Universiteit Leiden
Computer Science

A Comparative Performance Analysis of Feature Description Algorithms Implemented in OpenCV

Name: Nels Numan

Student number: s1459929

Date: August 20, 2017

1st supervisor: Michael Lew
2nd supervisor: Erwin Bakker

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

A Comparative Performance Analysis of Feature Description Algorithms Implemented in OpenCV

Nels Numan

Abstract

In this thesis, we perform a comparative analysis of feature description algorithms implemented in the OpenCV [Ope17] framework. Feature description algorithms have been an important topic in computer vision for the last few decades, which has resulted in many approaches to encoding information in local regions of interest. In this thesis we propose an evaluation framework that evaluates the performance of SIFT [Low04], SURF [BTVG06], BRIEF [CLSF10], ORB [RRKB11], BRISK [LCS11], FREAK [AOV12] and LATCH [LH16], under several kinds of geometric and photometric transformations. We define a performance metric, *matching score*, that shows the ratio between correct and false feature matches, and measures the consumed time and memory of the algorithms. We observe that BRIEF [CLSF10] performs better than expected under photometric transformations, and that FREAK [AOV12] and BRISK [LCS11] are the best overall performers among the evaluated algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the last decades, the ability to effectively register and recognize image data in computer vision has been a widely explored topic. It has led to a surge in applications such as object recognition [TCGP09], panorama stitching [BL07], and image classification and retrieval [LSDJ06], where large databases of images are continuously evaluated to find the most relevant result to a given query image. These applications have expanded to mobile platforms, which currently possess lower computational and storage capabilities than those of conventional desktop computers, which in turn increases the importance of low resource usage that these methods use.

All of the mentioned applications rely on an efficient and accurate generation of features, where local regions of interest of an image are decomposed and quantified. This process typically involves two major sub-processes: the detection of keypoints and the description of those keypoints, which are called feature descriptors. In the phase where keypoints are detected, local regions of interest are identified based on their properties such as being located on an edge or a corner. Depending on the implementation, the keypoint scale and orientation are measured as well.

In the feature description phase, the image data around the keypoint is extracted, characterized and stored in a feature vector. Obtaining a distinctive set of feature vectors of two images allows us to establish matched between them, as can be seen in Figure **??**. This matching process is done by a feature matcher, which matches feature vectors based on their value and distance defined by the feature descriptor. In order to form an accurate and invariant feature vector, we need a method to detect image features and a strategy to encode and characterize this interesting information. The requirement of highly distinctive and invariant descriptors has driven the development of feature descriptors that are able to efficiently generate feature vectors that are invariant to geometric and photometric image transformations. These developments have led to a wide range of proposed approaches to feature description. However, it is unclear what descriptor

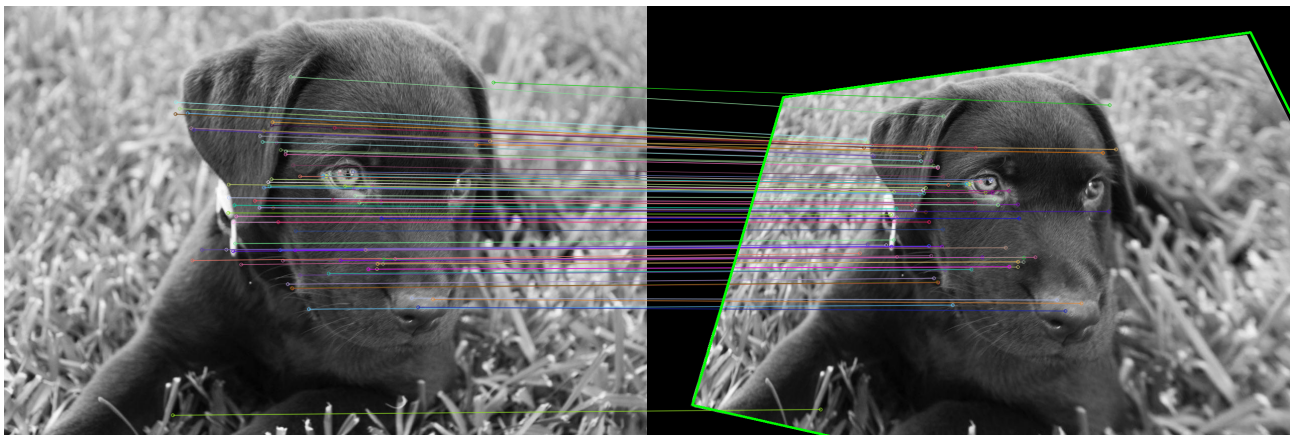


Figure 1.1: An example of matches between a source image (left), and its rotated counterpart using feature vectors detected and computed with the SURF algorithm. The matches in this image are filtered based on their distance between the descriptors.

algorithms should be used for which applications, as each requires a different level of accuracy and resource usage. In this thesis, we will be looking at the performance and computational and storage requirements of a number of feature descriptors supplied by the OpenCV [Ope17] framework. We will take a large data set of images and will make a number of comparisons between the source images and its transformed counterparts. The transformations that will be evaluated are scale change, rotation, Gaussian blur, illumination change, 2D perspective change and a combination of scale change and rotation.

To provide a comparative analysis we have defined a performance metric, *matching score*, which shows the ratio between correct matches and false matches for a pair that consists of a source image and a transformed image. To evaluate the resource usage of the feature description algorithms, we will measure the computation time and memory consumption while feature vectors are being generated.

In the following chapters we will firstly examine the approach and structure of each algorithm that we will evaluate in our framework. Following this, we will accurately describe the data set and performance metrics that we will be using in our evaluation. We will then proceed to the comparative analysis of the experiment's results, and will finally discuss our observations and conclusions.

Chapter 2

Related Work

As mentioned in the introduction of this thesis, feature detection and description and their performance have been a popular topic in computer vision for years. One of the first and most cited papers that evaluated feature descriptors, including the state of the art feature descriptor SIFT [Low04], was published in 2005 by Mikolajczyk et al. In this paper [MS05], the authors presented an experimental evaluation of scale and affine invariant feature descriptor algorithms by comparing their feature vectors. The evaluation of the algorithms was performed in the context of matching the same scene observed under different viewing conditions, such as scale change, rotation and blur. Most of which we also evaluate in this thesis. Mikolajczyk et al. defined two important metrics, 1-precision and recall, from which we derived the metric we use in our own evaluation: *matching score*. The authors also introduced their own feature description algorithm in the same paper, which returned the best results in their evaluation. In the period after the publishing of the paper by Mikolajczyk et al., their dataset and performance metrics have become widely used by papers that propose new feature description methods [LCS11, LH16]. In 2012 and Miksik and Mikolajczyk [MM12] performed similar experiments on more recent feature descriptors: BRIEF, BRISK, ORB, MRRID, MROGH and LIOP.

With the growth of applications of computer vision and feature description on mobile devices such as phones and tablets, the emphasis on minimizing computational and storage requirements became increasingly important. For this reason, J Heinly et al. have performed a performance analysis of binary feature descriptors BRIEF, ORB and BRISK. In their analysis, they tested the individual effects of several transformations to gain a better understanding of binary description algorithms and how they can best be combined with feature detection algorithms. The authors found that BRIEF's performance was most favorable but fell short under geometric transformations. For geometric transformations they found that SIFT had the best performance.

