BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Prof. Dr.-Ing. Kristian Hildebrand

Learning from Images
Filter, OpenCV Basics and K-Means (10 Points)

WiSe 2019/20

The goal of this exercise is a deeper understanding of OpenCV and image processing. Download the default skeleton source code from Moodle and implement the necessary functions. Extend and modify the source code as you wish. That's just an initial support.

Note: Make yourself comfortable with OpenCV: e.g. `http://docs.opencv.org/3.1.0/d6/d00/tutorial_py_root.html`. Data types and formats are very important when using OpenCV as images are represented as numpy matrices. OpenCV images are often loaded, edited and saved in *uint8: 0-255*. Many calculations will take place in *float* format. Conversions can be done with NumPy *np.float32(image)*.

Exercise 1: Computer Vision Basics + OpenCV (6 Points)

Exercise 1.1: Loading images (1 Point)

Load the image *Lenna.png* using OpenCV and display it side by side as both a grayscale image and a color image. The result should look like Figure 1. Note: *np.concatenate* can combine multiple matrices / images. You can also use this here by creating a new image with double the width. The final image must then have 3 color channels. That means you have to copy the grayscale image to the R / G / B channels. You can get the image size via *rows, cols = img.shape[: 2]*.

Figure 1: Loading an image

Exercise 1.2: OpenCV experiments (2 Point)

Experiment with the following - as in the lecture discussed - image processing algorithms in OpenCV:

a) Change of color spaces (HSV, LAB, YUV)

b) Adaptives thresholding in the variants Gaussian and Otsu-Thresholding.

c) Canny edge extraction

Please use the file *01_opencv_experiments.py* for your implementation. Implement a change of functionality on key press.

Exercise 1.3: SIFT in OpenCV (1 Point)

Please use with the SIFT descriptor in OpenCV to implement a video streaming example showing SIFT features and visualize its keypoints as illustrated in Figure 2 (http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html). Please use the file 01_features.py for your implementation.

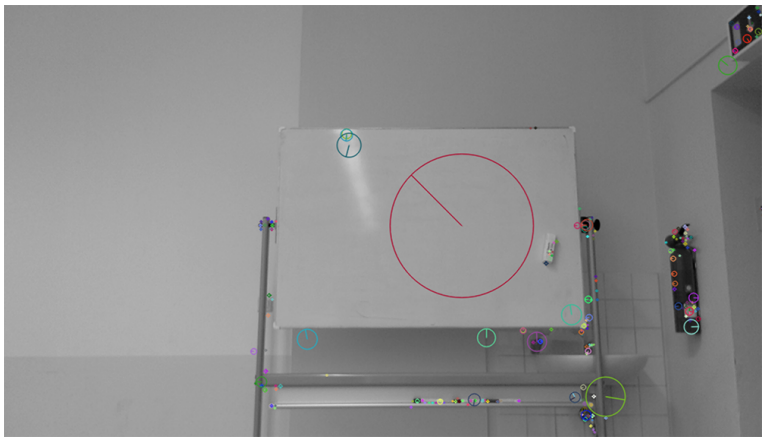

Figure 2: Image of a video stream showing the visualization of SIFT features.

Exercise 1.4: Convolution (2 Points)

Implement the convolution of an image with a filter mask on a grayscale image **without the functions available in OpenCV or scipy**. Use a Gaussian blur as filter mask and a Sobel filter (x- and y-direction) matrix as discussed in the lecture to calculate the image gradients. Generate a picture that represents the gradient strengths (Magnitude of Gradients) using your implementation. **Note:** Try to make the implementation as efficient as possible. The implementation of OpenCV then runs in real time - this is not necessary. Please use the file *01_filter.py* for your implementation.

Exercise 2: K-Means for color quantization (4 Punkte)

Machine learning methods play an important role in many computer vision applications. Throughout the course we will be discussing some recent algorithms. However many *classical* problems such as feature matching, image segmentation or color quantization (and many many more) rely heavily on well understand ML methods. One example the k-means algorithm. In this exercise you'll have to implement a simple color quantization (clustering) scheme based on k-means. In doing so, a pre-known number of k groups is formed from a set of similar data points. This standard method is one of the most commonly used techniques for grouping data points, e.g. RGB pixel values in our case (see illustration **??** for more details. **Please read the image caption for a better understanding of the representation.**).

The idea of *Color-based Segmentation* based on the k-means algorithm is to quantize / segment the colors in the image to group similar colors in the image into one group. The similarity between pixel values can be determined by the Euclidean distance between RGB, LAB or HSV values.

Based on the given source code and the comments, implement the k-means algorithm and test it for different k and different color spaces as shown in Figure 3. Use either the default colors (*cluster_colors*) or the colors of the cluster center (mean color). **Calculate the total error for each result and print it on the commandline.**

Please interpret your results of the clustering and answer the following questions **(please submit your answer as an additional .txt or .md file)**:

a) What are the problems of this clustering algorithm?

b) How can I improve the results?

