Image matching on a mobile device

Honours project

Authors:
Steve Nowee
Nick de Wolf
Eva van Weel

Supervisor:
Jan van Gemert

Universiteit van Amsterdam

Contents

Acknowledgements

We would like to thank Jan van Gemert, our supervisor for this project, and Raquel Fernández Rovira, the coordinator of the honours program. Also, we would like to thank Morris Franken for his assistance during the project.

1 Introduction

Computer vision is a field in Artificial Intelligence (AI) in which the acquiring, processing, analyzing and understanding of images are the core tasks. There is a wide variety of applications of the computer vision domain, e.g. navigation of an autonomous car, face recognition and image matching.

This project will focus on image matching, in particular the development of a mobile application which allows the user to take a picture and find the most similar image on his or her mobile device. In order to achieve this, images representing the same object need to be identified even if they are viewed from different viewpoints or if they are partially occluded.

Building such an application on a mobile device might require a different approach due to memory and speed constraints.

2 Theory

2.1 Bag of words

The bag-of-words model (BoW model) originates from the field of natural language processing and is used to represent a text as an unordered collection of words. By representing documents as vectors of word frequencies from this unordered collection of words, documents can be evaluated on similarity. This same model can be applied to the domain of computer vision by representing images as collections of visual words [Sivic and Zisserman, 2009]. The following steps are necessary to match images using the BoW model:

1. Region detection
2. Region description
3. Descriptor clustering
4. Frequency vector construction
5. Frequency vector comparison

These steps will be elaborated on in the next sections.

2.2 Region detection

In order to match images, features or region have to be detected for each image. After all, if two images have a lot of those features or regions in common, they will most likely be similar. Such a feature or region can be defined as an *interesting* part of the image. Parts which are describing for the image, e.g. blobs or corners. These corners and blobs can be found with feature or region detection algorithms. The results of two such algorithms are shown in Figure 1.

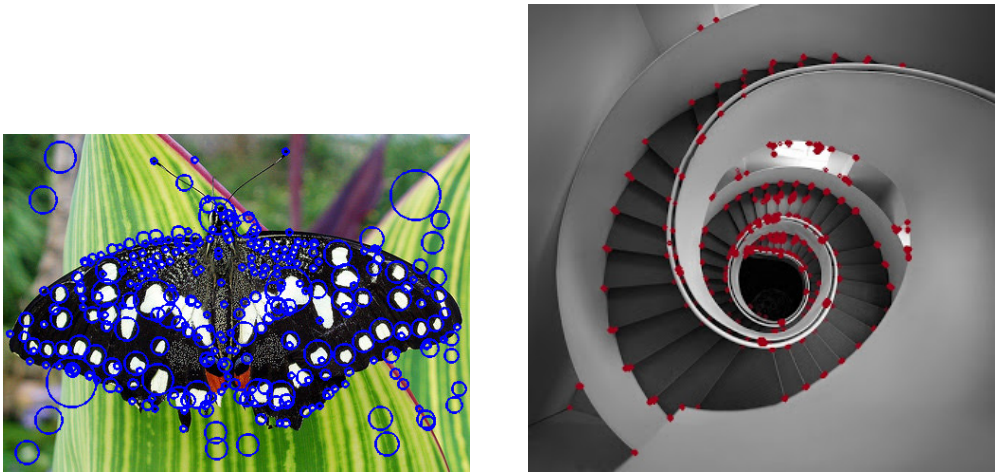

Figure 1: The result of blob (left) and corner (right) detection algorithms. [1]

Two images describing the same scene from different viewpoints should be evaluated as very similar. Since they describe the same objects and therefore the same regions. In order to achieve this, regions have to be detected in a viewpoint invariant manner, meaning that a region has to be detected regardless of the viewing angle. Furthermore, the same regions have to be detected in both images despite changes in illumination, scale or in case of partial occlusion. Three methods to detect regions with these properties are Maximally Stable Extremal Regions (MSER), Harris Corner Detection and Features from Accelerated Segment Test.

2.2.1 Maximally Stable Extremal Regions

Distinguished regions (DRs) are regions which can be detected in images regardless of the viewing angle, because they possess some invariant, distinguishing and stable properties. An example of such a DR are extremal regions [Matas et al., 2004]. The concept of extremal regions can be explained by giving all pixels below a certain threshold t the color black and all the pixel above that threshold the color white. Looking at a sequence of thresholded images I_t, the first image will be white. Increasing the threshold, black pixels will appear on the image. These smaller regions with black pixels, will eventually merge and form larger black regions. Ultimately, the image will be completely black.

[1] Left image from: `http://www.kixor.net/school/2008spring/comp776/assn1/`, right image from: `http://glowingpython.blogspot.nl/2011/10/corner-detection-with-opencv.html`

Maximally Stable Extremal Regions (MSERs) are those regions in the image, where the size of the regions remain stable over a large range of thresholds. These MSERs can be used to detect blobs in images, where these blobs have the property of being invariant to the viewing angle. Because of this property MSERs can be used to recognize objects in images and to match images.

2.2.2 Harris Corner detection

A corner in an image can be defined in two ways, 1. the intersection of two edges, where an edge is a sharp change in image contrast or 2. a point with different edge directions in its local neighborhood. To detect these corners, Moravec came up with a corner detector [Moravec, 1980]. His corner detector uses the average changes of image intensity which result from shifting a window in an image by small amounts in various directions. The sum of squared differences (SSD) is used to calculate the interest measure of the window based on four directions (horizontal, vertical and two diagonals). The interest measure is equal to the minimum of the four sums.