In 2008 a paper was published by Thomee et al. that focused on the detection accuracy and description time of various methods on a set of realistically transformed images embedded into a collection of over one million web images. The authors found that it is not always necessary to use a computationally intense feature descriptor to obtain high accuracy. They found SURF [BTVG06] performs poorly when using a small number of interest points.

Chapter 3

Examination of Feature Descriptors

As feature description has been a very relevant topic within the field of computer vision, a wide variety of approaches for evaluating features have been developed over the years. In this section, numerous approaches that are part of the OpenCV [Ope17] framework are examined for finding distinctive descriptors which will be further evaluated based on their performance in Chapter 5.

3.0.1 Scale-Invariant Feature Transform (SIFT)

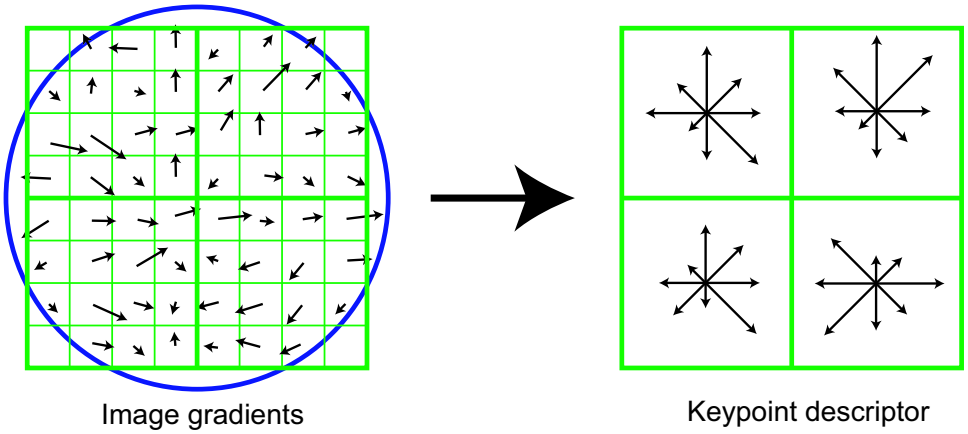
Scale-invariant feature transform (SIFT) was proposed by Lowe [Low04] and extracts and describes features that are highly distinctive and are scale and rotation invariant.

Firstly, to obtain the feature descriptors a rectangular grid, centered at the position of the keypoint, is laid out over the image. For each sample element of this grid, the gradient magnitude and orientation is calculated, as shown on the left in Figure 3.1. As can be seen by the circle, the sample information is weighted by a Gaussian window, which limits the effect of minor positional changes of the grid and decreases the prominence of the outer data. To achieve rotation invariance, the information in this grid is rotated to correspond to the orientation of the respective detected keypoint.

To finally obtain the feature descriptor, the sample data is used to create 4×4 sub-regions, all of which contain an 8-bin histogram. Weighted by their magnitude, each sample point is put into its appropriate bin. The bins each represent a range of degrees between 0° and 360°. Lowe [Low04] has found that a 4×4 histogram with 8 orientations performs best in the average case, which results in a vector of size $4 \times 4 \times 8 = 128$. This vector contains floating point values which each take 4 bytes to store. This means each vector will take up 512 bytes of memory in order to store it.

Lowe [Low04] found that a 4×4 histogram with 8 orientations performs best in the average case, which is

Figure 3.1: Centered around the keypoint location, the gradient magnitude and orientation of all the elements in the 16×16 sample array are calculated, as shown on the left. The sample array is used to create orientation histograms, divided by 4×4 subregions. In this example image, we use a 8×8 sample array with 2×2 subregions.



why we will use this configuration in our experiments. To reduce the effects of illumination changes, the sample vector has to be normalized to be relative with respect to each other. The vector is normalized to unit length, which results in a vector of which all the sample points have a joint sum of 1. Consequently, a change in illumination will theoretically not affect the level of distinctiveness of a SIFT descriptor.

3.0.2 Speeded Up Robust Features (SURF)

The SIFT algorithm described in the previous section is excellent at describing locally affine pieces of images [MS05], but has one major drawback. The amount of time computing the histograms with gradient magnitude and orientation requires a lot of time and resources. SURF [BTVG06] was proposed to improve this downside of SIFT, while maintaining its key properties. SURF uses integral images to speed up the computation and in essence, is an approximation of the SIFT algorithm.

Orientation Estimation

Before the construction of the SURF descriptor is started, the orientation of each keypoint is defined to achieve rotation invariance. To achieve this, a dominant orientation is defined by considering the local gradient orientation distribution, estimated based on Haar wavelet responses. The Haar wavelets for the horizontal direction x and vertical direction y are computed with a neighborhood radius of $6s$ [BTVG06], where s is the scale that the keypoint was detected. The size of the wavelets is relative to the scale of the keypoint as well and equals $4s$.

Figure 3.2 shows the Haar wavelet filters that are applied to the integral image, which can be seen as a matrix that makes up a two-dimensional lookup table [Cro84]. An integral image is a cumulative addition

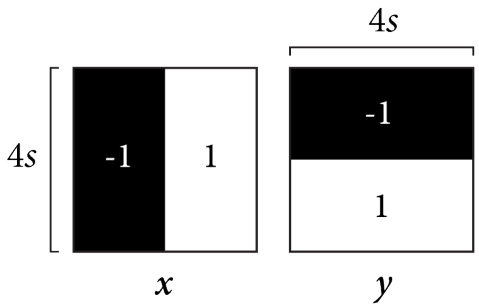


Figure 3.2: Haar wavelet filters to compute the responses in x (horizontal) and y (vertical) direction.

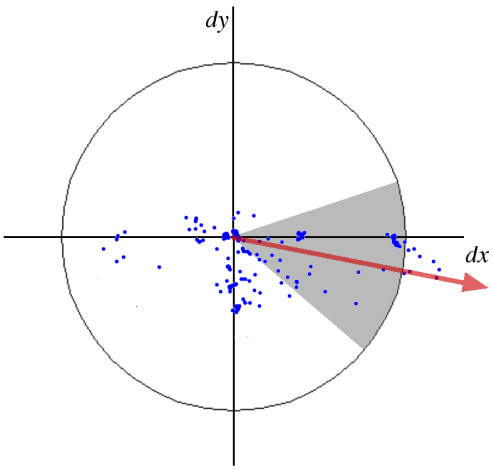


Figure 3.3: The dominant orientation (see red arrow) of the keypoint is calculated by taking the sum of all responses within the sliding orientation window (marked by the dark gray area) [BTVG06]

of intensities on subsequent pixels in both horizontal and vertical axis, which enables the computation of the sum of pixel values at any scale or position using only four lookups. After the responses are weighted by a Gaussian filter ($\sigma = 2s$) centered at the interest point, the wavelet responses are positioned in the coordinate plane as points, see Figure 3.8. The dominant orientation is then estimated by calculating the sum of all responses within a sliding orientation window of 60 degrees, which is marked dark grey in Figure 3.8;

Constructing the Descriptor

The calculation of the descriptor starts by laying a square region centered at the position of the keypoint, with the orientation as calculated in the previous paragraph and size $20s \times 20s$. Following this, the region is split up into 4×4 sub-regions. For each of these 16 sub-regions, the Haar wavelet responses are computed at 5×5 evenly spaced out sample points. Then, the x and y wavelet responses (d_x, d_y) are collected for each sub-region: $v = \sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|$. The combination of the sum of values and sum of absolute values for each direction demonstrates the intensity of the polarity of the sub-regions, and together form the descriptor for all 16 subregions of length 64, which takes 256 bytes to store. Lastly, the feature vector is normalized to achieve invariance to contrast changes.

