Faculty of engineering, LTH

Centre for mathematical sciences

Master Thesis

Fingerprint matching - hard cases

Johan Hagel
nek06jha@student.lu.se

Alexander Karlsson
dat11aka@student.lu.se

Supervisor

Magnus Oskarsson
magnuso@maths.lth.se

Examiner

Carl Olsson
calle@maths.lth.se

August 2016

Abstract

Today fingerprint matching software is widely used in applications such as mobile phones. Software for enrollment and verification of fingerprints is usually designed to work for "good" fingerprints, meaning that the fingerprint matching algorithms use minutiae locations. When such information is scarce due to scars, blisters or other damages, the algorithms do not work very well. In this work we demonstrate that gray scale matchers based on interest point detection and extracted local information can be used for matching fingerprints in such cases. Matchers with eigenvalue corner detection such as the Harris- and Shi and Tomasi- corner detectors has resulted in a matching performance of 5.92% false reject rate (FRR), which is a 0.97% improvement on the given database. Combining a gray scale matcher with an ordinary matcher can further reduce the FRR down to 2.54% and suggests that combining algorithms will result in a more secure system.

Keywords. Interest point, descriptor, matching, gray scale matcher, false reject rate.

Acknowledgements

We would like to thank Precise Biometrics for providing us with material, guidance, and for making this project possible. Special thanks to Fredrik Rosqvist for superb guidance and to Rutger Petersson for taking us on board. We further thank the R&D team at Precise Biometrics for helping us with technical struggle. We would also like to thank Magnus Oskarsson at LTH for great feedback and academic guidance.

Contents

Chapter 1

Introduction

1.1 General background

In the end of the 19th century it was discovered that no two individiuals share the same fingerprint. Fingerprints where first used for "booking" criminals and for other areas within law enforcement [26, p. 13]. But since then the use of fingerprints for recognition has become widely spread also to non-forensic areas. An example from recent years is the replacement of PINs for mobile phones. This development has been fueled by a general technology progression and therewith intertwined need for security and protection of data together with increased availability of computing power and less expensive fingerprint sensing hardware.

Precise Biometrics AB (in the following called "PB") develops technology for secure fingerprint identification that can be integrated in fingerprint sensors or hardware platforms. The company has developed fingerprint algorithms that have been implemented in over 160 million devices worldwide. PB's algorithm has achieved top results in various evaluations, such as the MINEX test evaluations.[1] However, the algorithm is less efficient if a finger is damaged so that the fingerprint partially has different features than a normal fingerprint. It would therefore be interesting to try to improve the algorithm for such cases.

1.2 Biometrics and biometric systems

The term Biometrics used in PB's name refers to the use of physiological and behavioral characteristics - called biometric identifiers or simply biometrics - for automatically recognizing a person. [26, p. IX]. Fingerprints is one of many such biometrics.

A *biometric system* is a recognition system that recognizes a person by determining the authenticity of a specific characteristic of that person, in our case the fingerprint. There are two basic modes of a biometric system, it may either be a *verification* or an *identification* system [26, p. 3].[2] A *verification* system authenticates a person's identity by comparing the captured fingerprint with the person's own previously stored template. It is a one-to-one comparison to determine whether the person is who he claims to be. A *identification* system searches a whole template database for a match. It's a one-to-many comparison that tries to decide the identity

[1] `http://www.nist.gov/itl/iad/ig/minex.cfm`
[2] `https://en.wikipedia.org/wiki/Biometrics`

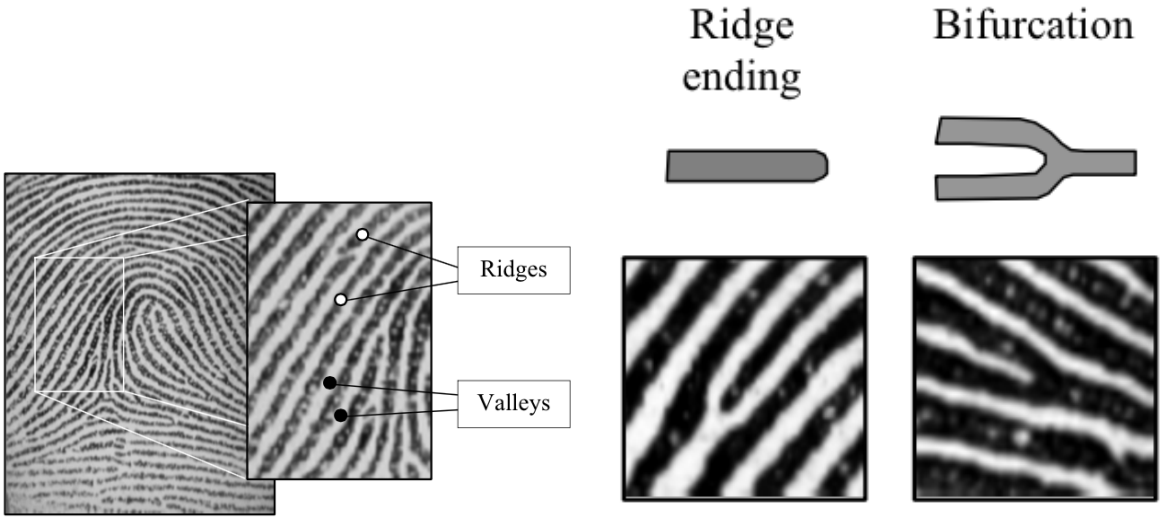
of an unknown individual. Thus, systems for accessing mobile phones are verification systems. If the distinction is unimportant, the generic term recognition can be used [26, p. 3].

The first time an individual uses a biometric system is called the *enrollment* phase. During enrollment the fingerprint of a person is registered and, after some processing, stored as a template. When a person subsequently tries to access the system during the so called *verification* phase, a fingerprint *matching* algorithm compares the fingerprint with the template. If the fingerprint and the template are deemed to be similar enough the person is accepted as the one he or she claims to be.

As will be explained in the following section, fingerprint matching may be a harder or easier problem depending on the quality of the fingerprint and its representation.

1.3 Outlines of fingerprint recognition

A *fingerprint* is produced when a finger is pressed against a smooth surface. The most evident structural characteristic of a fingerprint is the pattern with *ridges* and *valleys*, that sometimes *bifurcate* and sometimes *terminate* (end) [26, p. 83]. See figure 1.1 for an illustration. Terminations and bifurcations (and some other similar local details) are called *minutiae* (from latin minutia 'smallness').



(a) Ridges and valleys

(b) Bifurcations and terminations

Figure 1.1: From [26, p. 97,99]

Beside minutiae, fingerprints may also be represented by local ridge *orientation* and *frequency*. The local ridge orientation at [x,y] is the angle θ_{xy} that the fingerprint ridges crossing through an arbitrary small neighbourhood centered at [x,y] form with the horisontal axis. An orientation image may be produced, showing the orientation at discrete points, see figure 1.2.

The local ridge frequency f_{xy} is simply the number of ridges per unit length along a hypothetical segment centered at [x,y] and orthogonal to the local ridge orientation θ_{xy} [26, p. 91]. The local ridge frequency varies across different fingers, and may also noticeably vary across different regions of the same fingerprint [26, p. 91].

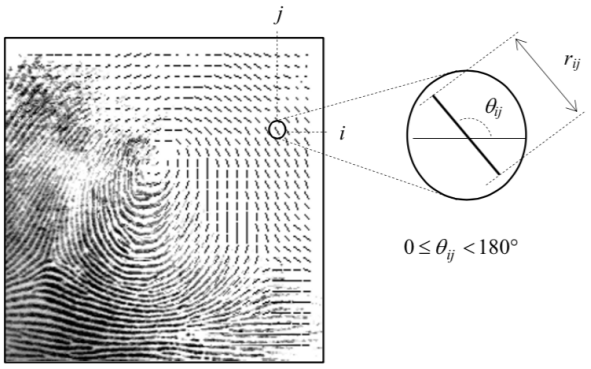


Figure 1.2: A fingerprint image faded into the corresponding orientation image. Each element denotes the local orientation of the fingerprint ridges; the element length is proportional to its reliability. (From [26, p. 88])

A fingerprint recognition algorithm may use both minutiae and spectral information to match two images. As pointed out previously, raw images have to be processed for the matching algorithms to work. The performance of fingerprint recognition techniques depends heavily on the quality of the input fingerprint images [26, p. 91]. If the image quality is insufficient ridges cannot be easily detected and minutiae cannot be precisely located. Therefore different enhancement-techniques are used. This includes contextual filters for 1) low-pass effect along edges in order to link small gaps and fill impurities and 2) bandpass effect orthogonal to ridges in order to increase discrimination between ridges and valleys [26, p. 91]. For example Gabor filters may be used. Most - but not all - methods do also require the gray scale images to be binarized. The binary image is often submitted to a thinning stage, which allows for the ridge line thickness to be reduced to one pixel. Next, an image scan allows detection of minutiae [26, p. 91]. After extracting the minutiae (features), a post-processing stage for removal of spurious minutiae is often a good idea. False minutiae are distinguished from real minutiae because they often have a different structure [26, p. 125], see figure 1.3.

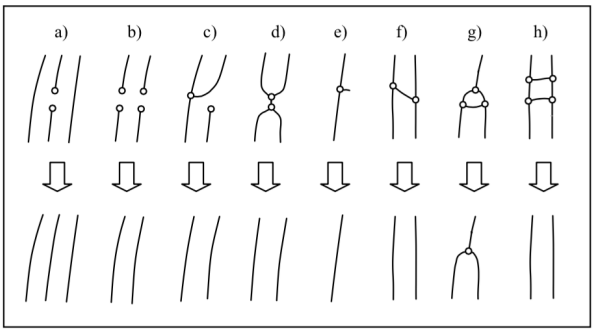


Figure 1.3: The most common false-minutiae structures (on the top row) and the structural changes resulting from their removal (bottom row)From [26, p. 125]

As previously indicated, in the matching stage two fingerprints are compared and the matching algorithm returns either a degree of similarity or a binary decision (matched/not matched). The vast majority of matching methods are minutia-based. The extracted minutiae are stored as a set of points in the 2D-plane. The matching then essentially consists of finding the alignment between the template and the input minutiae feature sets that results in the maximum number of minutiae pairings [26, p. 135]. The second most prevalent matching methods are ridge feature-

based methods. Such methods compare features of the ridge pattern (e.g. local orientation and frequency).

If fingerprints were of sufficiently good quality, there would be no incitement to use anything else but minutiae-based methods. But for parts of the fingerprint where the ridge pattern is damaged due to e.g. injuries or manual work, neither minutia based nor frequency based methods will work. See figure 1.4 for some examples of such fingerprints. Due to such cases (and also due to the difficulty in matching low-quality and partial latent fingerprints) fingerprint matching is still considered to be a challenging problem [26, p. 134].

Fingerpring matching is a *computer vision* problem, and methods from that field is used. We will therefore give a short overview of the parts that are important for this thesis.

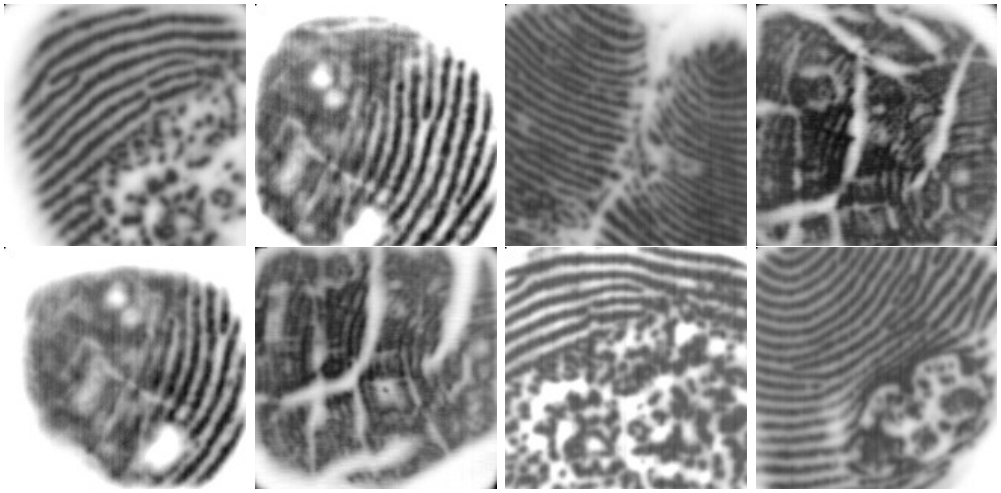


Figure 1.4

1.4 Computer vision

Computer vision, image analysis, image processing and machine vision are closely related fields and there is a significant overlap in the range of techniques and applications that these cover. The basic techniques that are used and developed in these fields are more or less identical, so one could interpret it as if there is only one field with different names. Computer vision includes methods for acquiring, processing, analyzing, and understanding images in order to produce numerical or symbolic information in form of e.g. decisions.[3]

Recognition is maybe the most typical and classical problem in computer vision. As the name suggests, it aims at recognizing objects in images [37, p. 1]. The number of techniques that can be used for recognition is a rather diverse area. As previously indicated fingerprint recognition is one recognition problem where computers have been successful [34, p. 588].

Not only fingerprint recognition, but recognition in general usually depends on various more or less basic image processing measures to improve the image in different ways before it can be used. Other examples than the ones already given are filtering in order to smoothen or sharpen a picture, detection of edges, binarization or segmentation of image parts. After such processing, various more or less specialized recognition methods may be used. Most recognition

[3] `https://en.wikipedia.org/wiki/Image_analysis#Techniques`

algorithms require a descriptor, i.e. a vector that represent the image by numerical description of certain features in the image. Such features may be edges, corners, blobs or other information and are detected by a *feature detector*. Which detector one should choose depend on the type of data. If for example, the images contain round bacteria cells, a blob detector should be used rather than corner detectors. If on the other hand the image is an aerial view of a city, a corner detector may be useful to find man-made structures. It is also important to consider the type of distortion present in the images and choose a detector and descriptor that addresses that distortion. For example, if there is no scale change present, a detector that does not handle scale should be considered. If the data contains a higher level of distortion, such as scale and rotation, then more computationally intensive feature detectors and descriptors should be used. Furthermore, performance requirements should be taken into account. Binary descriptors are generally faster but less accurate than gradient-based descriptors. For greater accuracy, it is also a good idea to use several detectors and descriptors at the same time.[4]

The matching process and the therefore necessary construction of descriptors will be further described in the theory chapter.

1.5 Problem formulation

Against the now presented background, the problem we want to investigate in this thesis may be formulated as follows: Conventional fingerprint recognition algorithms use minutiae information and spectral information when matching fingerprints. Neither of the methods is good at using information from parts of fingerprints where the ridge pattern is damaged or otherwise altered. How may the matching be improved in such "hard cases"?

1.6 Related work

This section introduces previous work on fingerprint matching for damaged fingerprints.

In [39] we are presented to fingerprint matching based on SIFT features [25]. Instead of comparing only minutiae keypoints, the SIFT descriptors are compared. The SIFT information is also combined with the minutiae-keypoints detection. In the pre-processing stage, the input image is high-pass filtered for brightness calibration and low-pass filtered for noise reduction. For ridge-enhancements a Gabor filter is used. Compared with only using the minutiae information, the proposed algorithm performed at 10-20% lower error rates.

It was shown in [40] that combining minutiae keypoints with a SIFT-descriptor performs well on various kinds of fingerprints, even those with a lot of cuts. Experiments showed that the proposed detector/descriptor combination improved the performance.

In [30], a number of fingerprint matching algorithms are compared. Among the algorithms, the locally binary pattern (LBP) [21] descriptor and the histogram of oriented gradients (HoG) [17] descriptor are evaluated and according to [30] showing promising results.

Above resources should imply that SIFT-like detectors/descriptors and binary descriptors such

[4] `http://se.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html`

as LBP would be interesting to look further into. It should also be interesting to study all kinds of interest point detectors and descriptors. It is important to note that the above resources base their studies on much larger fingerprint images than those that are provided for our work. The fingerprints used in [40, 39, 30] are images of either full or at least nearly full fingerprints, therefore it will be interesting to see how such algorithms will perform on smaller images.

Chapter 2

Theory

In this chapter we present the main theory that is needed for the experiments. The theory chapter is organized as follows. First a short overview of image processing techniques that often occur in the theory. Then an account for various methods for detection and description of interest points. Next we discuss matching of descriptors and in connection thereto the principles of RANSAC. Finally an introduction to biometric performance measuring with the details needed for evaluating the performance of biometric authentication.

2.1 Image processing

For improving the image quality a number of image enhancement algorithms were studied. Enhancing the images can imply a better foundation for the algorithms that will search for local features and extraction of region information. In this project, the studied algorithms are histogram equalization and image sharpening.
This section will also cover image processing algorithms such as integral images that allow for fast computations in rectangular windows. The concepts in the image processing theory will reoccur in the interest point detection and extraction of region descriptor theory.

2.1.1 Histogram equalization

A common way to make images easier to compare described in [28, p. 53ff] and [12, p. 59-62] is to distribute the image intensities in the same intensity span, for instance $[0, 255]$. The goal of histogram equalization is to find an image operation such that the image intensities maps to a (approximated) uniform distribution. It is an approximation since a histogram is a discrete distribution. The image operation is based on the cumulative histogram defintion. A cumulative histogram counts the content of each bin in the original histogram up to a specified bin j

$$C[j] = \sum_{i=0}^{j} H[i]. \tag{2.1}$$

The goal is now to find the operation that maps the image intensities in the orginal histogram to a linear cumulative histogram. Mathematically the mapping operation $f_{eq}(a)$ is expressed as

$$f_{eq}(j) = \left\lfloor C[j] \cdot \frac{K-1}{MN} \right\rfloor \tag{2.2}$$

where M and N are the image height and width respectively and K denotes the intentity range $[0, K-1]$. A special case is when an image is already uniformly distributed, equation (2.2) will not change the image content. An example of a mapping is depicted in figure 2.1.

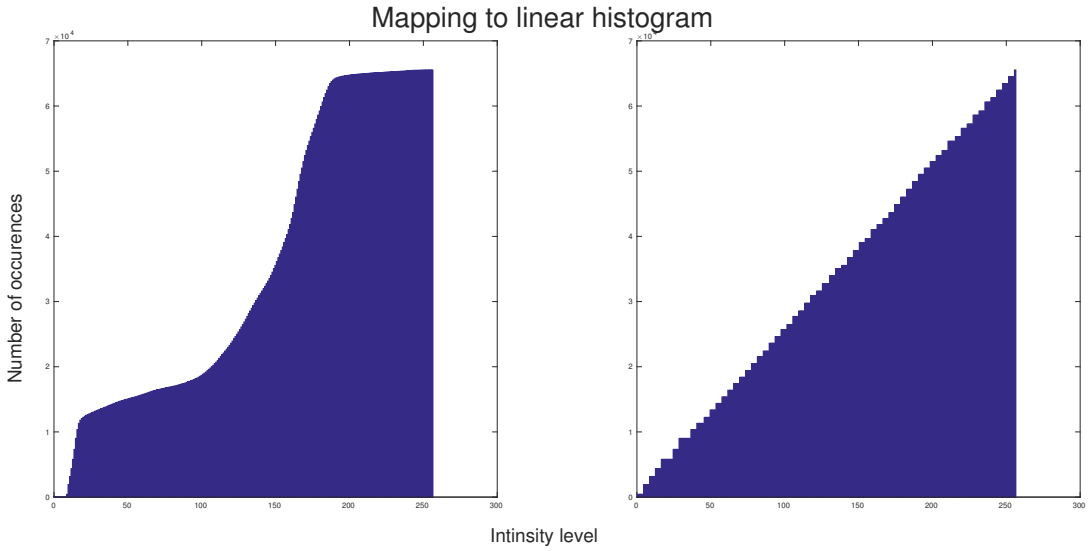


Figure 2.1: The mapping relationship from an ordinary to a linear histogram (right).

2.1.2 Image smoothing

Two common tools for image smoothing have been studied in [22, p. 56] and [22, p. 57], box filters and the Gaussian function respectively.

Box Filter

Box filters are square kernels of size $(2k+1) \times (2k+1)$ and can serve as a way to smooth images. A common box filter is

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{2.3}$$

Image convolving with 2.3 will assign the center pixel with the mean of the eight neighbors.

Gaussian Function

This filter function is a convolution in (x, y) and is defined by sampling from the two-dimensional Gaussian function

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2.4}$$

where σ is the standard deviation, also called radius of the Gauss function. An example of a Gaussian kernel of size 20×20 and radius $\sigma = 3$ is shown in figure 2.2.

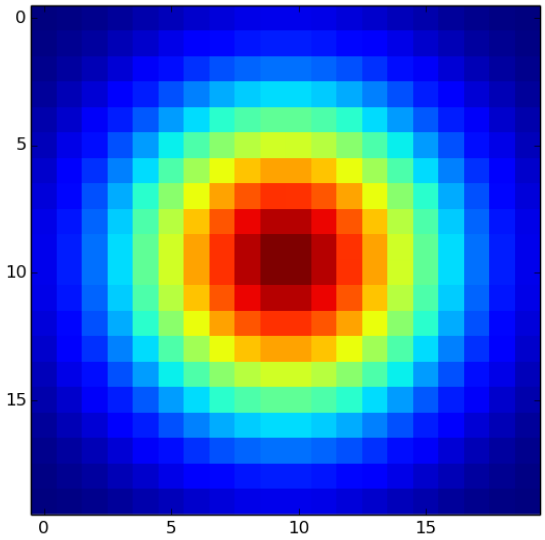


Figure 2.2: A Gaussian kernel of size 20×20 and $\sigma = 3$. The image was generated with python.

2.1.3 Image sharpening

For image sharpening the following method was proposed in [22, p. 44]. The method to sharpen an image will increase contrast of edges without adding noise in homogeneous areas. An unsharpening mask will be used to obtain the sharper image. Taking the difference of the smoothed image S of the image I will result in a residual image $R = I - S$. R is added to I and the sum is the sharper image J

$$J = I + \lambda[I - S] = [1 + \lambda]I - \lambda S \tag{2.5}$$

where λ is a scaling factor.

2.1.4 Integral images

Discovering integral images in [18][37, p. 60f] and [22, p. 52] we learned that these image can improve speed when computing rectangular windows. This is also the main reason for using integral images. When the integral image is calculated, there is no longer need for costly multiplications, only two subtraction and one addition. To obtain the integral image I_{int} the procedure is to sum all values from $I(0,0)$ up to $I(x,y)$, for each location the current sum is saved (see leftfigure 2.3)

$$S(p) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i,j). \tag{2.6}$$

Let W be a rectangular window and the pixels p, q, r and s, where $p = (x,y)$ is the lower right corner of W and q, r and s is one pixel outside of the window, see right figure in 2.3. The following expression will sum all the intensities in W

$$S_W = I_{int}(p) - I_{int}(r) - I_{int}(s) + I_{int}(q). \tag{2.7}$$

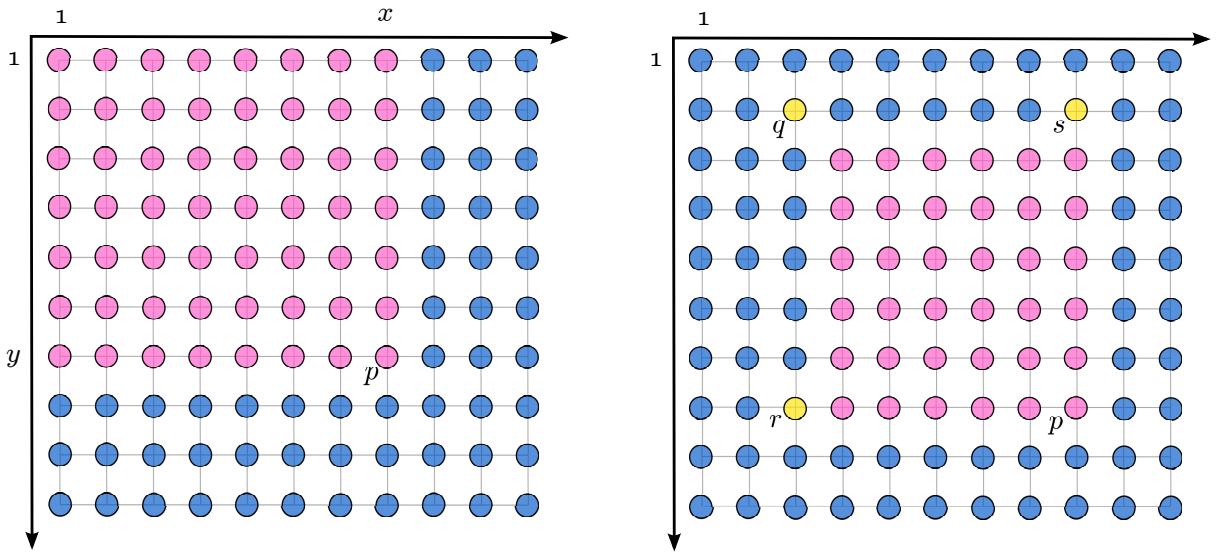


Figure 2.3: An illustration of integral image. The left image shows which pixels that will be summed with equation (2.6), and the right part shows which pixels will be used p, q, r and s in equation (2.7). Figure inspiration from [22].

