Universität
Augsburg
University

Evaluating the Robustness of Local Image Features

Bachelorarbeit

zur Erlangung des akademischen Grades

"Bachelor of Science"

im Studiengang Informatik und Multimedia
am Institut für Informatik

Lehrstuhl Multimedia Computing

Eingereicht bei:	Prof. Dr. Rainer Lienhart
Betreuer:	Dipl.-Inf. Fabian Richter
Vorgelegt von:	To Quyen Hoang
Geboren am:	17.09.1988
Matrikelnummer:	1027422
E-Mail:	to_quyen_hoang{at}yahoo.de

Contents

Abstract

In recent years SIFT features have become one of the standard tools in many applications in computer vision. Many variants and improvements are developed with much effort to get SIFT features with more robustness and distinctiveness, while scaling down complexity. For object recognition, one reason for this developments is that there are still many features that are detected in uninteresting image regions such as the background, that do not help recognize the object on the image. As a result removing these noisy features would reduce the number of mismatches, the allocated memory space for keypoints, and the overall number of detected keypoints, which could result in a faster and more robust matching.

In this thesis we examine a feature refinement technique that is based scaling and rotating images prior to the features extraction. The main idea is that stable features of an image are not affected by affine transformations of the image. As a consequence simulating these transformations and reducing the number of keypoints to the number of those keypoints that survive these simulations, results in a set of only stable keypoints for the image. To obtain the more stable keypoints, we first generate two new images from an image by transforming it in two ways and with different parameters. Then an feature extraction of these three images and a mapping of the features back to the approximate location in the original image give us two feature maps and a set of keypoints of the original image. With these feature maps we then determine which feature is stable enough to withstand these affine transformations, and which is not. The refined keypoints are then tested with two matching strategies. The first one is the nearest neighbor ratio approach, the second one the BoW approach. Both are tested in experiments within a Near-duplicate Image Retrieval, in which we evaluate the effectiveness of our approach with different scale and rotation parameters. Compared to the classical SIFT respectively SURF features, the first shows a significant rise in stability as well as faster matching due to a notable reduction of keypoints, whereas the second one shows that only a little loss in precision has to be taken, for a reduction of memory storage for feature points.

1 Introduction

Finding correspondences between objects of two different images has been an important issue in computer vision applications, such as pose estimation, image registration or object recognition. Especially the latter has become more and more important for the fast development of Internet applications, such as multimedia sharing websites. Users of websites such as Facebook, Flickr, or Youtube add images and videos very rapidly, pushing the amount of images and videos available in the Internet to an enormous size. As a consequence we require an efficient multimedia information retrieval method, with attributes like saving memory space, faster matching, or reliable features.

For recognizing images with a specific object in a database, feature-based methods have become more and more common. On this task the Scale Invariant Feature Transform (SIFT) proposed by Lowe [1] has been successfully applied as SIFT features are invariant to scaling, rotation, translation and at least partially to illumination changes, affine and 3D projections. In this thesis we want to especially make use of the scaling and rotation invariants of SIFT. We propose a variant of SIFT that has less, but more reliable features, that we get by transforming the image.

Many variants and extensions claim to improve the performance of SIFT features. Some of them try to improve the matching strategy with a visual vocabulary tree [2] or partitioning the keypoints set into sub-sets for each octave [3]. VF-SIFT [4] extends the SIFT features by several pairwise independent angles to cluster them after the extraction in dependency of their introduced additional angles.

Others try to improve SIFT by making a feature more informative, either by adding some information or subtract features that are in some kind of definition less informative. To mention a few, there is for example the approach of Dense Interest Points [5] which is an hybrid scheme that combines interest points such as SIFT with dense sampling. It takes the advantages of both, the high repeatability of interest points and the good object coverage of the dense sampling grid, to lower the disadvantage of each of them. Thus although some patches of the dense sampling grid do not contain accurately matching keypoints, they are still kept, because they could give some more information to interpret the whole scene of an image. Another approach, ASIFT [6] tries to extend the affine invariance of SIFT with two affine invariants for the angles that define the camera axis orientation. This is done by simulating these two angles, leading to a fully affine invariant SIFT. BSIFT [7] adds an additional background invariance through an anisotropic smoothing instead of the Gaussian smoothing at the feature detection step. Thus we get an object-boundary-respecting Gaussian scale-space pyramid at the detection step, helping us to detect the object with different backgrounds.

On the other hand, Turcot and Lowe [8] propose a way to improve SIFT for object recognition not by adding some attributes to the existing keypoints but by reducing the number of keypoints by selecting a set of useful features that have more information than the remaining ones. Briefly explained, a useful feature is in this context a feature, which is robust enough for a corresponding feature in the same object to match with it, stable enough that it can exist in multiple viewpoints and distinctive enough that, in a matching with the Bag-of-word approach from Nistér and Stewénius [2], the corresponding features refer to the same visual word. Not useful features are then removed. The result is a small set of keypoints, which results then in a reduction of memory requirements, which is more suitable for large image database problems. In a few words, the main idea of finding useful features is that contrary to useless ones that occur in only one single image, useful features can exist on more than one image of the same object or location and are geometrically consistent to one another. Thus these useful features can be found by seeking for those keypoints that are on images with the same object shown in a different view and determining geometrically consistent ones through BoW [2] and RANSAC [9].

In this work we investigate an approach proposed by Zhang *et al.* [10]. They proposed a way to refine SIFT keypoints to a subset of, in their context, more stable ones by scaling the image. They notice that many local features of an image are not stable and informative enough, as they are for example detected in the, for object recognition, uninteresting background. Moreover scaling an image obviously varies the number of keypoints of that image. The idea is then that some unstable features cannot withstand these transformations. Thus they scale an image in the horizontal and vertical direction, generating new images, which are then used to determine a set of stable features that withstand affine transformations, and then finding contextual visual words containing these features.

As our focus is on the results of the refinement itself, our approach differs from Zhangs approach as we only refine the features, but do not use contextual visual words like Zhang *et al.* do it in [10]. Instead we do a matching with a method proposed by Lowe, that we refer to by the nearest neighbor ratio [1] throughout the rest of this work and a BoW method likely to that of Nistér and Stewénius [2]. In addition we do not only scale the image but also rotate it, as SIFT features are also invariant to rotation, and furthermore test the technique with SURF(speeded-up robust features) [11], as SURF features have the same scale and rotation invariance as SIFT. They differ from SIFT mainly, because they use integral images and a Hessian blob detector instead of SIFTs difference-of-Gaussian(DoG) detector.

The thesis presents our method for extracting these refined features and which improvements it can have for which matching strategy. Section 2 gives a brief definition of stability and the refinement technique. The affine transformations and the refinement with the transformed image are introduced. Section 3 provides our experiments and their evaluations. In that section we present our dataset for the experiments, the results for the amount of remaining features, how the scale of a features is correlated with its likelihood to be removed in the refinement, and our approach in a near duplicate retrieval first with the nearest-neighbor ratio matching strategy and then with a BoW method.

2 Local Feature Refinement

In this section we describe how we refine features with affine transformations. First we briefly define what stability of a keypoint means in our context, giving a general idea of our approach. After that we discuss the refining technique, and the two affine transformations, scaling and rotation, that delete a certain number of less stable keypoints. To be sure what one simple kind of affine transformation can change, we do not mix these affine transformations, and only use simple ones such as scaling along one axis. The reduction of the number of keypoints results in a speed up of the execution time for a matching with any matching technique, whose execution time depends on the number of features per image, such as Lowes simple nearest-neighbor ratio matching strategy [1]. For these and other matching techniques the more stable keypoints result in less mismatches. With less features also comes a saving in memory space, which can be important for applications with a large image database and a limit in memory space.

2.1 Stability of Keypoints

Before we can discuss our technique of feature refinement and its improvements, we have to define what "stable keypoints" means in this thesis. According to Lowe [1] SIFT features are invariant to scaling and rotation, thus the local features should not be affected by these affine transformations. Therefore a stable keypoint, is a keypoint that withstands these transformations and hence should be found on every image showing a similar or almost similar object in different scale and rotation. Other keypoints on these images are seen as noise. Thus removing them results in a set of only stable keypoints. The same characteristics apply to SURF [11] features as well, as SURF features have the same invariants as SIFT.

2.2 Refinement Techniques

We refine local features of an image as proposed by Zhang *et al.* [10]: First we create new images from the original image by performing affine transformations. Then we extract SIFT-features from these images and the original image and compute a feature map for each affine transformed image. These maps show the position of the features extracted from the transformed image, on the original image. With this feature maps we get the refined features, because less stable keypoints can then be deleted by comparing features of the original image with those of the feature maps. Details are discussed in the next pages.

