Computer Vision 183.585 VU 3.0 4.5 ECTS

2nd Exercise Round

David Pfahler, Matthias Gusenbauer, Robin Melán
1126287, 1125577, 1029201

January 20, 2015

Assignment 4: Image Stitching

A. SIFT Interest Point Detection

As the first part of the assignment we need to extract SIFT features out of every image separately.[1] Therefore we convert the image to *rgb2gray(image)* and method $[textkey, desc] = vl\_sift(siftIm)$; to extract the features from our image. The *key* value consists of a 4 x n matrix, where n is the number of features found in the image, holding the x-coord., y-coord., the scale (s) factor and the orientation (in radiant). The *desc* value holds an 128 dimensional vector for every *key*, which describe the eigenvectors (See Example in Figure 1).

[1] See Matlab-file *GetSIFTFeatures.m*

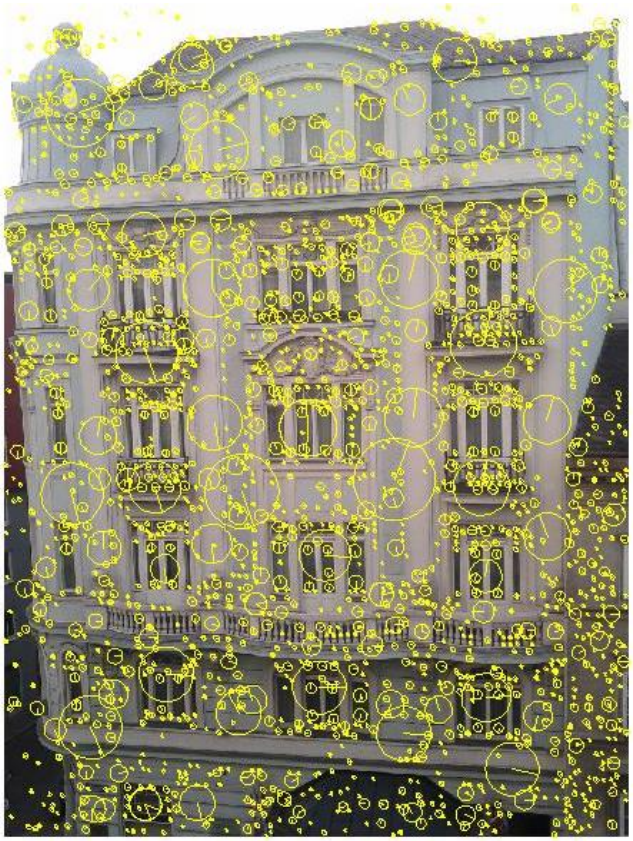


Figure 1: Extracted Sift Features

B. Interest Point Matching and Image Registration

The next step consists of determining the homographies between all image pairs from left to right. So we start of with computing the first homography $H_{1,2}$ and continue until homography $H_{5,4}$ (I switch 5 and 4 because I need to transform the 5th image into the 4th, if I am taking the image in the middle as reference, see further C.).[2]
To compute the homography we first need to follow a couple of steps, starting by matching our descriptors $[matches, scores] = vl\_ubcmatch(descriptor1, descriptor2);$. $Matches$ contains the indices of the original (first) and closest descriptor in the other (second) image. The *scores* value holds the squared euclidean distance between them. $vl_ubcmatch(DESCR1, DESCR2)$ uses a threshold $THRESH$ (the default value of $THRESH$ is 1.5), which means that a descriptor $D1$ is matched to a descriptor $D2$ only if the distance $d(D1, D2)$ multiplied by THRESH is not greater than the distance of $D1$ to all other descriptors.

[2]See Matlab-file *IntPointMatching.m*

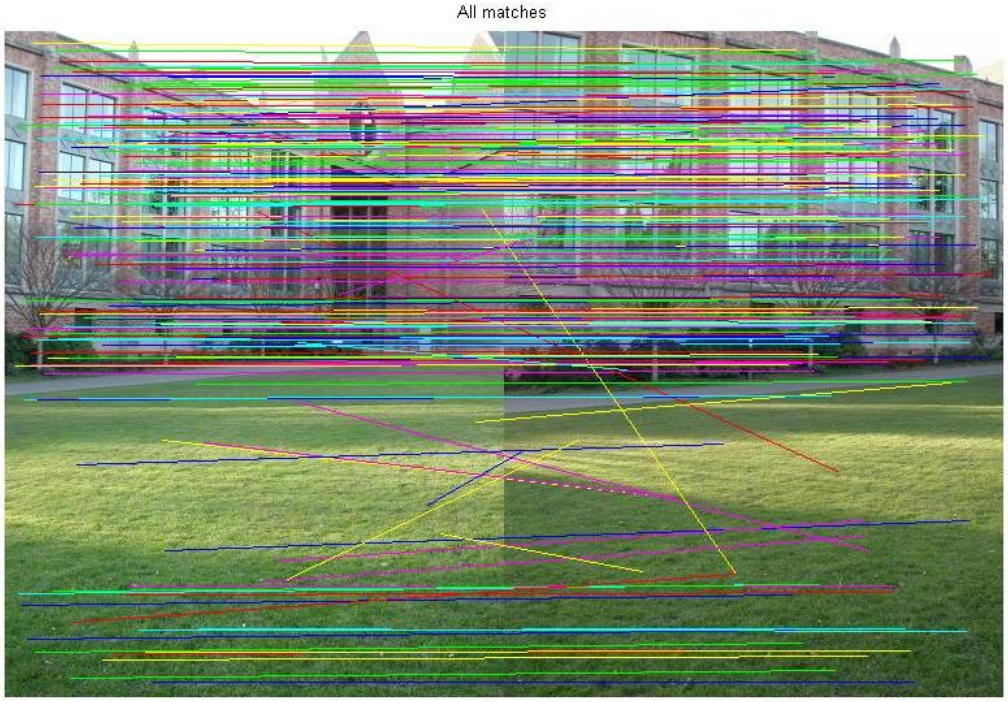


Figure 2: All the Matches that where found between the two Images, as we can see some of them are outliers - therefore we need the RANSAC scheme to get rid of them, so we don't get a distorted homography at the end.