If the selected window in the image is approximately constant in intensity, shifts in all four directions will result in a small SSD, since the adjacent windows look very similar. If the window encounters a line, a shift along the line will again result in a small change, while a shift perpendicular to the line will result in a large change. In case the window covers a corner, shift in all directions will result in large changes. Therefore, a corner can be detected if the minimum of the four sums, produced by the shifts, is large. A local maximum is found.

An improved version of Moravec's corner detector was made by Harris and Stephens [Harris and Stephens, 1988]. Moravec considered shifts in discrete 45 angles, which results in an anisotropic operator. An anisotropic operator is directionally dependent, because of which some edges are mistakenly chosen as an interest point. If the edge is not in the direction of its neighbours, the minimum SSD is large, suggesting the edge is a corner. Harris considered shifts in all directions, because of which a more accurate distinction between edges and corners can be made. Also, Harris used a circular Gaussian window instead of a rectangular window to reduce noise.

2.2.3 Features from accelerated segment test

Another algorithm to detect corners in images is called features from accelerated fragment test (FAST) [Rosten and Drummond, 2005]. FAST uses a circle of 16 pixels around a point p to verify whether this point is a corner.

P is detected as a corner when at least 12 of these 16 surrounding pixels are below or above the intensity of p by some threshold t. Figure 2 shows FAST corner detection for a small part of an image.

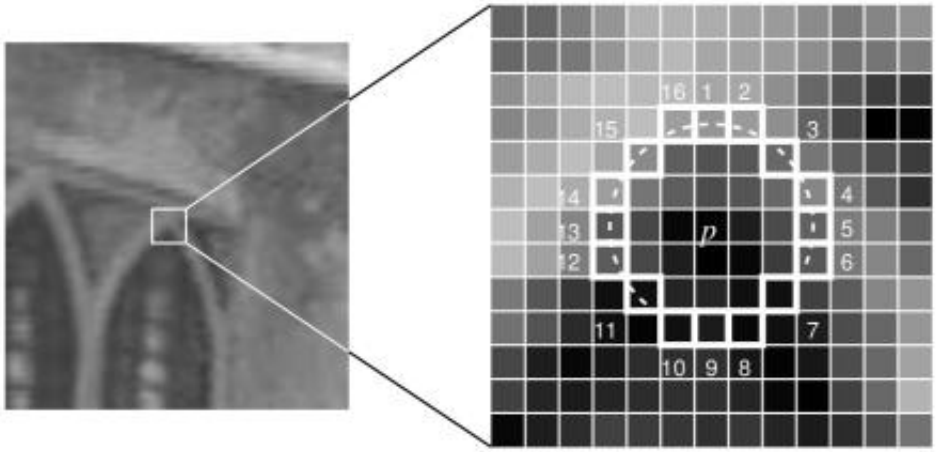

Figure 2: FAST corner detection. [2]

To be able to quickly reject points that are not corners, pixels 1, 5, 9 and 13 can be verified. The intensity of at least three of these pixels should be below or above the intensity of p, since a total of at least 12 pixels has to be reached to classify p as a corner. If this is the case, the other pixels will be verified as well before classifying p as a corner. If less than three pixels have intensities that are above or below the intensity of p then p will be rejected as a corner point.

2.3 Region description

The second step of the BoW model is to represent the regions and points of interest found in step 1 by means of feature vectors. Such a *descriptor* can have different sizes depending on the specific algorithm chosen to create the descriptor. The next sections describe a couple of these algorithms.

2.3.1 Scale invariant feature transform

Scale Invariant Feature Transform (SIFT) is an algorithm to detect features in images and to describe these features as descriptors [Lowe, 2004]. For the features to be reliable, these features should be detectable under all sorts of circumstances such as scaling, noise and illumination change. The SIFT

[2]Image from: [Rosten and Drummond, 2005]

descriptors are invariant to scaling and orientation and partially invariant to noise and changes in illumination. Below the process of creating a SIFT descriptor will be explained.

First, a region around the keypoint is selected in which the gradient magnitudes and orientations of sampling points are calculated, see the left image of Figure 3. The blue overlaying circle, indicates that the values are smoothed by a Gaussian window, to ensure that no sudden changes occur in the descriptor when changing the position of the window a little. Furthermore, gradients closer to the center get more emphasis than gradients near the edges.

Gradient orientation histograms with 8 bins (each representing a gradient orientation) are used to describe the gradients in each of the 4 subregions of the left image. These orientation histograms are shown in the right image of Figure 3. The values of the bins in the orientation histograms, i.e. the lengths of the gradient vectors in the right image, are put in a vector. This vector of orientation histogram bin values is the SIFT descriptor of that keypoint.

