TAMPERE UNIVERSITY OF TECHNOLOGY

Sebastian Rintala

OBJECT TRACKING IN VIDEO SEQUENCES

Master of Science Thesis

Examiner:
University Lecturer Heikki Huttunen

Examiner and topic approved by the
Council of the Faculty of Computing and
Electrical Engineering on
09 November 2016

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Program in Information Technology
RINTALA, SEBASTIAN: Object Tracking in Video Sequences
Master of Science Thesis, 52 pages
April 2017
Major: Signal Processing
Examiner: University Lecturer Heikki Huttunen
Keywords: Computer Vision, Feature Detection and Description, Scale-Invariant Feature Transform, Speeded Up Robust Features, Oriented FAST and Rotated BRIEF, Feature Matching, Object Tracking, Keypoints, Image Processing

In this thesis, three algorithms used in computer vision SIFT, SURF and ORB were compared in terms of object tracking. The purpose of this study was to examine the algorithms' suitability for different types of videos and different purposes such as in real-time systems but also in systems where real-time is not required. Corresponding studies in object tracking were not found with these algorithms and if the algorithms were studied, the comparison was made principally with image pairs. Comparing the algorithms SIFT and SURF to the newer ORB-algorithm will provide new information about its performance. So far, there are not as many studies concerning ORB.

The comparison is made with the algorithms' default- and optimized parameters, which were tested with four videos. Factors taken into consideration are the algorithms' accuracy, computation time and tolerance to scaling, rotation and viewpoint changes. Python programming language and OpenCV-library, which is intended for computer vision, are used in the testing environment.

The results will clarify, that all three algorithms can be used in object tracking. However, selection of the algorithm will depend on its use and the properties of the video. Especially ORB's tracking accuracy improved significantly with the optimized parameters. SIFT's and SURF's accuracy was hardly improved but their computation time was reduced with the optimized parameters. ORB performed the fastest in each video and in average SIFT was the most accurate. SURF was the slowest, which might limit its usage. According to the results, ORB could be recommended to be used in real-time applications with optimized parameters and SIFT for more accurate tracking. SURF was the most accurate in videos with motion blur and therefore could be recommended for situations as such.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
RINTALA, SEBASTIAN: Objektin seuraaminen videosekvensseistä
Diplomityö, 52 sivua
Huhtikuu 2017
Pääaine: Signaalinkäsittely
Tarkastaja: Yliopistonlehtori Heikki Huttunen
Avainsanat: Konenäkö, Piirteiden tunnistus, Kuvantaminen ja Sovittaminen, Scale-Invariant Feature Transform, Speeded Up Robust Features, Oriented FAST and Rotated BRIEF, Objektin Seuraaminen, Avainpisteet, Kuvankäsittely

Tässä diplomityössä vertaillaan konenäössä käytettyjen SIFT-, SURF- ja ORB-algoritmia objektin seurannassa. Tutkimuksen tavoitteena on tarkastella algoritmien soveltuvuutta erityyppisille videoille ja erilaisiin käyttötarkoituksiin, kuten reaaliaikaisiin järjestelmiin, mutta myös järjestelmiin, joissa reaaliaikaisuus ei ole vaatimuksena. Algoritmeja on aikaisemmissa tutkimuksissa vertailtu kuvaparien avulla, mutta tutkimuksia objektin seurannasta SIFT-, SURF- ja ORB-algoritmeja käyttäen ei löytynyt. SIFT- ja SURF-algoritmien vertailu niitä uudemman ORB-algormitmin kanssa tuo lisäksi uutta tietoa sen suorituskyvystä. Tutkimukset ovat olleet ORB:n osalta vielä vähäisiä.

Vertailu tehdään neljän eri testivideon avulla algoritmien vakioparametreilla ja optimoiduilla parametreilla. Vertailussa otetaan huomioon algoritmien tarkkuus, nopeus, sekä sietokyky skaalaus-, rotaatio- ja kuvakulmamuutoksille. Testiympäristössä käytettiin Python-ohjelmointikieltä ja konenäköön suunnattua OpenCV-kirjastoa.

Tuloksista selviää, että kaikki kolme algoritmia soveltuvat objektin seuraamiseen. Algoritmin valinta kuitenkin riippuu käyttökohteesta ja videon ominaisuuksista. Erityisesti ORB:n kohdalla tarkkuus parani merkittävästi optimoiduilla parametreilla. SIFT:n ja SURF:n tarkkuutta ei optimoinnilla juurikaan saatu parannettua, mutta niiden laskenta-aika lyheni. Algoritmeistä ORB oli jokaisessa videossa nopein ja SIFT keskiarvollisesti tarkin. Laskenta-ajallisesti SURF oli algoritmeista hitain, mikä voi rajottaa sen käyttöä. Tulosten perusteella ORB:n käyttöä voidaan suositella käytettäväksi reaaliaikaisissa järjestelmissä optimoiduilla parametreilla ja SIFT:n käyttöä puolestaan tarkempaan seurantaan. SURF:n tarkkuus oli paras tapauksissa, joissa videokuva oli heilahtanut, joten sen käyttöä voidaan suositella kyseisissä tilanteissa.

PREFACE

The master thesis you are about to read, or most likely browse through, was conducted for the Department of Signal Processing, Tampere University of Technology (TUT). Despite challenges, ups and downs during the making this thesis, I am glad for getting the opportunity to study this topic. It has been intriguing and mind-expanding.

Firstly, I would like to thank my supervisor Heikki Huttunen who enabled this research topic, advised and gave many corrections. I would also like to thank him for inspiring me to choose Signal Processing as my major. Although I was uncertain of my major in the first years and I didn't have a clear picture of the future, I can now clearly see that this was the right choice.

Secondly, I would like to thank my employer and colleagues for being flexible during the making of this thesis. For my beloved girlfriend, thanks for all the support, improvement suggestions and for the understanding. I want to thank my family for being supportive during all these years of studying. I also wish to express my gratitude to my friends for all the years shared. Hopefully the reader will find this thesis useful whether it is for studying, personal interest or work related.

CONTENTS

1. Introduction **1**
1.1 Tracking in Computer Vision 2

2. Keypoints and Feature Detectors **4**
2.1 Difference of Gaussian 5
2.2 Determinant of Hessian Matrix 6
2.3 Features from Accelerated Segment Test 8

3. Feature Descriptors **10**
3.1 Scale Invariant Feature Transformation 11
3.2 Speeded-Up Robust Features 12
3.3 Oriented FAST and Rotated BRIEF 14

4. Matching and Transforming **17**
4.1 Hamming Distance 17
4.2 Norms 18
4.3 Homography and Transformation Matrix 19
4.4 RANSAC 21

5. Implementation **23**
5.1 Implementation Phases 24
5.2 Jaccard Index 26
5.3 Software implementation 27

6. Experiments, Results and Evaluations **33**
6.1 Default Parameters 33
6.2 Optimized Parameters 40

7. Conclusions **47**

References **49**

TERMS AND DEFINITIONS

BRIEF	Binary Robust Independent Elementary Features
CPU	Central Processing Unit
DDR	Double Data Rate
DoG	Difference of Gaussian
DoH	Determinant of Hessian matrix
FAST	Features from Accelerated Segment Test
GB	Gigabyte
JPEG	Joint Photographic Experts Group
k-NN	K-nearest Neighbors Algorithm
LoG	Laplacian of the Gaussian
OpenCV	Open Source Computer Vision
ORB	Oriented FAST and Rotated BRIEF
RAM	Random Access Memory
RANSAC	Random Sample Consensus
SIFT	Scale Invariant Feature Transform
SURF	Speeded-Up Robust Features

1. INTRODUCTION

Computer vision application's development and usage has expanded considerably in the past few decades. Nowadays, many computer vision applications are used widely in many lines of business that already show a strong dependency to them. Its applications are used in medical engineering, self-driving cars, security and surveillance, paper industry and hospital robots. Computer vision has enabled dramatic increases in the productivity of a field.

Machine vision is one of the Computer vision's fields. Typically its main components are detection, recognition and tracking. To understand the role of each three components, let's demonstrate it with a hypothetical surveillance system in an airport. To track an object or a human of interest, the surveillance staff would have to manually locate them from a vast monitor system. With the aid of machine vision, the system would first automatically detect every object in the environment and pass their location information into recognition. If the surveillance would be interested in tracking down a human with a red jacket, recognition would distinguish all matching the descriptions. The tracker, which receives this information, would keep tracking the subjects and mark their locations. Therefore, the surveillance staff could easily locate the subjects from their monitors.

In detection there is a model, which has been taught what an object looks like. Detection's job is to detect that object and pass that location information for further processing. In the previous example the object was a human. Recognition's job is to tell what the objects are in an image or a video. There can be one or more unknown objects to be recognized. Usually the objects are labeled or categorized into different groups for instance cars, humans, suitcases, dogs, trees or buildings. In the previous example recognition's job was to locate a human wearing a red jacket and tell the tracker to track that human in particular. Another application would be to distinguish between a child and an adult.

The tracker's job is to track the given object(s) and return their coordinates. There can be multiple objects to be followed. There is no pre-learned model in the tracker and therefore the object's location in the first frame has to be given to the tracker, which is detection's and recognition's job. Henceforth, the tracker should automatically estimate the targeted object's location in the following frames [34]. Another applications would be to track a basketball or a specific player(s) in a game

or to track the moving ball in air and estimate it's landing spot. Usually, previous frames of a video are utilized to improve the tracking results.

Although object tracking has been studied for a long period of time it still poses a challenge in computer vision [28]. One of the main problem is to get sufficiently accurate and reliable results. The decision to choose one technique or algorithm over others, poses another challenge, for each have their own strengths. One could be suitable in general use and another in a particular type of application.

This thesis focuses on object tracking and it's techniques and algorithms. Three different algorithms Scale Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF) and Oriented FAST and Rotated BRIEF (ORB) are compared in terms of tracking accuracy, reliability, computation speed and practicality. Other factors such as environment, brightness, camera movement, viewpoint changes, scaling and video quality are considered in the comparison. As previously mentioned, earlier frames in the video can be utilized to improve the tracking results. In this thesis it is not used as it is not an original component of the algorithms but an addition and with it, the results might be manipulated. The intention is to get as realistic tracking results as possible from the algorithms used.

This thesis aims to define in which kind of applications tracking algorithms can be used and how they each perform. There are many publications about these algorithms and some papers of their results but not many regarding object tracking where all three algorithms are compared. As already stated, this thesis gathers three algorithms and presents their results from the perspective of object tracking side by side, which gives a better understanding of theirs differences. The reader should keep in mind that the results were based only on test cases, which were used in this thesis.

In the following chapters, necessary theories are explained to get proper perspective on what is needed in object tracking using SIFT, SURF and ORB algorithms. Tracking is studied in a general level later in this chapter, tracking components are studied more closely in chapters 2, 3, 4. In chapter 5 implementation, test environment and used testing materials are introduced, examined and explained. Chapter 6 not only presents the tracking results but also the experiments and evaluations are discussed. In the last chapter 7 the results are further analyzed and development ideas and improvements are discussed.

1.1 Tracking in Computer Vision

Purpose of the previous paragraphs was to get the general idea of the differences between detection, recognition, tracking and how they are used in computer vision. The following paragraphs concentrates on tracking and its algorithms. The example presented displays one tracking method. Although others exist, this method was

selected for its suitability in object tracking and comparing the feature algorithms.

Tracking can be divided in the following four main components: keypoints, features, matching and homography [24] [25]. Usually the detector and recognition pass the location information of an object to the tracker. In this thesis manual annotation of the object's true location was used in the first frame of the video. The tracker's first job is to find interest keypoints in the object (step 1 in Figure 1.1), which is known as feature detection. Interest keypoints refer to keypoints in an image, which are valuable in terms of object tracking.

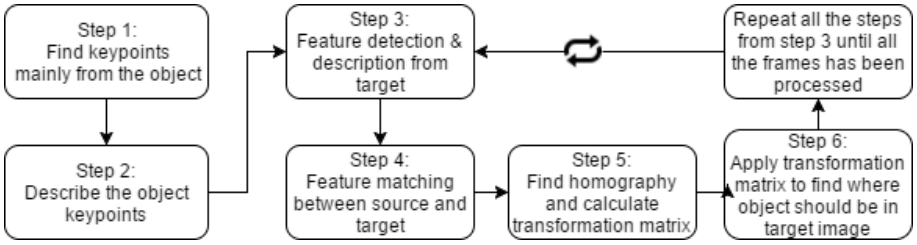

Figure 1.1: Block diagram of the tracker components.

Once the keypoints are found, they are described (step 2 in Figure 1.1). In general this means that different characteristics and properties are calculated from each keypoint [15]. This enables them to be used more effectively in tracking, which is explained later. Described keypoints are known as features. For other following frames the manual annotation of the object's true location is not used. Instead, keypoints are detected and described from the entire area of the following frames (step 3 in Figure 1.1). This determines how well the tracker will perform when following the object. The data is also suitable for comparing the feature algorithms among themselves.

Keypoints description enables them to be compared and matched. Therefore, the next phase in tracking is to compare and make the closest match between the object's features and frame features (step 4 in Figure 1.1). Closest match in this context means two features, which are the most alike using a certain threshold value. Every match is treated as a pair. [31]

Matched pairs are used to calculate a homography (step 5 in Figure 1.1) between an object and the following frame(s) features. In this context homography describes a similarity [25]. Using that information a transformation matrix is calculated, which is used to find the object's location in the following frames (step 6 in Figure 1.1). All these steps are repeated until the last frame or sequence is processed. In the following chapters the components are examined and explained more closely.

2. KEYPOINTS AND FEATURE DETECTORS

One of the early phases in object tracking is to find the keypoints in an image or more precisely the interest keypoints, which are distinct, have a well-defined position and the texture around the point is rich and significant in terms of object tracking [2, p.458] [25]. In object tracking the interest keypoints are located in the object itself, mainly in the edges, contrast differences and shapes. It would be preferable if they were unique compared to the others and therefore an identical keypoint could not be found within the same image.