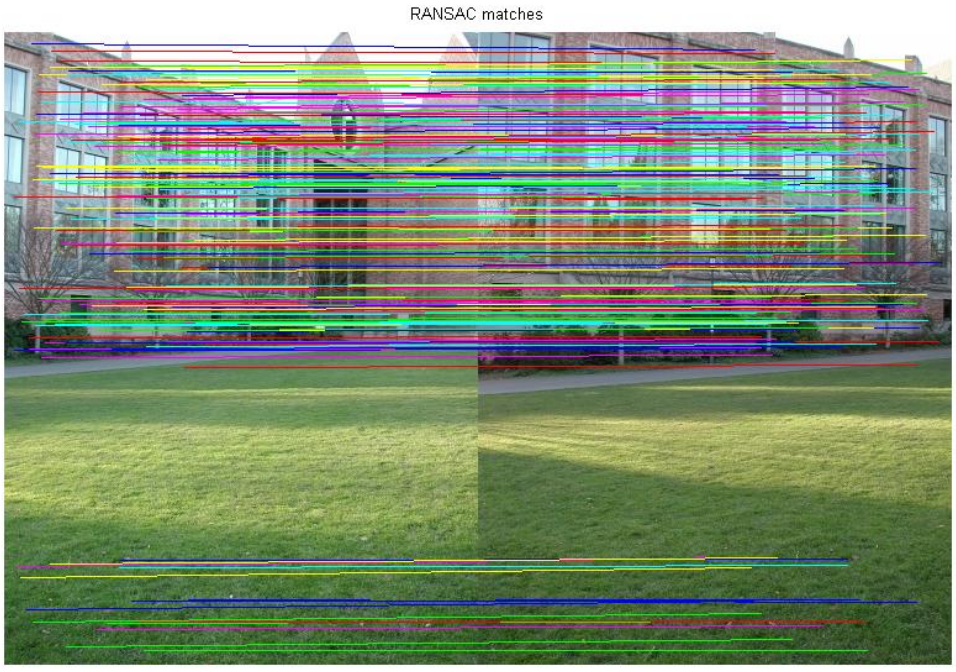


Figure 3: All with RANSAC method extracted inlier Matches from Figure 2

As we can see in Figure 2 not all of our found matches are essentially correct, some of them are considered to be *outliers* and if we would compute a homography including them we would get a distorted result. Therefore we apply the RANSAC scheme to receive a better estimation of the

homography between the image pair.[3]

We begin by selecting randomly 4 matched-points pairwise (see Figure 4) and estimate the homography between them by using the given function *cp2tform*() as follows.

```
[homography] = cp2tform(rndPnts1, rndPnts2, 'projective')
```

After acquiring the homography we transform all found points of interest and measure how many points are below a given threshold distance T using the euclidean distance as distance metric. This is done in the function tformAInliers() in starting at line 55 in the code below. If the number of inliers in the current iteration of the algorithm is greater than the previous best result then the best homography gets updated to the current one. After N iterations the inliers of the best matching homography are used to reestimate the homography (see Figure 3) which is then used to transform the image in step C).

```
1  function [ homography ] = PerfRANSAC( points1, points2 ,image1, image2, doPlot)
2
3  N = 1000;
4  bestInliersCnt = 0;
5  bestInliers = 0;
6  bestHomo = 0;
7
8
9  for(i=1:N)
10     randoms = randsample(size(points1, 1),4);
11
12     rndPnts1 = points1(randoms,:);
13     rndPnts2 = points2(randoms,:);
14
15     try
16         % b) - d)
17         [nrInliers, inliers, TFORM] = tformAInliers(points1, points2, rndPnts1, r
18
19         % if new calculation is better than old update result
20         if(nrInliers > bestInliersCnt)
21             disp(sprintf('Best match - #of inliers \%d', nrInliers));
22             bestInliersCnt = nrInliers;
23             bestInliers = inliers;
24             bestHomo = TFORM;
25         end
26
27     catch err
28         disp('Ouch!');
29     end
30 end
31
32 % 4) after N runs take best homography and reestimate with all points
33 % saving only the inliners from points1 and 2
34 m1 = zeros(bestInliersCnt,2);
35 m2 = zeros(bestInliersCnt,2);
36
37 pos = 1;
```

[3] See Matlab-file *PerfRANSAC.m*

```
38 for i = 1:size(bestInliers,1)
39     if (bestInliers(i) == 1)
40         m1(pos,:) = points1(i,:);
41         m2(pos,:) = points2(i,:);
42         pos = pos + 1;
43     end
44 end
45
46 % Output all inliers:
47 if (doPlot)
48     match_plot(im2double(image1{1,1}), im2double(image2{1,1}), m1, m2);
49     title('Matches of only Inliers!');
50 end
51 [~, ~, homography] = tformAInliers(points1, points2, m1, m2);
52
53 end
54
55 function [ nrInliers, inliers, homography ] = tformAInliers( points1, points2, rr
56
57 T = 5;
58
59 % b) estimate transformation ob rndPnts1 onto rndPnts2
60 homography = cp2tform(rndPnts1, rndPnts2, 'projective');
61
62 % c) transform the points
63 trnsfrmdPnts = tformfwd(homography, points1(:,1), points1(:,2));
64
65 % d) calc euclidean distance
66 distance = (trnsfrmdPnts - points2).^2; %one line sqrt(sum(.))
67 distance = sqrt(distance(:, 1) + distance(:, 2));
68
69 % Thresholding
70 inliers = distance<=T;
71 nrInliers = sum(inliers);
72
73 end
```

