) - \overline{I}_2)}{\sqrt{\sum_{x,y}(I_1(x,y) - \overline{I}_1)^2 \cdot \sum_{x,y}(I_2(x,y) - \overline{I}_2)^2}} \quad (1)$$

where $\overline{I}_1$ and $\overline{I}_2$ are the mean values of the images I_1 and I_2, respectively.
Hint: you might find the functions `circshift` and `corr2` helpful for your implementation.

Assignment 2: Image Segmentation by K-means Clustering (8 Points)

Image segmentation is the process of dividing an image into disjoint regions. Depending on the given application area, these regions describe either semantic objects (e.g. a person) or regions with similar appearance (e.g. color). A basic image segmentation method in computer vision is based on K-means clustering, a technique for clustering data points in a D-dimensional space. The parameter K defines the number of clusters and has to be defined beforehand.
The goal of this assignment is to apply K-means to images in order to segment regions having similar color values. Generally, for this purpose the D-dimensional data points $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ can be defined in two ways:

- **Color information only**: the data points are generated from the RGB-values of the N pixels of the image (i.e. $D = 3$).
- **Color and spatial information**: the data points are generated from the RGB-values and the pixel coordinates of the image (i.e. $D = 5$). This way, the spatial coherence of pixels is additionally considered for the segmentation.

In K-means clustering, the cluster centroids $\boldsymbol{\mu}_k$ ($k = 1, ..., K$) are computed as the mean of all data points with shortest distance to that centroid. To describe the assignment of data points to clusters, a binary indicator matrix $r(n, k) \in \{0, 1\}$ is used: if data point $\mathbf{x}_n$ is assigned to cluster k then $r(n, k) = 1$, and $r(n, j) = 0$ for $j \neq k$. We can then define an objective function (also called *distortion measure*), given by

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r(n, k) \left\| \mathbf{x}_n - \boldsymbol{\mu}_k \right\|^2 \tag{2}$$

which represents the sum of the squares of the distances of each data point to its assigned cluster centroid $\boldsymbol{\mu}_k$. Our goal is to find values for $r(n, k)$ and $\boldsymbol{\mu}_k$ so as to minimize J. The iterative algorithm for K-means clustering works as follows:

1. Choose random starting values for the centroids $\boldsymbol{\mu}_k$.
2. Assign all data points to their nearest cluster centroids

$$r(n, k) = \begin{cases} 1 & \text{if } k = \arg\min_j \left\| \mathbf{x}_n - \boldsymbol{\mu}_j \right\|^2 \\ 0 & \text{else} \end{cases} \tag{3}$$

3. Compute the new cluster centroids as the mean of all data points assigned to that cluster:

$$\boldsymbol{\mu}_k = \frac{\sum_n r(n, k) \mathbf{x}_n}{\sum_n r(n, k)} \tag{4}$$

4. Compute J and check for convergence. For this purpose, compute the ratio between the old and the new J. If the ratio lies under a given threshold, terminate the clustering, otherwise go to point 2.

Implement K-means clustering in Matlab using the the description given above (note: you are not allowed to use the Matlab function `kmeans` for this task). Apply your code to the images `simple.png`, `future.jpg` and `mm.jpg`. Visualize the results of your segmentations by coloring all pixels of a cluster with their mean color values. Where do you see - based on your results - the strengths and the weaknesses of the method?

Note: usually, color and spatial information are not equally scaled. You should thus normalize the data points to the range $[0, 1]$ before starting the clustering.

Issues to be addressed in the report:

- Show the results for all images in the case of 3D data points as well as 5D data points (using a fixed value of K). Discuss the results. Which data representation is better in your opinion?
- Apply different values of K to the image `mm.jpg` and show the results for both 3D and 5D data points. Interpret the results.
- Where do you see - based on your results - the strengths and the weaknesses of the method?

Assignment 3: Scale-Invariant Blob Detection (7 Points)

Interest point detectors are commonly used in computer vision in order to detect salient image regions which can then be described and, for instance, matched between two views of the same scene (see Section 4.1 in the textbook [Sze10]). However, objects may have have different sizes in two images, which means that local keypoints on the objects have to be detected and described in a scale-invariant way in order to be matched correctly. In this assignment we examine the *scale-normalized Laplacian of Gaussian* operator. This operator has shown to be a reasonable choice to discover an intrinsic scale to blobs in an image [Lin94, Mik02].

Theory

The Laplacian blob detector to be implemented for this assignment has to perform the following tasks:

- detect the spatial locations of interest points (spatial selection)
- assign a scale to the detected interest points (scale selection)

Spatial Selection: the Laplacian is a second order differential operator. The magnitude of the Laplacian response will achieve a maximum at the center of a blob (circle), provided the scale of the Laplacian is "matched" to the scale of the blob. This is illustrated in Fig. 2 for the 1D case.