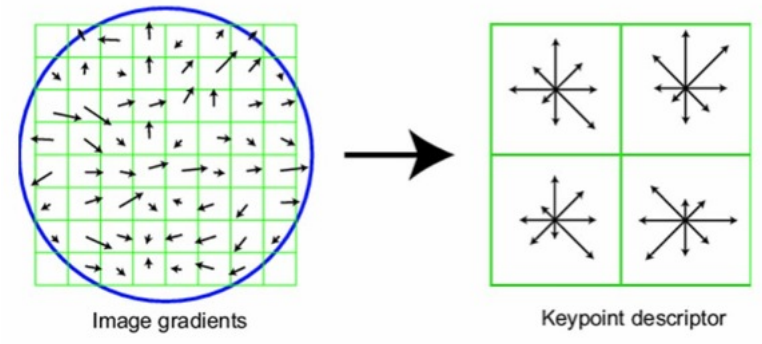

Figure 3: SIFT descriptor construction using orientation histograms. [3]

For Figure 3, the size of the vector would be $2 \times 2 \times 8 = 32$. According to [Lowe, 2004], the best results are obtained when using a 4×4 array of orientation histograms (instead of the 2×2 used above). This leads to a SIFT descriptor of size $4 \times 4 \times 8 = 128$.

2.3.2 Speeded up robust features

Since SIFT descriptors are expensive to create, both in time and memory usage, a speeded up version was needed. This speeded up version was first presented at the ECCV 2006 conference in Graz (Austria) and called Speeded

[3] Image from: [Lowe, 2004]

Up Robust Features (SURF) [Bay et al., 2008]. SURF can be used to detect keypoints and to construct descriptors. In this section, the construction of SURF descriptors will be explained.

The first step of the SURF descriptors construction is to assign an orientation to the keypoint. Instead of looking at the gradient of a keypoint, as is the case for SIFT, SURF uses the distribution of Haar wavelet responses in the x and y direction of a circular neighbourhood around it. By taking the sum of all the Haar wavelets in the x and y direction in a sliding orientation window of size $\frac{\pi}{3}$, a local orientation vector can be constructed, see Figure 4. The longest vector of all the local orientation vectors will be used as the orientation vector for the keypoint.

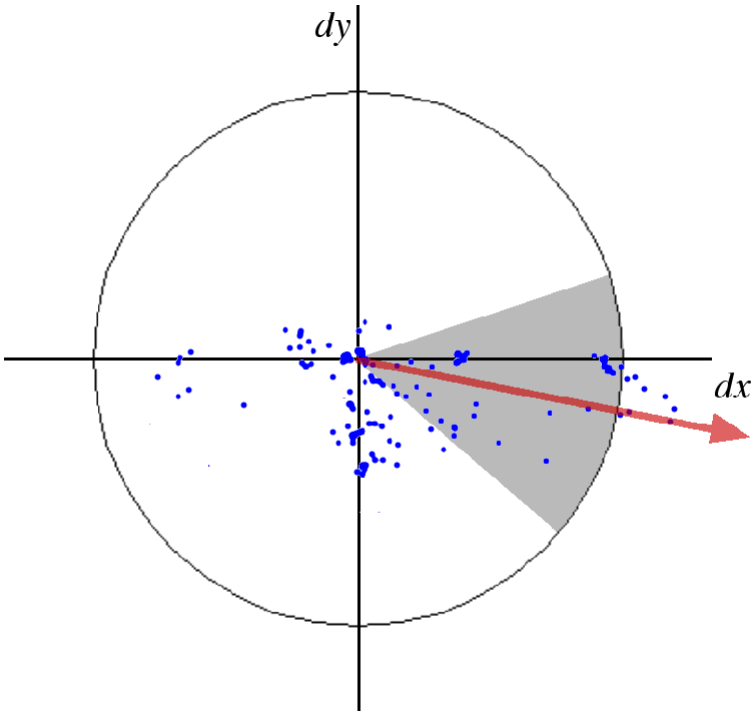

Figure 4: Keypoint orientation assignment using the sum of the Haar wavelets in the x and y direction of a sliding orientation window of size $\frac{\pi}{3}$.[4]

To construct the actual descriptor, a square region is placed on top of the keypoint. This square region is oriented towards the keypoint orientation vector. Next, the square region is divided into 4 subregions of 4×4 resulting in 64 subregions in total. For each of these subregions, the sum of the Haar

[4]Image from: [Bay et al., 2008]

wavelet responses is calculated. These sums represent the 64 values that form the SURF descriptor for the keypoint.

2.3.3 Fast Retina Keypoint

Fast Retina Keypoint (FREAK) is a binary feature descriptor based on the retina from the human visual system [Alahi et al., 2012]. Using binary descriptors instead of SIFT or SURF descriptors has to two advantages. First, the creation of such descriptors is faster and second, less memory is needed.

To construct such a binary descriptor, a sampling pattern is used to compare the intensities of pixel pairs around a keypoint. The sampling pattern used to create FREAK descriptors is similar to the pattern found in the retina. Working from the center of the sampling pattern to the edges, the points or receptive fields first strongly overlap after which they become less and less overlapping, see Figure 5.

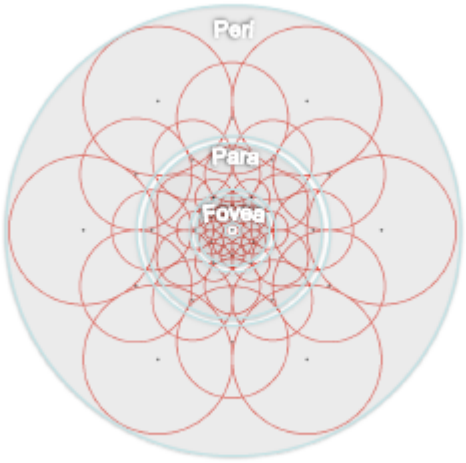

Figure 5: Sampling pattern of FREAK descriptor. [5]

Using 43 of these receptive fields yields about one thousand pixel intensity pairs. As stated in [Alahi et al., 2012], using 512 of these pixel intensity pairs to create the descriptor is optimal. Increasing the number of pixel intensity pairs does not increase performance any further after this amount of pairs. These 512 pairs come from 4 different clusters, with 128 pixel intensity pairs each. As can be seen in Figure 6, the first 128 pairs mostly reside in the outer

[5]Image from: [Alahi et al., 2012]

region of the pattern, while the last 128 pairs can be found at the center. This is ordering is called coarse-to-fine and resembles the behavior of the human eye.