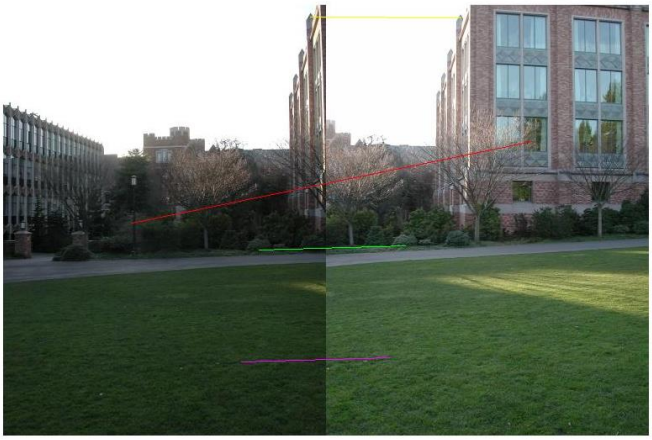


Figure 4: Select randomly 4 matched-points

C. Image Stitching

So after we determined the homographies between all the image pairs we select the image in the middle (in our case Image 3) to be our reference image and compute all homographies that map the other images to our reference:

$$H_{1,3} = H_{2,3} * H_{1,2} \qquad\qquad H_{5,3} = H_{4,3} * H_{5,4}$$

Before we can stitch our images into one panorama picture we need to compute the size the output image is going to have. For this information we can take the method *imtransform* with our new homographies and as a result we get the new size (xData, yData) of the transformed image, which is saved in our case for all 4 images (we don't need to compute the reference image, because we know it is going to be in the middle of the panorama picture) in an array:

```
%% saves the dimensions of all the transformed images
xLim = zeros(4,2);
yLim = zeros(4,2);

[transImage, xLim(1,:), yLim(1,:)] = imtransform(Images{1}, H_1_3);
...

%% Getting the Coordinates for final Image
xMin = min(min(xLim));
xMax = max(max(xLim));

yMin = min(min(yLim));
yMax = max(max(yLim));

%% Width and Height of panorama image.
width  = round(xMax - xMin);
height = round(yMax - yMin);
```

Now that we know the output dimensions of our final panorama image we can transform all of our images into that particular size by using again the *imtransform* method but this time adding our appropriate dimensions [xMin xMax] and [yMin yMax] as our XData and YData parameters. Given all transformed images, the final step is to blend them together in a way to avoid seams. To do this we use an α channel where the value of α for an image is 0 at all

border pixels and 1 at the maximum distance from the border. The remaining values are linearly interpolated (see Figure 5) and transformed with their respective homography to the panoramic image. For Example our interpolation value α would look like Figure 6 for all our 5 images. The final color at a pixel in the output image is computed as the α-weighted sum of overlapping images. More precisely, if there are n images overlapping at pixel position $(x, y), I_i(x, y), i = 1...n$ with color (R_i, G_i, B_i) and weighting factors α_i, the color values in the stitched output image O are computed as follows:

$$O(x, y) = \frac{\sum_{i=1}^{n} (R_i, G_i, B_i) * \alpha_i}{\sum_{i=1}^{n} \alpha_i}$$

This blending method is called *feathering*.

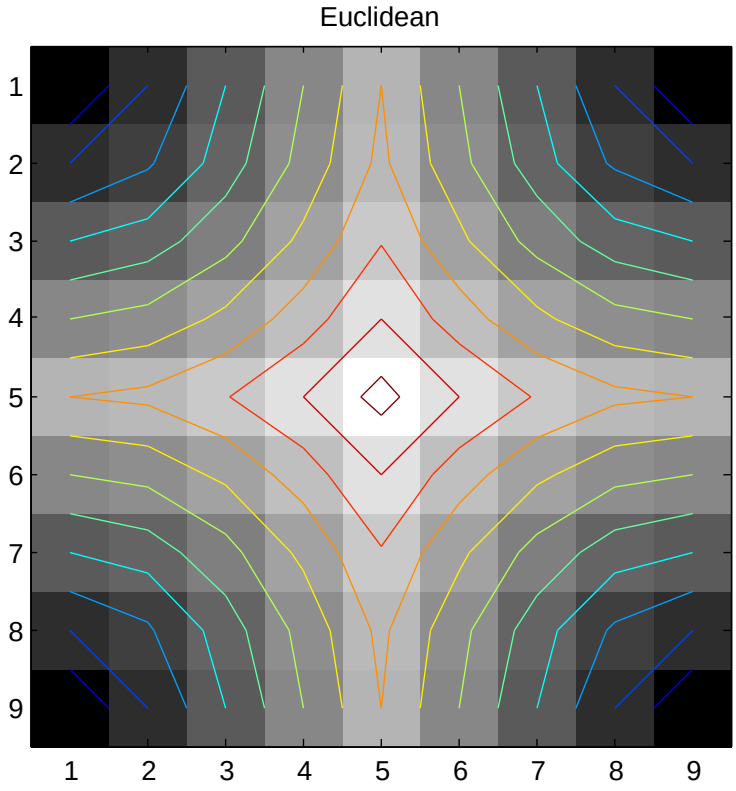


Figure 5: To compute our α channel we used the method *bwdist* with the euclidean distance method

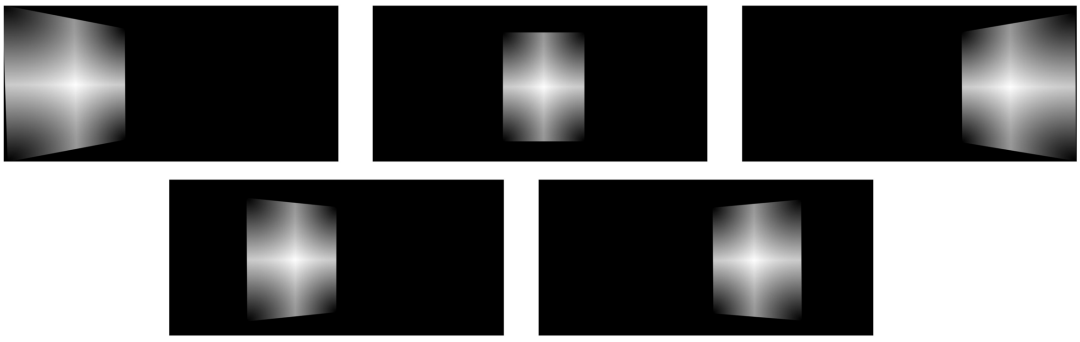


Figure 6: α channel the interpolation of the 5 images to the resulting panorama picture