3.1 Binary Descriptors

As shown in the previous section, patch descriptors such as SURF and SIFT use gradient operations which can leave a large memory and computing time footprint. Binary descriptors approach this in a different way, and were introduced to avoid these costly operations. Binary descriptors use pixel comparison operations to generate short binary strings. A clear advantage of binary features is that, on a modern CPU, it takes fewer resources to match binary features, as the Hamming distance between two features is used to match features, rather than the more resource-consuming Euclidean distance [CLSF10].

A binary descriptor is constructed with bits which present the outcome of the the binary test:

$$\tau := \begin{cases} 1, & \text{if } p(x) < p(y) \\ 0, & \text{if } p(x) \geq p(y) \end{cases} \tag{3.1}$$

where function $p(x)$ presents the intensity of point x on patch p. To achieve this, sample points must be chosen, the dominant orientation of each keypoint must be defined, and a way to decide which sample pairs to compare must be determined. Therefore, most binary descriptor algorithms consist of a sampling pattern, a way of deciding the dominant orientation of a keypoint, and $(x - y)$-pair selection process. A feature is constructed of a vector n filled with results of binary tests:

$$f_n(p) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(p; x_i; y_i) \tag{3.2}$$

In the following subsections, different approaches to this problem will be explained.

3.1.1 Binary Robust Independent Elementary Features (BRIEF)

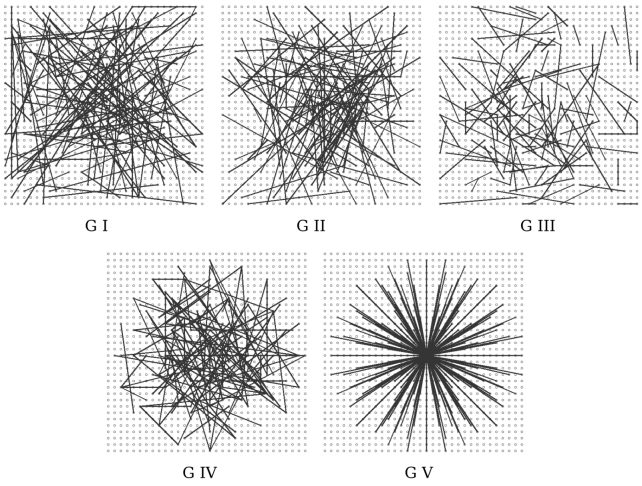
BRIEF [CLSF10] was the first binary descriptor that was published and takes a relatively simple random approach. Considering BRIEF takes information for the sample points at single pixels, it's very sensitive to noise. In order to improve stability, the algorithm first smooths the image patch using a Gaussian filter. BRIEF does not use a sampling pattern, but uses other means of deciding the spatial arrangement of the sample points (x_i, y_i) within a patch with size $S \times S$. Calonder et al. have experimented with five different methods [CLSF10], which are illustrated in Figure 3.4:

G I The locations for (x_i, y_i) are randomly sampled without any restriction,

G II The locations for (x_i, y_i) are randomly sampled based on an isotropic Guassian distribution,

G III The locations for (x_i, y_i) are randomly sampled where the first location x_i is sampled from a Gaussian

Figure 3.4: Five sampling strategies considered by Calonder et al. [CLSF10]



centered around the origin while the y_i location is sampled from another Gaussian centered on x_i.

G IV The locations for (x_i, y_i) are randomly sampled from discrete locations of a coarse polar grid.

G V The location of x_i is centered, while y_i takes every possible location on a coarse polar grid where the amount of points is equal to the size of the descriptor in bytes.

Calonder et al. [CLSF10] found that all strategies except for G V performed very similarly. With G II slightly standing out, the authors picked this strategy for the implementation of BRIEF.

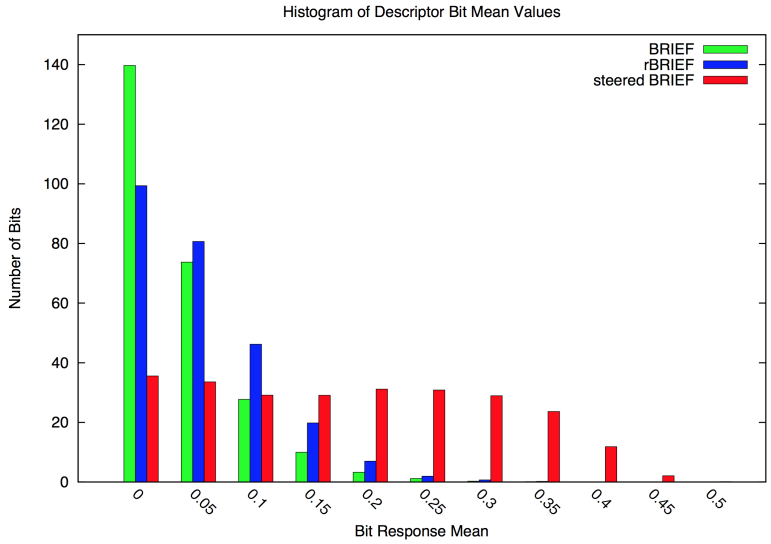
3.1.2 Rotation-Aware BRIEF (rBRIEF)

rBRIEF got its name from being a rotation invariant version of the previously discussed BRIEF descriptor algorithm and was developed alongside an oriented version of the FAST detector algorithm (oFAST), which combined is called *Oriented FAST and Rotated BRIEF* (ORB). Considering the focus of this paper on the evaluation of descriptor algorithms, only the descriptor extracting process of ORB will be discussed. The authors of ORB have made two major changes to the BRIEF descriptor which will be discussed in the following paragraphs.