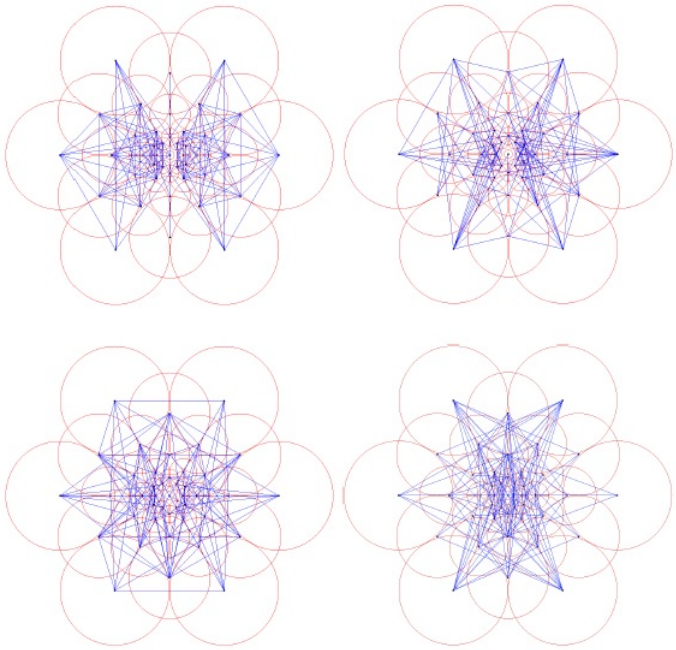

Figure 6: Coarse-to-fine ordering of the 512 pixel intensity pairs used to create a FREAK descriptor, from upper left to lower right (128 pairs per image). [6]

For each pixel intensity pair, a binary value of 1 or 0 has to be calculated to form a string of binary values, i.e. the binary descriptor representing the keypoint at the center of the sampling pattern. The following equation is used to set the value for a pixel intensity pair P_a to either 1 or 0.

$$T(P_a) = \begin{cases} 1 & \text{if } (I(P_a^{r1}) - I(P_a^{r2}) > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where $I(P_a^{r1})$ is the smoothed intensity of the first receptive field.

2.4 Descriptor clustering

The descriptors obtained from the image features could be used to match images. This could be done by comparing all the descriptors of the images. Such

[6] Image from: [Alahi et al., 2012]

a brute force method would be computationally very heavy, since images can contain a lot of descriptors. Instead of comparing all the descriptors, more distinctive elements should be used to compare. Such distinctive elements can be obtained by means of clustering. Clustering is the task of grouping similar objects or datapoints. Given a set of descriptors, clustering would lead to groups or *clusters* of similar descriptors. By considering similar descriptors as one, the number of descriptors to compare between images would be reduced substantially.

Depending on the number of clusters k, the individual clusters will be more or less distinctive. A low value for k will result in large clusters with descriptors that might not be similar at all and a large value for k will results in small clusters with few descriptors each. Two adjacent clusters might actually describe two sets of descriptors that could be considered equal. The number of clusters not only influences the way in which the descriptors are being grouped together, it also influences the time necessary to cluster all the descriptors. A large value for k will be more time consuming than a low value for k. Therefore, the ultimate number of clusters will vary given the dataset and the task.

The BoW model uses the frequencies of words in different documents to compare these documents. Since images do not contain actual words, the centroids of the descriptor clusters will be used as *visual words*. All these visual words together form a vocabulary. Such a vocabulary is built once on a training set of images. This vocabulary can subsequently be used to describe images in terms of visual word frequencies as in the original BoW model. Section 2.5 will elaborate further on how to describe images in terms of visual word frequencies. The next subsection will discuss the k-means clustering algorithm which can be used to find the visual words of a training set of images.

2.4.1 K-means clustering

K-means clustering is a cluster algorithm in which n datapoints are assigned to k clusters. Each datapoint is assigned to the cluster with the nearest mean. Figure 7 shows the four steps of the k-means algorithm. The left image shows the initial step of the algorithm. K random means, three in this case, are chosen in the data space. In the second image, the datapoints are assigned to the cluster whose mean minimizes the within-cluster sum of squares. This corresponds to the squared Euclidean distance and can thus be considered the mean closest to the datapoint. After all the datapoints are assigned to a mean, the cluster centroids of each cluster will be calculated. These cluster centroids will become the new means, as can be seen in the

third image.

