

# Report for Week 3

*k*-means & Bag-of-Visual words  
or how to create a Google Goggles-like system

Patrick de Kok (5640318)      Georgios Methenitis (10407537)

November 29, 2013

## 1 Individual steps of *k*-means

In the first step of the assignment we are focusing on implementing the *k*-means algorithm and check the results of these individual steps of the algorithm. Choosing the best K parameter always depends on the application. In the example that we have to cluster the datapoints which consist three clusters is best to use  $k=3$  but this parameter varies for different applications. For example, image segmentation often needs around 10 clusters to be computed. There is also a big trade off between number of clusters and the performance of the algorithm. We know that for a large value for  $k$  algorithm takes long time to converge. A really serious aspect of this algorithm is the relation between its random starting points in the state space and the solution that it gives. There is a big dependency and the algorithm can converge to a different final solutions depending on the starting points for the centers. Figure 1, depicts the results of the three steps of the algorithm with different  $k$  values.

## 2 Running your implemented *k*-means

In this part of the assignment we do the same algorithm but now an iterative approach. After the new assignment of the center points of the clusters we go to the second step again and we compute new centers. We continue the iterations until a maximum number of iterations is reached. Figure 2, depicts the results of the iterative method for 1, 5, 10, 20 iterations, you can see in the figure that the centers are going towards the actual centers of the clusters. However, this is not always the case as the random first step can lead to different results.

Apart from reaching a maximum number of iterations, there is another way of stopping the algorithm, and this is done just by computing the distance between the previous centers and the new ones, once the distance has become not large enough to be important we can stop the algorithm. The following part of code is the one that we implemented for this reason.

```
new_center_x = center_x/num
new_center_y = center_y/num
new_center_dist = math.sqrt(math.pow(new_center_x - means[i, 0], 2) +
                           math.pow(new_center_y - means[i, 1], 2))
if new_center_dist > max_change:
```

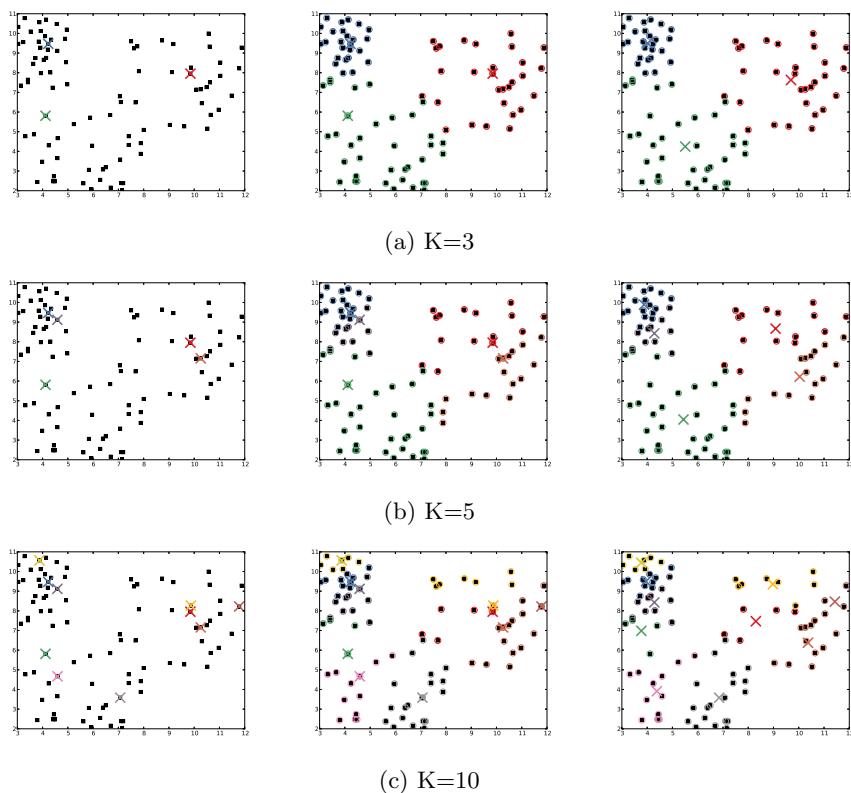


Figure 1: Illustration of the three first steps of the K-means algorithm for  $k=3,5,10$ .