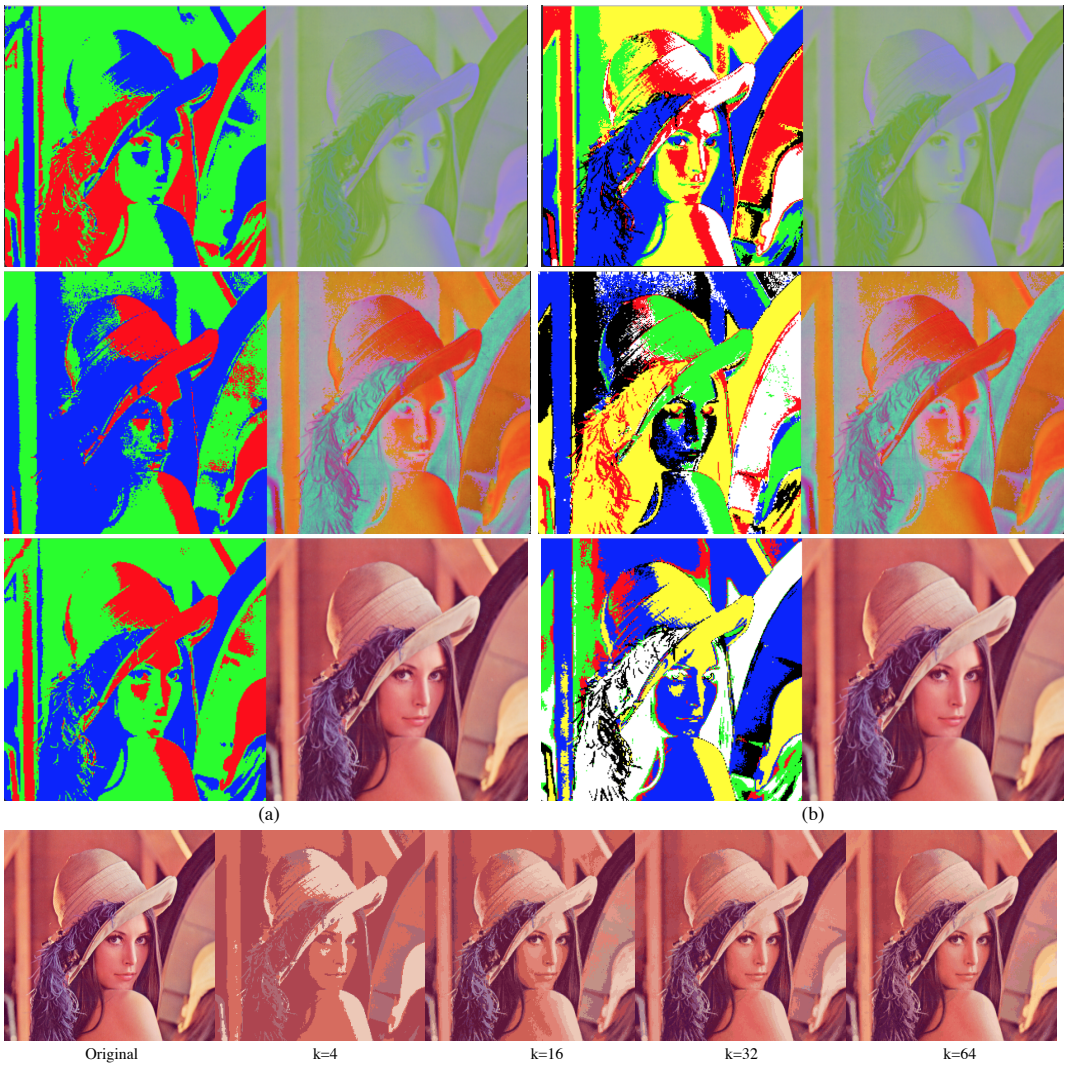

Figure 3: k-means based color clustering. Top: (a) k-means with $k = 3$ over the input image in the LAB, HSV, and RGB color spaces. **On the left, you see the respective color clusters displayed in red, green, blue, white, or black to make it easier to see which pixels belong to which cluster** (b) k-means $k = 6$ over the same color spaces. Bottom: Color quantization with growing $k = 4...64$

Implement an algorithm improvement. (0.5 out of 4) Note: If you find it necessary you are welcome to change the default source code. Please use the file 01_kmeans.py for your implementation.

Submission: This is *Exercise 1/4*. The tasks are designed to be solvable in 2 weeks. **Information on due date and a link to upload your solution as a .zip file is given in the Moodle system**. Please submit only one single .zip file with the sources of your solution. **Please insert all necessary images so that each task is directly executable. Otherwise I reserve point deductions.**. Submission after due date will be deducted with 3 points for each delayed week.

Please prepare yourself to demonstrate your solution and explain the code in the week after the due date. The quality of your demonstration and the code is crucial for my rating. I will randomly select 3-5 students for each assignment to demonstrate their work. Have fun!