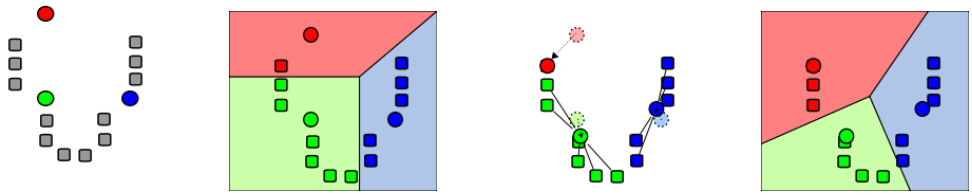

Figure 7: The four steps of the k-means cluster algorithm.[7]

The assignment and recalculation of the means will be repeated until convergence is reached as shown in the right most image. Convergence is accomplished when the means do not change anymore, an optimum is found.

2.5 Frequency vector construction

The visual words found by means of clustering can be used to compare images more efficiently. By describing each image in terms of visual words, the frequencies of these visual words can be compared instead of comparing all the descriptors of the images.

For each image, a histogram is used to represent the visual word frequencies of that particular image. Each bin in this histogram describes a visual word and its frequency. Figure 8 shows an example of three images and their corresponding histograms. As can be seen each histogram is based on the same set or vocabulary of visual words, but all these visual words hold different frequencies per image.

[7]Image from: `http://en.wikipedia.org/wiki/K-means_clustering`

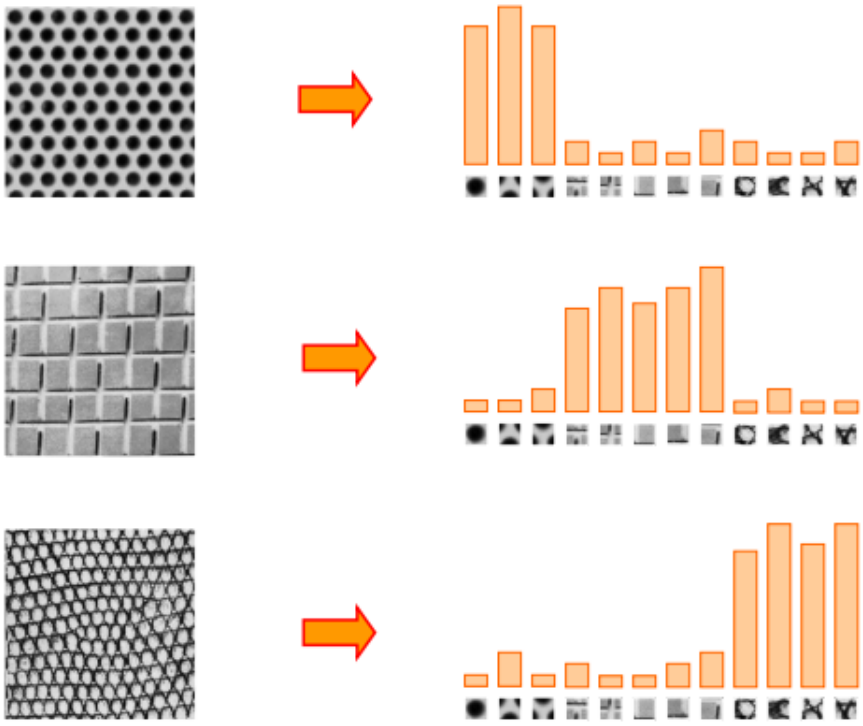

Figure 8: Histograms of visual word frequencies.[8]

To construct such a histogram, first the descriptors of the image have to be extracted. Next, each of these descriptors has to assigned to one of the visual words from the vocabulary. More specifically, to the visual word which is most similar to the descriptor. To determine which visual word is most similar to the descriptor a nearest neighbor search (NNS) can be done. After determining which visual word is most similar to the descriptor, the frequency of that visual word will be increased by 1. Once all the descriptors of the image are extracted and assigned to a visual word from the vocabulary, the construction of the histogram is finished. If these histograms are to be compared, it is necessary to normalize the histograms.

2.6 Frequency vector comparison

The final step of the BoW model is to compare the histograms constructed from the images to determine which images are most similar. Images with many identical or near identical visual word frequencies have a high proba-

[8]Image from: `http://littlecheesecake.wordpress.com/2013/04/24/research-bag-of-features-for-visual-recognition/`

bility to be similar. To calculate the similarity between histograms, and thus between images, different metrics can be used (e.g. Correlation, Chi-Square, Intersection, Bhattacharyya distance).

2.7 Noise reduction

There are multiple methods for improving image retrieval that, just as the bag of words model, originate from the text/document retrieval domain. Below are two of such methods.

2.7.1 Stop list

In the bag of words model for document retrieval, an improvement can be achieved by not using some specific words. Since some words occur in all documents in large amounts (e.g. 'the', 'a',), these words are not descriptive for the document. Not using these high-frequency words can improve the document retrieval accuracy.

To optimize the image matching using this method, a stoplist can be used to discard all the visual words that have very high frequencies. Just as the high frequency words in texts, these high frequency visual words represent parts of images that are very common among images and are therefore not informative.

2.7.2 Spatial consistency

Another method that can be used to improve document retrieval, is the use of spatial consistency. This takes the position of the visual words in the images into account. For every query region that matches to a retrieved region, the k nearest regions to the retrieved region are matched to the k nearest regions to the query region. A voting for these nearest regions takes place. The more nearest regions match, the higher the overall vote of a match of a query region to a retrieved region is. This way the matching regions take their surroundings into account, which should in turn increase accuracy of the similarity of image retrieval.