In object tracking, keypoints from the object are considered as inliers and others are treated as outliers. Outlier can also be defined as patterns in data, which varies from the notion of normal behavior [32, p.308]. Outliers are the most effective factors that decrease the image-based tracking performance [20]. There are few methods how to decrease the amount of outliers. One of which is used in this thesis and explained in section 4.4.

Figure 2.1: Keypoints in the image, shown as green circles.

Feature detector is an algorithm, which enables keypoints to be found automat-

ically from an image. One of the most important tasks for the feature detector is to find as many interest keypoints as possible from corners, edges, blobs and areas where contrast changes rapidly. Figure 2.1 represents one example of a set of interest keypoints in an image. Poor keypoints are usually located on background or flat area where the color and contrast are reasonably constant. Therefore, poor keypoints can be found repeatedly within the same image because they look alike and are not distinct and unique. Another respected characteristic of the detector is invariant to scale, rotation, viewpoint changes, illumination and fast performance. The detector should find the same keypoint from other images as well as under different viewing conditions [1, p.1]. This is known as a keypoint repeatability in feature detection.

2.1 Difference of Gaussian

Difference of Gaussian (DoG) is know as one of the keypoint detectors. DoG is based on Lindeberg's Laplacian of Gaussian (LoG) blob detector, which is one of the first and the most well known. It uses a convolution where first an image is convolved with Gaussian kernel to smooth the images with varying scales to form a Gaussian pyramid or scale-space representation (illustrated in Figure 3.1). From the scale-space the blobs or keypoints are detected. DoG is basically an approximation of LoG. DoG is more efficient than LoG because it uses a subtraction from the Gaussian images to create a scale-space. [35]

To calculate DoG Gaussian scale space images are first created. It consists of a predetermined number of octaves, where each octave consists of a predetermined number of layers. This is represented on the left in Figure 2.2. Each octave has layers where the original image has been scaled with a different scale factor and each layer has been created by smoothing the scaled image with different predetermined values [22]. The next step is to calculate the difference of Gaussian scale space images [37], which is represented on the right side of Figure 2.2. Practically this is done by subtracting two nearby Gaussian images within the same octave, when the resulting images create a DoG scale space [5]. In other words, when the Gaussian kernel is applied to blur the image, the high-frequency information is suppressed. When the two Gaussian images are suppressed, the resulting image will have only spatial frequencies.

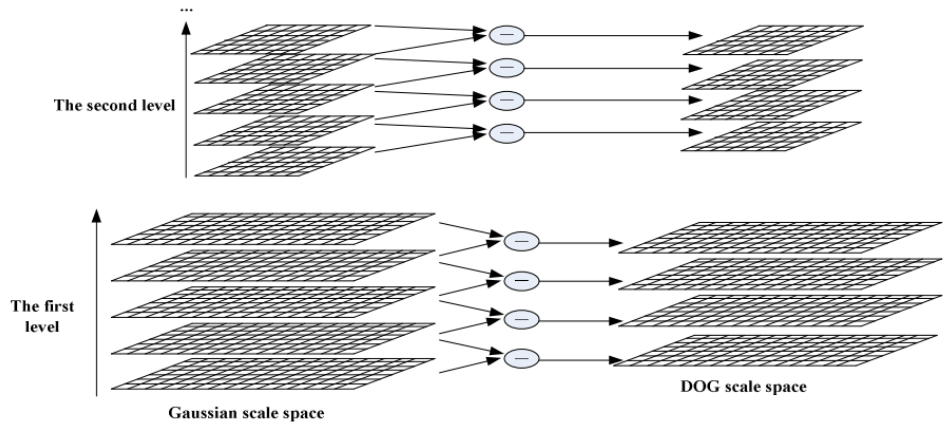

Figure 2.2: DOG (Difference of Gaussian) scale space structure. Picture origin [31].

Spatial frequencies in the final image (in difference of Gaussian) have preserved only corners, the brightest and the darkest areas. Those areas are considered as keypoint candidates and therefore, DoG can be used in keypoint detection. Difference of Gaussian feature detector is used in SIFT, which is explained later in chapter 3.1.

2.2 Determinant of Hessian Matrix

Determinant of Hessian matrix (DoH) is used in SURF (Speeded Up Robust Features) to detect keypoints. It is known for its accuracy [1]. In section 2.1 it was mentioned that DoG is based on the LoG. Determinant of Hessian matrix is also based on the LoG [35] but it approximates the LoG with Box Filters. To sum up, DoH is calculated from the partial derivatives of the image intensities within a box around the pixel.

The function of DoH is to find blob-like structures, which location's determinant is maximum. The blob-like structures are considered as keypoints. The DoH does not use subtraction between Gaussian images but it uses integral images to calculate the Determinant of Hessian Matrix. The use of Integral images makes the computation fast because a simple box type convolution filter is used for the calculation [37]. It can be calculated as

$$Im_{\Sigma}(\mathbf{x}) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} Im(i,j)$$

where Im_{Σ} is the integral image, $\mathbf{x}$ is a location $\mathbf{x} = (x,y)^{\top}$ and Im is an input image. Im_{Σ} is calculated from the sum of all pixels in the input image within a rectangular area that is formed by the origin and the $\mathbf{x}$ [1].

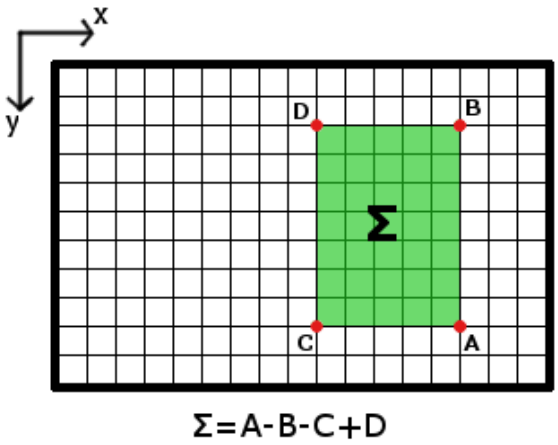

Figure 2.3: The use of an integral image. Σ is a sum of the intensities over a green rectangle in the image.

Figure 2.3 represents a situation that demonstrates how the integral image is used, where Σ is sum of the intensities over a rectangle area. When the integral image is calculated, three additions are needed to compute the Σ within a rectangular area of any size [1].

Hessian Matrix can be defined with the following equation

$$Hess(\mathbf{x}, \sigma) = \begin{pmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{pmatrix}, \tag{2.1}$$

where $\mathbf{x}$ is an image's point $\mathbf{x} = (x, y)$, σ is scale, $L_{xx}(\mathbf{x}, \sigma)$ is the convolution of Gaussian second order derivative with the image I_m in point $\mathbf{x}$. L_{xy} and L_{yy} are defined accordingly. Gaussian second order derivative can be represented as $\frac{\partial^2}{\partial x^2} g(\sigma)$, where g is the Gaussian function. [1]

Calculating the Hessian matrix with equation 2.1 is slow and takes computational power. A faster and more efficient way to do this calculation would be with the box filters, which are approximations of the second order Gaussian derivative. The last two boxes from Figure 2.4 represent those box filters and the first two the second order Gaussian partial derivative. The box filters can be evaluated using integral images. [1]

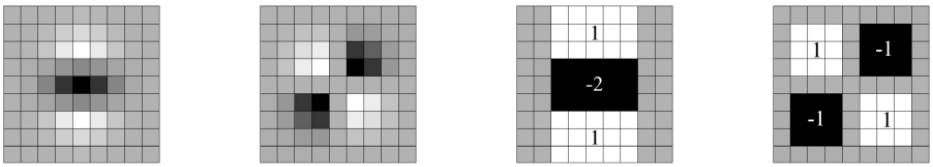

Figure 2.4: Two first boxes: Gaussian second order partial derivative in directions xy (L_{xy}) and y (L_{yy}), the last two boxes represent the approximation for the second order Gaussian partial derivative in directions y (D_{yy}) and xy (D_{xy}). Picture origin [1, p.3].

Now the equation can be written as

$$det(Hess_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2, \tag{2.2}$$

where $det(Hess_{approx})$ is a blob response in the image at location $\mathbf{x}$. D_{xx}, D_{yy} and D_{xy} are approximations of a Gaussian and are also know as box filters. They are used to compute the blob response maps. The term w is a relative weight, which is used to balance the expression for the Hessian's determinant. [1] Basically the determinant is calculated from the image's intensities within the size of the box filter around the pixel. The actual keypoint is selected from where that determinant is maximal. There is an option to use a threshold value for Hessian to decide if the point is accepted as a keypoint [37].

2.3 Features from Accelerated Segment Test

As FAST's (Features from Accelerated Segment Test) name indicates it is a fast algorithm for detecting corners in an image. It is faster than it's two biggest rivals DoG and Harris Detection [27]. FAST is used in ORB (Oriented FAST and rotated BRIEF) to detect keypoints in images.

The algorithm is based on a method developed by Rosten and Drummond. It compares 16 pixels with a candidate pixel p where the 16 pixels were taken from a circle of a fixed radius around the corner candidate p [14] as shown in Figure 2.5. Term N is a set of the adjacent pixels of those 16 pixels and t is a threshold value. There are two options how to determine if the candidate pixel p is chosen as a corner. In the first option the candidate pixel p will be chosen as a corner if its intensity value is darker than all the pixels of N plus a threshold value t. In the second option candidate pixel p will be selected if its intensity value is brighter than all the N pixels minus threshold value t. [37]

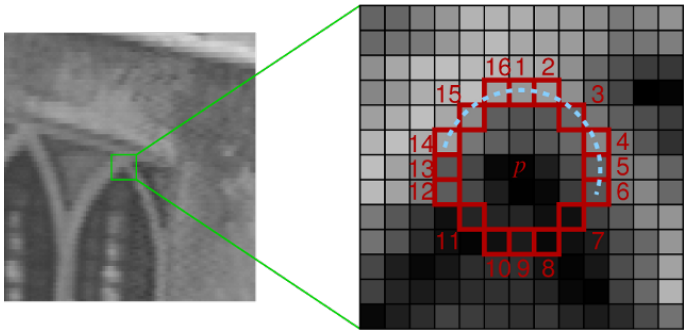

Figure 2.5: Circle of 16 pixels in FAST. Picture origin [26].

FAST's downside is that it does not have an orientation component, which is needed for extracting the invariant orientation feature. That being said, it does not cause a problem because the component can be added afterwards. Another downside is that FAST is not robust enough for selecting interest keypoints if there is too much noise.

3. FEATURE DESCRIPTORS

In object tracking, keypoints are not sufficient enough by themselves to create a fully functioning tracking system. This is why all the keypoints have to be presented or described by the feature descriptor. Data from the descriptor is stored in feature vectors where every element describes the characteristics of each keypoint. It will also allow a more effective way to compare features with each other, which increases the success rate of feature matching. To gain more benefit from the keypoints, their rotation and scale can be determined. There are differences in the way they are calculated but the main idea is that all the algorithms create invariant features to scale, rotation and some of them resistance to noise. There are different ways to describe the keypoints while in this work, SIFT-, SURF- and ORB-algorithms were used.

A good feature algorithm is hard to define but there are few main norms and properties that make a quality feature algorithm. Like previously mentioned, interest keypoints are located on corners, edges and blobs; it is also important that interest keypoints are found in terms of the final results. Basically this will build a base for object tracking. The feature should also be distinct enough and stand out compared to others, which is important when the features are matched with each other. One of the feature descriptor's job is to describe the keypoint and to make its feature distinct. If the features are distinct they can be matched better and this will have a straight impact to the final results. If the match is incorrect the final result itself is incorrect. Matching and feature distinctiveness are connected to the feature's repeatability where same points should be detected from other images despite viewing changes. Figure 4.1 presents an example of feature repeatability. Matching is explained more closely in chapter 4.

Favorable factors of a feature algorithm are invariance to illumination, scale, rotation and viewpoint changes. If the feature algorithm is invariant to all these factors the final results should not be changed. One example would be to track a player from a video, which has been recorded with free hand. If the sun occasionally shines directly to the camera the illumination will change. If the cameraman zooms in and out it will scale the video. Recording with free hand will slightly rotate the video. All the feature algorithms have their own methods for building invariance to the factors.

One property of a descriptor is its usability in real-time applications. It is crucial for the feature that it is distinctive and that the descriptor is fast enough to perform in real-time especially regarding low-energy devices. SIFT is known for its performance in terms of reliability and feature distinctiveness but it might have a longer computation time compared to SURF or ORB [18]. ORB is known for being an effective alternative for the widely used SIFT and SURF [29]. SURF is known for its speed while maintaining its performance [1]. The three descriptors are explained more closely in the following sections.

3.1 Scale Invariant Feature Transformation

SIFT (Scale Invariant Feature Transformation) was first introduced in 1999 but David G. Lowe introduced it again more comprehensively in 2004 [10]. SIFT consists of general parts of feature detection and description. SIFT is invariant to scale changes but also to rotation [22], which means that the feature can tolerate these variation and they should not affect the final results. The most significant features are selected taking into account their invariance to scale and rotation but also illumination is considered, which makes SIFT also partly invariant to small changes in illumination.

Scale-space enables invariance to scale changes, where keypoints and their equivalent points are found from other scales in matching [22]. Scale space is calculated by building a continuous and linear scaling function where Gaussian filter is used to smooth images. The image pyramid (demonstrated in Figure 3.1) is constructed from the smoothed images by sub-sampling. The image pyramid consists of different scale sizes. Another way to do this would be to calculate all possible scales from an infinite number of images, which would be slower and practically impossible to achieve. Therefore, calculating the scale-space will make scale invariance possible and take less computational time.

