Exercise Course: Assignments I

VU 183.585 Computer Vision WS 2014/15

Matthias Gusenbauer	1125577
Robin Melán	1029201
David Pfahler	1126287

Table of Content

- Assignment 1: Colorizing Images
  - Methodology
  - Bonus Task Improvements
    - Only compare the center of the images
    - Image Pyramid
    - Laplacian of Gaussian (LoG) Feature
  - Results
- Assignment 2: Image Segmentation by K-means Clustering
  - Methodology
  - Results
    - Image Sample: “simple.png”
    - Image Sample: “future.jpg”
    - Image Sample: “mm.jpg”
    - Image Sample: “mm.jpg” with different values of K
      - Strengths
      - Weaknesses
- Assignment 3: Scale-Invariant Blob Detection
  - Methodology
  - Results
    - Half sized images
    - LoG Response

Assignment 1: Colorizing Images

In this assignment digitized glass plate images, which were taken with blue, green and red color filters, had to get aligned onto each other to create a correct RGB image. (see Figure 1) Two different algorithms are presented to align the images (see Table 1,2)

Figure 1: Glass plate images (left) and a correct RGB image (right)

Methodology

After loading the image, the size of the search kernel gets determined. Experiments showed that the misalignment of the images is at a maximum of 5%. Because of that assumption the search size is set to 5% of the shortest edge of the image.
The digitized red filtered glass plate image gets set as the correct aligned image and the green and the blue image are getting aligned to the red image.
This alignment is implemented using the Normalized Cross-Correlation (NCC), which is a sufficient choice for the image matching metric. The image that gets aligned then gets shifted in x and y direction stepwise by 1 from -search size to +search size and for every shifted image the NCC gets calculated. The shift with the best NCC score gets returned and the image gets aligned.
Sadly this operations are quite expensive for big images and/or big search sizes, so we improved the standard approach with several concepts (see Table 1).

Bonus Task Improvements

Several improvements can help calculating the alignment faster and more precise. (see Table 1)

Only compare the center of the images

The border of the glass plate images contains several image artifacts and other flaws. In our implementation only the center of the image namly 70% of the image were taken into account for comparison. That reduces the number of pixels that have to get compared by the NCC-metric by 30% and due to the artifacts and flaws that occur on the border of the image the alignment is more precise. To compare the results with and without this optimization the image *00153v* was considered. This image failed by only aligning it with the image pyramid, but was aligned successfully with the named optimization (without using the LoG-Feature). (see Figure 2)

Image Pyramid

This optimization uses an image pyramid to reduce the search size to one and then apply the standard algorithm to all levels of this image pyramid with a search size of one.
First the number of pyramid levels has to get determined. In an image pyramid the size of the image gets quartered on each level or the sides of the image get bisected. Due to that the image kernel also can get bisected on each level. Therefore the maximum level of the image pyramid is the logarithmus dualis of the search size.
After the creation of the image pyramid for the input image and the image that has to get aligned, the algorithm iterates from the top level image (smallest) to the lowest (biggest).
On each iteration the best match shift of the alignment of the same levels of the two image pyramids gets calculated with the standard algorithm mentioned above, but only with a search size of 1. Additionally the previous calculated shift gets taken into account as offset for the calculation. This shift gets multiplied by 2 because it was calculated from a higher pyramid level. On each iteration the image alignments becomes more precise.

Figure 2: alignment by only comparing the center of the images (left) standard implementation (right)

Laplacian of Gaussian (LoG) Feature

A better feature for calculating the NCC is the LoG. It is better because the color channels of the glass plate images do not have the same brightnesses but do have the same edges (high frequencies) in the image. The LoG is an approach to extract the edges from an image and is therefore less sensitive to brightness differences.

Results

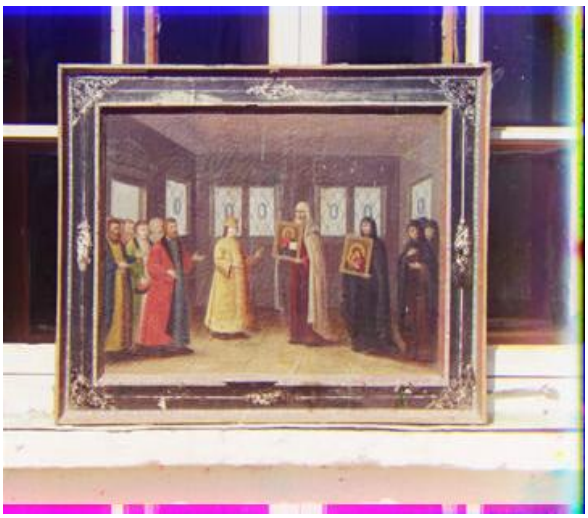
Table 1 shows the runtime improvements of the bonus task. Images with a size of 3000 x 3000 pixels need several hours to calculate with the simple standard algorithm and need a maximum of 5 seconds with the improved bonus algorithm. Obviously this is due to the search kernel that has to be 5% of the size of the smallest edge of the image which would be 150 pixel. This leads to 150^2 calculations of the NCC-metric, which is an expensive operation itself.

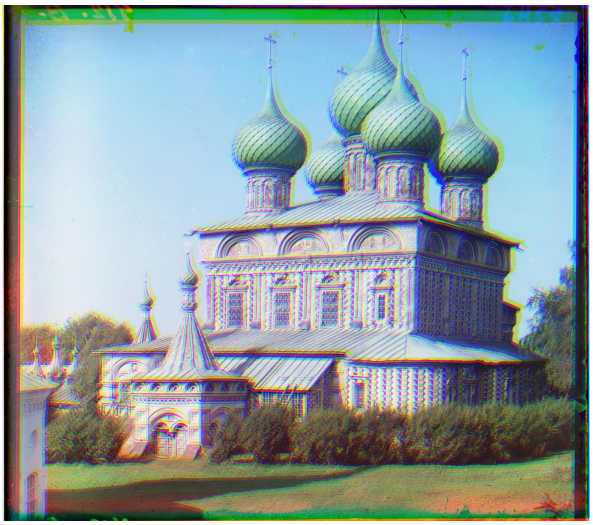
Image	Standard Implementation	Bonus Task Implementation
00125v	1.940 s	0.0752 s
00149v	2.099 s	0.0691 s
00153v	2.011 s	0.0643 s
00351v	2.015 s	0.0608 s
00398v	2.041 s	0.0634 s
01112v	1.999 s	0.0686 s
00882a	~5,5 hours[1]	4.9252 s
00876a	~5,5 hours	4.6978 s
00245a	~5,5 hours	4.9205 s
01043a	~5,5 hours	5.0468 s
01251a	~5,5 hours	4.8056 s