Suppose we have an image whose coordinate system has its origins in the upper left corner. Its x-axis is going from left to right and the y-axis is going from the top to the bottom. Then the coordinate of a pixel is (x, y). To transform the image we need a transformation matrix T. For scaling the matrix is given by

$$\begin{pmatrix} a1 & 0 & 0 \\ 0 & a2 & 0 \end{pmatrix},$$

where $a1$ is the parameter for scaling along the horizontal axis, and $a2$ the parameter for a scaling along the vertical axis.
And for the rotation with angle α and the rotation center in the middle of the image, i.e., $(x_{center}, y_{center}) = \left(\lfloor \frac{image_{heigth}}{2} \rfloor, \lfloor \frac{image_{width}}{2} \rfloor\right)$ T is:

$$\begin{pmatrix} \cos\alpha & \sin\alpha & (1-\cos\alpha)\cdot x_{center} - \sin\alpha \cdot y_{center} \\ -\sin\alpha & \cos\alpha & (1-\cos\alpha)\cdot y_{center} + \sin\alpha \cdot x_{center} \end{pmatrix}$$

Theoretically the new coordinate (x_{new}, y_{new}) of a pixel at (x, y) can then be computed by multiplying, i.e.,

$$\begin{pmatrix} x_{new} \\ y_{new} \end{pmatrix} = T \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

but this does not work in practice, as for an image there is need of interpolation. In this thesis for scaling the interpolation is done with the bicubic interpolation over a 4×4 pixel neighborhood, because except for one parameter all other parameters in our experiments scale down the images and for this affine transformation the bicubic interpolation has been a proper method in many experiments. For rotation we use the bilinear interpolation as experiments show that this is an appropriate interpolation method for our refinement technique.

In this thesis we only scale and rotate the image in one direction at a time, and after that we generate another transformed version of the original image with a scaling or rotation in the other direction. That means we first scale along the vertical axis and then along the horizontal one, or rotate counterclockwise and then clockwise. The reason is that with more than one transformation in one direction at a time, experiments show that the approximation made by interpolation changes the information we get from the image too much for our experiments. An extreme case of loss of image information through an interpolation is shown in Figure 2.1. As noted in the introduction of this chapter, our main goal is to find out what one simple affine transformation can do. Thus we only test these two transformations per image. Another reason is that otherwise we have to face a loss in our speed up with affine transformations slowing down our program, as well as our main goal is to find out how even these simple transformations can change the performance of SIFT and SURF respectively. Furthermore we do not mix the affine transformations.

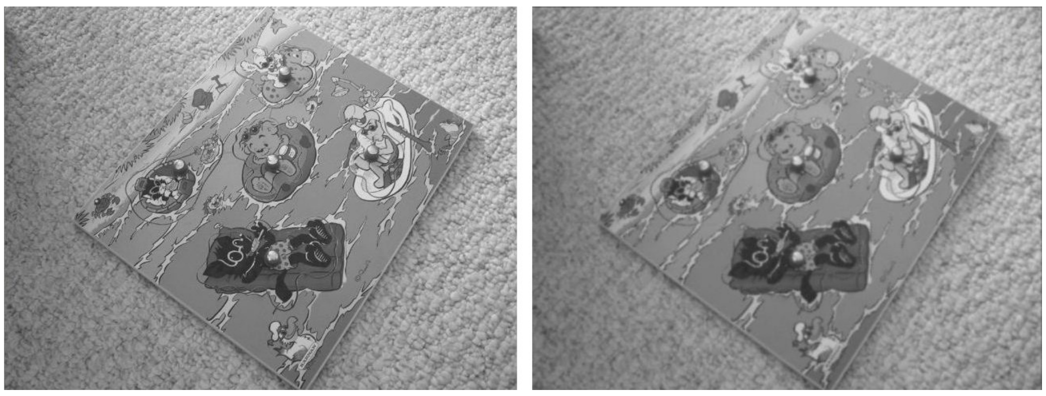

Figure 2.1: An example of information loss caused by 8 counterclockwise 45° rotations(right) of an image(left).

After generating two new images with the method above, we extract SIFT and SURF features of these two images and the original image, as illustrated in Figure 2.2. Then we compute the corresponding coordinates in the original image of each of the detected features in the new images. This is done with the inverse matrix of T, i.e.,

$$\begin{pmatrix} x \\ y \end{pmatrix} = T^{-1} \cdot \begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix}$$

For each transformed new image, we now have mapped all its features back to the original image, generating a feature map as can be seen in Figure 2.2. These feature maps, show all keypoints of the transformed image at the coordinate where they could be in the original image. Thus a features map, links every feature of the transformed image with the place where it could be before the affine transformation. Features on which the transformation has no effect are found on the feature map in a small neighborhood of the similar feature in the original image. Hence noisy keypoints of the original image have no partner in a feature map, and are removed. We therefore need these feature maps to find the more stable keypoints, with stability in a way as defined in section 2.1.

To do so, we have to find a measure for the similarity of a descriptor A in the original image and a descriptor B in one of its feature maps. That way we can be sure which descriptor B that we find in the neighborhood of A, is really the same feature as the feature A, but only slightly shifted on the image. According to Zhang *et al.* [10] this similarity can be computed with:

$$sim(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|},$$

where $\|\cdot\|$ is the L^2-Norm of A and B respectively. The denominator is a always nonnegative product of two distances, and A and B are two 128-dimensional SIFT descriptors or two 64-dimensional SURF descriptors. This sim value computes the cosine of the angle between the descriptor A and B. Hence the higher the sim value is, the smaller the angle between A and B gets and the more similar they are. As the SURF descriptors also have negative values, we use the absolute value of $A \cdot B$ as nominator to compute a nonnegative $sim(A,B)$. The reason is that a negative $sim(A,B)$ stands for an angle between A and B that is bigger than 90°. But from the two angles between two descriptors, we only want to know the size of the smaller angle, which is computed by using the absolute value of the cosine, respectively sim. Therefore we denote the similarity measure in this thesis as:

$$sim(A,B) = \frac{|A \cdot B|}{\|A\| \times \|B\|}$$

We now need the most similar local feature B within a small neighbor region by computing $sim(A,B)$ for each pair of keypoints A and B and determining the sim with the maximal value. As the size of this neighbor region is not defined by Zhang *et al.* [10] we empirically choose a size of 3×3 pixels. To remove less stable keypoints we only keep those that have a similarity larger than a threshold of 0.8.

Here we use the same threshold as in [10]. Finally we receive a set of refined features, as can be seen on the right side of the example depicted in Figure 2.2.

Throughout the rest of this work, we call our approach *SIFTplus* or *SURFplus*, as this makes further discussions easier to understand.

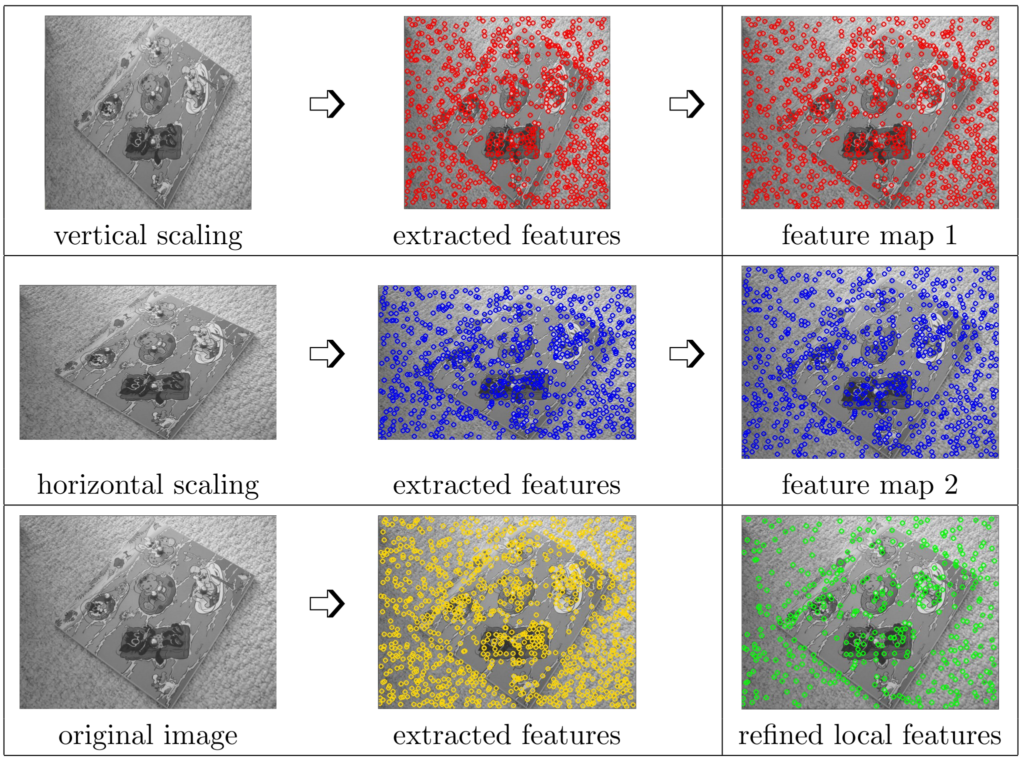

Figure 2.2: An example of local feature refinement by scaling with 0.8

3 Experiments and Evaluations

In this section we conduct some experiments to evaluate the refinement technique. This section gives us a summary about the dataset, the extraction of feature points, the dependency between feature scales and scaling, and the matching methods we used, the near duplicate image retrieval in which this approach could be used and our experiments within it. The results show not only, if SIFTplus or SURFplus has the improvements we expected, but also what parameters for the affine transformations give which improvements with respect to precision, recall, and execution time for feature extraction, refinement and matching. Furthermore we discuss the difference of the results of refinement of SIFT features on the one hand and SURF features on the other hand.