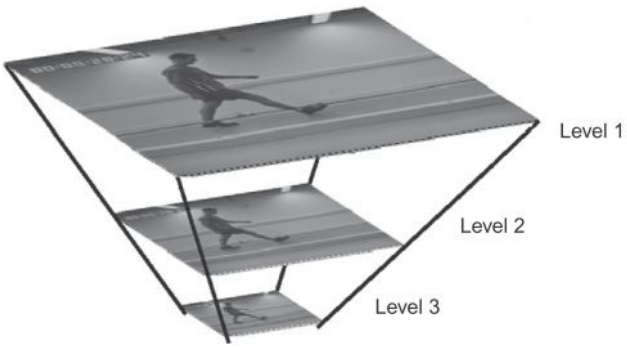

Figure 3.1: Image pyramid of different scale sizes. Picture origin [23, p.193].

In order to find keypoints SIFT uses a DoG (Difference of Gaussian) method, which uses previously created scale-space images. Actual keypoints are selected from the keypoint candidates based on their stability. The keypoint candidates are selected as local maxima or minima of the DoG images from each scale. In order to do this, each sample point or pixel is compared to neighbor pixels in DoG in a 3×3 area. The candidate pixel selection is based on the pixel comparison. If the pixel value is the maximum or minimum it is selected. Next step is to filter the candidate keypoints, which have low contrast or are not located in edges. The filtering is done by making a detailed fit to the neighboring data for location, ratio of principal curvatures and scale. Principal curvature in this content is the eigenvalues of the candidate keypoint's edges. [22]

Keypoint orientation assignment is based on directions of local image gradients, which enables invariance to the rotation. The gradient directions and magnitudes are calculated for every pixel around the keypoint's neighbor in the Gaussian-smoothed image. From this information an orientation histogram is created with 36 bins where the bin's interval is 10 degrees. From this histogram the peaks are treated as dominant orientations and more specifically keypoint's orientation information is selected if the local and highest peaks are within 80% of the highest. [22]

The preceding three paragraphs describe how keypoints are found from particular scales and how the orientation is assigned to them. Descriptor vector is calculated from each keypoint keeping in mind that the descriptor is a highly distinctive and partly invariant to illumination and viewpoint changes. A model that is used is based on biological vision, particularly for complex neurons in primary visual cortex. These neurons respond to a gradient, which has a specific orientation and spatial frequency. [22] Therefore, the model is suitable for constructing the feature vector.

The orientation and magnitude of the gradient is calculated from each image sample point in a 16x16 area surrounding the keypoint (in the left image of Figure 3.2), which are weighted by a Gaussian window. From the 16×16 region the size of 4×4 subregions are summarized, which creates orientation histograms with 8 bins each (shown in right in Figure 3.2). The actual feature vector consists of all the values from the histograms. [22] Brute force is used for matching the feature descriptors with the least square distance (Euclidean distance as in the equation 4.2). The brute force method is described in the beginning of chapter 4.

3.2 Speeded-Up Robust Features

SURF (Speeded-Up Robust Features) was first introduced in 2006, which was 7 years later than SIFT. The main innovator for this algorithm was to make a faster keypoint detector and descriptor than SIFT while not sacrificing performance [37]. For keypoint detection SURF uses the determinant of Hessian matrix method de-

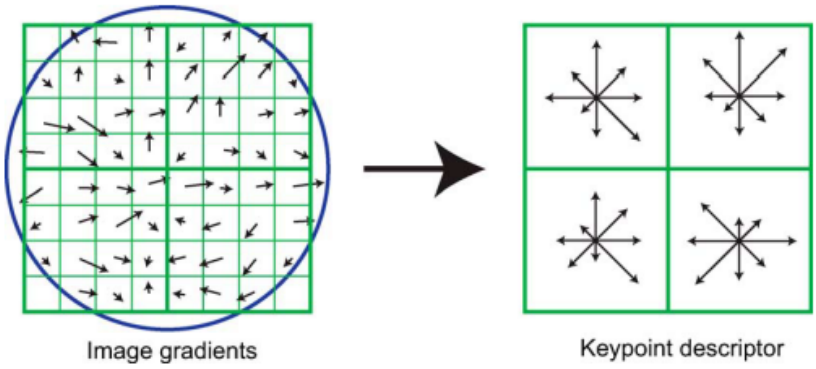

Figure 3.2: Left image demonstrates the orientations and amount of gradients, which are calculated within the 16x16 region and weighted with a Gaussian window. Right image demonstrates what the actual feature vector looks like in the graphical form. It consist of weighted histograms, which are calculated from the left images 4x4 regions. Picture origin [22, p.101].

scribed in section 2.2 where the keypoint is chosen from the locations where the determinant is maximal [1]. Detection itself doesn't make the keypoint invariant to scale and rotation and for this SURF has its own methods.

For making SURF invariant to scale it uses a scale space like SIFT. The main difference is that SURF does not build a scale space pyramid by repeatedly smoothing images with a Gaussian filter or by sub-sampling in order to build a higher level of pyramid like Figure 3.1 represents. For making a feature invariant to scale, SURF uses different sizes of box filters and integral images, which were created while building the determinant of Hessian matrix [37]. Different box filter sizes are applied to the integral image to filter it gradually with bigger masks [1]. The scale-space is created by only up-scaling the filter size so there is no need to change the image size contrary to SIFT. This method is more efficient in terms of computational time [18]. Because downsampling is not needed there is no aliasing. Box filters preserve high frequency components, which can reduce the scale-invariance especially in different viewpoint scenes [1].

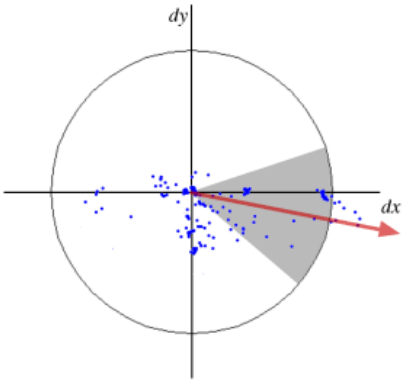

Figure 3.3: Haar wavelet response represented as points in a coordinate system, where sliding orientation windows size is $\frac{\pi}{3}$. Picture origin [1, p.6].

For making rotation invariance, Haar wavelet responses are calculated in x- and y-directions. The responses are represented in space or more precisely in a coordinate system with a circle, which can be seen in Figure 3.3. The orientation is estimated with the sliding orientation window of size $\frac{\pi}{3}$ where the sum of the responses are calculated within this window in horizontal and vertical directions. Orientation of the keypoint is defined with two of the summed responses and the longest vector of all windows. [1]

From the descriptor's point of view SURF is based on the sum of Haar wavelet responses. Before calculating the Haar wavelets square regions are constructed and centered around the keypoints. The square orientation is calculated according to the previous paragraph, which is used to orient each square region. To preserve important spatial information the square region is split up into smaller 4×4 square sub-regions from which the actual Haar wavelet responses are calculated from 5×5 regularly spaced sample points. Haar wavelet responses in horizontal d_x and vertical d_y directions are summed up from every sub-region of which the final feature vector is consisted of. [1]

For matching, SURF uses the sign of Laplacian for making indexing fast [1]. This means that features are compared to each other only if their contrast is similar [37]. Brute force is used for the matching process with the least square distance (Euclidean distance, presented in the equation 4.2).

3.3 Oriented FAST and Rotated BRIEF

ORB (Oriented FAST and Rotated BRIEF) was first introduced in 2011 by Ethan Rublee. One of the developing inspirations for ORB was to make a faster feature descriptor while maintaining performance efficiency. Also, one of the reasons was to have a theoretical chance of using feature descriptors in real-time applications.

ORB is a binary descriptor, which is based on the FAST (Features from Accelerated Segment Test) feature detector and BRIEF (Binary Robust Independent Elementary Features) feature descriptor with some modifications. It is invariant to scale, rotation and resistant to noise. ORB is free of license restrictions while the usage of SIFT and SURF has a license cost. [29]

ORB uses FAST to find the keypoints. For filtering the undesirable points Harris corner is used to select the top N points from each keypoint. Because FAST does not provide the orientation component it has to be created in another way. This is achieved with intensity centroid, which measures the corner orientation assuming that the corner's intensity is offset from its center. Orientation in this case means the direction from corner to centroid. The result is called $oFAST$, which is also known as FAST keypoint orientation. FAST does not provide the scale invariant feature itself so it has to be created with a similar technique as in SIFT, which is by creating a scale image pyramid. [29]

BRIEF is a feature descriptor, which finds binary strings and uses them as features. It makes binary tests for smoothed images by selecting location pairs on a smoothed image patch and making pixel intensity comparisons. Pixel pairs are selected with a unique way where first random sampling is used to form a sampling geometry GII in figure 3.4, which determines the location of the pairs in a smoothed image patch.

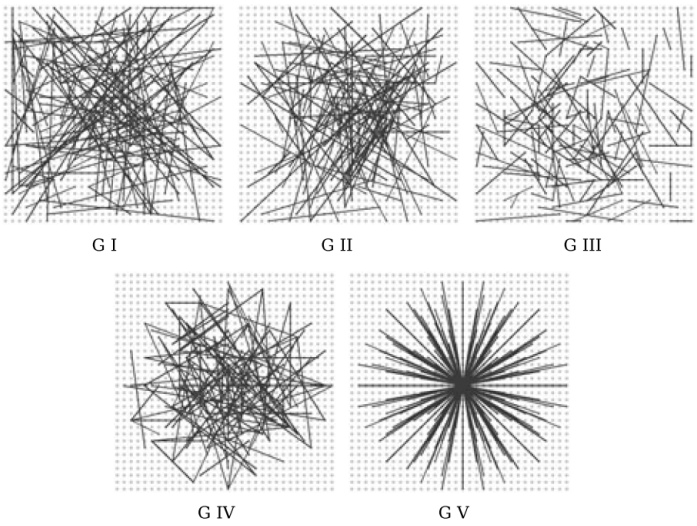

Figure 3.4: Five potential and different sampling geometric approaches for choosing the test locations, where G II is used in BRIEF. G I to G IV are selected by random sampling. Picture origin [17, p.782].

More precisely, the end points of a line in GII determines the locations for x and y. The result of pixel intensity comparison is always either 1 or 0 depending on the first

pixel's intensity, which can be higher or lower than the other pixel. In mathematical form this can be expressed as

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

where τ is a binary test, I is an image patch, x and y are the location of the pairs and $I(x)$ and $I(y)$ are the pixel's intensities. The comparison is done for all the pairs in the smoothed image until the bit string vector is constructed. [17, p.778-792]

Because the BRIEF's performance in rotation is poor, the ORB developers steered the features according to the orientation of the keypoints. This is done by first defining the $2 \times n$ matrix

$$S = \begin{pmatrix} x_1, ..., x_n \\ y_1, ..., y_n \end{pmatrix}$$

where S is a feature set of n binary test at location (x_i, y_i). The steered version is constructed with the rotation matrix R_θ, which has been formed with the patch orientation θ. From the equation

$$S_\theta = R_\theta S$$

the steered version of the features are constructed, where S_θ is the steered version of S, which has been calculated with rotation matrix R_θ. The angle is discretized to increments of $2\pi/30$, which is 12 degrees. This is constructed as a lookup table of precomputed BRIEF patterns. [29]

An important property of BRIEF's is that every bit feature has a large variance and a mean near 0.5. This information is lost and distributed once S_θ is created (oriented along keypoint direction). Because the variance is high it makes a feature more discriminative. Another method, which impacts the results, is to get the tests uncorrelated. There is a method how to get rid of all of these issues. To do this ORB makes an avid search for all the binary tests to find the bit features with a mean near 0.5 and a high variance as well as being uncorrelated. These methods will create the so called $rBRIEF$ descriptors. [37] [29]

4. MATCHING AND TRANSFORMING

When the keypoints and feature descriptors are calculated from at least two images the matching can be done. Basically this means that the features are compared to each other in order to find the features that are most similar. There are a few ways to do this.

Brute force matcher is maybe the most simplest matcher. It will take the first feature, compare it to the other image's features and it will select that feature, which has the closest distance compared to the first feature [2, p.416]. All the other features in the first image are going through the same pattern until the match vector is formed. In brute force, there are two ways to calculate the distance: the Hamming distance and the well known Euclidean distance. Other methods exist but only these two are explained in the following sections.

Usually there is an option to choose how many k matches is returned to the match vector. This option can be used if the ratio test is to be applied. Ratio test is also used in SIFT-algorithm where incorrect matches are filtered [22, p.19-20]. In the simplest form ratio test is done by selecting two best matches between a feature from the first image and from the second. The distance of the second best match is multiplied with the given value (*for example 0.8*) and it is compared to the distance of the best match. If the distance is lower after the multiplication it will be selected. Otherwise neither one is selected as a matched feature. This method will reduce unwanted matches and make them more unique. Practically two background features would be filtered out due to their similarity, which will increase validity of the match.

4.1 Hamming Distance

Hamming distance is well known for comparing two strings and it is used in ORB for matching. It is a distance between two strings or words of the same length. Hamming distance is the same as word differences in symbol places. [8] Basically, the smaller the distance, the more similar the two compared words are. For CPU the Hamming distance calculation means comparing two binary codes, which makes it efficient [13]. Furthermore, ORB features are stored as a bit string [29], which makes the use of Hamming distance in matching powerful and reasonable to use.

Hamming distance can be defined with the following equation

$$d_H^n(X,Y) = \sum_{i=1}^{n}(x_i \oplus y_i), \tag{4.1}$$

where d_H is Hamming distance, X and Y are the binary strings, i indicates bit position in bit strings and $\oplus$ is a XOR operator. Lets define the bit strings as

$$X = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

and

$$Y = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Now Hamming distance between X and Y can be calculated using the equation 4.1. The following table represents the results,

	bit
X	1 1 0 1 0
Y	0 1 1 1 1
$d_H(X,Y)$	1 0 1 0 1

Table 4.1: Hamming distance between two bit strings, distance for d_H is 3.

where the actual Hamming distance for d_H is 3.