Results Our first set of images $campus1...5.jpg$ with $feathering$:

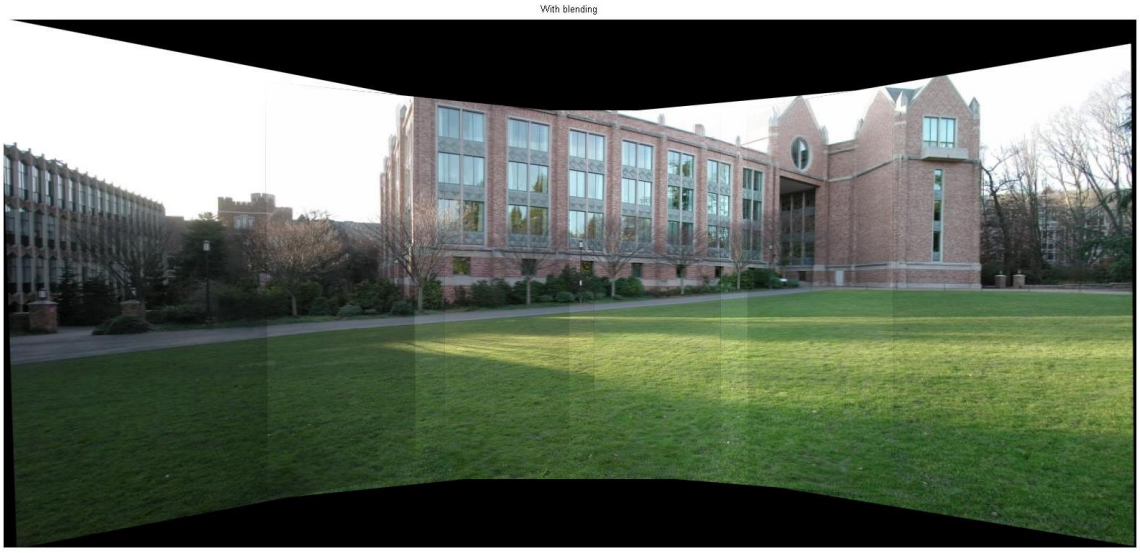


Figure 7: Sample Set campus{1...5} with feathering

And without $feathering$ (the transformed images are laid over each other from left to right):

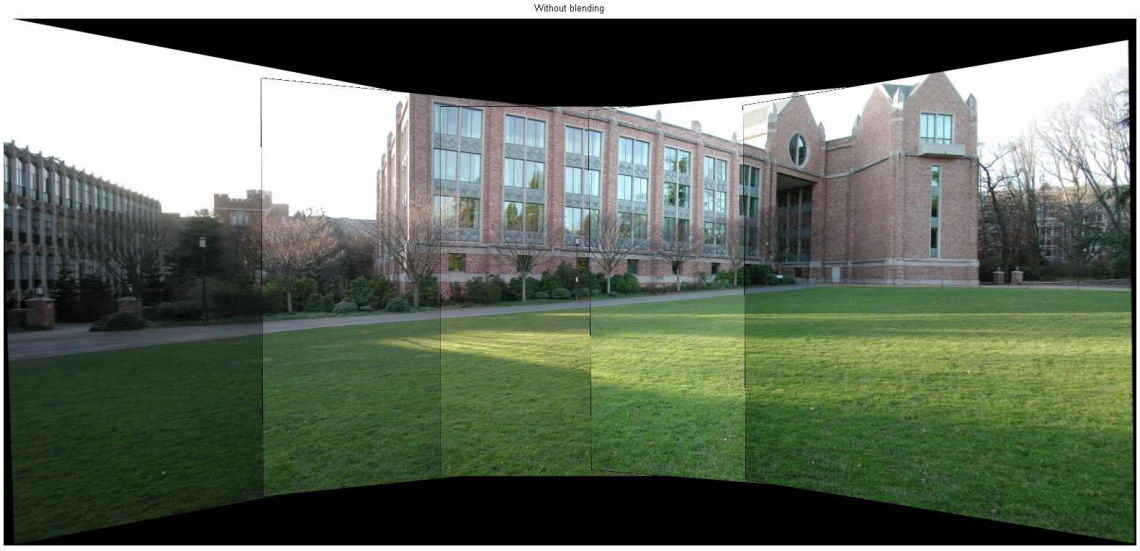


Figure 8: Sample Set campus{1...5} without feathering

Our second set of images $officeview1...5.jpg$ with $feathering$:

Figure 9: Sample Set officeview{1...5} with feathering

And without *feathering* (the transformed images are laid over each other from left to right):