3.1 Dataset

As a dataset for the following experiments with near duplicates we choose the *ukbench* dataset generated by D. Nistér and H. Stewénius [2]. The 10200 images of the size 640×480 pixels are appropriate for our experiments for the following reasons: Firstly these images show enough different items in different ambient conditions, for example items of daily life, in different locations like inside a families home or a super market, as well as in different lighting conditions and as a single object or as a group of smaller objects like a bookshelf. Secondly an image of ukbench has three near duplicates, thus a class consists of four images. Images of one class show the same object, but with different capturing conditions as for example with a different capturing angle or distance, or under different lighting conditions. Some examples are given in Figure 3.1.

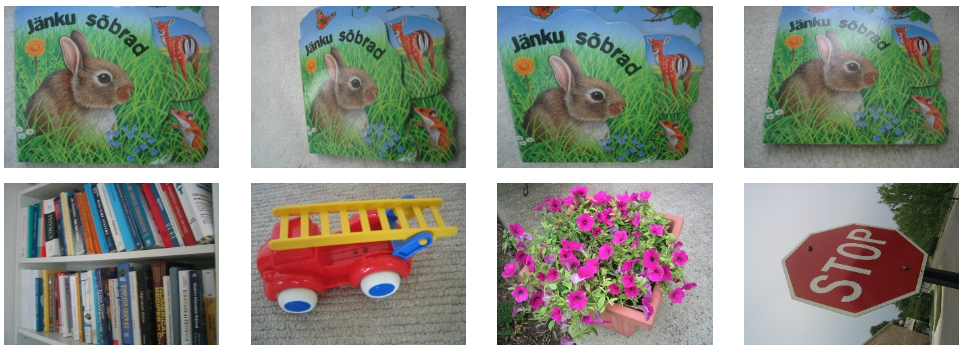

Figure 3.1: A ukbench class with its duplicates(top) and some example of other classes(bottom)

3.2 Affine Transformations

As mentioned in the last chapter we need affine transformations for the refinement of features that lead to less but more stable keypoints, by computing SIFT and SIFTplus as well as SURF and SURFplus features with different scale and rotation parameters. Here we want to find out how many keypoints we can remove with this refinement, as less keypoints also result in less memory space, and sometimes also less execution time for matching. Zhang *et al.* [10] have a scaling parameter of 0.8 in vertical and horizontal direction and SIFT features are invariant to a rotation as big as 30° [1], therefore we want to test parameters that are in the range of these values. Hence for scaling we choose the parameters 0.70, 0.75, 0.80, 0.85, 0.90, 0.98, 1.2 to see if the parameter mentioned in Zhangs work is the best one. For rotation the parameters are 20°, 25°, 30°, 35° and 40°, as we want to see if a invariant boundary has any effects on our kind of refining. To get an overview of how many features remain after these affine transformations we extract SIFT and SIFTplus features, respectively SURF and SURFplus features, of 500 images, each from different image classes of ukbench and compute the average percentage of remaining features.

The following details should be considered while interpreting the results of our experiments. For SIFTs features extraction octave 0 or octave -1 can be chosen as a first octave for the scale space pyramid, where octave -1 means that right before the pyramid is build, the image is doubled using linear interpolation. This can be done if more keypoints are needed than a start from octave 0 can provide. As our images are big enough and also because we want to avoid interpolation through enlarging the images before doing some explicit affine transformations, we start the SIFT features extraction from octave 0. For SURF a Hessian threshold gives the minimum for the Hessian value of all detected features. Features with a Hessian smaller than the threshold are not extracted. As our dataset consists of sharp images with a high contrast, we empirically chose a threshold of 500.

A third point is, that for scaling the steps discussed in 2.2 can be followed with no further considerations, but for rotation we may have to consider some extra property compared to the original image. For example in our experiment, rotation causes an additional black background. This background must be considered as it makes the image bigger, thus feature extraction needs more time, as well as computing the feature map has to be done in respect of the black space of the generated image. Furthermore it leads to additional noisy features as there is an additional edge between the rotated image and the black space. An example of this problem is illustrated in Figure 3.2. Our experiments show that through refinement with rotation features at the edge of the image are removed, leading to a small border with no keypoints. This phenomenon is depicted in Figure 3.3.

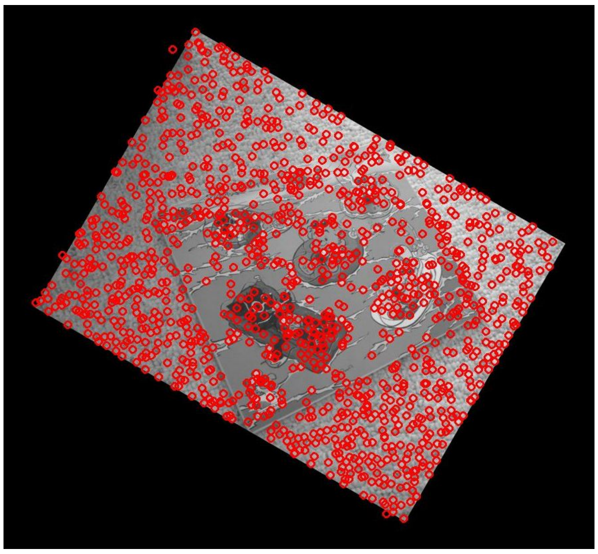

Figure 3.2: An example of local features on a image rotated with 30°

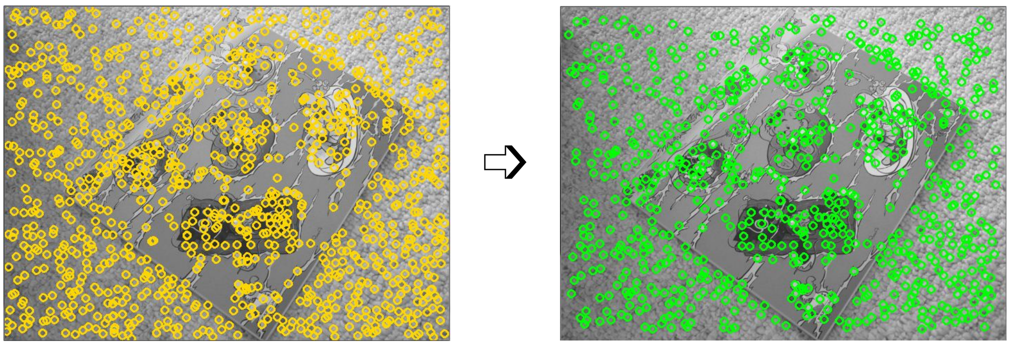

Figure 3.3: SIFTplus with a rotation 30° results in a border without keypoints (image on the right)

The results of our tests for SIFT are illustrated in Table 3.1 and Table 3.2.

Table 3.3 and Table 3.4 show the results of this experiment for SURF. Each row stands for a scaling or rotation in both directions, horizontal and vertical resp. clockwise and counterclockwise. Scaling leads to a great loss of keypoints, whereas rotation only removes about $\frac{1}{3}$ of all features. Moreover there is no real difference in the number of keypoints of different rotation parameters. Hence we do not expect a great difference between the experiments with different rotation parameters, but do expect this in the case of scaling.

Another important point regarding affine transformations is also the execution time needed to transform the image and compute feature maps. Then we can see, for a certain application, if our approachs change of precision or recall (see section 3.4) and this additional time for affine transformations is acceptable compared to a possible speed up by matching with less features. As the execution time depends on implementation and the computer, we only explain briefly what is expected regarding our experiment with 896 images. The average execution time for extracting SIFTplus features with affine transformation is several times higher than the one for SIFT features. This is caused by the additional time for the transforming, but especially for the additional two times of features extraction from the transformed images. These results are illustrated in Figure 3.13 to Figure 3.16 in section 3.4.1.