Table 1: Runtime comparison of the two algorithms on an Intel Core i7-3770K, 4x 3.50GHz

[1] This time is calculated with the assumption that a 300x300 pixel image has a search kernel of 15 px and needs a calculation time of ~2 seconds that are $15^2=225$ calculations of the NCC of the 300x300 pixel image. a 3000*3000 pixel image has a search kernel of 3000/20 = 150 (=5%) and needs $150^2 = 22500$ calculations (100 times more calculations) of the NCC of the 3000x3000 pixel image. The time of calculating the NCC is also dependable of the image size. The image size increased by 100 times (3000x3000 / 300x300) so the computation time extends to 100*100 times = 2 * 100 * 100 = 20000 seconds = 5,55 hours

Without Alignment	Aligned Image	Comment
		this is the image *00351v.*
		this is the image *00153v*. This image made several problems with the unmodified algorithm. Only after taking the LoG feature into account the alignment was correct.

		this is the image *00149v.*
		This is the image *00125v.*
		This is the image *01112v*.

		This is the image *00398v*.
		This is the image *01251a.* It has a size of 3589 x 3146 pixel
		this is the image *00245a*. It has a size of *3633 x 3199* pixel The image is blueish because the original white balance of the image is wrong.

		This is the image *01043a*. It has a size of 3712 x 3202 pixel
		This is the image *00882a*. It has a size of 3571 x 3190 pixel
		This is the image *00876a*. It has a size of 3434 x 3177 pixel The original images are also a little bit rotated and the lower part is missing.

Table 2: Output Comparison of the two algorithms

Assignment 2: Image Segmentation by K-means Clustering

Methodology

After reshaping the RGB image from a MxNx3 Matrix into a (M*N)xD matrix we need to define a K to start our K-Means Clustering function. The (M*N)xD matrix represents our objects (data points) with a dimension either D = 3 for evaluating only the RGB values or D = 5 taking not only the RGB values but also the pixel coordinates into our account. The parameter K defines the number of clusters. In the example pictures below we took for K = 3.

1. First all dimensions have to get normalized so that they are in the same range. For that every entry gets subtracted by its minimum and then divided by its maximum, so that all dimensions values reach from 0 to 1.
2. The K-Means Clustering Algorithm starts off by choosing random values for our starting cluster centroids (depending on our K and D):

```
u = rand(K,D);
```

3. All data points from the (M*N)xD matrix are assigned to their nearest cluster centroid, by computing their euclidean distance to every cluster centroid and taking the minimum.
4. After assigning every data point to a cluster we compute the objective function (distortion measure which measures the sum of the quadratic deviation to the cluster centroids), which is given by:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r(n,k) \, ||x_n - \mu_k||^2$$

This distortion measure is our termination condition for our loop because after step 5 we compute J again with the new cluster centroids and terminate our kmeans function if J stayed the same (which means the cluster centroids did not move) or the new J is bigger than the old J (which means we have reached our minimal/optimal deviation)

5. We compute the cluster centroids considering the assigned data points.
6. The distortion measure is computed again (new J) to compare it with the old J

Results

Our results for the three images (simple.png, future.jpg and mm.jpg) with a constant value of K = 3 and considering the dimension value of D on the one hand D = 3 and on the other hand D = 5.

Image Sample: “simple.png”

Table 3 presents the output of the K-means Clustering with the color information only (D = 3) or the color and spatial information (D = 5) of the image “simple.png” with K=3. The clustering worked fine and it did not show any differences between the output from D=3 and D=5.

Figure 3 shows the labeling of the same image with K=4 and D=5, which is one example where the clustering between D=5 and D=3 is not the same. Because the clusters stay the same when there are only 3 different data points possible like in this example (RGB).

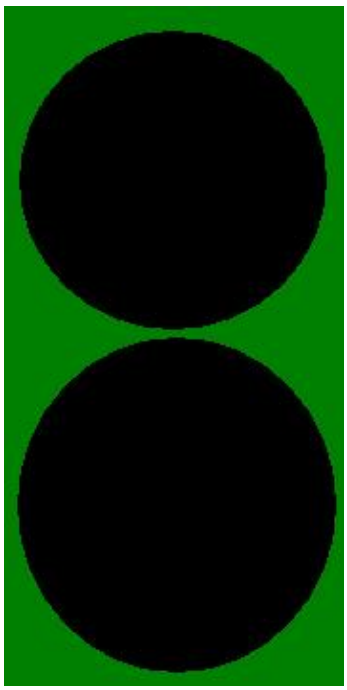
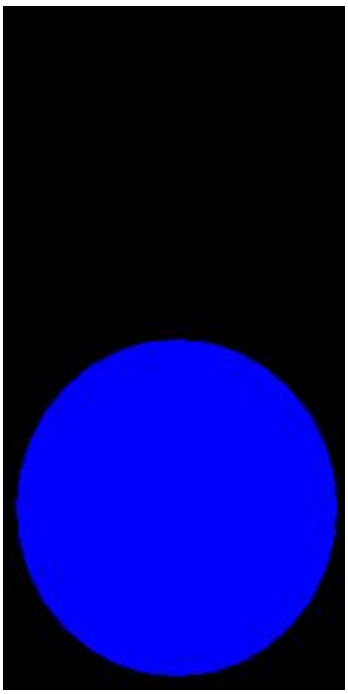
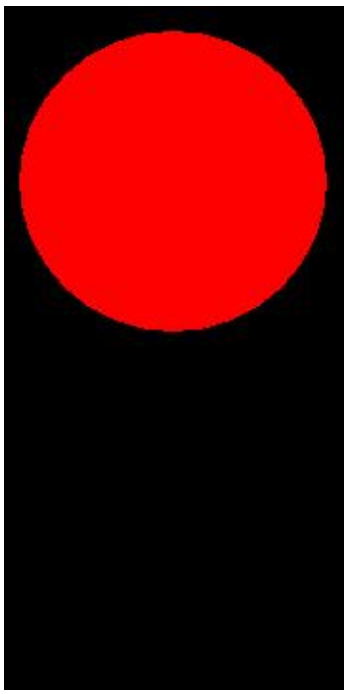
Cluster 1	Cluster 2	Cluster 3
		