4.2 Norms

A norm can be thought as the total size or length of the vectors in matrices. Simply put, a norm of the $\mathbf{x}$ can be marked as $||\mathbf{x}||$. L2-norm is maybe the most used norm and its equation can be written as $||\mathbf{x}||_2 = \left(\Sigma_i |\mathbf{x}_i|^2\right)^{1/2}$, which is also known as $2-norm$. Euclidean distance is maybe the most used $2-norm$. It can be used to calculate a distance between two vectors. When the Euclidean norm is applied to a feature vector, the Euclidean distance is calculated from the following equation

$$||\mathbf{x}_1 - \mathbf{x}_2||_2 = \sqrt{\Sigma_i (\mathbf{x}_{1_i} - \mathbf{x}_{2_i})^2}, \tag{4.2}$$

between two feature vectors where x_1 and x_2 are the feature vectors. [36] [19]

A norm worth mentioning is the p-norm with the following equation

$$||\mathbf{x}||_p = \left(\Sigma_i |\mathbf{x}_i|^p\right)^{1/p},$$

where p is a real number ($p \geq 1$) [21]. If we set $p = 2$ we get the same equation as in $2-norm$. Other norms like L0-norm and L1-norm exist but are not used in this thesis.

4.3 Homography and Transformation Matrix

In computer vision homography describes the relation and similarity between two images. It describes where the first image's points are located on the second image's coordinate system [11]. Camera calibration [16], aligning images and image stitching are practical and known applications where homography is used.

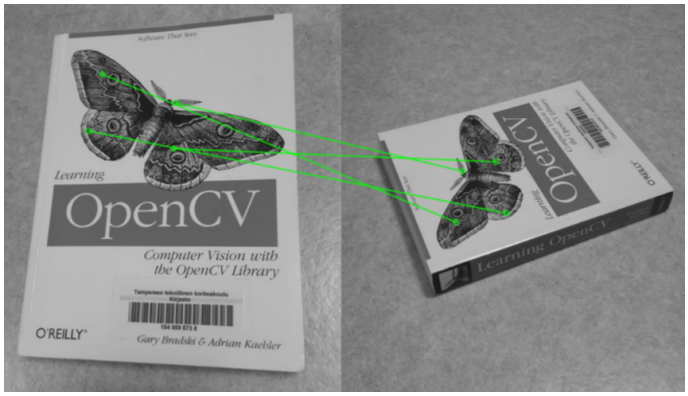

Figure 4.1: 4 homography coordinate points between two images after matching.

Figure 4.1 shows where the first image's (left) four points are correspondingly located on the second image (right). With this similarity information the homography can be calculated. Practically, the location of the matched feature pair is used to find the homography. With the aid of homography, transformation from any source point can be done to find the equivalent point in the target.

If there are many geometric transformations performed, it would be convenient to perform the transformation by matrix multiplication [6, p.178]. This can be done with the homography, which is represented as homogeneous coordinates as 3x3 matrix [30, p.332] and it can be calculated from the equation

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{4.3}$$

or in other words

$$\begin{pmatrix} wx' \\ wy' \\ w \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.4}$$

where $\mathbf{H}$ is a homography matrix (or transformation matrix), $\mathbf{p}$ is 2D vector and $\mathbf{p}'$ is a transformed vector. 2D images can be described as an image planar or, in this context, with a projective geometry. The term w is an extra dimension in addition to the x and y and here it describes the distance from the image plane to the camera so it can be considered as a scaling factor [9]. Correspondingly, coordinate 1 is added to the end of the term $\mathbf{p}$ to handle and avoid scaling in translation [30, p.332].

However, w will not be equal to 1 after the matrix multiplication and therefore it has to be mapped back to the real plane. This is also known as a homogeneous divide or normalization and it is done by dividing each component by w [33].

Transformation matrix $\mathbf{H}$ has the relation information between two images and it can be used to transform any point from the source image and get the equivalent point from the destination image. Lets consider an example with Figure 4.1 where we would like to know where the butterfly's head is located on the right image. To do this the head's source points in x- and y-directions and the transformation matrix $\mathbf{H}$ is needed. The transformation operator is straightforward with the equation

$$\frac{\mathbf{Hp}}{\mathbf{p}'[3]} = \mathbf{p}', \tag{4.5}$$

which has been concluded from equation 4.4. It includes also a normalization or a homogeneous divide as previously described. In equation 4.5 $\mathbf{p}$ is the source's point, $\mathbf{p}'$ is the destination's point and $\mathbf{p}'[3]$ is the last element of the destination point, which is also known as the term w in equation 4.4.

Lets consider an example with the following transformation matrix $\mathbf{H}$

$$\begin{pmatrix} -0.828 & 0.045 & 1851 \\ -0.609 & 0.113 & 1241 \\ -0.004 & 0.003 & 1.000 \end{pmatrix}$$

and the image coordinates

$$(1000, 1000),$$

which should be transformed with the matrix $\mathbf{H}$. To do this, coordinates must be represented as a matrix to make a matrix multiplication. Using the equation 4.4, the transformation

$$\begin{pmatrix} -0.828 & 0.045 & 1851 \\ -0.609 & 0.013 & 1241 \\ -0.003 & 0.003 & 1.021 \end{pmatrix} \begin{pmatrix} 1000 \\ 1000 \\ 1 \end{pmatrix} = \begin{pmatrix} 1068 \\ 645 \\ 1.021 \end{pmatrix}$$

is solved but doesn't have the right coordinates yet because they are represented in the wrong plane. To attain the right coordinates each component is divided with w, which in this case is 1.021. After the division, the final matrix

$$\begin{pmatrix} 1046 \\ 631.7 \\ 1.000 \end{pmatrix}$$

is normalized. The destination image coordinates are 1046 for x and 631.7 for y.

4.4 RANSAC

It's important to determine if the matched feature from the second image is an outlier or an inlier. In this case outliers are the incorrectly matched points. If outlier points are used in matching, they reduce the results dramatically. Figure 4.2 represents a situation where red lines are incorrectly matched points or outliers and green lines are correctly matched also known as inliers.

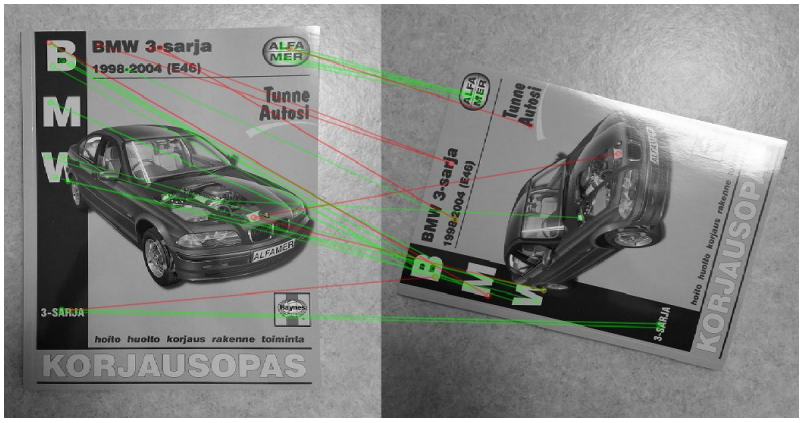

Figure 4.2: RANSAC: green lines represent inlier points and red lines outliers.

RANSAC (Random sample consensus) is one of the outlier detectors. The method is pretty simple. RANSAC will select two random points, it draws a line between the points and counts other points that are collinear [12]. The user can determine how long this loop lasts. RANSAC algorithm can be divided into the following steps:

1. Randomly select two points in the data set
2. Fit a model/line to the selected two points (draw a line between the two points)
3. Count the number of inliers with a given threshold value
4. Repeat steps 1-3 for predetermined number of iterations and select a model with the largest amount of inliers

The line, which has the most collinear points is selected and its points are treated as inliers [7]. This is represented in Figure 4.3.

Usually the user has the ability to choose how wide the collinear area is where the inlier points are taken from. This is known as reprojection error, which is the distance between the actual feature's location found during the matching process and the feature location predicted by the model [3]. Distance is calculated with the following equation

$$R = \sum_j \sqrt{(\tilde{x}_1^j - x_1^j)^2 + (\tilde{x}_2^j - x_2^j)^2}, \tag{4.6}$$

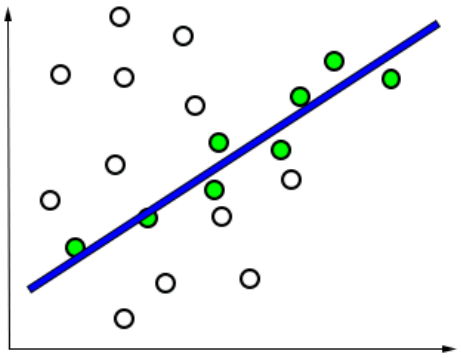

Figure 4.3: Green points represent inliers and the white outliers. Blue line represents a RANSAC model.

where $\tilde{x}^j$ is the projected location of feature j and x^j is the detected location of feature j, which is the same as the keypoint's calculated location. The reader should keep in mind that RANSAC will only estimate, which keypoint is an outlier and which an inlier. That being said, RANSAC will make the matching more robust and will remove the incorrect matches.

5. IMPLEMENTATION

In this chapter the actual implementation process is explained. It describes, which environment, which method and what test materials were used and how the comparisons were made. The three feature algorithms SIFT, SURF and ORB were compared. By selecting a moving object from a test video the algorithms were compared on the ability to track said object. The algorithms tracking accuracy, computational time and suitability for the particular application were considered. Real-time application suitability was also taken into account. Test videos were selected keeping in mind that the following parts were covered in testing perspective: scale changes (zooming), still background, viewpoint changes and video resolution and quality.

The four videos used were found from the Visual Tracking website [38]. The source from which the videos were downloaded was originally split into video sequences where every frame in the video is presented as a JPEG-image file. This was favorable in terms of testing as it would have been done anyway. In this manner the frames don't have to be extracted from the video file because the frames are already extracted to JPEG-files. This makes it easier to calculate the keypoints and feature vectors and also to reduce the computation time.

The ground truth values for every sequence were provided within the same packet. Ground truth values in this case reveal the object's actual location in every sequence. Location information was given in the form of a box around the object. The box's starting coordinates in x- and y-directions and its width and height were provided. Using the information the actual dimension for the box can be calculated easily. The box should only enclose the object in every frame/sequence.

Figure 5.1: Bounding box around the moving object. Picture origin [38] with modifications.

Figure 5.1 represents a drawn box around the moving object where the box's coordinates were taken from the ground truth values. Using the ground truth limits the tracking and comparison only for the preselected object because the ground truth values are only given to that object. Because only a single object was intended to be tracked, and was selected reasonably, the limiting factor didn't cause any disadvantage.

5.1 Implementation Phases

The main phases of implementation and the most important OpenCV functions are represented in Figure 5.2. In this section phases and practical solutions are explained in more detail. Python's OpenCV functions are explained in section 5.3.

First step in the implementation was to crop the object from the first frame of the source image using the ground truth values. Cropping was done because the source's keypoints were preferably taken from the object itself and not from the surrounding background. When the keypoints and the information about the scale space and orientation are extracted, they are assigned to a feature vector. The process is known as feature detection and description, and it's done with the OpenCV's function `detectAndCompute` (step 1 in Figure 5.2). Source features create a base for tracking and are also used in every matching iteration operation per video sequence. Therefore, source feature vector is constant after its creation.

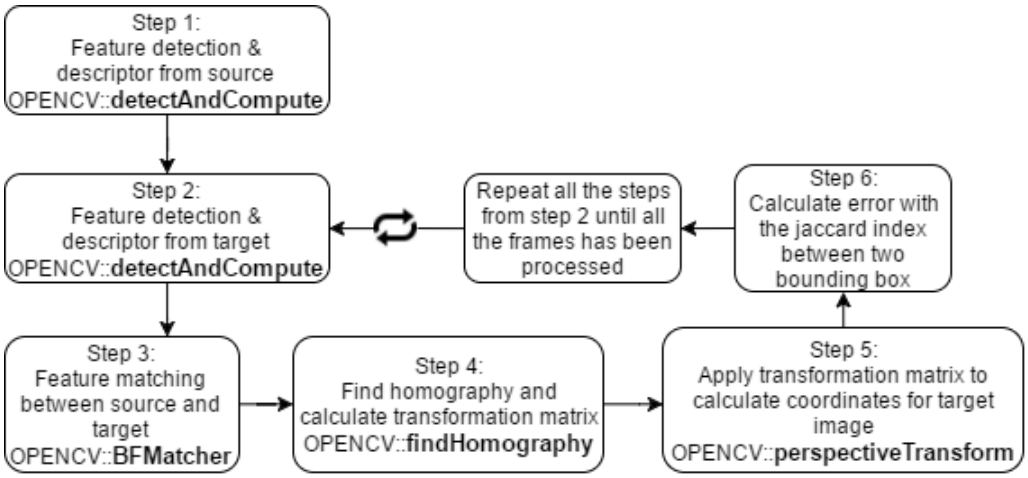

Figure 5.2: Block diagram of the tracking process with the most important OpenCV-functions.

For the following frames, the process (step 2 in Figure 5.2) was similar to that described in step 1. The only difference was that the cropping phase was not used. In real situations there is no information of the objects area of location. To compare and benchmark the feature algorithm results, cropping phase was excluded. With this method, better results for the comparison are achieved.

When the features are detected, described and stored in the feature vector, they can be matched with the function `BFMatcher` (step 3 in Figure 5.2). Matching was done with the brute force method based on either Euclidean distance or Hamming distance depending on which algorithm was used. K-Nearest Neighbors algorithm (k-NN) was used to store the two best matches. Only one or neither match was selected with the following equation

$$(m, n) := \begin{cases} m & \text{if } d(m) < 0.8 * d(n) \\ Null & \text{otherwise} \end{cases}$$

where m and n are the two best matches for one keypoint and d is the distance of the matched keypoint. Threshold value of 0.8, which was also used by Lowe with SIFT algorithm, was used in the matching phase [22] for each algorithm. With this value, all matches with a greater distance ratio than 0.8 are rejected. This method discards only less than 5% of the correct matches but will eliminate as much as 90% of the false matches [22], which improves the final results.

