Multi-View Object Recognition and Classification. Graph-Based Representation of Visual Features and Structured Learning and Prediction.

Bernard Hernández Pérez

KTH Datavetenskap
och kommunikation

Degree project in
Computer Science
Second cycle
Stockholm, Sweden 2013

ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

MASTER THESIS

Multi-View Object Recognition and Classification. Graph-Based Representation of Visual Features and Structured Learning and Prediction.

Author: Bernard Hernández Pérez
Supervisor: Mårten Björkman
Examiner: Stefan Carlsson

TRITA xxx 2013-10

ABSTRACT

Computer Vision is a subfield within artificial intelligence that includes methods for acquisition, processing, analysis and understanding of images to get results in numerical or symbolic form. The information provided by the results is used to make decisions. We do not speak of Computer Vision in isolation, interaction with other fields is inevitable and deserve particular attention *image processing*, *pattern recognition* and *Machine Learning*.

The main objective of this project is to analyze the behavior of visual feature extraction algorithms and their effectiveness in decision making. The detection of an object in an image, its classification and recognition are the type of decisions that are studied.

Feature extraction algorithms are applied to attempt *multi-view object recognition*. To tackle this problem a new approach is proposed. This approach creates a graph-based representation of the object using cluster analysis recursively. The nodes of the graph represent the main physical components that make up the object. *Support Vector Machines* (SVMs) are used to classify the nodes, thus classes are classified independently. Finally, the graph-based representation of the object is exploited to drop the assumption of independence and find relations between classes using *Structured Output-Support Vector Machines* (SO-SVMs).

ABSTRAKT

Datorseende är ett delområde inom artificiell intelligens som innehåller metoder för förvärv, bildbehandling, analys och förståelse av bilder för att få resultat i numerisk eller symbolisk form. Informationen som resultatet ger används för att fatta beslut. Vi kan inte tala om visioner i isolering, samspel med andra områden är oundviklig och förtjänar särskild uppmärksamhet *bildbehandling*, *mönsterigenkänning* och *maskininlärning*.

Huvudsyftet med detta projekt är att analysera beteendet hos visuella algoritmers särdragsextraktion och deras effektivitet i beslutsfattande. Upptäckten av ett objekt i en bild, dess klassificering och erkännande är den typ av beslut som studeras.

Algoritmers särdragsextraktion tillämpas för att försöka *erkänna objekts mång-vy*. För att tackla detta problem har ett nytt tillvägagångssätt föreslgits. Detta tillvägagångssätt skapar en grafbaserad representation av objektet med hjälp av rekursiv klusteranalys. Noderna i grafen representerar de viktigaste fysiska komponenterna i objektet. *Support Vector Machines (SVMs)* används för att klassificera noderna, dessa klasser klassificeras självständigt. Slutligen, grafbaserad representation av objekt utnyttjas för att släppa antagandet om oberoende och hitta relationer mellan klasser genom att använda *Structured Output - Support Vector Machines (SO-SVMs)*.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

API	**A**pplication **P**rogramming **I**nterface
ASIFT	**A**ffine **S**cale **I**nvariant **F**eature **T**ransform
BGR	**B**lue **G**reen **R**ed
BSD	**B**erkeley **S**oftware **D**istribution
CV	**C**omputer **V**ision
DSF	**D**jango **S**oftware **F**oundation
FLANN	**F**ast **L**earning **A**pproximated **N**earest **N**eighbors
GPL	GNU **G**eneral **P**ublic **L**icense
GPS	**G**lobal **P**ositioning **S**ystem
GPU	**G**raphics **P**rocessing **U**nit
HTML	**H**yper**T**ext **M**arkup **L**anguage
KNN	**K** **N**earest **N**eighbors
LoG	**L**aplacian **o**f **G**aussian
MoG	**M**ixture **o**f **G**aussians
PSF	**P**ython **S**oftware **F**oundation

1

INTRODUCTION

In this project report some of the algorithms for detection and description of visual features in images are analyzed. These features are used in applications related to *Computer Vision* [1] to detect, classify or recognize objects in images.

This project report is organized as follows. First, Chapter 1 introduces the reader within the context of this project, *Computer Vision*, and discuss some of its possible applications. It also motivates the selection of the project topic and describes the project statement and used tools. The theoretical background upon which the method proposed in this project is based is reviewed in Chapter 2. In Chapter 3 some approaches to solve the multi-view object recognition problem are briefly explained and a new approach is proposed. The experiments and results are discussed in Chapter 4. Finally, the conclusions and some extensions are left for Chapter 5.

This chapter begins with some existing applications that make use of *Computer Vision*. The generic statement of the problem to be addressed and the activities to be carried out are explained below. Finally, this chapter concludes with a brief description of the tools used in this project.

1.1. Context

Computer vision (CV) aims to reproduce some of the capabilities of the human visual system in order to provide to machines the ability to see, recognize and interpret images [1] [2]. For humans the eye is without doubt one of the most important senses and is being increasingly exploited, within its limitations, for various computer systems. To emphasize the importance of computer vision, not only in the academic field but also in everyday life, some applications that make use of the discoveries

[1] Subfield of artificial intelligence that aims to program a computer to understand a scene or features in an image.

and advances made by various researchers in the field are classified within their main topic in [3]. A basic introduction to some of the mentioned applications are explained below.

A) Medicine

The basic goal of computer vision in this area is the processing and analysis of images, detecting patterns and sometimes doing 3D reconstruction, to facilitate and assist the correct diagnosis by the specialists. Common applications include automatic detection and characterization of tumors, analysis of digital mammographies or hyperspectral image analysis to extract the composition of the elements contained in an image. Also it is used in more complex surgery as an aid; for example, estimating the motion of the surface of the brain.

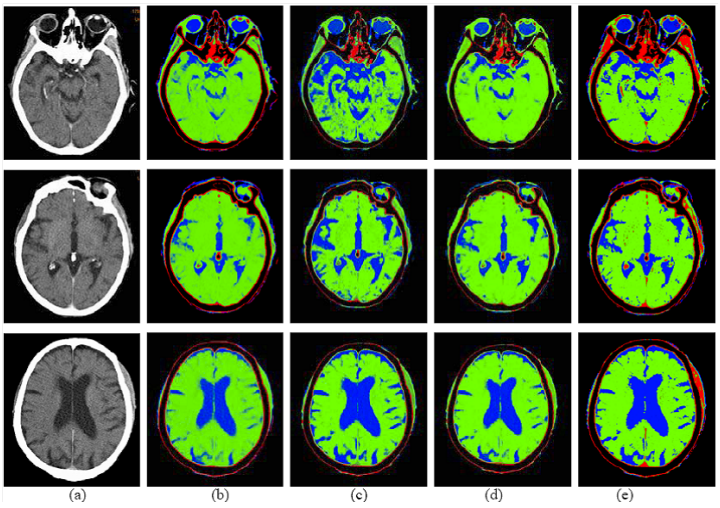


Figure 1.1: Brain segmentation in CT images. Images captured with a CT scanner which generates a two-dimensional image of a section pertaining to a three dimensional object. The colors representation are: red for low intensity bones, green for brain matter and blue for fluids. The original image and the result of applying different methods of image segmentation are shown. Image taken from *Soft segmentation of CT brain data* [4].

B) Security systems

In this area there are numerous applications. There exist many systems to control the access to restricted areas, automatically detecting intruders by analysing the images recorded by a video camera. Additionally, there are systems that not only detect intruders but may also

identify his face comparing it with those stored in a database. There are systems that can detect abandoned objects or strangers, usually inside buildings [5].

The previously mentioned systems are considered as large scale security software but there are also non-large security software, for example to restrict access to computers or mobile devices.

C) Game industry

This is a key sector and has contributed significantly to the development of computer vision. Examples of its use are the *Kinect* device, developed by *Microsoft* ®, and the *Eye Toy* device, developed by *Play Station* ®, for gesture recognition. In the new generation of video game consoles the researchers focus lot of their effort to improve player-console interaction with such devices. The analysis of the gaze, *eye tracking*, is incorporated to estimate the locations of the screen to which the players are paying more attention to, for example, help to point to one target in shooting games [6] [7]. A relatively new company that is dedicated to the analysis of the gaze is *Tobii* ® [2].

D) Robotics

In the process of developing a robot, one of the most complicated goals to achieve is to interpret the world around them through the information from their environment [8]. This information can be captured by various devices such as sensors or video cameras, and can be used to locate/recognize objects or even monitoring them.

E) Visual shopping

This field is under development and involves taking a photo of a product in a store, search in internet for information about this product or similar products and thus compare their features, quality and price. An innovative company in this sector is *OculusAI* ® [3], with its visual shopping mobile application *Productify* ®.

F) Automatic code reading and object recognition

The use of bar codes and QR codes [4] is very common to detect and recognize objects (i.e. airport tickets or products in supermarkets). It

[2]Link Tobii: http://www.tobii.com/
[3]Link OculusAI: http://oculusai.com/
[4]QR code is the name of a trademark for a special type of two-dimensional bar code.

also helps to collect statistics and information (i.e. metro users or personal information scanning ID cards or passports). Furthermore object recognition aims to find an identify objects in an image or video based on their properties (i.e. shape, features, pose, ...) as human beings. This task is still a challenge for computer vision systems.

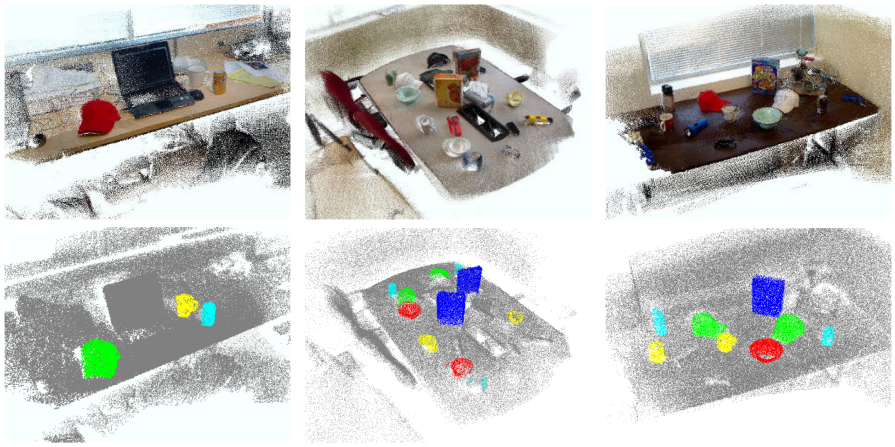


Figure 1.2: Result obtained when labelling 3D complex scenes. In the first row the 3D scene reconstruction are shown and in the second row their elements are labelled. The objects are coloured by their categories: bowl in red, cap in green, cereals box in blue, coffee cup in yellow and the can of soda in cyan. Image taken from *Detection-based object labelling in 3D scenes* [9].

The above are just some examples of applications and companies engaged in research and development of systems based on computer vision. For more information, see the list [5] of companies developing computer vision products categorized by their main area of application.

1.2. Motivation

Nowadays, all electronic devices (i.e. mobile phones or tablets) have an integrated camera. This feature can be exploited by the users in many ways. One is the so called *"visual shopping"* which main idea is to snap pictures of products, recognize matches against a database and suggest similar products to the user. This recognition problem can be extended

[5] Link List: http://www.cs.ubc.ca/~lowe/vision.html

to many areas such as robotics, statistics recollection or game industry among others.

It is desirable to recognize the product more or less regardless of the position in which the picture was taken. As is said *"An image worth more than thousand words*", but not all the images are equally important/useful for this purpose. For a given picture, the features captured and their relevance vary between different views, and these variations lead common recognition algorithms to fail. This enhancement would improve the product recognition both quantitative and qualitative and consequently the user experience. This is a wide and very challenging problem so a gradual process will be carried on in this thesis.

1.3. Project Statement

The subject of this MSc thesis project is to investigate, compare and implement one or more existing approaches to enhance the *multi-view object recognition* performance and potentially propose new ideas. Furthermore, the methods will be adapted and evaluated in real objects, presumably multi-view shoe recognition. The main reasons to use shoes for recognition are:

- The shoes have multiple canonical views. Some of them are more representative than others, allowing a thorough analysis of the relevance within the views.
- The shoes contain visual features that will facilitate their classification or recognition. The visual features are the elements that make up the shoe.
- Find images of shoes in internet is relatively easy. In particular, it is important to find images with different canonical views of the same object.

Preliminarily, partial experiments and results to verify the correct implementation of the software are analyzed. Those experiments are related to necessary partial goals which are explained below. The partial goals will help to achieve the main goal; that is, object recognition and classification.

A) Feature extraction

In this process the points of interest, also known as *keypoints*, to tackle image object recognition are detected. For each keypoint the region of interest is defined and its properties are described using a descriptor vector. This vector contains magnitudes and orientations describing the surrounding keypoint region. Thus, a set of keypoints and their descriptor is the desired output.

Once the keypoints and their descriptor are obtained, a comparison between the different extraction techniques available, their computational time, their robustness to image transformations and their performance are evaluated. Furthermore, the detector and descriptor that best suit the image object recognition task are selected.

B) Clustering the keypoints to represent visual features.

The previously extracted features are grouped to identify important physical elements in the object. This approach attempts to mimic the method of selection, classification and recognition of objects of human beings; that is, extracting inner structures and physical properties.

C) Classification of visual features

It is important to analyze whether the previously obtained descriptor is feasible for object classification and its limitations. The total amount of different classes, the inter-class variability or the intra-class variability are some possible drawbacks that can appear when performing the classification step. Afterwards, the similarities between the objects are computed. It seems reasonable to expect objects with many similarities to lye in the same region of the space. This concept is shown in Figure 1.3 with different shoes; particularly trainers and high-heel shoes.

D) Object recognition

Other ways to improve object recognition with several canonical views are evaluated. A graph-based representation of the object is attempted to improve the performance. Note that this representation relates nodes of the graph with physical elements in the object. Thus, this new graph-based representation provides many possible criteria to compute the similarity within two objects.

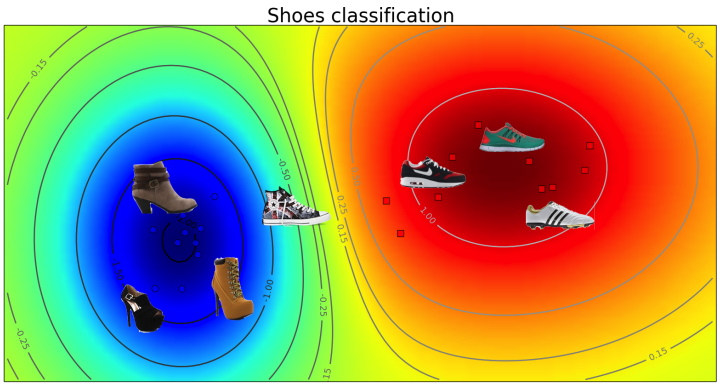


Figure 1.3: Example of shoe classification based on their visual features. Particularly two classes are considered; sport shoes and high-heel shoes.

In this project several methods for object recognition in images are studied, compared and implemented. The final goal is to develop a system able to recognize objects in a database. Since the range of possibilities is very wide, other approaches are explained and a new own approach proposed. The following tasks have been carried out during this project:

1. Study the literature related to the methods for detection and description of visual features in images and the main concepts related to computer vision.

2. Study the literature related with the machine learning methods to perform pattern recognition and classification and the algorithms to group elements in clusters.

3. Comparison and evaluation of different types of visual features detectors and descriptors. Selection of the most appropriate detector and descriptor to tackle the problem of object recognition.

4. Simple database creation and use of existing dataset collections. Effort to set up the experiment with modifications and conclusions about those datasets.

5. Write the report.

1.4. Tools

All the work related to this project was developed in a MacBook Pro with an Intel i7@2.9GHz processor and 8 GB RAM. The tools used for the development of the final project are described briefly below. Note that, as far as possible, open source projects and libraries have been used. The goal of this section is not to describe in detail the tools. To that end, pointers to more information are given where it is necessary.

A) Python

Python is a high-level programming language which emphasizes code readability. It is a dynamic language, so the data types are assigned in execution time, and generally is used to code *scripts*. It supports multiple programming paradigms, including object-oriented, imperative and functional programming. Python is a free and open source software managed by the *Python Software Foundation* (PSF). Most part of the experimental figures included in this report were produced using Python [10] and the Matplotlib Python plotting library.

B) Matlab

MATLAB[6] (***mat**rix **lab**oratory*) is a numerical computing environment developed by *MathWorks*. MATLAB allows matrix manipulations, function and data representation, user interface creation and interaction with programs written in other programming languages. It has a proprietary license and currently the version 2013b is available for Windows, Linux and Mac OS.

C) OpenCV

Open Source Computer Vision [7] (OpenCV) [11] is a computer vision library initially developed by *Intel* ®, though currently is open source software published under BSD license. This library provides a widely set of functions to work with computer vision, emphasizing their use in real-time applications. For this reason it is developed in optimized C/C++. It can use *Intel* proprietary optimized routines to accelerate itself. It has bindings for C, C++, Python and Java. OpenCV is also cross-platform, so it is supported on Windows, Linux, Mac OS, iOS and Android.

[6] Link MATLAB: http://www.mathworks.com/
[7] Link OpenCV: http://opencv.org/

D) Scikit-learn

Scikit-learn[8] [12] is a machine learning library for Python. Its documentation and implementation are open source and it is published under BSD license. It started as a *Google Summer of Code* project developed by David Cournapeau and has been rewritten and extend by other developers since then. It is designed to interoperate with the Python numerical and scientific libraries *NumPy* and *SciPy*. Among their content it includes algorithms to perform classification, regression and clustering.