Table 3: clustering of “simple.png” for D=3 and D=5 and K=3

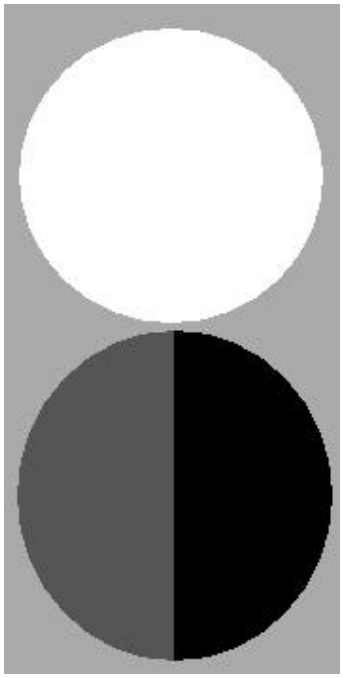


Figure 3: clustering of “simple.png” for D=5 and K=4

Image Sample: “future.jpg”

Table 4 presents the output of the K-means Clustering with the color information only (D = 3) and the color and spatial information (D = 5) of the image “future.jpg” with K=3. For this image there are significant changes drawn to the data points:

K Means Clustering with **D = 3**	K Means Clustering with **D = 5**
 Cluster 1	 Cluster 1
 Cluster 2	 Cluster 2

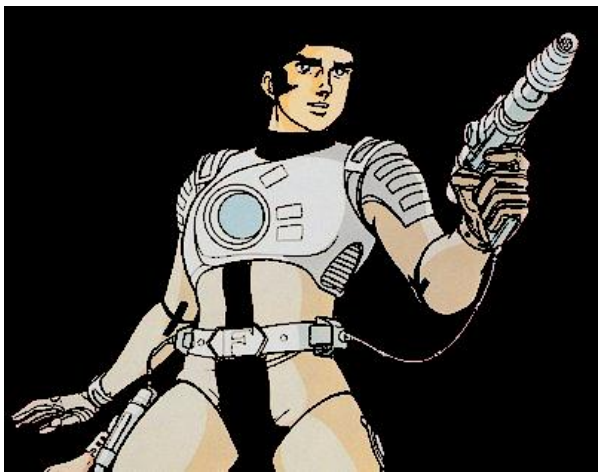
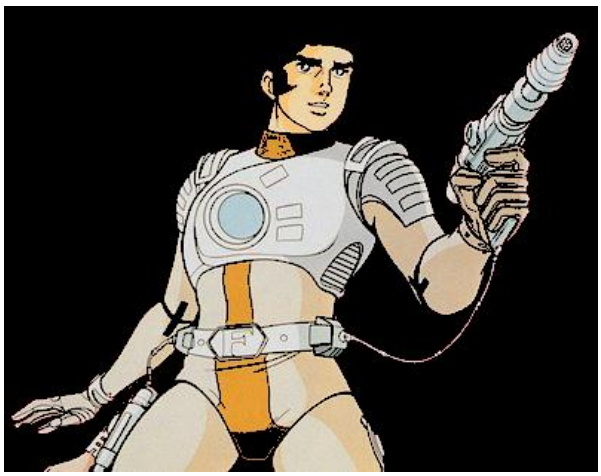
 Cluster 3	 Cluster 3

Table 4: clustering of “future.jpg” for D=3 and D=5 and K=3

Looking at this example it is preferable to choose data points only consisting of the RGB values.
If we apply in general the k-means algorithm on images which contain animated non-real world content (e.g. Logos, Comics, Graphics), these images are composed of RGB values which are easy to distinguish and normally showing objects surrounded of background which does not contain a very high color gradient. In that case for D = 3 such a area will be defined in one cluster but for D = 5 since the pixel coordinates are included in the algorithm they will be separated into 2 or more clusters depending on the value of K. Therefore the advantage here for not including the pixel coordinates in the data points is that objects are easily segmented from a simple (one colored) background.

Image Sample: “mm.jpg”

Table 5 presents the output of the K-means Clustering with the color information only (D = 3) and the color and spatial information (D = 5) of the image “mm.jpg” with K=3. As for this example there are again significant changes between processing only the RGB-values on the left and having data points which consist of RGB-values and their pixel coordinates on the right.