The next important phase is to calculate the homography from the filtered matches between source and target images (step 4 in Figure 5.2) to get the transformation matrix. Before calculating the homography, the source keypoints location has to be modified. Because the source keypoints were calculated from a cropped image,

the location information was false with respect to the coordinates where the object is actually located in the source frame. Therefore, the source keypoint coordinates needed to be modified by adding the ground truth's top left coordinates to the source keypoints coordinates. Not modifying the keypoints would have impacted the results negatively. Once the coordinates were modified, the transformation matrix was calculated with the function `findHomography`. Within this process the RANSAC was applied to filter the outliers with a threshold value of 3, which is known as a reprojection error threshold value. Two variables were returned as output, one was the actual transformation matrix (in 3×3) and the other a mask. The mask tells if a match is an inlier or an outlier. It is basically a vector with values of either 1 or 0, 1 denoting an inlier and 0 an outlier.

Finally, the 3×3 transformation matrix was applied to the source object's ground truth values with the function `perspectiveTransform` to get the target object's coordinates (step 5 in Figure 5.2). From the output coordinates only the upper left and lower right were used to draw a bounding box (rectangle) in the target image.

5.2 Jaccard Index

Jaccard index was used for calculating the amount of tracking accuracy. It defines how much of the target images rectangles were drawn incorrectly compared to the ground truth values, which were provided within the same packet as the video (step 6 in Figure 5.2). Jaccard index defines the ratio of the intersection size between two sets and the size of their union [4]. Zero "0" means that both sets are not connected at all so they don't have a common union. One "1" means that the two sets are completely aligned with one another and are fully united. Figure 5.3 represents a situation where the left (black) and right (blue) rectangle's intersection is marked with the red stripes.

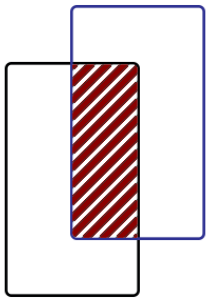

Figure 5.3: Intersection of two rectangles A and B ($A \cap B$) marked as red stripes.

Jaccard index can be defined with the following equation

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.1}$$

where J is the jaccard index, $\cap$ is the intersection and $\cup$ is the union. If $J = 1$, rectangles are fully aligned. In terms of tracking accuracy $J = 1$ means that tracking accuracy is 100%, when $J = 0$ means accuracy of 0%. The tracking error was simply calculated with the equation $E = 1 - J$, where $0 \leq E \leq 1$. Basically, the difference will tell how much error there is between the sets A and B. Every frame's Jaccard index values were stored into their own array.

All the implementation steps from 2 to 6 from Figure 5.2 are repeated until all the video frames were processed. Following section will introduce the earlier mentioned OpenCV functions as well as the testing environment and feature algorithms constructors.

5.3 Software implementation

The framework for object tracking was implemented using the programming language Python (release 2.7) and the OpenCV library (release 3.1.0). Continuum Analytics offered Python packages including an editor called Spyder. The OpenCV was downloaded separately. OpenCV is an abbreviation from the Open Source Computer Vision and it is developed originally by Intel in Russia 1999. It is written in C/C++, which also enables it to be used in real-time applications. Python and OpenCV-library offer a powerful tool for testing any kind of computer vision applications especially in object tracking.

The very first step in software implementation was to create a feature constructor. Every three feature classes were created with their own constructions. While building a feature class, the possibility to adjust the parameters provided the option to alter the behavior of the feature class in feature detection, description and matching. These parameters had a huge impact on the found keypoints, matching results and finally on the amount of accuracy of the bounding box. Following tables presents parameters for every three feature algorithm.

	Parameter	Default value	Description
SIFT	**nfeatures**	0	Amount of best features to retain. Zero "0" means that all the features are retained.
	nOctaveLayers	3	Layers used in each octave. 3 is the value used in the paper by Lowe [22].
	contrastThreshold	0.04	Contrast threshold value used to filter out weak features in low-contrast regions. Larger threshold value produces less features by the detector.
	edgeThreshold	10	Threshold value, which is used to filter features located on edges. The larger the edgeThreshold, the less features are filtered out.
	sigma	1.6	Value of the Gaussian's sigma, which is applied to the input image at the octave #0.

Table 5.1: OpenCV SIFT constructor class.

The SIFT constructor's parameters are all in an important role. The `ContrastThreshold` had maybe the biggest impact on the amount of found keypoints. It is used to filter insignificant features in low-contrast regions. The larger the threshold value is, the more features are filtered out. The `sigma` value determines how much the Gaussian images are smoothed. This parameter was hard to choose due to its behavior. Too low and too high values reduced the amount of found keypoints enormously so work had to be done to find the optimal value. The `nOctaveLayers` determines how many layers are used in each octave in scale pyramid creation. The more layers used, the more found keypoints there were. Although at a certain point, increasing the parameter value no longer increased the number of keypoints. [37]

	Parameter	Default value	Description
SURF	**hessianThreshold**	100	Threshold value for Hessian keypoint detector.
	nOctaves	4	Number of pyramid octaves used by the keypoint detector.
	nOctaveLayers	3	Number of octave layers in each octave.
	extended	FALSE	Extended flag for descriptor. The value "false" will use 64-element descriptors, value "true" will use extended 128-element descriptors.
	upright	FALSE	Up-right or rotated flag for features. The value "true" will not compute orientation of features, value "false" will compute orientation.

Table 5.2: OpenCV SURF constructor class.

SURF had the least parameters to adjust the behavior of the detector and descriptor. Parameter `hessianThreshold` had the biggest impact on the found keypoints but also to the amount of computational time. Hessian is calculated from the derivatives of the image intensities around the pixel using the box filters, which approximates the Gaussian derivatives. Parameter `hessianThreshold` adjusts, which point is considered as a keypoint. Parameter `nOctaveLayers` determines how many layers are used in every octave as in SIFT. Parameters `extended` and `upright` should have an impact on feature matching. [37]

	Parameter	Default value	Description
ORB	**nfeatures**	500	Maximum amount of features to retain.
	scaleFactor	1.2	Decimation ratio of image pyramid.
	nlevels	8	Amount of image pyramid levels.
	edgeThreshold	31	The border-size in which the features are not detected.
	firstLevel	0	In this implementation this value should be "0".
	WTA_K	2	Amount of points that produce the elements of the ORB-descriptor.
	scoreType	HARRIS_SCORE	Default value is "HARRIS_SCORE", which means that feature ranking is based on the Harris-algorithm.
	patchSize	31	The patch-size used by ORB.
	fastThreshold	20	Threshold value for FAST comparing the candidate pixel intensity with other surrounding 16 pixels.

Table 5.3: OpenCV ORB constructor class.

Maybe the most influential parameters of the ORB class constructor are `scaleFactor`, `nLevels` and `fastThreshold`. The `scaleFactor` defines the decimation rate for the scale pyramids. If the `scaleFactor` = 2, each level has $4x$ less pixels compared to the previous level. A high `scaleFactor` will reduce the number of found keypoints and the feature matching results. A low `scaleFactor` will increase computation time. The second parameter `nLevels` has a similar kind of behavior as `scaleFactor`. It will determine how many pyramid layers are created; the more layers created, the better are the results but will increase computation time. Therefore, an optimal value had to be found. The parameter `fastThreshold` determines the threshold value for FAST-algorithm when comparing the candidate pixel intensity with other 16 surrounding pixels. Threshold value itself affects how the candidate pixels are chosen and therefore it will affect how many keypoints are found. [37]

Function		Used parameters	Description
detectAndCompute	input	image	Detects keypoints and computes the descriptors from the given image. Function, which is in each feature class.
	output	kp, des	Found keypoints "kp" and computed descriptors "des".
BFMatcher	input	normType	Constructor for the Brute-Force matcher. "normType" defines, which norm is used.
	output	bfMatcher	BFMatcher class instance.
findHomography	input	srcPoints, dstPoints, method, ransacReprojThreshold,maxIters	Finds a perspective transformation and homography matrix between two set of keypoint coordinates. "srcPoints" and "dstPoints" are the keypoint coordinates. "method" is a method used to compute homography matrix. "ransacReprojThreshold" determines the threshold value for RANSAC. "maxIters" determines the maximum number of RANSAC iterations.
	output	H, mask	H is a homography matrix and mask tells, which keypoints are inliers and outliers.
perspectiveTransform	input	src, H	Performs the perspective matrix transformation with the transformation matrix "H" and source vector of points "src".
	output	dst	Destination points after the transformation.

Table 5.4: Other important functions in OpenCV.

After the feature class is created with the constructor, one function of the class `detectAndCompute` can be used to detect and describe the keypoints. Constructor's parameters have a huge impact on how the keypoints are detected and computed as descriptors. The `BFMatcher` is a Brute-Force matcher, which is used to match the features with each other. Parameter `normType` defines, which norm is used in distance calculations. Possible values are `NORM_L1`, `NORM_L2`, `NORM_HAMMING` and `NORM_HAMMING2` where only `NORM_L2` is used in this thesis. `findHomography` is used to calculate the transformation matrix between the two given sets of keypoint coordinates. To get results the minimum keypoint coordinates of 4 have to be used. A parameter `method` describes, which method is used to filter out the

inliers in computing the homography matrix. Possible values are `0` (which uses all the keypoints), `RANSAC`(RANSAC-based robust method), `LMEDS`(Least-Median robust method) and `RHO`(PROSAC-based robust method), where only the `RANSAC` is used. The `perspectiveTransform` is used to find the destination coordinates with the transformation matrix and the given source coordinates.

6. EXPERIMENTS, RESULTS AND EVALUATIONS

In this chapter the results for all three feature algorithm SIFT, SURF and ORB are presented in terms of tracking accuracy and its validity, found keypoints and computational performance. Results were executed with computer's assembly of Intel's quad core i7-6700K processor and 16.0 GB of DDR4 RAM. The reader should keep in mind that the results are based only for the test cases and videos that are used in this thesis and might differ if the test videos are chosen differently. However, the purpose was to choose different kind of videos to get general and comprehensive results.

One challenge in the making of this theses was to get acceptable results. The `cv2.perspectiveTransform`-function (with the transformation matrix and original coordinates) sometimes placed the target's rectangle in the wrong location. The issue was addressed by trying to calculate the rectangle's coordinates manually using equation 4.5 with the transformation matrix instead of using `cv2.perspectiveTransform`-function. However, the results remained unchanged. Another attempt was to try to find bugs in the code, especially in matching and homography. Again, there was no effect on the results.

The reason behind the wrong coordinates turned out to be the transformation matrix calculations in `cv2.findHomography`-function. The minimum requirement of four keypoints was barely satisfied and there was too much differentiation between the keypoint's locations in terms of homography. The problem occurred especially in the low resolution videos. This combination led to inaccurate results. One major improvement was found by increasing a video resolution with a scale factor. Every frame was scaled up twice before the keypoints were calculated. This led the feature detector to find more keypoints more accurately, which increased the tracking accuracy. Maybe an even more effective improvement was to configure the feature class parameters for each feature algorithm in the phase were parameters were optimized.

6.1 Default Parameters

First video is a scene of an concert where a singer dressed in white is meant to be tracked. The camera zoomed out while moving to the left with respect to the singer. At one point, lights from the stage are directly pointed towards the camera.

In other words, the video contains scale-, illumination- and viewpoint changes. From the video itself 160 frames were selected, which was enough coverage for the results. The video has a good resolution of 624 x 352 pixels and has rich and sharp details.

First test was done using the default parameters of feature classes. The scale factor was not used in this test case to resize the video sequences. The results are presented in Figure 6.1,

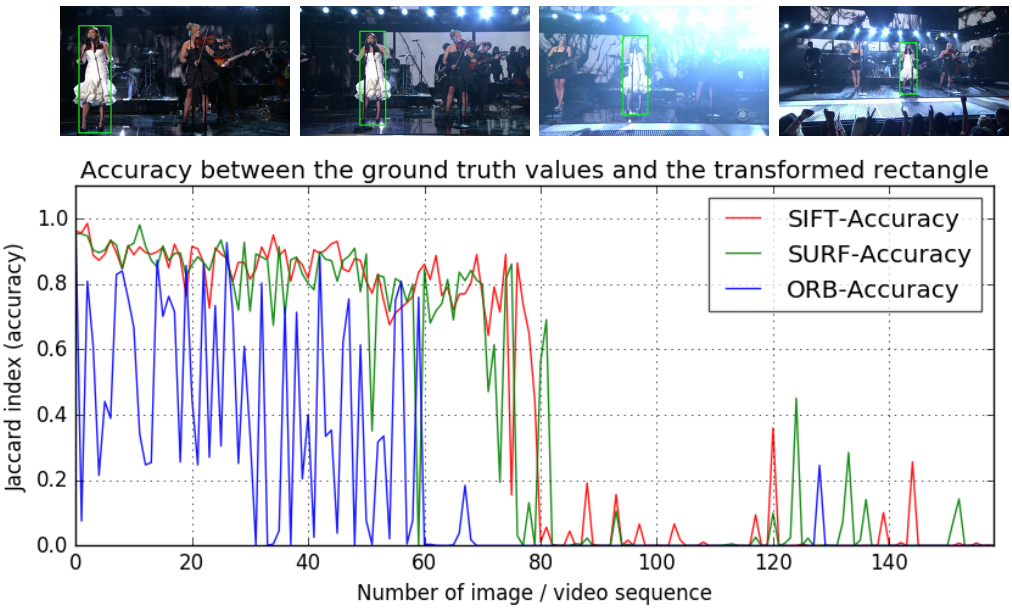

Figure 6.1: Results of singer-video, default parameters.

where the scene is demonstrated with four thumbnails of video sequences, which were taken between regular intervals; the first one is from first sequence, the last one is the last 160th sequence and the middle two thumbnails are from between the first and last sequence. The singer itself has been marked with a green rectangle in every thumbnail. The rectangle coordinates were taken from the ground truth values. The graph lines present the amount of accuracy during every sequence. The accuracy was calculated using the method described in section 5.2. The graph lines were drawn with different colors to stand out better.

