Computer Vision

Exercise Course: Assignments I
WS 2015/16

Harald Scheidl, 0725084
Thomas Pinetz, 1227026
Velitchko Filipov, 0726328

Assignment 1: Colorizing Images

The assignment was to align and combine the three channels of an image (R, G, B) to form a colorized image. The NCC (Normalized Cross Correlation) image matching metric is used to compute the correlation of the different channels and find the shift in pixels between the channels of the image. The NCC value refers to how similar two images (color channels) are to each other. If the value from the NCC computation is high that means that there is a larger the shift between the color channels of the image. Once the NCC value has been computed the R, G, B channels can be aligned and combined to form a colorized image. The NCC can be computed using the following formula (implemented in Matlab by the ***corr2(A, B)*** function):

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

Once the NCC has been computed, the image channels need to be aligned. This is done by using the Matlab function ***circshift(A, [x y])***, which shifts the elements of the matrix (image channel) ***A*** by ***x*** in the horizontal dimension and ***y*** in the vertical dimension. In this specific assignment the shifts in the image channels can be between -15 and 15 pixels in both dimensions therefore, we exhaustively search over the interval [-15,15] for both the x and y dimensions of the image until we find the maximum displacement. The displacement value is then used to align the image channels.

The results of our implementation are presented in the following images:

Red channel | Green channel | Blue channel | Reconstructed

Red channel | Green channel | Blue channel | Reconstructed

Red channel | Green channel | Blue channel | Reconstructed

Red channel | Green channel | Blue channel | Reconstructed

Red channel Green channel Blue channel Reconstructed

Red channel Green channel Blue channel Reconstructed

Assignment 2: Image Segmentation by K-Means Clustering

There are many different algorithms to segment images into regions, and K-Means is one of those. K-Means itself is not limited to images.

As input, the K-Means algorithm takes a list of data points (which can be n-dimensional vectors themselves), and a number of clusters k. For each cluster, a mean value will be calculated. Each of those mean values (centroids) has the same dimensionality as a single data point and gets initialized with random numbers. Those random numbers are the reason why the segmented images look a bit different on each run.

Now the data points get assigned to their nearest centroid. The centroids now get recalculated, they represent the mean of all assigned data points.

The so called distortion measure maps all data points and centroids to a real number, telling us how much error we have made by using the current centroids. The ratio of this number between iteration n and iteration n+1 tells us if the algorithms converges. When a user defined threshold is reached, the algorithm stops.

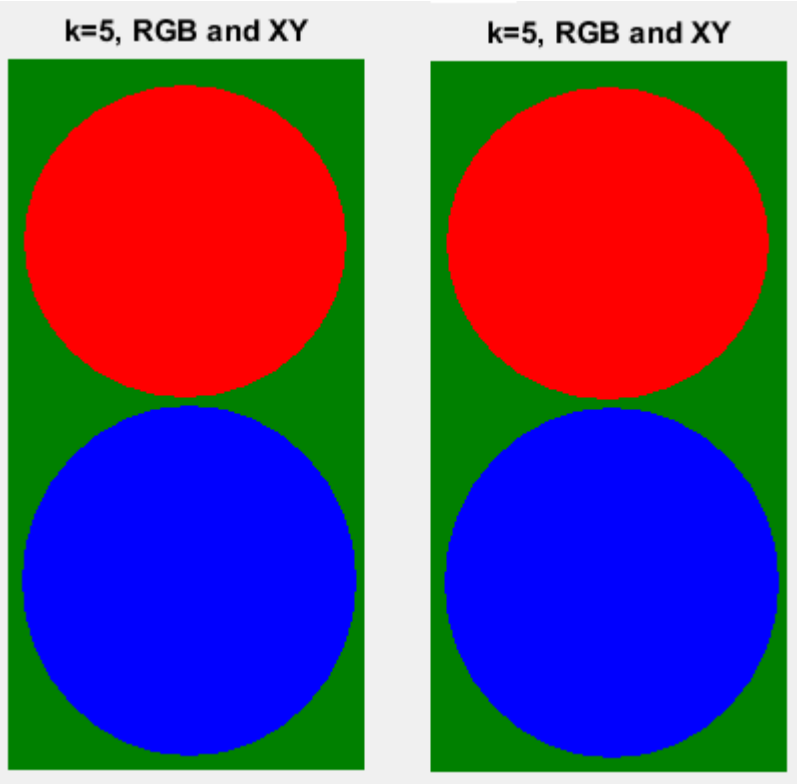
Color images have 3 dimensions, while K Means expects a list of data points. Therefore, the images get converted into a list of (R, G, B) data points in the 3D case or (R, G, B, x, y) in the 5D case, where the latter one also includes spatial information.