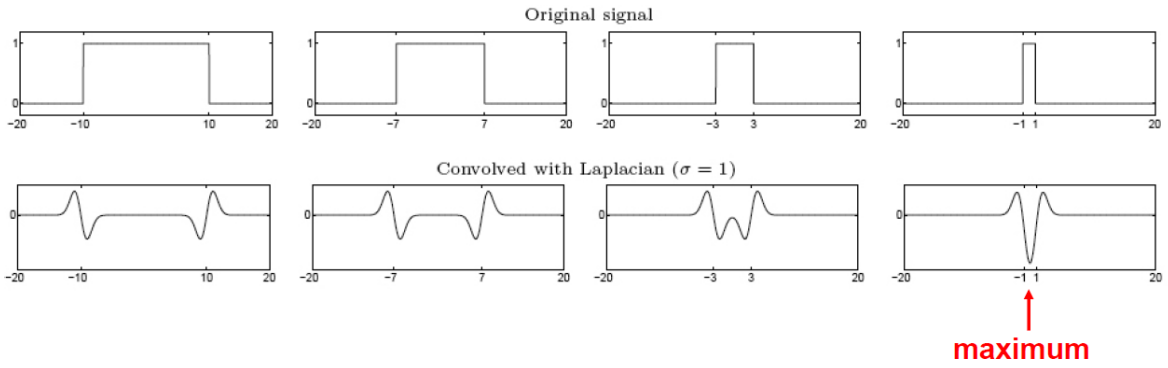

Figure 2: Spatial blob selection using the Laplacian.

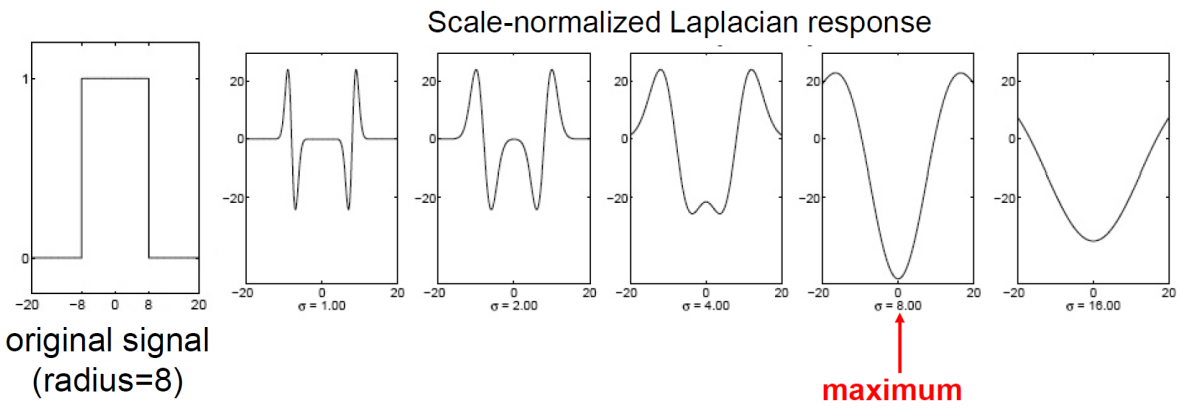

Figure 3: Scale selection using Laplacians.

Scale Selection: the characteristic scale of a blob can be detected by convolving it with Laplacians at several scales and looking for the maximum absolute response. This is illustrated in Fig. 3.

These two properties of the Laplacian operator show that it can be used for blob detection with scale selection. However, for increased robustness, the second derivative needs to be insensitive to noise. Therefore, the image has to be smoothed first before computing the second derivatives. These two operations can be merged into one step by the use of the Laplacian of Gaussian (LoG) filter, which is the Laplacian applied to a Gaussian filter. An example is shown in Fig. 4.
Based on this theory, the scale-invariant blob detector works as follows:

1. Start with an initial scale σ_0 and repeatedly increase the scale by a constant factor k

   (a) Build a scale-normalized Laplacian of Gaussian filter with current scale σ.

   (b) Convolve the image with the filter and save the absolute response of LoG for the current level of scale space.

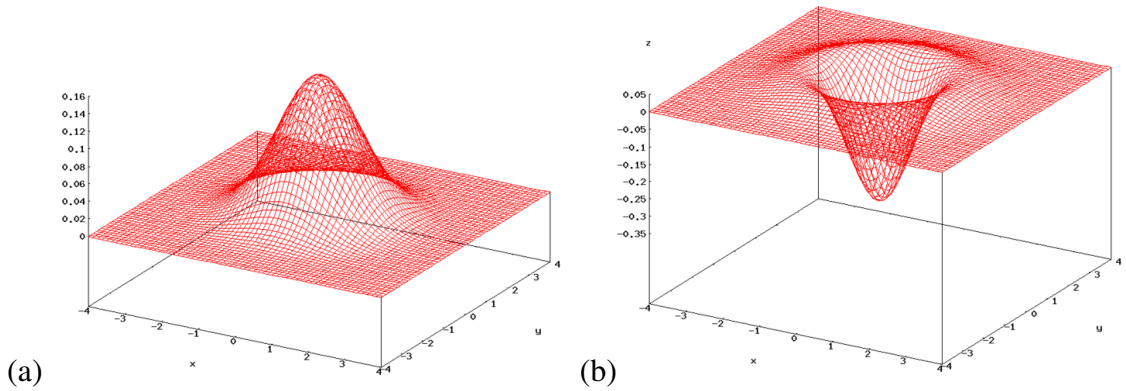

Figure 4: (a) Gaussian filter kernel and (b) corresponding Laplacian of Gaussian.