2.2 Interest point detection and description of regions

In Computer Vision, interest point detectors and region descriptors are applied in recognition, tracking and matching of objects in images and for 3D scene reconstruction of 2D images. In [23] we learned that there is a large number of algorithms at hand in situations like these. The studied interest point detectors and region descriptors in this project will by customized for fingerprint matching. In litterature intereset points are often denoted keypoints, both are used in this report. *Feature point* is also commonly used, that is a word for both the detected point with its descriptor.

Interest point detectors are usually designed to find locations with special propterites. Notably the Harris corner detector [20] and local the local extrema detector/scale space descriptor SIFT [25]. In [23] important features of detectors are presented. One is repeatability, the same interest points should be detected under varying conditions such as when images are noisy or vary in illumination. In other words, the detection of interest points needs to be robust to any kind of image transformation. It is also desierable to find the best possible interest points. Furthermore, interest point detectors often have a time constraint; the interest points have to be detected quickly. It is hard to fullfil both robust interest points and fast computations, if fast computations is fulfilled it often implies weaker matching performance, and the other way around. Instead a proper balance is sought.

What is meant by finding the *best possible interest points* is to find local points rich in contrast, see figure 2.4. Point a is impossible to localize, it can be found anywhere in the sky (bad point), b can only be localized in one direction, along the edge (bad point), c and d have a lot of texture and are hopefully good points [2].

Region descriptors are utilized for saving information extracted in the neighborhood around an interest point. Descriptors save information so that it is possible to discriminate images

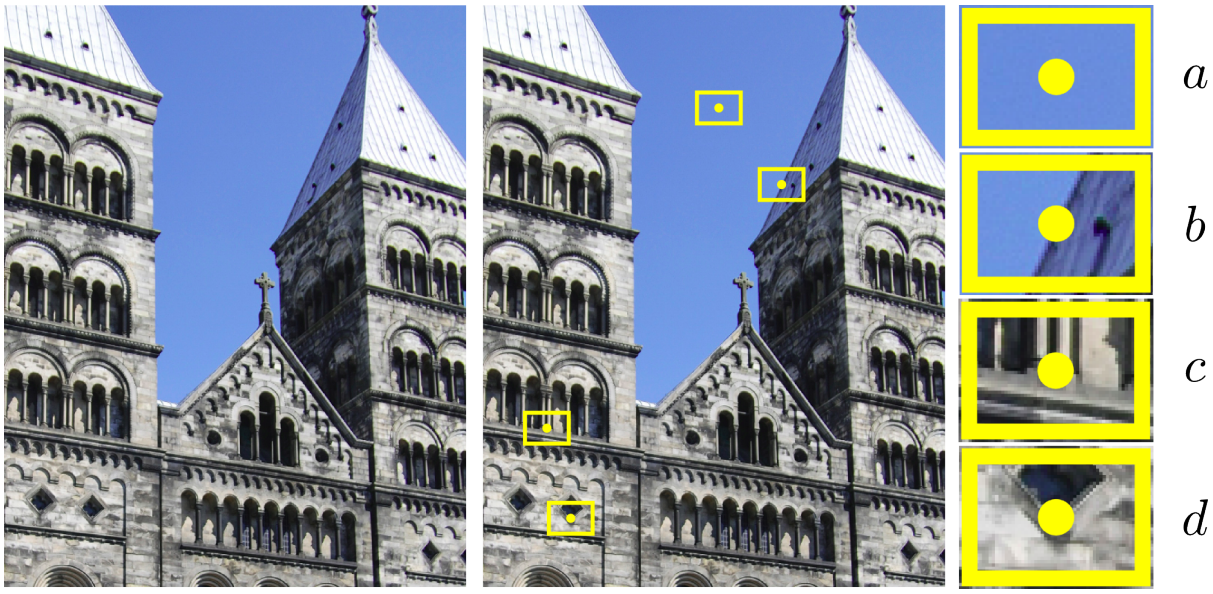


Figure 2.4: Lund cathedral with four mock interest points. See text for description of $a - d$. Figure inspiration from [2].

and objects, and hence it can be possible to match points by comparing the descriptors. The usual procedure of interest point detection and region description listed and described below is presented in detail in [11][37, p. 145ff] and [22, p. 344f].

1. Insert image for extraction of interest points and region descriptors.
2. Detect interest points.
3. Extract information for description in the neighbourhood of the interest points in a descriptor.

The descriptors studied in this project are either based on a histogram model or on a binary sampling pattern. A typical histogram descriptor saves information from a patch around the interest point in a heuristic way such as gradient analysis and then store the number of occurrences of a particular orientation in a predefined bin in the histogram. The binary sampling pattern is usually a circular pattern around the interest point, and by comparing pairs of sampling points in the pattern the descriptor can be constructed. Each comparison will assign one bit in the descriptor string.

Matching and recognition are possible through comparison of the extracted descriptors. The usual manner is to extract and save the descriptors at some training stage for later matching and recognition of the descriptors.

2.2.1 SIFT: Scale Invariant Feature transform

The SIFT algorithm was proposed by David G. Lowe in 2004 [25] for extraction of rotation- and scale-invariant features. SIFT consists of a scale-space extrema detector and a histogram descriptor.

Difference of Gaussians and the scale-space pyramid

To find scale-space extrema locations Lowe [25] proposed to find them with a difference of Gaussian (DoG) function that is convolved with the image. By taking the difference of two images in nearby scale, only separated by a constant factor k the DoG is mathematically expressed as

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{2.8}$$

where $G(x, y, \sigma)$ is given in (2.4) and $*$ is the convolution operator. $D(x, y, \sigma)$ is efficient to compute since only an image subtraction is required, it is also a good approximation of the Laplacian of Gaussian, which has been shown to produce very stable image features. The stage of constructing $D(x, y, \sigma)$ is given in figure 2.5a, the left stack depicts an incremental convolution

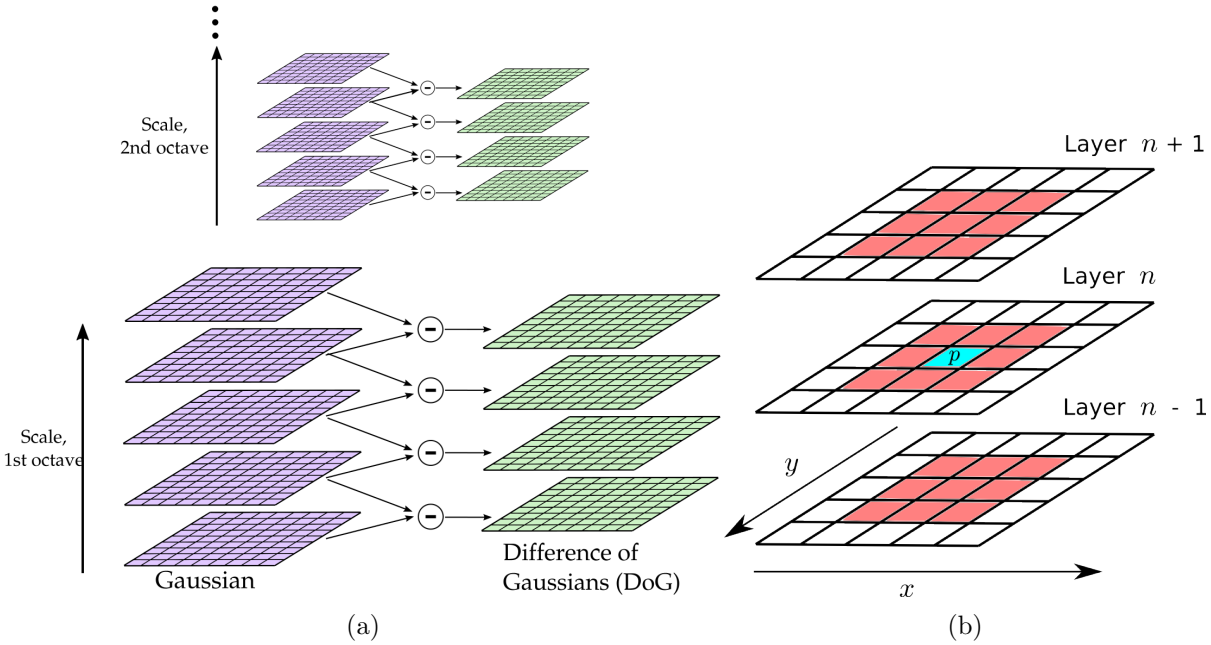
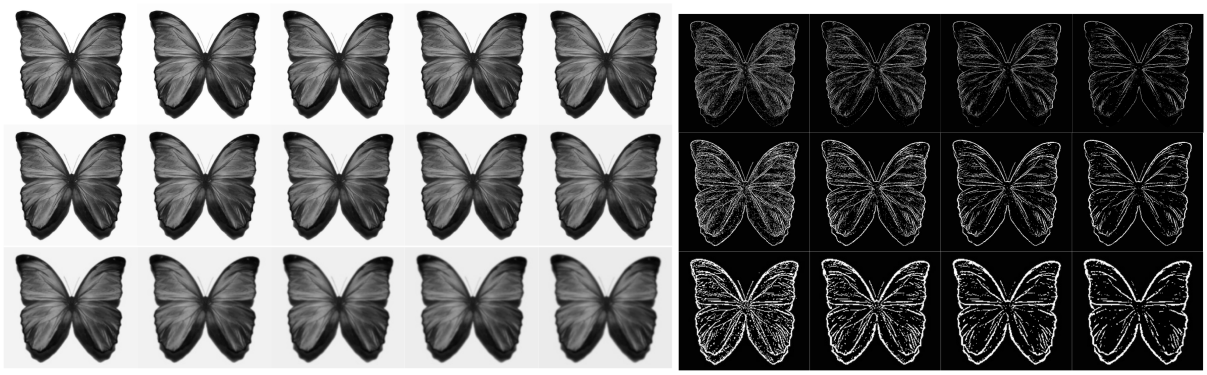


Figure 2.5: (a) Gaussian pyramids (purple) with two octaves and difference of Gaussians (green). (b) detection of extrema. Figure inspiration [25].

with Gaussian functions that differ with k. Two adjacent layers in the left stack are subtracted to produce one layer in the right stack. A purple stack in the left stacks is called an octave and contains $s + 3$ images, therefore they differ by $k = 2^{1/s}$ in scale. When an octave is completed, the second image from the top of the stack is re-sampled (half the image resolution) and the next octave can be produced. Examples of the smoothed images in the left stacks are shown in 2.6a and the corresponding DoG images in 2.6b.

Detection of local extrema

To find the extrema locations of $D(x, y, \sigma)$, Lowe [25] presents a comparison procedure where a sample location is compared to its eight neighbors in same scale, and nine neighbors one layer below and nine neighbors one layer above. If the sample is either smaller than all its neighbors or larger than all its neighbors it is marked as a candidate keypoint. Since many of the samples will fail the test early in the test process the overall cost is rather low. See figure 2.5b for an illustration.



(a) (b)

Figure 2.6: (a): Images smoothed by the Gaussian function. Each row corresponds to one octave in the pyramid. (b): The DoG images of the left figure. The figure was created with a python-script and the butterfly image is public domain.

Localization of keypoint

Localization of a keypoint is carried out in order to eliminate bad keypoints such as those that are sensitive to noise or poorly localized along an edge in the image. Keypoints with low contrast will also be removed. The essence of the localization stage is to compare the candidate keypoint with its neighbors and fit it to the nearby data of the keypoint location, scale and rotation of principal curvature. For a fully mathematical model, please refer to the articles [25, 24].

Assignment of orientation

If the orientation of each keypoint can be extracted, it is possible to use the orientation when constructing the region descriptor and that in turn will make the descriptor rotation invariant. The computations are carried out in the same scale as the keypoint was found in, thus the smoothed image L is used for computation of the magnitude $m(x, y)$ and the orientation $\theta(x, y)$ of the image sample $L(x, y)$

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \\ \theta(x,y) &= \tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}. \end{aligned} \tag{2.9}$$

This information is extracted in a small neighborhood of the keypoint, which will construct a 360° orientation histogram of 36 bins. The samples are weighted by the magnitudes. When the orientation histogram is computed, the highest peak is selected for orientation assignment of the keypoint. If there is a peak that is within 80% of the highest peak, an extra keypoint is created with that orientation. Creating more keypoints contributes to the stability of matching. The theory is presented in detail in [25].

Keypoint description

In the SIFT paper [25] the descriptor extraction procedure is presented as following. The keypoint descriptor is extracted in the same scale as the keypoint was found. A SIFT descriptor

is extracted by first sampling the magnitudes and orientation around the keypoint. Rotation invariance is achieved by rotating the gradient with the calculated orientation for the keypoint. For efficient computation of this step, the gradients are pre-computed for the entire image-pyramid. It is proposed to give less emphasis for the gradients that are far from the center, this is achieved by weighting the gradient window with $\sigma =$ one half of the descriptor window. In figure 2.7 this step is marked with an orange circle around the gradient. The descriptor is now constructed by taking 4×4 sub-windows (red squares in 2.7) and computing an eight-bin orientation histogram, where each bin corresponds to the lengths of the arrows (see 2.7) and then concatenate all histograms resulting in the final 128 length descriptor. A Final modification to avoid sensitivity to illumination changes is to normalize the vector.

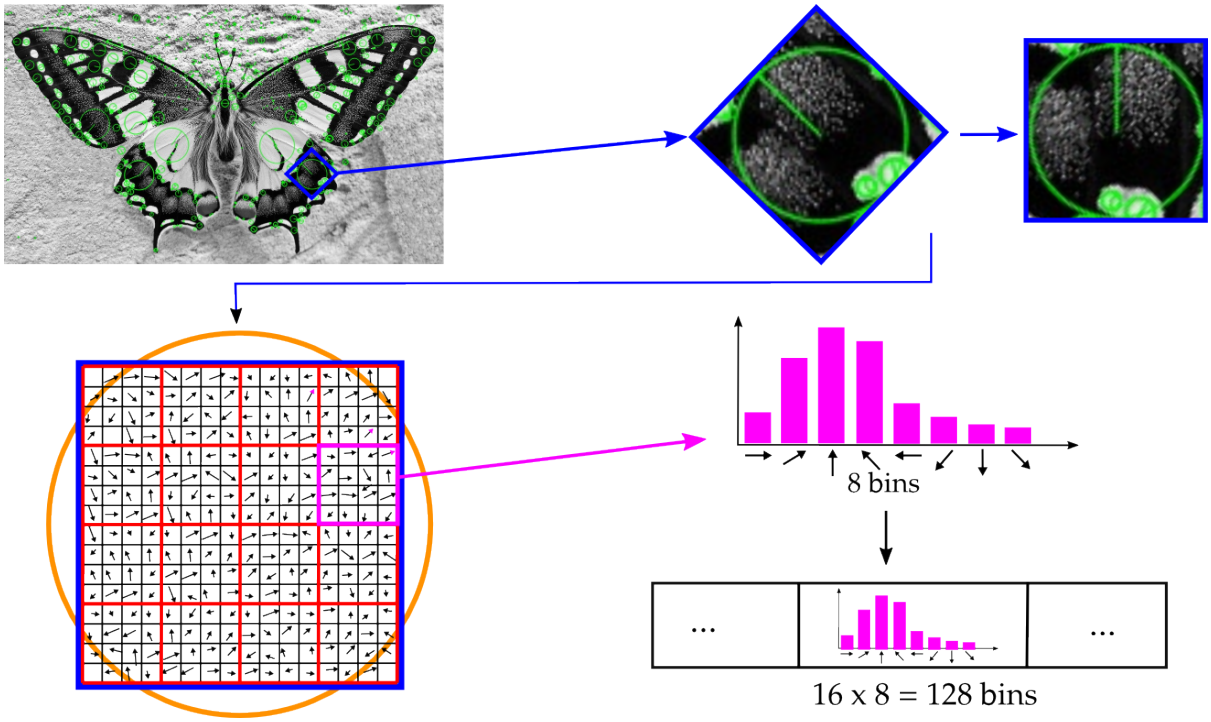


Figure 2.7: Construction of the SIFT-descriptor. SIFT extracts the descriptor in the same scale as the keypoint was detected, then rotated by the keypoint orientation, gradient (precomputed and Gaussian weighted) extraction, then extraction of 4×4 8-bin orientation histograms that are concatenated. The butterfly image is public domain.

2.2.2 SURF: Speeded Up Robust Features

Similar to SIFT, SURF is both an interest point detector and a descriptor. SURF was presented in 2006 by M. Bay et al. [11] as a faster way to compute interest points while also being scale-and rotation-invariant.

Interest point detection

The recipe of extracting SURF features in [11] suggests a number of ingredients for fast computations such as using integral images (equation (2.6)). Furthermore, SURF has a detector that

is based on the Hessian matrix, which can be efficiently and accurately computed. At $p = (x, y)$ with scale σ, the Hessian in SURF is given by

$$\mathcal{H}(p, \sigma) = \begin{bmatrix} L_{xx}(p, \sigma) & L_{xy}(p, \sigma) \\ L_{xy}(p, \sigma) & L_{yy}(p, \sigma) \end{bmatrix}. \tag{2.10}$$

where $L_{xx}(p, \sigma)$, $L_{yy}(p, \sigma)$ and $L_{xy}(p, \sigma)$ are the Gaussian second order derivatives. SURF use this as an extension of the SIFT way of approximating a Gaussian by using box filters. Box filters can approximate the second order derivatives with integral images. The approximations are D_{xx}, D_{xy} and D_{yy}. To find the interest points, the determinant of the Hessian is computed and is a measure of change around a local point. An interest point is selected where the determinant is maximal. With approximations the determinant is calculated with

$$\det(\mathcal{H}) \approx D_{xx}D_{yy} - (0.9D_{xy})^2. \tag{2.11}$$

SURF descriptor

When the keypoints are found Bay [11] demonstrates how the SURF region descriptor is extracted. To be invariant to rotation the SURF algorithm uses Haar wavelets, which has response in both x- and y-directions. The calculation of the responses is in a neighborhood of $6s$ at the interest point location, where the s indicates the scale where the point was found. s is defined by the filter size chosen for smoothing the image. The smallest filter is of size 9×9 and yields $s = 1.2$, a 27×27 filter will then give $s = 3.6$. Next, the responses are weighted with a Gaussian function and the dominant orientation is found by summing all responses, but only for responses within an orientation window of size $\pi/3$. This will result in a local orientation vector and the longest vector overall will be used as orientation. See figure 2.8 where the main orientation is found in the right-most plot. It is now possible to extract the descriptor. Consider figure 2.9, a

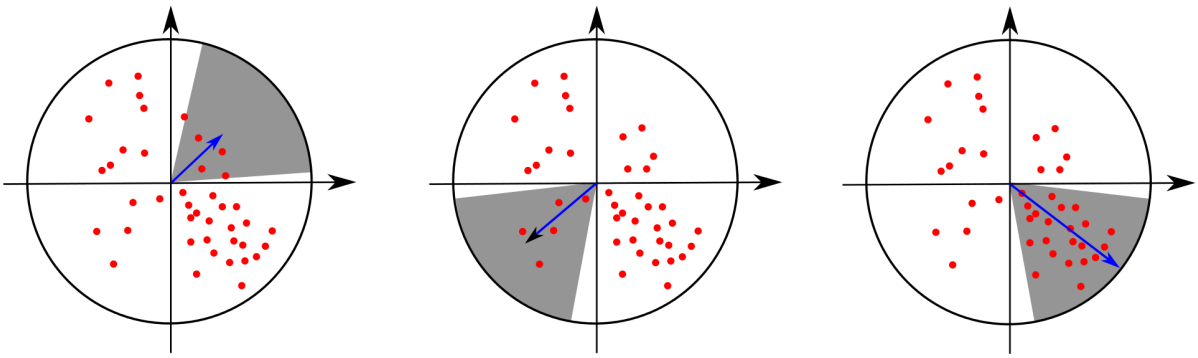


Figure 2.8: Haar wavelet responses in x- and y- direction. Here the orientation is computed, the window (gray) is of $\pi/3$ width.

square window of $20s \times 20s$ around a keypoint is selected and rotated by the main rotation that was found earlier. Then the window is divided into 4×4 sub-windows, where each window contains 25 sample points. For each of the sample points the Haar wavelet response is computed, the x, y, $|x|$ and $|y|$ components are summed, and for each of them stored in an array of four elements, thus the descriptor will contain 64 elements when it is fully extracted.

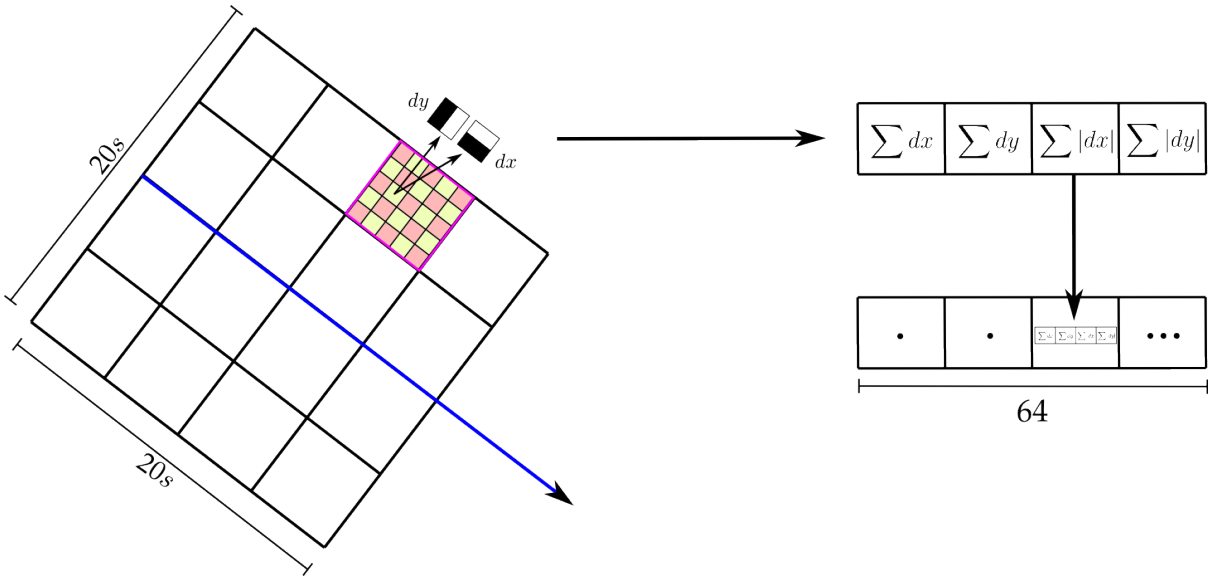


Figure 2.9: Construction of the SURF-descriptor, the main-direction of the descriptor is the blue arrow, which was computed in figure 2.8.