After applying K Means to the data, the data gets reshaped to its original dimensions.

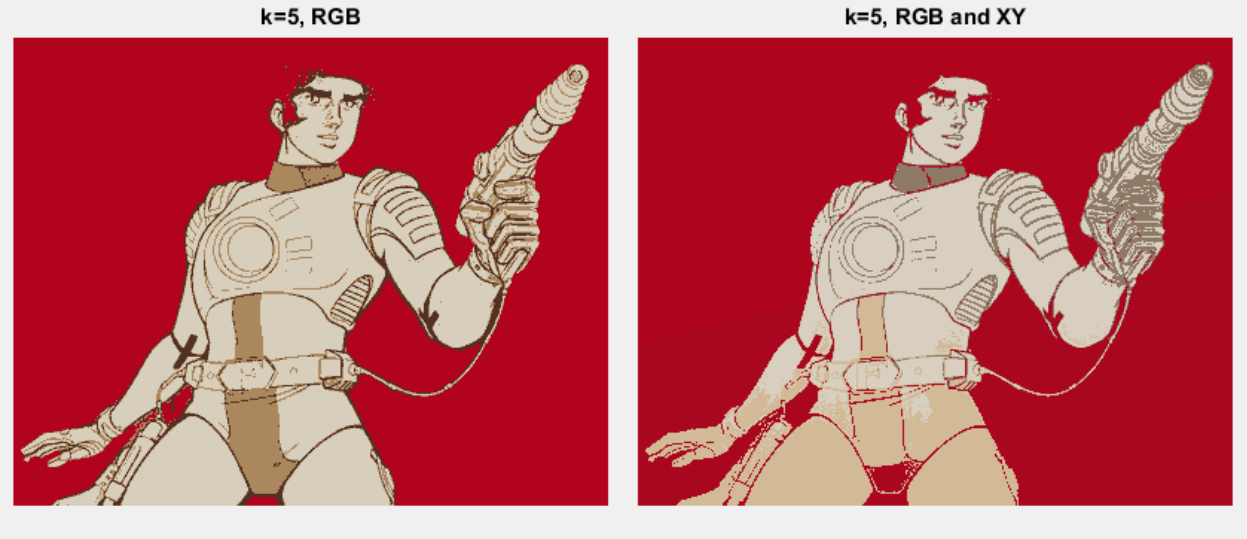
Results

In the following section we show the results of our k-Means implementation. Each picture shows the case of 3d (RGB) data on the left and the case of 5d (RGB and XY) data on the right.

All images with fixed k=5 using 3d (RGB) or 5d data (RGB and XY)