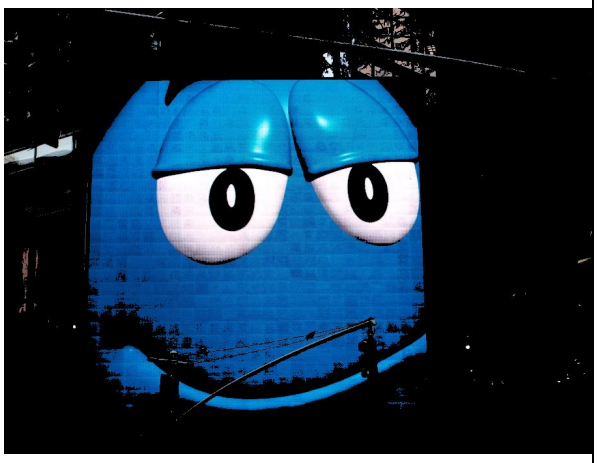
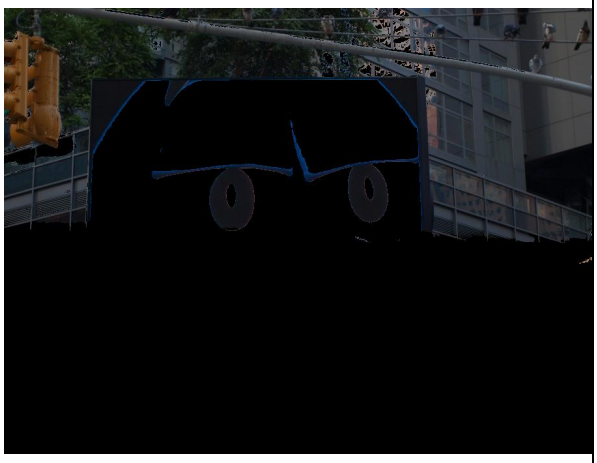
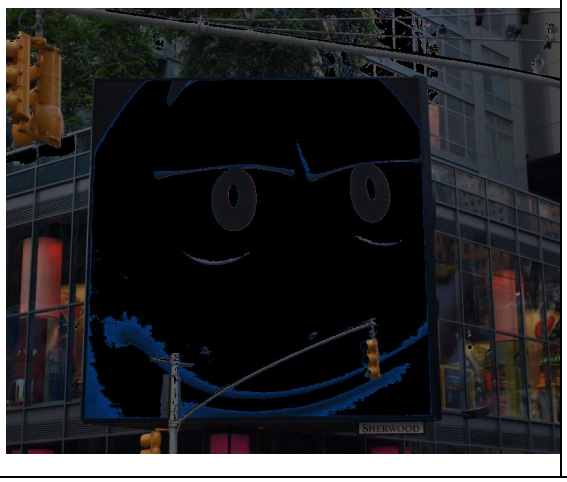
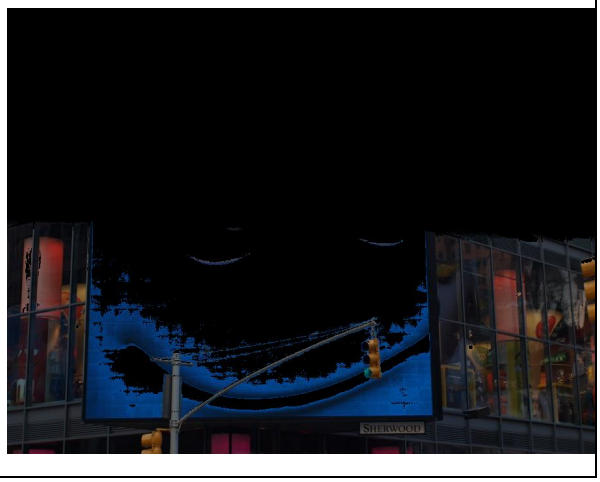
K	K Means Clustering with **D = 3**	K Means Clustering with **D = 5**
1		
2		
3		

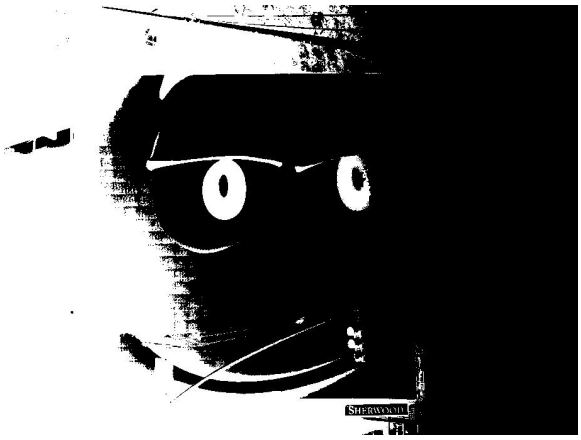
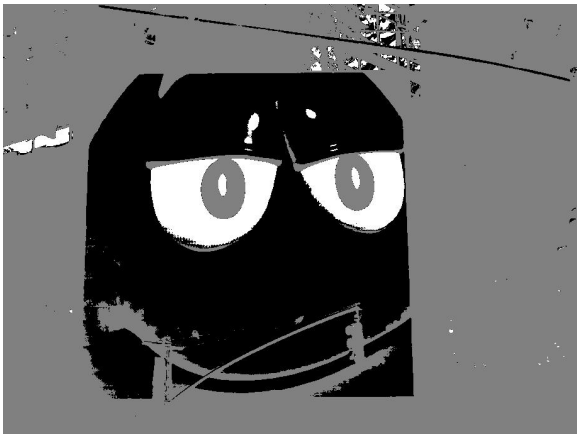
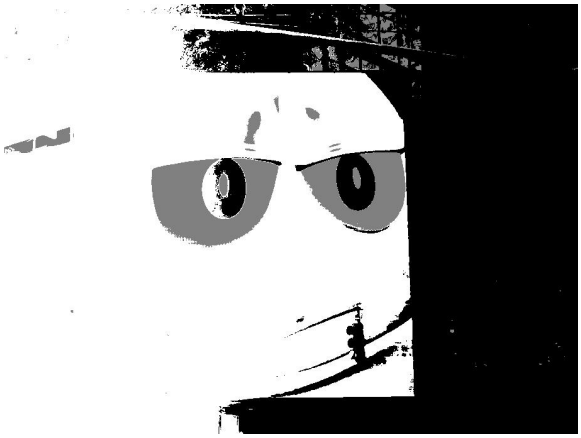
Table 5: clustering of “mm.jpg” for D=3 and D=5 and K=3