Analysis of variance and correlation of oriented features

As the keypoint detection algorithm oFAST in ORB provides keypoints that have a given orientation, the authors of ORB [RRKB11] needed to find a way to utilize this useful information. To achieve this, they introduced steered BRIEF, which defines a matrix S of the size $2 \times n$, where n is the size of the vector of binary tests performed for each feature. rBRIEF constructs a steered version of this matrix using the orientation of

Figure 3.5: Distribution of means for feature vectors: BRIEF, steered BRIEF, and rBRIEF [RRKB11]



the keypoint region θ and its corresponding rotation matrix R_θ. This means the steered BRIEF descriptor is constructed as follows:

$$g_n(p,\theta) := f_n(p)|(x_i, y_i) \in S_\theta \tag{3.3}$$

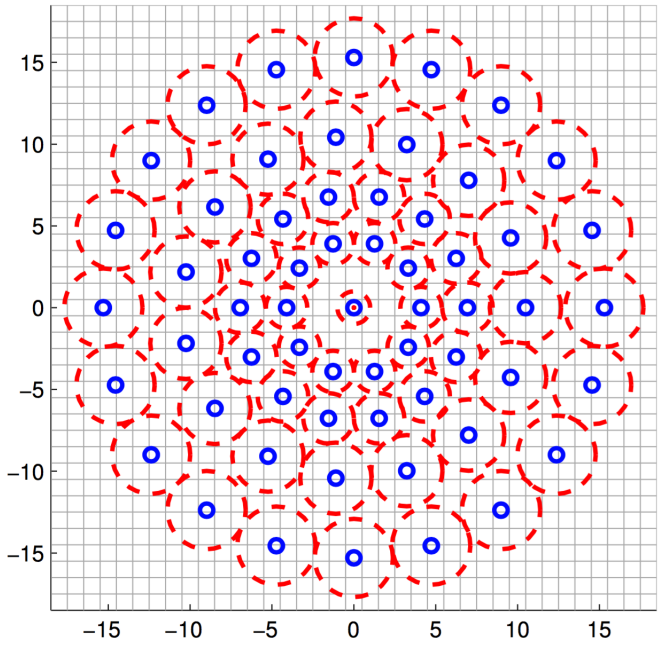
A key property that sampling pairs should have is high variance, which results in a more discriminative feature. Besides this, it is preferable for sampling pair tests to be uncorrelated, so that each sampling pair brings new information to the test set. As stated in [RRKB11], one of the pleasing properties of BRIEF is that each feature test vector has a large variance and a mean near 0.5, which results in a favorable amount of variance and uncorrelation.

However, once this data is steered by R_θ the mean values of the result sets of the features become more distributed as can be seen in Figure 3.5. This means BRIEF's approach of randomly selecting sample pairs is not a good choice. Instead of this, the authors of ORB have implemented a learning method for the sampling pairs. The learning is done with a training set of keypoints drawn from the PASCAL [EZW+06] 2006 set, and an enumerated set of all possible sampling pairs. Following this, a greedy algorithm is applied to obtain a set of 256 sampling pairs with means near 0.5. Once this algorithm has terminated and has obtained the best sampling pairs, the generation of the descriptor is complete.

3.1.3 Binary Robust Invariant Scalable Keypoints (BRISK)

The Binary Robust Invariant Scalable Keypoints [LCS11] (BRISK) algorithm takes a different approach than both BRIEF and ORB. BRISK aims to expand on the quality of binary features, by defining the characteristic direction of each keypoint and taking a unique approach of selecting samples and sample pairs.

Figure 3.6: Sampling pattern for $N = 60$ used by BRIEF [RRKB11]



BRISK uses a particular sampling pattern which can be seen in Figure 3.6. For each sample point, we take a region with size σ of the standard deviation of that is applied to each sample point. Two types of sampling pairs (p_i, p_j) are used, short pairs and long pairs. The set of short pairs S is defined by sample points that have a threshold below d_max, and the set of long pairs L are sampling pairs that have a distance above d_min where $d\_min > d\_max$. The exact definitions of these sets can be seen in equations 5.4 and 5.5.

$$S = \{(p_i, p_j) \in A | \parallel p_j p_i \parallel < \delta_{max}\} \subseteq A \tag{3.4}$$

$$L = \{(p_i, p_j) \in A | \parallel p_j p_i \parallel > \delta_{min}\} \subseteq A \tag{3.5}$$

BRISK uses long sample pairs to estimate the rotation of the descriptor and the short sample pairs are used for binary intensity tests like we have seen in other binary descriptor algorithms.

Orientation Estimation

To estimate the orientation of keypoints, BRISK [LCS11] uses local gradients, see equation 3.6 where $g(p_i, p_j)$ defines the local gradient for the sampling pair (p_i, p_j). I defines the intensity of the respective sample points, smoothed by a Gaussian filter with a standard deviation σ proportional to the distance between the two sample points.

$$g(p_i, p_j) = (p_j - p_i) \cdot \frac{I(p_j, \sigma_j) - I(p_i, \sigma_i)}{\| p_j - p_i \|^2} \tag{3.6}$$

To finally compute the orientation of the keypoint, the sum of all local gradients of the long pairs g is taken, and the arctangent y-component of g by the arctangent of the x-component of g: $\alpha := arctan2(g_y, g_x)$.

Building the Descriptor

As we have seen with other binary feature descriptors, the construction of a feature is done by performing binary tests that compare the intensity of each point in a sample pair. BRISK assembles a descriptor d_k by performing the following for each sample pair:

$$b := \begin{cases} 1, & \text{if } I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0, & otherwise \end{cases} \tag{3.7}$$

Note that in this equation the keypoint orientation α is used, the intensity of both sample points in smoothed by a Gaussian filter and only the short sample pairs are used. A collection of 512 comparisons makes up the feature vector generated by BRISK, which is stored in 64 bytes.

3.1.4 Fast Retina Keypoint (FREAK)

The Fast Retina Keypoint (FREAK) descriptor [AOV12] suggests to take inspiration of the human retina to create a retinal sampling grid, where the density of the sample points decrease exponentially as we near the center (see Figure 3.7). Each of these sample points is smoothed with a Gaussian filter to decrease sensitivity to noise. In Figure 3.7 the radius of the red circles around each sample point illustrate the size of the standard deviation of the Gaussian kernel.

The authors of FREAK have considered multiple ways of creating the sample pairs to describe an image patch. One possibility is to use the approach of BRISK [LCS11], where pairs were selected based on spatial distance as was discussed in the previous subsection. Rather than this approach, the authors have opted for the strategy used by [RRKB11] where learning is used to obtain pairs that are more uncorrelated and discriminant. Please refer to subsection 3.2.2. for the explanation of this concept.

The process of learning to maximize variance and uncorrelation brings an interesting structure of the resulting sample pairs to light. The first sample pairs that are selected are mainly located in the outer rings of the

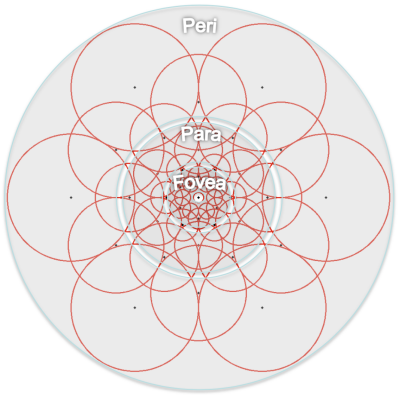


Figure 3.7: Sampling pattern used by FREAK [RRKB11]

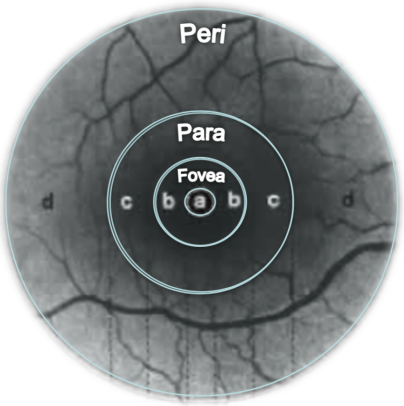
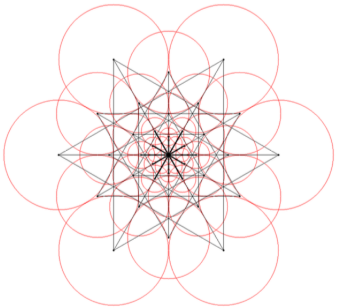


Figure 3.8: The human retina [HAW71]

Figure 3.9: The 45 predefined pairs used by FREAK to determine orientation [AOV12]



sampling pattern, while the last few selected pairs are mainly located in the inner rings of the sampling pattern. This behavior resembles our understanding of the model of the human retina.