E) PyStruct

PyStruct [9] [13] is a structured learning and prediction library for Python. It was created by Andreas Mueller and it is open source. Currently its first stable version *PyStruct 0.1* is available.

[8] Link Scikit-learn: http://scikit-learn.org/
[9] Link PyStruct: http://pystruct.github.io/

2

BACKGROUND

In this chapter, the basic foundations of feature extraction, classification and clustering are introduced. These are very important to fully understand the methodology described in Chapter 3. Nonetheless, the aim is not to be as exhaustive as possible and instead provide with basic knowledge.

Section 2.1 explains the basic concepts of feature extraction, briefly reviews some of the feature extraction methods and focus on *Scale Invariant Feature Transform* (SIFT) and *Speeded-Up Robust Features* (SURF). The classification problem and its achievement using *Support Vector Machines* (SVMs) is explained in Section 2.2. In Section 3.3, some cluster analysis methods are described and evaluated in a synthetic example. Finally, possible interesting research questions are stated in 2.4.

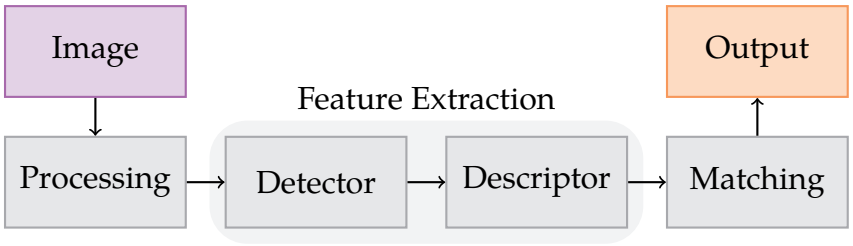


Figure 2.1: Generic diagram to perform object recognition. The preprocessing is optional and can boost the performance. Feature extraction is the core block and matching is use to find correspondences within objects using the precomputed features.

Object recognition systems aim to find and identify objects in an image and many approaches to solve this task have been implemented. The general diagram for object recognition based on feature extraction is shown in Figure 2.1. The preprocessing step is optional but can boost the final performance of the system considerably. Possible preprocessing techniques involve image enhancement and segmentation to obtain a set of sub-

images, sometime referred to as regions-of-interest (ROIs). This concept reflects the fact that images frequently contain collections of objects each of which can be the basis for a region. Feature extraction aims to transform the input data (i.e. an image) into a set of features and it is explained in Section 2.1. Matching is a general concept to describe the task of finding correspondences between two elements that has to be carried out for recognition or classification. It can attempt a simple comparison between the features extracted or more complex comparison systems (i.e. graph-based matching implements a big set of comparison techniques).

An image is a 2-dimensional representation of a 3D scene. Signals and images in our environment are analog; that is, they exist in a continuous time and have a continue range of values. Thus, it is necessary to digitalize the images to be stored and used by computers. Consequently, digital images are defined in a discrete time and a discrete range of values.

2.1. Feature Extraction

Once the image is acquired and the preprocessing is done, the most relevant features have to be extracted in order to recognize the object. The amount of data in an image is huge and thus *detectors* are used to locate the pixels of interest; that is, positions $\mathbf{p} = (x, y)$ in the image. Afterwards, the *descriptors* are used to describe the region around the previously detected positions.

A) Detectors

To select the most important points in an image, usually denoted as *keypoints*, detectors are used. Usually, the keypoints are located in regions of the image with high contrast, edges or corners. Note that the keypoint itself does not provide enough information, thus a region of interest around the keypoint is defined.

B) Descriptors

The region of interest related to each keypoint is described using descriptors. Generally, the result is $\mathbf{v} \in \mathbb{R}^d$, so $\mathbf{v}$ is a d-dimensional vector describing the physical region properties; that is, texture or shape among others properties are described.

The possible transformations in the projection of the object in two different images are many. It seems reasonable to select those detectors and

descriptors that are not affected by transformations. They will perform better in complex scenarios due to their robustness against transformations.

2.1.1. Local Feature Detectors and Descriptors

The literature related with visual feature detection and description in images is very wide and it is very difficult to explain all the algorithms and contributions thoroughly. The aim of this subsection is to overview some of the main ideas proposed.

It started with *global descriptors* to describe all the content of the image using its color or intensity. To overcome the limitations of these methods and improve their performance a new approach, based on the segmentation of an image in smaller regions, was proposed. Thus, the descriptors applied to regions of an image are called *local descriptors* and are core in this project. Along the time, several techniques were developed to handle and palliate the effects of transformations in images. Those transformations can be scale variance, illumination variance, geometric transformation or even noise. Nowadays, there exist many detectors and descriptors and some of them are briefly explained below.

A) FAST

Features from Acelerated Segment Test (FAST) [14] detector explores the intensity value of those pixels within a circle with radius r around the selected pixel $\mathbf{p} = (x, y)$ (see Figure 2.2a). A pixel within the circle is considered *"bright"* if its intensity is at least t times higher than the candidate pixel $\mathbf{p}$. Opposite, it is considered *"dark"* if the intensity is at least t times lower. The candidate pixel $\mathbf{p}$ is marked as point of interest if there exist a n pixels length arc in the circle. In the original implementation the values are $r = 3$ and $n = 9$.

B) BRIEF

Binary Robust Independent Elementary Features (BRIEF) [15] descriptor uses binary *strings* to describe the keypoints. Thus, the *Hamming* [1] distance is used to calculate the similarities between descriptors in

[1]Given two *strings* with same length, the *Hamming* distance is the total number of positions where the symbols are different. It measures the minimum number of replacements needed to make both *strings* equal.

the matching step. Usually, *Hamming* distance is computed more efficiently than *Euclidean* distance. BRIEF is very sensitive to noise, thus the image is blurred by applying a median filter.

The *bit* values in the descriptor are computed comparing all possible pairs of intensity values within an image region centered in the keypoint. Let us define the pair of pixels to compare as $\mathbf{p_1} = (x_1, y_1, i_1)$ and $\mathbf{p_2} = (x_2, y_2, i_2)$, where i_i represents the intensity value. If the intensity value of the first pixel i_1 is higher than the intensity value of the second pixel i_2, the bit is set to 1, otherwise is set to 0. The original descriptor length is 512 bits for a region of 48x48 pixel. The basic form of BRIEF is not rotation invariant.

C) STAR

The authors of STAR [16] aimed to create a new multi-scale detector with full spatial resolution because other detectors (i.e. SIFT) decimate the input image. This decimation process leads to problems in the location precision of the keypoints detected in decimated images with respect to the original image. STAR uses a *Laplacian of Gaussian* (DoG) approximation and gets its name from the shape of the mask used (see Figure 2.2b). This mask is invariant to ration and allows the use of integral images to compute the calculations efficiently. The scale-space is created using masks with different size, thus the interpolation is avoided.

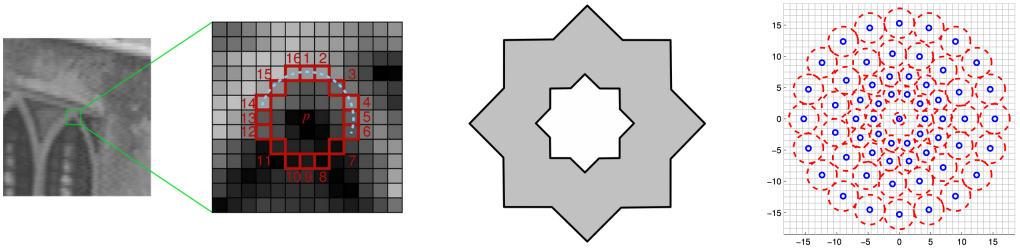
D) BRISK

Binary Robust Invariant Scalable Keypoints (BRISK) [17] detector and descriptor uses the scale-space technique to detect keypoints. The location and scale of each keypoint is obtained fitting a quadratic function in the continuous domain. The main idea is to use a pattern (see Figure 2.2c) to sample the keypoint's neighbourhood obtaining gray values. Later, pairs of those values are compared, setting the bit to 1 if the intensity value of the first element is higher or 0 otherwise. The matching is done efficiently due the binary nature of the descriptor.

E) ORB

Oriented-FAST and Rotated-BRIEF (ORB) [18] is a detector and descriptor proposed as a computationally efficient alternative to SIFT and SURF. The authors define the robustness and matching accuracy performance as good as the one provided by SIFT and better than the

performance provided by SURF. It is based on the FAST detector and the BRIEF descriptor. However, ORB adds an orientation component to FAST (oFAST) and compute the BRIEF descriptors more efficiently and improving their robustness to rotation transformations (rBRIEF).



(a) *detector* FAST **(b)** *detector* STAR **(c)** *descriptor* BRISK

Figure 2.2: (a) Example of FAST detector; (b) Mask used in STAR detector; c) Sample pattern used in BRISK. Images taken from the official FAST webpage (http://www.edwardrosten.com/work/fast.html), [16] and [17] respectively.

For a thorough explanation in the history and evolution of detectors and descriptors, some implementation details and its general classification see [19]. The feature extraction techniques proposed depend on the purpose and the area of application, see [20] to understand the most modern feature extraction algorithms within robotics. In this project the *Scale Invariant Feature Transform* (SIFT) [21] and *Speeded-Up Robust Features* (SURF) [22] are explained.

2.1.2. *Scale Invariant Feature Transform*

Scale Invariant Feature Transform (SIFT) is an algorithm to detect points of interest and describe the local features related with those points in images. SIFT was published by David Lowe [21] and patented in the US; the owner is the University of British Columbia. It has been reviewed and modified by other researchers to improve its performance and robustness. As a result, some variants as PCA-SIFT [23] or ASIFT [24] have been proposed. To achieve a reliable object recognition system, it is important to extract features from images that are robust to possible variations in the input image. Potential changes are scale, illumination, rotation or geometric transformations among others. Usually, the keypoints detected and

described by SIFT correspond to areas of the image with high contrast, as object edges or corners. Another interesting property to consider in feature extraction algorithms is the ability to extract the same keypoints no matter the projection of the scene in the image.

Figure 2.3 shows the detection and description blocks to extract the SIFT descriptors from an image. The aim and the implementation details for each of the steps shown in Figure 2.3 are explained below. They are explained sequentially.

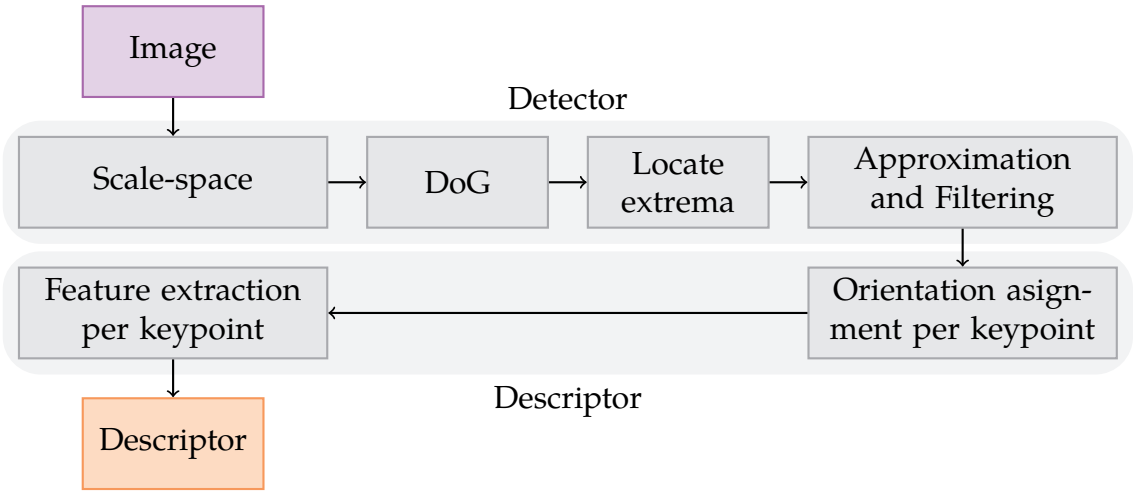


Figure 2.3: Process diagram for object recognition with detection and description blocks to extract the SIFT descriptors from an image.

A) Scale-space

In the first step of the algorithm, the image $I(x, y)$ is convolved iteratively using the spatial gaussian filter $G(x, y, \sigma)$ defined in Equation 2.1.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{2.1}$$

The convolution of the original image $I(x, y)$ with the spatial gaussian filter $G(x, y, \sigma)$ filters high frequencies while low frequencies remains. Thus, the original image is smoothed (*blurred*) resulting in a new image $L(x, y, \sigma)$ defined in Equation 2.2.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.2}$$

Scale-space aims to create a set of images (see Figure 2.4 for *Lena* image) to obtain a *scale-space pyramid*. The scale-space varies within the available SIFT implementations. This report is focused on the implementation proposed by Lowe [21].

Figure 2.4: SIFT scale-space for the *Lena* image.

The factor k indicates the scale increment and it is defined by $k = 2^{1/s}$, where $s + 3$ is the total number of images per octave. In the original SIFT implementation, Lowe uses 4 octaves with 5 images per octave. Table 2.1 shows the generic values to create the scale-space. Furthermore, the image is decimated by 2 when σ doubles its value, thus the computational time is reduced as well.

	Octaves							
Scale	1	2	3	4	1	2	3	4
blur 1	$k\sigma = \sigma_1$	$2\sigma_1$	$4\sigma_1$	$8\sigma_1$	0.707	1.414	2.828	5.656
blur 2	$k^2\sigma = \sigma_2$	$2\sigma_2$	$4\sigma_2$	$8\sigma_2$	1.000	2.000	4.000	8.000
blur 3	$k^3\sigma = \sigma_3$	$2\sigma_3$	$4\sigma_3$	$8\sigma_3$	1.414	2.828	5.656	11.313
blur 4	$k^4\sigma = \sigma_4$	$2\sigma_4$	$4\sigma_4$	$8\sigma_4$	2.000	4.000	8.000	16.000
blur 5	$k^5\sigma = \sigma_5$	$2\sigma_5$	$4\sigma_5$	$8\sigma_5$	2.828	5.656	11.313	22.627

Table 2.1: Scale factor values; 4 octaves with 5 images per octave. The initial value of σ is $\sigma = 1/2$. The colour represents the overlapping scale factor between two octaves; red for octaves 1-2 and green for octaves 3-4.

One octave has half of its scale factor values in common with the previous octave (see colours in Table 2.1). The decimation applied to the image does not cause large losses in accuracy because of the overlapping within octaves.

B) Difference of Gaussians (*DoG*)

The spatial filter *Laplacian of Gaussian (LoG)* detects borders or edges robustly. However LoG is computationally expensive so it is replaced by an approximation based on *Difference of Gaussians* (DoG). The validity of this approach was demonstrated by Lindeberg [25]. The difference of gaussians $D(x, y, \sigma)$ is defined in Equation 2.3.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.3}$$

Thus, the approximation $D(x, y, \sigma)$ is the difference of two adjacent scales in the same octave. This step provides a set of images called *DoG-pyramid* and the aspect of the images are similar to the ones represented in Figure 2.5.

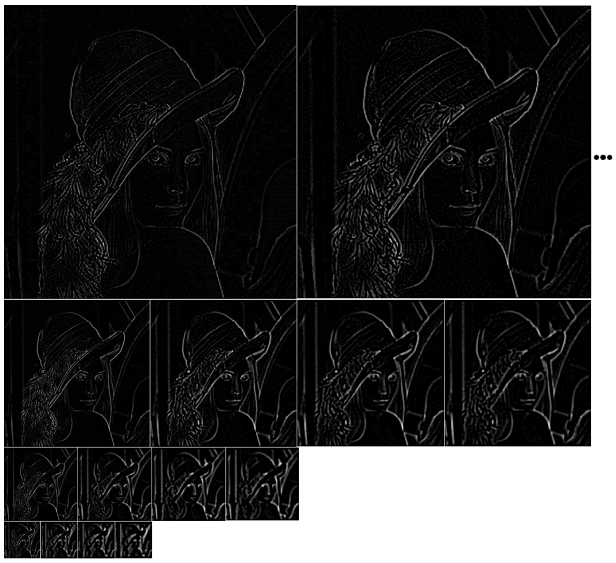


Figure 2.5: Example of the Difference of Gaussians (*DoG*) used in SIFT for the *Lena* image. Note that each octave has 4 images.

C) Locate extrema in *DoG*

Given the previously obtained *DoG-pyramid* with the set of images at different scales, the *extrema* within the images are explored because

they are potentially considered points of interest (keypoints). To locate the extrema, the intensity of each pixel $D(x, y, \sigma)$ is compared with its neighbors. Thus, it is compared with 8 neighbors at the same level, 9 neighbors at a lower level and 9 neighbors at a higher level. This comparison is shown in Figure 2.6. It is only considered an extremum if the value of the point that is being compared is higher (or lower) than its 26 neighbors.

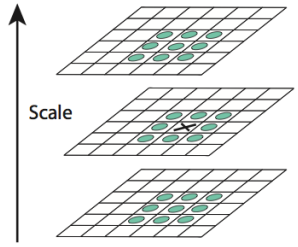


Figure 2.6: Extrema location in *DoG* images. The pixel represented with a cross is compared with its 26 neighbours represented as circles. Image taken from [21].

D) Keypoints approximation and filtering

The positions (x,y) for the extrema located in the previous step are obtained at different scales. Consequently, those positions can be defined with decimal values, but an image matrix can be accessed only to positions defined by integer values. In the most recent SIFT implementations an interpolation method, based on approximating the data around the extremum with a 3D quadratic function, is used. Let us call each point around the extremum as *sample point*. The approximation, illustrated in Figure 2.7, is explained below.