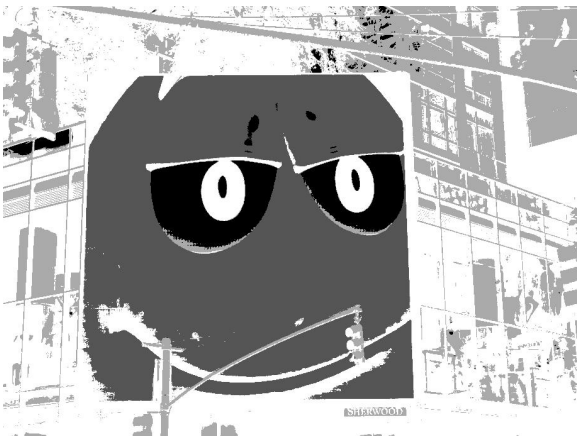
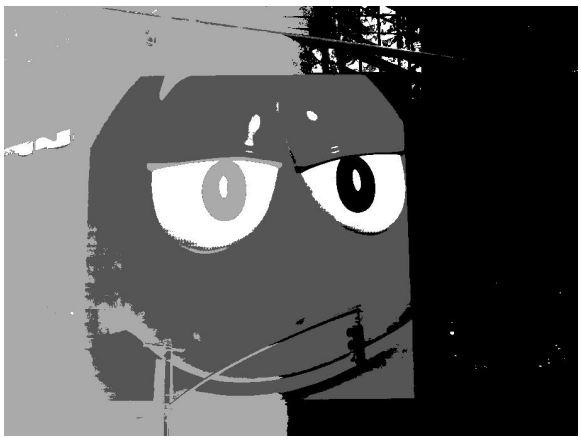
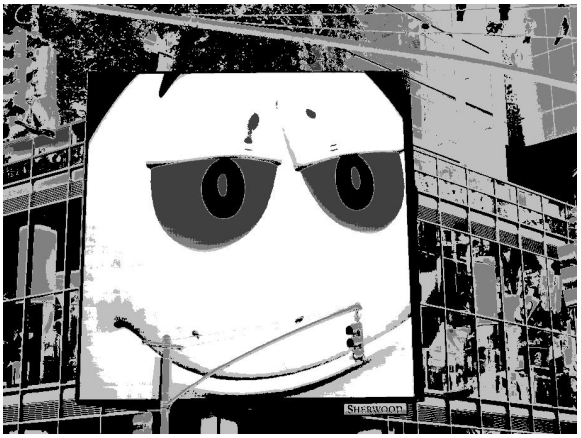
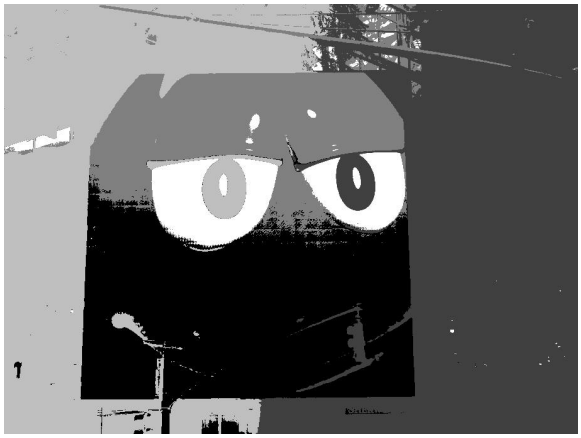
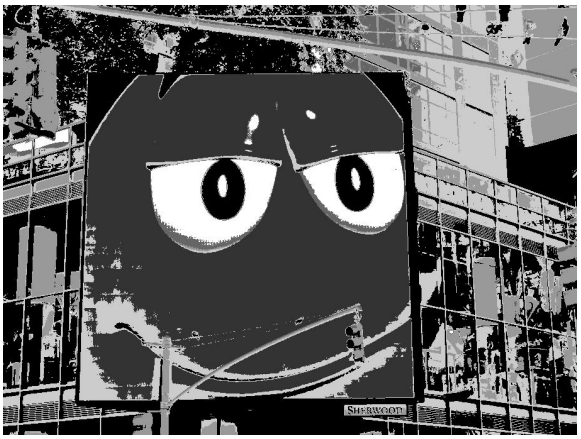
By taking the pixel coordinates into the our observation, we detect the foreground object more robustly, like shown in the first (top right) cluster taking the eyes and the face as one segmented partition. On the other hand depending on the random value of the starting cluster

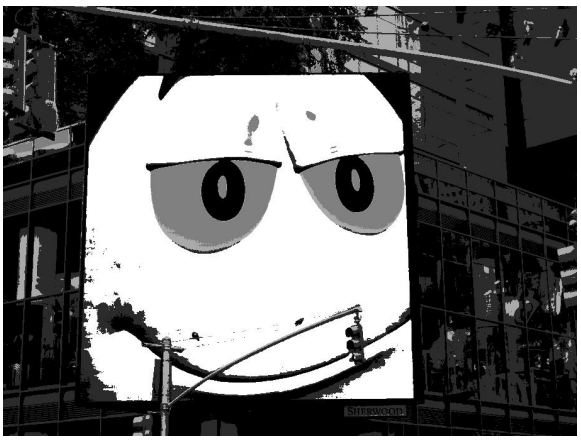
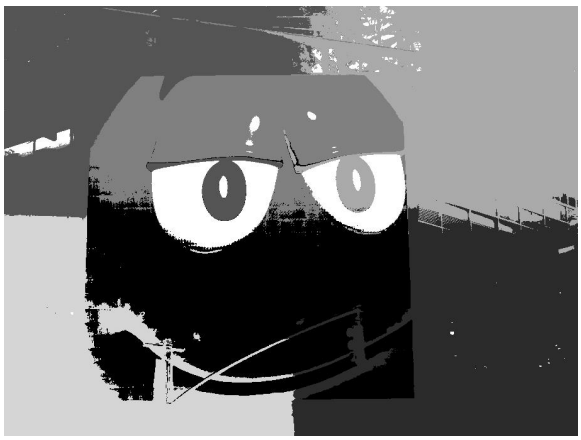
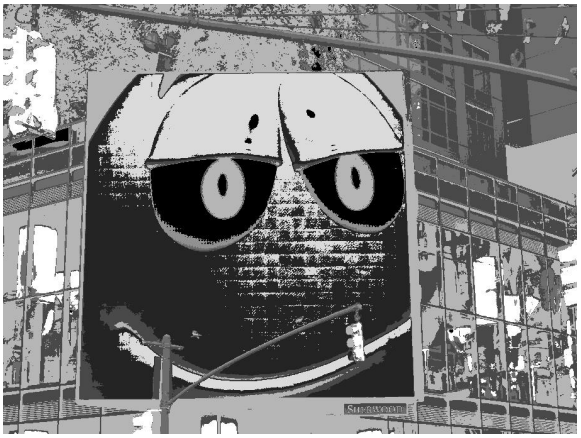
centroids the image background is splitted up horizontally or vertically, which is of course not the case for the clusters on the left side taking only the RGB values into account.