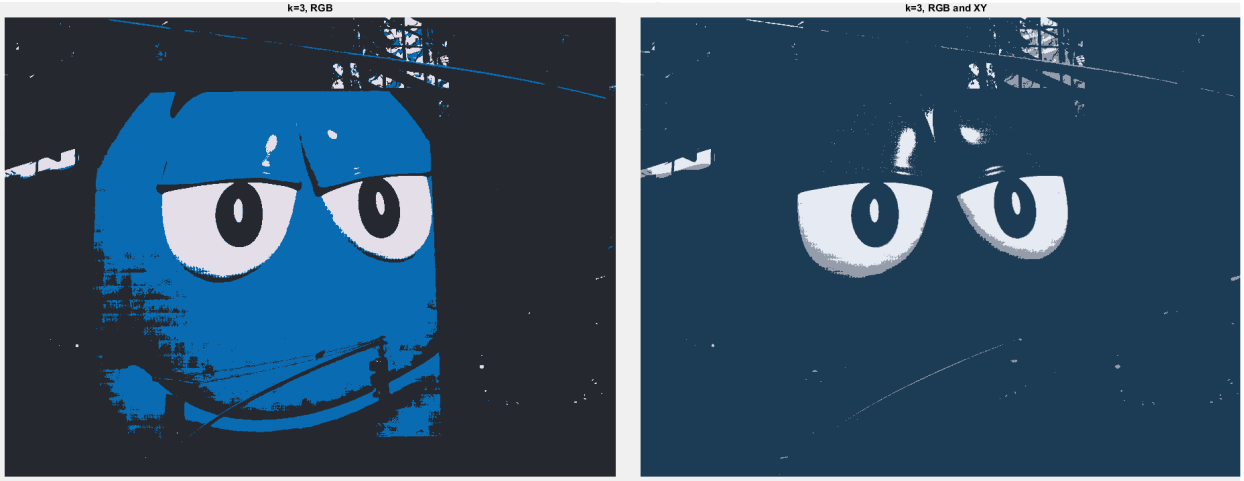
simple.png



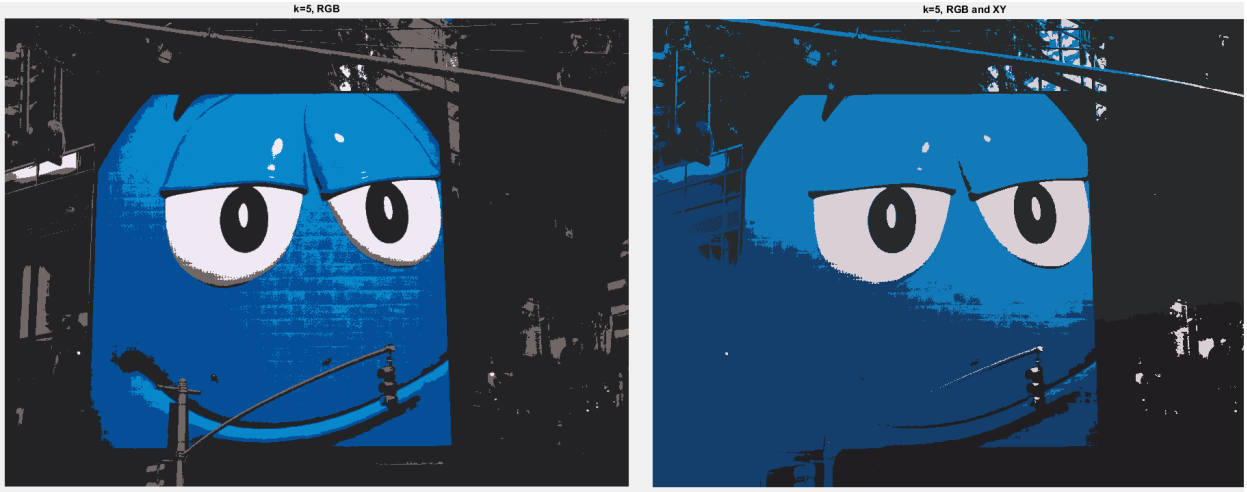
future.jpg

mm.jpg

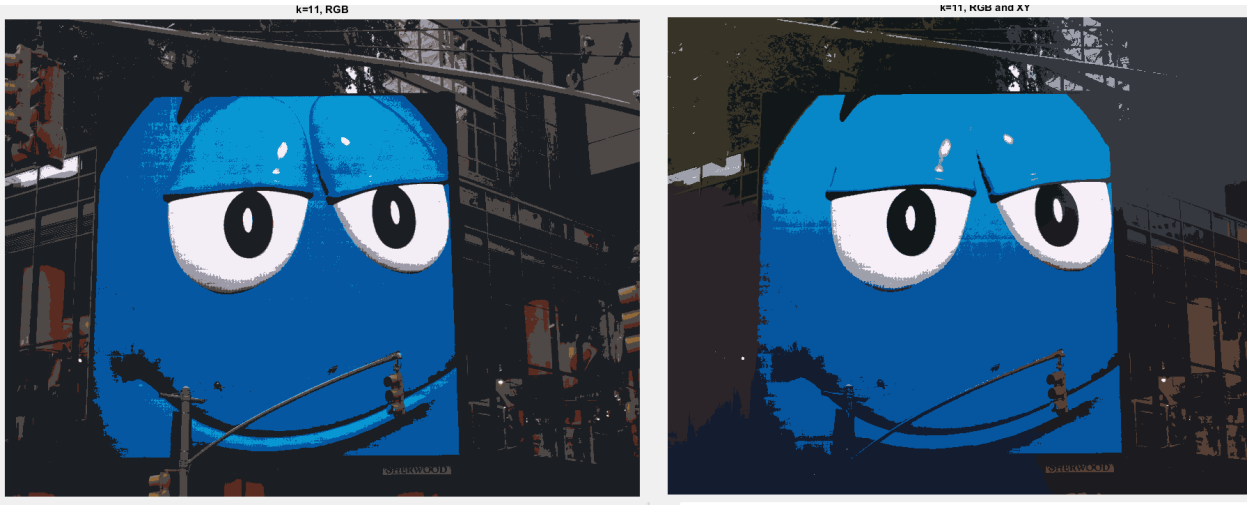
Image mm.jpg with k = {3, 5, 11, 17}



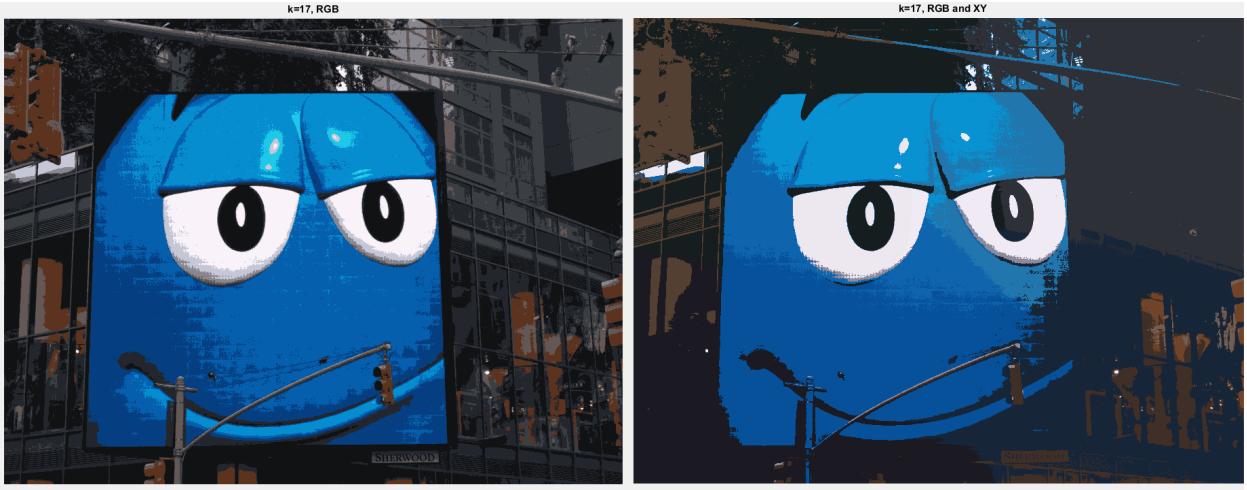
k=3



k=5



k=11



k=17

Discussion of the results

Let us now discuss the questions for this assignment:

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?

As the 5D data also uses spatial information to cluster the data, one would expect that regions adjacent to each other but with slightly different color are seen as one cluster. When looking at the results, this is exactly what happens: when looking at mm.jpg, you can see that regions near the blue foreground object are assigned the same color as the foreground object in the resulting image. You also can see that the foreground object sometimes gets split into two clusters – even though it is the same object. This also results from the spatial information in the data, which forces splitting of data into two clusters if they are far apart (w.r.t. Euclidean distance).

An advantage of 5D data is that regions are smoother, because if a pixel has an outlying color compared to the neighbors (caused by e.g. noise), it still is contained in the cluster of the neighbors.

But as details in images matter, the 3D data leads to better results, even if the results are not as smooth as with 5D data.

Apply different values of K to the image mm.jpg and show the results for both 3D and 5D data points. Interpret the results.