The interpolation is done using the quadratic Taylor series expansion over the difference of gaussians $D(x, y, \sigma)$ as expressed in Equation 2.4. D and its derived form are evaluated in the sample points and $\mathbf{x} = (x, y, \sigma)$ is the offset for the sample points.

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial^2 \mathbf{x}}\mathbf{x} \tag{2.4}$$

The final extremum location $\hat{\mathbf{x}}$ coincides with one of the sample points. In the Figure 2.7 the final extremum location is the same as the sample point represented in green. This location is found deriving the Taylor

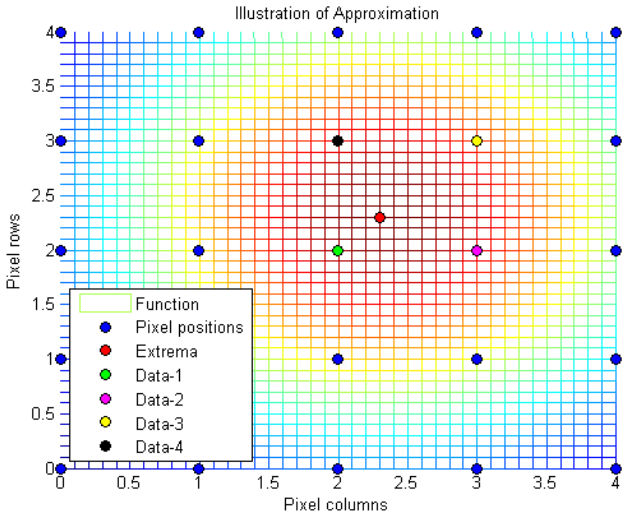


Figure 2.7: Extremum approximation using its samples points. The function represented is a gaussian to make the illustration more intuitive.

series expansion with respect to $\mathbf{x} = (x, y, \sigma)$ and setting it to 0. The coordinate axis is centered in the current sample point defining the *offset* $\hat{\mathbf{x}}$ as the distance between the sample point and the extremum. The expression to obtain the *offset* $\hat{\mathbf{x}}$ is defined by Equation 2.5.

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial^2 \mathbf{x}} \frac{\partial D}{\partial \mathbf{x}} \tag{2.5}$$

If the *offset* $\hat{\mathbf{x}}$ is higher than 0.5 in any of the dimensions, the extremum is lying closer to another sample point and the process is repeated for a new one. The final *offset* $\hat{\mathbf{x}}$ is added to the extrema to interpolate the final extrema position. This addition is represented in Equation 2.6 where $\hat{\mathbf{x}}_I$ is the interpolated position, $\mathbf{x}_E$ is the extremum and $\hat{\mathbf{x}}$ is the *offset*.

$$\hat{\mathbf{x}}_I = \mathbf{x}_E + \hat{\mathbf{x}} \tag{2.6}$$

The extrema are located in the images within the scale-space, hence the amount of keypoints detected is huge and some of them are unstable. In the next step the extrema with low contrast or with variation in only one direction are remove because they are very sensitive to noise.

Firstly, the value obtained evaluating the function represented in Equation 2.7 in the extremum, $D(\hat{\mathbf{x}}_I)$, is used to remove unstable extrema due to low contrast. In the experiments realized by Lowe, the extrema with a value $|D(\hat{\mathbf{x}}_I)| \leq 0{,}03$ are removed.

$$D(\hat{\mathbf{x}}_I) = D + \frac{1}{2}\frac{\partial^2 D^T}{\partial^2 \mathbf{x}}\hat{\mathbf{x}}_I \tag{2.7}$$

Secondly, those extrema located along the edges are removed. Intuitively the edges have a small curvature (variation or gradient) along themselves and a high curvature in the perpendicular direction. If $\hat{\mathbf{x}}_I$ is a corner the curvature is high in both directions. To discard the extrema located in edges the *ratio* between the two curvatures is calculated and those with high ratio are removed. Those curvatures can be computed with a *Hessian* matrix approximation $\mathbf{H}$. This approximation is calculated using pixel differences in the same location and scale that the evaluated extrema; that is $\hat{\mathbf{x}}_I = (x, y, \sigma)$. The Hessian matrix is defined by the Equation 2.8. It is symmetric so $D_{xy} = D_{yx}$.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \tag{2.8}$$

The trace and determinant for the *Hessian* matrix $\mathbf{H}$ are computed as shown in Equation 2.9 where $\alpha = r\beta$ and r is the ratio.

$$\begin{aligned} Tr(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta \\ Det(\mathbf{H}) &= D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta \end{aligned} \tag{2.9}$$

The decision is made using the Equation 2.10. In Lowe experiments, the value used for the *ratio* is r=10. Thus, extrema with a curvature ratio higher than 10 are removed and the remaining extrema are considered keypoints.

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r+1)^2}{r} \tag{2.10}$$

E) Dominant orientation assignment to keypoints

A dominant orientation is assigned to each keypoint to attempt a rotation invariant description for the surrounding pixels. For a given pixel, the gradient parameters are *magnitude* $m(x, y)$ and *orientation* $\theta(x, y)$.

They are approximated using pixel difference as indicated in Equation 2.11.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = tan^{-1}\left(L(x,y+1) - L(x,y-1)\right) / \left(L(x+1,y) - L(x-1,y)\right) \tag{2.11}$$

The pixel magnitudes and orientations lying in a gaussian which standard deviation $1{,}5\sigma$ are used to create an histogram (see Figure 2.8). Note that the represented gaussian has into account the scale where the keypoint was found $\mathbf{k} = (x, y, \sigma)$. The histogram has 36 bins in the x axis, one for each 10 degrees and the amplitude is represented in the y axis. Each pixel contributes to the histogram adding its magnitude value weighted with a gaussian window centered at the keypoint.

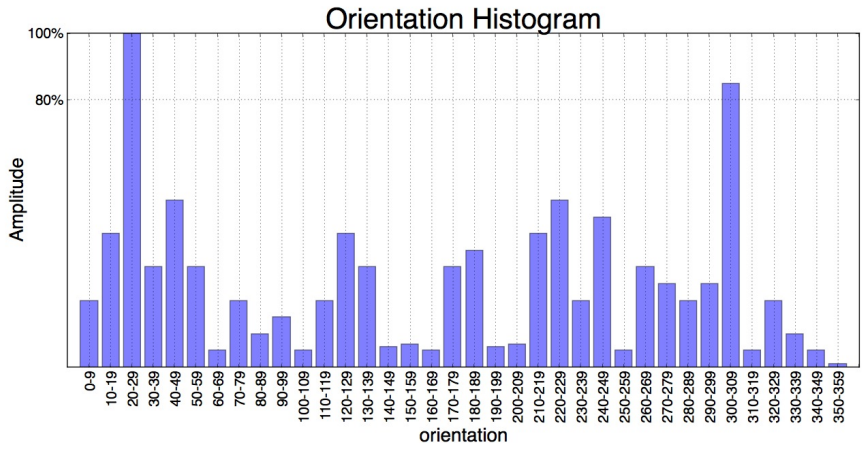


Figure 2.8: Orientation histogram with 36 bins (one for each 10 degrees). The amplitude of each bin represents the sum of magnitudes weighted by a gaussian window centered at **k** of all the pixels which orientation corresponds to that bin.

Once the histogram has been created, the dominant orientation represented by the higher bin is assigned to the keypoint. Thus, the *keypoint* is now defined as $\mathbf{k} = (x, y, \sigma, \theta_D)$, where θ_D is the dominant orientation. The keypoint is copied for all the bins which amplitude is at least 80 % the maximum amplitude. For those copies, the dominant orientation is interpolated by fitting a parabola to the three adjacent bins; the aim is to obtain higher precision.

Finally the keypoints are defined by $\mathbf{k} = (x, y, \sigma, \theta_D)$, where (x, y) is the keypoint location, σ is the scale and θ_D is the dominant orientation calculated for its region of interest.

F) Feature extraction to create the descriptor

In previous steps keypoints at different scales were selected and the orientation for its region of interest assigned. This step attempts to make the description robust to shift, scale and rotation variations in the image. Also, it attempts to describe the region around the keypoint singularly.

It is desirable to describe the region around the keypoint $\mathbf{k} = (x, y, \sigma, \theta_D)$ robustly, in particular to illumination and perspective variations. The 16x16 pixel region around the keypoint is selected and divided in 4x4 pixel windows. An orientation histogram is computed for each of the windows. Those histograms have 8 bins; that is, one for each 45 degrees as shown in Figure 2.9.

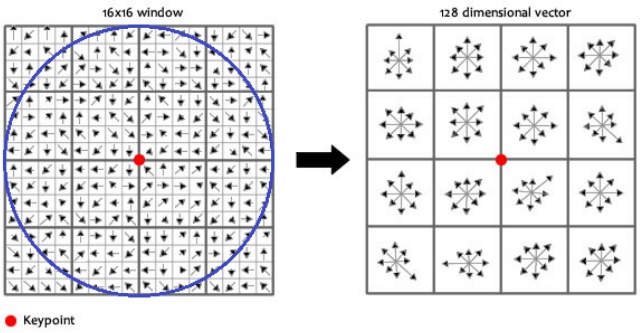


Figure 2.9: Orientation histogram to describe the region around the keypoints. The magnitudes are weighted by a gaussian (blue circunference) centered at the keypoint and added to its corresponding bin. Each histogram has 8 bins (orientations) and the magnitude is represented by the length of the arrow.

Finally, a vector with all histogram values is created for each keypoint. This vector has 128 values; that is, 4x4 windows with 8 orientations per window and it is called *descriptor*. To achieve rotation invariance, the dominant orientation for the keypoint is subtracted to its descriptor values. For illumination invariance the vector is normalized. To remove no linear effects (i.e. camera saturation) Lowe reduces the influence

of high magnitudes saturating those values higher than 0.2 within the vector and the vector is normalized again.

2.1.3. *Speeded-Up Robust Features*

Speeded-Up Robust Features (SURF) is a robust feature detector and descriptor proposed by Herbert Bay [22] and partially based in the previously explained SIFT [21]. The standard version of SURF is several times faster than SIFT and its use is widely extended in real-time applications (i.e. object tracking in video).

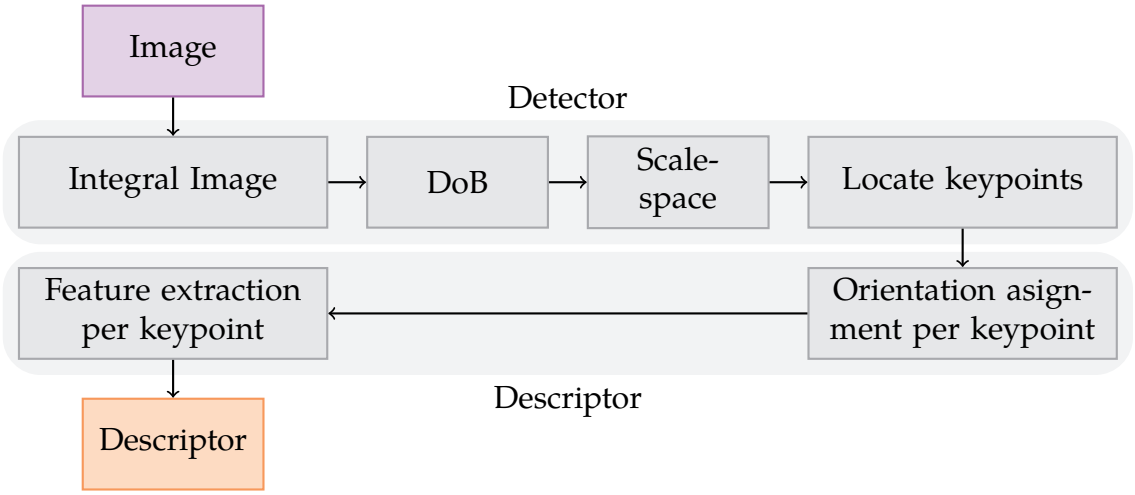


Figure 2.10: Process diagram for object recognition with detection and description blocks to extract the SURF descriptors from an image.

In the comparison of the SURF diagram (see Figure 2.10) and the previously explained SIFT diagram (see Figure 2.3) the main differences are:

- SIFT uses a Laplacian of Gaussian (LoG) approximation denoted as Difference of Gaussians (DoG). In SURF the approximation is computed more efficiently using *Difference of Boxes* (DoB). The improved efficiency is due to the use of *integral images* to calculate the sum of intensities within a box. This technique was introduced by Viola and Jones [26].
- SIFT iteratively convolves the input image with a spatial gaussian filter $G(x, y, k\sigma)$ to create the scale-space; that is, a set of blurred and

resized image at different scales. Due to Difference of Boxes (DoB), SURF creates the scale-space varying the size of the boxes directly. There exist slightly differences in the pyramidal scale-space created by SIFT and SURF, those differences will be explained later on.

The aim and the implementation details for each of the steps shown in Figure 2.10 are explained below. They are explained sequentially.

A) Integral Image

The computational time is reduced considerably due to the use of the integral image and it is the main difference introduced in SURF. The integral image value $I_\Sigma(\mathbf{x})$ at the position $\mathbf{x} = (x, y)$ represents the sum of all the values in the rectangular region defined by the origin and the position $\mathbf{x}$. The integral image expression is shown in Equation 2.12.

$$I_\Sigma(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j) \tag{2.12}$$

Once the integral image $I_\Sigma(\mathbf{x})$ is computed, the computational time to perform the sum of intensities within a rectangular area is independent of the image. The sum of intensities only needs three operations. This property is very important because filters with big size are used to create the scale-space.

B) Difference of Boxes *(DoB)*

The *Hessian* matrix is used to detect the keypoints. Given a point $\mathbf{x} = (x, y)$ in an image $I(\mathbf{x})$ and the scale σ, the *Hessian* matrix $\mathbf{H}(\mathbf{x})$ is defined in Equation 2.13. The term $L_{xx}(\mathbf{x}, \sigma)$ is the second order derivative of the spatial gaussian filter convolved with the image $I(\mathbf{x})$ at the position $\mathbf{x} = (x, y)$. Similarly the terms $L_{xy}(\mathbf{x}, \sigma)$ and $L_{yy}(\mathbf{x}, \sigma)$ are defined.

$$\mathbf{H} = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{yx}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{2.13}$$

In SURF the approximation of the Laplacian of Gaussian (DoG) was pushed one step further, due to the good performance obtained by SIFT the approximation based on Difference of Gaussians (DoG). The simpler approximation used in SURF is based on Difference of Boxes

(DoG). Figure 2.11 shows an example of this approximation. The use of both, the spatial filters and the integral image $I_{\Sigma}(\mathbf{x})$, allows an efficient calculation for the sums of intensities within a rectangular area (box).

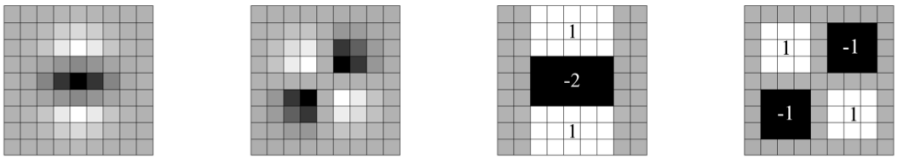


Figure 2.11: Laplacian of Gaussian (LoG) approximation based on Difference of Boxes (DoB). From left to right: second order derivatives for the gaussian filter in L_{yy} and L_{xy} directions respectively. Second order derivatives approximation for the gaussian filter ussing Difference of Boxes in D_{yy} and D_{xy} directions respectively. The value for the gray regions is 0.

C) Scale-space

SIFT iteratively convolves the input image with a spatial gaussian filter $G(x, y, k\sigma)$ to create the scale-space; that is, a set of blurred and resized images at different scales conforming the *scale-space pyramid*. Due to Difference of Boxes (DoB) and the integral image $I_{\Sigma}(\mathbf{x})$, SURF creates the scale-space varying the size of the boxes to define the mask. The mask is applied directly to the image $I(\mathbf{x})$ without incrementing the computational time. In SURF the size of the mask is increased while in SIFT the image is decimated. Thus, the *scale-space pyramids* are inverted as shown in Figure 2.12.

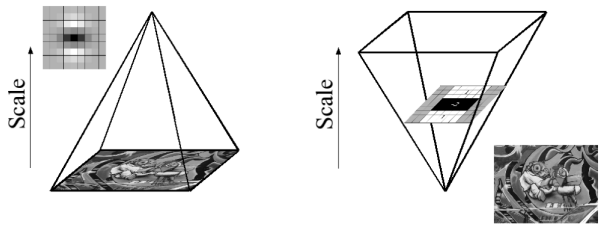


Figure 2.12: Scale-space comparison SIFT vs. SURF. The scale-space in SIFT is created by decimation of the image iteratively (left). Opposite, in SURF it is created by increasing the size of the mask (right). Image taken from [22].

The results are grouped in octaves. The octaves overlap to cover the whole scale. In the implementation proposed by Herbert Bay [22] there are 3 octaves expandable to 4. The number of pixels to define de Difference of Boxes (DoB) for each scale is shown in Figure 2.13. Note that the masks have and odd number to guarantee the existence of a central pixel.

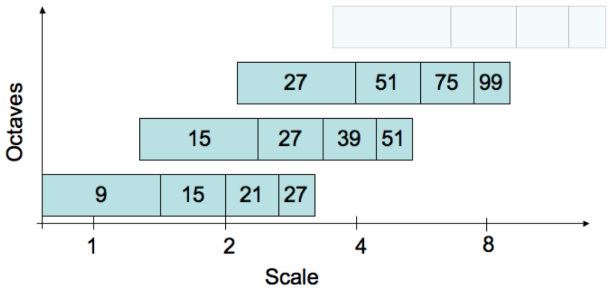


Figure 2.13: Octaves in SURF. In this representation there are 3 octaves (y axis) and different scales (x axis). The numbers represent the amount of pixels used to create the Difference of Boxes (DoB) mask. For example, 9 defines a 3x3 mask. Image taken from [22].

