VU 183.585

Computer Vision

Exercise Course: Assignments II

WS 2016/17

Sebastian Zambanini

Computer Vision Lab
Institute of Computer Aided Automation zamba@caa.tuwien.ac.at

Preliminary Note

For these assignments you will need the VLFeat open source library which can be downloaded from http://www.vlfeat.org/download/vlfeat-0.9.20-bin.tar.gz. In order to get the library to work you have to unpack the files into a folder and start the function `vl_setup.m` in the `toolbox` subfolder. This should instantly work for Windows, Mac and Linux (i.e. also in the lab environment), as the toolbox contains precompiled versions of all functions for these platforms. Additionally, you will need the Matlab function `match_plot.m` which can be downloaded from the TUWEL course site.

Assignment 4: Image Stitching (14 Points)

In this assignment, we will go beyond the detection of interest points (as in assignment 3) and will explore the matching of such points for the purpose of image stitching, the process of combining multiple images with overlapping fields of view to produce a panoramic image.
First of all, if you are not yet familiar with the following topics, please read them up in the textbook [Sze10]:

- Scale Invariant Feature Transform (SIFT)[Low04]: pages 217-219 and 223-224.
- RANDOM SAmple Consensus (RANSAC): pages 318-319.
- Homography: pages 37 and 56.

The goal of this assignment is to match all images of a photographic sequence to a reference image. If only the rotation around the vertical axis changes and the viewpoint is constant the transformation between the images can be described by a homography. If two images have an overlapping field of view, one can match and align them as follows:

1. Find interest points in both images.
2. Describe the local appearance of interest points.
3. Get a set of putative matches between the local descriptors of the two images.
4. Perform RANSAC to estimate the correct transformation in the presence of false matches.

There are two image sequences provided for this assignment:

- `officeview1.png...officeview5.png`: images taken out from my office's window.
- `campus1.png...campus5.png`: images from the web.

The images are numbered from left to right. First, try to make your algorithm work on these image sequences. Afterwards, make your own image sequence of at least four images, apply the algorithm to it and show the results in your report. You can use your mobile phone or any other camera for this task, but pay attention to not move the camera too much between shots.
We now approach the goal of building a basic image stitching system in a step-by-step manner:

A. SIFT Interest Point Detection

Before doing the overall image stitching, you should try to extract SIFT features out of one of the images. Convert the image to grayscale, use the function `vl_sift` for feature extraction and overlay the features on the image using `vl_plotframe`.

Issues to be addressed in the report:

- Show the output of `vl_plotframe` on the chosen image. What is the meaning of the size of the drawn circles and lines inside the circles?

B. Interest Point Matching and Image Registration

In order to learn how to match interest points and align an image pair, perform the following steps on two consecutive images from one of the image sequences.

1. Convert the first and the second image to grayscale and extract the SIFT keypoints and descriptors using `vl_sift`.
2. Match the descriptors using `vl_ubcmatch` and plot the matches using `match_plot`.
3. You will recognize that not all matches are correct, i.e. some matches can be considered as *outliers*. Therefore, apply the RANSAC scheme to estimate the homography between the first and the second image. For this purpose, perform the following steps N times (e.g. $N = 1000$):
   (a) Randomly choose four matches, i.e. four points of the first image and the corresponding four points of the second image. The function `randsample` might be helpful.
   (b) Estimate the homography between these points using `cp2tform`. Choose `'projective'` as transformation type. Attention: `cp2tform` will throw an error if the four point pairs are not chosen properly (e.g. three of them lie on the same line). The easiest way to handle this is to use a `try-catch`-block when calling `cp2tform`.
   (c) Transform all other points of putative matches in the first image using `tformfwd`.
   (d) Determine the number of *inliers*: compute the Euclidean distance between the transformed points of the first image and the corresponding points of the second image and count a match as inlier, if the distance is under a certain threshold T (e.g. $T = 5$).
4. After the N runs, take the homography that had the maximum number of inliers. Re-estimate the homography with all inliers to obtain a more accurate result.
5. Transform the first image onto the second image. For this purpose, use the function `imtransform` and specify the arguments `'Xdata'`, `'Ydata'` and `'XYScale'` to get the same dimension as the second image.