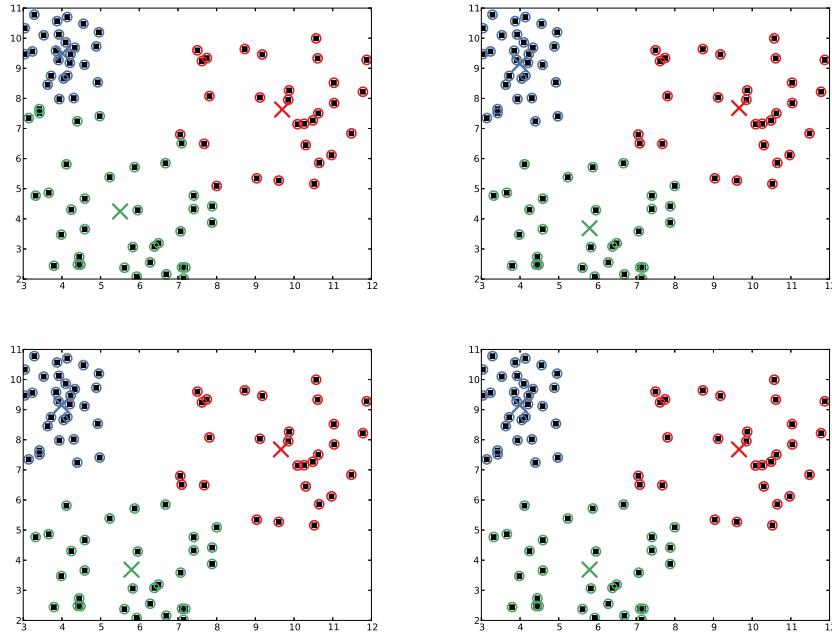


Figure 2: Illustration of the iterative K-means algorithm for different maximum iterations levels, top left:1, top right:5, bottom left:10, bottom right:20.

```

max_change = new_center_dist
means[i,:] = [new_center_x, new_center_y]
if max_change < 0.001:
    break

```

### 3 Color-based image segmentation

Now, we apply the same algorithm on actual images. The main idea is the same but now we are dealing with a high-dimensional space, so we expect this algorithm to be really slow. Thing that applies especially for large K values. We ran the k-means algorithm in the given image for  $K = 3, 5, 10, 50, 100$ . Figure 3 illustrates the resulting color segments of the image. We can see that for small values for K the image for example  $K = 3$ , each pixel is assigned a value of one of the three clusters, the resulting image has only three colors and different features in the picture can easily extracted. As the number of clusters increases the clustering is becoming more and more detailed. For  $k = 100$  we can hardly see any difference between the clustered and the original image.