D) Locate keypoints

This procedure is similar to the one used in SIFT, but the Taylor series expansion is applied to the *Hessian* matrix defined in SURF. The result is shown in Equation 2.14.

$$\mathbf{H}(\mathbf{x}) = \mathbf{H} + \frac{\partial \mathbf{H}^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 \mathbf{H}}{\partial^2 \mathbf{x}}\mathbf{x} \tag{2.14}$$

The final location for the extremum $\hat{\mathbf{x}}$ is obtained deriving the Taylor series expansion with respect to $\mathbf{x} = (x, y, \sigma)$ and setting it to 0. Its expression is defined in Equation 2.15.

$$\hat{\mathbf{x}} = -\frac{\partial^2 \mathbf{H}^{-1}}{\partial^2 \mathbf{x}}\frac{\partial \mathbf{H}}{\partial \mathbf{x}} \tag{2.15}$$

E) Orientation assignment

The orientations for the previous keypoints are computed to provide invariance to rotations. First, a region with radius 6σ is selected. Remember that σ is the scale where the keypoint was detected. The

distribution of the *Haar Wavelets* is created in x and y directions and the result is weighted by a gaussian mask centered at the keypoint with radius 2σ.

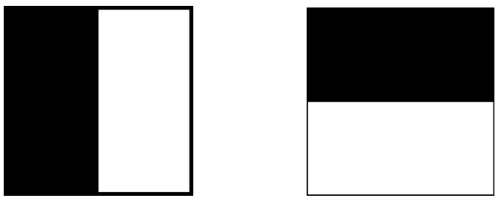


Figure 2.14: Haar Wavelets; x and y directions.

The *Haar Wavelets* are applied in x and y directions and the results are represented in a cartesian coordinate system. The dominant orientation is estimated doing the sum of the responses within a window with angle $\frac{\pi}{3}$ radians (see Figure 2.15). A scan for all possible windows is performed and the direction with the highest magnitude is added as a new parameter to the keypoint.

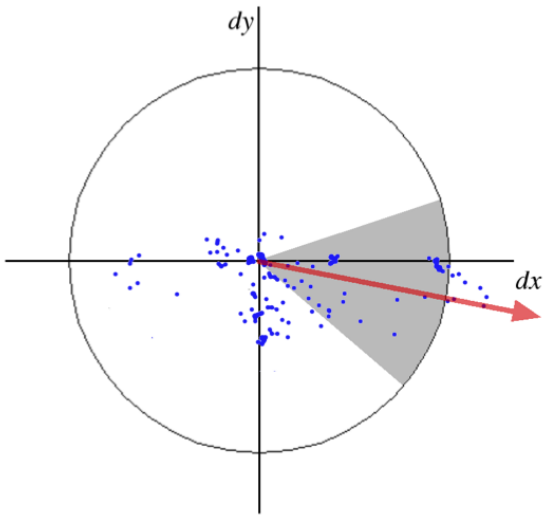


Figure 2.15: Dominant keypoint orientation. Blue points are *Haar Wavaletes* responses in both directions. The dominant direction is obtained as the sum of all the responses within a $\frac{\pi}{3}$-angle window. The response with the highest magnitude is selected. Image taken from [22].

F) Feature extraction to create the descriptor

To extract the parameters defining the region around the keypoint, a square region with size 20σ is selected. This region is divided in 4x4 subregions; that is, the size of each subregion is 5σ. Furthermore, the *Haar Wavelets* responses are calculated for 25 points evenly spaced in each subregion. The response in horizontal direction is denoted as d_x and the response in vertical direction is denoted as d_y (see Figure 2.16).

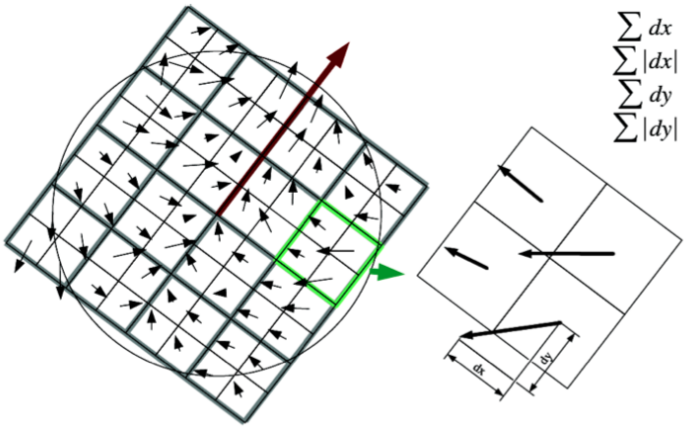


Figure 2.16: Describing the region around the keypoint. The selected region size is 20σ divided in subregions of 4x4 pixels. Each subregion has 5x5 evenly spaced points and the *Haar Wavelet* responses are computed on those points (left). The 2x2 divisions are related with the parameters $\sum dx$, $\sum |dx|$, $\sum dy$ and $\sum |dy|$ calculated for the descriptor. Image taken from [22].

For each of the 16 subregions within the region of interest, a vector with 4 parameters $\mathbf{v} = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$ is calculated. For a given subregion in the image with constant value dx and dy variations will be null. If the image has white and black vertical lines, there will be high variation in x direction. However, the black-white and white-black transition gradients have opposite values, thus $\sum dx$ will be 0 but $\sum |dx|$ will be very high. Finally, all the 4-dimensional vectors for each of the 4x4 subregions are stored providing a descriptor vector with 64 dimensions. The *Haar wavelet* responses are invariant to illumination changes. To achieve invariant to contrast changes the vector is normalized.

2.2. Classification

Machine Learning handles the design and implementation of algorithms that can learn and extract patterns from data. One essential desired property for these methods is the ability to *generalize* and consequently perform well on unseen data. There are different categories depending of the input samples available during training the machine.

- *Supervised learning*: The available samples in the training data consist of two components: the input and its label or output. These algorithms attempt to find a function from inputs to outputs that perform well on unseen inputs. Classification is a classic supervised example where the output belongs to a set of discrete elements.
- *Unsupervised learning*: The available samples in the training data are unlabelled: that is, only the input is provided. The algorithms to perform clustering are a common example of unsupervised learning. These algorithms find properties or patterns in the data and use those properties to cluster the input in different classes.
- *Semi-supervised learning*: These methods combine both labelled and unlabelled samples to generate a function or classifier. Many researchers have shown that the use of unlabelled data with a small set of labelled data can increase considerably the precision in the learning process. Moreover the acquisition of labelled samples requires human being intervention while non labelled samples can be acquired easily.

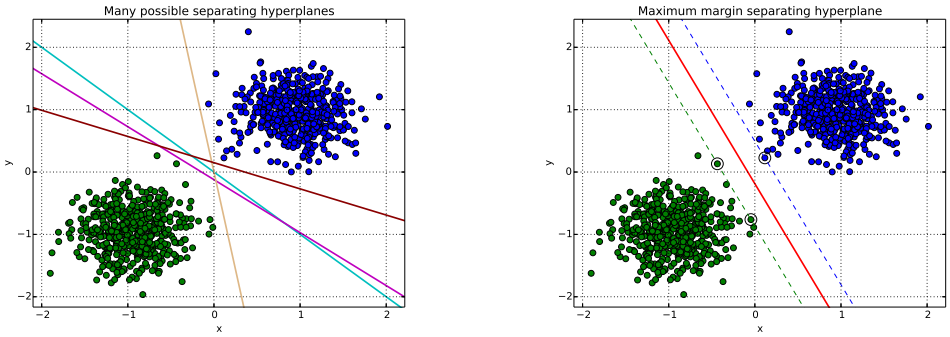
Data mining is a field very related to machine learning. They often use the same methods and can overlap significantly. However the main difference resides in the objective of each one. Machine learning attempts the prediction of new observations using the extracted properties of the training data. In contrast, data mining performs the discovery of unknown properties in the data.

Support Vector Machines (SVMs) are a set of well known supervised methods used for classification and regression. However, there exist some implementations for semi-supervised learning. In the next subsection the classification process will be defined and its achievement using Support Vector Machines (SVMs) explained.

2.2.1. Support Vector Machine for Classification

Given a new sample $\mathbf{x} \in X$ and a finite and discrete set of classes $Y = \{C_i\}_{i=1}^{k}$, classification attempts to assign one class to each sample $\mathbf{x}$. Generally the samples are $\mathbf{x} \in \mathbb{R}^d$ so $\mathbf{x}$ is a d-dimensional vector with real numbers. Using a predictive point of view, the aim of classifying is to find a model or function mapping $X \to Y$ using the training data $\{(x_i, y_i)\}_{i=1}^{n}$. Afterwards, the model will be used to predict the class $C_i \in Y$ for new unseen samples $\mathbf{x}$.

For the sake of clarity, the discussion in this section starts by proposing a classifier that is able to perfectly discriminate two-class linearly separable data. Later on, these two assumptions (linear separability and binary classification) will be dropped, giving rise to a more general setting. When the data is binary and linearly separable it is possible to find a *hyperplane* able to discriminate all the data samples; that is there exists a hyperplane that can split the space so the samples belonging to each class lie in a different region of the space. There are many possible hyperplanes that can separate the data (see Figure 2.17a). It seems reasonable to choose the hyperplane with highest distance within the two classes to generalize better and avoid misclassifications. If this hyperplane exists, it is called maximum margin hyperplane (see Figure 2.17b).



(a) Many possible hyperplanes **(b)** Maximum margin hyperplane

Figure 2.17: Linear Support Vector Machine: (a) many possible hyperplanes; (b) maximum margin hyperplane. The marked samples are denoted as support vectors.

Let us define the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. Any hyperplane can be expressed as in Equation 2.16 where $\mathbf{w}$ is the normal vector to the hyperplane and b is the bias.

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{2.16}$$

If the data is linearly separable it is possible to define two hyperplanes that separates the two classes without any sample lying between both hyperplanes. Furthermore, it is desired to maximize the distance between the two hyperplanes. These two hyperplanes can be defined as in Equation 2.17.

$$\begin{aligned} \mathbf{w}^T\mathbf{x} + b &= 1 \\ \mathbf{w}^T\mathbf{x} + b &= -1 \end{aligned} \tag{2.17}$$

Using geometry, the distance between the hyperplanes is $\frac{2}{\|\mathbf{w}\|}$. Thus to maximize the margin we have to minimize the $\|\mathbf{w}\|$. To prevent samples lying in the margin two constraints are added (see Equation 2.18).

$$\left.\begin{array}{ll} \text{if} & y_i = +1 \rightarrow (\mathbf{w}^T\mathbf{x}_i + b) \geq +1 \\ \text{if} & y_i = -1 \rightarrow (\mathbf{w}^T\mathbf{x}_i + b) \leq -1 \end{array}\right\} \; y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{2.18}$$

The complete optimization problem with the objective function to minimize and the constraints is stated in Equation 2.19.

$$\min_{(\mathbf{w},b)} \|\mathbf{w}\| \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad i = 1..n \tag{2.19}$$

The optimization problem defined in Equation 2.19 is difficult to solve because its dependence of $\|\mathbf{w}\|$ that includes a square root. Fortunately, it is possible to modify the function, replacing $\|\mathbf{w}\|$ by $\frac{1}{2}\|\mathbf{w}\|^2$, without altering the solution. Note that the factor $\frac{1}{2}$ is introduced for convenience. Finally, it can be expressed as a quadratic optimization problem in its primal form as shown in Equation 2.20.

$$\min_{(\mathbf{w},b)} \frac{1}{2}\|\mathbf{w}\|^2 \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad i = 1..n \tag{2.20}$$

Therefore, the classification problem is now solved choosing the hyperplane that maximizes the margin. Nevertheless, the Equation 2.20 does not have a solution if the data is not linearly separable. Intuitively, there are samples that lie in the wrong side of the hyperplane. To allow some misclassifications while maximizing the margin the *Soft Margin* method is used. This method introduces the so called *slack variables*, ξ_i. The variable ξ_i measures the amount of error of the misclassified sample. Thus, if the sample is correctly classified $\xi_i = 0$, otherwise ξ_i is the distance between

the sample $\mathbf{x}_i$ and the hyperplane. The constraints can take into account this improvement by introducing the slack variables as indicated in Equation 2.21.

$$\left.\begin{array}{ll} \text{if} & y_i = +1 \rightarrow (\mathbf{w}^T\mathbf{x}_i + b) \geq +1 - \xi_i \\ \text{if} & y_i = -1 \rightarrow (\mathbf{w}^T\mathbf{x}_i + b) \leq -1 + \xi i \end{array}\right\} \; y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \qquad (2.21)$$

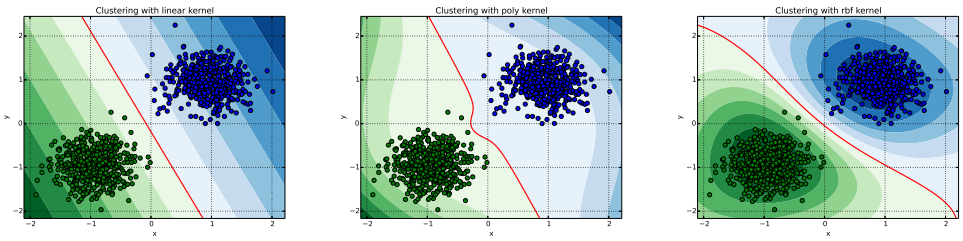
The objective function is also modified adding a term to penalize misclassifications; that is, penalizing the samples where ξ_i is not zero. The optimization problem attempts to find a balanced solution between maximizing the margin and having a small penalty error. If the function considered to model the penalty error is linear, the entire optimization problem is stated in Equation 2.22.

$$\min_{(\mathbf{w},b)} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad s.t. \quad \left\{\begin{array}{rl} y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i & i = 1..n \\ \xi_i \geq 0 & i = 1..n \end{array}\right. \qquad (2.22)$$

So far, the boundary separating the two classes is a line, and thus the classifier is linear. For more complex boundaries the *kernel trick* is applied. In this trick, the training samples $\mathbf{x}_i$ are mapped using a function (not necessary linear) to a new space with higher dimensionality. The boundary found in the high dimensional space is an hyperplane, but can have a non linear shape in the original data space. Some of the most popular *kernels* are shown in Equation 2.23, where γ, r, d are kernel parameters. Subsection 2.2.2 will show how to approach the selection of these parameters. Figure 2.18 illustrates the boundary found when applying different kernel functions to classify the same synthetic training data.

$$\begin{array}{lll} K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j & & \rightarrow \text{Linear} \\ K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma\mathbf{x}_i^T\mathbf{x}_j + r)^d, & \gamma > 0 & \rightarrow \text{Polynomial} \\ K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2), & \gamma > 0 & \rightarrow \text{Radial Basis Function (RBF)} \\ K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\gamma\mathbf{x}_i^T\mathbf{x}_j + r) & & \rightarrow \text{Sigmoid} \end{array} \qquad (2.23)$$

Finally, dropping the binary classification assumption lead us to multiclass classification, where the output set can have more than two classes $Y = \{C_i\}_{i=1}^k$. The generalization applied to the optimization problem stated in Equation 2.22 in order to attempt multiclass classification is based on the work in muticlass kernels done by Cramer K. y Singer Y. [27]. However, this approach is not unique to provide multiclass classification to



(a) Linear kernel **(b)** Polynomial kernel **(c)** RBF kernel

Figure 2.18: Classification example using different kernels: (a) linear kernel; (b) polynomial kernel; (c) radial basis function kernel. Note the difference in the boundary and the distribution of probabilities (contours).

SVMs. In fact, the dominant approach is based on the reduction of the multiclass classification task into several binary classification problems. These methods are grouped in the so called *Error-Correcting-Ouput-Codes* (ECOC) [28]. The simplest ECOC are *One-vs-rest* and *One-vs-one*.

- *One -vs-rest* [2]: This approach uses K binary classifiers, $Y = \{C_1, C_2\}$, where K is the number of total classes. The samples $\mathbf{x}_i$ in the training data can belong to C_1 if $y_i = k$ or C_2 if $y \in \{1..K\} \backslash \{k\}$. The final classification is the one with highest score among all the binary classifiers. It is important to calibrate all the functions to produce scores in a similar range.

- *One-vs-one*: This approach uses $K(K-1)/2$ classifiers, thus one binary classifier for each possible pair of classes is trained. The prediction of each binary classifier assigns a class to the new sample $\mathbf{x}$. Finally the class with more votes is selected.

That concludes the basic theoretical introduction to classifiers and in particular to Support Vector Machines. A thorough description of Support Vector Machine is well explained in [29] and [30].

Structured Output-Support Vector Machines are machine learning algorithms that generalize the previously explained SVMs. The SVMs attempt

[2] *One-vs-rest* is also known as *One-vs-all* in the machine learning community. In this report we will use the first nomenclature since the notation *One-vs-all*, in our opinion inaccurate, implies discriminate samples that belong to the class k with samples that belong to the rest of the classes $1, ..., K$ including k.

to determine the most likely class for a given observation $\mathbf{x}$ and are trained with the training data $\{(x_i, y_i)\}_{i=1}^{n}$, where generally $\mathbf{x} \in \mathbf{R}^d$ and $y \in \mathbf{R}$. Thus, each sample is a d-dimensional vector and the goal is to predict the most probable class (usually a label or number) for this new sample. Is important to highlight that SVMs assume that all classes are independent. On the other hand, SO-SVMs drop that assumption being able to learn relations between different classes. Additionally, the goal is not to predict a label or number but possibly more complex objects, as sequences or graphs. The theory related with those algorithms is far more complicated. And introduction to both SVMs and SO-SVMs and its application to network data is described in [31]. For a detailed SO-SVMs explanation in the context of computer vision see *Structured Learning and Prediction in Computer Vision* [32].