The difference regarding 3D vs. 5D data points was discussed in the last section. Let's look at the effect of the K value: the bigger the value, the more clusters are used and the more the resulting images looks like the original image. Using small values leads to an image in which main objects (foreground, background) survive, just what you would expect from a segmentation algorithm.

Where do you see - based on your results - the strengths and the weaknesses of the method?

The algorithm is simple and quite fast. The algorithm is defined in a pretty general way, so you can apply it to color images, color images with spatial information, or some completely different type of data.

One drawback is that the value K must be specified. The quality of the results depends heavily on this value. Another drawback is that the results are some times better, some times worse, depending on the randomly initialized cluster centroids.

Assignment 3: Scale-Invariant Blob Detection Assignment

Matching an object in an image with the same object in another image is a fundamental task in Computer Vision. Objects can appear in different scale, position and rotation in different images. Therefore scale invariant blobs are needed to be able to match objects to each other. One such technique is scale-invariant blob detection. Scale-invariant blob detection makes use of a Laplacian of Gaussian filter to correctly detect such points, across scales.

For this algorithm there are 3 important parameters:

1. α_0 = Initial Scale
2. levels = The number of scales used for searching.
3. k = Factor with which the scale is increased.

My testing images were extracted with following set of parameters $\alpha_0 = 2$; levels = 10; and k = 1.25.