Issues to be addressed in the report:

- Plot the matches after step 2. Describe in detail what `vl_ubcmatch` does.
- Plot the matches of the inliers after step 4. What is the difference to the set of all putative matches we plotted before?
- Apart from small errors, the images should be aligned after step 5. Demonstrate that by showing the absolute differences between the two images.
- Examine if the presented scheme of SIFT interest point detection and RANSAC-based alignment is invariant to changes in image rotation and scale. Thus, likewise to assignment 3, resize and rotate the second image and repeat the alignment procedure.

C. Image Stitching

The image registration method built in the last step represents the building block for our image stitcher. We can now align overlapping pairs of images and consequently register all images to a reference image. Comprise the following steps to stitch all images of a sequence together:

1. Determine the homographies between all image pairs from left to right, i.e. align image 1 with image 2 ($H_{1,2}$), image 2 with image 3 ($H_{2,3}$), and so on, by using the method described in step B.
2. Choose a reference image (usually the center image, e.g. image 3) and compute all homographies that map the other images to the reference images. If you have more than three images in your sequence you have to create composite homographies to map an image over a sequence of consecutive images to the reference image. For instance, if image 3 is the reference image, you can construct the homography $H_{1,3}$ that maps image 1 to the coordinate frame of image 3 by $H_{1,3} = H_{2,3} \cdot H_{1,2}$. The homography matrix of a projective `TFORM`-struct `H` can be accessed via `H.tdata.T`. The inverse homography can be accessed via `H.tdata.Tinv`.
3. Before transforming each of the images, first compute the size of the output panorama image by determining the range of transformed image coordinates for each input image. Do this by transforming the four corners of each input image to the reference coordinate frame to determine its coordinates in the output image. Then compute the minimum x, minimum y, maximum x, and maximum y coordinates to determine the size of the output image.
4. Transform all images to the plane defined by the reference image by using the function `imtransform` and setting `'Xdata'` and `'Ydata'` to the appropriate output dimensions determined before.
5. Given all transformed images, the final step is to blend overlapping pixel color values in such a way as to avoid seams. One simple way to do this, called *feathering*, is to use weighted averaging of the color values to blend overlapping pixels. To do this use an α-channel where the value of α for an image is 0 at all the border pixels and 1 at the

maximum distance from the border. The remaining values are linearly interpolated (the function `bwdist` can be used for this task). These α-images are also transformed to the reference coordinate system and 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), $I_i(x, y)$, $i = 1 \dots n$ with color values (R_i, G_i, B_i) and weighting factors α_i, the color values in the stitched output image O are computed as:

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

Issues to be addressed in the report:

- Show and discuss the achieved results (with the two provided image sequences and your own sequence). The result might look quite realistic at a first glance but can you spot any errors by looking on details?
- Compare a result achieved with feathering to a result where no blending has been performed (i.e. the color values of only one image are taken for the stitched image). What is the difference of the two results?

Assignment 5: Scene Recognition with Bag of Visual Words (7 Points)

In this assignment we will again use SIFT features, however this time for a completely different purpose. We will use them for the task of scene recognition, which means that 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.

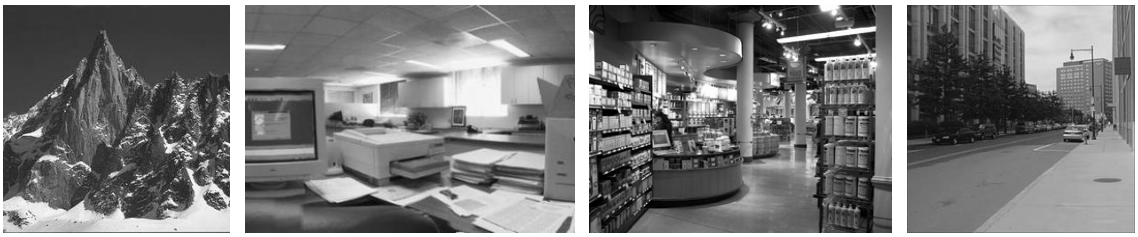
Bag of words models are a popular technique for image classification inspired by models used in natural language processing. An image is basically described by the distribution of its small local structures, where all local structures of an image are assigned to a visual "word" based on a predefined "vocabulary". The model ignores the word arrangement (spatial information in the image) and describes an image by the histogram of the frequency of visual words. The visual word vocabulary is established by clustering a large corpus of local features. Please see Section 14.4.1 in [Sze10] for more details on category recognition with quantized features. In addition, Section 14.3.2 discusses vocabulary creation.