Figure 10: Sample Set officeview{1...5} without feathering

Figure 11: First Set of our pictures {1...5} with feathering

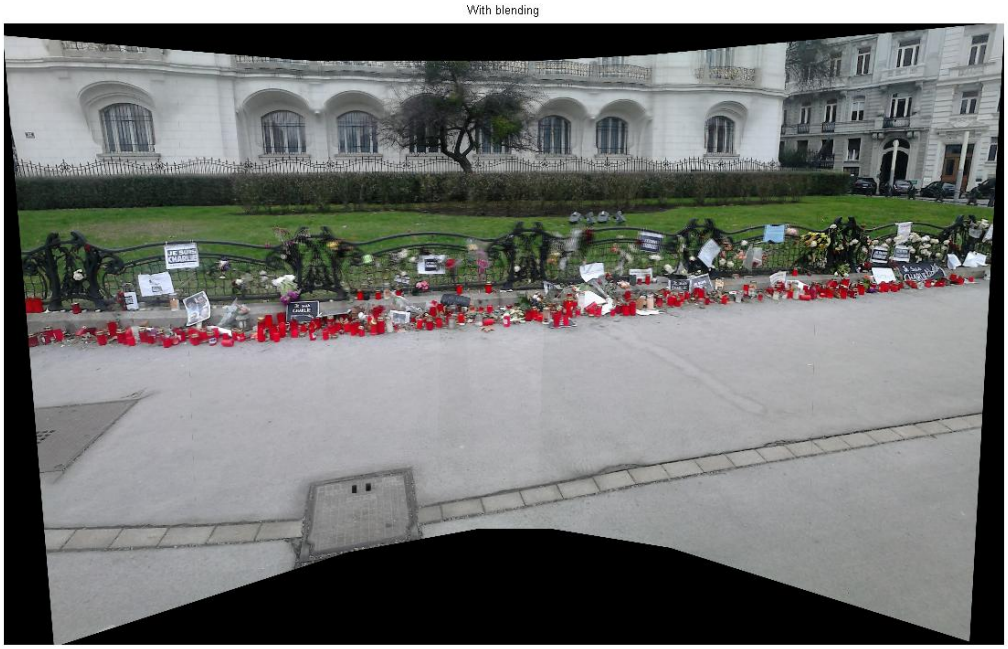


Figure 12: Second Set of our pictures {1...5} with feathering

Especially looking at the first and the first self-made image (see Figure 7 11) we see that some of the images could not be overlaid correctly, which means that there still were some small outliers which distorted the homography so that the final image is at some parts a little blurry. With the second sample and the self-made images (see Figure 9 11 12) we detect the *feathering* effect much better than for the first image because the shading through the image is continuous and no seams are visible from overlapping one image after another.

Assignment 5: Scene Recognition with Bag of Visual Words

In this assignment we want to classify images into scene categories like bedroom, kitchen etc. We use the standard bag of visual words model to achieve this task. Subsection A. presents the implementation of the category recognition and Subsection B. presents our results.

A. Methodology

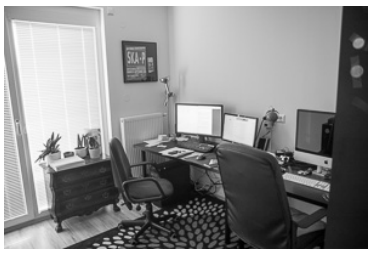
First the test- and the training-set gets loaded and for every image its category is saved. The next step is to build a vocabulary of visual words. We will form this vocabulary by sampling many local features from our training set and then clustering them with K-means. The number of K-means clusters `num_clusters` is the size of our vocabulary. We used 50 clusters which means that the 128 dimensional SIFT feature space is partitioned into 50 regions.
Every time a new SIFT feature gets observed, the cluster it belongs to can easily get figured out by getting the nearest centroid of our original clusters. Those clusters are our visual word vocabulary.
To extract these SIFT features from the images in the training set to collect them for k-means clustering, 100 SIFT features get extracted from every image. These features get added to the feature space which is $128 \times$ size of training set $*$ number of features per image. In out example the training set contains 800 images and the number of features per image is also 100, that means the feature space is 128×80000 big. After that the 50 clusters get determined and stored in a matrix.
The next step is to build a feature representation for every image in the training set that can be used for the classification of new images later on. An image is represented by the normalized histogram of visual words, which means that all SIFT features of an image are assigned to visual words and the number of occurrences of every word is counted. Therefor we have to get the SIFT features for every image from the training set again, but this time it gets sampled more densely than before. We used a step size of 2. Then we determine the nearest clusters of the sampled SIFT-features and count the number of occurrences. This histogram represents the bag of words for this image.
The last step is to classify all the images of the test set to investigate the classification power of the bag of visual words model for our classification task. This is obviously quite similar to the previous step. (Get SIFT features densely sampled (step size is 2) and count number of nearest clusters to this features as bag of words) But this time the visual word histogram of an image is used for actual classifying it by means with the nearest neighbor method (we used a k of 3) with the previously learned training features and class labels. The result is stored in a confusion matrix. (See Figure 15)

B. Results

Classification Rate Figure 15 shows the classification results of the given test set in form of a confusion matrix. A cell in this matrix describes how many of the test images of the category of the row (e.g. the first row is the category 'bedroom') got classified as the category of the column. For example 92% of the test images of forests got classified as forests.
In general our scene classification results are all higher than 35% which is really bad. But this is only the case because pictures of bedrooms and living rooms are very similar and it is really hard to distinguish them. Figure 19 shows two pictures (one living room and one bedroom) which contain many similarities. For example both rooms contain: Lamps, a television, a bed/ a sofa, windows and other furniture.
Also pictures of kitchens got confused as livingrooms (16%) and offices (13%). And stores got

(a) Bedroom

(b) Livingroom



(c) Office

(d) Kitchen

(e) Forest

Figure 13: Own scene pictures

wrongly detected as livingrooms (21%) and streets (9%). Figure 20 shows that in contrast it is very easy to distinguish between a living room and a forest or a mountain, because nearly no features appear in both images. Only through the windows of the rooms or pictures on the walls it is possible to see nature in human environments. This is also reflected in the classification rate of the forest images which is at 92%. We

Own Test Images Figure 13 shows our own test images. We took 5 images from 5 different categories and tried to categorized them with our scene recognizer. Figure 14 shows the results of the scene recognition. As you can see only the picture of the bedroom and of the forest could get recognized correctly, the other 3 images were recognized wrong as bedroom.
This is also consistent with the above findings.

Parameters

- We tried to categorize the pictures with a `step size` of 1, which means the double amount of SIFT features per image. Figure 16 shows that the results were worse than with the step size of 2.
- We tried to change the `k` of the `knnclassify` method and we changed it from 3 to 8 which helped us to improve the results for the given training and test set for every category significantly except of the living room and the street category. (See Figure 17)
- And we looked what happens if we enhance the size of the bag of words to 100 clusters instead of 50 with k=8 (see above) and the results were worse for the recognition of the bedroom, but much better for complicated structures that repeat itself very often like in the category for the store or the office. (See Figure 18) By reducing the k back to 3 the classification results get slightly worse but the classification rate for the bedroom gets to 43% which is the best result.