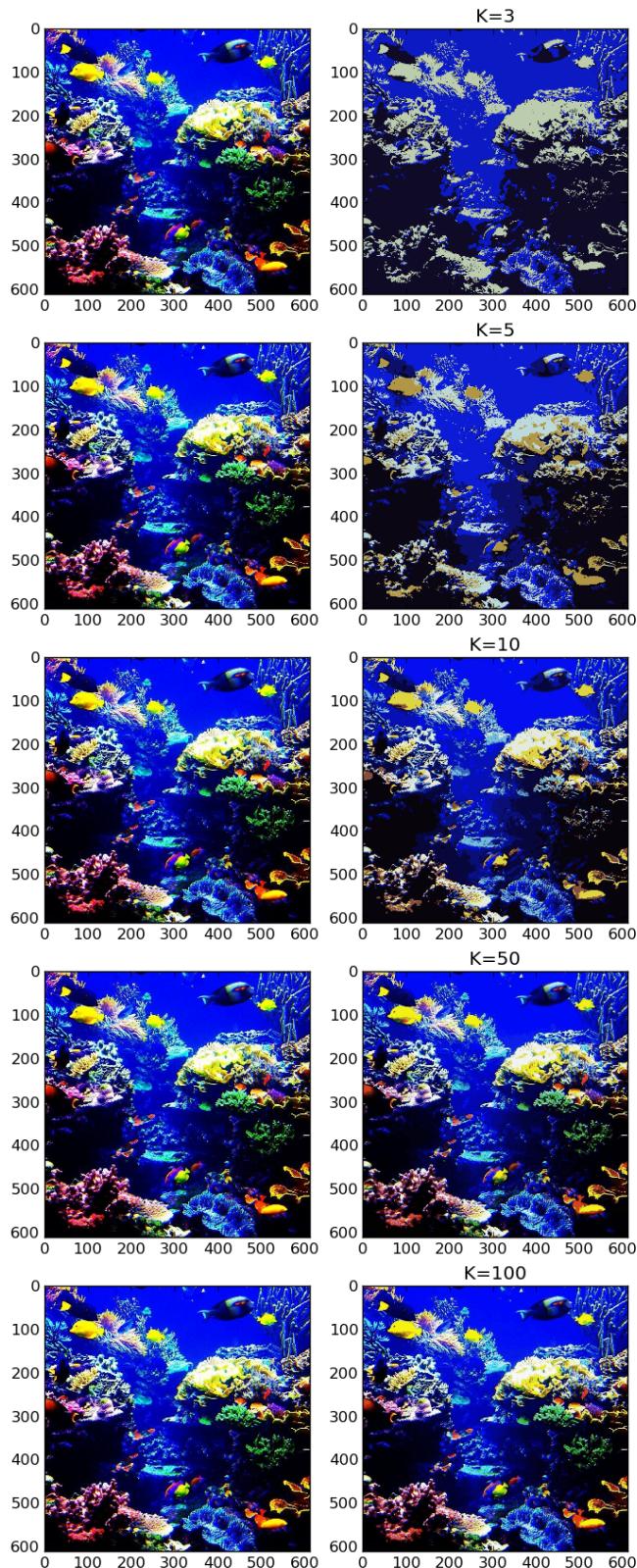


Figure 3: Illustration of K-means algorithm for the above coral image, for  $K = 3, 5, 10, 50, 100$ .

## 4 *k*-means, SIFT & Bag-of-Words: calculating storage for raw SIFT

Each SIFT vector has 128 elements. Each element is represented by an integer<sup>1</sup>. Assuming each integer consists of 2 bytes and there is no storage overhead for the vector, 1 SIFT vector needs 256 bytes of storage. To store all 50,000 vectors that belong to one image,  $12,800,000 \text{ bytes} = 12.8 \cdot 10^6 B = 12.8 \text{ MB} \approx 12.21 \text{ MiB}$  is needed. For Facebook's daily addition of new images, one needs  $12.8 \cdot 10^6 \cdot 350 \cdot 10^6 B = 4.48 \cdot 10^{15} B = 4.48 \text{ PB} = 4,480,000,000 \text{ MB} = 4272460937.5 \text{ MiB} \approx 3.98 \text{ PiB}$  of storage.

## 5 Visualizing the words on the image, the contents of the words and the Bag-of-Words histograms

Visual words of five randomly selected images from the dataset have been plotted in Figure 4. Note that words tend to be focused on structures that reoccur often, such as lines. This can be seen in Figure 4a and 4b at the roof, and in Figure 4c and 4e at the 3D structures on the wall.

The patches of five randomly selected words are shown in Figure 5. The patches do look like each other, when you ignore the differences in hue (but not in intensity), rotation and scale.

Finally, the bag-of-words-histogram representation of the images of Figure 4 is presented in Figure 6.

## 6 Visualizing Bag-of-Words retrieval

See Figure 7 for the ranking.

## 7 Evaluating Bag-of-Words histograms

see section  
5.2

## 8 Bonus 1

## 9 Bonus 2

---

<sup>1</sup>The binary invoked to extract the SIFT vectors returns integers. However, the `read_sift_from_file(sift_path)` function interprets these as Python `float` objects. In the standard Python implementation, CPython, a `float` always consists of 64 bits, which is 8 bytes. This would result in quadrupling all final answers.