	SIFT	SIFTplus, scaling with:						
		0.70	0.75	0.80	0.85	0.90	0.98	1.2
AVG number of kp	939	191	270	354	446	548	716	559
AVG percentage of kp	100%	19%	28%	36%	46%	57%	75%	58%

Table 3.1: Comparing the average number and percentage of remaining keypoints (kp) of SIFT to SIFTplus with different scaling parameters

	SIFT	SIFTplus, rotation with:				
		20°	25°	30°	35°	40°
AVG number of kp	939	638	637	639	638	636
AVG percentage of kp	100%	66%	66%	66%	66%	66%

Table 3.2: Comparing the average number and percentage of remaining keypoints (kp) of SIFT to SIFTplus with different rotation parameters in degree

	SURF	SURFplus, scaling with:						
		0.70	0.75	0.80	0.85	0.90	0.98	1.2
AVG number of kp	569	158	204	251	303	363	461	357
AVG percentage of kp	100%	28%	36%	44%	53%	64%	82%	61%

Table 3.3: Comparing the average number and percentage of remaining keypoints (kp) of SURF to SURFplus with different scaling parameters

	SURF	SURFplus, rotation with:				
		20°	25°	30°	35°	40°
AVG number of kp	569	273	262	258	260	271
AVG percentage of kp	100%	46%	44%	44%	44%	46%

Table 3.4: Comparing the average number and percentage of remaining keypoints (kp) of SURF to SURFplus with different rotation parameters degree

3.3 Feature Scales and Refinement by Scaling

One kind of our refinement is a refinement by scaling the image. For each feature a scale value can be assigned to it, describing the scale of an image in an octave where the feature was found. Hence it is possible that there is a dependency between the scale value of a feature and its likelihood to be removed by a refinement with scaling. Our experiments in this section can therefore be useful to define for which scale features are more likely to be rejected and hence are more likely less stable.

Our experiments for this issue is done with a set of 100 image, where all of them are first images of 100 image classes. As scaling parameter we chose 0.9, because a greater scaling leads to too less features to compare feature scales. We then extract SIFT and SIFTplus as well as SURF and SURFplus features from these 100 images. For the SIFT and SIFTplus features we then generate histograms with 600 bins, each of them representing a scale interval of the size 0.1, hence the maximum scale is 60. For SURF and SURFplus we generate histograms with 800 bins, where each of them represents a scale interval of one, thus the maximum scale here is 800. First we do an experiment to see how many keypoints exist for each bin of scale, to get an overview of keypoints and scales. Then another experiments shows us in percent how many features per scale from the original ones are removed.

The results for the first experiment for SIFT are illustrated in Figure 3.4 Keypoints come mainly from smaller scales between 1.5 and 10, thus also the removed keypoints are mainly from smaller scales when respecting the sum of all removed keypoints. Figure 3.5 shows the results of the second experiment. Almost all SIFT features from the smallest scales are removed, as well as a great part of some bigger scales from 15 upward, whereas for scales between 2 and 15 only about 30% to 40% of the original keypoints at that scales are removed. To sum it up, SIFT keypoints from really small scales as well as partly also bigger scales tend to get removed more likely, and hence, with respect of our definition for stability (see section 2.1), tend to be less stable than the keypoints from other scales.

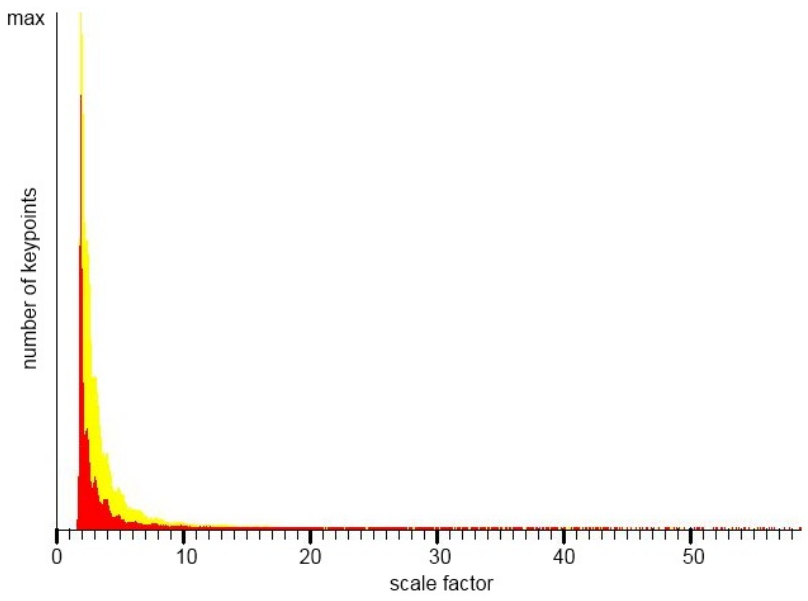

Figure 3.4: The number of keypoints per scale interval of size 0.1 up to an amount of $max \approx 8000$. Yellow illustrates the number of original SIFT features per scale bin, red the number of features that were removed during the refinement. For a clearer illustration the unit of the horizontal axis is 1.0.

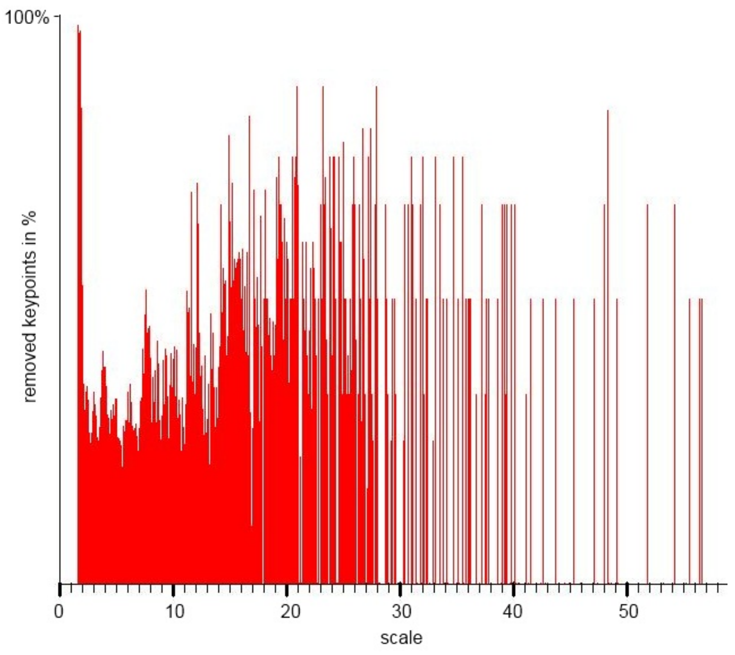

Figure 3.5: The percentage of removed SIFTkeypoints per scale interval of size 0.1. For a clearer illustration the unit of the horizontal axis is 1.0.

The histograms of the same two experiments with SURF are depicted in Figure 3.6 and 3.7. The results are quite similar to those of the experiments with SIFT. Again the keypoints come mainly from smaller scales between 10 and 90, as well as therefore most of the removed keypoints come from small scales. But slightly different from SIFT, SURF features of bigger scales from 100 upward are even most likely to be deleted than those from these scales. And also here the features from the smallest scales, i.e. scales less than 20, tend to be more likely to be removed. Hence in summary we have the same result as for SIFT: keypoints from really small scales and those from bigger scales tend be more likely to get removed, which means they are in our definition less stable.

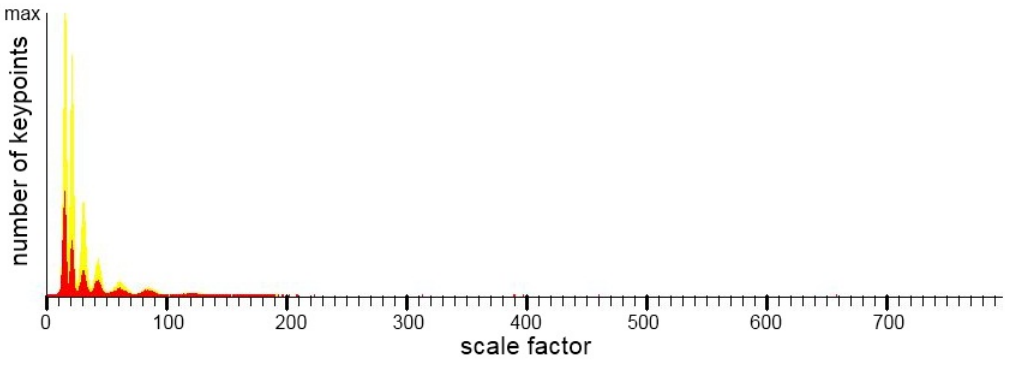

Figure 3.6: The number of keypoints per scale interval of size 0.1 up to an amount of $max \approx 4700$. Yellow illustrates the number of original SURF features per scale bin, red the number of features that were removed during the refinement. For a clearer illustration the unit of the horizontal axis is 10.