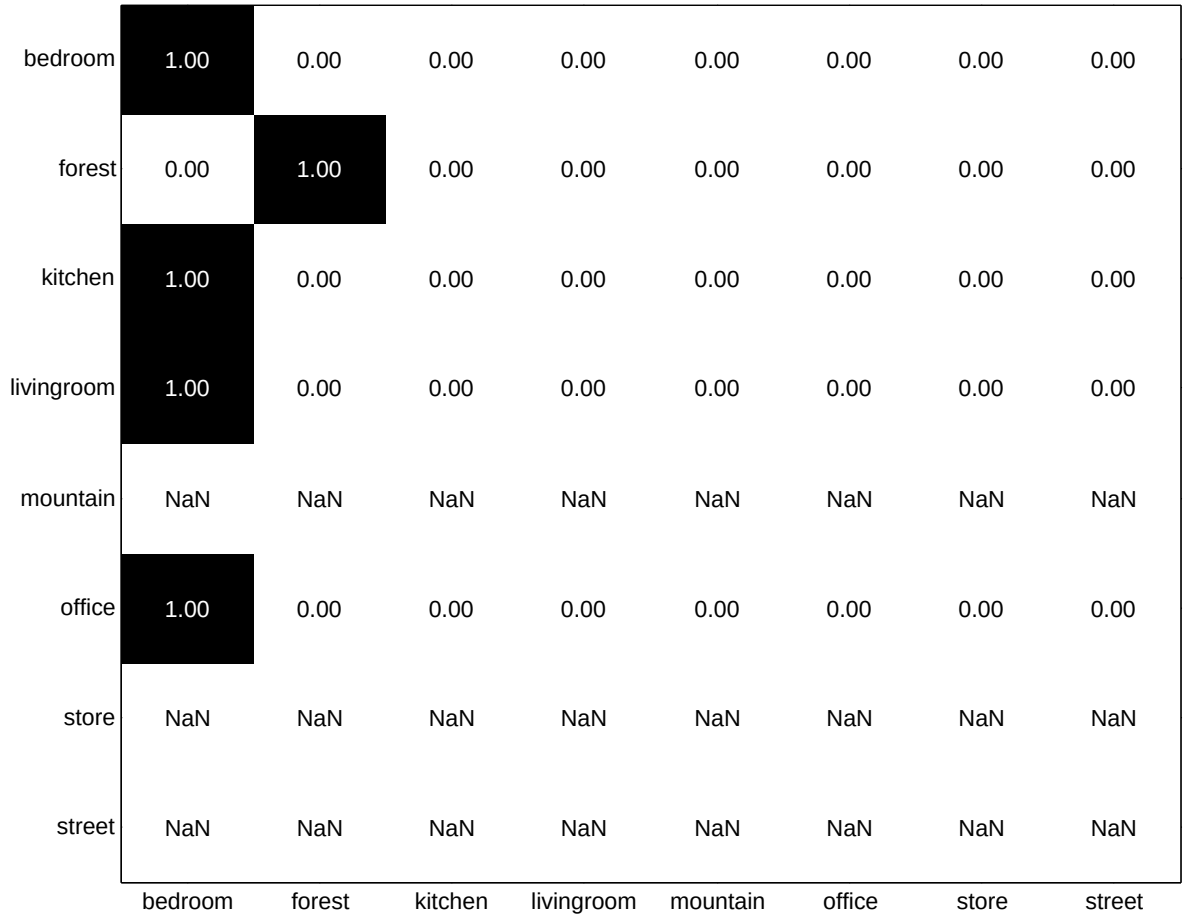


Figure 14: classification results in percentage of the own pictures in form of a confusion matrix