The coarse-to-fine structure allows FREAK to have an advantage during the matching process. At first, while comparing two descriptors, just the first 128 bits of each descriptor have to be compared. If this comparison shows potential by showing the distance between these two sets of bits is low, the next set of 128 bits of each of the descriptors is compared. This allows for a large portion of the matching candidates to be discarded without having to compare the full descriptors first.

The orientation assignment approach of FREAK is similar to that of BRISK [LCS11] which is discussed in section 3.2.3. FREAK uses a predefined set of pairs to determine the orientation BRISK uses long distance pairs for orientation assignment, see Figure 3.9.

Just like BRISK, a collection of 512 comparisons makes up the feature vector generated by FREAK, which is stored in 64 bytes.

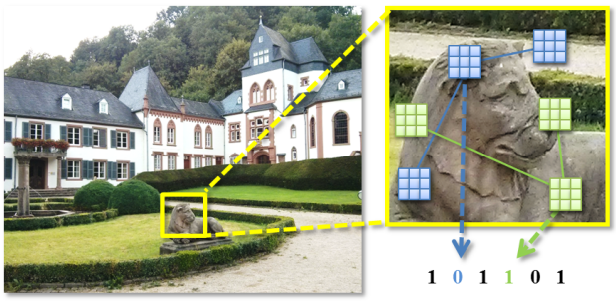


Figure 3.10: A visualization of the LATCH descriptor [LH16]

3.1.5 Learned Arrangements of Three Patch Codes (LATCH)

The Learned Arrangements of Three Patch Codes [LH16] (LATCH) descriptor was introduced in 2015 by Levi and Hassner. The authors build upon the concept of other binary descriptors, but instead of using pixel comparisons, have opted for comparing the patches themselves. They have made this choice to make the feature descriptors less sensitive to noise and slight distortions. They also propose to compare triplets instead of pairs to improve the spatial support of the patch comparisons. The comparison between the three points together form a single bit. A couple of triplet arrangements can be seen in Figure 3.10 in green and blue.

Triplet Comparisons

In equation 3.2 of section 3.1 we have described the binary test used by most binary descriptors. Binary descriptor algorithms generate binary feature vectors that take up a low amount of storage space and don't require much time to be matched. Besides improving other properties, LATCH aims to also keep the aforementioned benefits. To accommodate comparing triplets instead of pairs, the authors have redefined Equation 3.2.

Each triplet contains one of the patches that is denoted as the *anchor* referred to as P_a and two other patches that are denoted as *companions* referred to as P_1 and P_2. In the comparison function g, the similarity of the anchor patch P_a is compared to both companion patches P_1 and P_2 using their Frobenious form. The resulting binary value is produced by redefining function f from Equation 3.2 with g from Equation 5.7, with W being defined as the detection window.

$$g(W) = \begin{cases} 1, & \text{if } ||P_a - P_1||_F^2 > ||P_a - P_2||_F^2 \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

Learning Patch Triplet Arrangements

There exist many possible triplet arrangements even within a small detection window W. To decide which arrangements to use, the authors of LATCH have defined a novel selection process. A data set consisting of three different collections of detection windows, pairs of which have been labeled as ”same” or ”not-same” and together form a benchmark of 500K comparisons. To select the appropriate amount of triplet arrangements for some image, triplets are randomly selected and are evaluated over all window pairs in the benchmark. The quality of an arrangement is defined by the number of times it correctly returned the same binary value for the ”same”-labeled window pairs and different values for the ”not-same”-labeled window pairs.

Chapter 4

Experiment Setup

In the following chapter, the large data set used for the experiments will be described. After this a description follows about the way the source data will be modified in order to evaluate the performance of the previously examined algorithms. Finally, the criterion and performance measures that we have chosen are discussed.

4.1 Data Set

The descriptors are evaluated on a large scale with the images provided by The MIR Flickr collection [HL08]. This collection contains images provided by Flickr that are under the Creative Commons license. It contains 25000 images, of which 2500 will be used to evaluate the performance of the descriptors on a large scale. During the experiments, each image is transformed for each variation of each transformation.

4.2 Source Transformations

In the experiments, the descriptors will be evaluated using images that are photometrically and geometrically transformed. The following transformations will be evaluated:

4.2.1 Scale Change

Scale changes are performed by changing the size of the image. In these experiments, the images will be scaled in the range of 0.5 to 2 with an interval of 0.25. See Figure 6.1a for an example of this transformation for argument 0.5.

4.2.2 Rotation

Rotation is done by simply rotating the camera position within a range of 0 to -90 degrees with an interval of -5 degrees. See Figure 6.1b for an example of this transformation for argument -60 degrees.

4.2.3 Illumination Change

Change in illumination is achieved by adding a specified value to each pixel point of the image, making it more intense. The values added range from -175 to +175 with an interval of 25. See Figure 6.1c for an example of this transformation for argument -100.

4.2.4 Two-dimensional Perspective Change

When modifying the two-dimensional perspective of an image, we combine rotations on the x-axis and y-axis to achieve a perspective which resembles us changing the orientation of a 2D poster. For our experiment's dataset we rotate the images in the range 0 to 40 degrees with an interval of 10. All the combinations of these values are evaluated, which brings the total number of variations to 25. See Figure 6.1d for an example of this transformation for argument 30 degrees for both rotations.

4.2.5 Gaussian Blur

Blurring images is done by applying a Gaussian blur filter over the image. This will result in a less detailed image. In our experiments we will use images blurred with a kernel size ranging from 1 to 15, with an interval of 2. See Figure 6.1e for an example of this transformation for argument 13 for kernel size.

4.2.6 Scale Change + Rotation

To further look into possible image transformations that may occur in applications of feature descriptors, we also evaluate a combination of scale change and rotation. We combine the effects of scaling in the range of 0.75 to 1.75 with an interval of 0.25, with the effects of rotation in the range of 0 to -45 degrees with an interval of -15 degrees. See Figure 6.1f for an example of this transformation for argument -30 degrees (rotation) and 0.5 (scale).