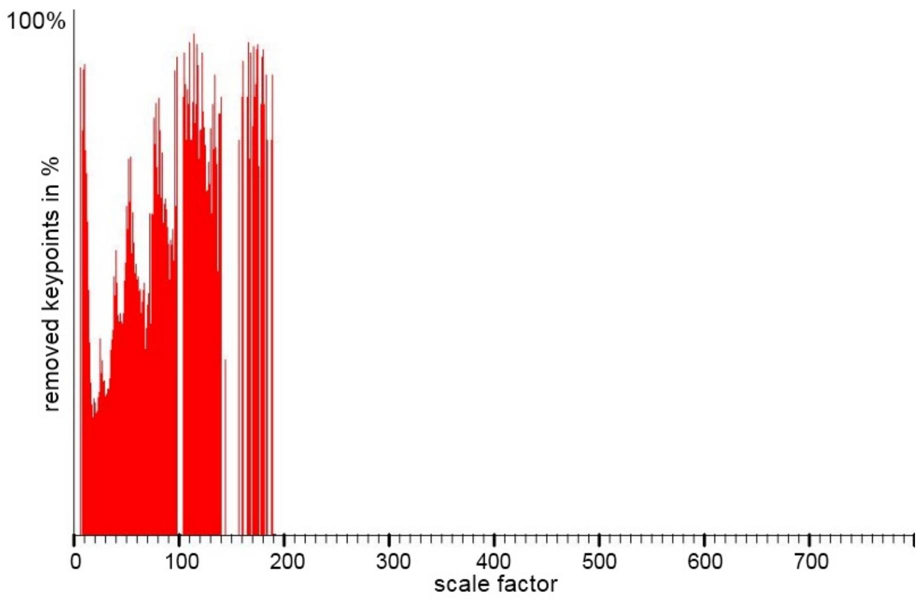

Figure 3.7: The percentage of removed SURF keypoints per scale interval of size 0.1. For a clearer illustration the unit of the horizontal axis is 10.

3.4 Near-duplicate Image Retrieval

Near-duplicate Image Retrieval has been successfully used in copyright issues or large-scale web image retrievals [2, 12], to efficiently find duplicates to reduce redundant contents on the web to save memory space, or detect image spam [13]. Hence there is a huge demand on improving any part of Near-duplicate Image Retrieval. Also matching in a Near-Duplicate dataset gives no uncertainty for our evaluation, which image is a match to a sample image, i.e. an image that is a near- or partial-near image as the sample image, and which one is not. Thus the Near-duplicate Retrieval lets us evaluate our matching experiments without any doubt of an image being a true match or not. For these reasons we chose the Near-duplicate Image Retrieval to test our approach.

As a matching technique for our experiments we wanted to use both, a simple one with as less approximation as possible, as well as a faster one with approximation. Thus we first chose matching proposed by Lowe [1], the nearest neighbor ratio matching. On the one hand it uses no approximation and is really straight forward, but on the other hand it is very slow, as each descriptor from the sample has to be compared to each descriptor from the test image. And above all a match is only based on three matching descriptors. Thus it does not work properly on a large database, as there is a great chance of a matching descriptor by accident. Therefore we also use another approach, the Bow that has a quantization, hence an approximation, but is much faster than the first approach allowing us to do experiments with a larger sub dataset of ukbench. Also the chance of mismatches, generated by randomly matching descriptors, decreases, as there is only one vector for each image, but that vector describes the whole image. For the test we select a sub dataset with s sample and t test images. The sample images are a set of first images of the first s classes of ukbench, whereas the test set consists of the sample image and its three duplicates and $t - 4$ first images of just as many classes. Hence for a experiment with s samples and t test images we need $s \cdot 4 + t - 4$ images.

To compare SIFT to SIFTplus and SURF to SURFplus and also SIFTplus to SURFplus in the case of matching, we first extract the original resp. plus features with the parameters mentioned before in section 3.2. After that we try to find matches for each sample with one specific matching technique. The results are summed up to compare precision and recall of SIFT and SIFTplus with different parameters. To compute the precision and recall we count the number of true matches and false matches, meaning test images that are declared by the matching technique as a match but are not a match. In the best case we should receive four true matches and zero false matches. In general the number of true matches is called the number of *true positives* (tp), the missing true matches are called *false negatives* (fn) and the number of false matches, or mismatches *false positives* (fp). With these values we can compute the precision and recall values of a matching experiment, denoted as

$$Precision = \frac{tp}{tp + fp}$$

and

$$Recall = \frac{tp}{tp + fn}$$

3.4.1 Nearest Neighbor Ratio

We begin with a rather simple and intuitive approach for matching that is proposed by Lowe [1]. The main idea is that a matching test image to some sample images must have several descriptors that are very similar to descriptors from the sample image. To find matching descriptors, every descriptor from the sample image has to be compared with each from the test image, by computing the Euclidean distance and looking for the shortest distance. As a last step we then have to be sure that this closest descriptor is really a matching one. That is where the concept of the ratio is used, meaning a correct match has to be significantly closer than the closest incorrect match. The reason is that for false matches there are some other false matches that are almost as close as the first one to the descriptor we are matching to. Therefore the second-closest is used to give an estimator of the density of false matches withing a portion of false matches, that are accidentally seen as the closest descriptor and hence are in fact incorrect matches. The ratio of closest to second-closest neighbor of the descriptors is than a measure for the similarity.

In detail we first compute the Euclidean distance between a descriptor D_s of the sample image and each descriptor D_t of the test image. Then we determine the nearest and second nearest descriptor D_{t1} and D_{t2}, to compute the ratio of their distances with respect to D_s, i.e.,

$$\frac{\|D_s - D_{t1}\|}{\|D_s - D_{t2}\|}.$$

A matching descriptor D_s to the test image is then defined as a descriptor whose ratio of distances is lower than some threshold. Lowe [1] proposed that if we find at least three matching descriptors in a test image, this test image is a match. We first set this threshold to 0.8 as seen in [1]. But this threshold is inappropriate for our matching, because first experiments show that every test image is seen as a match, meaning that we have the highest number of mismatches that is possible. Based on Lowes graph [1], represented in 3.8, and his example on his web page [14] we decide to use a threshold of 0.6 in our experiments. This threshold is chosen because it is the threshold in the middle between 0.45, the threshold where the probability of correct matches is the highest, but also has a high risk of loosing true positives, and 0.8 the lowest threshold, where we loose less true positives, but start to get more false positives. Since the ratio uses the idea of seeking for the exact nearest neighbor, this approach does not need an approximation but is very slow, so if we want to manage an experiment in an appropriate time, we have to pick a rather small dataset.

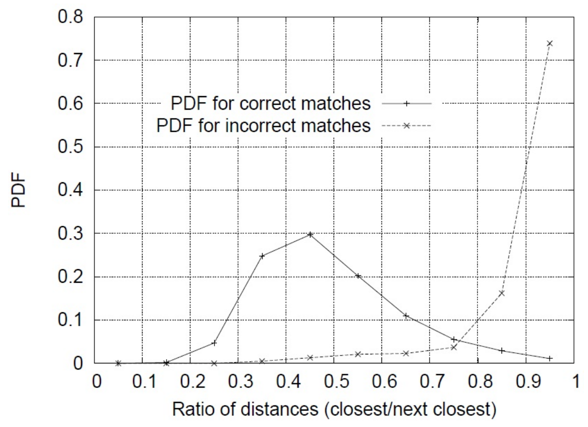

Figure 3.8: Lowes Figure 11: The PDF of the ratio of distances for correct matches(solid line) and incorrect matches(dotted line)

Our first experiment is with a sub dataset of ukbench with 100 samples and 500 test images and the parameters mentioned before in section 3.2. Following the steps of section 2.2, we generate SIFTplus and SURFplus features. Then we try to find matching test images for each sample. Table 3.5 and Figure 3.9 to Table 3.6 and Figure 3.10 show that with an increasing reduction of the image size the precision increases too, whereas the recall decreases. In other words, the more we scale down the less false matches we get but then we also lose true matches. This holds for SIFTplus as well as for SURFplus, but with a varying degree of increase and decrease. While SIFTplus results in an increase of up to five-time of the precision, the precision of SURFplus has only a maximum of 1.3-time increase. In the case of recall both of them lose about 25% of their original recall value. Thus scaling seems to have a greater effect on SIFT then on SURF features. An extension of the image size has the same effect as a reduction, considering recall and precision, but here we have to consider on an additional time in the case of feature extraction. Comparing this to rotation, depicted in Table 3.7 and Figure 3.11 to Table 3.8 and Figure 3.12, we can not see a significant change in precision and recall, when increasing the degree of the rotation angle. There is only a difference between the recall of SIFT resp. SURF with rotation and without rotation of about 10%. In the case of precision there is no significant change.