2. Perform non-maximum suppression in the 3D scale space: a point is detected at position (x, y) with scale σ_i if the point is above a certain fixed threshold and a local maximum when compared to its 8 neighboring points at scale σ_i as well as to its 9 neighboring points at scales σ_{i-1} and σ_{i+1}, respectively.

Your task is to implement a LoG blob detector and use the images `butterfly.jpg` and an own image for testing.

Details and Hints

- **Scale space creation:** It is useful to use a three-dimensional matrix to represent the scale space. This matrix can be initialized as follows:

```
% [height,width] - dimensions of image
% levels - number of levels in the scale space
scale_space = zeros(height,width,levels);
```

- **Filter creation:** use the `fspecial` function with parameter `'log'` to create the LoG filters. Since the response of LoG decreases as σ increases, you have to scale-normalize the filter by multiplying it with σ^2. In order to decrease computation time for smaller σ, you should set the filter size proportional to σ. A good choice is to restrict the filter size to the range $[-3\sigma, 3\sigma]$, i.e. the filter size is computed as $2\lfloor 3\sigma \rfloor + 1$.

- **Convolution**: for scale space creation, in the convolution step you should avoid (a) different dimensions of the output images due to different filter sizes and (b) artefacts at the image borders, i.e. high responses caused by unreasonable assumptions for values outside the image. The best way to achieve this is to use the `imfilter` function with parameters `'same'` and `'replicate'` for convolution. You can read up the effects of these parameters in the function reference.

- **Parameter selection:** You have to choose the initial scale σ_0, the factor k by which the scale is multiplied each time and the number of levels in the scale space. A reasonable choice is for instance $\sigma_0 = 2$, $k = 1.25$ and 10 levels.

- **Non-maximum suppression:** The LoG filter produces positive as well as negative responses and thus for extrema detection one has to search for local minima in the negative domain as well as for local maxima in the positive domain. However, as local minima and maxima are treated likewise, it is more convenient to take the absolute responses and search for local maxima only.

- **Visualization of results:** You can use the function `show_all_circles` to visualize the detected blobs. The function takes the circle centers and corresponding radii and draws the circles in an image. However, to choose the correct radii, the relationship between the detected scale and the radius of the circle that most closely "approximates" the circle is needed. In other words, at what scale does the Laplacian achieve a maximum response to a binary circle of radius r? As illustrated in Fig. 5, for the maximum response the zeros of the LoG have to be aligned with the circle. As the LoG is given by $(x^2 + y^2 - 2\sigma^2)e^{-(x^2+y^2)/2\sigma^2}$, the maximum response occurs at $\sigma = r/\sqrt{2}$ (the proof is left as an exercise for interested students). As a consequence, you have to multiply the detected scale by $\sqrt{2}$ to get the correct radii of the circles to be drawn.

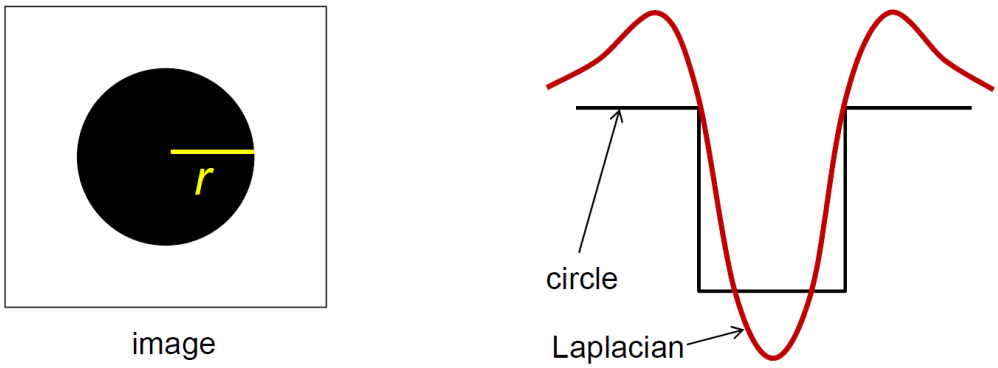

Figure 5: Circle with radius r and LoG with highest response to the circle.

Issues to be addressed in the report:

- Apply the method to both the original images as well as to half-sized versions of them. Draw the detected blobs as circles with appropriate scale. Is the method able to find blobs in a scale-invariant way? If there are errors, what are the reasons for them?

- Pick a detected keypoint and plot the response of the LoG for all scales in both image versions. The outcome should be a 2D plot where the x-axis represents the scale of the filter and the y-axis the filter response at the selected keypoint position. Describe and explain the difference between the two curves.

References

[Lin94] Tony Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2):224–270, 1994.

[Mik02] Krystian Mikolajczyk. *Detection of local features invariant to affine transformations*. PhD thesis, INPG, Grenoble, 2002.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.