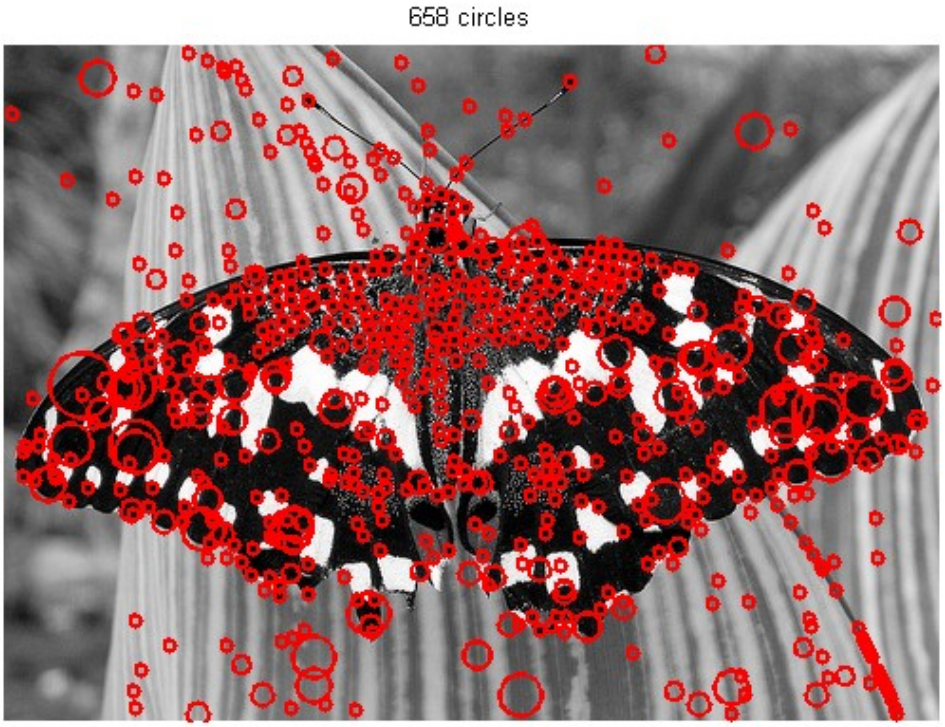
Then the algorithm works as following:

1. We need to construct the scale space.
    a. To do this we initialize a matrix to hold our scale space.
    b. Then we iterate over the levels parameter and do the following
        i. Set the appropriate α for the current level which is α_0 * (k ^ current level).
        ii. Create a Laplacian of Gaussian (LoG) filter using α as parameter and 2* [3 α + 1] as filter size.
        iii. Use the filter on the original image and store it in the scale space matrix
2. We need to find local maxima as scale invariant points.
    a. To do this we compare every point to each of his neighbors on his scales and his neighboring scales.
3. The algorithm terminates with a vector of points with corresponding scale information.

Results

In the following section we will present our results from implementing this algorithm.

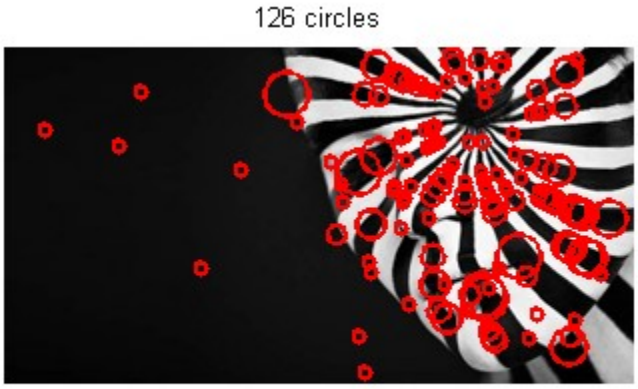
Test image:



Test image on half size:

My image:

My image on half size:



Discussion of the results

In this section we will discuss our interpretation of our resulting images.

As can be seen in all images most of the circles are very similar in both images. Most circles are of similar size on similar regions. However on the smaller image a lot less circles have been detected. This has to do with the information loss on applying the resize operation. Therefore a lot of the smaller circles have now vanished. In addition some pixel have been shifted to other local maxima and therefore changed the circle size a bit.

In the two test images in particular it can be seen that circles on thin edges are lost on the lower resolution image, for example in the lower right corner. In contrast to that, in my own image it is a lot less obvious where the circles are lost as we reduced the resolution. As before many smaller circles are lost. However as there are less thin edges, it is not as obvious.