Image Sample: “mm.jpg” with different values of K

Table 6 presents different values of K to the Image mm.jpg with both D = 3 and D = 5 data points. The different levels of gray show the k-means labels depending on the value of K (= Number of Clusters).
For example in the first row we applied the k-means function on the image with K = 2 and the result shows one cluster with its labelvalue “white” and the second label value “black”.

K	RGB	RGB + Position
2		
3		

4		
5		
6		

7		
8		
9		

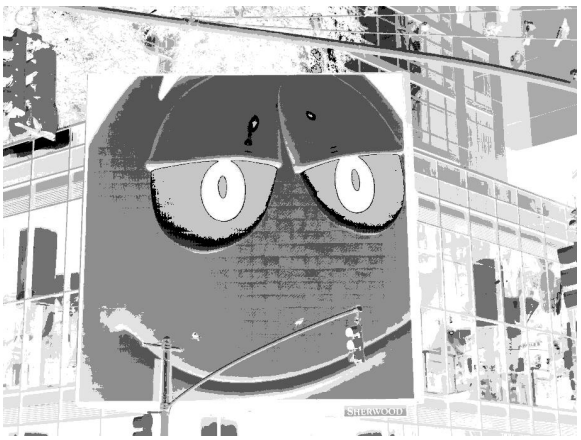
10		

Table 6: Different results for different k values for the image mm.jpg

Running k-means with D = 3 (only RGB values) or D = 5 (including pixel coordinates) the larger K is chosen, the more partitions will be added and more intensities are available. However for the first column (D=3) the result clusters for larger k consist of regions which are not well-connected therefore not very meaningful. Therefore it is important to select a “proper” value for k. As for the second column (D=5) there is less noise and regions are better segmented. But the large foreground region (after k=5) and large background regions (after k=4) which should be coherent have been broken up, because points got too far from the center. Also after k=5 it is easy to see that neither the color clustering, nor the color and position clustering were able to identify the texture of the foreground object and broke this cluster into hard to detect borders.

Strengths

The main strength of the k-mean algorithm is the computational efficiency on a big set of data. According to Lloyd's algorithm[2] the time complexity of the k-means algorithm is O(knI), where **k** is the number of clusters, **n** the number of objects and **I** the number of iterations (which is dependent on the stopping criterion). Additionally it is easy to understand and implement the algorithm in contrast to other clustering algorithms and it is easy to extend the algorithm with more features or dimensions.

Weaknesses

The k-means algorithm prefers groups with a low variance distribution and similar size. Noise or outliers influence the computation of the cluster centroids significantly and the algorithm doesn’t contain measures against such effects.
Furthermore the selection of the starting cluster centroids can be very significant to the evolving segmentation. In our implementation the initial cluster centroids are set by random values between 0 and 1 which of course can lead to strange initial grouping which are determining the clusters significantly. For example the algorithm got stuck in a local minimum.

[2] Lloyd, Stuart P. (1982), *Least squares quantization in PCM*, IEEE Transactions on Information Theory 28 (2): 129–137

Another disadvantage as shown in *“Image Sample: “mm.jpg” with different values of K”* is that the number of Clusters (k) has to get defined beforehand and by choosing an “inappropriate” value for k a poorly clustering can be the result.
In addition the algorithm assumes that all dimensions have the same weight, which is a problem because it is not possible to guess which dimension contributes more to the grouping process.
One weakness in our implementation of the algorithm is that we use the arithmetic mean to calculate the cluster centroids, which is not robust to outlines, which could deflect the centroids with data points that are far away. An alternative measurement would be to calculate the cluster centroids with the median, but this is a more expansive operation.
The shape of the cluster is dependent on the used metric. If the distance to the cluster points is calculated with the euclidean distance the clusters are circular shaped. This is not a real weakness but the user of the algorithm should keep that in mind that the used metric changes the result.

Assignment 3: Scale-Invariant Blob Detection

Methodology

Scale invariant blob detection can be seen as a two stage process in general. The first step is building a scale space of filter responses using laplacian of gaussian as filter kernel.The second part is to find maxima in the scale space, which is also there to identify blob positions in the image. Scale invariance should be more theoretical because there is not infinite computing power and memory available in computers. Therefore we created a scale space of ten levels which is just a part of the total theoretical scale space. In order to do that we first initialize the sigmas with the given parameters $sigma_0$, k and levels. $Sigma_0$ is the smallest sigma for the LoG and k can be considered the resolution of our scale space since it is the parameter we multiply $sigma_0$ with to get $sigma_1$. The precalculated sigmas are responsible for the size of the LoG kernels we use to convolute the input image with. The LoG is equivalent to scaling the image and then calculate the second derivative of the image signal. These two steps can, and have been combined into one step by using the Laplacian of Gaussian to increase performance. The image is convoluted level-times to become an image width x image height x levels matrix that acts as our scale space.
In the second step of our scale invariant blob detector, which is also referred to as non-maximum suppression, we iterate over the scale space voxel by voxel and check each if it is a maximum. A maximum is defined as the highest value in its immediate neighbourhood. That means that in a three dimensional space all 26 connected neighbouring voxels have to be less than the currently investigated voxel. If the voxel qualifies as maximum it is also compared to a given threshold to minimize artifacts.
In the last step the coordinates of the maxima are used to overlay the image with circles where x and y in the scale space correspond to the center coordinates of the circle and the z coordinate or level to the radius of the circle. In order to draw the correct size one must multiply $sigma_i$ with $\sqrt{2}$ to obtain the correct radius for the circle. For this step we used

show_all_circles() with a minor tweak to print several images with the overlaid circle instead of just one image that will be redrawn each call.