In this project, the results obtained with SVMs are good enough and the ones obtained with SO-SVMs do not improve the performance considerably. The main reasons are the reduced size of the database, the number of different classes used and the dimensionality of the inputs (d=128 for SIFT). Anyway, we recommend to use SO-SVMs for object recognition and classification with multiple views to exploit the proposed graph-based representation of the object and the dependency/relations between classes.

2.2.2. Grid-Search and Cross-Validation

It is rather common within machine learning to deal with problems that depend on one or several parameters that are not in the search space of the learning algorithm. In the SVMs the parameter that is in the search space of the algorithm is the hyperplane $\mathbf{w}$. However, there are values which no straightforward way of choosing them exist. An example of such parameters are $< C, \gamma, d >$ in kernel functions (see Equation 2.23). They are considered *free parameters* and this section explains some techniques to choose their values. The basic idea is to train a model several times with the same data using different combinations of the free parameters. Finally, the parameters that achieved the best result are selected.

Firstly, a search in the possible range of values for those parameters has to be done. It is desirable to generalize the model to perform well in unseen data. Therefore, a very high precision in the training data can be useless, leading to *overfitting*. To search the appropriate values for the related free parameters we use *Grid-Search*. In this approach all the possible

combinations within the feasible values of the free parameters are evaluated. To define those feasible values, a good approach is to create an exponentially growing value sequence.

To compare the performance of the models a score value is needed. The traditional approach, called *Cross-Validation*, splits the data in two sets: training set and evaluation set. The basic idea is to train the model with the training data and evaluate its performance in unseen data; that is, evaluation data. This is a valid approach, however a better use of the training data can be done to obtain a more reliable score. This technique is called *ν-fold Cross-Validation* and can help us to prevent overfitting.

In ν-fold cross-validation the training set is split in ν subsets with the same size. Each of this subsets is evaluated with the model trained using the other $\nu - 1$ subsets. The result of this process is a set of ν scores, one for each fold, called *cross-validation error for fold ν_i* and its average called *cross-validation error*. This error is an estimation of the performance of the model in unseen data, providing that the data represents accurately the real problem domain.

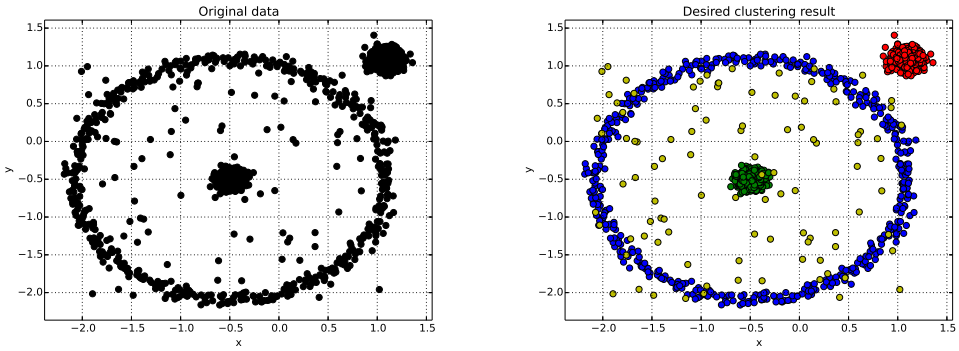
Based on the previously obtained cross validation error, the expected model behaviour can vary in two different ways; the estimate could be optimistic, performing worse in real data. On the other hand, it could perform better in the real data. With the model selection process explained so far, the estimate is almost certainly optimistic. In order to avoid the optimistic estimation introduced with this process, we apply cross-validation one more time. The key idea is to evaluate the performance of the model using a completely previously unseen set of data samples.

Evaluating the models using the evaluation data provides a measure of their performance in real data. The score obtained is defined as *cross-evaluation error*. Finally, the model with smaller cross evaluation error is trained with all the available data: training data and evaluation data.

2.3. Cluster Analysis

Cluster analysis or *clustering* is the task of grouping objects in different sets (*clusters*) having into account their properties and similarities. Thus, objects belonging to the same cluster will have more similarities between them than with objects within other clusters. The similarities considered and its estimation process varies resulting in different families of algorithms. Typical cluster models based on the concept of similarity include: connectivity-based model, density-based models, centroid-based models,

distribution-based models and graph-based models among others. In this project, we use three clustering algorithms, each one based in a different model. The algorithms are explained below and their similarity measure is defined. The results shown for each clustering algorithm will be based on the original synthetic samples represented in Figure 2.19a. The ground truth for these samples is shown in Figure 2.19b. Note the yellow samples are noise.



(a) Synthetic samples **(b)** Ground truth

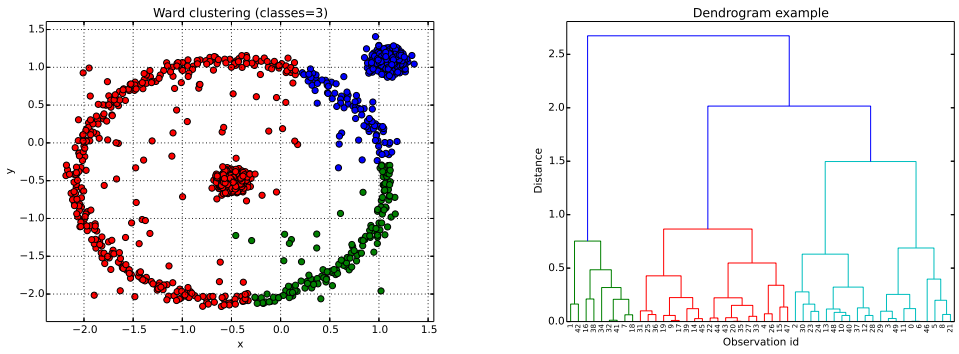
Figure 2.19: Clustering example: (a) Synthetic data; (b) Desired result. The samples represented in yellow are considered as noise.

2.3.1. *Connectivity based clustering*

Connectivity based clustering, also known as *hierarchical clustering*, is based on the core idea of objects being more related to nearby objects than the objects farther away. Thus, the similarity is defined as the distance between two samples. This clustering model does not find a single data but provides a hierarchy of clusters that are merged at certain distances. This hierarchy is represented using *dendograms* (see Figure 2.20b), where the y axis represents the distance where two clusters are merged and the samples are placed in the x axis such that the clusters do not mix. Generally, connectivity based clustering algorithms are divided having into account the strategy in:

- *Agglomerative strategies*: Each sample starts being considered a cluster, consequently there are the same number of clusters than samples. As we go forward the hierarchy the clusters are merged.

- *Divisive strategies*: All the samples belongs to the same unique cluster. As we go forward the hierarchy the clusters are partitioned.



(a) Result obtained using *ward* **(b)** Dendogram

Figure 2.20: Clustering example using *ward*: (a) Result with number of clusters *k=3*; (b) Example of dendogram with 50 samples.

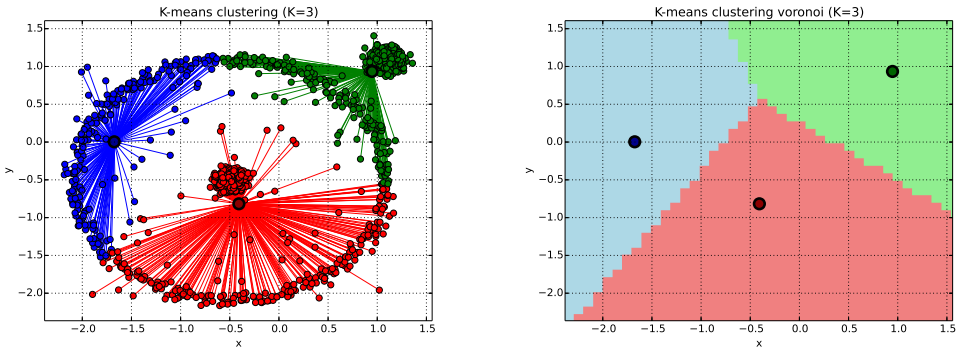
Our selected connectivity based clustering algorithm is *ward* and the result obtained for the original synthetic samples is shown in Figure 2.20a. The main goal of this algorithm is to minimize the total variance within the clusters. Ward follows an agglomerative strategy starting with one cluster for each sample. The initial distances between clusters are the euclidean distances between samples; $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. In each recursion, the pair of clusters that leads to a minimum total variance increment within clusters are merged. This process is repeated until the total number of clusters indicated by the user is reached.

2.3.2. *Centroid based clustering*

In this clustering model the clusters are defined by one single representative sample called *centroid*. There exist different algorithms depending of the centroid selection/creation methods. *k-means* calculate the centroid as the average of all samples assigned to one cluster, consequently the centroid is not necessarily one sample belonging to the data. On the other hand *k-medoids* restricts the centroid to be one of the samples in the data. Instead of the average calculated by *k-means*, the algorithm *k-medians* finds the median. All these algorithms have a main drawback: the number of total clusters k has to be specified previously. Their tendency to create

clusters with similar size or the influence of outliers are other drawbacks inherent to these methods.

Among the algorithms mentioned previously we have selected *k-means*. The goal of these algorithms is to separate n samples in k clusters (see Figure 2.21a). Therefore a partition of the data space in cells called *Voronoi diagram* is obtained (see Figure 2.21b).



(a) Result obtained using *k-means* **(b)** *Voronoi* diagram

Figure 2.21: Clustering example using *k-means*: (a) Result with number of clusters *k=3*; (b) *Voronoi* diagram. The centroids are marked with bigger size.

Given a set of samples $\mathbf{O} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where each sample is a d-dimensional vector and a set of partitions $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_k\}$, the objective is to minimize the square error within clusters as indicated in Equation 2.24. Note that μ_i is the average of the samples assigned to cluster S_i, commonly denoted as centroid.

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|^2 \tag{2.24}$$

This optimization problem is NP-complete and due to efficiency reasons approximate solutions are searched. The *Lloyd's* algorithm [33] is a commonly used approximation that finds a local minima. This is an iterative algorithm and has three well differentiated steps: initialization step, assignment step and update step.

1) *Initialization step*: This step determine the initial position of the centroids. The initial positions (*seeds*) are critical and affect considerably the solution obtained. For this reason the algorithm is executed several

times with different initializations and the one with the best result is chosen. Some possible initializations are:

- *Forgy's method*: k samples are selected randomly as centroids. This initialization tends to find initial centroids well separated.
- *Randomly partition*: One within the k possible classes is assigned randomly to each observation. Afterwards, the update step calculates the initial positions or seeds.
- *kmeans++*: The basic idea is to find well separated centroids within the data space. This algorithm is less random than the others and leads to a considerably improvement in the final k-means error. Though this initialization takes more computational time, the k-means algorithm convergence is faster reducing the computational time. For more information see *k-means++: the advantages of careful seeding* [34].

2) *Assignment step*: Each sample is assigned to a cluster which average μ_i produces the minimum square error within the cluster. As the quadratic error $\|\mathbf{x}_j - \mu_i\|^2$ is the euclidean distance, intuitively the closest centroid is being chosen.

3) *Update step*: The new centroids μ_i are calculated as the average of all samples belonging to cluster S_i as expressed in Equation 2.25.

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j \tag{2.25}$$

2.3.3. *Density based clustering*

The clusters in this clustering model are defined as densely populated regions in the data space. The samples placed between two densely populated regions are considered as noise. The most familiar density-based clustering method is *Density-based Spatial Clustering of Applications with Noise* (DBSCAN). This method connects samples situated within a certain limit distance. Those samples have to fulfil also a density criteria. Some of the benefits of DBSCAN are:

- It is not necessary to set the number of clusters previously.
- The clusters can have arbitrary shapes.

- There exist a notion of noisy samples (*outliers*).

Some of the drawbacks exhibited in previously clustering methods have been solved. Despite all the benefits, DBSCAN has a clear weakness: it can not group samples in clusters with very different densities. To solve this problem Mihael Ankerst et al [35] presented a generalization of the algorithm called *Ordering Points To Identify the Cluster Structure* (OPTICS). First, the samples are arranged such close neighbouring samples are nearby in the sorted vector. Additionally, the distance that should be accepted to keep two samples in the same cluster is stored. This distance is called *reachability distance*.

Let us define the maximum ratio ε and the minimum number of samples to form a cluster $MinPts$. The sample $\mathbf{p}$ is considered central if exists at least $MinPts$ in its neighbourhood $N_\varepsilon(\mathbf{p})$. All samples are assigned a *core distance* defined by Equation 2.26, where $\mathbf{c}$ is the *MinPts*-th closest sample.

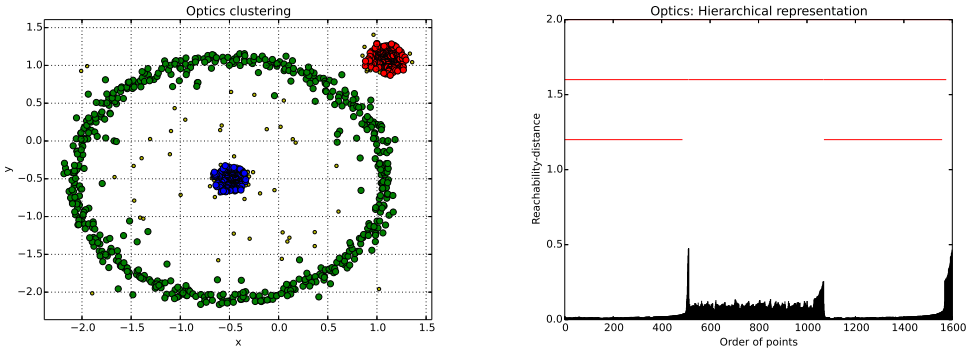
$$\text{core-dist}_{\varepsilon,MinPts}(\mathbf{p}) = \begin{cases} None & \text{si } |N_\varepsilon(\mathbf{p})| < \text{MinPts} \\ \|\mathbf{p}-\mathbf{c}\| & \text{otherwise} \end{cases} \tag{2.26}$$

Finally, we define the *reachability distance* between two samples $\mathbf{p}$ and $\mathbf{o}$ as shown in Equation 2.27, where $d_1 = \text{core-dist}_{\varepsilon,MinPts}(\mathbf{o})$ and $d_2 = \|\mathbf{p}-\mathbf{o}\|$.

$$\text{reach-dist}_{\varepsilon,MinPts}(\mathbf{p},\mathbf{o}) = \begin{cases} None & \text{si } |N_\varepsilon(\mathbf{o})| < \text{MinPts} \\ max(d_1,d_2) & \text{otherwise} \end{cases} \tag{2.27}$$

Once the *reachability distance* is calculated for all samples, they can be represented as an special type of dendogram. The x axis represents the sorted samples and the y axis shows its *reachability distance*. Usually, clusters have low reachability distance within their closest neighbours and consequently clusters in this representation are shown as valleys (see Figure 2.22b). The deeper the valley the denser the cluster. In Figure 2.22a the clustering result obtained for the original synthetic example is shown.

There are many ways to evaluate the quality of the clusters obtained. *Internal evaluation* techniques use the previously clustered samples to measure the quality. They usually tends to assign better score to those algorithms that form clusters with high similarity within cluster's samples and low similarity between clusters. If the samples used to measure the quality are not in the data then they are called *external evaluation* techniques.



(a) Result obtained using *optics*

(b) Dendogram

Figure 2.22: Clustering example using *optics*: (a) Clustering result without specifying the number of clusters; (b) Samples sorted by their *reachability distance*, where valleys represent clusters. Note the presence of *outliers* marked in yellow and with smaller size.

The effectiveness of this second evaluation technique has been discussed recently. The drawback is that for the new samples the ground truth is required. Due to the complexity to evaluate the clustering algorithms in non synthetic data, in our case real images, we don't use any mathematical function but our own criteria following the quote: *"Clustering is in the eye of the beholder"* [36].

2.4. Research questions

There are many questions to answer in this open-field. Some of the possible questions that can be addressed during this project are stated below:

- How much transformation can the algorithms support? Are the algorithms enough robust for recognition?
- Does the difference between image resolutions affect the algorithm's performance?
- Are the algorithms valid for real-time recognition systems?
- Are the algorithms valid for big-size databases?
- Are the algorithms valid for objects with multiple canonical views?

- How many canonical views are needed for object classification?
- How to estimate the minimum number of views of the same object needed to the performance without an exaggerate increase in the retrieval time from the database?
- How the way human agents pick views affects the performance?

3

GRAPH-BASED OBJECT REPRESENTATION APPROACH

Real objects can be divided in two categories attending to the disposition of the elements they are conformed by. If all the elements can be captured in a single image or picture, they are considered one single canonical view objects (i.e. book covers or paintings). Due to the 3D structure of the objects, its elements are usually distributed in different views, also called multi-view objects with several canonical views. This inherent feature increases the developing and designing complexity of object recognition systems. In addition, the size of the database will grow considerably when storing different views of the objects. Comparing two images in order to recognize the object may not be sufficient because they can be representing different views.

This chapter will explain the approach proposed to design multi-view object recognition systems. Previous work done by the author of this report in objects with a single canonical view indicate that the descriptor SIFT is suitable to address this problem.