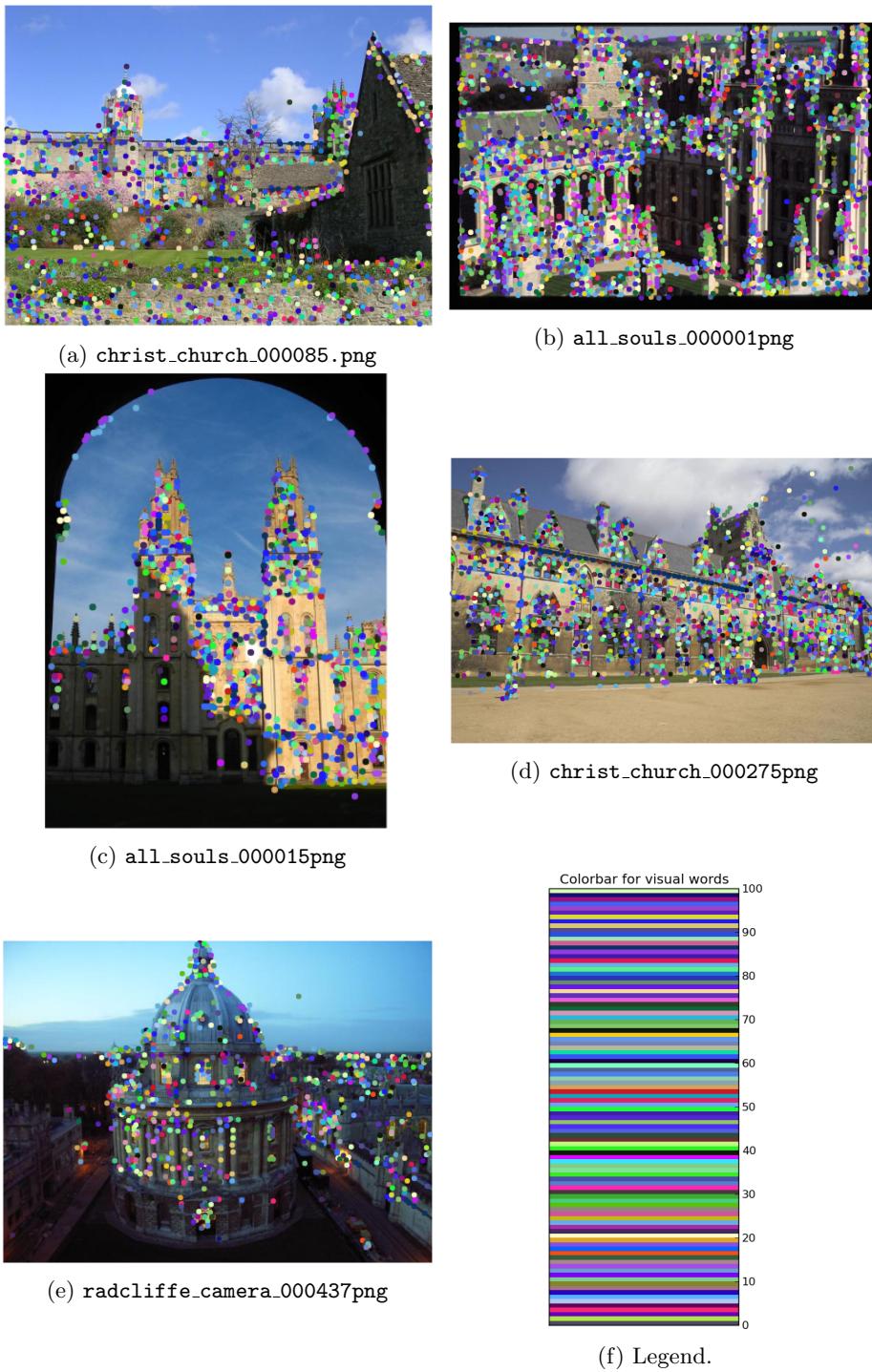


Figure 4: SIFT vectors that have been assigned to words, plotted on images. Figure 4f functions as a legend for each feature: dots of the same color represent the same SIFT vector. Its number can be looked up on this bar.