3 Software implementation

This section will discuss the implementation of the created software. The software can be divided into two parts, which are the PC application and the Android application.

3.1 PC application

3.1.1 C++

The software has been written in *C++*. This programming language was used because of its compatibility with *OpenCV* and the fact that it could be ported to the Android application. Also, since OpenCV was originally written for C++, there was a good amount of documentation and examples for this language. A large part of the image matching application was built using the OpenCV library.

3.1.2 OpenCV

OpenCV stands for Open source Computer Vision and is released under the BSD license. As OpenCV's original focus is to be efficient in real-time applications and it already had many functions implemented that were essential for the application, this library was chosen as framework upon which the application would be built. The application uses binary descriptors from the FREAK algorithm, which increases the speed of the application greatly compared to SURF and SIFT descriptors, but in exchange has lower accuracy. Due to the fact that real-time speed is essential for the application, the FREAK method was also used in the Android application.

3.1.3 FREAK

The functions for FREAK in OpenCV can not detect keypoints within images, hence the FAST keypoint detector was used. FAST is an algorithm that is mainly known for its speed, which is an advantage for the implementation of the Android application. The FAST algorithm needs to have an intensity threshold defined, which can greatly alter the speed/precision ratio. A high valued threshold lets the algorithm retrieve less keypoints from the images, increasing the speed but decreasing the amount of keypoints and thus descriptors extracted from an image. Intuitively, a low valued threshold has the reversed effect. Choosing this value should be done with care, depending on the task the algorithm is used for. Using the keypoints extracted by FAST, the FREAK method can then extract the descriptors. As the cluster algorithm does not accept binary descriptors, they have to be converted to *float*. This conversion causes a slight decrease in clustering speed. However, since the clustering has to be executed only once, while making a vocabulary, this decrease in speed is tolerable. After the clustering the descriptors to visual words, this resulting vocabulary is converted back to binary values.

This conversion back to binary values is necessary for the vocabulary to be usable.

3.1.4 Histogram creation & Image matching

After the vocabulary has been created, it can be used to create histograms of images based on the visual words in the vocabulary. For every image in the testset, a histogram is created and stored into a file. The images for which no descriptors can be extracted, due to a lack of keypoints, are moved to a different folder and are no longer used. Since a lack of descriptors leads to an empty histogram. These moved images are no longer considered possible matches, as they could potentially contain anything.

The image matching itself, loops through the file in which the histograms are stored. Every histogram found in this file are matched to the histogram of the reference image. Since the histogram creation and image matching are separate processes, the histogram creation can be performed offline. This way the real-time matching time is improved. When the testdata or vocabulary changes, the histogram creation will have to be performed again to ensure the right results.

3.2 Android application

The application has been written in Java for the Android OS. In this application, the user can do multiple things. One of these actions should be performed first, on the first run of the application. This is the setting of the path to the images the user wants to do the comparison with. These images will be called *comparison images*. Before the application can be used for image matching, the histograms of the comparison images have to be created and saved in a file on the device. This is done using the button that says "*Update*". If the set of comparison images is large, it is advisable to plug the mobile device into an electric outlet while it calculates and saves the histograms. Next to this button is either a green check mark, or a red cross. The red cross, means that the comparison image histograms are out of sync and that an update is necessary. Conversely, the green check mark means that the comparison image histograms are up to date.

If the update is finished, or the comparison image histograms are already up to date, the user can choose to either capture a new image or to select an existing image to perform the image matching on. Pressing the button with "*New picture*" will start the camera, whereas pressing the button with "*Use existing*" will start the Gallery in which all images on the device are shown.

After the existing or new image is matched with the comparison images,

the existing or new image is displayed together with a list of matches and the percentage of similarity. Clicking one of these matches will show the image that corresponds to the clicked match.

Multiple parts of the application have been written in *C++*. The writing of the comparison image histograms, the loading of a vocabulary with visual words and the matching of the images itself is all written in C++. These parts are connected to the Java Android application through the *Java Native Interface* (JNI). The JNI is a programming framework that enables Java code to call native applications, such as C++. It lets native methods use Java objects in the same way that Java code uses these objects, which can be returned to the Java code as well. The reason JNI and C++ were used is because of the OpenCV Library, which is written in C++.

The application seems to have a slight deviation when calculating the similarity. When matching an existing image while this image is also in the comparison image set, one would expect to find a similarity of 100% between the image and itself. However this is not the case. Even though the similarity is the highest of all the matches and almost 100% it is slightly less. Also, this similarity is not constant. It fluctuates around a certain similarity when matching the same image multiple times. On each matching iteration the same amount of keypoints was found in the images. After the keypoint detection, these keypoints are used to create detectors and in turn histograms. However this detector and histogram creation is done in an OpenCV function, which creates a black box effect. Since the descriptor extraction should be constant, the only logical explanation for this similarity fluctuation is that the creation of the histograms, which is done by nearest neighbor between descriptor and visual words, is not done invariably.

4 Software evaluation

Before the software can be presented to a user, a software evaluation has to be done. Given the mobile device application, two criteria are important to evaluate. First, performance: how good does the application match images and second, speed: how fast can the application perform its task on such a mobile device. How these criteria are evaluated and on which dataset will be explained in the following subsections.