Section 3.1 will explain some techniques developed by other researches to approach this problem and will show our proposal: a graph-based representation of the object where the nodes identify its main physical elements. In addition, the developing process and its steps will be defined individually. In Section 3.2 we will present the interface developed to create the database and its structure. In this project recursive cluster analysis is used to group the keypoints and detect the main elements in the object. This technique and some examples are shown in Section 3.3. Once the elements are detected, a previously trained classifier is used to predict the most probable class for each node. Support Vector Machines (SVMs) are the classifiers selected for this project. An improvement called *Weighted-Support Vector Machines* (w-SVMs) is briefly commented and illustrated with examples. To close the chapter, the different criteria used

to predict the class of each cluster based on the individual classification of its observations is explained in Section 3.5 and some examples are shown.

3.1. Literature review and Proposed Approach

Clearly, 3D object classification and recognition based on 2D images is a very generic problem statement. There are many methods to approach this task and most of them have in common the use of several canonical views. Some presented solutions by other researches are:

A) Brute force

The easiest approach is to store several views of the same object in the database and perform the matching between the input image and every image in the database, retrieving the one with highest score. The benefits; an easy implementation and similar results to those obtained for single-view object recognition. The drawbacks; the computational time.

The average matching time using SIFT descriptor in a reduced size image (400x262) is about 6 *ms*. It seems negligible but this approach has two important weaknesses:

- The product images provided by the companies are high quality so their size is large.
- Having several views of the same object increases the size of the database drastically.

Hence, the computational time needed when working with large size databases is high. Companies as *Google* ® solve the problem storing the images ordered in different servers and using the distributed computing paradigm.

B) Shape-based object recognition

The approach proposed by Cristopher M. and Benjamin B. [37] creates a graph with the possible views of the object. Each node of the graph represents one view and they are connected sequentially depending of the position in which they were obtained. The set of views for each object is discrete and the recognition of the object is done based on its shape. For a new image, the most similar views are sorted by similitude. The way to calculate the similitude can vary; curvature-based matching and graph-based matching are possible approaches. For a more

detailed description see *A Similarity-Based Aspect-Graph Approach to 3D Object Recognition* [37].

C) Patch extraction on a 3D model

In this method, presented by Rothganger et al [38], the 3D shape of the object based on the different available views is computed. Afterwards, the surface is divided in small and invariant patches where the detectors and descriptors are applied. The keypoints position, the descriptor vectors and additional information about the three-dimensional relations between the patches are stored. For a detailed explanation see *Toward True 3D Object Recognition* [39].

D) Vocabulary of visual features (*Bag of words*)

This approach groups the previously extracted features, in our case keypoints and their description, in a vocabulary of visual words called *bag-of-words*. To create this vocabulary a tree-based structure was developed by Nister and Stewenius [40] and evaluated by Tomasik, Thiha and Turnbull [41]. This tree can be built using k-means. The classification of new images can be done using k-NN or a modified version where each vote is weighted based on the proximity with the reference image.

Taking into account all the previous proposed approaches to solve the problem, their benefits and their drawbacks we have decided to propose our approach. This approach is based on a graph-based representation of the object and the use of structured learning and prediction algorithms. The process is divided in three different steps well separated. These steps are:

- *BBDD*: We will use a database with different canonical views of the same object. Due to the shortage of databases with SIFT keypoints and descriptors, we will implement our own interface to make its creation easier. Section 3.2 will explain the use, structure and implementation of the database.

- *Object representation*: The approach is based on the *bag-of-words* technique. Our own implementation is coded to, not just cluster SIFT vectors by its properties, but to take into account their position in the image and detect the presence of outliers. This process is called

clustering and a set of clusters will be obtained. These clusters, also known as nodes, are related directly with the physical properties of the object (sewing, button, cord, ...) and are joined using arcs to obtain a graph-based representation of the object.

- *Object prediction*: Instead of considering the object as a whole, we will tray to predict each of the features or physical elements that conforms the object. Intuitively, one label has to be assigned to each node. Firstly, Support Vector Machines (SVMs) are used to predict the labels for each node independently. Afterwards, Structure Output-Support Vector Machines (SO-SVMs) are introduced. Due to the graph-based representation of the object and the SO-SVMs it is possible to take into account the relations between labels. For example, buckle and cord are not independent; that is, given a buckle the probability of finding a cord is smaller and vice versa.

The diagram used to develop the proposed multi-view object recognition system (see Figure 3.1) is divided in two main blocks: *Model construction*, where the database is created and the classifier is trained and *Input image prediction*, where the physical elements that appear in a new image are detected and predicted to recognize the object.

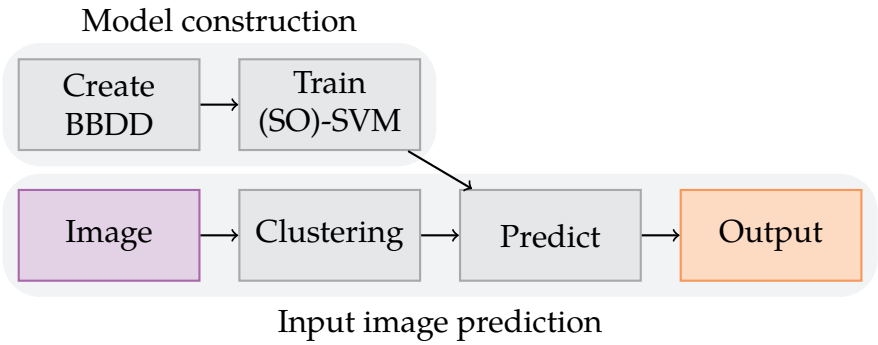


Figure 3.1: Multi-view object recognition diagram with a graph-based representation of the object and prediction using a classifier.

The main goal and the basic functionality of the steps shown in the diagram of Figure 3.1 are briefly explained below:

A) Database creation

For a set of images with different views of the objects their keypoints and descriptors are computed. Afterwards, these keypoints are selected and a label representing the class they belong to is assigned. The label can be introduced manually by the user (i.e. sewing) or can be automatically generated (i.e. a number or ID).

B) Training the classifier

When the database has been created it is possible to train a classifier with the previously stored data. This training process takes time but once the model has been defined the prediction step is computed efficiently.

C) Cluster analysis (*Clustering*)

The amount of keypoints detected in an image can be huge. Thus, the keypoints are grouped in different clusters or nodes. Each node represents a physical feature or element present in the object.

D) Prediction

Given a set of clusters defining the main physical elements of the object and the previously trained model, the most probable class or label is predicted for each of the clusters.

In the following sections the steps represented in the diagram (see Figure 3.1) are explained and their related background theory described.

3.2. Databases

Finding a suitable database to evaluate the performance of a particular design is not always an easy task since the database has to fulfil certain requirements. In this project, the specifications are very detailed. For an image, its keypoints, their associate description and the clusters that have been found are stored. It is interesting to start with a reduced size database labelled by us to facilitate and improve the interpretation extracted from the results. For these reasons, we decided to create our own database. To make this task fun and easy we have developed an intuitive graphic user interface.

This interface (see images in Appendix B) provides elements to shift within the set of images to label; *Prev* button and *Next* button to move among the images sequentially and the bottom bar to select the desired

image manually. Different display options to select the elements to show are available for the user: keypoints, centroids, labels and background color. The labelling of the keypoints can be done in two different ways:

- *Centroid mode*: In this first mode, the algorithm computes the clusters automatically. In addition, for each cluster the centroid is calculated by the average of all its assigned keypoints. Then the user can select one or more clusters by clicking in their centroid. Note that the closest centroid to the mouse position is shown and highlighted automatically (see Figure B.1a). Once the label is introduced for the centroids, it is propagated to all their associated keypoints. This method does not allow an exhaustive keypoint selection and is dependent of the clustering algorithm.

- *Lasso mode*: This mode allows an exhaustive selection of the keypoints. Previously labelled keypoints are displayed with different colors. The color is different depending on the cluster to which they were assigned. The points that lie within the area selected by the user (see Figura B.1b) are highlighted in white and the rest are transparent. This method allows to group and label the keypoints as they are perceived by the user.

Once we are able to select the keypoints and assign them the corresponding label, it is necessary to define the format an structure used to store all the information. In this project the format used to store the data is JSON [1] (*JavaScript Object Notation*). JSON is a language-independent storing/sharing data format with parsers available for many languages. The benefits of this format are many: it is easy for machines to analyze/generate JSON files and it can be read/modified/created by humans because it is plain text. For each image a JSON file is generated providing another advantage: the creation of a large database can be done by merging smaller databases. Physically, this involves placing all the files in the same directory. A brief example with the JSON structure used in our files is shown in Listing 3.1.

[1] Link JSON: http://www.json.org/

Listing 3.1: JSON sample

```
{"'image_path":"./database/shoes-db/sample.png",
 "image_type":"shoe",
 "total_clusters":1,
 "total_keypoints":2,
 "cluster_dict":{
                  "(2,4)":{
                           "id":1,
                           "label":"button",
                           "keypoints_ids": [1, 2]}
                },
 "keypoints_dict":{
                  "(2,5)":{
                           "id":1,
                           "sift":[22.0, 7.0, ..., 1.0]},
                  "(2,3)":{
                           "id":2,
                           "sift":[12.0, 0.0, ..., 21.0]}
                },
}
```

The graphic interface and the data structure presented previously is used to create the database. The information stored in the database can be used in many different ways, some of them are explained below.

- The graphic interface is used to manually label the keypoints. It is not necessary to label each keypoint individually; that is, the keypoints belonging to a previously computed cluster or those lying in a region selected by the user can be labelled together. As a result the labels are human-understandable (i.e. sewing or cord).

- Labelling all the data available in a database is a tough task. Using the previous set of labelled keypoints, it is possible to propagate the labels (i.e. using *labelling propagation* techniques) to the rest of the data. Hence, the complete database is labelled using a set of human-understandable labels.

- The human-understandable labels may not be necessary. For example, if the output only needs to retrieve the most similar results, there is no need to show the label for each cluster. In this case, automatic labels can be assigned using clustering methods directly on the whole set of keypoints available in the database.

- The previous techniques are applied directly on the set of available keypoints. Thanks to the cluster analysis computed for each image,

the database have also information about the clusters in each image. Thus, defining a representative sample for the cluster (i.e. centroid), the methods can be applied on clusters.

In this project the objects to classify/recognize are shoes. The first database is composed by generic images of shoes taken from different views. It is called generic because the same number of canonical views is not required for each shoe. Intuitively, not all the views for an object are equally important and there could even be views with no information at all. To perform this experiment a second database with five different views for each object is created. The images have been downloaded from *dbshoes* [2] and the five available views are shown in Figure 3.2.

(a) View 0 **(b)** View 1 **(c)** View 2 **(d)** View 3 **(e)** View 4

Figure 3.2: Available views in the second database. These views are used in following experiments to determine their relevance.

3.3. Recursive Cluster Analysis

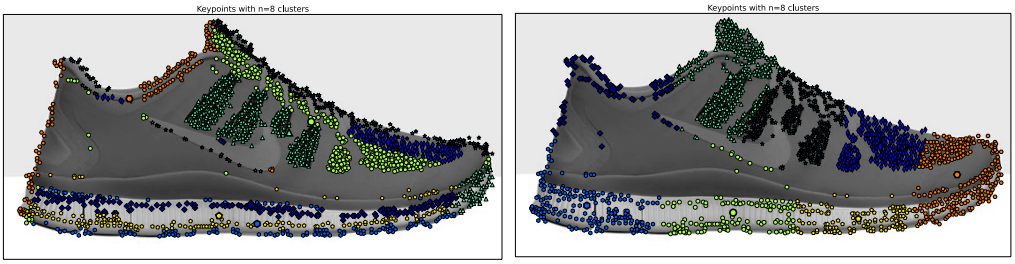
The goal of this clustering step is to group the keypoints in clusters representing the main physical components of the object. This approach is called *bag-of-words*. So far, we have shown examples of clustering algorithms with synthetic data representing 2D space positions $\mathbf{x} = (x, y)$. However, in our implementation the dimensionality of the samples is higher: the position of the keypoint and its description. For a given keypoint described using SIFT we have its position in the image $\mathbf{p} = (x, y)$ and its descriptor $\mathbf{v} = \{v_1, ..., v_{128}\}$. Thus, the sample is $\mathbf{x} = \{\mathbf{v}, \mathbf{p}\}$.

Recursive clustering is used to group keypoints representing physical elements of the input object. It is possible to apply clustering algorithms on the different components of the sample $\mathbf{x} = \{\mathbf{v}, \mathbf{p}\}$. The possible clustering algorithms in each recursion are:

[2] Link dbshoes: http://www.dbshoes.net/

- *k-means* applied to the position, the descriptor and/or both.
- *ward* applied to the position, the descriptor and/or both.
- *optics* applied to the position.

The key idea of taking into account the position $\mathbf{p}$, and not just the descriptor, is based on the assumption that keypoints making up an element lie on the same region in the space. This fact together with the variety of possible clustering algorithms are different to the approach described in [40]. If the clustering algorithms are applied to the descriptor vector the keypoints are grouped based on their surrounding region. On the other hand, if they are applied to the position they will be clustered by proximity (see Figura 3.3b). Similarly happens when applying ward.



(a) *k-means* applied to $\mathbf{v}$ **(b)** *k-means* applied to $\mathbf{p}$

Figure 3.3: Clustering examples using *k-means*: (a) applied to the descriptor $\mathbf{v}$; (b) applied to the position $\mathbf{p}$. The number of clusters in both representations is $k = 8$.

Recursive cluster analysis can be represented as a graph structure, commonly a tree (see Figure 3.4). The whole set of available samples, defined by the node *data* in the figure, is divided in several clusters recursively using one of the methods explained before. Let us define the nomenclature for each node $A^i_{\mathbf{d}}$, where A represents the algorithm (k for k-means, w for ward and o for optics), d is the dimension ($\mathbf{v}$ for the descriptor vector and $\mathbf{p}$ for the position) and i is the i-th cluster obtained. The amount of clusters obtained is previously specified in k-means and ward (K in the figure) and automatically determined in optics (N in the figure). Clearly, when the amount of recursions increases the number of clusters increases as well. Consequently, the total number of keypoints within each cluster is reduced. Visually it means that a physical component of the object (i.e. sewing, button, ...) is represented by several clusters. A stop condition

is used to guarantee the proper functioning and result of the recursive cluster analysis. In the code, the minimum number of keypoints within a cluster to be divided is set.

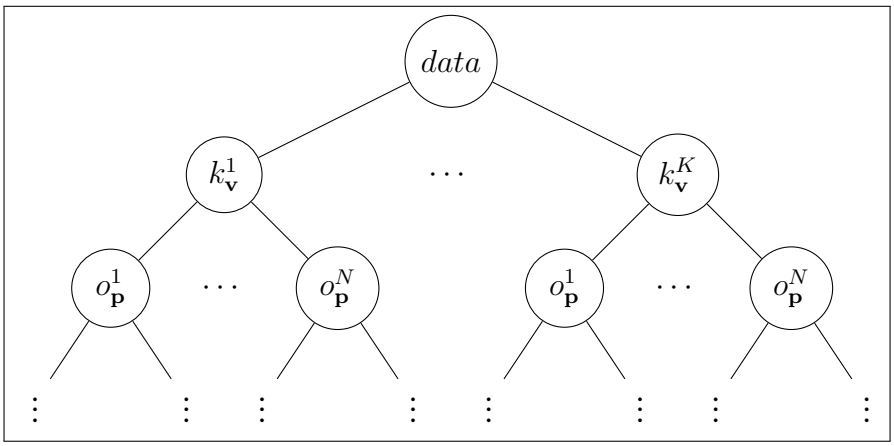
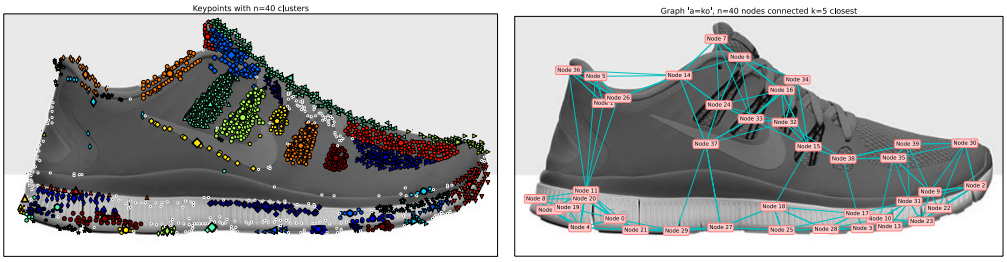


Figure 3.4: Recursive cluster analysis representation. The nomenclature used in the nodes is $A^i_{\mathbf{d}}$, where A represents the algorithm (k:k-means, w:ward and o:optics), d is the dimension (**v**:descriptor vector and **p**:position) and i is the i-th cluster obtained (K:manually set, N:automatically set).

The representation in Figure 3.5a has been created using recursively k-means over **v** and optics over **p**. Intuitively, we are grouping keypoints describing similar regions first (see green cluster in Figure 3.3a). Next, we group the samples within previously obtained clusters by density extracting the different physical objects (see corresponding clusters in Figure 3.5a). Once the different elements that make up the object are obtained it can be represented using a graph structure. The set of clusters obtained are the nodes of the graph. There are different criteria for creating arcs between a pair of nodes, such as joining one node with the k closest nodes or with those ones which distance is smaller than a threshold distance. The resulting graph is shown in Figure 3.5b, where each node is connected with its $k = 5$ closest nodes.