2.2.3 CenSurE: Center Surround Extremas detector

CenSurE was proposed in 2008 by M. Agrawal et al. [8] and was promised to outperform the traditional detectors. Further the CenSurE detector also supports faster computations, and can be used in real-time applications. Two features that are highly desirable when designing a detector is stability and accuracy. CenSure aimed to achieve both.

The difference of Gaussians was implemented in SIFT [25] for approximation of the Gaussians. In CenSurE [8] a simple method is implemented; the difference of boxes (CenSurE-DOB), which results in a Haar Wavelet. The DOB-filter can be found in figure 2.10. Block sizes of $n = \{1, 2, 3, 4, 5, 6, 7\}$ are altered at different scales for scale invariance. For this it is important that zero DC response is fulfilled. If I_n and O_n are the inner and outer weights of the DOB respectively, zero DC response will occur if

$$O_n(4n+1)^2 = I_n(2n+1)^2 \tag{2.12}$$

and after normalizing due to different area at the scales

$$I_n(2n+1)^2 = I_{n+1}(2(n+1)+1)^2. \tag{2.13}$$

CenSurE-DOB is not rotation invariant. On the other hand, if the approximation is implemented with octagon shaped filters instead, rotation invariance can be achieved. Still, the DC response has to be zero, and the filters have to be normalized. The octagon shaped filter is characterized by m and n, see the right image in figure 2.10. Integral images are used for fast computations, see equation (2.6).

2.2.4 MSER: Maximally Stable Extremal Regions

MSER detects blobs in images, a blob is a coherent set of pixels. The algorithm is proposed in the paper from 2002 by J. Matas et al. [27] for finding reliable correspondences in images taken

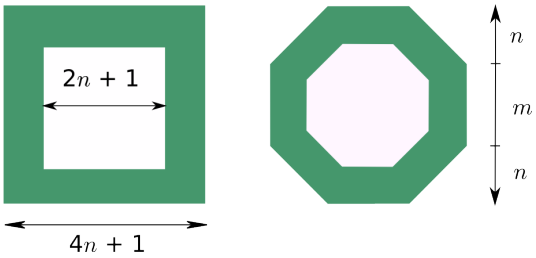


Figure 2.10: (Left) CenSurE-DOB filter. (Right) octagonal filter.

from different viewpoints, possibly with different cameras and with varying illumination. A new way of finding distinguishing objects is presented, which is called extremal regions. Those regions posses desirable features when e g. transforming image coordinates. Furthermore, a fast and (almost) linear complexity algorithm is presented for finding the (maximum stable) extremal regions, which yields the name "MSER", The proposed algorithm is presented as very robust, since the features are very discriminative. That is motivated by that two regions are rarely equal in size.

Lets informally define what a region is. A region is defined by its intensity and the boundary of the region. An intuitive way to understand regions is to think in terms of thresholding and binarization of an image I. Let I have 256 intensity levels, then I_i is a thresholded and binarized image, $i \in [0 \quad 255]$. The first image I_0 is a complete black image, and as i is incremented white regions will emerge in the binary image. At some i white regions will merge into larger regions, and at the last thresholding level, the final image I_{255} is white. The connected components of all levels forms the set of maximal regions. In figure 2.11 the process is depicted, the first image is transformed into gray scale and then thresholded at $i = \{0, 40, 80, 120, 160, 200, 240\}$. Here, each image is threshold at levels that differ 40 in intensity, that difference is called the δ-parameter and determines how many steps the detector will use for testing the stability.
For some images, binarization will reveal regions that are stable over several levels. Finding such regions are desirable since those regions have nice properties. They are, e g. multi-scale detectable, invariant to affine transformations, stable over a number of binarization levels. The Matlab MSER documentation [3] and the MSER paper [27] have extensive documentation of MSER regions and features, if deeper understanding is needed.

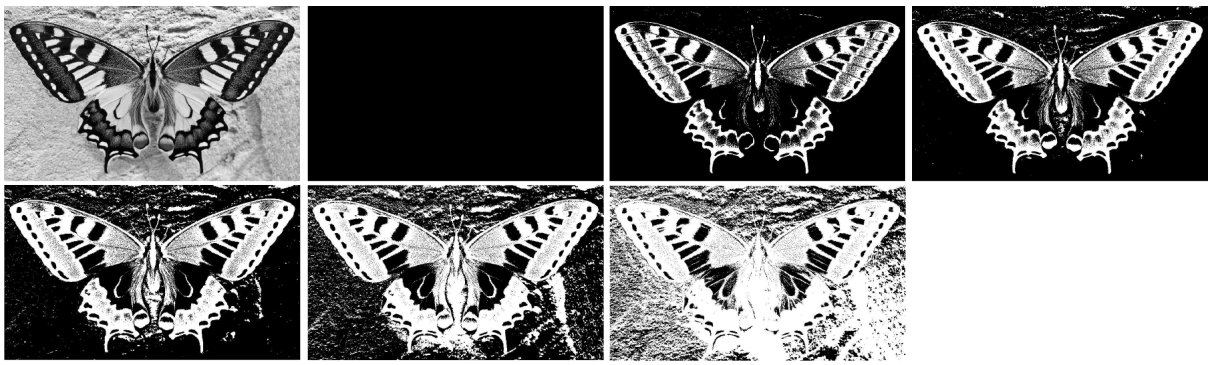


Figure 2.11: Example for graphically showing binarization over levels $i = \{0, 40, 80, 120, 160, 200, 240\}$. This example is binarized at very coarse levels, but they concept should be clear. We see that some parts are stable over multiple intensity levels and how some regions merge. The figure was generated with Matlab.

The procedure follows by enumeration of the extremal regions and construction of a data structure. Computing the data structure takes $\mathcal{O}(\log \log n)$ time and its content is the area of each component. To access the components, the intensity level is used. When the process is finished the output (an MSER region) is represented by its position of a threshold value and the local intensity minimum [27].

2.2.5 Harris corner detector

In 1988 C. Harris and M. Stephens [20] proposed a combined edge and corner detector. What Harris and Stephens mean with a combined edge and corner detector is to find the junctions, where lines cross, thus corners. The Harris corner detector is based on the Moravec's corner detector [1] and finds the corners in the image I by maximizing

$$E\left(x,y\right) = \sum_{u,} \sum_{v} w\left(u,v\right) \left(I(u+x, v+y) - I(u,v)\right)^2 \tag{2.14}$$

where w is the image window and (x, y) is the shift. Harris and Stephens express it as: *"Look for local maxima in E above some threshold value"*. The concept of computing (2.14) is to examine the change in I over a small patch (u, v), see figure 2.12 where there is an illustration of the conceptual idea of the Harris Corner detector; it detects locations where there is change in all directions. Harris and Stephens mention a few problems with Moravec's algorithm such as too sensitive to noise and edges and too few shifts in (2.14). After an extensive mathematical

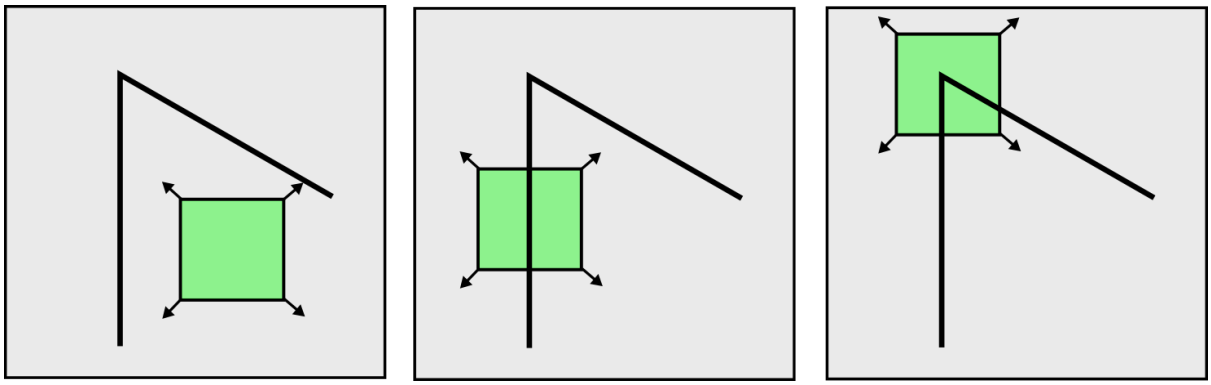


Figure 2.12: Conceptual idea of the Harris corner detector. Left: flat area, no change. Middle: Edge, no change along the edge. Right: Corner, change in all directions. Image inspiration [35].

analysis, Harris and Stephens presents the Harris matrix $\mathbf{M}$. Please refer to [20] for a complete mathematical deduction of $\mathbf{M}$. $\mathbf{M}$ is computed from the image derivatives:

$$\mathbf{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.15}$$

where I_x and I_y are the first order derivatives of I. Let λ_1 and λ_2 be the two eigenvalues of $\mathbf{M}$. The eigenvalues represent change in the intensities in the orthogonal direction in $I(p)$. There is a more effective way than computing the eigenvalues which need costly square root computations,

[1] `https://en.wikipedia.org/wiki/Corner_detection#Moravec_corner_detection_algorithm`

and that is to calculate the cornerness measure or response function [22, p. 67f][20]. It is derived from matrix diagonalizing:

$$\mathbf{M} = \mathbf{V} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathbf{V}^T. \tag{2.16}$$

Where $\mathbf{V}$ is a rotation matrix. Now the response function can be computed:

$$R = \det(\mathbf{M}) - \kappa \cdot \mathrm{tr}(\mathbf{M})^2 = \lambda_1\lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2. \tag{2.17}$$

where κ is a small constant and

$$\begin{aligned} \mathrm{tr}(\mathbf{M}) &= \lambda_1 + \lambda_2 \\ \det(\mathbf{M}) &= \lambda_1\lambda_2. \end{aligned} \tag{2.18}$$

If the eigenvalues are large and $\lambda_1 \sim \lambda_2$, a corner is found. An edge is found if one eigenvalue is small and one is large. If both eigenvalues are small, a flat surface is found.
If $R > 0$ then a corner is found, if $R < 0$ an edge is found and finally a flat surface is found if R is small. In [16] an intuitive model to map eigenvalues to corners is presented, see figure 2.13. By looking at the shape of the ellipse, we can find out what is found. The case when the eigenvalues are large then we will have a large circle and that indicatess change in all directions, so that maps to a corner. Having one large and one small eigenvalue will indicate that an edge is found. Finally, when two small eigenvalues are found, we have found nothing interesting.

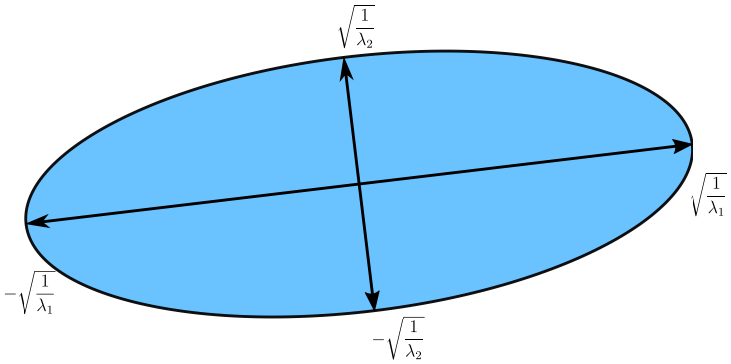


Figure 2.13: An ellipse depicting the relationship between eigenvalues and the structure at a location in the image. Large eigenvalues and $\lambda_1 \sim \lambda_2$ indicate change in all directions, thus a corner.

2.2.6 Shi and Tomasi corner detector

J. Shi and C. Tomasi at the Cornell University and Stanford University respectively published a paper in 1994 [33] on "Good Features to Track". GFTT is a second name for this detector. Good Features are connected to features that make the feature tracker work best during the tracking process of an object. To achieve this Shi and Tomasi present a new corner detector algorithm. Mathematical details and deduction of them could be read in detail in [33]. We restrict our findings in the paper to the new response function that exploits eigenvalues to find corners. The response function in the Harris detector was given by

$$R = \det(\mathbf{M}) - \kappa \cdot \mathrm{tr}(\mathbf{M})^2. \tag{2.19}$$

Shi and Tomasi modified the response function a little and it is given by

$$R = \min(\lambda_1, \lambda_2). \tag{2.20}$$

If R is greater than a given threshold, a corner is found. In the original paper [33] equation (2.20) is written as $\min(\lambda_1, \lambda_2) > \lambda$, other notations are borrowed from [4].

2.2.7 FAST: Features from accelerated segment test

E. Rosten et al. [31] proposed in 2006 a corner detector that exploits machine learning algorithms. FAST was intended to be used in real-time applications that require fast computations, especially in video applications with high frame rates, without sacrificing quality in the detected features.

A FAST interest point is detected in the following manner. Assume that we have a candidate pixel $I(p)$, a threshold value t and 16 neighborhood pixels around $I(p)$ in a circular shape. There is an example in figure 2.14 of how that pattern looks like, the candidate pixel is p. If 12 contiguous pixels are brighter than $I(p) + t$ or darker than $I(p) - t$, then $I(p)$ is a corner. An improvement in time is feasible if instead pixels 1, 5, 9 and 13 are considered, then at least three of them have to be brighter than $I(p) + t$ or darker than $I(p) - t$ in order to mark $I(p)$ a corner. However the second alternative is less accurate. [31] suggests to train a decision tree

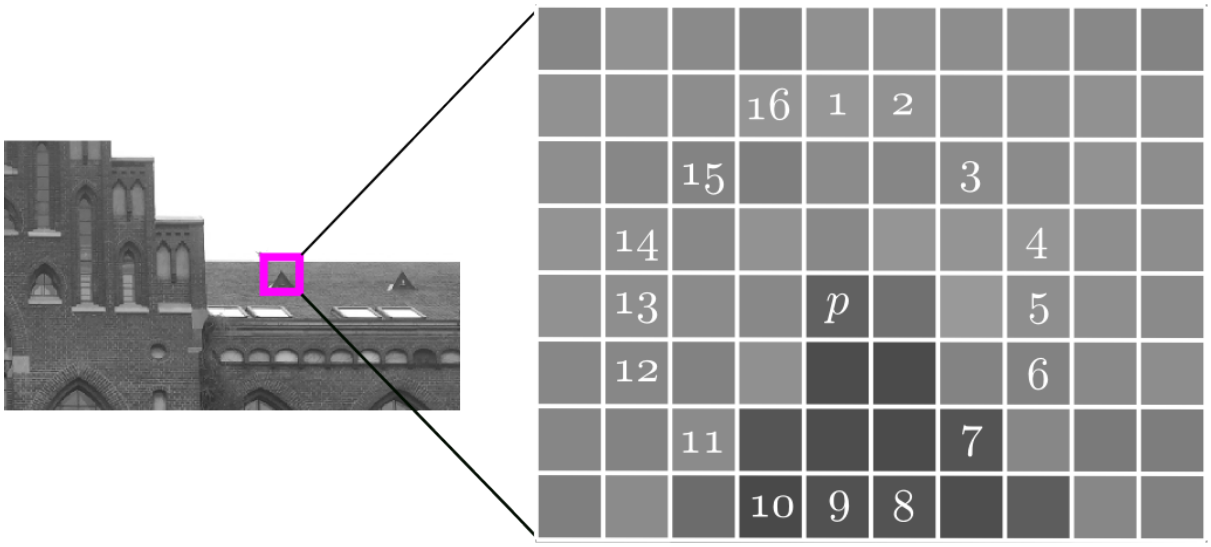


Figure 2.14: An example of finding FAST features. A candidate pixel is marked p in the right image part. 12 contiguous pixels have to be brighter or darker than the candidate in order to mark it as a corner.

for faster computations. In implementations such as OpenCV [22, p. 68], decision trees have been replace with SIMD (Single Instruction, Multiple Data)[2] instructions, which are supposed to be faster than decision trees. FAST also includes non-maxima suppression[3] for reducing the amount of detected features.

[2] https://en.wikipedia.org/wiki/SIMD
[3] http://users.ecs.soton.ac.uk/msn/book/new_demo/nonmax/

2.2.8 Dense interest points

When this kind of feature detection algorithm was developed, the inventor and author of the Dense paper [38] T. Tuytelaars stated that the "best of two worlds" was adopted: a combination of interest points and dense sampling over image patches. The interest points are there for repeatability and the dense sampling is implemented because of the high variation in the number if interest points extracted; the number is highly dependent on the image information. When implementing the dense sampling algorithm, it allows for a large quantity coverage of the image while it also outputs a constant number of interest points. It has a weakness: dense sampling does not allow for as good repeatability as the interest points can yield. Dense interest points are the combination of interest points and a dense sampler. To find these points the first step is to apply the dense sampler over a number of patches and scales, see Figure 2.15 smaller scale require a finer sampler grid. The sampled patches are then refined in scale and location and is done by finding a local maximum within a patch. According to the Dense paper [38], this allows for freedom to adopt to the image content while simultaneously preserving the structure of the sampling grid.

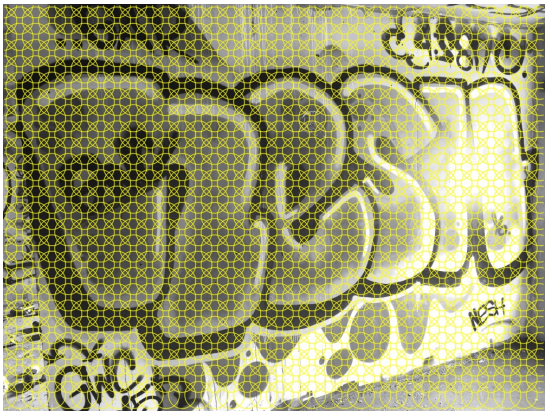


Figure 2.15: A dense sampling grid over a test image.

2.2.9 The BRIEF descriptor: Binary Robust Independent Elementary Features

The BRIEF descriptor algorithm was proposed in 2011 by M. Calonder et al.[13] as highly discriminative and can be computed with a small amount of intensity difference tests. For better evaluation performance, the Hamming distance can be used instead of the L_2 norm. BRIEF is proposed to be fast when matching. In [22, p. 344f] the simple procedure of constructing the BRIEF descriptor is presented. After minor Gaussian smoothing of the input image I, n pixel pairs (p, q) are selected randomly in a $(2k + 1) \times (2k + 1)$ neighborhood, see figure 2.16 for a visualization. The resulting descriptor is not ration- or scale-invariant. Each pixel pair is tested with the formula

$$s(p, q) = \begin{cases} 1 & \text{if} \quad I(p) - I(q) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

And the binary descriptor string F will for n selected pairs be constructed as

$$F = \sum_{i=0}^{n-1} s(p_i, q_i) \cdot 2^i. \tag{2.22}$$

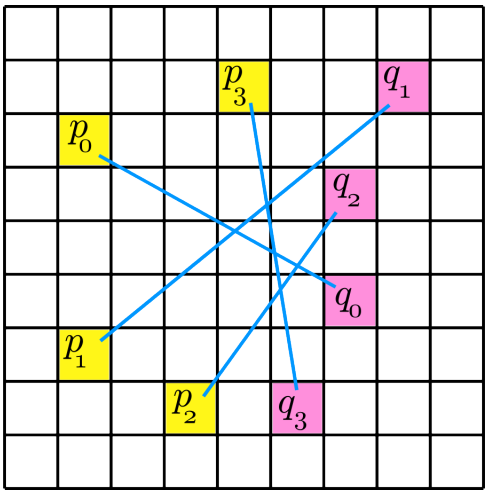


Figure 2.16: An example of selecting four pixel pairs randomly. Image inspiration: [22].

2.2.10 ORB: Oriented FAST and Rotated BRIEF

The algorithm for detection of ORB interest points was presented in 2011 by E. Rublee et al. [32] at the Willow Garage, California. ORB is in the original paper promised to be "two orders of magnitudes" faster than the SIFT detector. ORB was also shon to be eqaul in matching performance to SIFT. ORB was invented to save energy in smart phones when e.g. stitching panorama images.

Interest point detection

[32] suggests FAST interest point detection as a base of the ORB detector, the only difference is that an orientation component is added. FAST is used as is, setting the threshold for the center pixel in the ring (see Figure 2.14), and implements support for multi-scale features. At each level in the scale-pyramid, FAST features are computed.

The corner orientation is calculated with the intensity centroid which is calculated with, assuming that the corner intensity is unbalanced from the center

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{2.23}$$

where m_{ij} is the moment

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \tag{2.24}$$

of the pixel location (x, y). If a vector $\vec{OC}$ is constructed, where O is the corner center, the orientation is then

$$\theta = \arctan \frac{m_{01}}{m_{10}}. \tag{2.25}$$

ORB descriptor

In [32] we learn that ORB uses a slightly modified BRIEF descriptor. The orientations of the interest points will be used to calculate a dominant direction. ORB will use this information for rotation invariance. What ORB does in addition is to learn which sampling pairs that are best to use. The key properties, uncorrelated and discriminative pairs are desirable. To achieve this, ORB suggests a searching algorithm that will reduce 205590 tests (Equation 2.21) to 256 tests.

2.2.11 BRISK: Binary Robust Invariant Scalable Keypoints

BRISK was proposed in 2011 by S. Leutenegger et al. [23] as an alternative to the state-of-the-art detectors/descriptors SIFT and SURF. Using a scale-space FAST detector and a binary descriptor made BRISK even faster than SURF.

Keypoint detection

In [23], the FAST detector is extended to meet scale-space and rotation-invariance, which is needed for high quality keypoints. The extension is an image pyramid. At each level of the pyramid, BRISK runs a FAST detector in a neighbourhood of 16 pixels, with the requirement that at least 9 consecutive pixels are either darker or brighter than the center pixel.

Keypoint description

The descriptor implementation proposed in [23] makes use of a sampling pattern such as that in figure 2.17. The magenta locations indicate sampling points, and the green circles have radius σ that is the standard deviation of the Gaussian kernel, for smoothing at the sampling points. Aliasing effects are also avoided when using the Gaussian kernel. $N(N-1)/2$ sampling pairs $(\mathbf{p}_1, \mathbf{p}_j)$ are selected, and the smoothed intensity at these locations are $I(\mathbf{p}_i, \sigma_i)$ and $I(\mathbf{p}_j, \sigma_j)$, they are in turn used for gradient estimation

$$\mathbf{g}(\mathbf{p}_i, \mathbf{p}_j) = (\mathbf{p}_j - \mathbf{p}_i) \cdot \frac{I(\mathbf{p}_j, \sigma_j) - I(\mathbf{p}_i, \sigma_i)}{||\mathbf{p}_j - \mathbf{p}_i||^2}. \tag{2.26}$$

All pairs are divided in two groups $(\mathcal{S}, \mathcal{L})$, short and long distance pairs respectively, divided by choosing a threshold of the length. $\mathcal{L}$ is then used for estimating the direction of a keypoint

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{L}} \mathbf{g}(\mathbf{p}_i, \mathbf{p}_j) \tag{2.27}$$

where L is the size of the number of pairs in $\mathcal{L}$. Now the descriptor can be constructed. The descriptor string is assembled by rotating, $\alpha = \arctan(g_y/g_x)$, the sampling pattern and comparing $(\mathbf{p}_i^\alpha, \mathbf{p}_j^\alpha) \in \mathcal{S}$. A bit b in the descriptor string is assigned the value

$$b = \begin{cases} 1 & \text{if} \quad I(\mathbf{p}_j^\alpha, \sigma_j) > I(\mathbf{p}_i^\alpha, \sigma_i) \\ 0 & \text{otherwise.} \end{cases} \tag{2.28}$$

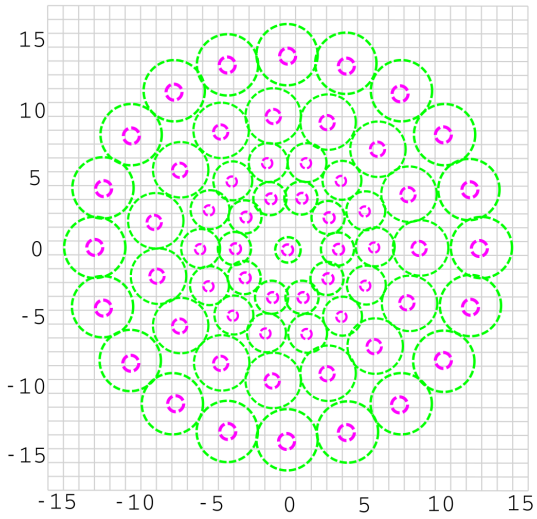


Figure 2.17: The BRISK sampling pattern, magenta points are the sampling points, the size of the green circle indicate the std of the Gaussian kernel used for smoothing. Image inspiration: [23].