	SIFT	SIFTplus, scaling with:						
		0.70	0.75	0.80	0.85	0.90	0.98	1.2
Precision	1.59%	8.97%	6.34%	4.76%	4.19%	3.41%	4.22%	3.26%
Recall	94.68%	71.86%	78.92%	81.36%	84.93%	84.40%	84.93%	87.44%

Table 3.5: Comparing the Precision and Recall of SIFT to SIFTplus with different scaling parameters

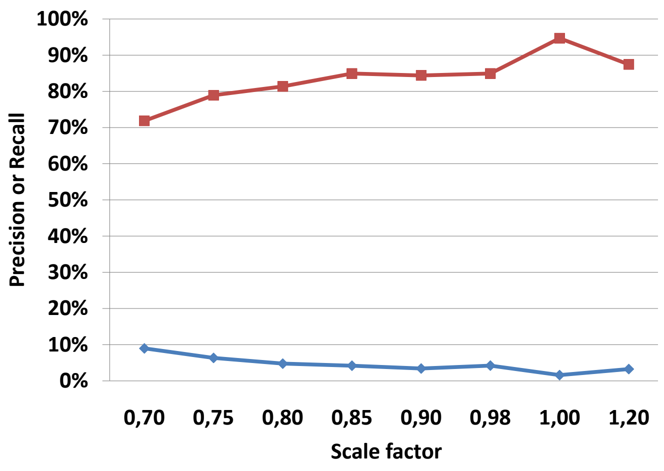

Figure 3.9: Precision(blue) and Recall(red) of SIFT and SIFTplus with different scaling parameters

	SURF	SURFplus, scaling with:						
		0.70	0.75	0.80	0.85	0.90	0.98	1.2
Precision	2.27%	3.03%	2.90%	2.77%	2.73%	2.73%	2.51%	2.34%
Recall	85.92%	61.89%	68.09%	72.69%	77.63%	78.86%	81.53%	75.33%

Table 3.6: Comparing the Precision and Recall of SURF to SURFplus with different scaling parameters

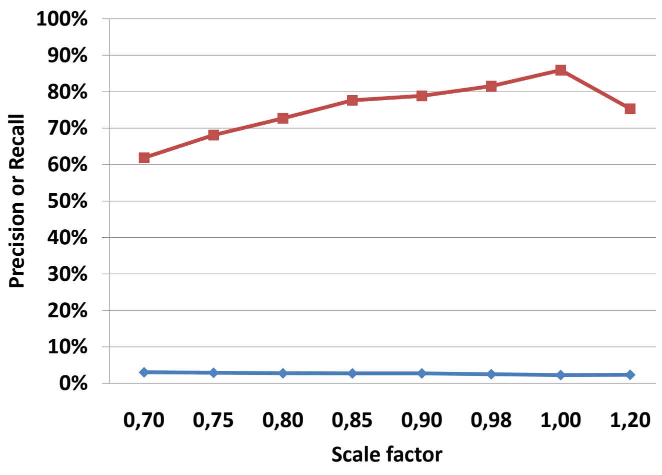

Figure 3.10: Precision(blue) and Recall(red) of SURF and SURFplus with different scaling parameters

	SIFT	SIFTplus, rotation with:				
		20°	25°	30°	35°	40°
Precision	1.59%	3.31%	3.03%	3.10%	3.22%	3.16%
Recall	94.68%	86.98%	86.51%	86.45%	86.38%	87.44%

Table 3.7: Comparing the Precision and Recall of SIFT to SIFTplus with different rotation parameters

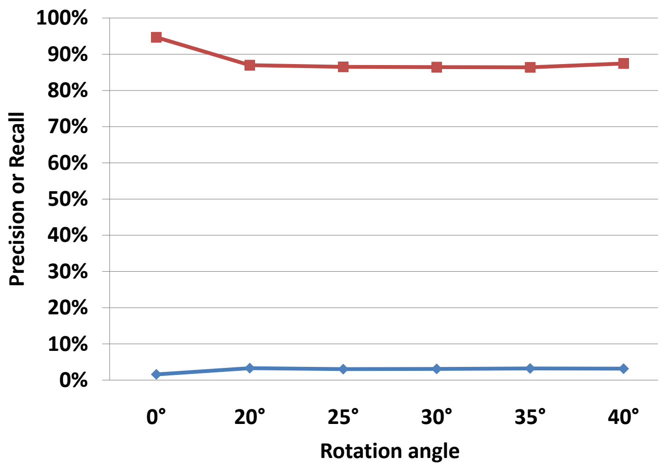

Figure 3.11: Precision(blue) and Recall(red) of SIFT and SIFTplus with different rotation parameters in degree

	SURF	SURFplus, rotation with:				
		20°	25°	30°	35°	40°
Precision	2.27%	2.54%	2.58%	2.52%	2.51%	2.46%
Recall	85.92%	76.76%	72.80%	73.88%	74.49%	73.58%

Table 3.8: Comparing the Precision and Recall of SURF to SURFplus with different rotation parameters

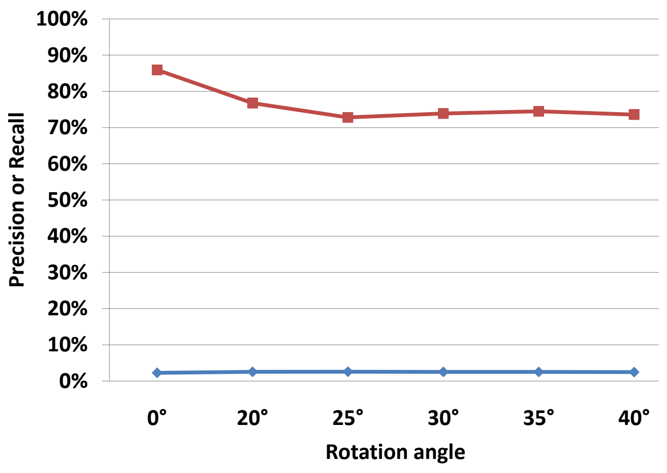

Figure 3.12: Precision(blue) and Recall(red) of SURF to SURFplus with different rotation parameters in degree

In this context we also have to look at the execution time, as this also has an influence on the pro and contra of this approach for a near duplicate retrieval. As Lowes nearest neighbor ratio matching strategy demands a comparison of each descriptor with every other descriptor, the execution time for finding the duplicates of a sample image is heavily dependent on the number of features per image, thus the execution time sinks the more features we remove through affine transformations. As illustrated in Figure 3.13 to Figure 3.16 the time penalty discussed in section 3.2 for scaling is negligible small compared to the big speed up we get from a smaller set of keypoints, whereas the loss of time for rotation is almost so high that, especially for SURF, the speed up for the matching becomes less effective for the whole retrieval process. Note that SIFT and SURF have a very big difference in the execution time as the features are extracted on different computers.

To sum it up, scaling clearly has an effect on SIFT and SURFplus features, but with more effect on SIFT than on SURFplus features, whereas rotation has almost no effect on their precision. A decrease in the recall of both SIFT and SURF features matching can be seen, but it does not matter how large the rotation angle is. A enlargement of the images has less advantages than an reduction, as enlarging leads to a greater additional time for features extraction, but with the same effects in precision, recall, and number of keypoints, as a reduction of the image. Although we lose some matches, we still get a rise in the precision, hence get less mismatches and a great speed up. Also the decreasing number of mismatches is characteristic for less noise and therefore an evidence that only more stable keypoints remain, as discussed in section 2.1. With such different results for different scaling parameters, the parameters for a certain application have to be chosen individually, as there is no general optimal value.

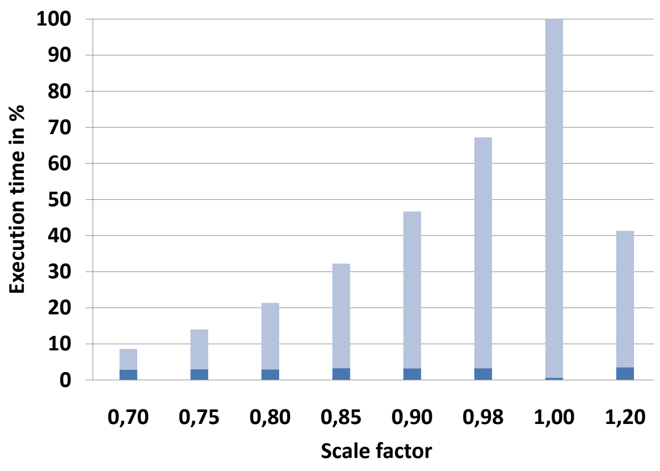

Figure 3.13: SIFT and SIFTplus with different scaling parameters: The execution time for extracting features(dark) for one image and looking for its matches(light), relative to those of the original SIFT(scaling with 1.0).

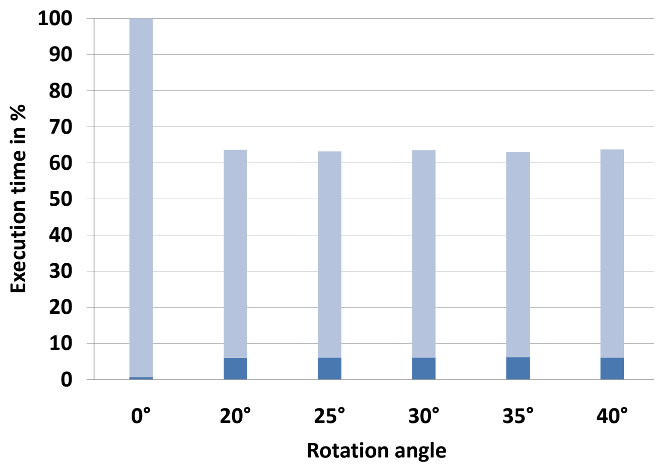

Figure 3.14: SIFT and SIFTplus with different rotation parameters in degree: The execution time for extracting features(dark) for one image and looking for its matches(light), relative to those of the original SIFT(angle 0°).