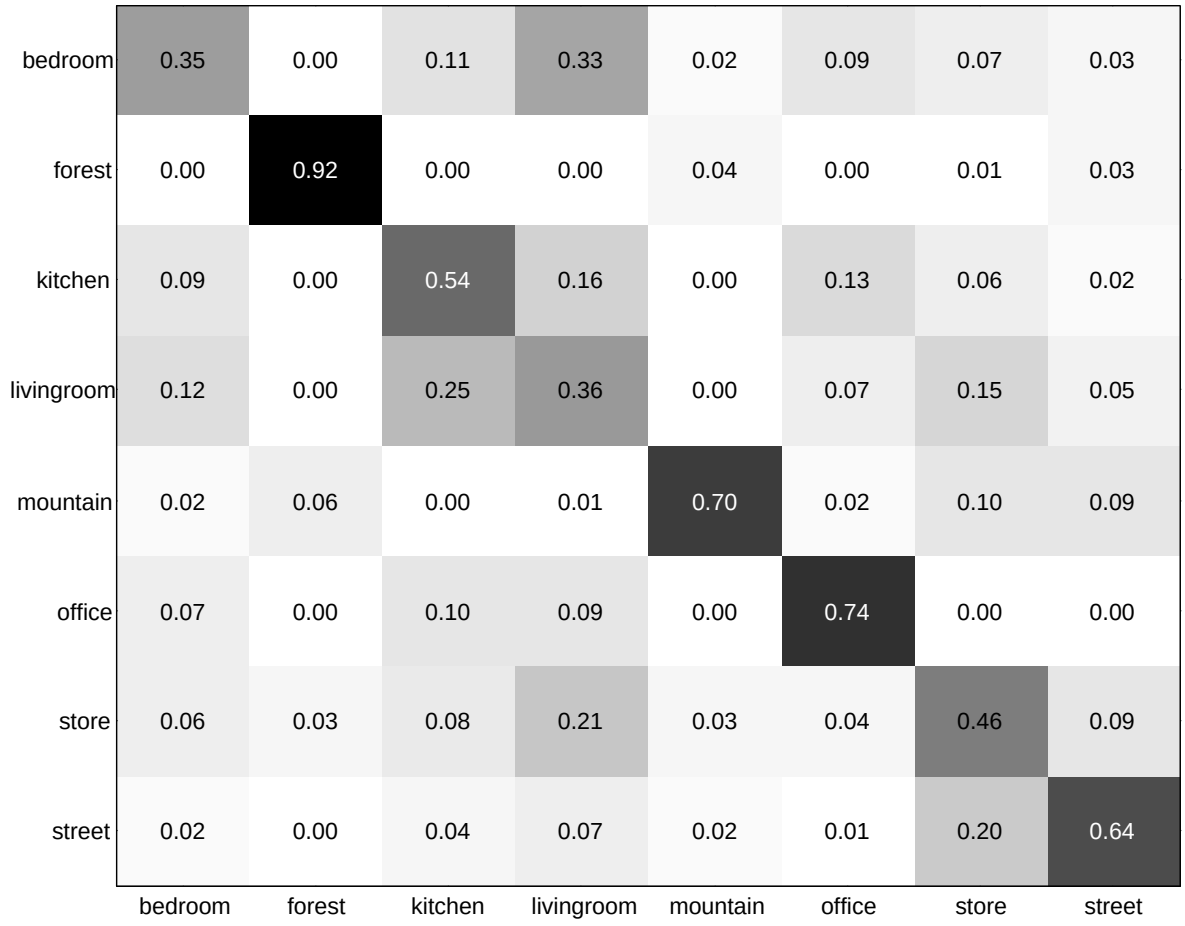


Figure 15: classification results in percentage of the given test set in form of a confusion matrix

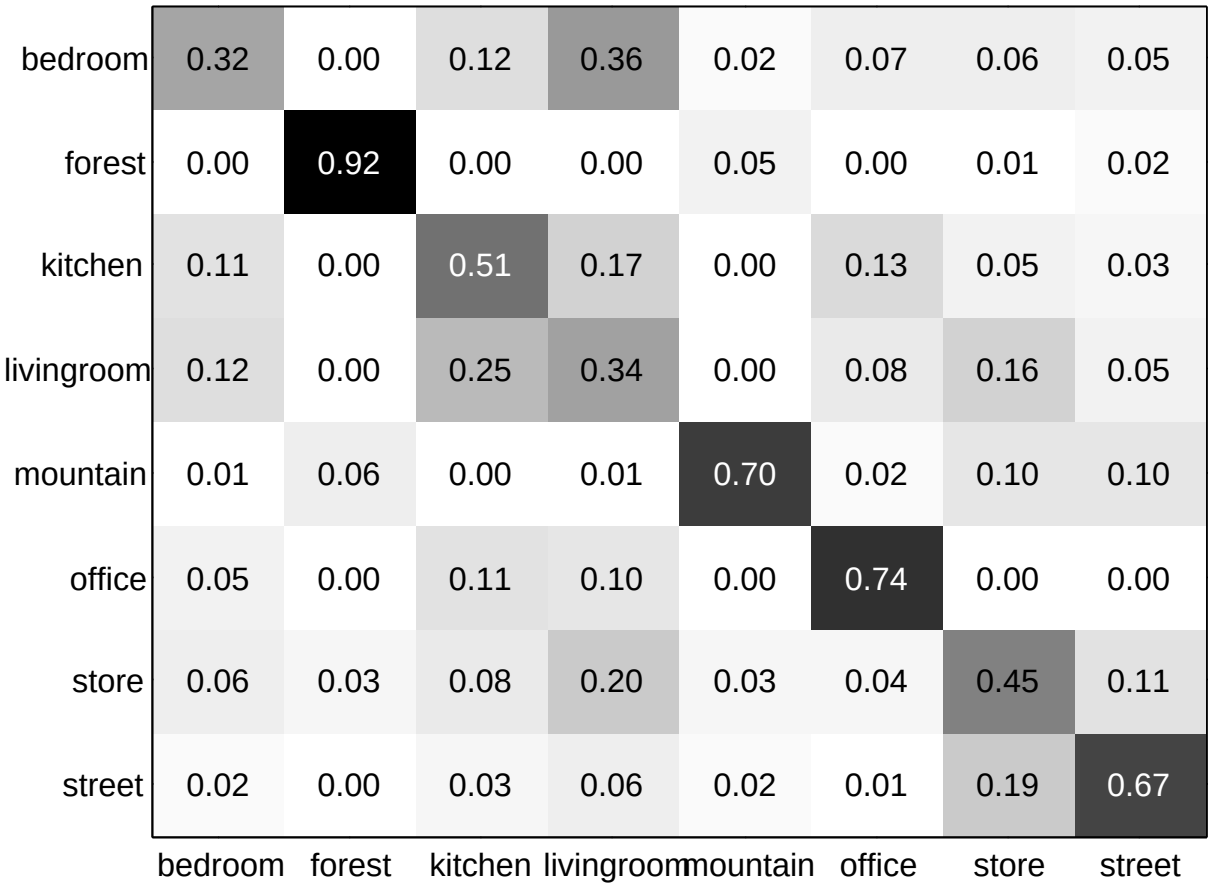


Figure 16: classification results in percentage of the given test set in form of a confusion matrix with step size = 1