2.2.12 FREAK: Fast Retina Keypoint descriptor

The FREAK algorithm was proposed by A. Alahi et al. [9] in 2012. FREAK consists of a binary descriptor and the sampling pattern is similar to the human retina and the fovea, the closer the center of the fovea the more receptors are found. Aside from this, FREAK is similar to the BRISK and ORB descriptors. The pattern reminds of the BRISK pattern, and has adopted the machine learning tools for finding optimal set of sampling pairs. The FREAK descriptor has implemented a circular sampling grid, it is constructed in such a way that more sampling points are located closer to the center of the grid and a less amount of samples are located at the rim of the grid. All points sampled, are smoothed with a Gaussian kernel. The kernel's standard deviation is smaller closer to the center of the sampling grid. This pattern reminds of an eye's retina; The fovea has a larger quantity of receptors closer to the center, see figure 2.18a and the how the sampling pattern reminds of the distribution of the receptors in figure 2.18b. For more details on the FREAK descriptor please refer to the FREAK paper [9] and the guide [1].

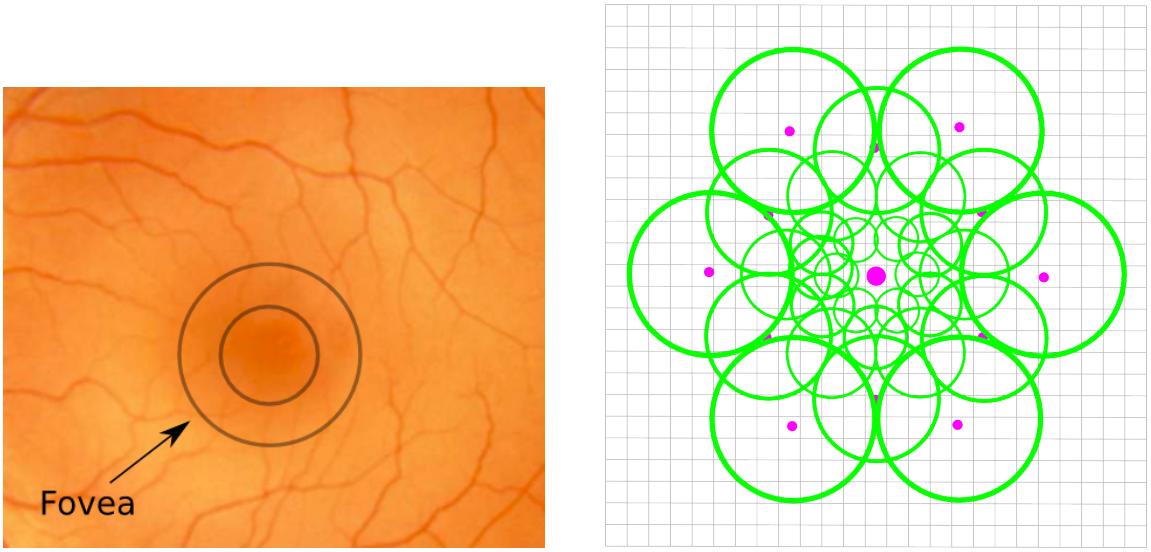
Now, the descriptor F of size N is constructed as a binary string containing a sequence of one-bit DoG:

$$F = \sum_{0 \leq a < N}^{N-1} T(P_a) 2^a \tag{2.29}$$

where P_a is a pair of receptor fields (a green circle), and also

$$T(P_a) = \begin{cases} 1 & \text{if} \quad I(P_a^{r_1}) - I(P_a^{r_2}) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.30}$$

Where r_1 and r_2 indicate receptive field one and two respectively. However, a large amount of pairs will be produced and some will not effectively describe the image content. A solution to this problem is to use an ORB approach in order to get the best pairs. The orientation invariance is achieved by using an approach similar to BRISK, but instead of selecting long pairs, FREAK selects the pairs that have symmetric receptive fields with respect to the center.



(a) (b)

Figure 2.18: (a): Human retina and its fovea. (b): the FREAK sampling pattern. Image inspiration (b): [9].

2.3 Matching descriptors

2.3.1 Introduction

After having found interest points in an image and calculated their descriptors, the descriptors may be "compared" with descriptors of points in another image in order to determine whether an object in the first image is depicted in the second image. See for example figure 2.19a-b, where the first image depicts a box and the second image a cluttered scene containing the box. Finding the object in the image with the cluttered scene is the same problem as matching two fingerprint images.

(a) Image of object (b) Image of cluttered scene containing the object

Figure 2.19: The images are taken from the matlab example Object Detection in a Cluttered Scene Using Point Feature Matching

Descriptors are numerical vectors and one way to compare them is to calculate the euclidean

distance between them. If two descriptors describe the same point in two different images, the distance should be relatively small. (The absolute values may depend inter alia on how noisy the images are.) For example, let $im1_1$ be a vector describing point 1 in $im1$ and let $im2_3$ and $im2_4$ be the vectors describing points number 3 and number 4 in im_2, with for example the values

$$\mathbf{im1_1} = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix}, \mathbf{im2_3} = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} \text{ and } \mathbf{im2_4} = \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix}.$$

Then the distance between $im1_1$ and $im2_3$ is

$$\sqrt{\sum \left(\begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix} - \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} \right)^2} = 2.$$

And the distance between $im1_1$ and $im2_4$ is

$$\sqrt{\sum \left(\begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix} \right)^2} = 14.$$

Since the first distance is smaller, the descriptors $im1_1$ and $im2_3$ are more likely to describe the same point in the two images, than are the descriptors $im1_1$ and $im2_4$.

The L2-norm (which gives the Euclidean distance) is not the only norm that can be used. A faster (but less exact) way is to calculate hamming norm, i.e. bitwise XOR, followed by bit count. For example, bitwise XOR of

$$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \text{ is } \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

Thus, summing the two ones in the resulting vector gives that the hamming distance is 2. A prerequisite for using the hamming norm and calculating the hamming distance is of of course that the descriptors are binary. This applies inter alia to descriptors such as BRISK or FREAK, that rely on pairs of local intensity differences.

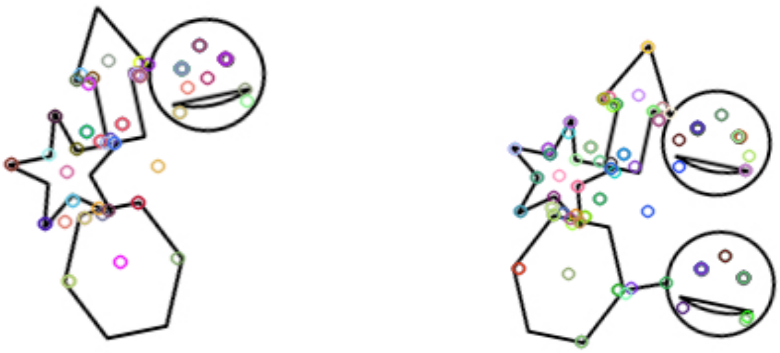
By comparing each descriptor from $im1$ with each descriptor from $im2$ (so called brute force-matching) the best maches are found. When the distance between two descriptors is small enough - smaller than some predefined ϵ - they are deemed to be a "match". The matches are collected in an n-by-2 matrix, where n is the number of matches and the two entries in each row are the indices of the keypoints in respective image. Example:

$$\text{keypoints\_im1} = \begin{bmatrix} 1 & 2 \\ 4 & 7 \\ 3 & 9 \\ 8 & 12 \\ 3 & 5 \\ 9 & 13 \end{bmatrix}, \text{ keypoints\_im2} = \begin{bmatrix} 3 & 1 \\ 2 & 6 \\ 4 & 10 \\ 12 & 13 \\ 8 & 7 \\ 12 & 17 \end{bmatrix}, \text{ matches} = \begin{bmatrix} 1 & 2 \\ 3 & 3 \\ 5 & 6 \end{bmatrix},$$

Thus, in the example the point with index 1 (and coordinates (1,2)) in $im1$ has been deemed to match the point with index 2 (and coordinates (2,6)) in $im2$, etcetera. The uncolored points in the images have been deemed not to have any matches in the other image.

If there are many keypoints, it may be expensive to try all the possibilities to find the best matches. If the images contain around 1000 keypoints each, it would mean 1 000 000 comparisons. An alternative is to use a Flannmatcher, an approximative method that finds good - but not necessarily the best - matches [29].

Some of the found matches will be wrong, namely if the distance between two descriptors is smaller than ϵ even though they describe different points. This may happen if the neighborhood of two keypoints is similar. An example is shown in figure 2.20a-b. Consider for example the keypoints in the forehead of each face. Since the neighborhoods of the keypoints are similar (or identical if the radius of the neighborhood isn't choosen to big), the descriptors should be to. This applies to other keypoints in the faces as well. Therefore a matching algorithm will match keypoints in the face in the left image not only to keypoints in the correct face in the right image, but also to keypoints in the face in the lower right corner of the right image.



(a) Found keypoints in stylized image 1 (b) Found keypoints in stylized image 2

Figure 2.20

Indeed, after running the matcher we get the result in figure 2.21 (a sift-matcher is used.) As foreseen there is a number of false matches between the face in $im1$ and the lower face of $im2$. But there are also many correct matches. As for example the center of the star in $im1$ is correctly matched to the center of the star in $im2$, and the center of the arrowhead in $im1$ is correctly matched to the center of the arrowhead in $im2$. To remove the false matches we use the so called RANSAC algorithm (explained in the section below). After running RANSAC we get the result in figure 2.22, were the false matches are removed.

Comparing this with fingerprints, the second image can be thought of as an imposter that looks much like the genuine, but with a slightly different trait (the second face). However, it may also be two images of the same fingerprint, except that the second image contains a larger portion of the fingerprint.

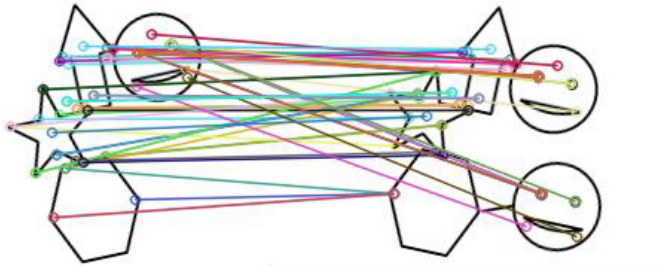


Figure 2.21: Matches before RANAC

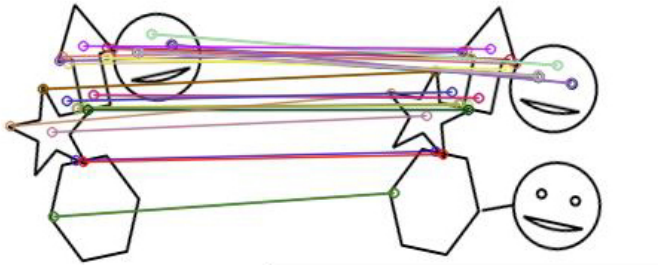


Figure 2.22: Matches after RANAC

2.3.2 RANSAC

General

RANSAC (RANdom SAmple Consensus) [19] is an iterative method for estimating the parameters of a model from a data set that contains outliers. The algorithm works by identifying the outliers in the data set and estimating the desired model using data that does not contain outliers.[4] Thus, RANSAC may also be seen as an outlier detection method, and is often used in the correspondence problem, i.e. the problem of finding a set of points in one image which can be identified as the same points in another image.[5]

This is done in the following steps:

1. Randomly select a subset of the data set, and call the set the *hypothetical inliers*.
2. Fit a model to the selected subset.
3. Determine the number of inliers.
4. Repeat steps 1-3 for a prescribed number of iterations and keep track of the maximum number of inliers (and/or the model that gives the most inliers).

The procedure is most easily exemplified with the problem of fitting a line to a dataset that includes outliers, see figure 2.23. In that case the subset is two random points, and the model fitted to the subset consists of a and b in the equation $y = ax+b$. Next, each x-coordinate is put into the model (which means that it will be on the line) and it is checked whether the distance between this modelled value and the real y-value is smaller than some predefined distance δ. In that case the point is deemed to be an inlier. In figure 2.23 there are seven inliers.[6]

[4] `http://se.mathworks.com/discovery/ransac.html`

[5] `https://en.wikipedia.org/wiki/Correspondence_problem`

[6] One way to calculate the distance is to check only the y-direction, the so called total least squares method.

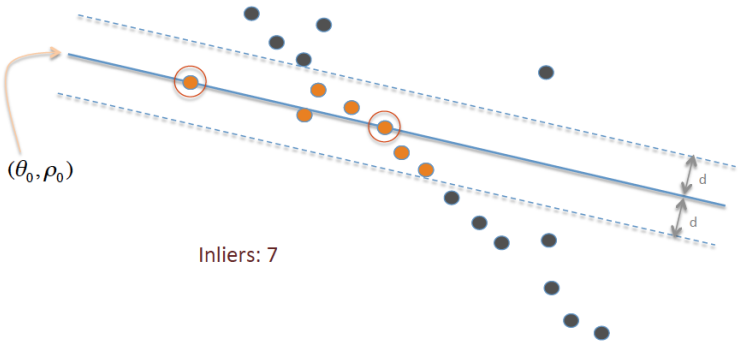


Figure 2.23: Illustration of RANSAC

If this is the best run so far, the number of inliers will be saved as max_nbr_of_inliers before the procedure is repeated. (If one instead was interested in the model rather than the number of inliers, the model would be saved instead of the maximum number of inliers.)

The principle is the same when it comes to images, but instead of a and b the model consists of the transformation matrix T, which in its turn consists of the rotation matrix R and the translation vector t. The variables in the model are the rotation angle θ and the translations t_x and t_y in the x- and y-directions respectively. The matrix looks as follows:

$$T = \begin{bmatrix} cos\theta & -sin\theta & t_x \\ sin\theta & cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}, \text{ or more compactly: } T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}.$$

Afterwards the model may be further refined by using all inliers from the best iteration to reestimate the model parameters using singular value decomposition. That is however of greater interest if the purpose of using RANSAC is to find the model rather than just removing outliers.

There is always a possiblity that RANSAC is unlucky in all iterations and doesn't find a subset that only contains inliers. In that case the model will not be anyway near the reality and it is likely that the only inliers are the subset used to calculate the model (and maybe a couple more points (or point pairs) that are "lucky" and mapped close to where they would be with a more accurate model).

This case may be dealt with in the code. If the number of inliers is smaller than some predefined value, it may be assumed that RANSAC failed and treat that case as if no inliers were found. (However, in practice it won't matter in most cases anyway because if there are to few matches it will be deemed not to be a match anyway.) For example the following values could be possible to choose as predefined values:

But in the example the total least squares method is used. There is more than one possible choice of distance function.

Model	Point pairs needed	Min number of inliers accepted
Euclidean	2	5
Similarity	2	6
Affine	3	7
Projective	4	10

Even if RANSAC finds subsets containing inliers, the best subsets from two different runs may be different and thus the model may be slightly altered if the best runs doesn't contain exactly the same inliers. In the fingerprint matching case it could mean the difference between granted and denied access if the score is close to the threshold. Why this is not a problem will be discussed later.

Alternative models

Above we have discussed a model with three degrees of freedom, i.e. the rotation θ and the translation t_x and t_y. This is called a Euclidean transformation, and it preserves distances between points. Other 2D-models than Euclidean transformation may be considered. See figure 2.24 for an overview of such models.[7]

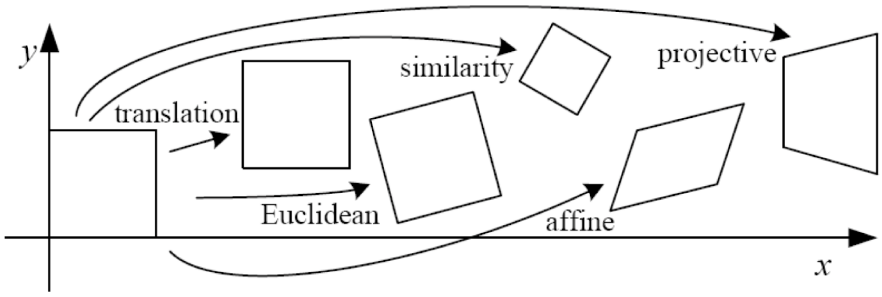


Figure 2.24: Illustration of 2D-transformations

An seen from the figure, an even simpler model than euclidean transformation is plain translation, a model with only two degrees of freedom and thus only one point pair is necessary to find the parameters of the model.

The next step after Euclidean transform is to allow scaling, while still preserving all angles - the *similarity transform*. This adds another degree of freedom compared to the Euclidean transform. Thus, four points, or two pointpairs, are needed to determine the model. The matrix of the transform is simply the same as for the euclidean transform, but with an added s for scale:

$$T = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix}.$$

Still two more degree of freedom is added if using the *affine transform* as model. The affine transform preserve parallell lines, but do not require preservation of angles. Thus the R for

[7] Image from `http://www.math.louisville.edu/~pksaho01/teaching/Lecture3.pdf`

rotation is replaced with an A, where A is an arbitrary invertible matrix. In this transform the scale may be different in the x- and y-directions. Furthermore, *skew*, is added as another degree of freedom.

$$T = \begin{bmatrix} sA & t \\ 0 & 1 \end{bmatrix}.$$

The now discussed transforms are all special cases of a more general class of transforms, the *projective transformations* or - with another name - *homographies*. In the matrix representing the homographies the last row is arbitrary, meaning that the transformation may change also the z-coordinate and thus the effect will be as if the image is seen from different angles and distances (see figure 2.24).

$$T = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

Accordingly, parallel lines will not necessarily be mapped to parallel lines. There are nine element in the matrix, but the scale doesn't matter (i.e. T and λT represents the same element) and there are therefore only eight degrees of freedom.

It is not obvious which transform is the best for comparing fingerprint images. One thought is the projective transformation since it gives the most freedom to match two fingerprint images even if they are distorted in different ways. However, more ways to match two fingerprints also mean more ways to get false matches. Limiting the degrees of freedom also mean that some unrealistic matches will be prohibited.

We have used the projective transformation in all our calculations, mostly because then we could use prewritten code from the OpenCV-library and we have limited time for this thesis. But if we had more time next step would be to try Euclidean and similarity transforms. As will be shown in the section below, this also makes life easier for RANSAC.

2.3.3 Efficiency of RANSAC

The probability that RANSAC finds at least one set of only inliers depend on 1) the number of outliers in the set, 2) the numbers of point pairs that have to be picked out in each iteration and 3) the number of iterations that the algorithm is run. By knowing any two of these parameters, a third parameter can be calculated.

When the model is an Euclidean transform, RANSAC only has to pick out two point pairs in each iteration, while four pairs are needed when using the homograpy/projective transformation. Of course the probability for RANSAC to pick out only inlier pairs is higher when only two pairs are picked than when picking four pairs. This means that RANSAC needs fewer iterations if using Euclidean transform as model, than if using the projective transform.

So how many iterations should RANSAC be set to? It depends on the required probability that RANSAC finds at least one inlier set.

Assume for example that there is a set with 10% outliers and we want a 99% probability that RANSAC finds an inlier set. Also assume also that we use the euclidean transform as a model so that two point pairs need to be picked out in every iteration.

The number of iterations needed may be calculated in the following way:

$0.9^2 = 0.81$	Probability that all points are inliers in one iteration
$1 - 0.9^2 = 0.19$	Probability that some point is an outlier in one iteration
0.19^x	Probility that some point is an outlier in each of x iterations.

We want this probability to be smaller than 0.01, i.e.:
$0.19^x < 0.01$

Taking the logaritm and solving for x gives
$x \cdot ln(0.19) < ln(0.01) \Rightarrow x > 2.7730$

Thus, RANSAC needs three iterations to achieve the required probability. If the set instead contained 60% outliers RANSAC would need 27 iterations. If in addition four point pairs were needed, RANSAC would require 178 iterations.

In reality the inlier ratio varies very much, i.e. some images many false matches and others only very few false matches. This depends on inter alia the characteristics of the fingerprint. For a fingerprint with a repetitive pattern and many similar features there may be many false matches (as in the case with a brick wall). If the fingerprint is more varied due to for example scars or other damages one could expect less false matches. What to do about that?

One idea is to check a large number of images and calculate the mean and standard deviation of the inlier ratios (where inlier ratio is inliers/number_of_matches). **N.B. that this demands that the inlier ratio is normally distributed.** If for example the mean is 0.7 and the standard deviation is 0.3, then 84% percent (corresponding to one standard deviation) of the matches will have an inlier ratio over 0.4 and almost 98% (corresponding to two standard deviations) of the matches will have an inlier ratio over 0.1. Or put another way, almost 98 % of the matches contain less than 90% outliers.

For these 98% we can calculate the number of iterations needed to get a 99% probability that RANSAC finds an inlier set in the same way as in the example above if the outlier ratio is changed to 90% instead of 10%. (It can be mentioned that it is a little sloppy to say that the probability will be 99% for this whole set. Since RANSAC is run with the same number of iterations every time, the probability will actually be 99% only for the matches with the lowest inlier ratios, for all other inlier ratios it will be higher. It would therefore more correct to say that it is a lower limit for the probability. To calculate the total probability one would instead integrate or sum over the probabilities for all inlier ratios greater than 0.1.)

For the other 2% of the matches that contain less inliers, RANSAC would need more iterations to reach 99% probability that at least one inlier set is randomly selected during the iterations. Thus, for those matches the probability will be lower than 99% (and thus 99% is an upper limit) with the same number of iterations. The more outliers, the lower the probability will be. As said earlier, if no inlier set is found, the best model will be bad and only the randomly selected points and maybe a few more "lucky" will be inliers.

In this example we did exactly the same thing as in the first example and found the number of iterations needed from 1) the number of outliers and 2) the required probability. But we could also have decided beforehand on the number of iterations that we can afford and used that information and information about the number of outliers to calculate the probability that RANSAC finds an inlier set.

However, one cannot assume for sure that the inlier ratios are normally distributed and that standard deviation can be used! To get a picture of the distribution it may therefore be necessary

to do a histogram of the inlier ratios.

Let's say that for at least 95% of the matches we want a at least 99% probability that RANSAC finds an inlier set. Then we check the histogram (from the right) an find the area corresponding to 95% of the histogram. We then check what inlier ratio this corresponds to, maybe for example 0.08. We can then calculate the number of iterations needed by RANSAC in the same way as the first example above. Thus the question would be: "Assume there is a set with 92% outliers and we want a 99% probability that RANSAC finds an inlier set. Also assume also that we use the euclidean transform as a model so that two point pairs need to be picked out in every iteration. How many iterations would be needed?"