4.1 Datasets

To evaluate the software, an image dataset is required with annotations about the similarity between the images. That way, we can compare the results with a ground truth based on the annotation of the dataset. One such annotated image dataset is the INRIA Holidays dataset [Jegou et al., 2008]. The dataset consists of 1491 holiday images from different scenes, e.g. natural, man-made, water and fire effects [9]. The 1491 images are divided in 500 groups and each group consists of 1 query image and 1 or more corresponding similar images. These similar images can be rotations, different viewpoints, different illuminations or blurred versions of the query image. Figure 9 shows an example of an image group from the INRIA Holidays dataset.

Figure 9: An image group from the INRIA Holidays dataset. The query image on the left and the corresponding similar image on the right.

A query image is always the first image of a group, e.g. 104300.jpg. The corresponding similar images are the remaining images of the group, for the previous example this would be 104301.jpg. All the images in the dataset are high resolution images.

A second dataset is used to create the visual word vocabularies. Based on such a vocabulary the images are represented in terms of visual words making it more easy to compare them as described in Section 2.5. Using the image

[9] Can be downloaded at: `http://lear.inrialpes.fr/~jegou/data.php`

dataset of *The PASCAL Visual Object Classes Challenge* [Everingham et al.] different vocabularies are constructed to evaluate the effect of the vocabulary on the accuracy as will be elaborated on in the next section. The dataset consists of 5011 images, all matching one of twenty object classes, e.g. person, animal or vehicle [10].

The INRIA dataset offers a software package [11] to evaluate the results obtained from testing your image matching software on the dataset. This package consists of Python code to calculate the mean average precision of a result file and several result files produced and evaluated as described in [Jegou et al., 2008]. By using the same evaluation software and query dataset, the results produced with our matching software can be compared with the results from Jegou et al.

In order to evaluate the precision of the matching software on the 500 query images, a file is used which contains all the histograms from the test images, i.e. all 1491 images from the INRIA set. This way, the histograms have to be constructed once and can subsequently be accessed from the file to compare a query image with all the images from the set. If such a file would not be used, every time a query image is compared, the histograms of all the images have to be constructed again. Given our set of query images, that would be 500 times 1491 histograms. Therefore, the use of this file speeds up the matching process substantially.

The software evaluation is conducted on the PC application. It is assumed that the degree of performance on the PC application holds for the Android application, e.g. if a total of 1000 clusters yields the best performance on the PC application this will be the case for the Android application as well.

4.2 Performance evaluation

To evaluate the performance of the software, a measure called Mean Average Precision (MAP) is used. For each query image, the average precision (AP) is calculated as shown in (1).

$$AP = \frac{\sum_{r=1}^{N} P(r) \times rel(r)}{C}, \tag{1}$$

where N is the total number of ranked images, $P(r)$ the precision at rank r, see Equation (2), $rel(r)$ a binary relevance indicator (0 for an incorrectly ranked image and 1 for a correctly ranked image) and C the total number of correctly ranked images.

[10] Can be downloaded at: `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/#testdata`

[11] Can be downloaded at: `http://lear.inrialpes.fr/~jegou/data.php`

$$P = \frac{TP}{TP + FP}, \tag{2}$$

where TP is the number of correctly ranked images and FP the number of incorrectly ranked images. Using the average precisions of the query images, the MAP can be calculated as in (3).

$$MAP = \frac{\sum_{rq=1}^{Q} AP(q)}{Q}, \tag{3}$$

where $AP(q)$ is the average precision of query q and Q the total number of queries.

4.2.1 FREAK: baseline implementation

As in [Jegou et al., 2008], MAP scores of different parameter combinations will be compared to evaluate the performance of the image matching software. Table 1 shows the baseline results of Jegou et al. on the INRIA dataset. Descriptors are obtained by using a Hessian-Affine detector and a SIFT descriptor extractor. K-means clustering on the independent Flickr60K dataset is used to create the visual word vocabularies. The number of clusters, i.e. the number of visual words, k will be specified for each experiment.

	k = 20000	k = 200000
Baseline	0.4463	0.5488

Table 1: MAP scores for the baseline implementation of [Jegou et al., 2008]. The implementation is based on a Hessian-Affine detector and a SIFT descriptor.

Table 2 shows the MAP scores for our baseline implementation (default parameters) and the MAP scores obtained by using different histogram matching metrics (MM). FAST is used to detect keypoints and FREAK is used to create descriptors. FAST offers the possibility to specify the threshold t, which describes the amount of pixel intensity difference that is allowed between a pixel P and its neighbours to be classified as a corner, as described in Section 2.2.3. All images from the INRIA dataset from which no keypoints could be detected with the default value of $t = 30$ are automatically removed from the set. These removed images include, for example, blurred images from which keypoint extraction is difficult. All our results are therefore based on a subset of 1481 images of the INRIA dataset.

	k = 1000	k = 5000
Baseline (MM = correlation)	0.3060	0.2882
MM = chi-square	0.2047	0.2038
MM = intersection	0.2636	0.2661
MM = Bhattacharyya	0.2206	0.2228

Table 2: MAP score for the Freak baseline implementation and MAP scores obtained by using different histogram matching metrics (MM).

As can be seen, the baseline implementation with correlation as histogram matching metric and 1000 visual words performs the best (MAP = 0.3060). Comparing this MAP with the baseline MAP from the experiments from Jegou et al., our implementation scores substantially lower. Several implementation aspects can be the cause of this difference in MAP.