Finally, we have to predict the class each node belongs to. In that way we will be able to identify the elements that make up the object and retrieve from our database the objects with more similarities. Note that the graph-based representation provides many benefits. Firstly, the mentioned representation allows to capture relations between the different elements that make up the object. It allows to model the "strength" between two nodes just giving a value to the arc that joins them. To define this value



(a) Recursive clustering **(b)** Graph-based representation

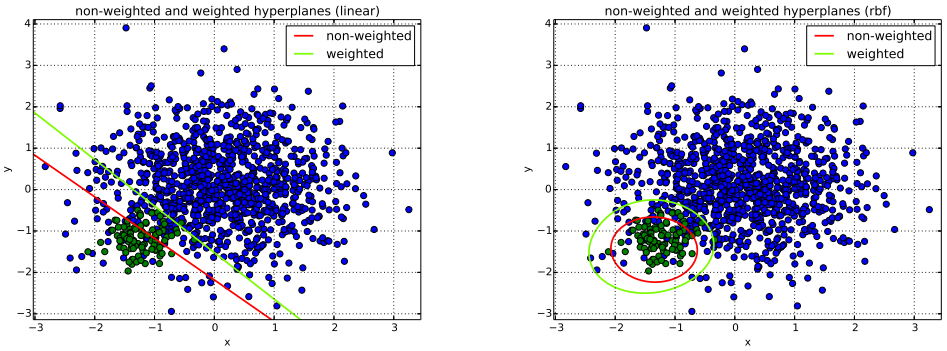
Figure 3.5: Recursive clustering example and graph-based representation: (a) Firstly *k-means* ($k = 8$) is applied to descriptor $\mathbf{v}$. Next for each obtained cluster *optics* is applied to the position $\mathbf{p}$. Keypoints marked in white are *outliers*; (b) Graph-based representation for the previous clustering result.

we can use the euclidean distance between nodes or more complex measures based on graphs. Finally, we can have information about the shape of the object and use structure learning and prediction algorithms. Appendix C shows the recursive clustering result in different objects (in our case shoes). The clusters representing the physical elements are shown and also the graph-based representation is exhibited to clarify its previous mentioned benefits.

3.4. Classifier Training and Selection

Given a database and a new input image, the goal is to distinguish the different elements making up the object in the image. In this context machine learning methods, in particular Support Vector Machines (SVMs), for classification are used. Let us consider a new sample $\mathbf{x} \in X$ and a finite and discrete set of classes $Y = \{C_i\}_{i=1}^{k}$, classification attempts to assign one class to each sample $\mathbf{x}$. Generally, the samples are $\mathbf{x} \in \mathbb{R}^d$ so $\mathbf{x}$ is a d-dimensional vector with real numbers. Using a predictive point of view, the aim of classifying is to find a model or function mapping $X \to Y$ using the training data $\{(x_i, y_i)\}_{i=1}^{n}$. Afterwards the model will be used to predict the class $C_i \in Y$ for new unseen samples $\mathbf{x}$.

In the set of classes $Y = \{C_i\}_{i=1}^{k}$, they are not equally supported; that is, the number of samples available for each class in the database is different. Usually, this property is inherent to the databases and can lead to misclassifications. An example with synthetic data is shown in Figure 3.6.



(a) Linear weighted-class SVM **(b)** RBF weighted-class SVM

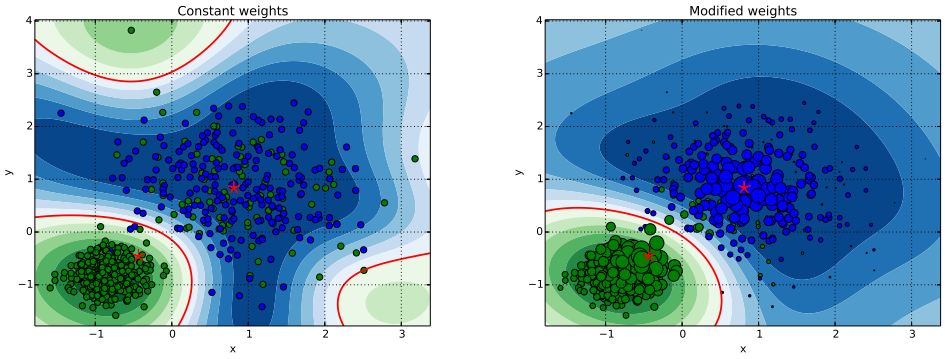
Figure 3.6: Example weighted-class *Support Vector Machine*: (a) linear kernel; (b) radial basis function kernel. The relation between classes is 1:10.

The data is divided in two classes; blue class with a large amount of samples and green class with only a few samples. In this particular example, there are 1000 samples supporting blue class and 100 samples supporting green class. Therefore, the relation between both classes is 1:10. The red line represents the separating hyperplane with uniform weights within classes. This hyperplane stands very close to the center of the green cluster and provides a bad separation between classes. Let us recall the optimization problem solved in Support Vector Machines for binary non-linearly separable classes stated in Equation 3.1.

$$\min_{(\mathbf{w},b)} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad s.t. \quad \begin{cases} y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i & i = 1..n \\ \xi_i \geq 0 & i = 1..n \end{cases} \tag{3.1}$$

The penalty term $C\sum_{i=1}^{n}\xi_i$ in the objective function can be used to handle the non-equal support for the classes in the database; this is called *Weighted-Support Vector Machines* (w-SVMs). The green line in Figure 3.6 represents the hyperplane obtained with weights defined inversely proportional to the amount of samples in the class. These weights are an indicator of the amount of error given a misclassification. Thus, the penalty term obtained in a misclassified green sample is higher than the penalty term obtained in a misclassified blue sample, provided that both lie within the same distance from the current hyperplane; that is, ξ_i is equal. The result is a hyperplane that has been pushed forward due to the importance of classifying green samples correctly to keep the penalty term low.

Another problem to tackle is the presence of *outliers* within the samples in the database. In our database the outliers can appear for many reasons; for example, the labelled clusters are composed by non-uniform samples or the labelling propagation procedure is not totally correct. There are many algorithms to detect and remove outliers, but instead of adding a new step to the recognition process, the Weighted-Support Vector Machines can be used to handle this problem. An example with synthetic data is shown in Figure 3.7. The data is divided in two classes with the same amount of noise. The centroids are represented with a red star mark. The red line in Figure 3.7a represents the separating hyperplane with uniform weights within samples. The hyperplane obtained is very complex and some regions are misclassified due to the presence of outliers (i.e. green area at the top of the figure). The green line in Figure 3.7b represents the hyperplane obtained with weights defined inversely proportional to the distance between the sample and the centroid. These weights are represented with the size of each sample. The result is a more general hyperplane that seems to generalize better.



(a) RBF non-weighted SVM **(b)** RBF weighted-sample SVM

Figure 3.7: Example weighted-class *Support Vector Machine*: (a) linear kernel; (b) radial basis function kernel. The relation between classes is 1:10. The amount of noise is the 30 % of the available samples.

The free parameters for the kernel functions used in the Support Vector Machines are estimated using Grid-Search and Cross-Validation. This step provides a model to perform classification for new unseen data.

3.5. Graph-nodes Prediction

In the previous section we obtained clusters, formed by many keypoints, representing the most representative physical elements of the object. The next step is to identify these elements by predicting their most probable class. For this purpose, the model with best result within the *Grid-Search* and *Cross-Validation* process will be used. Considering a cluster with n samples or keypoints the two prediction schemes implemented are:

- Calculate the centroid for the n samples and predict the most probable class for the centroid. The class assigned to the cluster will be the same that we obtained for the centroid.
- Predict for each of the n samples the most probable class and assign to the cluster the one with more votes.

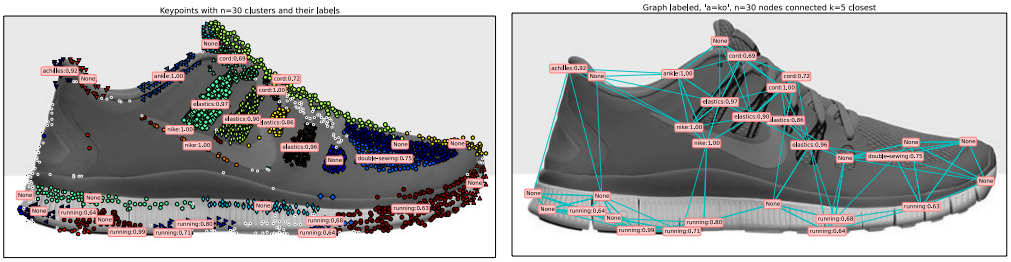
The predicted class C_p obtained for the cluster could not be reliable. There are two main reasons for this drawback. First, high variance within the samples of a cluster will lead us to error. The cause is that the centroid is not representing the samples accurately or the existence of many different labelled classes within the cluster samples. Second, the absence of samples representing that class in the training data. To measure the reliability in both scenarios a probability score is used. Setting a threshold value will ensure no uncertainty in the labelled clusters.

Let us define the probability of a sample $\mathbf{x}_i$ belonging to the cluster C_p as p_{ip}. Thus, the final score is the average as represented in Equation 3.2, where n is the number of samples in the cluster.

$$score = \frac{1}{n}\sum_{i=1}^{n} p_{ip} \tag{3.2}$$

Hence, it is necessary to have homogeneous clusters with high probabilities p_{ip} to obtain a global score that exceeds the threshold. Clusters belonging to a non previous trained class or with high variety within the predicted classes will have low score. Figure 3.8a shows the clusters labelled and the final predicted probability. All the labels assigned with probabilities lower than 0.6 are useless and its value is set to *"None"*. Figure 3.8b shows the graph-based representation with the nodes labelled.

Finally, having a system able to assign labels for each node within the graph, the criteria to search, classify or recognize a new object is up to the



(a) Recursive clustering labelled **(b)** Graph-based representation labelled

Figure 3.8: Labelled recursive clustering example and graph-based representation: (a) Firstly *k-means* ($k = 8$) is applied to descriptor $\mathbf{v}$. Next, for each obtained cluster *optics* is applied to the position $\mathbf{p}$. Keypoints marked in white are *outliers*; (b) Graph-based representation for the previous clustering result. Note that nodes are labelled and the *"None"* label denotes nodes with a probability lower than the threshold, in this case 0.6.

developer. Given the graph-based representation we suggest to use their elements and features (nodes, arc and structure) to calculate the similarities and make decisions.

4

EXPERIMENTS AND RESULTS

To obtain good performance, in computational time and recognition quality, the different feature extraction algorithms are evaluated in this chapter. Furthermore, once their main benefits and drawbacks are assessed, the algorithm that best suits the problem or performs generally better can be selected.

In Section 4.1 the experiments are focused in single view object recognition. The properties and robustness of the feature extraction algorithms are tested within the experiments. First, the section starts comparing the number of detected keypoints depending of the algorithm in Subsection 4.1.1. Next, the amount and percentage of correct matches, to measure the robustness of the algorithms against transformations, are shown in Subsection 4.1.2. In Subsection 4.1.3 the computational time for each recognition step (detection, description and matching) is briefly analyzed. For all the previously explained tests the *Lena* image is used. Finally, an evaluation of the behaviour of the algorithms in different databases is carried on in Subsection 4.1.4. With these experiments a better understanding of the algorithms for single view canonical objects is obtained.

Section 4.2 is focused in the graph-based representation approach proposed to attempt multi-view object recognition. Let us recall the use of a classifier to predict the most probable class for the physical elements in the input object. To select the parameters of the model (classifier) that provide best accuracy Grid-Search and Cross-Validation techniques are used. The model selection process and the accuracies obtained are explained graphically in Subsection 4.2.1. Recall that the classifiers used in this project are the Support Vector Machines (SVMs). The relevance of the different views for recognition is studied in Subsection 4.2.2, where the particular objects considered are shoes. Finally, some predicted examples are shown in Subsection 4.2.3

4.1. Feature Extraction Analysis

Feature extraction techniques are composed by two main blocks: detection and description. In this section, some detection and description algorithms and their properties are analyzed. The experiments, conclusions and decisions are also explained.

4.1.1. Detected keypoints

The first block in the feature extraction process attempts to detect the relevant keypoints in an image. Thus, the first parameter to analyze is the amount of keypoints detected. In Figure 4.1 the number of detected keypoints are shown for some detectors and images. In particular, the image used is the same (*Lena* image) but its size varies. The image sizes are 64x64, 128x128, 256x256 and 512x512 pixels.

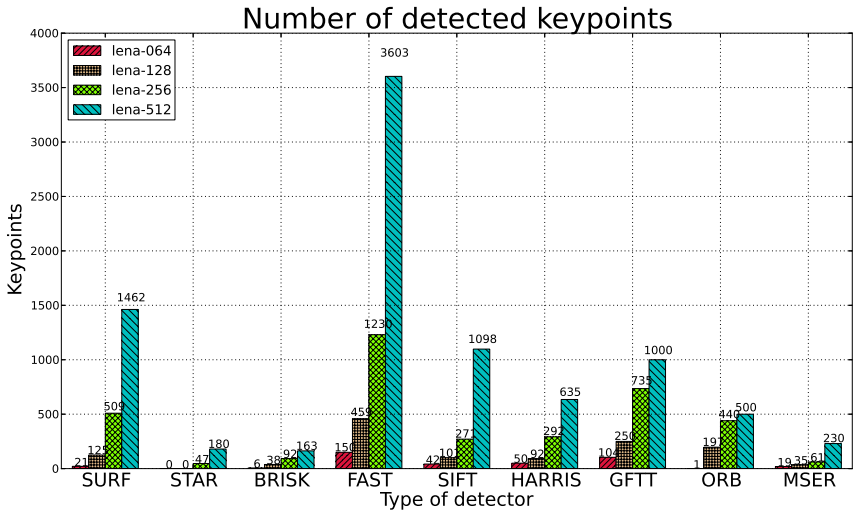


Figure 4.1: Amount of keypoints detected for each detection algorithm. The sizes used for the square *Lena* image are 64, 128, 256 y 512.

The algorithm with the highest amount of detected keypoints is FAST, while some algorithms perform poorly, detecting only a few points, in images with reduced size (i.e. STAR, BRISK or ORB). In this first step, it is preferred to have a large amount of keypoints (they can be filtered in further steps) than have an insufficient set of detected keypoints for recognition. However, the larger the set of detected keypoints the higher the computational time needed to describe all of them. In this project the

algorithms considered are FAST, SIFT and SURF. Any combination of detector and descriptor is feasible and thus the notation used will be *detector-descriptor*.

4.1.2. Robustness against transformations

The robustness of the feature extraction algorithms against transformations in the input image is an important factor to measure their quality. In following experiments, the transformed *Lena* image and the original one have been matched to quantify their robustness. Since the transformation is simulated, the *ground truth* is known; that is, the correct matches between the original image and the transformed image are known. A *tolerance region* around the desired position in the transformed image is used to define the correctness of a match. Let us define the original image keypoint $\mathbf{p} = (x, y)$, which desired location for a given transformation is $\mathbf{p}' = (x', y')$, and the matched transformed point determined by the algorithm as $\mathbf{t} = (x'', y'')$. Note that $\mathbf{p}$ and $\mathbf{p}'$ are the same for some transformations (i.e. illumination or blurring), but could be different (i.e. scale or rotation) as shown in Figure 4.2 for $\mathbf{p}_3$. Given a match $< \mathbf{p}, \mathbf{t} >$ determined by the matching algorithm (i.e. FLANN), it will be considered correct if $\mathbf{t}$ lies within the tolerance region centered in $\mathbf{p}'$ (green cells in the figure); otherwise it will be considered incorrect (red cells in the figure). In this project a 3×3 pixel tolerance region has been used.

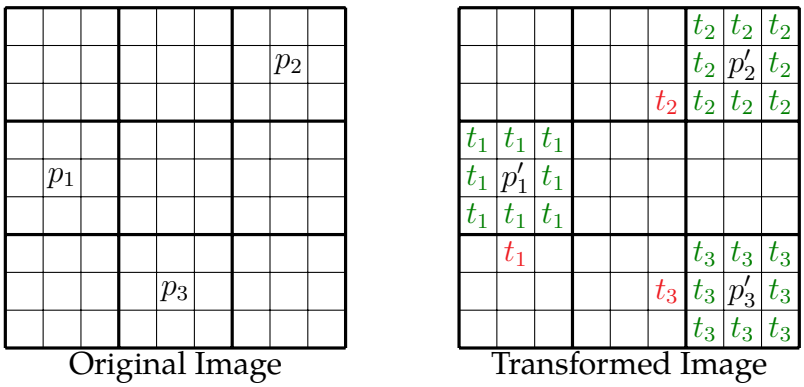


Figure 4.2: Example of 3x3 pixel tolerance region for the keypoints $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$. Desired transformed points $\mathbf{p}_i^{'}$ in the transformed image are represented and the tolerated matches $< \mathbf{p}_i, \mathbf{t}_i >$ are shown in green.

In the results, the left graphic represents the amount of correct matches within the range of possible transformation values. This measure di-

rectly depends on the number of keypoints detected and is biased (see Figure 4.1). Thus, in the right graphic the percentage of detected keypoints that are correctly matched is represented. Below are the results and conclusions for some possible transformations.

A) Scale transformation

It is common to find the same object at different scales in two images associated with the same scene, either due to the distance from which the picture is taken or the device configuration (i.e. zoom). The amount and percentage of correct matches obtained for different scaling factors applied to the *Lena* image are shown in Figure 4.3.