Alternative RANSAC models

The RANSAC algorithm was first published in 1981 and has since then become a fundamental robust tool in the computer vision and image processing community.[8]

Because of the great importance of RANSAC there is a variety of contributions and variations to the original algorithm, mostly meant to improve the speed of the algorithm, the robustness and accuracy of the estimated solution and to decrease the dependency from user defined constants. Those RANSAC-modifications all have different names and usually when reading a paper one does not have to take much notice of the characteristics of the method used in a particular case. However, if using RANSAC is a central part of an algorithm that one is working on, it may be a good idea to investigate different alternatives to see if the algorithm may be improved with some other RANSAC-alternative. Below we shortly mention a couple of alternatives.

In [36] *MLESAC* is presented as a generalization of RANSAC; it samples in the same way as RANSAC but instead of maximizing the number of inliers it chooses the points that maximizes the likelihood. Experiments have shown to perform better than RANSAC [36].

PROSAC implements a different sampling function. Instead of sampling uniformly, PROSAC samples from a set of top-ranked correspondences. PROSAC is proposed as faster than RANSAC and it also has better matching performance, in worst case, PROSAC converges to the performance of RANSAC [15].

R-RANSAC is a randomized version of RANSAC created to reduce the computational burden to identify a good consensus set. The basic idea is to initially evaluate the goodness of the currently instantiated model using only a reduced set of points instead of the entire dataset, [14].

2.3.4 Alternatives to RANSAC

What alternatives are there to count the number of inliers when trying to find the best model?

One could imagine using something like $-number\_of\_inliers$ or $-1/number\_of\_inliers$ and calling it an error function. But it would be directly linked to the number of inliers, so it is in principal the same thing as counting the number of inliers.

What about calculating the sum of distances between the modeled values (Tx) and the corresponding inliers (y). That would lead to a larger error the more inliers that is found, and that is not what we want. Even if we divided by the number of inliers this would often be the case,

[8] `https://en.wikipedia.org/wiki/RANSAC`

since the mean of many inliers will be higher than the mean of only two or three inliers which are all close to the line, see figure 2.25 for an illustration. As is seen from the figure, the red line would be chosen as the best model since the two inliers really close to the line, although it is obvious that the blue line is a better model. However, one could imagine using this technique as a complement when choosing between two models with the same number of inliers.

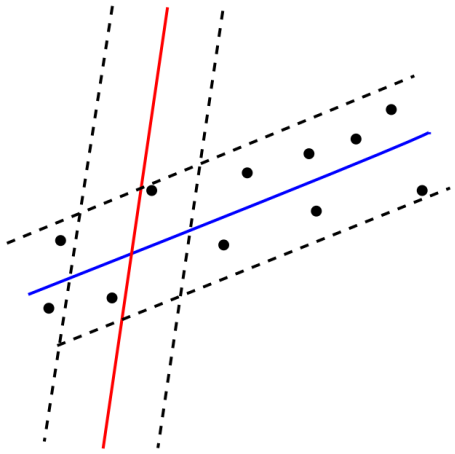
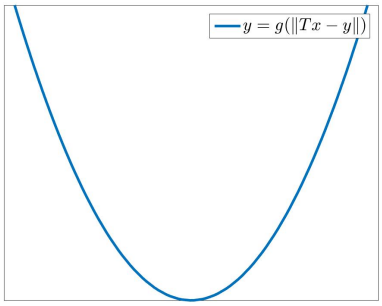


Figure 2.25: Two possible models, one good and one bad

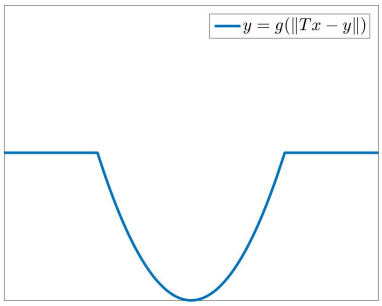
So what about moving away from RANSAC completely and use some error function g not connected to the number of inliers? One may think about using the squared distance to the all the points, i.e. $g(\|Tx-y\|) = \|Tx-y\|_2^2$. But we don't want that because then the algorithm will adapt T to the outliers and make T a bad model for the transform. Since the error function will be of the form as is shown in figure 2.26a, outliers will have more influence than inliers.

Then one may instead think of using some error function that looks as in figure 2.26b. With such function the outliers will only have a constant influence on the model even when the distance is big. This means that they will not really have any influence at all, just as any constant does not have any effect when minimizing an arbitrary function. The outlier error must however still be bigger than the inlier error, because otherwise the optimal solution would be to find a real bad model - i.e. a T - that makes everything outliers. The problem with this function and other similar functions is however that they are hard to solve and not convex. Due to the non-convexity it is possible to get stuck in local minima and local optimization may be necessary.

Thus, calculating inliers is the easiest robust alternative. But it may also be interesting to look at other alternatives, even if it may be harder to use and local optimization may be necessary. There is a lot of literature about robust error norms, robust penalty functions etc. for the interested!



(a) An error function



(b) Another error function

Figure 2.26

2.3.5 Feature vectors and decision

After applying RANAC the remaining matches are used to decide whether the images shall be deemed to represent the same fingerprint or not. This is done in the following steps.

1. Choose some relevant features. Features may include for example number of matches (feature 1), strength of the matches (feature 2) and the convex hull of the matches (feature 3).
2. add a weight to each feature and sum the elements to get a *similarity measure*, the total score. The total score will then be: $total\_score = w_1 \cdot f_1 + w_2 \cdot f_2 + w_3 \cdot f_3$.. Here w_1, w_2 and w_3 are the weights.
3. Compare the total score to a threshold value. If the score is higher than the threshold the images are deemed to represent the same scene or - in the case of fingerprint recognition - the same fingerprint. Otherwise the images will be deemed to represent different scenes.

2.4 Introduction to biometric performance

Let's formalize to some extent what we wrote in the end of the previous section about scores and threshold values. Let T_1 be the previously saved template and T_2 the input template. Then, with terminology from statistics and communication theory, the fingerprint verification problem may be formulated as follows [26, p. 13].
H_0: $T_2 \neq T_1$, i.e. T_2 and T_1 do not come from the same person.
H_1: $T_2 = T_1$, i.e. T_2 and T_1 come from the same person.
The associated decisions are:
D_0: The person is not who she claims to be.
D_1: The person is who she claims to be.

The decision is based on the previously discussed total score - a *similarity score* - and a predetermined threshold. If the score is higher than the threshold the input is accepted as coming from the same finger as the previously saved template. If the total score is lower, the fingerprint is deemed to not come from the same person. From this it is clear that fingerprint verification systems can commit two types of errors:

Type 1: *False accepts* (D_1 is decided when H_0 is true).
Type 2: *False rejects* (D_0 is decided when H_1 is true).

The rate of these errors are called **FAR** (false acceptance rate) and **FRR** (false rejection rate). [26, p. 13]. To evaluate the accuracy of the verification system one has to run tests and collect scores both from templates representing the same finger - the *genuine distribution* $p(s|H_1 = true)$ - and from templates representing different fingers - the *impostor distribution* $p(s|H_0 = true)$.

An example of impostor and genuine distributions are shown in figure 2.27. The figure also contain a graphical illustration of FAR and FRR for a threshold t.

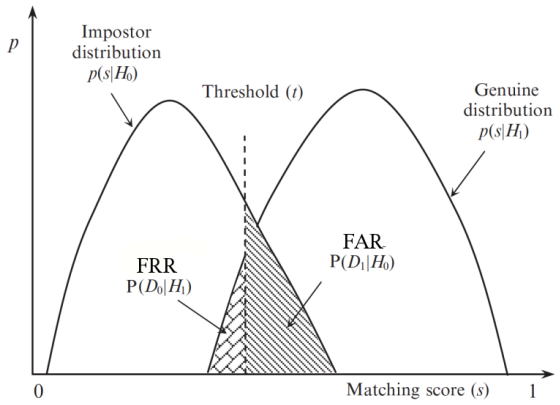


Figure 2.27: FRR and FAR for a treshold t, compare [26, p. 14]

As can be seen from the figure, FRR will be bigger and FAR smaller if the threshold is moved to the right, and the opposite if the threshold is moved to the left. Thus, if it is extremely important that no unauthorized person is accepted the threshold is set higher, and if it is more important that authorized persons are accepted and it is not a very big deal if some unauthorized persons slip through, then the threshold is set lower. Since FRR and FAR as shown are dependent of t, they can be written FRR(t) and FAR(t). An example of FRR(t) and FAR(t) are shown in figure 2.28.

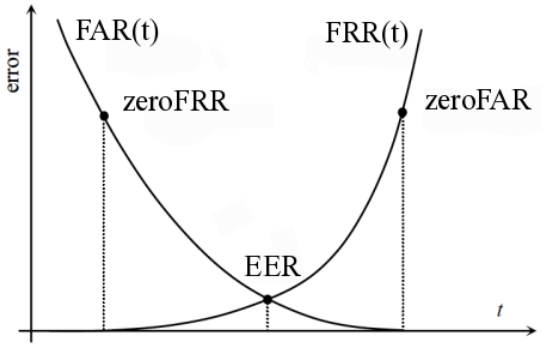


Figure 2.28: FRR and FAR for a treshold t. Also showing *Equal-Error Rate* EER (where FAR=FRR), the *zeroFRR* (the lowest FAR at which no false rejects occur) and *zeroFAR* (the lowest FRR at which no false accepts occur) compare [26, p. 17]

To check the system performance (i.e. FRR and FAR) for different thresholds one can plot a **ROC**-curve (*Receiver Operating Characteristic*. A ROC curve is a graphical plot that illustrates

the performance of a binary classifier system as its discrimination threshold is varied.[9] The ROC curve was first developed during World War II for detecting enemy objects in battlefields. ROC analysis since then has been used in medicine, radiology, biometrics, and other areas and is increasingly used in machine learning.

The ROC curve is a plot of FAR versus 1-FRR for various decision thresholds, which gives a *positive* correlation. However it is also common to use FRR directly, and get a *negative* correlation between FAR and FRR. An example of the latter is shown in figure 2.29.

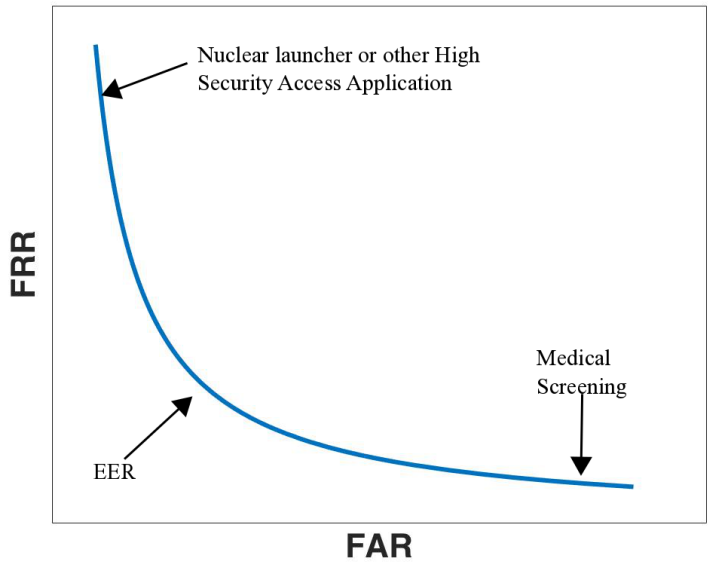


Figure 2.29: Example of ROC curve

As accounted for above, we can see that if false accepts decrease then false rejects will increase. A high FAR may be ok for applications such as medical screenings, where a false accept indicates risk of a disease - which is later confirmed or refuted by a doctor. But if the application is a launch control for nuclear arms, the false accept rate should be low because of the damage a random person could cause.

Of course, the ideal case is the lower left corner: No false accepts and no false rejects. Different systems have different performance, but no system can achieve that.

One can decide on an acceptable FAR and try to get the FRR as low as possible. For a fingerprint verification system in a mobile phone one could for example accept a FAR of 10^{-4}. This corresponds to the probability of finding the right four digit code when randomly choosing four numbers. Then 0.05 maybe could be an acceptable FRR level, i.e. in one try out of 20 a genuine is rejected and has to try again.

[9] `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`

Chapter 3

Technical framework and Experimental setup

3.1 Introduction to the BioMatch Framework

PB has developed the BioMatch Framework (BMF), a collection of code for biometric applications such as physical access solutions.[1] It is implemented in the C programming language and follows the ANSI C standard. The framework can be used both for embedded systems and more powerful operating environments. One purpose of the framework is to separate different aspects of a biometric system into reusable and exchangeable components.[2] Some of the main components of the framework are:

- Objects
- Modules
- Algorithms
- Controllers

Objects are mainly data containers with only few operations such as a create()-function and access to it's data. Examples are Template and Image. A Template is a compact representation of a fingerprint and is used when matching fingerprints. An Image object holds a fingerprint image and has some basic image transformations.

Modules perform specific tasks such as fingerprint template extraction (Extractors), enrollment (Enrollers) and matching (Verifiers). An Extractor takes an image sample as input, perform feature extraction and creates a template. An Enroller takes one or several templates created by the extractor and creates the best possible representation of the finger. A Verifier takes two templates and generates a similarity score. The first template is called *gallery* and the second is called *probe*.

Controllers are the business logic and process of operations, such as the complete flow of enrollment. Controllers uses Modules and Objects.

[1] The information is this section is mainly gathered from PB's guides and references.
[2] Precise BioMatch Mobile/Embedded Developer's Guide

An *Algorithm* is both an Object and a Module. Components such as different extractors, verifiers, templates and certain parameters may be combined in many different ways. In order to relieve developers from having to do that, the Algorithm concept was created. An Algorithm contains combinations of extractor, verifier, etc. and therefore the developer only has to select an Algorithm.

BMM/BME is a set of objects, modules and controllers that implement interfaces from BMF. It provides a framework to support fingerprint template extraction and fingerprint verification.

3.2 The PerformanceEvaluationController and PerfEval

For biometric performance evaluations, PB has developed a Controller called *PerformanceEvaluationController*. The PerformanceEvaluationController provides the framework for evaluations and does all the work, but it need to be configured with Extractor, Enroller, Verifier and some other options for controlling how the evaluation is made. The controller can be used by an application called *PerfEval* (as in Performance Evaluation, discussed below). During evaluation the PerformanceEvaluationController will create gallery and probe templates and send them to the verifier to produce a similarity score. The principles for performance evaluation processes have been described before, but an illustration of the performance evaluation process is shown in figure 3.1.

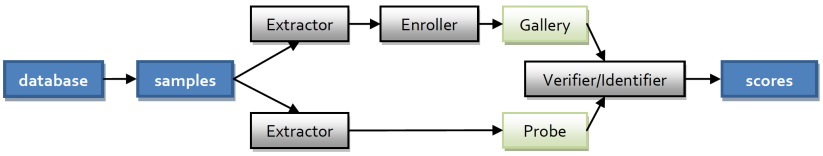


Figure 3.1: Outlines of performance evalutation process

In each evaluation information from the run is saved in text-files. Information about the similarity scores is divided between the files `genuines.txt` and `impostors.txt`. There is also a file named `scores.txt` that contain all performed verifications with the score and the number of genuine and impostor matches that resulted in the score. The files are stored in a separate directory, which is named by the caller. As will be seen below, these text-files are used for analyzing the results.

3.3 Our code

In our work we have used the framework from BMM/BME that implements BMF, specifically the Extractor and Verifier interfaces. Extractors and Verifiers are structs that contain pointers to its member functions. (One effect of putting pointers to the members instead of the members themselves within the struct is that the size of the struct is known beforehand.) One central member function of the Extractor interface is `extract_template` which takes an image of a fingerprint and returns a template representing a fingerprint. The corresponding function in the Verifier interface is `extract_feature_scores` that takes two such templates and returns a vector with feature scores. This function is however not called directly but through the function `external_verify_template` which after `extract_feature_scores` returns assigns a weight to

each feature in the feature vector and returns a total score and a decision as to whether the two templates shall be deemed to represent the same finger print.

PB has provided implementations of the extractor as the struct `external_extractor`, and the verifier as the function `external_verifier`. We have also been provided with the function `external_verify_template` as an implementation of `verify_template`.

Furthermore, we have been provided with *stubs* to the functions `external_extract_template` and `extract_feature_scores`. Those functions are the framework within which we have implemented the object recognition methods described in the theory chapter. Thus, **what** the templates shall contain, and **how** this content is transformed to a vector of feature scores is what we have spent this thesis on. In `external_extract_template` the detection and description of keypoints is performed, and the resulting data from the keypoint descriptors is saved in a template. If two such templates are sent to the `external_verify_template` function it will match the descriptors, remove bad matches with RANSAC and then create a vector with feature scores.

After doing this for two different images, the verifier may be used to match the templates and get a decision as to whether the templates represent the same fingerprint or not. In order to do this the verifier member method `external_verify_template` is called with (inter alia) pointers to the two templates as parameters. This function in its turn calls another function - which one depends on whether the gallery template is a multitemplate or not. In the case of single (1-1) verification as described here, the method `external_extract_feature_scores` is called.

So, how did we implement `external_extract_template` and `extract_feature_scores`?

We decided to use the OpenCV (Open Source Computer Vision) [5] library, a library of programming functions for computer vision. OpenCV is written in C++ and its primary interface is in C++. OpenCV contain functions for inter alia detection, description and matching of keypoints.

Since it is not possible to write C++-code in the C-functions `external_extract_template` and `extract_feature_scores`, we let these functions call two C++-functions that we named `extract_feature_data` and `matcher`. These, are the only C++-functions that are called from the C-code framework, and they are written in the same .cc-file.

To be able to return the key points and descriptors from the C++-function `extract_feature_data` to the calling C-function `external_extract_template` the data is "packed" in a C-array and a pointer to the data is returned. Then the template is created (with a BMM/BME function called `pb_template_create`). The data is later "unpacked" in the matcher.

The main steps of our code follow tutorial code from Matlab[3] and OpenCV[4] and can be summarized:

Step 1: Detect keypoints.
Step 2: Calculate descriptors (feature vectors).
Step 3: Match descriptor vectors using FLANN or BRUTE FORCE matcher.
Step 4: Quick calculation of max and min distances between keypoints to remove "worst matches".
Step 5: Use RANSAC to remove outliers.

[3] `se.mathworks.com/help/vision/examples/object-detection-in-a-cluttered-scene-using-point-feature-matching.html`

[4] `docs.opencv.org/2.4/doc/tutorials/features2d/feature_homography/feature_homography.html#feature-homography`

With OpenCV the detection and descripton using for example SIFT is easy and reduces to essentially the following code:

```
SiftFeatureDetector detector;
detector.detect(image, keypoints);
SiftDescriptorExtractor extractor;
extractor.compute(image, keypoints, descriptors);
```

Accordingly, the essential part of the matching is the following:

```
BFMatcher matcher;
matcher.match( descriptors_1, descriptors_2, matches);
remove_outliers_ransac(keypoints_1, keypoints_2, matches, inliers);
```

Between the second and third row, it is a good idea to add code for sorting out the worst matches, i.e. the matches with greatest distance between the descriptors. One can for example keep matches if the distance $\leq 2.5 \cdot$ meandistance.

To test our code during the implementation we used a test function. The function reads two images from file. It then calls `extract_template` for the two images so that two templates are created. Finally, `verify_template` is called. The main parts of the function is shown here:

```
//testfunction
int main()
{
image1 = read_image_from_file("file1.bmp");
image2 = read_image_from_file("file2.bmp");
external_extractor.extract_template(image1, &template1);
external_extractor.extract_template(image2, &template2);
external_verifier.verify_template(template1, template2, &score, &decision);
}
```

To be able to perform evaluations at all, one first needs a dataset. We have been using a database with "bad" fingerprints. Here "bad" means scars or other damages. The database consists of 160·160-pixel images of the thumbs, index fingers and middle fingers of 19 persons. There are 26 images of each finger. Thus, the database contains $19 \cdot 6 \cdot 26 = 2964$ images.

3.4 Note about Enrollers

Now we have only talked about single (1-1) verification, i.e. when each sample in the database is used as both a single gallery template and a single probe template. That is the simplest case.

However, for very small sensors multiple samples need to be combined into one gallery template representation in order to get an acceptable biometric performance. Otherwise it will often happen that the probe template will come from a different part of the fingerprint than the gallery template, and thus will not match even if it they represent the same fingerprint.

The process of creating a single template from multiple fingerprint image samples from the same finger is called "enrollment training", and is done by the enroller.

When using enrollment training the samples in the database can be divided in several different ways for training and verification. One example of enrollment parameter is *enr_num* which is the maximum number of samples to train each gallery template with. Another example is *enr_enr* that specifies the number of samples that should be part of initial enrollment phase. (This value must of course always be $\leq$ enr_num.) The default value is equal to enr_num. A third example is *enr_dyn* whic specifies the number of samples that should be part of a dynamic update.

We have not had to implement any Enroller ourselves, since we have been able to use existing from BMM/BME.

3.5 More about PerfEval

As said above the application PerfEval is used to run the PerformanceEvaluationController. PerfEval is an executable that can be built from a certain directory. It is composed of the file perf_eval.c and linked to the BMF library. Specifically, it depends on the Performance Evaluation Controller and Extractor and Verifier Modules. The main() function in perf_eval.c first parses the command line for a number of parameters that sets the options to be used by the PerformanceEvaluationController. Second, it calls the PerformanceEvaluationController and the evaluation is performed.

An example of a typical command for running PerfEval with our code is:

```
./PerfEval -algo=algo_alex_johan_1 -scorefiles=1
   -db_idrange=13-16//1-6:13-16//7-12 -enr_num=6 -threads=8 -n
   ../../EVALUATIONS/gftt_sift_5 ../../HW_db_20160119/index.txt
```

The most important parts are the following:

`-algo=algo_alex_johan_1` The Algorithm to be used during the evaluation. Since we have developed our own Extractors and Verifiers we need one Algorithm for each combination of Extractor and Verifier. Alternatively we can use the same Algorithm and replace the Extractor and/or Verifier.

`-scorefiles=1` This is added in order to create the scorefiles for later analysis of genuines and imposters.

`-db_idrange=13-16/1-6/1-6:13-16/1-6/7-12` The part before the colon regards enrollment and the part after colon regards verification. The three range values correspond to personID, fingerID and transactionID. Thus, person 13-16, fingers 1-6, and images number 1-6 of each finger will be used for enrollment (gallery templates), while the same persons and fingers but images 7-12 will be used for verification (probe templates). If any range is left out it means all values. Thus, specifying fingers 1-6 is not necessary since that is the whole range of fingers.

`-enr_num=6` The maximum number of samples to train each gallery template with (as explained above).

`../../EVALUATIONS/gftt_sift_5 ../` Here `EVALUATIONS` is the previously mentioned "run-name" directory, and `gftt_sift_5` is the name of the run. Since the run has 5 as ending suffix one may suspect that previous evaluation runs `gftt_sift_1` ... `gftt_sift_4` have been saved in the directory before.

`../../HW_db_20160119/index.txt` The PerformanceEvaluationController indentifies samples in the database based on the three numbers personId, fingerId and transactionId. (The transactionID is in the range 0-25 and identify images of the same finger). The information is contained in an (arbitrarily named) index file that is placed in the root directory of the database. This part of the command specifies the path to the index file.

3.6 Analyses after running PerfEval

After running PerfeVal we want to analyze the results with the help of the previously mentioned text files. This is done mainly with the help of Matlab (or Octave).