Our database for scene recognition consists of eight categories with 100 training and 100 test images per category (see Figure 1).

For this assignment you should write the three functions `BuildVocabulary.m`, `BuildKNN.m`, `ClassifyImages.m` as well as a script `main.m` that performs the three basic steps by calling these functions:

- `C = BuildVocabulary(folder, num_clusters)`: The first step is to build a vocabulary of visual words. We will form this vocabulary by sampling many local features from our

(a) Bedroom (b) Forest (c) Kitchen (d) Living Room



(e) Mountain (f) Office (g) Store (h) Street

Figure 1: The eight categories of the scene recognition database.

training set (i.e. 100's of thousands of features) and then clustering them with K-means. The number of K-means clusters `num_clusters` is the size of our vocabulary. For example, if `num_clusters=50` then the 128 dimensional SIFT feature space is partitioned into 50 regions. For any new SIFT feature we observe, we can figure out which cluster it belongs to as long as we save the centroids of our original clusters. Those clusters are our visual word vocabulary.

In contrast to the previous assignment, we do not extract SIFT features at detected keypoints but rather densely sample them on a regular grid in the image. Therefore, the function should extract dense SIFT features from all the images contained in the eight category subfolders of the general training folder (argument `folder`) and collect them for K-means clustering. For feature extraction we use the function `vl_dsift`. Please note that we do not necessarily need to extract a SIFT feature for every pixel for vocabulary creation, e.g. around 100 features per image are enough to "capture" the approximately correct distribution of SIFT features for the given image data. Hence, use the parameters `'Step'` and `'Fast'` for `vl_dsift` and optionally select only a random subset of features per image for the overall set (the function `randsample` might be helpful). To iterate all images, the function `dir` can be used.

After all SIFT features have been collected, you should apply K-means clustering to find the visual words. Instead of the Matlab function or your own K-means function from Assignment 2, `vl_kmeans` can be used here for performance reasons. The words are finally stored in the matrix `C` of size $128 \times$ `num_clusters`.

- `[training, group] = BuildKNN(folder,C)`: 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 num-

ber of occurrences of every word is counted. The vector (histogram) is normalized to unit length to account for changing image resolutions. As we will use the Matlab function `knnclassify`, the output of `BuildKNN` should be a matrix `training` of feature points, where the rows represent the 800 images of the training set, as well as the vector `group` that indicates the class labels of the 800 images. For convenience, one can simply take the directory index provided by the function `dir` for class labeling.
In this step the SIFT features should be more densely sampled than before, thus the step size should be 1 or 2. In order to build a histogram of SIFT words the functions `knnsearch` and `histc` can be used.

- `conf_matrix = ClassifyImages(folder,C,training,group)`: 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. The function is similar to `BuildKNN` but this time the visual word histogram of an image is used for actually classifying it by means of the Matlab function `knnclassify` (e.g. $k = 3$) and the previously learned `training` features and class labels `group`. The final result is stored in the confusion matrix `conf_matrix` whose elements at position (i, j) indicate how often an image with class label i is classified to the class with label j.

Issues to be addressed in the report:

- In general, this approach should achieve a classification rate of around 60%. Show and discuss your results in the report. Show the confusion matrix and describe if there are classes which can be more easily identified than others. Additionally, describe if there are class pairs which are confused with one another more often than other pairs.

- Take some own test images and investigate if they can be correctly classified by your simple scene recognizer. For instance, make a photo of your own kitchen or living room and classify it. You should at least test one own photo from three of the eight categories.

References

[Low04] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.