(a) Scale change with argument 0.5

(b) Rotation with argument -60 degrees

(c) Illumination change with argument -100

(d) 2D perspective change with 30 degrees rotation on both axes

(e) Gaussian blur with kernel size 13

(f) Rotation and scale with arguments -30 degrees and 0.5 respectively

Figure 4.1: Examples of transformed images

4.3 Performance Metrics

4.3.1 Matching Score

When the framework starts with the experiment, it loads images from the data set and evaluates them one by one. At this time, the keypoints of the source image are detected using the SURF detector using default parameters. This same detector is used for the evaluation of all description algorithms to allow our framework to focus on the performance of the description algorithms, rather than including the variance in results that the usage of different feature detectors could cause. After this, it goes through each algorithm described in Chapter 3 paired with each transformation described in section 4.2. In this process, it goes through each variation of the current transformation and transforms the source image with the parameters of the current variation. Then, the evaluation framework detects the keypoints of the transformed image and computes the feature vectors of both the transformed image and the source image using the current description algorithm. Following this, the evaluation framework matches the feature vectors using the brute-force matcher implemented in OpenCV [Ope17]. The process of finding matches is as follows: the matcher takes the descriptor of each feature in the source feature set and compares it with each feature in the feature set of the transformed image. Before returning any matches, the matcher performs a cross check to validate any matches made. During this cross check, the matcher repeats its first process but now compares each feature from the transformed image with every single feature in the source image. A match is only valid if these two processes obtain the same match pair. After the matcher returns the set of found matches, we filter them based on whether their Euclidean distance is below the threshold of $t = 3$ pixels. If the Euclidean distance is below this threshold, we assume it's a *correct* match. If it is not, we count it towards the number of *false* matches.

The metric we use to measure the accuracy of the feature description algorithms is called the *matching score*. The *matching score* answers the following question: given a detected match, how likely is it to be correct? To calculate the *matching score* for a given pair of images, the number of correct matches relative to the number of false matches is taken:

$$matching\ score = \frac{\#correct\ matches}{\#false\ matches + \#correct\ matches}$$

This metric is based on the precision metric used by Mikolajczyk et al. [MS05] in 2005.

The average *matching score* per variation of each transformation for each algorithm will be discussed in Chapter 6.

4.3.2 Resource Consumption

In addition to the *matching score*, which measures the quality of the descriptors that the algorithms generate, it is of interest how many resources the algorithms consume. In current applications, resource usage is often just as important as the accuracy of algorithms. The metrics we will measure are *time consumed per descriptor* and *memory allocated per descriptor*.

To measure the time consumption of the description algorithms, the clock tick count of the processor is measured $before$ and $after$ the features are computed. To obtain the time between these tick counts in milliseconds, we take $(before - after) * T$ where T is the clock's tick frequency per second. During the computation of the feature descriptors of the transformed images, we measure the consumed time per transformed image. Using this value, we calculate the *consumed time per descriptor* by dividing the consumed time per transformed image by the number of feature descriptors that are computed for the image. The average of this metric per variation of each transformation will be discussed in Chapter 6 for each algorithm.

Memory consumption is registered by measuring the amount of memory each description algorithm allocates during its computation time. To do this, we have modified the $fastMalloc$ function from OpenCV [Ope17] to track this information. During the computation of the feature descriptors of the transformed images, we measure the memory allocated which gives us an impression of how much memory the feature description algorithm uses during the computation of the feature descriptors. To finally obtain the *memory allocated per descriptor* metric, we divide the recorded amount of memory allocated by the number of features that are computed. The average of this metric per variation of each transformation will be discussed in Chapter 6 for each algorithm.

4.4 Implementation of the Framework

For the implementation of the framework we have adapted an existing framework built by the Github user BloodAxe named OpenCV-Features-Comparison [Blo14], which is written in C++. To be able to use this framework for our experiments we had to make numerous changes and improvements. Among other changes, we upgraded the framework for use with OpenCV 3.3 [Ope17], added new algorithms, new and improved image transformations, memory tracking with a custom compiled OpenCV build, a different statistics file structure and large folder processing support using the Boost C++ library [Boo17].

We will compare the feature description algorithms using their default parameters set in OpenCV [Ope17]. The default size of the feature vectors of the description algorithms are shown in Table 4.1.

Algorithm	Feature vector size in bytes
BRIEF	32
BRISK	64
FREAK	64
LATCH	32
rBRIEF	32
SIFT	512
SURF	256

Table 4.1: Default feature vector size per algorithm

The system that we run the experiments on has four hyper-threaded quad-core Intel Xeon CPU's, type E5-2667, running at 3.30GHz and has 48GB of RAM. The program has been optimized to work with OpenMP [DM98], which enables the framework to evaluate multiple variations of a transformation in parallel using 32 threads. This dramatically improves the performance of framework.

Chapter 5

Experiment Result Analysis

In our experiments we have evaluated the effects of multiple image transformations on the performance of feature description algorithms. In the following subsections we analyze our defined metrics to acquire insight into the behavior of description algorithms. We firstly analyze the performance of the description algorithms with regards to the *matching score* per variation of each transformation. Finally, we will discuss the time and memory consumption per descriptor of the algorithms, for each variation of the defined transformations.

5.1 Matching Score

The average results of the experiments with the subset of the MIR Flickr [HL08] data set are summarized in Figure 5.1 per transformation.

5.1.1 Illumination Change

When first examined, there are a few interesting observations that can be made when looking at Figure 5.1a. The general trend of performance is clear. Most of the descriptors perform very similarly, except for BRIEF, which outperforms others. This could have to do with BRIEF's simple random approach, which is able to create sampling pairs without being restricted by a pattern.

5.1.2 Gaussian Blur

The effect of Gaussian blur on the evaluated feature descriptors can be seen in Figure 5.1b. We immediately see one clear method that performs significantly less than others: SURF. In contrast, BRIEF outperforms the other algorithms once again, which could be a result of the simple intensity comparisons BRIEF performs.

5.1.3 Rotation

The results of the effects of rotation on the description algorithms is less converged, as we can see in Figure 5.1c. BRIEF does not provide rotation invariance, which explains its bad performance for this transformation. Furthermore, it is notable that the patch descriptors, SIFT and SURF, perform less well than others.

5.1.4 Scale Change

Upon inspection of Figure 5.1d, which shows the results of the effects of scale change, we see that the performance of rBRIEF, LATCH and BRIEF is not favorable. For BRIEF this is completely expected, as BRIEF does not provide scale invariance. LATCH relies on the scale of the keypoint calculated by the detector. While the SURF detector used in our experiments does calculate the orientation of each keypoint, it appears to be that this is not taken into account. This is likely due to the fact that the implementation of the new LATCH algorithm is currently sub-optimal in the OpenCV [Ope17] framework, as stated on the website of the algorithm [lat]. The lesser performance of rBRIEF is a result of lack of compatibility with the keypoint scale calculated by the SURF detector. The OpenCV [Ope17] implementation does not take the calculated keypoint scale into account, and also does not calculate a keypoint scale itself, which in turn results in bad performance regarding scale change.

5.1.5 Scale Change + Rotation

The effect of scale change can be seen in Figure 5.1e. It is clear that the descriptors which do not take advantage of the keypoint scale, rBRIEF, LATCH and BRIEF, do not perform well when the scaling of the image is changed. Besides this, the behavior of the other descriptors is very similar to the individual transformation rotation (Figure 5.1c) and scale change (Figure 5.1d) that were combined in our experiment. BRISK and FREAK perform the best, and have a similar performance, whereas SIFT and SURF also perform similarly. This is due to the fact that both pairs of these descriptors are very similar in concept, as discussed in Chapter 3.