Figure 5: Patches for five randomly selected words, as they occur in the images of Figure 4.

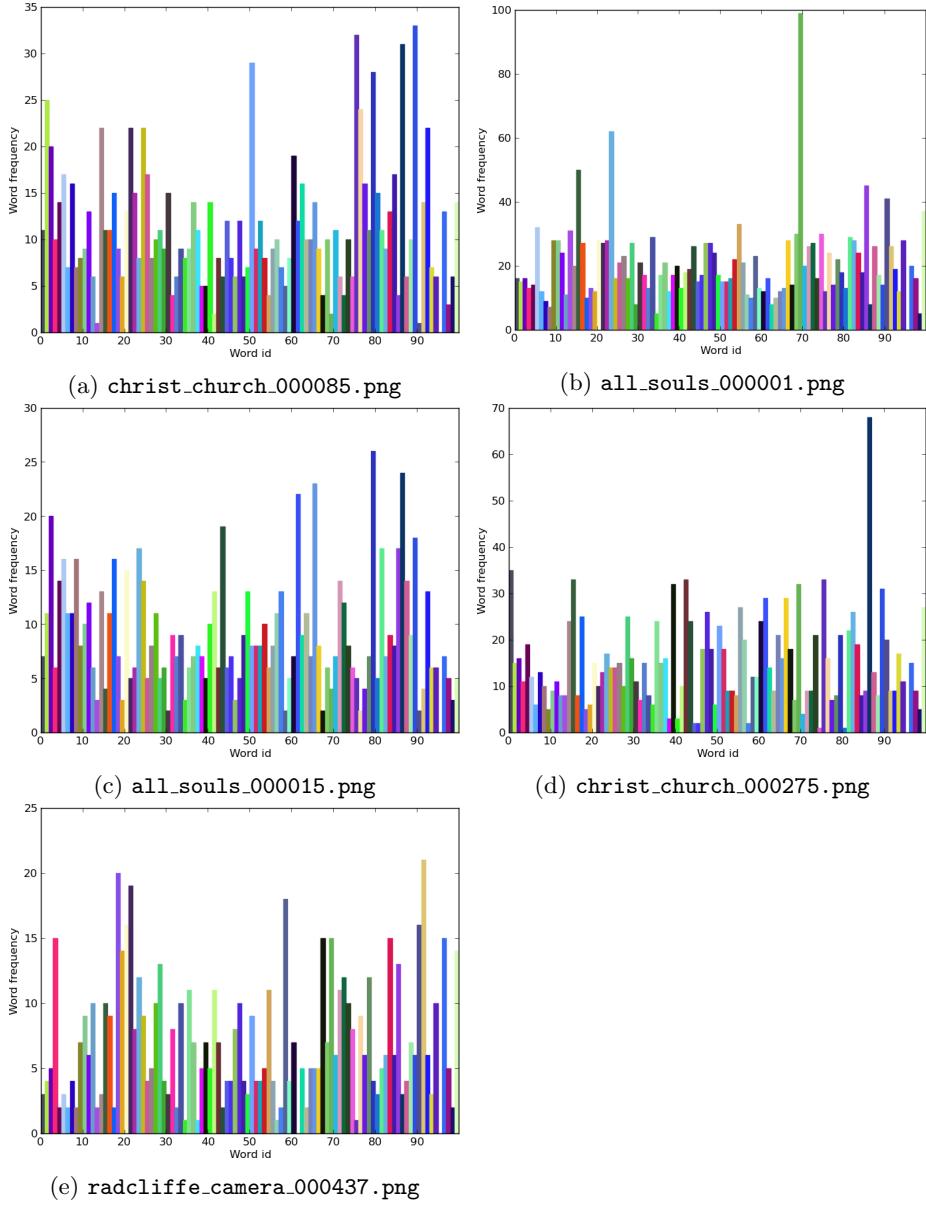


Figure 6: Frequencies of visual words per image. Each bar is visualised in the corresponding SIFT vector color. Figure 4f shows these colors.



Figure 7: Ranking of retrieved images based on different distance measures between histograms of bags of words.