First of all a DET graph may be created for the database.[5] The graph is created with the function `pef_compare_runs`. Parameter to the function is the path to a folder containing information about a finished run (i.e. a number of text-files where the most important are `genuines.txt`, `impostors.txt` and `scores.txt`). Example of a DET graph is shown in figure 3.2.

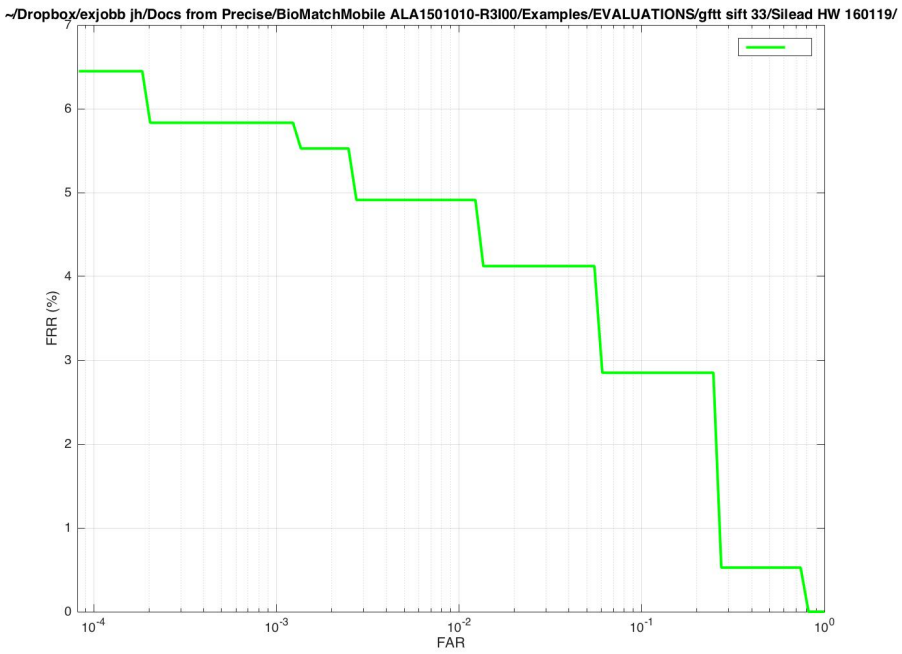


Figure 3.2: Example of DET graph

If more than one run evaluation has been saved, two runs may - as previously mentioned - be included in the DET graph for easy comparison. Then the function `pef_combine_runs` is used. Parameters to the function are the paths to the folders containing information about the two finished runs. Example of a graph with combined runs is shown in figure 3.3. As seen in the graph two different runs may also be combined to get at lower FRR.

Furthermore, there are functions for analyzing genuines and impostors, namely the functions `pef_analyze_genuines` and `pef_analyze_impostors`.

[5] A DET graph is in principle the same AS a ROC curve, but at least one of the x- and y-axes are scaled non-linearly by by logarithmic transformation and thereby yielding tradeoff curves that are more linear than ROC curves.

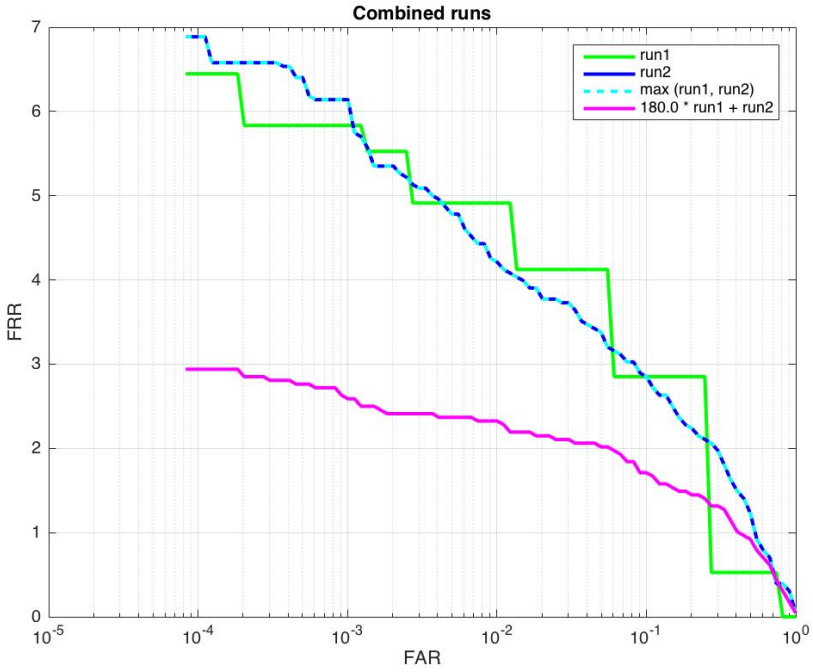


Figure 3.3: Example of combined runs

The first function gives information about the genuine scores in the form of a table on the following form:

Lowest 100 genuine scores

Person	Finger	Attempt	Person	Finger	Attempt	Score
11	6	1	11	6	21	7
2	1	1	12	1	25	7
.	.	.	.	.	.	.
.	.	.	.	.	.	.

The 20 most common persons among the lowest genuine scores was:

Person	Part
14	28.00
13	14.00
.	.
.	.

The 20 most common fingers among the lowest genuine scores was:

Person	Finger	Part
14	5	11
14	4	9
.	.	.
.	.	.

The 6 most common fingers among the lowest genuine scores was:

Finger	Part
Right Little (5)	30.00
Left Thumb (6)	23.00
.	.
.	.

The first table means person 11 finger 6 has matched person 11 finger 6, thus it is a genuine. Attempt 1 and Attempt 21 refers to the transactionId (as explained previously). The score was 7. When running the function there is also plot of the fingerprints in the table. The two first are shown in figure 3.4. This gives the possibility to visually investigate what could be the reason for a low score. From the figure it seems as if "good" fingerprints without scars etc. do not give a good score, which may not be a big problem if the minutiae matcher perform well for such prints. The rest of the information is self explanatory. The information received when running `pef_analyze_impostors` is similar, and we do not account for it here. There is a lot of other functions for analyzing the results, but these are the ones that we have been using.

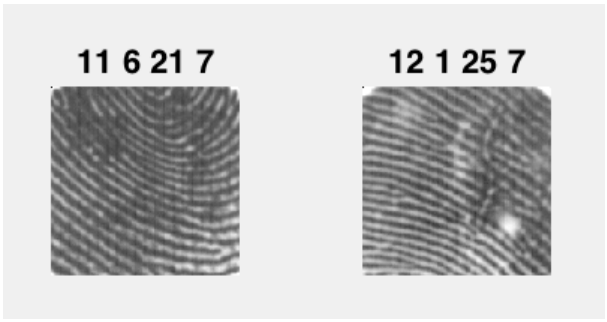


Figure 3.4: Lowest 2 genuine scores

Chapter 4

Results

In this chapter the experimental results will be presented. The result is presented as following. First, we present a brief section about the preprocessing and image enhancement stage. That section is followed by a presentation of the interest point detection, principally by showing examples of how the various detectors perform and additional comments. Then the main results of matching are presented. It will be presented with the biometric performance data and time consumptions for extraction and verification. Further it will also show matching examples in order to demonstrate how well the detectors and interest points are suited for matching fingerprints. The last section will display how well our algorithms can be combined with a traditional matcher such as one based on minutiae detection.

4.1 Preprocessing results

In figure 4.1 there are two examples of preprocessing techniques used in this work. The first column in 4.1 shows the original fingerprint image, the second column shows the result of image sharpening, which is a bit hard to see on paper. Finally, the third column shows the result of histogram equalizing two fingerprints. An empirical study was carried out to find out which enhancement techniques were best suited for each detector. In some cases a combination of image sharpening and histogram equalization worked the best. The algorithm or combination that contributed to the best FRR at 1/10000 was kept. On average, the preprocessing stage lowered the FRR about 1-2%. Below, in table 4.1 the preprocessing algorithm or the combination of algorithms that worked the best for each interest point detector is listed.

4.2 Interest point detection results

In this section illustrative examples of finding interest points in partial fingerprints will be shown. The results will be based on a set of selected fingerprints in order to see how well the individual algorithms perform. Selected fingerprints for these experiments are typical for the database. It is important to note that experiments summarized in tables and commented in general are based on full tests containing all fingerprints. The chosen example fingerprints for interest point detection are shown in figure 4.2. For the various test images with interest points included the preprocessing is activated.

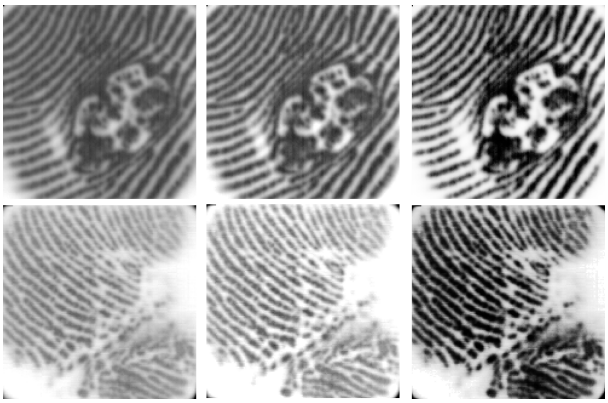


Figure 4.1: Examples showing how preprocessing images can enhance the images.

Detector	Image sharpening	Histogram equalization
SIFT	×	
SURF	×	×
CenSurE	×	
MSER	×	
Harris	×	×
GFTT	×	
FAST	×	
Dense	×	×
ORB	×	
BRISK	×	×

Table 4.1: The preprossesing algorithm or combination of algorithms that worked best for each interest point detector.

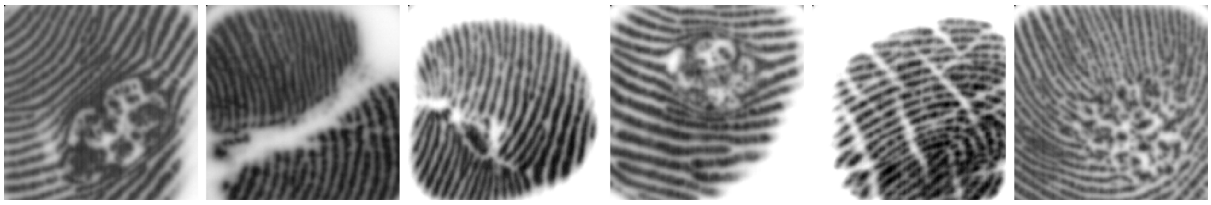


Figure 4.2: Chosen fingerprints for presenting results of interest point detection.

4.2.1 SIFT interest point detection

In figure 4.3 the detected keypoints of the example images are shown. Accordingly to 4.3 SIFT detects keypoints rather uniformly except for the brighter parts of the fingerprint image. The parameters for the SIFT detector are given in table 4.2. n keypoints are selected, and in SIFT the keypoints are sorted by some quality measure and the n keypoints with the highest quality are returned. It is recommended in [25] to use 4 octave layers.

SIFT	Number of keypoints selected	octave layers
	500	4

Table 4.2: SIFT detector parameters.

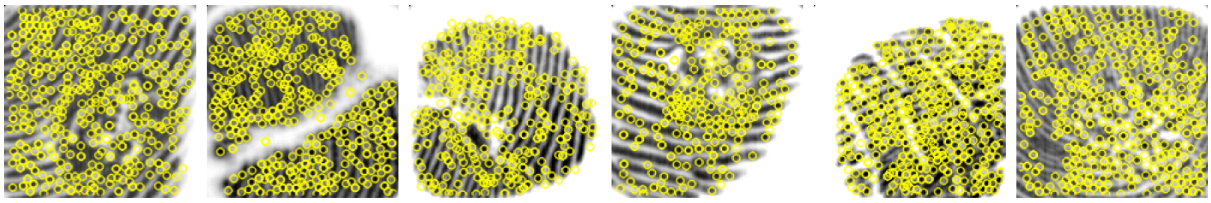


Figure 4.3: SIFT keypoints.

4.2.2 SURF interest point detection

Examples of detected keypoints are shown in figure 4.4. It is noticed that the keypoints are located mostly at locations that seem to be very discriminative for the fingerprints. By trial and error, the best possible parameters for our data set were found and are shown in table 4.3. The set Hessian threshold is rather high compared to recommendations in [11] (600-2500) and OpenCV (300-500). However, it is likely that fingerprint images were not the target when developing SURF.

SURF	Hessian treshold	Octave layers
	3500	4

Table 4.3: Parameters for the SURF detector.

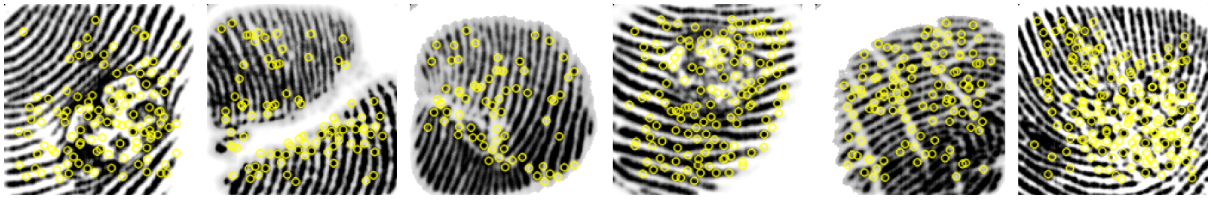


Figure 4.4: SURF interest points.

4.2.3 GFTT interest point detection

The detected interest points returned by the GFTT detector were distributed such that more points are located where more information is available such as in the vicinity of blobs (scars from blisters) and large edges (scars). However, many interest points are located in the border regions of the image of the fingerprint that can cause alarm, these points can possibly be matched with other fingerprints with such interest points. The foremost important parameter when tuning the GFTT detector is the maximum number of corners n. GFTT returns the first n corners, sorted by some quality measure. For our database, $n = 319$. For examples see figure 4.5.

4.2.4 Harris interest point detection

Not surprisingly, the resulting output of the Harris corner detector are similar to the GFTT detector since they are both very similar algorithms. The small differences between GFTT and Harris are different preprocessing approaches and a small change in the number of selected

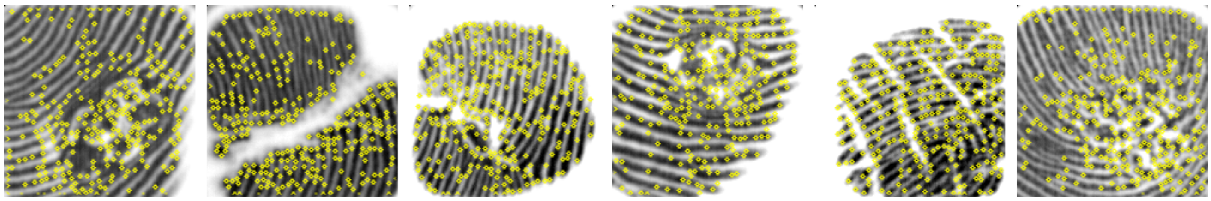


Figure 4.5: Examples of detected GFTT interest points.

corners n. In table 4.4 the parameters for GFTT and Harris are presented. Harris example corners are presented in figure 4.6.

	Preprocessing	n
GFTT	Sharpening	319
Harris	Sharpening & histogram eq.	330

Table 4.4: Parameters for GFTT and Harris.

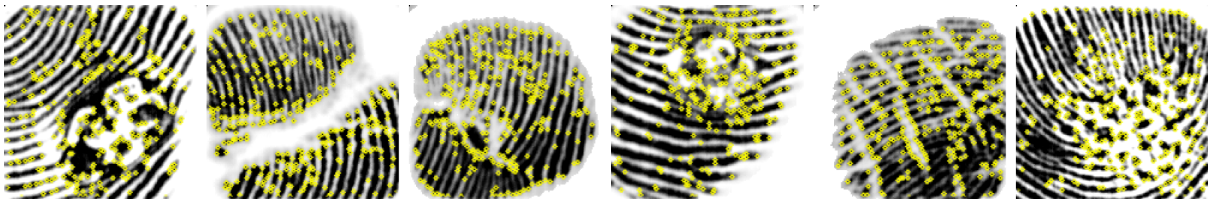


Figure 4.6: Harris corners.

4.2.5 MSER interest point detection

By examining the example interest points in figure 4.7 it is hard to distinguish any particular pattern. What can be said is that only one point is located for a coherent set of pixels I_C, which makes sense if MSER approximates an ellipse around I_C and set the center of the ellipse to the interest point location. After tuning the MSER algorithm, it found the interest points with the parameters in table 4.5.

Minimum size of I_C	Maximum size of I_C
5	2000

Table 4.5: MSER parameters.

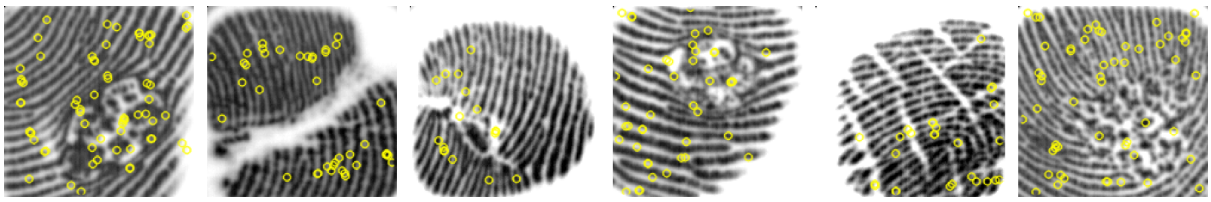


Figure 4.7: MSER interest points.

4.2.6 CenSurE interest point detection

CenSurE does not find any interest points near the borders of the image, which can both be good and bad. The bad point is that we loose valuable interest points if there is a highly discriminative location here. The good point is that we do not get points that are affected by a shadow at the edges or corners of the image. Example interest points in figure 4.8.

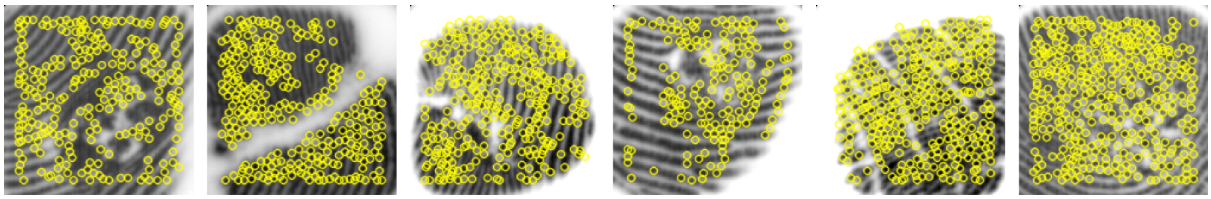


Figure 4.8: CenSurE interest points.

4.2.7 Dense interest point detection

As we can see in the example fingerprints in figure 4.9, there is a fine regular pattern of interest points produced by using the sampling grid. It is hard to judge the quality of using these kind of interest point detection, therefore the biometric performance will expose if this detection algorithm is useful or not. There are a few parameters available for tuning, the most useful is the size between the sampling points, and has been set to 3 and 6, depending on descriptor.

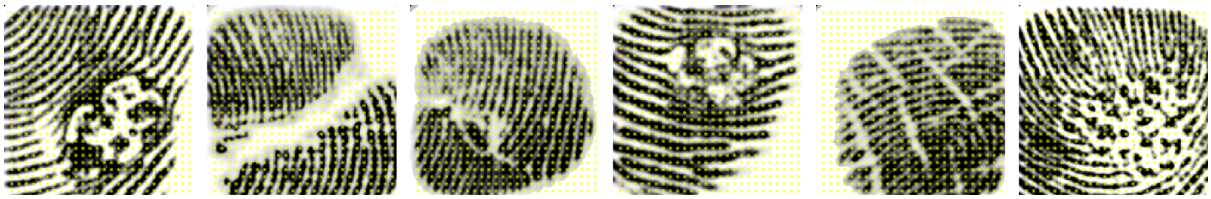


Figure 4.9: Dense interest points.

4.2.8 BRISK interest point detection

The example interest points in figure 4.10 seem to be highly discriminative. What is perplexing is the density of the points, they are very densely located near the center of the example images, but very sparsely or not at all in the outer regions of the images. Thus loss of discriminative information and biometric performance should be expected.

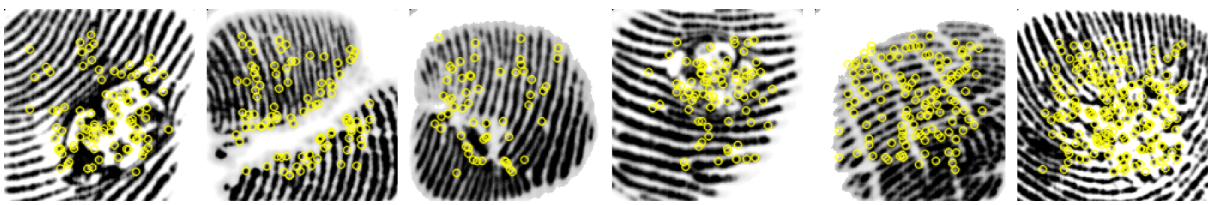


Figure 4.10: BRISK interest points.

4.2.9 FAST interest point detection

FAST has clearly found a lot of corners in the example fingerprints, see figure 4.11. Apparently, the testing of finding a number of consecutive pixels that are either lower or higher than a center pixel has produced the large number of corners. Looking closer to a fingerprint image, many locations have a structure that probably would pass the test. The best possible biometric performance using the settings returned this number of interest points.

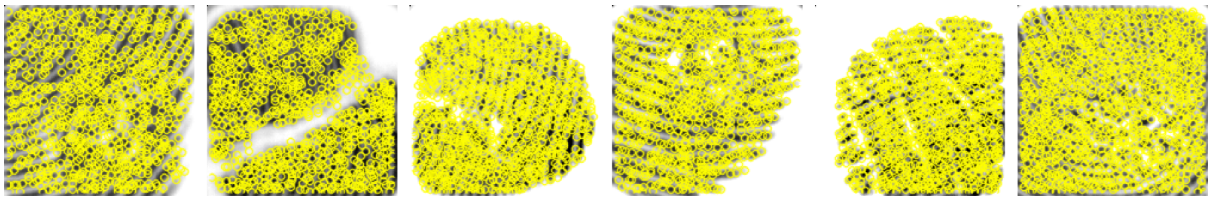


Figure 4.11: FAST interest points.

4.2.10 ORB interest point detection

As we already know, the ORB detector is a modified FAST detector with orientation and scale. Theoretically this algorithm should return better corners than FAST. By comparing the figure 4.11 and 4.14, ORB clearly has detected points that by at least visually are more discriminative than FAST.

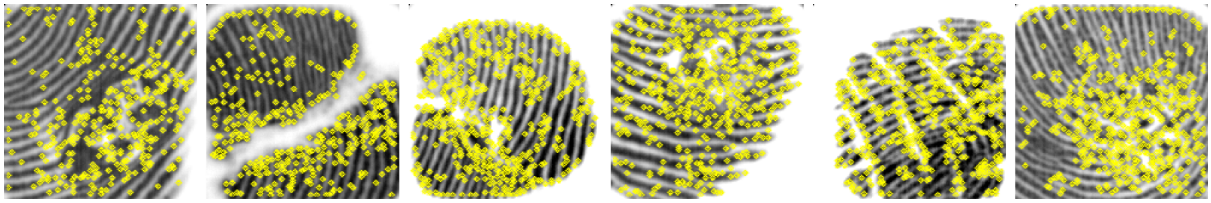


Figure 4.12: ORB intereset points.