Results

The results we have found showed that the algorithm we had to use still has room for improvements. As seen in the images below it happens that one blob is detected several times if the shape is not entirely circular. Furthermore is the blob detection input image size dependent since it is only compute a certain range in the scale space. One improvement we found was to calculate the overlay of circles and if it exceeds a given threshold then the circles are merged into a single circle for a blob. Additionally we could observe that the detected blobs are the same if the input image is inverted. That is of course the case, because the scale space only contains the change in the intensities. (See Figure 4)

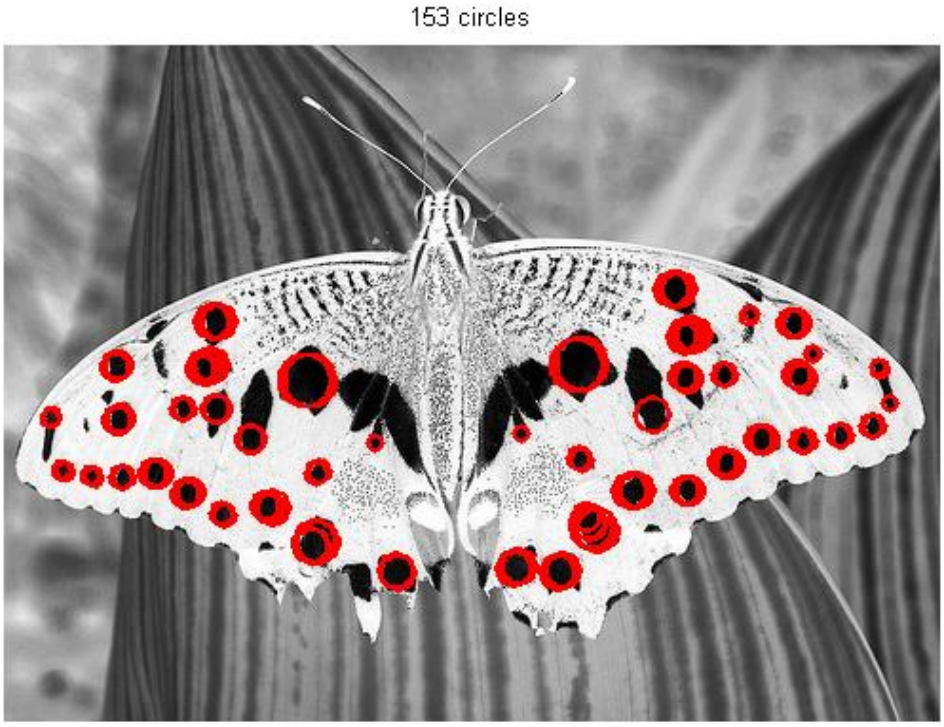


Figure 4: Blob detection for the inverted input picture

Half sized images

Table 7 shows the calculated blob detected images and the half-sized versions of them. The detected blobs are marked with a red circle.

153 circles 	As can be seen in this image the blob detector found 153 circles and a lot of them are overlapping and mark the same blob.
57 circles 	In the smaller version of the image only 57 circles are found. This is a result of the fact that by decreasing the image size many blobs are not in the range of $sigma_0$ to $sigma_{max}$ and are therefore not recognized by the LoG.

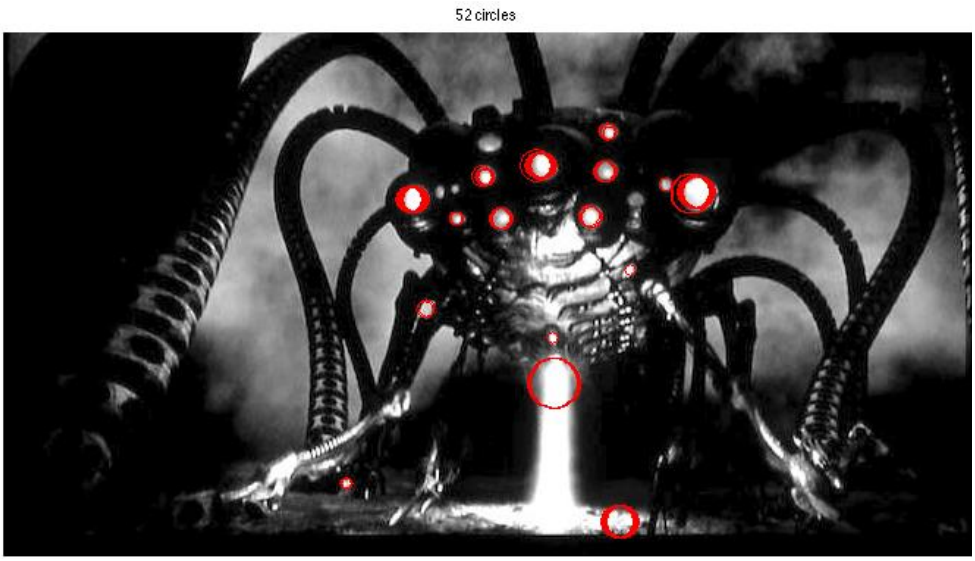
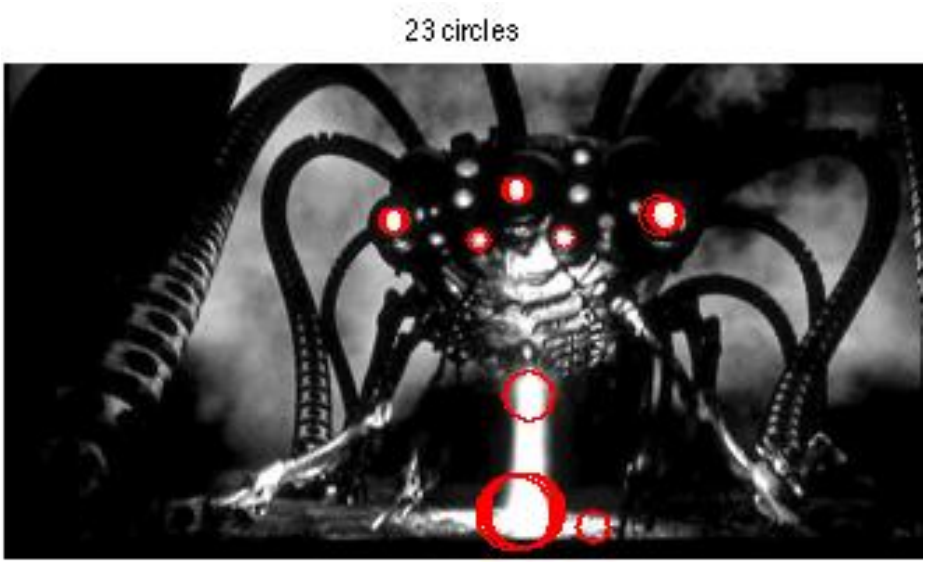
	In this image one can see that the detector perfectly detects the circular features in the image that are within the given sigma band. The detector detected 52 circles which means that in this image there are many detected blobs that are overlapping.
	The smaller version of the same image results in only 23 detected circles and more artifacts. There are less of the circular eyes detected than before.