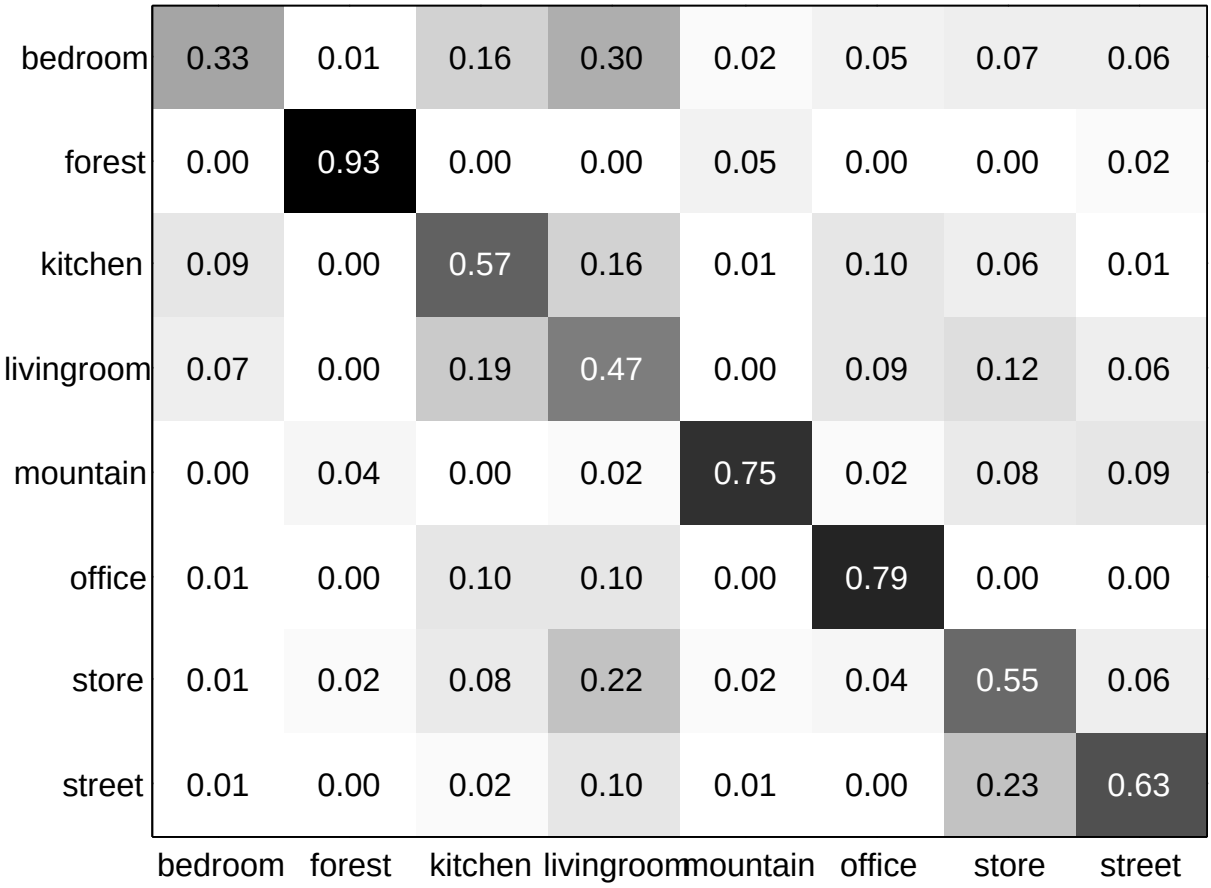


Figure 17: classification results in percentage of the given test set in form of a confusion matrix with a knn classification of k = 8

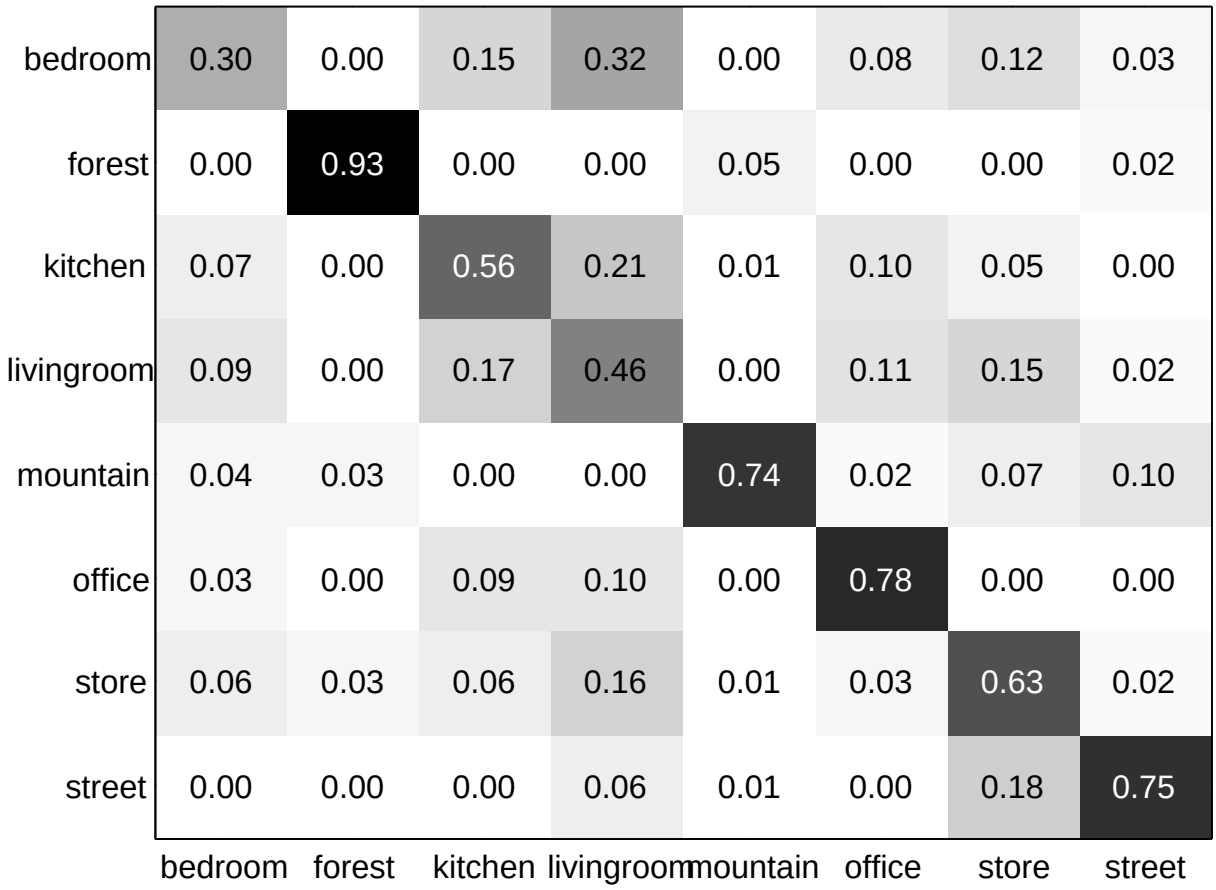


Figure 18: classification results in percentage of the given test set in form of a confusion matrix with a knn classification of k = 8 and 100 clusters

(a) Bedroom (b) Livingroom

Figure 19: Bedrooms and livingrooms are hard to distinguish and therefore to detect

(a) Forest (b) Livingroom

Figure 20: Forests and livingrooms are easy to distinguish and therefore to detect