5.1.6 2D Perspective Change

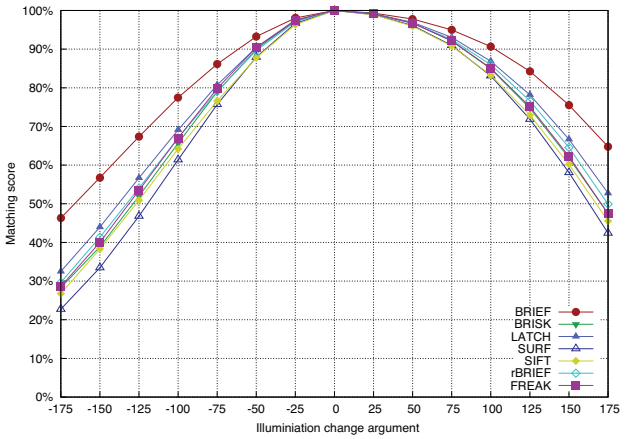
The results of the effects of 2D perspective change are shown in Figure 5.1f. Upon first sight, it is striking that all algorithms perform relatively similarly. As we have seen with rotation and scale change, BRISK and FREAK both perform best, with FREAK excelling the closer we get to more dramatic 2D perspective changes. While BRIEF has shown its limited performance with those same transformations, it performs relatively well under 2D perspective change. This likely has to do with that the plane rotations of this transformation are of a low value, which has given BRIEF a chance to still obtain a reasonable amount of correct matches. We can see that BRIEF's performance decreases when the rotation argument increases.

5.2 Consumed Memory and Consumed Time

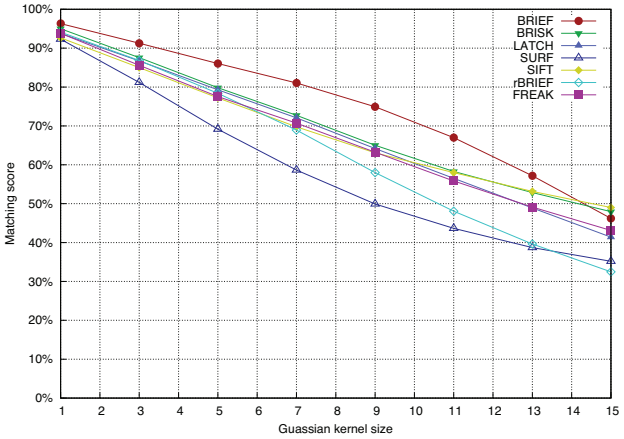
During each feature description process of the various algorithms we have evaluated, we have measured the time and the memory it took to generate the descriptors. This data provides an insight into what computational resources each algorithm requires. The average results of both consumed time and consumed memory are shown in Table 5.1 and Table 5.2 with accompanying comparison charts respectively.

A look at Figure 5.2 gives us an impression of one of the strongest properties of binary descriptors. BRIEF, BRISK, FREAK and rBRIEF all have a shorter computation time by an order of magnitude over SIFT, which takes about 143μs per descriptor. Furthermore, LATCH and SURF both have a significantly lower computation time than SIFT.

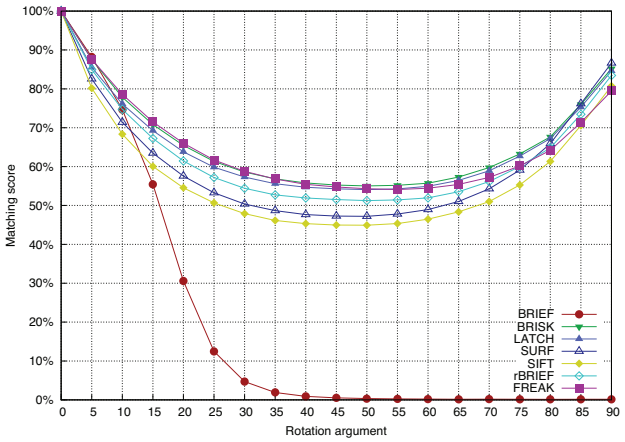
Table 7.3 shows the memory requirements of the algorithms by listing the average consumed memory per feature it took to generate a descriptor, for each algorithm. This data is greatly related to the memory each algorithm uses to store the descriptors, but also gives us insight into how much memory the OpenCV [Ope17] implementation of each algorithm uses for the calculation of these descriptors. As we can see in Figure 5.3, SIFT has a memory footprint that is much larger than any other of our evaluated algorithms. SIFT descriptors are large in size, and take 512 bytes to store. In contrast, the binary descriptors consume an amount of memory that is lower by an order of magnitude when compared to SIFT. While SURF and SIFT share a conceptually similar approach, SURF has a much lower memory footprint.



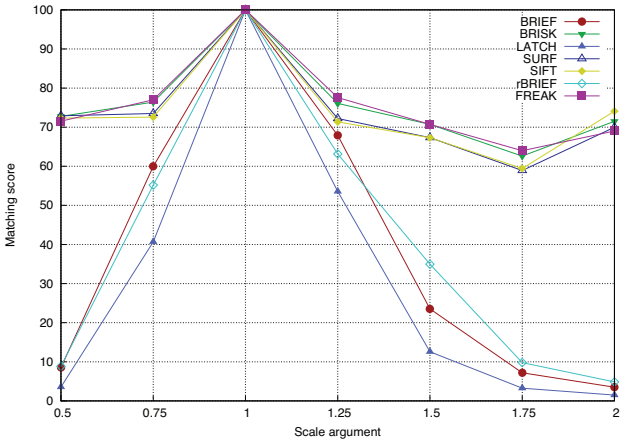
(a) Illumination Change



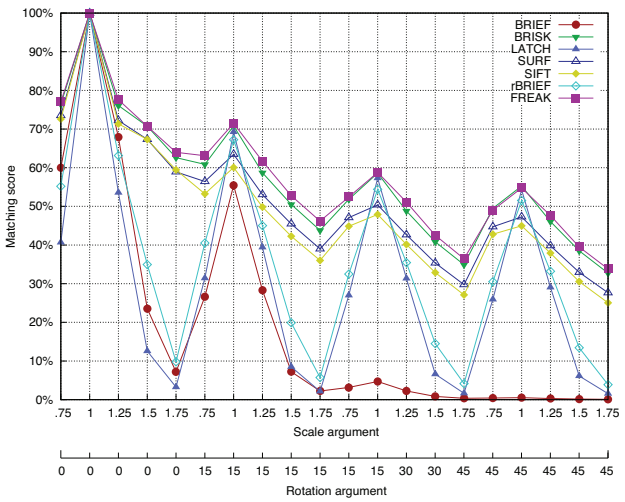
(b) Gaussian Blur



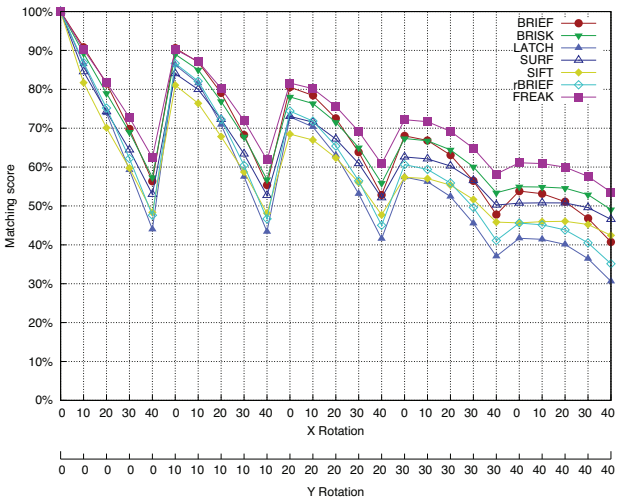
(c) Rotation



(d) Scale Change



(e) Scaling + Rotation



(f) 2D Perspective Change

Figure 5.1: Average *matching score* per variation for each transformation

Algorithm	Consumed time (μs)
BRIEF	5.54225
BRISK	14.408456
FREAK	11.139673
LATCH	82.065743
rBRIEF	6.761497
SIFT	142.720711
SURF	56.663863