4.3 Matching result

In this section we present the main results of matching fingerprints in our database. Each descriptor is given its own section where the result is presented and commented. The descriptors will also be provided with sample matching image pairs, which can be seen in figure 4.13 without matching data. In this figure, the left-most pair is considered as easy to match, which means that it contains a lot of discriminative information. The middle pair is considered harder to match since it is heavily translated in location, the same goes for the last matching pair but it has less corresponding locations. It is important to note that no preprocessing information is shown in the matching pairs, but the matching is carried out with preprocessing. The overall matching result is presented below. We will begin by presenting the False Reject Rate (FRR)[1] performance in table 4.6. Further in table 4.9 the template (interest points + descriptor) extraction times are presented. Finally in table 4.10 the verification (matching time) times are presented. All time measurements are presented in ms.

[1]Measured in %.

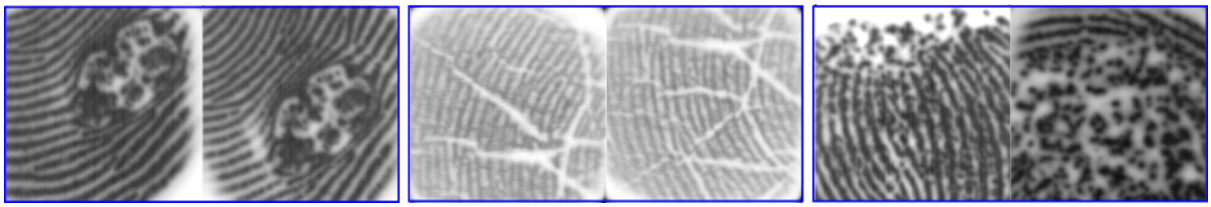


Figure 4.13: Sample images for matching.

4.3.1 False Reject Rate performance

The lowest (better) results for each descriptor row is colored green. Collecting the result was made at a False Acceptance Rate (FAR) of 1/10000. No average value per descriptor is presented. A strong belief is that detectors/descriptors delivering results over 80% are plainly not meant to be used for matching fingerprints. Displayed average values would be contaminated. A gray scale matcher is lightly regarded as well performing if the resulting FRR is under 30% that was set as an overall goal to reach when the project was planned. The best performing

DES/DET	SIFT	SURF	GFTT	Harris	MSER	CenSurE	Dense	BRISK	FAST	ORB
SIFT	9.87	99.47	5.92	6.97	100	16.23	87.85	100	99.78	15.31
SURF	62.50	31.62	98.60	97.50	99.87	99.52	100	78.42	100	92.24
BRIEF	24.08	24.08	21.62	38.51	33.46	64.30	23.68	87.63	18.11	22.24
ORB	DNF	DNF	47.54	24.87	32.54	58.07	28.99	57.11	16.05	54.08
BRISK	14.61	14.61	14.12	11.58	17.32	100	75.66	89.82	100	35.22
FREAK	25.22	25.22	19.82	22.28	25.79	57.59	99.74	90.92	100	20.18

Table 4.6: FRR resuls for all detector/descriptor combinations. The lowest performance result for each descriptor is marked green. Horizontally we see descriptors and vertically the detectors.

algorithm is the GFTT corner detector in combination with a SIFT descriptor. It performs at 5.92%, which is an -0.97% improvement over the conventional minutiae matcher. It is surprising that some non rotational interest point detectors perform better than those that have implementation for rotation invariance. However it is believed that the database may be scant on rotation so it is not very fair to discuss rotation impact in detail. It may be interesting to investigate this matter in future work. On the other hand, the tests are carried out in the same manner as when evaluating the minutiae matcher. It should be noted that eigenvalue detectors are rotational invariant, however they do not compute an angle at the keypoint detection stage which is needed by a few region descriptors for descriptor rotation. This implies that eigenvalue detectors do not support for rotational invariant descriptors.

Scale and rotation dependency

Table 4.7 shows whether a GS-matcher is rotational- or/and scale-invariant. α and β indicate if the combination is rotation or scale invariant respectively. It was not obvious to distinguish if rotation and scale have an impact on the test results. By studying table 4.8 we see similar mean values of the α, β, $\alpha + \beta$ and - groups. We see that scale (β) has lower FRR on average than matchers without scale and rotation invariance protection. The result should be taken with a grain of salt since it is affected by many combinations near 100%.

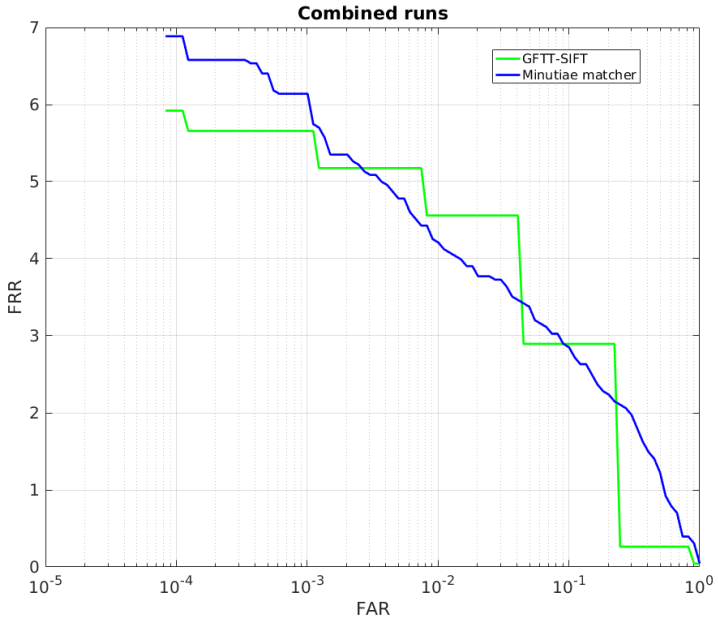


Figure 4.14: Results for the GFTT-SIFT combo (green) and the result for the minutiae matcher (blue).

DES/DET	SIFT	SURF	GFTT	Harris	MSER	CenSurE	Dense	BRISK	FAST	ORB
SIFT	α,β	α,β	-	-	-	-	-	α,β	-	α,β
SURF	α,β	α,β	-	-	-	-	-	α,β	-	α,β
BRIEF	β	β	-	-	-	-	-	β	-	β
ORB	DNF	DNF	-	-	-	-	-	α,β	-	α,β
BRISK	α,β	α,β	-	-	-	-	-	α,β	-	α,β
FREAK	α,β	α,β	-	-	-	-	-	α,β	-	α,β

Table 4.7: Rotation- and scale-invariance experiment.

α	β	$\alpha+\beta$	-
50.91	48.84	50.91	54.55

Table 4.8: Comparison.

Brute-Force- and RANSAC-matching evaluation

In pursuit of finding the presented results, Brute-Force (BF) matching was used for coarsely removing false matches. Tests on Flann matching were also carried out. With Flann, a small time boost was experienced, but instead Flann caused a drop in biometric performance. BF matching is slower than Flann matching but BF outperforms Flann in Biometric performance. Since the focus was set on biometric performance, BF was retained for all experiments. For lack of similar algorithms, RANSAC was the only algorithm that was tested for removing false matches returned from the BF-matcher. Matching by comparing points that were indicated as matches after BF-matching was also tested but did not exceed the performance of RANSAC. However finding such an approach will hopefully result in a boost in time performance. It is left for future work to discuss more alternatives to RANSAC.
Determining a score for a pair of fingerprints was difficult. Several models were tested without

any luck. For instance, adding the difference of convex hulls of the inlier points of each fingerprint in tandem with the total number of matches. In the end it was decided that the number of matches was the only score that was distinct enough to base a decision on. One reason why the convex hull area is a poor score is that it can fluctuate extremely between the fingerprints and may ruin the score.

To understand if the percentage of the matched points after BF- and RANSAC-matching was correlated to the FRR results 1000 fingerprint matchings were sampled and the average of the BF-matches/number-of-points and RANSAC-matches/number-of-points was calculated respectively. Of course, the number of inliers after RANSAC must be correlated to the FRR since this is the true matching score. This can be seen in the right plot of figure 4.15. We can see a tendency that the lower FRR the higher RANSAC percentage. Considering the left part of 4.15 there is a slight tendency that for a low FRR we want a BF percentage of around 50%. One point in each plot maps to a gray scale matcher.

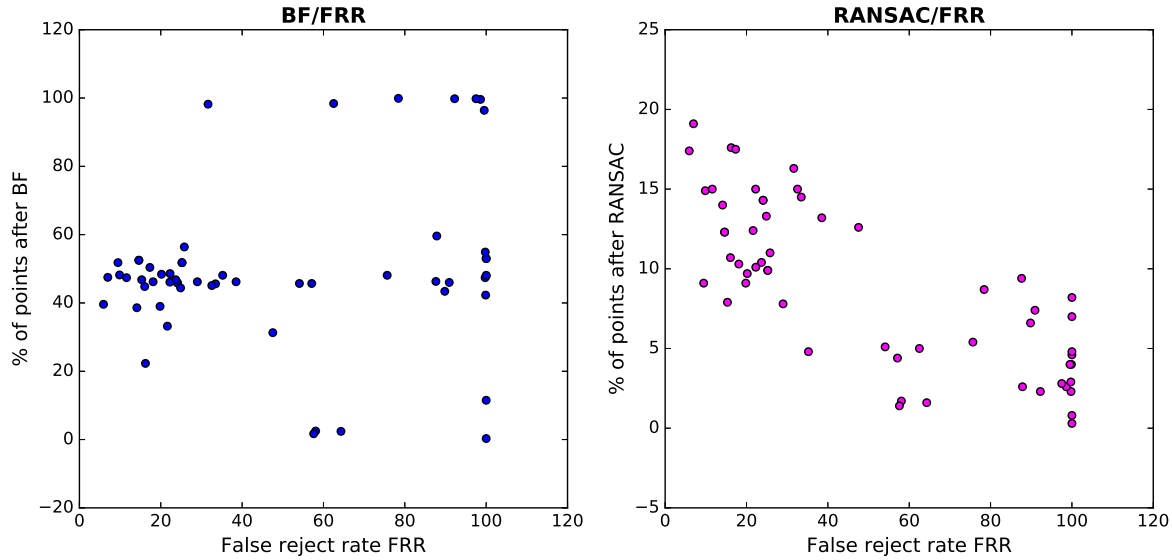


Figure 4.15: The ratio of points regarded as matches.

4.3.2 Template extraction- and verification-time performance

Even though this result is not as important as the FRR result, it is a hint if the gray scale matcher can be implemented in a real application. By that it is meant that extraction and matching of templates should not take too long. Verification time should be considered as more important since this is the time the user will experience when claiming access. Extraction is made once but it is important in a real application that the extraction is not too slow. It could potentially irritate the user. On the other hand, faster extraction and verification may result in poor FRR results. The time performance of extracting templates is presented in table 4.9 and the verification times in table 4.10 .

DES/DET	SIFT	SURF	GFTT	Harris	MSER	CenSurE	Dense	BRISK	FAST	ORB	Mean
SIFT	61	77	18	20	149	17	35	412	130	34	95
SURF	55	34	39	40	60	45	35	491	70	37	90
BRIEF	27	23	13	14	24	2	16	410	17	16	56
ORB	DNF	DNF	18	15	26	2	22	23	21	22	19
BRISK	407	423	405	393	418	409	430	883	498	481	472
FREAK	71	71	58	59	74	46	78	428	71	60	102

Table 4.9: Time for extraction of templates.

DES/DET	SIFT	SURF	GFTT	Harris	MSER	CenSurE	Dense	BRISK	FAST	ORB	Mean
SIFT	256	197	240	241	262	48	299	218	487	282	253
SURF	286	240	269	279	256	295	79	279	299	279	256
BRIEF	225	206	187	193	211	5	207	182	232	221	187
ORB	DNF	DNF	230	215	228	4	246	231	234	255	205
BRISK	251	247	226	256	243	14	268	223	267	297	229
FREAK	324	241	230	243	248	11	278	249	296	233	236

Table 4.10: Time for matching of templates.

4.3.3 Induvidual descriptor comments

SIFT-descriptor

SIFT- and CenSurE-detectors seem to find local extrema with higher descriptive information than the SURF- and Dense-descriptors did. Approximating the Gaussian function with less accurate methods can have an impact on the performance. A blob detector such as MSER may also output locations where the SIFT-descriptor extracts less descriptive information than at extrema locations. The best detectors for SIFT are the eigenvalue corner detectors. It is likely that these detectors find locations where the cornerness value is high and thus makes it a good location for the SIFT descriptor to collect the neighborhood data.

Using the SIFT-descriptor in a real application, poor extraction times will likely not to be experienced, it performs in the midrange. The verification process is on par with other descriptors. Slow verification time is expected when the number of interest points is large, such as when the FAST-detector is used.

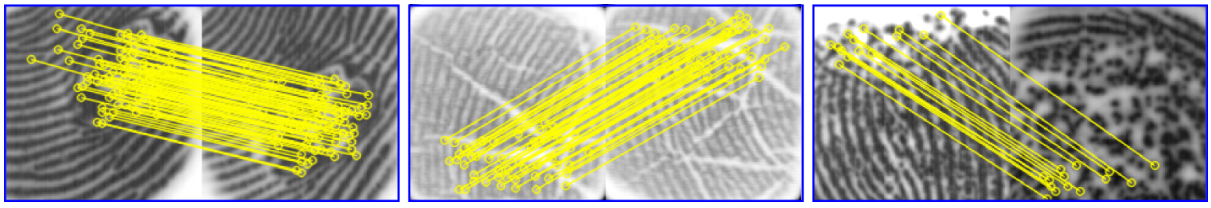


Figure 4.16: SIFT matching sample pairs. No indication of crossing lines, thus no false matches. The interest points were detected with GFTT. This combination has shown to be the best combination.

SURF-descriptor

By studying the FRR result table it should be clear not to recommend any use of the SURF-descriptor when matching fingerprints. There is no gray scale matcher with a SURF descriptor performing under 30%. The relatively high FRR-scores are believed to be a consequence of scaling floating point numbers to and from 8-bit integers. The SURF algorithm in OpenCV is implemented with floating point numbers. Further the local data structures for storing fingerprint templates are written in C-code. For optimization floating point numbers are not supported. A loss of performance was noted when converting the floating point values. This may cause loss of important description information. In figure 4.17 we observe that not a single pair of points has been correctly matched. Yet the plot shows the lowest SURF combination.

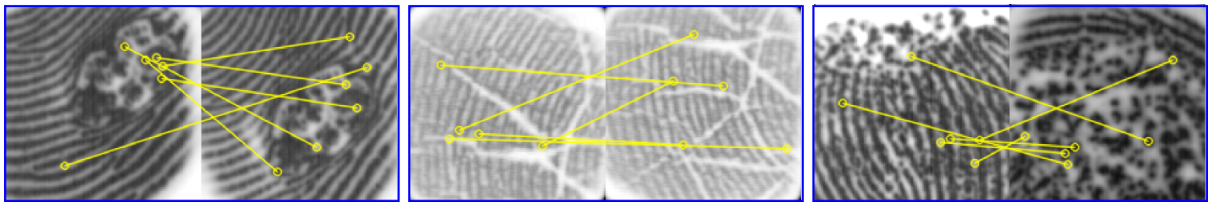


Figure 4.17: SURF matching sample pairs. It is hard to find a correct match.

BRIEF-descriptor

BRIEF was presented as the first binary descriptor in the theory chapter. It was followed by the ORB-algorithm that was an improvement of BRIEF. We observe in the FRR performance that the ORB-descriptor indeed has a lower min-value than BRIEF, but interestingly, the matching results with the BRIEF-descriptor show lower results on average than ORB. If only the min-values of the binary descriptors are compared, BRIEF is on par with the other descriptors. In figure 4.18 we notice that a number of false matches have not been filtered, yet it should not be concluded by studying only the images whether BRIEF is better than any other descriptor. We see that the time of verifying two templates is the lowest on average. We believe that the low matching time is a consequence of the random binary pattern that presumably compare a lower number of points.

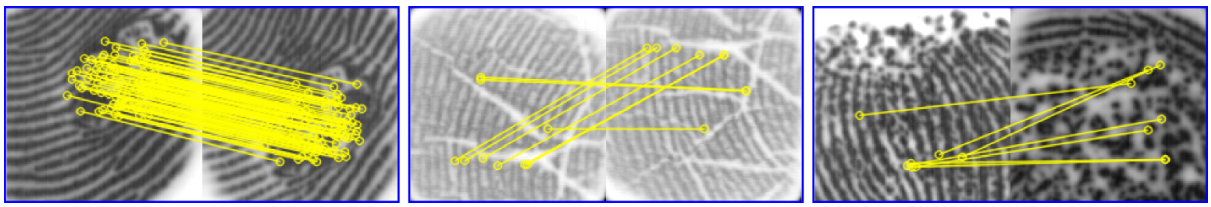


Figure 4.18: BRIEF matching sample pairs. The FAST detector is used.

ORB-descriptor

Matching with the ORB-descriptor was a bit problematic. Two detectors in combination with ORB did not even compile, therefore no FRR-result. Something interesting observed in the

result is that FAST-detection in combination with the ORB-descriptor outperformed the ORB-detector/ORB-descriptor. The latter was to compensate for non-rotational invariance. However, our database may have too little rotation, so it may be unfair to conclude whether the FAST algorithm is better than ORB.

On the upside in using the ORB-descriptor is the template extraction time. ORB has a searching algorithm for reducing the number of comparisons when sampling point-pairs. This may have resulted in low extraction time. It can be observed in figure 4.19 that matching with ORB handles the two first image pairs with no false matches, but fails on matching the last pair.

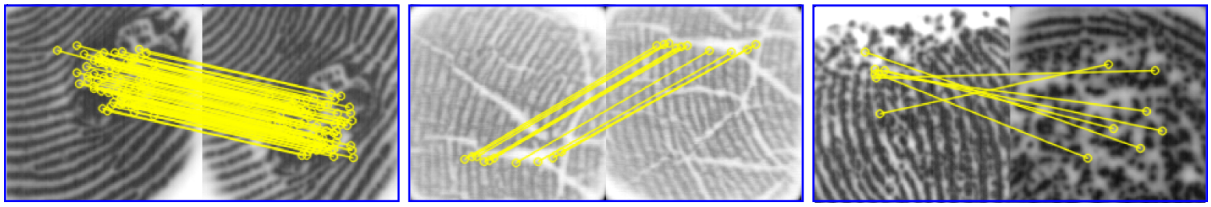


Figure 4.19: ORB sample matching pairs. Left-most matching pair perform similar to other descriptors. Te two remaining matching pairs contain a number of false matches.

BRISK-descriptor

In the theory chapter the BRISK descriptor was proposed as a fast and robust interest point detector and descriptor. It is shown in the result that this is not the case. Using any BRISK detector or descriptor often implies several magnitudes slower template extraction time, often close to half a second. This implies that enrolling fingerprints with any kind of BRISK involvement will be slow. What is positive is that four detectors in combination with the BRISK-descriptor perform under 15% FRR, more than twice as low as the overall goal. Rather surprising is that the eigenvalue corner detectors perform better than the original BRISK detector. Better local information at eigenvalue corner-locations may better be suited for extraction of descriptor-information after all.

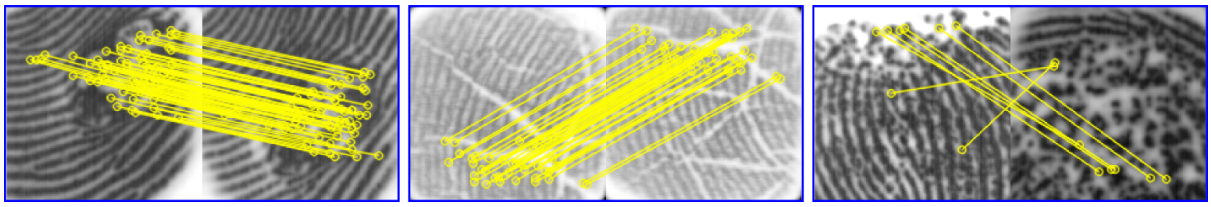


Figure 4.20: Matching with the BRISK-descriptor.

FREAK-descriptor

FREAK's proposed sampling pattern that reminds of the fovea in the human retina may not be the best sampling pattern for fingerprints. In table 4.6 it can be seen when matching with FREAK-descriptors the second highest (19.82%) FRR minimum value is obtained, only the SURF-descriptor produces a higher minimum. The rather poor result is perhaps a consequence

of using the sampling pattern FREAK is proposed to use. Since the intensities around an interest point are rather uniform, important information can be lost if the sample points are too aggressively blurred.

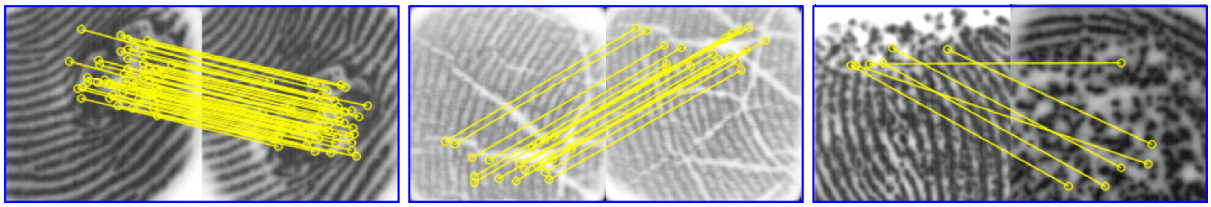


Figure 4.21: Matching with the FREAK-descriptor.

4.4 Combinations of gray scale- and minutia- matchers

The top result presented in table 4.6 were promising. However there was still a belief that an improvement could be made on the FRR scores. Consider the fingerprint pairs in figure 4.22. These fingerprint pairs are a few examples of false matches, they were found by executing a Matlab-script that analyses the performance of an individual benchmark test (C-code). Even though there seem to be structure in the image that could be described sufficiently good with a descriptor there are quite many fingerprints with minutiae information. This motivated experimenting on a combined gray scale- and a conventional minutiae-matcher. Experimenting on the combinations was carried out with an expectation of reduction in FRR-percentage, since the two parts detects separate information. The gray scale matchers explored in this work detects interest points such as corners or local extrema, and matching based on this information will result in a score that can be compared to the score produced by the minutiae matcher. This can be good in a situation when either of the two parts are uncertain whether a match is present or not. Then we can ask the second algorithm of its score and base a decision on this. The fingerprint pairs in figure 4.22 are collected with the best performing algorithm; a GFTT corner detector with a SIFT-descriptor.

It was discovered that 28 out of 60 gray scale matcherss combined with the minutiae matcher reduced the FRR performance. It should be noted that the minutiae matcher resulted in a 6.89% FRR on the givem database. In table 4.11 a TOP-5 table is presented. Notably, the best combination showed to be 4.35% better than the minutiae matcher alone.

Detector-descriptor with minutiae matcher	FRR 1/10k	Improvement
GFTT-SIFT	2.54	-4.35
Harris-SIFT	3.51	-3.38
Harris-BRISK	4.17	-2.72
SIFT-BRISK	5.00	-1.89
SURF-BRISK	5.00	-1.89

Table 4.11: The top-5 best performing gray scale matchers combined with a minutiae matcher. About 47% of the combinations resulted in a FRR reduction that was better than the minutiae matcher's FRR performance.

In figure 4.23 the result of the GFTT-SIFT combined with the minutiae matcher is plotted.