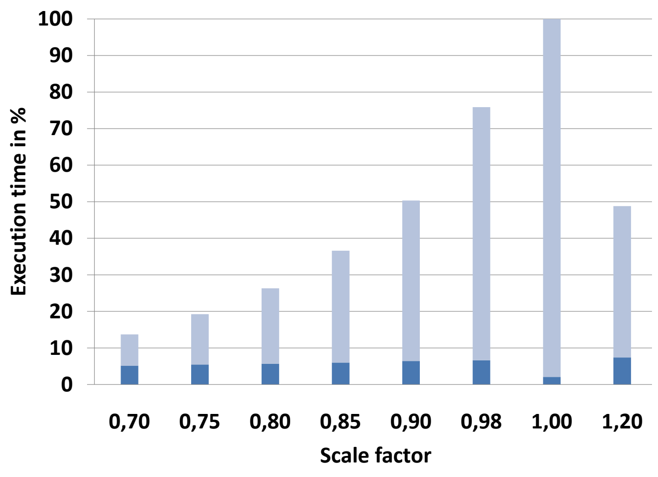

Figure 3.15: SURF and SURFplus with different scaling parameters: The execution time for extracting features(dark) for one image and looking for its matches(light), relative to those of the original SURF(scaling with 1.0).

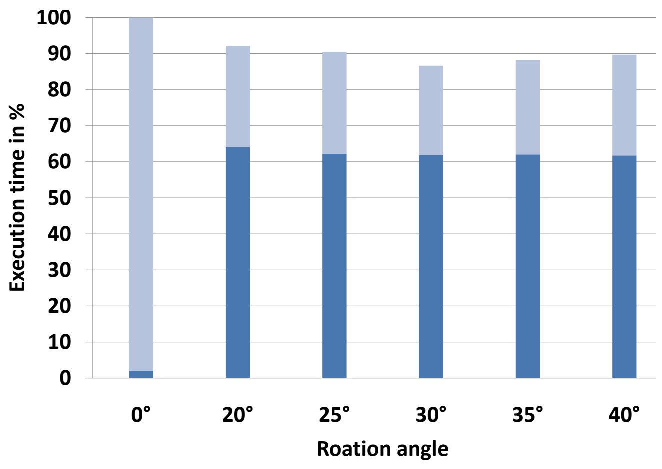

Figure 3.16: SURF and SURFplus with different rotation parameters: The execution time for extracting features(dark) for one image and looking for its matches(light), relative to those of the original SURF(angle 0°).

3.4.2 Bag-of-words

Our second matching strategy is a BoW model, that is similar to the one proposed by Nistér and Stewénius [2]. Here extracted features such as SIFT or SURF are quantized and indexed in one step, to get a bag of visual words for each image. To receive these visual words, each descriptor of an image is assigned to a visual word by a tree with a hierarchical k-means clustering. As a visual word is only a simple integer, each descriptor of an image is then represented by an integer. After this, we sum these IDs up by counting how many of this visual words exist in the image, to get a histogram, where each bin represents a visual word. A comparison of images is then done by comparing the normalized histogram vectors. Since each image has only one histogram vector, there is only one comparison per image needed, resulting in less comparisons than a all to all comparison of descriptor vectors, and hence to a faster near duplicate retrieval. Also matching images by accident is less possible, as we do not compare each single descriptor to each other, but with the histograms, the whole content of an image to the content of another one.

Following section three of [2] we first use a large set of descriptors, extracted from about 16000 flickr images as our training data to build the vocabulary tree. To create this tree we compute k clusters by doing a hierarchical k-means clustering on the training data. Here k denotes the branch factor, meaning each node in the tree has k children. Each vector of the training set is then assigned to the closest cluster center. Thus we get k groups of vectors. The clustering and partitioning process is then done with each group recursively, up to a maximum of l-times, whereby each partitioning into k groups only respects the distribution of the parent quantization cell. Figure 3.17 is Figure 2 from [2], illustrating the process of building the vocabulary tree.

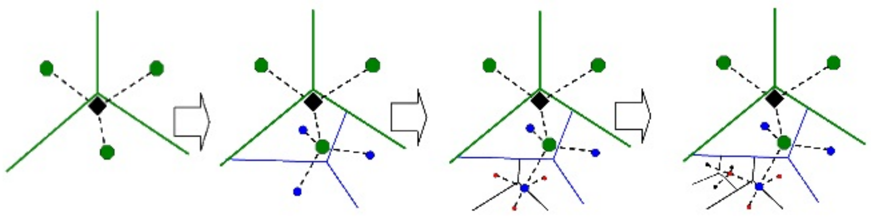
Figure 3.17: The process of building the vocabulary tree, with $k = 3$.

To find the visual word for one descriptor, we compare the descriptor with each cluster center at each level. The closest cluster center is then a node of the descriptors path through the tree, from the root to the leaf. This process cost $k \cdot l$ dot products, which is very efficient as long as k is not too large. Each target leaf represents a visual word, that is then assigned to the descriptor, whereas the path down to the tree can be used as an integer ID for a scoring method, as it is done by Nistér and Stewénius [2]. Finding visual words for each descriptor of an image then results in a bag of visual words for this image.

Unlike Nistér and Stewénius [2] we now create a histogram vector from the bag of visual words of each image. Each bin of the vector represents a visual word, counting how many of these visual words appear in the image. To be able to compare the histograms properly, each histogram is normalized to one, to get a relative frequency of each word in an image. To find matches of a sample image, the L^2-distance between the histogram of the sample image and the histograms of the test images are computed. The closer the histogram of the test image is to the histogram of the sample image, the more similar they are, and thus the more likely the test image is a match. Compared to the nearest-neighbor ratio used before in this chapter, the whole process for finding matching images is much more efficient and reliable. The reason is first of all, that here we only have to compare one histogram vector per test image, which is way more efficient than comparing each descriptor of the sample with each one of the test image. Also as the number of comparisons decreases, and as one histogram vector does not stand for only one features out of many, but for the whole content of the image, accidentally matchings are less likely. And also the extraction of visual words and the computation of the histogram as an additional step after the features extraction can be done very efficiently.

In our experiments we empirically chose a vocabulary of 3000 visual words, where k is 3000 and l is 1. For our experiments we use a set of 500 sample images and 2500 test images. First we assign the visual words with the extracted original SIFT and SURF features of the images, and then with the SIFTplus and SURFplus features with different scale and rotation parameters, as mentioned in section 3.2. We then evaluate the four nearest neighbors to a histogram of a sample. In the best case, four of them should be true positives showing the sample image and its duplicates. As we only receive four images per query, which is the size of a class in ukbench, the number of false positives and the number of false negatives are the same. Hence precision and recall values are also of the same size, so that we only show the values for the precision in our following figures.

As illustrated in Table 3.9 to Table 3.10 and Figure 3.18 to Figure 3.19 there is no significant change in precision for a BoW matching with SIFTplus features, instead there is a slight decrease of the precision from 29.5% up to 26.0% with rising scale factor and up to 27.7% with rising rotation factor. For SURF this decrease is even more significant for all scaling and rotation parameters of SURFplus. Table 3.11 to Table 3.12 and Figure 3.20 to Figure 3.21 depict that, with refined features through scaling or rotation the precision sinks from 34.8% to a minimum of 27.8% and 28.8%. Therefore in a BoW matching like ours, with this vocabulary size, SIFTplus respectively SURFplus features appear not to be more stable than the original SIFT features, as they do not reduce the number of mismatches.

Also there is no noticeable speed up. On the one hand because the features extraction and the visual words computation for SIFT and SIFTplus, respectively

SURF and SURFplus features the execution time is not very different. On the other hand because the number of comparisons for looking for matches for a sample image, is the same for the original features and the refined ones, as there is only one descriptor for each image and this descriptors have always the size of the vocabulary.

Still there is one point on the upside: the memory space. For a certain application with a large image database, where the extracted features are stored, to be assigned to visual words and a later histogram matching, SIFTplus can still be considered as an memory saving alternative to SIFT, as a scaling factor of only 0.9 removes about 40% of the features while having almost the same precision. The same applies for the rotation, where a rotation angle of only 20° can save about 34% of the number of keypoints while having almost no loss of precision. For Surf this saving of both, keypoints and precision, is valid as well.

	SIFT	SIFTplus, scaling with:						
		0.70	0.75	0.80	0.85	0.90	0.98	1.2
Precision	29.5%	26.1%	26.3%	26.8%	27.0%	29.3%	28.2%	27.4%

Table 3.9: Comparing the Precision of SIFT to SIFTplus with different scaling parameters

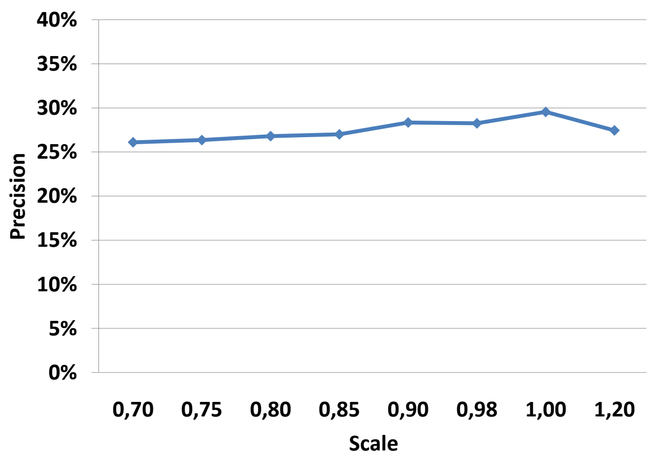

Figure 3.18: Precision of SIFT and SIFTplus with different scaling parameters. Scaling parameter 1.0 denotes the original SIFT.