The algorithms SIFT and SURF performed better than ORB with this setup as shown in Figure 6.1 and table 6.1. SIFT had an average accuracy values of 42.7% and SURF reached values of 40.7% when ORB had a low accuracy value of 17.3%. The low accuracy level might be due to the fact that the maximum number of features was limited to 500 in ORB as default. Furthermore, 70 of the 160 cases are from situations where the transformation matrix cannot be calculated due to the lack of used keypoints (the minimum keypoints of 4 was not satisfied), which further decreases the accuracy for ORB.

Both SIFT and SURF performed well and steadily (tracking accuracy around 80%) until the light shined directly towards the camera (around sequence 80-140). Under these circumstances the visible material changed dramatically and made tracking more inaccurate. Interestingly, the same effect continued even after the light turned from the camera (after sq. 140). This might be caused from large viewpoint and scale changes compared to the first sequence. One should keep in mind that all sequence keypoints are matched with the first sequence's keypoints, which were calculated only once .

	SIFT	**SURF**	**ORB**
Avg accuracy(%)	42.7%	40.7%	17.3%
Avg time(s)/seq	0.090	0.185	0.006
Total time(s)	17.784	34.714	1.209
Found kpoints from object(mainly)	274	185	114
Avg found kpoints/frame	1185.28	1331.86	499.70
Avg used points in H calculations	21.02	21.50	4.57
Cannot transform (cases/sequences)	2/160	0/160	70/160

Table 6.1: Results: singer, default parameters.

Regarding the computational time, ORB was clearly the fastest. It was mainly because the transformation matrix could not be calculated. SIFT had a computation time almost half of SURF's, which was interesting because SURF papers indicate that it should be faster than SIFT [1]. The total computation time consists of finding and describing keypoints from the object and from the following sequences plus the transformation matrix calculations, which are the steps from 2 to 6 from Figure 5.2. The average time per sequence was calculated only from the sequence's keypoint detection and description without the matching and transformation matrix calculations (step 2).

Average used points in H calculations indicate how many keypoints were used when the transformation matrix H was calculated if the execution was possible. These keypoints were left to be used in the matrix calculations after the matching and RANSAC. Table 6.1 also shows how many keypoints were found from the cropped source image in the beginning and the average found keypoints from the following sequences' entire frames.

In the second video, small bottles are placed on a table and their arrangements are changed. The camera's location remains still and there is no scaling (no zooming) within the 1741 frames. The video's resolution is 640 $\times$ 480 and the picture's quality moderate. Every bottle has its own unique label. Feature algorithm comparison was

conducted with default parameters when tracking the location of a single bottle.

The results of tracking accuracy can be seen in Figure 6.2. Once again results for ORB are poor with its default parameters. The exact accuracy for ORB was 0.0% as can be seen from table 6.2 meaning that the tracking error was 100%. In contrast, SIFT and SURF both performed almost flawlessly for the first 300 frames until the tracked bottle was moved. This scene is presented in the second thumbnail picture in Figure 6.2. During the transition, the bottle's label is blurred. The bottle is placed back on the table around the frame 450. The rapid peaks in frames 500, 730 and 770 are produced when other bottles are moved in front of the observed bottle.

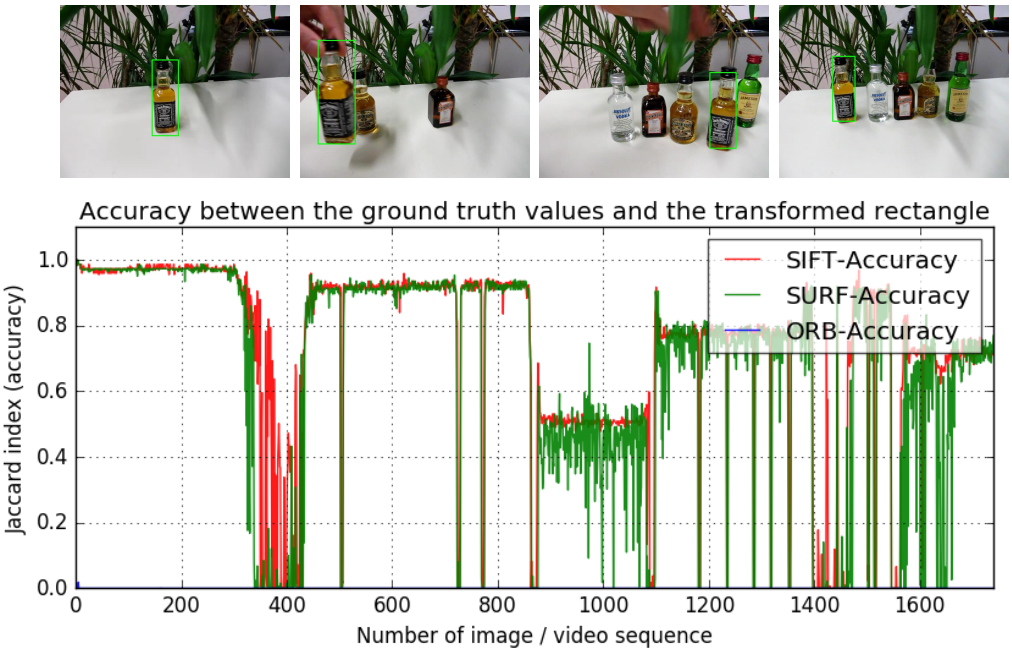

Figure 6.2: Results of liquor-video, default parameters.

The next interesting part is seen between the frames 900 and 1100 (thumbnail 3). In that period the bottle is rotated and placed in another location, which decreased the tracking accuracy almost by half. This is an interesting observation because the same visible information in evident but only the label shapes are in a different angle. Within this scene SIFT seems to perform better than SURF. The last scene worth mentioning is around frame 1420 where the bottle is rotated around its vertical axis. As one might guess, none of the feature algorithms could track the object because the label is not visible from where the features were calculated.

	SIFT	SURF	ORB
Avg accuracy(%)	72.8%	66.5%	0.0%
Avg time(s)/seq	0.126	0.251	0.008
Total time(s)	251.438	448.747	15.208
Found kpoints from object(mainly)	203	117	50
Avg found kpoints/frame	1579.27	1714.06	500.00
Avg used points in H calculations	55.25	28.07	4.00
Cannot transform (cases/sequences)	39/1741	117/1741	1692/1741

Table 6.2: Results: liquor, default parameters.

Table 6.2 presents more data of the comparison. SIFT performed best in terms of tracking accuracy, computational time, keypoints detected from the object and successful transformations. It is interesting to see that SIFT used the most location points in calculating the transformation matrix. This might not always be a good thing especially if some of the location points are classified as outliers.

The third video is a short capture of a scene in a settlement where two humans and a car are moving along the street. For it's simplicity, the video sample could also be from surveillance footage. The camera stays still throughout the video and there is no scaling. Video's resolution is 768×576 so one might suppose the quality to be adequate but that is not the case. The video does not have rich details and the overall quality is poor. Furthermore, the tracked human is relatively small, which complicates things even more.

Few tests were run for the original video but the results were poor and meaningless in terms of comparing the feature algorithm. Therefore, the video was scaled up twice the size using linear interpolation. With this transformation the video resolution increased to 1536×1152 and more details and keypoints could be extracted. This increased the overall results and gave more value for the comparison.

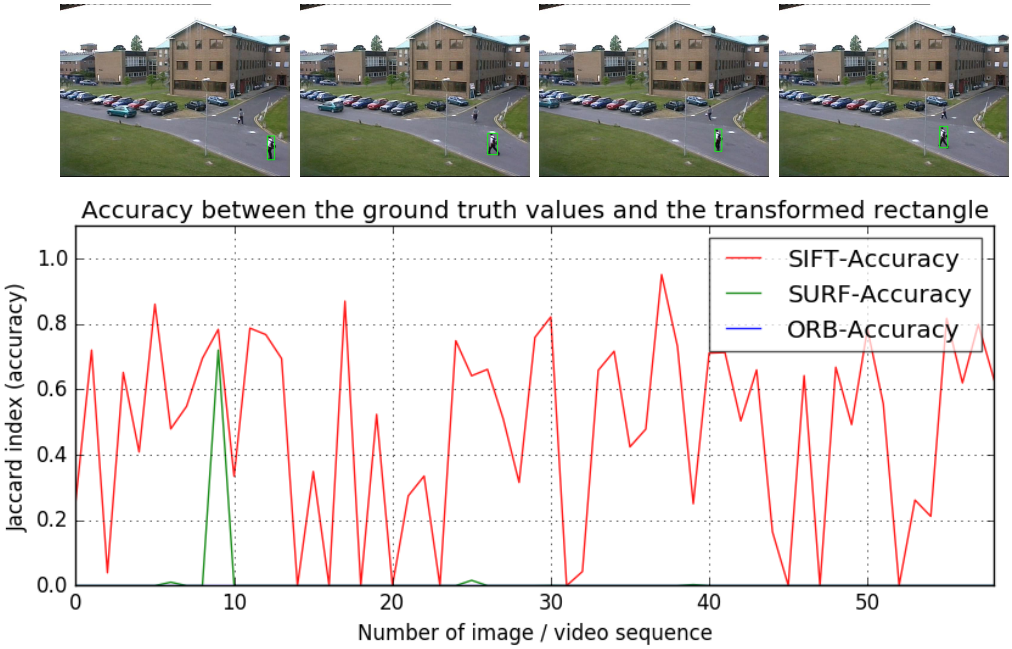

Figure 6.3: Results of walking-video, default parameters.

The third video's results are presented in Figure 6.3. As the graph shows, both SURF and ORB had difficulties tracking the object and SIFT's accuracy varied greatly within the 60 frames. The reason for the poor accuracy was the video quality, which meant that the details of keypoints were blurry and indistinct. Because some keypoints are also located in the humans legs and because their location doesn't remain the same relatively to the first frame, the transformation matrix gives false information periodically.

	SIFT	**SURF**	**ORB**
Avg accuracy(%)	48.0%	1.3%	0.0%
Avg time(s)/seq	0.512	1.363	0.035
Total time(s)	30.527	80.603	2.039
Found kpoints from object(mainly)	27	14	500
Avg found kpoints/frame	2944.22	6100.37	500.00
Avg used points in H calculations	6.79	4.00	N/A
Cannot transform (cases/sequences)	3/60	53/60	59/60

Table 6.3: Results: walking, default parameters.

Table 6.3 confirms the poor results for SURF and ORB also seen in Figure 6.3. ORB's tracking accuracy was 0.0% with default parameters and that of SURF's

was only 1.3% better. However, SIFT's tracking accuracy was 48.0%, which was fair taking into account the challenging video. SURF's average found keypoints of 6100 seems to be conflicted with the average accuracy of 1.3%. Having that many keypoints and because 14 were found from the object itself, one could imagine that the results would be better. However, SURF's keypoint detector might have selected keypoints mostly not located on the human or the matching algorithm filtered out too many keypoints. From the succeeded transformations (7/60) only one (frame 9) gave adequate results as Figure 6.3 presents. Once again, ORB was unusable with it's default parameters and therefore may not be adequate in applications with these kind of videos.

Fourth and the last video is recorded from a moving car by hand where the car driving in front is meant to be tracked. The video resolution is 640 × 480, it has 585 frames and no scaling factor is used to resize the video. Video contains scale- and viewpoint changes. Recording by hand caused shaking of the video and the object was motion blurred most of the time. It seems that optical image stabilization was not used in this video. This creates a huge challenge for the feature algorithms to track the car accurately but was also the reason for selecting this video. Also, the presence of other cars might disturb the tracking accuracy.

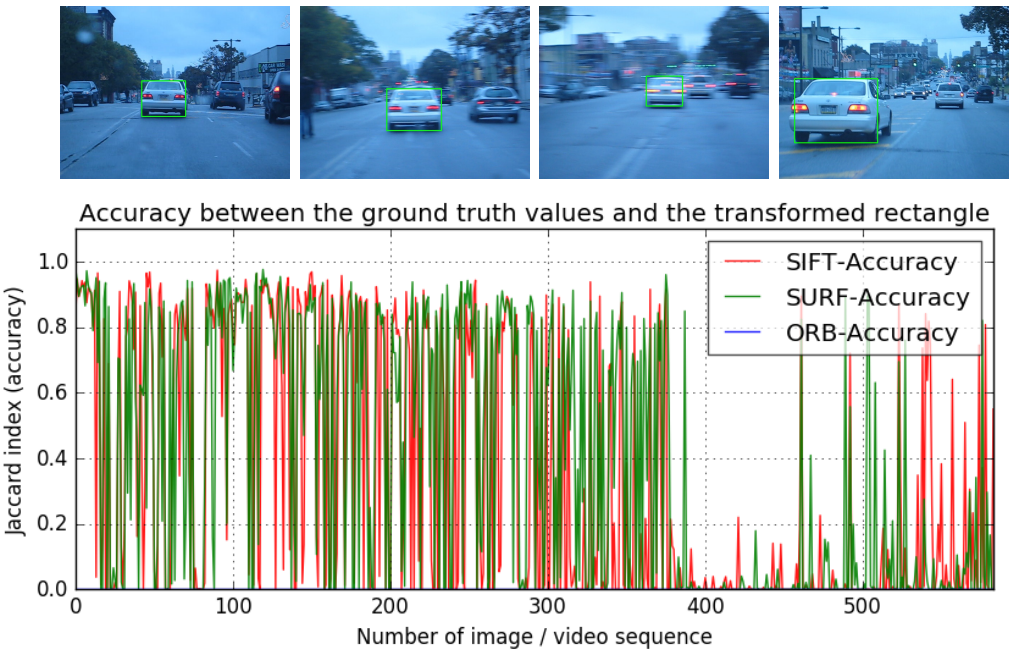

Figure 6.4: Results of motion blur car-video, default parameters.

Results for the fourth video can be seen in Figure 6.4. The tracking accuracy changed rapidly for SIFT and SURF. It was continuously varying between 0% and 80% and occasionally it reached values as high as 90%. Results for ORB were poor

(tracking accuracy 0%) with default parameters, which is also evident from table 6.4. The lowest results for all three feature algorithms were between the frames 390 to 460, from where the third thumbnail image in Figure 6.4 was taken from. Within those frames the camera moved more aggressively, which increased the motion blur in the video. Accuracy of the feature algorithms was not affected by the surrounding cars, which was undoubtedly due to the blurriness.