Table 5.1: Average consumed time per descriptor in μs

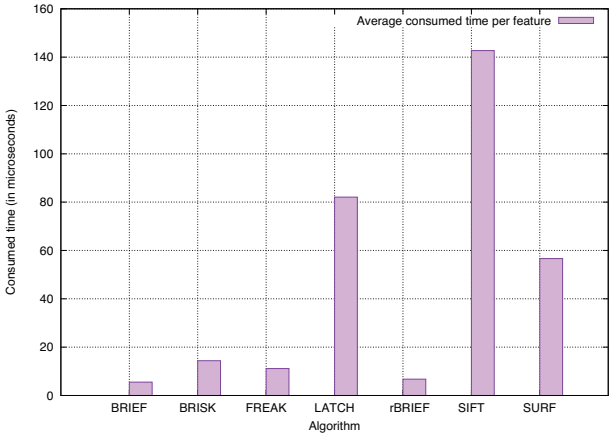


Figure 5.2: Comparison chart of average consumed time per descriptor in μs

Algorithm	Consumed memory (bytes)
BRIEF	1029
BRISK	1208
FREAK	1756
LATCH	998
rBRIEF	1079
SIFT	12794
SURF	1198

Table 5.2: Average consumed memory per descriptor in bytes

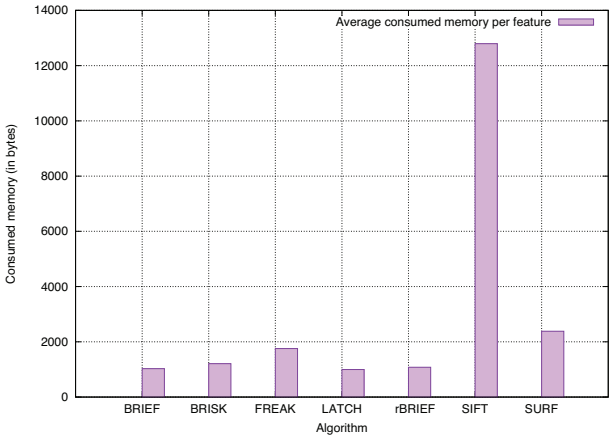


Figure 5.3: Comparison chart of average consumed memory per descriptor in bytes

Chapter 6

Discussion and Conclusions

In this thesis, we have proposed a framework to evaluate the performance and resource consumption of recent feature description algorithms in the OpenCV [Ope17] framework. We have attempted to gain more insight into the behavior of feature description algorithms when exposed to several transformations, namely scale change, rotational change, illumination change, 2D perspective change, Gaussian blur and a combination of rotational and scale change. By having defined a set of performance metrics, we have created a comprehensive overview of the effects these transformations have on the algorithms. From the result analysis of our experiments, we can derive a number of notable observations.

Firstly, when we consider non-geometric transformations such as Gaussian blur and illumination change, we can deduct that BRIEF has favorable performance. BRIEF benefits from the fact that the geometric composition stays intact under these transformations. Taking into account the time and memory it takes to compute a BRIEF descriptor, BRIEF is a highly recommended feature description algorithm to use in situations where geometric transformation is not a factor of concern.

The overall best *matching score* was achieved by FREAK and BRISK, with the former performing marginally better in some situations than the latter. Both algorithms show their robustness under all of the transformations, while both having the lowest memory consumption and computation time. Based on these results, it makes them suitable for evaluating scenes that have geometric as well as photometric changes.

Another notable remark is that LATCH and rBRIEF achieve a lower *marking score* than expected when it comes to scale changes. LATCH relies on the scale of the keypoint calculated by the detector, but is likely not optimized for using the scale property of the keypoints found by the SURF [BTVG06] detector we have used in our experiments. The lack in performance of rBRIEF is a result of lack of compatibility with the keypoint scale calculated by the SURF detector.

When evaluating other published papers that evaluated feature descriptor algorithm performances [LCS11] [LH16], we observe the fact that while a feature descriptor performs excellently in one experiment, it can perform differently in others. As an example, BRISK [LCS11] outperforms SIFT [Low04] in many of the tests of Leutenegger et al. [LCS11], while SIFT outperforms BRISK in many tests by Levi and Hassner [LH16]. This illustrates that experiments often only show one perspective of the behavior of a feature descriptor. There are many contributing factors to the matches found using a specific feature descriptor, such as: the data set that is used for evaluation, the parameter configuration and the kind of detector and matcher used to handle the features.

This brings us to our last point. Though our experiments have given us interesting insights into the behavior of various feature description algorithms provided by the OpenCV [Ope17] framework, the amount of understanding into their behavior that it gives us is limited. The framework could be expanded to evaluate more algorithms and (advanced) transformations, such as combinations of the transformations discussed in this paper. Moreover, different performance metrics such as overlap error, precision and recall used by Mikolajczyk et al. [MS05] could provide us with a different frame of reference to evaluate feature descriptors.

Bibliography

[AOV12] 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.

[BL07] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[Blo14] (Github User) BloodAxe. Opencv-features-comparison. https://github.com/BloodAxe/OpenCV-Features-Comparison, 2014.

[Boo17] Boost. Boost C++ Libraries. http://www.boost.org/, 2017.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

[Cro84] Franklin C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18(3):207–212, January 1984.

[DM98] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[EZW+06] Mark Everingham, Andrew Zisserman, Christopher KI Williams, Luc Van Gool, Moray Allan, Christopher M Bishop, Olivier Chapelle, Navneet Dalal, Thomas Deselaers, Gyuri Dorkó, et al. The 2005 pascal visual object classes challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 117–176. Springer, 2006.

[HAW71] Michael J. (Michael John) Hogan, Jorge A Alvarado, and Joan Esperson Weddell. *Histology of the human eye : an atlas and textbook.* Philadelphia : Saunders, 1971. Includes bibliographies.

[HL08] Mark J Huiskes and Michael S Lew. The mir flickr retrieval evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 39–43. ACM, 2008.

[lat] The learned arrangements of three patch codes (latch) project. http://www.openu.ac.il/home/hassner/projects/LATCH/.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[LH16] Gil Levi and Tal Hassner. LATCH: learned arrangements of three patch codes. In *Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[LSDJ06] Michael S Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 2(1):1–19, 2006.

[MM12] Ondrej Miksik and Krystian Mikolajczyk. Evaluation of local detectors and descriptors for fast feature matching. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2681–2684. IEEE, 2012.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.

[Ope17] OpenCV. Open source computer vision library. https://github.com/opencv/opencv, 2017.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[TCGP09] Duy-Nguyen Ta, Wei-Chao Chen, Natasha Gelfand, and Kari Pulli. Surftrac: Efficient tracking and continuous object recognition using local feature descriptors. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2937–2944. IEEE, 2009.