	SIFT	SIFTplus, rotation with:				
		20°	25°	30°	35°	40°
Precision	29.5%	28.6%	28.1%	27.9%	27.8%	27.7%

Table 3.10: Comparing the Precision of SIFT to SIFTplus with different rotation parameters

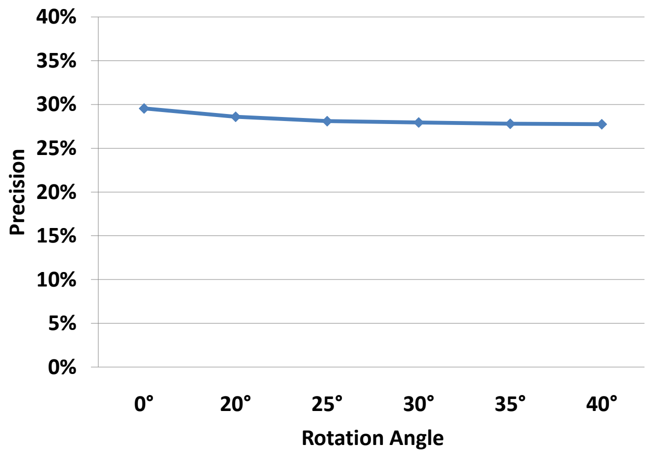

Figure 3.19: Precision of SIFT and SIFTplus with different rotation parameters. Rotation angle 0° denotes the original SIFT.

	SURF	SURFplus, scaling with:						
		0.70	0.75	0.80	0.85	0.90	0.98	1.2
Precision	34.8%	27.3%	27.4%	28.5%	29.5%	31.1%	32.1%	30.8%

Table 3.11: Comparing the Precision of SURF to SURFplus with different rotation parameters

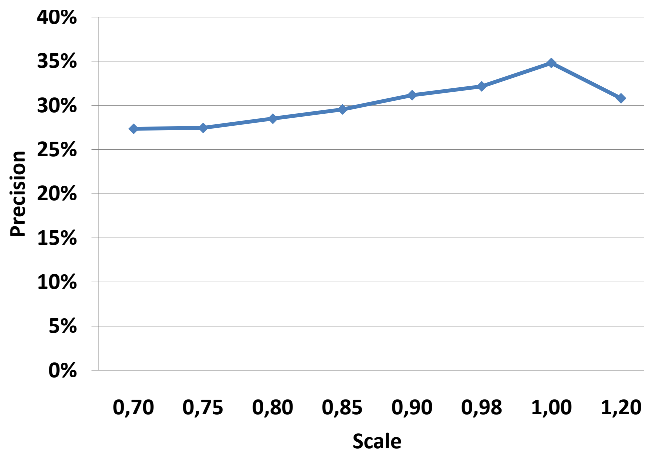

Figure 3.20: Precision of SURF and SURFplus with different scaling parameters. Scaling parameter 1.0 denotes the original SURF.

	SURF	SURFplus, rotation with:				
		20°	25°	30°	35°	40°
Precision	34.8%	28.3%	28.8%	28.8%	28.2%	28.8%

Table 3.12: Comparing the Precision of SURF to SURFplus with different rotation parameters

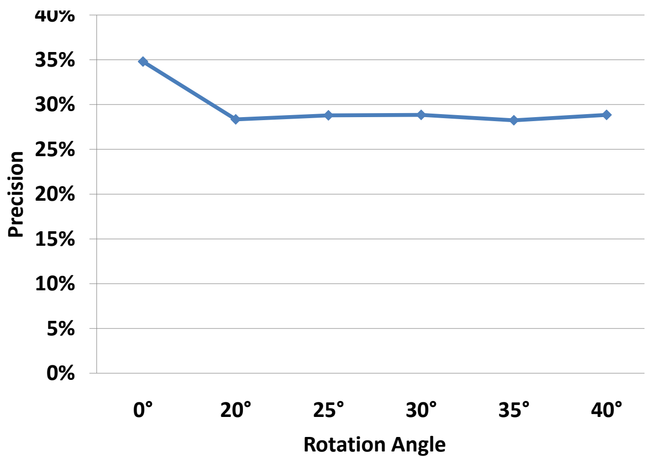

Figure 3.21: Precision of SURF and SURFplus with different rotation parameters. Rotation angle 0° denotes the original SIFT.

4 Conclusion

We have presented a method to refine SIFT and SURF features to a set of more stable SIFTplus and SURFplus features. The approach is build upon the fact that SIFT and SURF features are invariant to affine transformations, i.e. scaling and rotation, and therefore are not affected by such transformations of the image. Thus they are seen as more stable features. We introduced a feature map for each transformed image, that contains the corresponding coordinates in the original image, for each feature. With these features maps less stable features are found and then removed. The performance of our the remaining features are then tested in a near-duplicate image retrieval with two different matching techniques.

Our experiments have shown that even with a light scaling with 0.9 the number of keypoints can be reduced to 57% for SIFT features and 64% for SURF features. For rotation the size of the angle does not matter. In any case rotation results in a reduction of keypoints to about 66% for SIFT and 45% for SURF. We have seen that there is a connection between the scale from which the feature comes from and its likelihood to be removed during refinement with a scaling. Experiments show that especially keypoints from really small scales and from a certain bigger scale up, are almost completely removed.

In a nearest-neighbor ratio matching our approach has shown some good improvements. Although the recall sinks, the approach is still an alternative for some applications as we get a great plus in precision, and hence less mismatches. Furthermore a speed up brings another advantage, considering the execution time for features extraction and matching. We figured out, that our refined features can at least half the execution time for queries in a near-duplicate retrieval.

A matching experiment with a vocabulary tree with a vocabulary size of 3000 has shown that, although no improvement in precision, recall or execution time can be expected, we may still benefit from the reduced number of keypoints in memory space demanding applications.

As we have only examined each affine transformation individually for two matching techniques, future work with our approach can be done in the case of parameters, the similarity threshold, or combination of these two affine transformations. Also other vocabulary trees may give a better result, leading to application that can take advantage from both, refined features and BoW matching methods.

Acknowledgements

At the end of this work we acknowledge the Multimedia Computing Lab of the University of Augsburg, especially Prof. Dr. Rainer Lienhart, Dipl.-Inf Fabian Richter, Dipl.-Inf. Stefan Romberg and Ma. of Sc. Christian X. Ries for their support.

Bibliography

[1] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Journal of Computer Vision*, 2004.

[2] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[3] Faraj Alhwarin, Chao Wang, Danijela Ristić-Durrant, and Axel Gräser. Improved sift-features matching for object recognition. *BCS International Academic Conference*, 2008.

[4] Faraj Alhwarin, Danijela Ristić-Durrant, and Axel Gräser. Vf-sift: Very fast sift feature matching. *Lecture Notes in Computer Science*, 2010.

[5] Tinne Tuytelaars. Dense interest points. *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[6] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2009.

[7] Andrew Stein and Martial Hebert. Incorporating background invariance into feature-based object recognition. *Seventh IEEE Workshops on Application of Computer Vision*, 2005.

[8] Panu Turcot and David G. Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems. *IEEE Conference on Computer Vision Workshops*, 2009.

[9] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981.

[10] Shiliang Zhang, Qingming Huang, Gang Hua, Shuqiang Jiang, Wen Gao, and Qi Tian. Building contextual visual vocabulary for large-scale image applications. *ACM,Proceedings of the international conference on Multimedia*, 2010.

[11] Herbert Bay, Tinne Tuytelaars, , and Luc Van Gool. Surf: Speeded up robust features. *Computer VisionECCV*, 2006.

[12] Zhong Wu, Qifa Ke, Michael Isard, and Jian Sun. Bundling features for large scale partial-duplicate web image search. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[13] Xin Yang, Qiang Zhu, and Kwang-Ting Cheng. Near-duplicate detection for images and videos. *ACM Workshop on large-scale multimedia retrieval and mining*, 2009.

[14] David G. Lowe. Demo software: Sift keypoint detector. *URL: http://www.cs.ubc.ca/ lowe/keypoints/*, 2005.

Attachement

The attached DVD contains the following data:

- The Visual Studio 2008 project that was used for the experiments
- The libs MMCLib, Rlib, tbb and ClusterAPI from the Multimedia Computing Lab
- The OpenCv 2.2 lib and the vlfeat 0.9.9 lib
- Tables and *.txt files with the output of the experiments (as far as they are saved)
- Realted work papers as pdfs
- The latex, pdf and image files for the thesis