1. Different feature detection algorithms are used (FAST versus Hessian-Affine)

2. Different descriptor construction algorithms are used (FREAK versus SIFT)

3. Due to memory and time restrictions of the computers used, the number of clusters k is much lower in our implementation (1000 and 5000 versus 20.000 and 200.000).

4. Due to memory and time restrictions of the computers used, the number of descriptors used to create the visual word vocabulary is much lower in our implementation ($\pm$ 300.000 (baseline) and 600.000 versus 140M descriptors).

5. Due to memory and time restrictions of the computers used, the images used from the INRIA set are resized (800×600 versus 1024×768). This reduction in image size results in less descriptors and possibly less descriptive ones. After all, the resolution decreases and features in images will be more difficult to detect.

4.2.2 FREAK: more descriptors

One of the possible explanations of the lower MAP of our software compared to the software of Jegou et al. was the amount of descriptors used to create the vocabularies of visual words. For this reason the performance evaluated again, but this time with twice the amount of descriptors per image (baseline is sixty descriptors per image). In order to evaluate the effect of increasing

the number of descriptors obtained from the images, the other parameters are kept the same as for the results in Table 2. The effect of the increment of descriptors on the MAP is shown in Table 3.

	k = 1000	k = 5000
DPIm = 120 (MM = correlation)	0.3025	0.2538
MM = chi-square	0.2505	0.1304
MM = intersection	0.2917	0.2212
MM = Bhattacharyya	0.2481	0.1505

Table 3: MAP scores for the Freak implementation with more descriptors obtained per image (Descriptors Per IMage = 120) and the effect on the MAP of different histogram matching metrics.

Unfortunately, the MAP does not increase when using more descriptors the create the visual word vocabulary. An increase in performance can be intuitively expected when using more descriptors per image. For this reason, the decrease in performance was unexpected.

4.3 Speed evaluation

Since the image matching application will be used on a mobile device and the user does not want to wait too long to be able to get the matching results, it is important to verify the time it takes for the application to calculate the keypoints of an image, to construct descriptors and to create a histogram. Table 4 shows the average time of this process on the PC application and the Android application on a mobile device.

PC application	Android application
7.67 s	3.69 s

Table 4: Average time in seconds per image to detect keypoints, construct descriptors, create a histogram based on the visual word vocabulary and write the histogram to the histogram file on the PC and the Android application on a mobile device.

The above results show that the Android application processes the images faster than the PC application. One might actually expect the opposite, since the processor of a PC is usually more powerful than the processor of a mobile device. Looking at the amount of processor capacity being used on the PC, it appears that only about 20% of the capacity is used. This

could be a possible explanation for the unexpected lower speed of the PC application. However, since the application has to run on a mobile device, these unexpected results are an advantage.

5 Discussion

Although the results of the FREAK implementation of the application were already quite good, there are still improvements that could be made and other options to be tested in both the PC and Android application.

5.1 Future work: PC application

Apart from the application using FREAK, an implementation of image matching using the SURF algorithm was also developed. However, this implementation did not function with the large dataset that was used with the FREAK implementation of the application. The implementation using SURF issued an "Failed to allocate memory" error. This indicates a possible memory leak. If this memory issue could be resolved, the application using SURF could produce result files and could in turn be evaluated. The SURF algorithm should produce more accurate results, but also take a longer time than the FREAK algorithm. Still, it would be worthwhile to try and improve the time complexity of SURF and the performance of FREAK.
Also, as described in Section 4, Jegou et al. used considerably larger amounts of descriptors and clusters. The results could improve if these amounts of descriptors and clusters are used. However, taking into account the amount of time the current processed descriptors and clusters have taken, it should be considered to increase the processing power of the device running the application.
Some other improvements of the results could be achieved by implementing the methods described in Section 2.7, the stoplist and the spacial consistency. These methods should improve the performance of the application and could also be implemented for the Android application.

5.2 Future work: Android application

As it is, the Android application has all functionality necessary. However, there are multiple aspects that could be improved. One of these aspects is the catching of certain occurrences. For example, if in an image no keypoints are found it will still try to match this image, even though it would be better to notify the user that something went wrong and that he should take a

better picture. These type of occurrences do not happen a lot. An almost completely black image with just a small corner of dark objects will still produce keypoints. Even so, it would be more appropriate to catch these occurrences.
Another aspect is the aesthetics of the application. The calculation of the histograms and saving them to a file can take a long time. In the version of the application, the screen becomes black in the time it is calculating and saving the histograms. It would be aesthetically better if some sort of loading animation could be included, to let the user know that the application has not crashed, but is still running.
The phenomenon that is reported in Section 3.2, of the same images not returning a 100% similarity, could be attempted to be solved in the future as well, since two identical images should of course return a 100% similarity.

6 Conclusion

An application has been developed for both the PC and for a mobile device running on Android OS. With the mobile application being based on the PC application. The PC application has been evaluated and compared to the results of the INRIA paper by [Jegou et al., 2008]. Keeping in mind that, although FREAK is reportedly faster than SIFT, and FREAK is also less accurate than SIFT, the results of the PC application (0.3060 MAP) are acceptable in comparison to the INRIA results (0.4463 MAP).
Even so, there are still possibilities in improving both the PC application and the Android application in the future.

References

Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517. IEEE, 2012.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110 (3):346–359, 2008.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Computer Vision–ECCV 2008*, pages 304–317. Springer, 2008.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767, 2004.

Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005.

Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4):591–606, 2009.