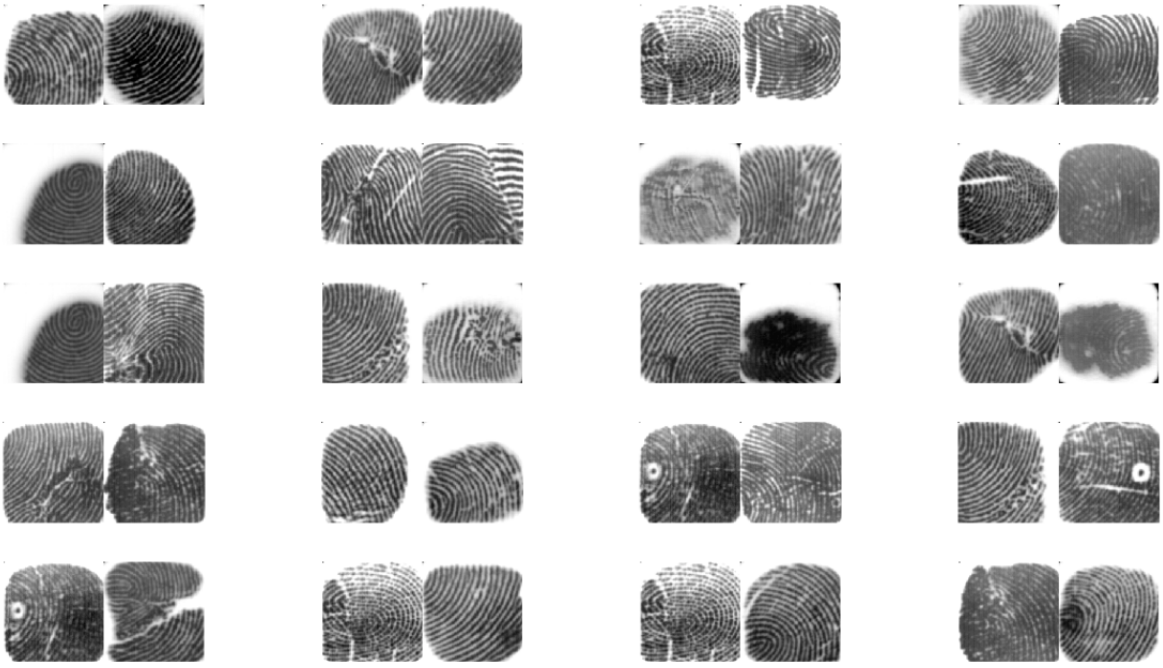


Figure 4.22: A few examples of false matches with the GFTT-SIFT combination. These images presumably contain information that could be found by the minutiae matcher.

4.23 contains four plots:

1. Minutiae matcher result: `run1`.
2. GFTT-SIFT result: `run2`.
3. (Cyan) Scores balanced with: `max(minutiae,GFTT-SIFT)`
4. (Magenta) Scores balanced with: `0.5*GFTT-SIFT + minutiae`.

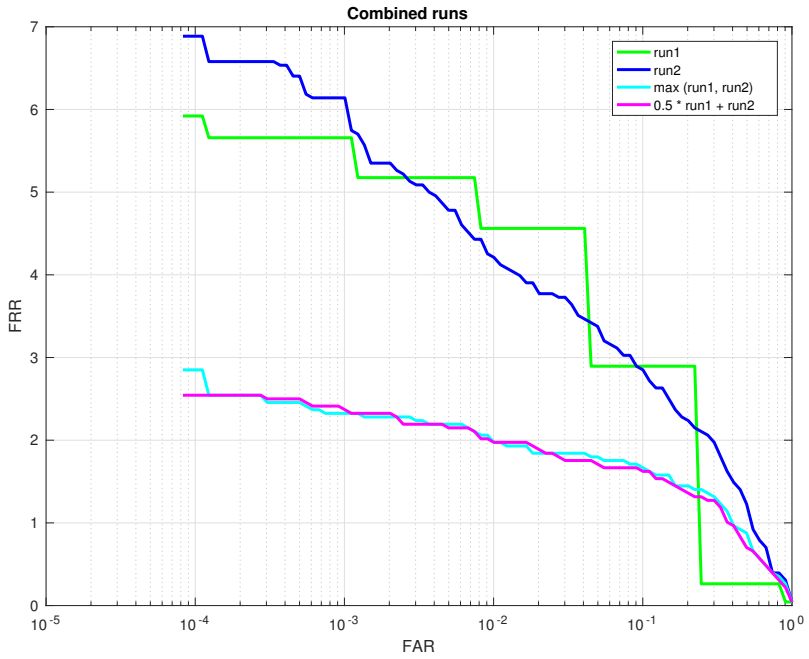


Figure 4.23: Plots for GFTT-SIFT matcher (green), minutiae-matcher (blue), combination of them (cyan and magenta).

Chapter 5

Future work & discussion

In this chapter we discuss possible future work, i.e. additional work that maybe could improve performance. We also discuss the experimental results.

5.1 Future work

5.1.1 Further preprocessing

The implemented image enhancements in this project are based on image sharpening and histogram equalization. It was mentioned that pre-preprocessing introduced an improvement of 1-2%. It is believed that the enhancement of the fingerprint images can be improved further for a reduction in FRR. It is suggested in [39] that a filter structure depicted in figure 5.1 will result in more detectable (SIFT) features. Hopefully this imply that more or better interest points are detectable by all tested detectors in this work. Implementing ridge enhancement was discussed but was feared to remove image information that could be used by the detectors.

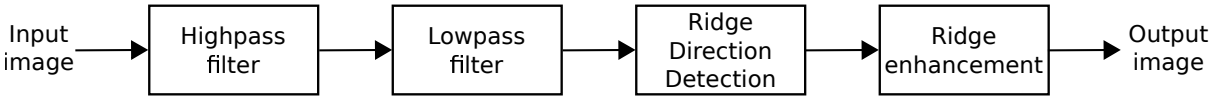


Figure 5.1: Proposed image enhancement filter structure in [39].

5.1.2 Interest point detectors and region descriptors

The OpenCV library offers multiple algorithms for interest point detection and descriptor implementations. All detectors and descriptors found in OpenCV 2.4 are tested in this project. It was mainly used because SIFT and SURF were available. In OpenCV 3.X, these algorithms are moved to a separate library that must be built with Cmake/Makefile differently than with OpenCV 2.4. It was therefore more convenient to use OpenCV 2.4. Unfortunately, rater late in the experimenting phase, new algorithms were found in OpenCV 3.X. It should be interesting to experiment on these algorithms such as KAZE [10] in future work.

It would also be interesting to investigate other computer vision libraries other than OpenCV

such as VLFeat[1]. It has both an API for Matlab and C. Dlib[2] is a second library that includes computer vision algorithms implemented in C++.

KAZE features

KAZE (P. Alcantarilla et al. [10]) is a Japanese word for *wind*. The main point of KAZE features is to, unlike many other detectors, not to use Gaussian blurring. Natural boundaries in images are not respected by Gaussian blurring. With KAZE, interest points and descriptors are found in nonlinear scale space. This makes it possible of an adaptive blurr of image data and will reduce noise and respect object boundaries, *"obtaining superior localization accuracy and distinctiveness"*.

Fingerprint matching using neural networks

It could be worth the effort to try fingerprint matching based on neural networks (NN). *Unfortunately*, literature on matching *damaged* fingerprints are scarce. *Fortunately*, literature on matching *ordinary* fingerprints is plenty. The question is whether it is possible to adapt NN designed for ordinary fingerprints to a system designed for damaged fingerprints.
A successful fingerprint system was implemented based on artificial NN in [7]. 140 images of right index finger of 90 individuals were scanned in the research, which is roughly 4.5 times more images than the database in this work. An average FRR of 0.0022 was presented.

5.1.3 RANSAC

RANSAC is a good algorithm for to removing false matches, but it is also time consuming. Depending on the number of interest points and outliers, RANSAC is responsible for about 70-90% of the verification time. Time performance has never been the most important aspect when experimenting, but none the less it is important. As stated previously it would be interesting to try out different RANSAC transforms. The current implementation has a homography-model which allows for more degrees of freedom than may be needed. Other transforms that may be tested are Euclidean, similarity and affine transforms. It would also be interesting to look deeper into different modifications of RANSAC, or any of the alternatives to RANSAC presented previously.

5.1.4 Combining grayscale matchers

It was presented in the results chapter that combining algorithms can reduce the biometric performance quite significantly. Combining the matching scores of the GFTT-SIFT matcher and that of the minutia matcher resulted in a decrease of 4.35% in FRR compared to the minutiae matcher alone. Finding the promising results introduced a discussion about combining the gray scale matchers and if it could be implemented. Due to lack of time this was unfortunately not implemented. However, with Matlab the scores of the different matchers were combined, and it was possible to find out if an FRR reduction would occur it was implemented. In table 5.2 the top five combinations are presented, and it can be seen that the FRR can be reduced further.

[1] `http://www.vlfeat.org/sandbox/index.html`
[2] `http://dlib.net/imaging.html`

The top performing algorithm is plotted in figure 5.2. It may not be surprising that combining the GFTT-SIFT and SIFT-SIFT matchers would result in an FRR reduction since the detectors search for different kinds of points. Further questions to be answered is if this combination can be combined with the minutia matcher to reduce the FRR even more.
It is with hope that combining gray scale matchers can be implemented, then more in depth analysis can be made of live results. A last question that needs to be answered if it is feasible to have any kind of combination in a real application, concerning the time consumption.

Combinaion	FRR (%)
GFTT-SIFT & SIFT-SIFT	3.42
GFTT-SIFT & ORB-SIFT	4.08
GFTT-SIFT & SIFT-BRISK	4.08
GFTT-SIFT & FAST-BRIEF	5.75
GFTT-SIFT & Harris-SIFT	5.83

Table 5.1: The top five results of combining gray scale matchers.

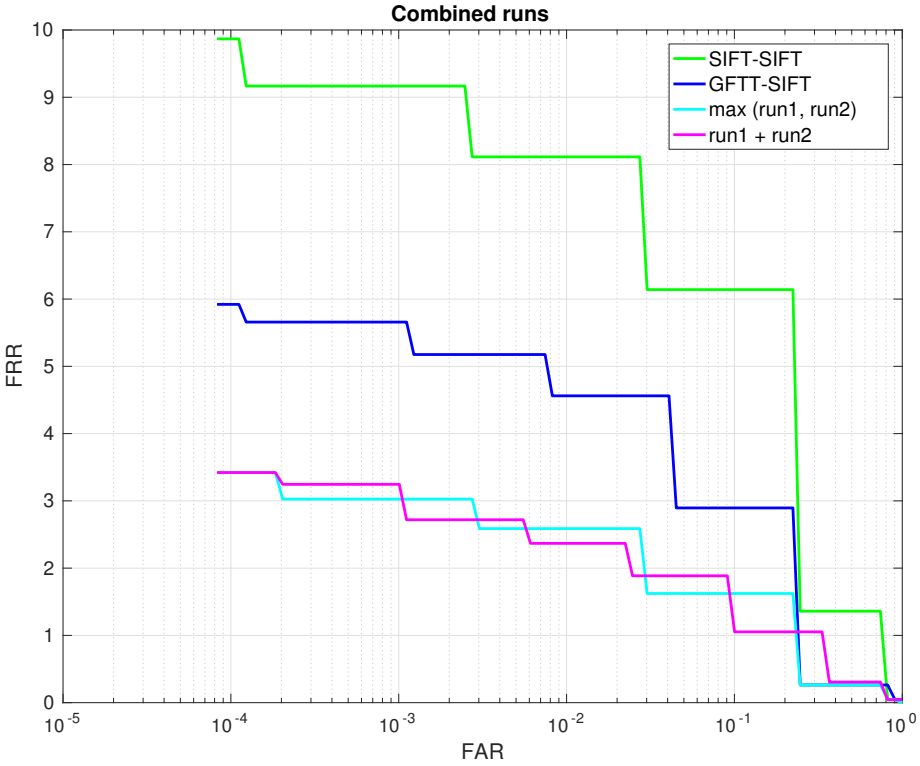


Figure 5.2: Plots for FRR/FAR when combining gray scale matchers.

5.1.5 Investigate rotation- and scale-invariance in detail

A future project is to remove the C++- and OpenCV dependency and implement the best performing system, the GFFT-SIFT gray scale matcher, completely in C-code.

It should be recommended to fully investigate the rotation- and scale-invariance before implementation. This may be carried out by simulating rotation and scale in the performance benchmarks. Then the new test results can be compared with the results without the rotation and scale simulations. It is believed that it is not a big problem, the performance benchmarks are executed with exactly the same preconditions as when the minutiae matcher (rotation invariant) is tested. If it will be a problem, hopefully it will suffice that most mobile users don't add rotation or scale that differ too much from the enrolled images.
The reason why this cause attention is that a eigenvalue detector does not calculate an angle, which SIFT use to rotate the neighborhood when extracting the descriptor.

5.2 Discussion

As mentioned earlier this section will mainly consist of a discussion about the main parts of the project and how it could be improved.

5.2.1 Choice of methodology

The choice has shown to be successful and this is motivated by the fact that the best found algorithm in this project delivered better biometric performance than current state-of-the-art fingerprint matcher on the given database. It is further motivated by the fact that many of the gray scale matchers perform under the the project goal of 30% FRR.
Something that can be improved is image preprocessing. It is assumed that if a few gray scale matchers were sacrificed, that time could have been invested in image enhancement work, and in turn that could have introduced even better FRR performance. On the other hand, what if the eigenvalues detectors were sacrificed? In that case, we would not experience the results of that matcher. After all, we did not know the upcoming biometric performance of each gray scale matcher, so then we would not know which to sacrifice anyway.
A portion of time invested in interest point detection and region descriptors could also have been invested in a second method, for instance deep neural networks. In the end though, we strongly believe that doing a project in this subject in the given time frame, we sooner or later had to choose either a detector-descriptor-matching approach or a neural network matching approach. Any approach is believed to be time consuming, especially when training networks. Moreover, we did not possess the computer power that may be necessary for training neural networks.

5.2.2 Experimental setup

It should be pointed out that combining all the detectors and descriptors was very time consuming. A testing session of 20-40 minutes does not sound very much, but one has to remember that every combination depends on a number of different parameter settings such as a RANSAC threshold, Brute-Force threshold, number of corners to collect from the Harris detector and etcetera. The point is that every tweak often needs a complete performance run to understand if the tweak did improve the results or not. In the end, finding the (probably) best parameters could take a couple of working days. The current version of the GFTT-SIFT matcher took more than a week to tune. To save a little time, a small subset of fingerprints can be evaluated against each other. It is hard to choose such a subset since the fingerprints vary greatly. So

only testing on a third could potentially show completely different figures than testing on the full set of fingerprints. What could have been done differently is to collect a small subset of fingers that were harder to match than the average fingerprint, and run performance tests on this subset. Most certainly, testing time will be saved and hopefully, knowing the FRR of the new subset, then the FRR of the full superset should not exceed.

5.2.3 Results

Detectors and descriptors

A number of interest point detectors has been tested with various results. The detector that accordingly to the test results is the best choice for fingerprints is an eigenvalue corner detector, the Shi and Tomasi corner detector. The reason is believed to be connected to the cornerness value that is connected to eigenvalue corners. A high cornerness value is proportional to how a pixel in the image is connected to a structure [6]. Comparing the Shi and Tomasi- and Harris-detectors both show relatively low results, this supports the claim that eigenvalue detectors perform well for matching fingerprints such as those in the given database.

The FAST-detector is another corner detector. Any FAST/descriptor combination does not perform nearly as good as the eigenvalue corner detectors. In the result chapter it can be seen that FAST finds plenty of corners. Harris and GFTT also find plenty of corners. The key to failure for the FAST detector is that it returns all points that are found, and many of them may be of poor *cornerness* quality. It is possible to decide how many corners that the eigenvalue corner detectors shall return, and the returned corners are the best ranked corners. It is very likely that combining n best ranked eigenvalue corners are superior to combining m FAST-corners of varying quality. Implementing a similar quality measure in the FAST detector may improve the algorithm.

A decently good detector is the SIFT-detector. The ordinary SIFT algorithm showed a result under 10%, and combining extrema locations with GFTT corners in matlab-tests gave an FRR rate of 3.42%. This reduction is likely caused by the detector implementations. SIFT detects extrema location points and GFTT detects eigenvalue corners. Investing further time in selecting better extrema points could perhaps lower the FRR rate.

Blob detection showed poor results when combining it with a SIFT- or SURF-descriptor. A reason for this can be that putting a point in the middle of a set of coherent pixels, will result in a "coherent" descriptor. Matching such descriptors would presumably match to other similar descriptors.

SURF was presented as a faster algorithm than SIFT to compute interest points. Comparing only the SIFT-SIFT and SURF-SURF matchers, it is true that SURF is faster, though combining SURF with other detectors and descriptors showed no significant improvement. In fact, combining any detector with the SURF descriptor showed poor overall results. It is believed to be caused by an implementation issue. The SURF-descriptor is a floating point descriptor and needs extra time for conversion to integer representation. SURF also needs time for taking care of negative values, a byte has to be inserted in the descriptor vector that indicates whether the descriptor value at vector position i is positive or negative. Furthermore, precious information is lost when converting a 64-bit floating point number to and from an 8-bit inte-

ger, thus loss in information may have contributed to the overall poor FRR results of SURF. Further improvements that would most certainly result in better performance is an alternative fingerprint-template implementation that supports ± floating point numbers.

Binary descriptors were also scrutinized in this project. Comparing the min-values for BRIEF, ORB, BRISK and FREAK showed lower FRR results than SURF. However, the binary descriptors do not reach as low as GFTT-SIFT. Description of local information with a binary descriptor may after all not be the best choice. Further optimizing of these algorithms may however result in lower FRR performance.

Matching interest points and descriptors

Overall the Brute Force- and RANSAC matchers work well. There is room for testing other algorithms such as PROSAC and MLESAC in order to hopefully obtain lower FRR rates and faster computation times. However, to lower the FRR further, we think that the way the matching score is computed must be revisited. As of now, the only score is the total number of correct matches returned from RANSAC. A score based on area or circumference of the inliers could be added if that could be computed in a way that improves the current performance.

The RANSAC score has shown to deliver a distinct matching score and can be the difference of granting access or not. As regards the randomness of RANSAC the following may be said. RANSAC does not always find an inlier set. Even if RANSAC finds subsets containing inliers, the best subsets from two different runs may be different and thus the model may be slightly altered if the best runs doesn't contain exactly the same inliers. In the fingerprint matching case it could mean the difference between granted and denied access if the score is close the the threshold. However, such cases would be very rare. First of all, most matches are either high above or far below the threshold and in that case one match more or less has no effect. (Remember that we talk about the threshold for descriptor matching. The score for minutiae matching may very well be under the threshold for minutiae matching. In fact it probably is, otherwise the descriptor matching wouldn't be used to start with.) Only few cases are close to the threshold. We know this since otherwise it would be impossible to reach the high discrimination rates that we present in this thesis. In addition, many other other factors such as for example coverage of the finger, moist, dirt etc. are much more important. To summarize, the randomness of RANSAC does not really have any consequences in practice.

5.3 Conclusion

This project has resulted in a number of gray scale matchers, some of them performed better and some worse. Achieving a best result of 5.92% FRR and 2.54% when combining with the minutiae matcher at FAR 1/10000 is rather good. This result was achieved with the GFTT-SIFT combination, and was even better on the given data set than the currently used matcher. However, it is important not to misunderstand the result. The FRR on its own is perhaps too high for a real application. It must be understood that any algorithm in this project has never been intended to replace any existing algorithm. The gray scale matcher will rather assists them. Most of the approximately 70 - 80 billion fingerprints are not similar to those in the data set. So when such a fingerprint is detected in an application, the current algorithm can ask the gray scale matcher for advice. We have seen that combining algorithms can achieve lower FRR

performance drastically and this should indicate that a gray scale matcher can be a secondary algorithm intended to be used on special occasions.
Before implementing a system completely in C-code it is recommended to investigate if the addition of more orientation and scale will damage the current performance. It is also recommended to experiment on RANSAC substitutes too see if verification time can be reduced. Moreover, a reduction of the biometric performance can presumably be expected if further image enhancement is implemented and also if a decision score with more dependency of the structure in the interest points matched is implemented.

Bibliography

[1] Gil levi's computer vision blog: A tutorial on binary descriptors – part 5 – the freak descriptor. https://gilscvblog.com/2013/12/09/a-tutorial-on-binary-descriptors-part-5-the-freak-descriptor/. Accessed: 2016-08-06.

[2] Lecture slides, computer vision course at lund institute of technology. http://www.ctr.maths.lu.se/media/FMA270/2016/forelas4_1.pdf. Accessed: 2016-08-06.

[3] Matlab mser documentation. http://se.mathworks.com/help/vision/ref/detectmserfeatures.html. Accessed: 2016-08-06.

[4] Opencv documentation: Shi-tomasi corner detector & good features to track. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html. Accessed: 2016-08-06.

[5] Opencv information. http://opencv.org/about.html. Accessed: 2016-08-06.

[6] Vlfeat documentation: Cornerness measures. http://www.vlfeat.org/api/covdet-corner-types.html. Accessed: 2016-08-06.

[7] Mark Abernethy and Shri M Rai. An innovative fingerprint feature representation method to facilitate authentication using neural networks. In *International Conference on Neural Information Processing*, pages 689–696. Springer, 2013.

[8] Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas. Censure: Center surround extremas for realtime feature detection and matching. In *European Conference on Computer Vision*, pages 102–115. Springer, 2008.

[9] 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.

[10] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *European Conference on Computer Vision*, pages 214–227. Springer, 2012.

[11] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[12] Wilhelm Burger and Mark J Burge. *Digital image processing: an algorithmic introduction using Java*. Springer, 2016.

[13] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[14] Ondrej Chum and Jirı Matas. Randomized ransac with td, d test. In *Proc. British Machine Vision Conference*, volume 2, pages 448–457, 2002.

[15] Ondrej Chum and Jiri Matas. Matching with prosac-progressive sample consensus. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 220–226. IEEE, 2005.

[16] Robert Collins. Lecture 06: Harris corner detector. http://www.cse.psu.edu/~rtc12/CSE486/lecture06.pdf. Accessed: 2016-08-06.

[17] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[18] Shoaib Ehsan, Adrian F Clark, Klaus D McDonald-Maier, et al. Integral images: Efficient algorithms for their computation and storage in resource-constrained embedded vision systems. *Sensors*, 15(7):16804–16830, 2015.

[19] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[20] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[21] Dong-Chen He and Li Wang. Texture unit, texture spectrum, and texture analysis. *IEEE transactions on Geoscience and Remote Sensing*, 28(4):509–512, 1990.

[22] Reinhard Klette. *Concise computer vision*. Springer, 2014.

[23] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. IEEE, 2011.

[24] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

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

[26] Davide Maltoni, Dario Maio, Anil Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

[27] 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.

[28] Thomas B Moeslund. *Introduction to video and image processing: Building real systems and applications*. Springer Science & Business Media, 2012.

[29] Marius Muja and David G Lowe. Flann, fast library for approximate nearest neighbors. In *International Conference on Computer Vision Theory and Applications (VISAPP'09)*. INSTICC Press, 2009.

[30] Loris Nanni and Alessandra Lumini. Descriptors for image-based fingerprint matchers. *Expert Systems with Applications*, 36(10):12414–12422, 2009.

[31] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[32] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.

[33] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[34] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[35] Razieh Toony. Project 4: Image stitching. http://vision.gel.ulaval.ca/~jflalonde/cours/4105/h14/tps/results/tp4/raziehtoony/index.html. Accessed: 2016-08-06.

[36] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[37] Marco Alexander Treiber. *An introduction to object recognition: selected algorithms for a wide variety of applications*. Springer Science & Business Media, 2010.

[38] Tinne Tuytelaars. Dense interest points. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2281–2288. IEEE, 2010.

[39] Ru Zhou, SangWoo Sin, Dongju Li, Tsuyoshi Isshiki, and Hiroaki Kunieda. Adaptive sift-based algorithm for specific fingerprint verification. In *Hand-Based Biometrics (ICHB), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[40] Ru Zhou, Dexing Zhong, and Jiuqiang Han. Fingerprint identification using sift-based minutia descriptors and improved all descriptor-pair matching. *Sensors*, 13(3):3142–3156, 2013.