	SIFT	**SURF**	**ORB**
Avg accuracy(%)	33.2%	38.6%	0.0%
Avg time(s)/seq	0.102	0.203	0.007
Total time(s)	63.325	120.147	4.427
Found kpoints from object(mainly)	85	49	19
Avg found kpoints/frame	987.36	1186.74	499.61
Avg used points in H calculations	7.76	7.64	4.00
Cannot transform (cases/sequences)	86/585	107/585	566/585

Table 6.4: Results: motion blurred car, default parameters.

Interestingly, SURF got a higher tracking accuracy (38.6%) than SIFT (33.2%). This means that SURF was more robust to the motion blur than SIFT. SURF did not only get better tracking results but it also used less keypoints (found from the car) in matching. Furthermore, SURF had 21 cases more than SIFT where the transformation matrix could not be calculated. On that account, when SURF was able to calculate the car's location, it was more accurate than SIFT. On the other hand, SIFT performed unsurprisingly faster (63.3s) than SURF (120.1s) in terms of computation time. As previously mentioned, ORB got poor results (tracking accuracy 0%) and the location of the object could only be calculated for 19 frames.

6.2 Optimized Parameters

This section's purpose is to present the results with optimized parameters. Optimized in this case means that the feature algorithm constructor parameters are adjusted in a way that would improve the results for all the three feature algorithms. Other parameters remained untouched. The actual optimized parameters were chosen with the following logic. First all three feature algorithm parameters were adjusted for all the videos separately until the best results for tracking accuracy were found. In this way, every video had different parameter values for every three feature algorithm. From these parameters values the average values were calculated, which were then used for all the videos. Compromises had to be made when average

values were used for all three feature algorithms. The same four videos described in the previous section were used.

		Parameter	Value
Optimized parameters	SIFT	nfeatures	1500
		nOctaveLayers	3
		contrastThreshold	0.04
		edgeThreshold	9.5
		sigma	1.275
	SURF	hessianThreshold	145
		nOctaves	4
		nOctaveLayers	3
		extended	False
		upright	False
	ORB	nfeatures	1500
		scaleFactor	1.2
		nlevels	3
		edgeThreshold	5
		firstLevel	0
		WTA_K	2
		scoreType	FAST
		patchSize	20
		fastThreshold	20

Table 6.5: Optimized parameters for all three feature algorithms.

Table 6.5 represents the average parameter values used. While choosing the parameters, the following aspects were taken into consideration: honesty and validity in terms of comparison. This means that parameters, which had the same function for all the features, were set as equal. For example one parameter was the number of octave layers (in image pyramid), which was set to 3. SURF's Hessian threshold determined how many keypoints were found. Average found keypoints for SURF was around 1500 and so the $nfeatures$ for SIFT and ORB were set to 1500. As such, no more than 1500 keypoints were retained. Because there wasn't a possibility to adjust the number of octaves (in image pyramid) in SIFT and ORB, SURF's corresponding parameter was set to 4. Another criteria for choosing parameters was that the computational time doesn't increase too much. An attempt to decrease the computational time especially for SURF was relevant for its high original values although it might decrease the tracking accuracy.

For the singer-video, one major difference between the default parameters and optimized parameters are the results for ORB. They improved significantly as Figure

6.5 and the table 6.6 present. Surprisingly, ORB's results were now comparable with SIFT and SURF and actually had better results in accuracy than SURF, which were 42.1% for ORB and 40.9% for SURF. Therefore, it would seem that ORB could now be used in some applications for tracking. SIFT got the best results (43.0%) in accuracy but overall, all three performed almost equally well in terms of tracking accuracy.

In terms of computational time ORB had the lowest values. In fact, ORB was 56.7% faster than SIFT and 72.3% faster than SURF. SURF's rapid drop around frame 60 in Figure 6.5 is caused by rapid background changes and the increase of backlight.

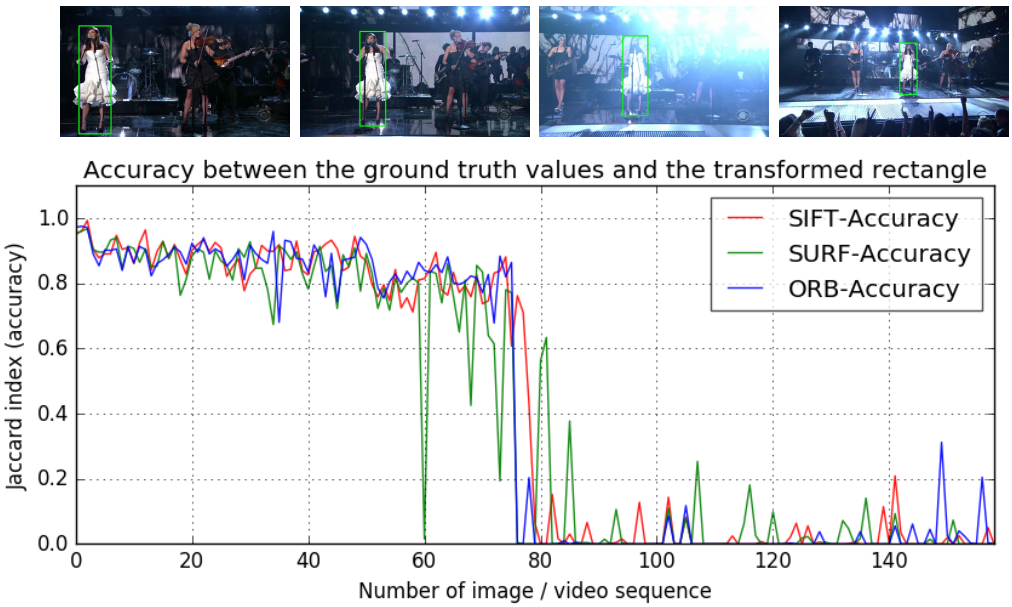

Figure 6.5: Results for singer-video, optimized parameters.

It is also worth mentioning that SIFT's tracking accuracy was improved by 0.3% but SURF's decreased by 2.3%. On the other hand, SURF's total computational time decreased by 6.1 seconds and SIFT's increased by 0.5 seconds. Furthermore, none of the transformations failed with ORB after optimization. With default parameters, transformation failed in 70 frames of the total 160 frames.

	SIFT	SURF	ORB
Avg accuracy(%)	43.0%	40.9%	42.1%
Avg time(s)/seq	0.081	0.171	0.006
Total time(s)	18.293	28.626	7.920
Found kpoints from object(mainly)	355	183	821
Avg found kpoints/frame	1441.66	1194.67	1499.60
Avg used points in H calculations	24.19	17.09	32.19
Cannot transform (cases/sequences)	0/160	5/160	0/160

Table 6.6: Results: singer, optimized parameters.

The second test video contained several bottles. As in the singer-video, ORB's results were improved significantly after the parameters were optimized. The tracking accuracy was 64.9% when it was 0.0% with the default parameters. The graphs for SIFT and ORB were now similar as seen from Figure 6.6. SURF's tracking problems between the frames 900 and 1100 with default parameters did not changed in the optimized version.

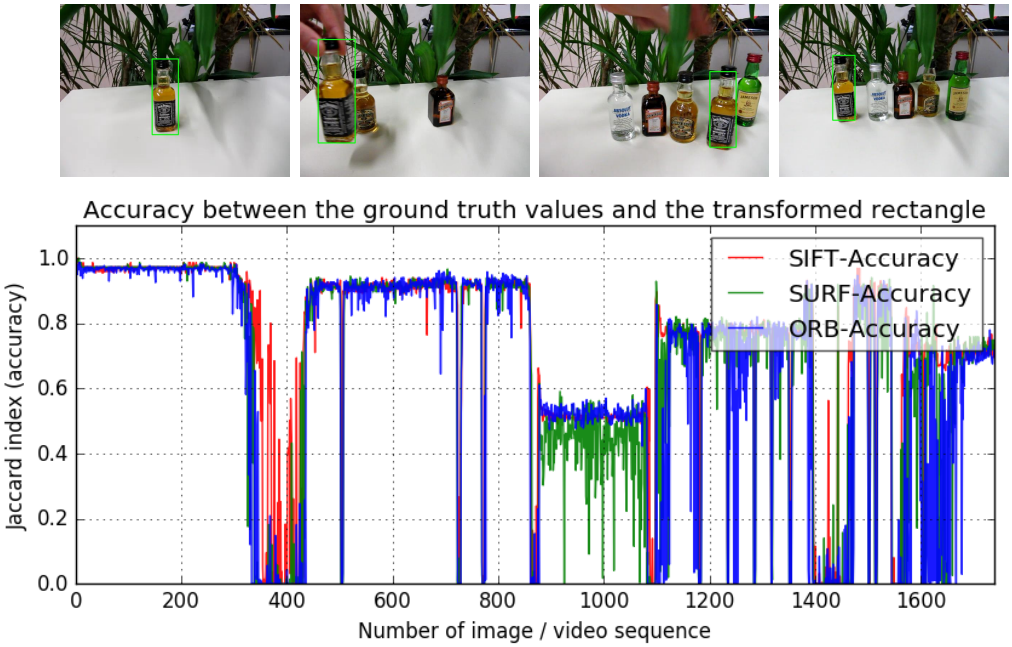

Figure 6.6: Results for liquor-video, optimized parameters.

As the table 6.7 represents, SIFT's tracking accuracy improved only by 0.5% but SURF's total computational time improved by 10.5 seconds. In terms of computational time, the best results were obtained with ORB with 55.8 seconds while SIFT

performed in 222.7 seconds and SURF in 438.3 seconds. None of the transformations failed in ORB and also SIFT improved its results from 39 to 8. Overall, SIFT had the highest values in tracking accuracy of 72.3%. Interestingly, it also used most of the average keypoints (62) in calculating the transformation matrix while SURF used 28 and ORB 42.

	SIFT	**SURF**	**ORB**
Avg accuracy(%)	72.3%	66.3%	64.9%
Avg time(s)/seq	0.104	0.246	0.007
Total time(s)	222.724	438.226	55.800
Found kpoints from object(mainly)	287	114	671
Avg found kpoints/frame	1499.52	1617.57	1500.00
Avg used points in H calculations	62.04	27.69	42.13
Cannot transform (cases/sequences)	8/1741	111/1741	0/1741

Table 6.7: Results: liquor, optimized parameters.

The third video was the street view in which the video's quality was low in terms of sharpness and details. This video was scaled up to 2 times the original size to get better results and valid comparing information. After the parameters were optimized, ORB's tracking accuracy improved by 45.8% and every transformation was successful as presented in table 6.8.

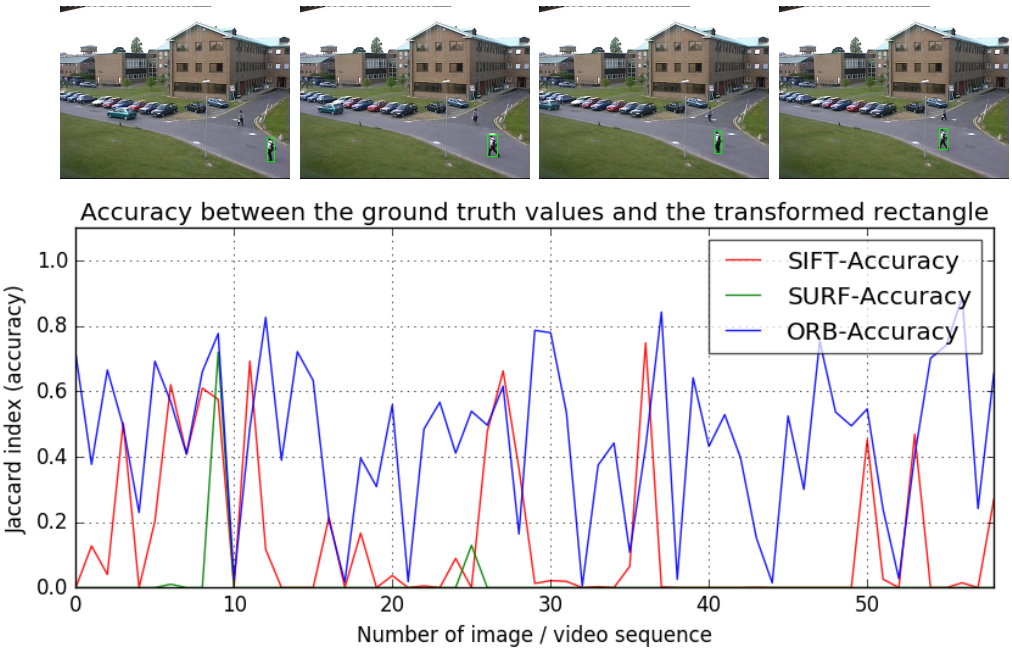

Figure 6.7: Results of walking-video, optimized parameters.

The average found keypoints on the object was 109 and the average used points in H calculations was 15, which is an impressive improvement for ORB. In fact, ORB performed best in every perspective compared to SIFT and SURF. It had 44.3% better accuracy in tracking than SURF and 32.2% better than SIFT. In terms of computational time, ORB had a total time of 1.6 seconds, which is 97.8% less than SURF and 92.8% less than SIFT.

	SIFT	**SURF**	**ORB**
Avg accuracy(%)	13.6%	1.5%	45.8%
Avg time(s)/seq	0.379	1.252	0.024
Total time(s)	22.460	73.978	1.614
Found kpoints from object(mainly)	22	14	109
Avg found kpoints/frame	1500.12	5264.46	1500.00
Avg used points in H calculations	4.71	4.00	15.07
Cannot transform (cases/sequences)	25/60	55/60	0/60

Table 6.8: Results: walking, optimized parameters.