Table 7: The calculated blob detected images and the half-sized versions of them.

The detector works well enough to detect the majority and most prominent blobs. Given by the fact that the LoG response is dependent on the shape and the hardness of the contours of a blob we found out that our own sample image containing the “squid” from the movie “Matrix“ was difficult in its original state. At first the edges of eyes, the circular bright spots, were too blurry to be detected by the algorithm. We had to do a histogram clipping in order to increase the sharpness of the edges and as a result the response of the LoG convolution.

In summary one can say that the detector works scale invariant within the interval given by the limits of $sigma_0$ and $sigma_{max}$. Furthermore as the LoG is a circular kernel the detector is limited to circular or almost circular shapes.

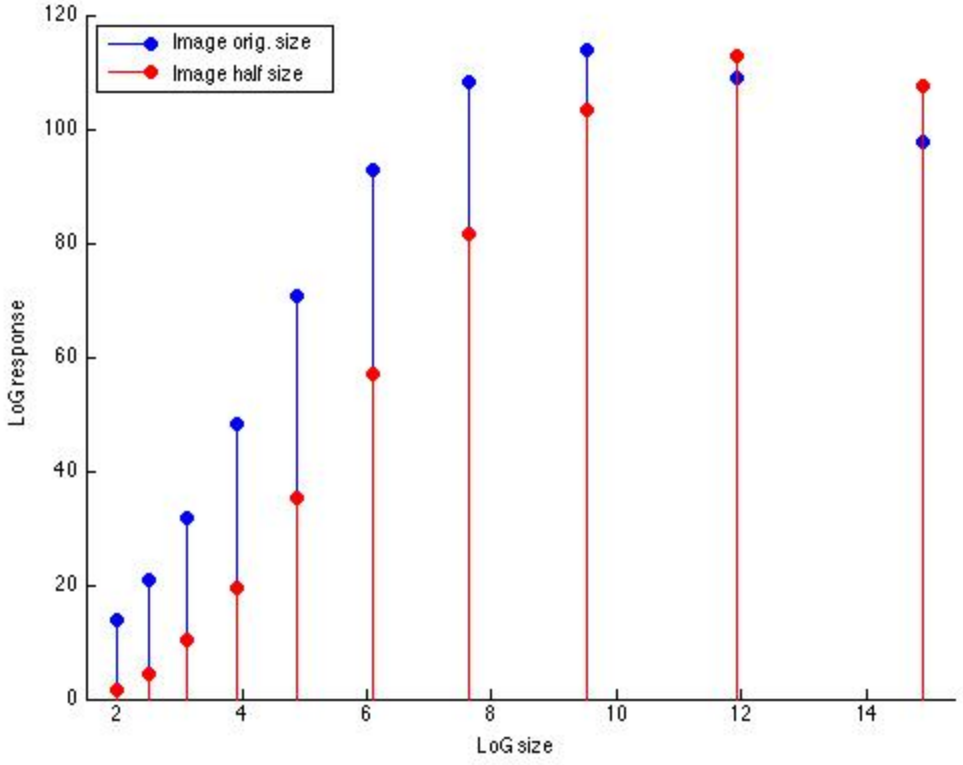


Figure 5: The repsonse of the LoG for all scales for a single detected keypoint of the original image and the image with half the size.

LoG Response

Figure 5 shows the size or sigma of the LoG kernel on the x axis and the responses of the laplacian of gaussian on the y axis. The blue dots indicate the value of the response in the original sized image whereas the red dots correspond to the response when the input image has been scaled by a factor of 0.5. It can be seen that the response in the smaller image are lower for $sigma_0$ than for the original sized image. This is due to the fact that at a smaller scale the initial LoG size is less close to the ideal size than in the original image. In that respect one could say that the responses of the half sized image are similar to the ones for the original size but shifted to the right on the x axis.
This fact is confusing at first but one has to think what the LoG actually means in regards to the image. A smaller sigma means that less of the higher frequencies have been filtered out which can be considered closer to the original image in terms of image size. That means that a higher sigma, which evens out the Gaussian distribution, results in an image with less resolution than the original image. Since the input image of the red bars in the figure above correspond to the smaller image it is clear that the LoG convolution results in a higher response at a higher sigma. The blob in the image scaled by a half is smaller as well and

therefore the response will be higher at a higher sigma due to the fact that a higher sigma for the scale space means that a smaller representation of the input image is analysed. That is the reason why the the responses of the half sized image are higher at a sigma around 12. The responses of the original sized image are already decreasing where the responses for the half sized image still have not reached their maximum by a better match of LoG kernel size and the blob.