Interestingly, SIFT got lower results in tracking accuracy with optimized parameters (13.6%) than with defaults (48.0%) and failed transformations increased by 22. Reason for this is unclear but it was hypothesized that because SIFT used only an average of 6.79 points for transformation matrix in default parameters, which is already close to the minimum value of 4. With the optimized parameters an average of 4.71 points were used and therefore it could not simply find enough points to calculate the transformation matrix. SURF's results remained relatively unchanged. That being said, it was noted that the total computational time reduced by 6.6 seconds.

The results for the fourth video can bee seen from Figure 6.8 and from table 6.9. All three algorithms improved their results in tracking accuracy; SIFT by 0.2%, SURF by 0.5% and ORB by 33.0% compared to the default parameters results as the table 6.9 presents. ORB's results are now comparable with it's tracking accuracy of 33.0%. SIFT's accuracy was 33.4% and SURF's 39.1%. SURF performed with the highest percentage with optimized parameters as it did with default parameters.

Figure 6.8 shows that the tracking results for SIFT and SURF between the frames 390 and 460 barely improved. Interestingly, ORB performed slightly better here, were the object had the most motion blur.

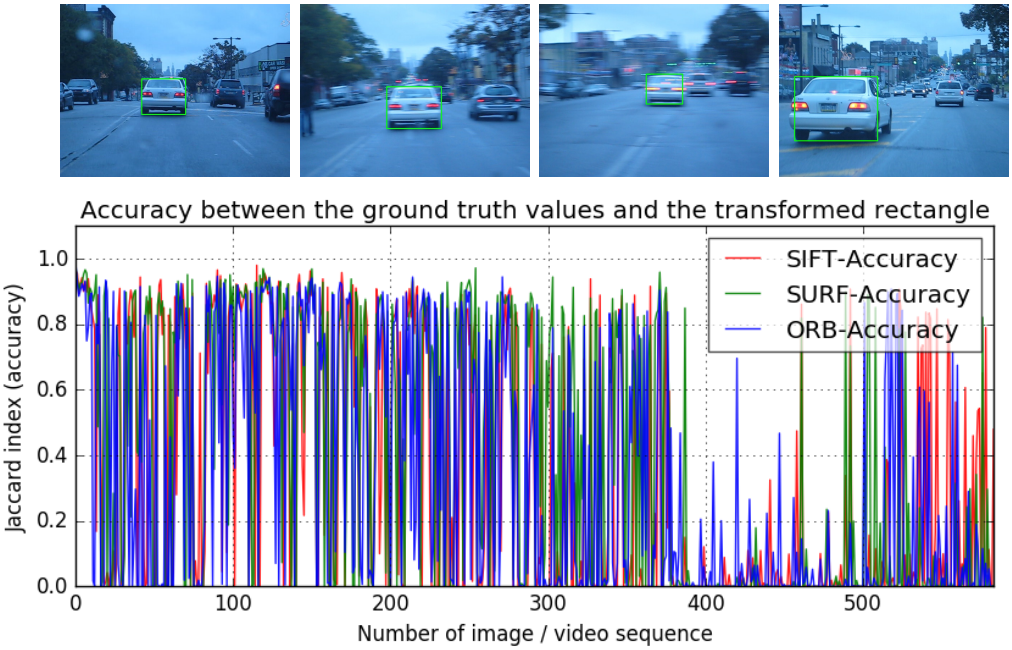

Figure 6.8: Results for motion blur car -video, optimized parameters.

SURF managed to improve the computation time by 7.0 seconds but the total time of 113.2 seconds did not compare with SIFT's time of 55.7 seconds, which also was improved by 7.6 seconds. As one might guess based on the previous results, ORB had the lowest computation time being only 7.4 seconds.

	SIFT	**SURF**	**ORB**
Avg accuracy(%)	33.4%	39.1%	33.0%
Avg time(s)/seq	0.087	0.192	0.006
Total time(s)	55.721	113.186	7.410
Found kpoints from object(mainly)	108	47	175
Avg found kpoints/frame	1056.38	1077.84	1413.61
Avg used points in H calculations	7.99	7.44	9.53
Cannot transform (cases/sequences)	65/585	104/585	16/585

Table 6.9: Results: motion blur car, optimized parameters.

An interesting observation was that ORB had the most found keypoints from the car and the frames, most used points in the transformation matrix calculation and the least number of failed transformations. Still it did not achieve a higher tracking accuracy. Usually, good key attribute values indicate better accuracy but was not so regarding this video.

7. CONCLUSIONS

All three feature algorithms proved that they can be used in tracking an object from a video as long as the video quality is good enough. This means that the video has to contain sharp enough details to get distinct features. If the quality is blurry, features have more similar contents, which makes it impossible to track an object accurately. As demonstrated by the results, blur had the most influence on the results. Surprisingly, SURF was more robust to the motion blur than SIFT and ORB in the fourth video. SURF can be a worthy option in cases where the video has been recorded by free hand without image stabilization.

The results would have been better for all three algorithms if the parameters would have been optimized separately for each four videos. In this thesis, the idea was to get optimized parameters for general use and that's why only one set of optimized parameters were used for the videos. One of the challenges in this thesis was to optimize parameters for SIFT and SURF. In general, the obtained benefit was minor and in some cases the tracking accuracy results were even worse than with default parameters. However, ORB's tracking accuracy improved significantly (around 43%), which was surprising considering what the results were with default parameters. Though, this came at a price.

ORB's computation time increased by 217% while still being 78% lower than its worst competitor SIFT. SURF on the other hand had the highest computation times. ORB computation time was around 88% faster than SURF. This was really interesting taking into account that in theory SURF should be faster than SIFT, which has also been stated in the article [1]. The parameters could not be adjusted properly, which influenced the results negatively. Though, the main reason seemed to be the reason behind the algorithm itself and how it was implemented in OpenCV.

Overall all three feature algorithms can be used in tracking applications but when it comes to real-time applications, ORB is the clear selection. Although, computer performance limits how it can be used, ORB's parameters will give a wide opportunity to adjust the computation time. SIFT and SURF may also be considered for in real-time applications but they pose high performance demands for the computer.

In this thesis, source keypoints were calculated only once for every case, which sets certain limits. One limit for the viewpoint changes is that the same side of

the object has to be visible also in the following frames. Otherwise, the object is impossible to track. Another factor that can decrease the results is the location that the keypoints are picked from. Although the object was cropped from the source image and the keypoints picked from this image, they may include those taken from the background around the object. One improvement in this thesis would have been to eliminate the features, which were not taken from the object.

To pre-process the images or frames before handing them over to the tracker, would also improve the results. The pre-processing would include image enhancements like deblurring, filtering the additional illumination, noise removal and image sharpening. For this thesis three feature algorithms were only meant to be compared and that's why the image enhancements were not used. In terms of future development, adding a function that enables the algorithm to adapt to changes would increase the algorithms tracking accuracy. Basically this mean that the algorithms would learn how the object looks like from the following frames and would adapt by calculating new source keypoints for better matching. Therefore, it might be called as a self-learning system allowing wider viewpoint changes, scaling and rotations.

In conclusion, SURF may not be as suitable in object tracking as the other two due to its computation performance. That's why it might fit better in applications such as recognizing an object and camera calibration. If one had to choose between ORB and SIFT, ORB would be selected for real-time applications with optimized parameters and SIFT would be used in general purpose applications but also for more accurate tracking.

REFERENCES

[1] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding, Vol. 110, Iss. 3, 2008, pp. 346-359.

[2] G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. McMahan, J. Jerald, H. Zhang, S.M. Drucker, C. Kambhamettu, M. E. Choubassi, Z. Deng, M. Carlson, Advances in Visual Computing: 10th International Symposium, ISVC 2014, Las Vegas, NV, USA, December 8-10, 2014, Proceedings, Part I, Springer International Publishing, Cham, 2014.

[3] G. Bebis, B. Parvin, Y. Kuno, R. Pajarola, A. Hinkenjann, C.T. Silva, Advances in Visual Computing: 5th International Symposium, ISVC 2009, Las Vegas, NV, USA, November 30 - December 2, 2009, Proceedings, Part I, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[4] M. Bouchard, A. Jousselme, P. Doré, A proof for the positive definiteness of the Jaccard index matrix, International Journal of Approximate Reasoning, Vol. 54, Iss. 5, 2013, pp. 615-626.

[5] P. Burt, E. Adelson, The Laplacian Pyramid as a Compact Image Code, IEEE Transactions on Communications, Vol. 31, Iss. 4, 1983, pp. 532-540.

[6] G. Casella, Matrix Algebra : Theory, Computations, and Applications in Statistics, Springer New York, New York, NY, 2007.

[7] O. Chum, Two-View Geometry Estimation by Random Sample and Consensus, Thesis, Czech Technical University in Prague, July 1, 2005.

[8] J. Daintith and E. Wright, A Dictionary of Computing (6 ed.), Oxford University Press, England; United Kingdom, 2008.

[9] T. Dalling, Explaining Homogeneous Coordinates & Projective Geometry, Tom Dalling, Feb 2014 [WWW]. [accessed on 2th February 2017].
Available at: http://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry/.

[10] E.R. Davies, Computer and Machine Vision: Theory, Algorithms, Practicalities, Fourth; 4; 4th ed. Academic Press, US, 2012.

[11] S. Emami, K. Levgen, D.M. Escrivá, D. Millan Escriva, N. Mahmood, J. Saragih, R. Shilkrot, Mastering OpenCV with Practical Computer Vision Projects, 1st ed. Packt Publishing, Olton, 2012.

[12] M.A. Fisher, R.C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, in: Communications of the ACM, ACM, 1981, pp. 381-395.

[13] P. Frasconi, N. Landwehr, G. Manco, J. Vreeken, Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part III, Springer International Publishing, Cham, 2016.

[14] L. Guo, J. Li, Y. Zhu, Z. Tang, A novel Features from Accelerated Segment Test algorithm based on LBP on image matching, 2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 355-358.

[15] J. Howse, OpenCV computer vision with Python: learn to capture videos, manipulate images, and track objects with Python using the OpenCV library, Packt Publishing, 2013.

[16] Z. Hu, Y.(. Tsai, Homography-Based Vision Algorithm for Traffic Sign Attribute Computation, Computer-Aided Civil and Infrastructure Engineering, Vol. 24, Iss. 6, 2009, pp. 385-400.

[17] D. Hutchison, T. Kanade, J. Kittler, Computer Vision – ECCV 2010 : 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010 : Proceedings, Part VI, Springer, Berlin/Heidelberg, 2010.

[18] E. Karami, S. Prasad, M. Shehata, Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images, Newfoundland Electrical and Computer Engineering Conference, IEEE, Newfoundland and Labrador Section, At St. John's, NL, November 2015.

[19] K.J. Karande, S. Talbar, Independent Component Analysis of Edge Information for Face Recognition, 1; 2014 ed. Springer India, New Delhi, 2014.

[20] J. Lee, W. Yu, Concurrent Tracking of Inliers and Outliers, 2014.

[21] Q. Liu, Y. Li, p-Norm SDD tensors and eigenvalue localization, Journal of Inequalities and Applications, Vol. 2016, Iss. 1, 2016, pp. 1-13.

[22] D.G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision, Vol. 60, Iss. 2, 2004, pp. 91-110.

[23] M.S. Nixon, A.S. Aguado, I. Books24x7, Feature extraction & image processing for computer vision, third edition, 3rd ed. Academic Press, Kidlington, Oxford, U.K, 2012.

[24] P.M. Panchal, S.R. Panchal, S.K. Shah, A Comparison of SIFT and SURF, International Journal of Innovative Research in Computer and Communication Engineering, Vol. 1, Issue 2, April 2013.

[25] B. Přibyl, A. Chalmers, P. Zemčík, L. Hooberman, M. Čadík, Evaluation of feature point detection in high dynamic range imagery, Journal of Visual Communication and Image Representation, Vol. 38, 2016, pp. 141-160.

[26] E. Rosten, FAST Corner Detection [WWW]. [accessed on 7th January 2017] Available at: https://www.edwardrosten.com/work/fast.html.

[27] E. Rosten, R. Porter, T. Drummond, Faster and Better: A Machine Learning Approach to Corner Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, Iss. 1, 2008, pp. 105-119.

[28] Y. Ruan, Z. Wei, Discriminative descriptors for object tracking, Journal of Visual Communication and Image Representation, Vol. 35, 2016, pp. 146-154.

[29] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, ORB: An efficient alternative to SIFT or SURF, 2011 International Conference on Computer Vision, pp. 2564-2571.

[30] L.G. Shapiro, G.C. Stockman, Computer vision, Prentice Hall, New Jersey, 2001, pp. 328-339.

[31] Z.M. Shi, B.Y. Geng, Z.H. Wu, Y.W. Dong, An Image Matching Method Based on SIFT Feature, Applied Mechanics and Materials, Vol. 170-173, 2012, pp. 2855.

[32] K. Singh, S. Upadhyaya, Outlier Detection: Applications And Techniques, International Journal of Computer Science Issues (IJCSI), Vol. 9, Iss. 1, 2012, pp. 307-323.

[33] J.E. Solem, Programming Computer Vision with Python, O'Reilly Media, 2012

[34] Y. Wu, J. Lim, M. Yang, Object Tracking Benchmark, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 37, Iss. 9, 2015, pp. 1834-1848.

[35] H. Xu, C. Lu, R. Berendt, N. Jha, M. Mandal, Automatic Nuclei Detection based on Generalized Laplacian of Gaussian Filters, IEEE Journal of Biomedical and Health Informatics, 2016, pp. 1.

[36] Z. Yang, B.W. Ling, C. Bingham, Approximate affine linear relationship between L1 norm objective functional values and L2 norm constraint bounds, IET Signal Processing, Vol. 9, Iss. 9, 2015, pp. 670-680.

[37] Open Source Computer Vision 3.1.0 Documentation [WWW]. [accessed on 10th November 2016].
Available at: http://docs.opencv.org/3.1.0/.

[38] Visual Tracker Benchmark [WWW]. [accessed on 22th November 2016].
Available at: http://www.visual-tracking.net.