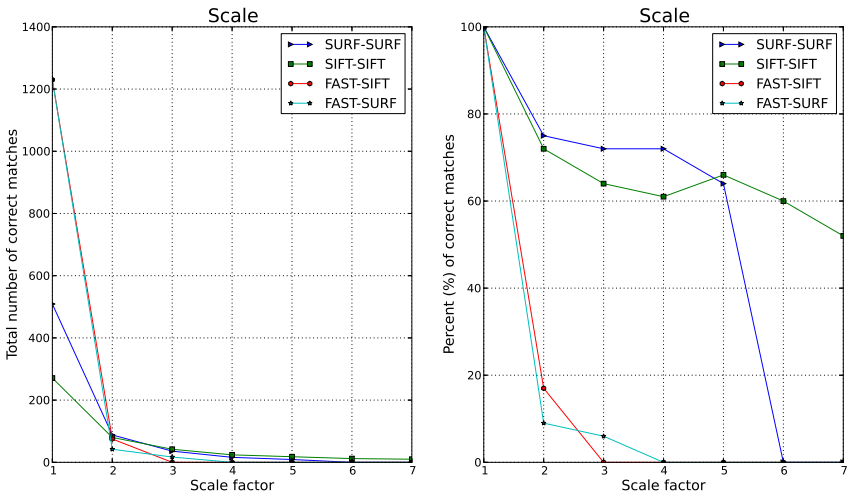


Figure 4.3: Amount and percentage of correct matches (3×3 tolerance region) for different scaling factors and detector-descriptor combinations.

From the analysis of Figure 4.3, SURF is the most robust algorithm up to 5, for higher values the robustness deteriorates drastically. Opposite, the behaviour of SIFT is regular within the range of scale factor values. Recall that the robustness against scale transformations is tackled using the scale space technique; the standard scale space defined in SIFT (Lowe [21]) has 4 octaves versus the 3 octaves defined in SURF (Bay [22]). Thus, the regular behaviour of SIFT seems reasonable. In addition, let us consider very blurred images within the scale space; that is, last octaves and high blurring factor. The regions of interest associated to the keypoints are supposed to be very similar due the

lack of high frequencies. Therefore, the dimensionality of the descriptor vector becomes relevant to discriminate within small variations. Notice the dimensionality of the descriptor vectors: 128 in SIFT and 64 in SURF. On the other hand, FAST detector provides more detected keypoints and the amount of correct matches increases.

B) Illumination transformation

This transformation refers to changes in the illumination; that is, variations in the intensity levels of the pixels. These variations can be produced by the nature (i.e. sunset or sun-shadow) or synthetically (i.e. camera flash or lights). The illumination transformation considered in the project is uniform, thus an offset is added to the intensity value for all the pixels in the image. The amount and percentage of correct matches obtained for different offsets applied to the *Lena* image are shown in Figure 4.4.

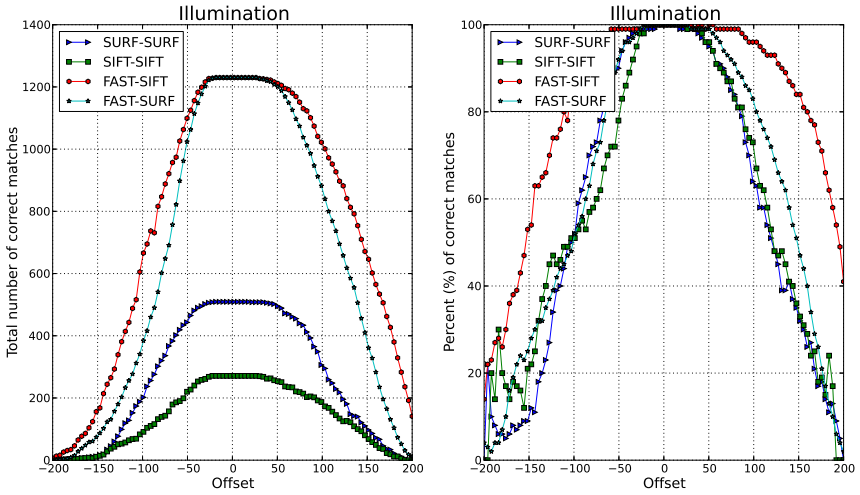


Figure 4.4: Amount and percentage of correct matches (3×3 tolerance region) for different illumination offsets and detector-descriptor combinations.

Except for the FAST-SIFT combination, which provides the best performance, all the algorithms perform similarly for constant transformations in the illumination. In Figure 4.4 the performance decreases when the offset increases. Indeed the behaviour described seems reasonable. Recall that the robustness against illumination transformations is tackled normalizing the descriptor vector.

C) Blurring transformation

The interest of this transformation comes from the acquisition of images using devices with poor quality camera, helping to evaluate the performance in mobile devices (though the cameras in mobile devices are getting better). To emulate the blurring, a gaussian spatial filter is used. There are two possible parameters to modify: the mask size, defined as $S \times S$, and the standard deviation σ in the approximated two-dimensional gaussian expression.

The percentage of correct matches obtained for different blurring transformations on the *Lena* image are shown in Figure 4.5. The combination FAST-SIFT provides a poor performance and is not represented to improve the visualization and analysis. The color notation remains the same; blue, green and red for SURF-SURF, SIFT-SIFT and FAST-SURF respectively.

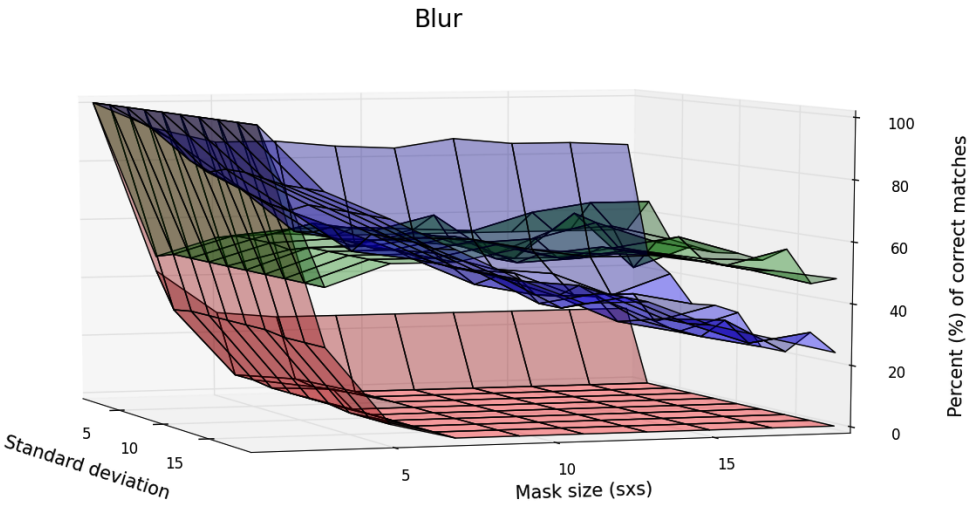


Figure 4.5: Percentage of correct matches (3×3 tolerance region) for different mask sizes ($S \times S$), standard deviations (σ) and detector-descriptor combinations (colors).

From the analysis of Figure 4.5, SURF is the most robust algorithm for small transformations, but its robustness deteriorates for large transformations. Opposite, the behaviour using SIFT is regular within the range of tested parameter values. Recall that the Laplacian of Gaussian (DoG) is approximated accurately in SIFT using Difference of Gaussians (DoG). However, the approximation used in SURF, based on Difference of Boxes (DoB), is loosely.

D) Rotation transformation

The analysis of this transformation emulates the potential rotation suffered on the object due to the acquisition process and device orientation. The percentage of correct matches obtained for different rotation transformations on the *Lena* image are shown in Figure 4.6.

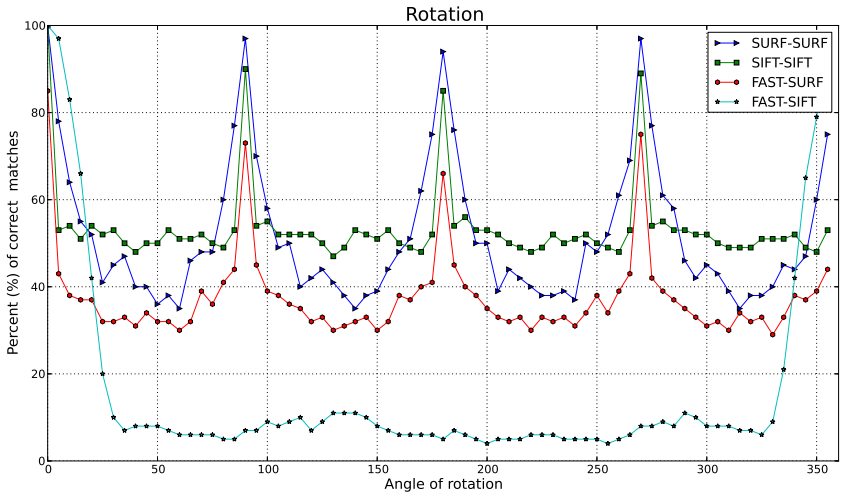


Figure 4.6: Percentage of correct matches (3×3 tolerance region) for different rotation angles (in degrees) and detector-descriptor combinations.

Again, the behaviour using SIFT is regular within the rotation angle values. Recall that the robustness to rotation is tackled assigning a dominant orientation to the keypoint and subtracting it from the descriptor vector. Histograms are used to estimate the orientation. SIFT implements a histogram with 36 containers, thus 36 orientations can be distinguished. In contrast, SURF calculate the orientations in $\frac{\pi}{3}$ radians windows, thus 6 possible orientations can be distinguished. Notice that the matching percent is close to 100 % for 90, 180 and 270 degrees.

The main conclusion extracted from the experiments is that SIFT and SURF are robust against transformations. However, the FAST detector is recommended if the input image and the reference image are the same or similar. Notice that both, the amount of detected keypoints and the percentage of correct matchings, are high. It is important to remember that FAST is an algorithm for detection, while SIFT and SURF implement both,

detection and description. The detection and description algorithms can be combined among themselves. For more information about the robustness of SIFT and SURF consult [42] [43].

4.1.3. Computational time

The computational time is a key factor in real-time applications. This time can be divided according to the three main object recognition phases (detection, description and matching) described in Chapter 2. The three considered times and a brief intuition about what they represent are listed below.

- Detection time

  Time needed to detect all the keypoints in an image. Given an algorithm, the detection time depends on the content and size of the image. The detection time varies within algorithms due to the scale space creation method and the used approximations.

- Description time

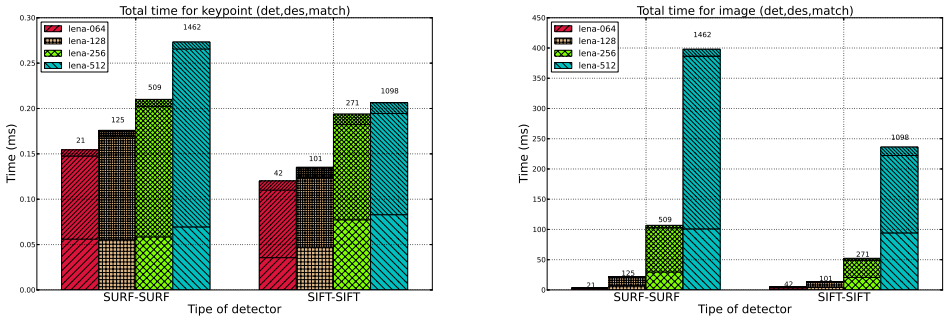
  Time needed to compute the descriptor vector for the regions of interest associated to the previously detected keypoints. The description time depends on the amount of previously detected keypoints and the algorithm used.

- Matching time

  Time needed to search matches between two images. The matching time depends on the amount of keypoints and the dimensionality of the descriptor vector. It also depends on the type of descriptor vector; for example, binary vectors are compared more efficiently using the Hamming distance. Let us notice that detected keypoints and descriptor vectors for the database images may be previously calculated and stored. However, it is always necessary to match the input image with the images available in the database.

To obtain a reliable measure of the detection, description and matching times 10 realizations were executed and the average time computed. The analysis considered is the simplest one; that is, the input image and the reference image are the same, thus the keypoints detected and the descriptor vectors are also the same.

A bar diagram is used to represent the computational times. Each bar is divided in three sections indicating *detection time*, *description time* and *matching time* from bottom to top respectively. Thus, the total computational time is the sum of the three previously computed times; that is, the total height of the bar in Figure 4.7. Sometimes, the obtained measure is very small and difficult to distinguish in the graphics. Figure 4.7a represents the average computational times for a single keypoint and Figure 4.7b represents the total computational times for the images. In particular, the image used is the same (*Lena* image) but its size varies. The image sizes are 64x64, 128x128, 256x256 and 512x512 pixels.



(a) Keypoint computational time **(b)** Image computational time

Figure 4.7: Detection, description and matching computational times: (a) average computational times per keypoint; (b) image computational times for all detected keypoints. At the top of each bin, the total amount of keypoints is indicated.

The SURF implementation proposed by Herbert Bay [22] is outstanding for its computational time improvement; paraphrasing their authors: *"SURF computational time is several times more efficient than SIFT"*. It is not possible to obtain a convincing conclusion based only in the information provided by Figure 4.7. However, seems reasonable to assume that the efficiency property is not fulfilled for the SURF implementation provided in OpenCV. Furthermore, the analysis performed in the next subsection will provide a better understanding and support to this assumption. The next subsection explains the behaviour of the algorithms when using single-view object databases with different properties. The big amount of different databases and their different properties is a very common issue. This particular problem has to be carefully studied and analyzed to select the correct feature extraction algorithm.

4.1.4. *Single-view Database Performance*

The behaviour of the feature extraction algorithms and matching techniques when working with databases (i.e. composed by images) is very important. In this section, the behaviour of the algorithms in different single-view object databases is analyzed. In particular, the objects selected are book covers. The first database, denoted by BBDD-1, is used to retrieve images from the database based on content; that is, the descriptor vector singularity is studied. Note that all the book covers are 400×262. The second database, denoted by BBDD-2, is used to evaluate the behaviour of the algorithm in databases with different image sizes. The size of the images varies from 143×93 to 1186×702. The detailed explanation of the databases and its images are shown in Appendix A.

Let us continue with the analysis of the computational time to better understand the correctness of the previously stated assumption. The detection, description, matching and total computational times are shown in Figure 4.8, together with the amount of keypoints detected for the images in BBDD-1. Note that the feature extraction algorithms used are SIFT and SURF. For matching the FLANN algorithm is used.

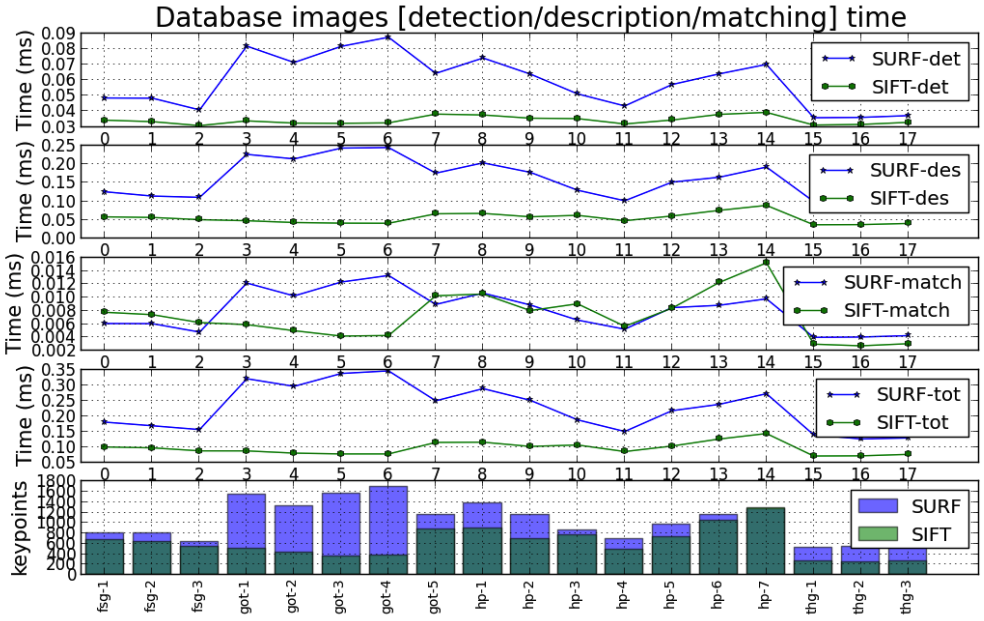
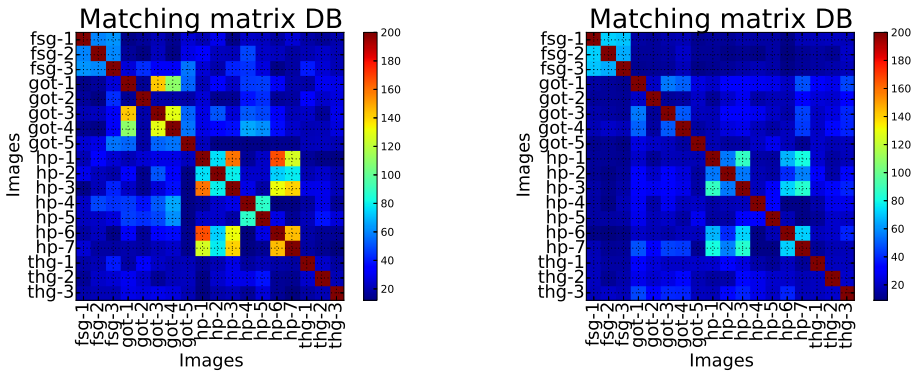


Figure 4.8: Detection, description, matching and total computational times, together with the keypoints detected for the images in BBDD-1. The feature extraction is done by SIFT/SURF algorithms and FLANN is used for matching.

Observing the graphics in Figure 4.8, it is possible to state that the detection and matching computational times are negligible in comparison with the description time for a single image. For this reason, the graphic shape representing the total computational time (sum of the detection, description and matching times) is similar to the description time shape. The matching time using SIFT is higher than the matching time using SURF. The descriptor vector dimensionality (128 for SIFT and 64 for SURF) explains this behaviour. Note that, although matching time is negligible, can become a problem for large databases. Analyzing the bottom representation in Figure 4.8 some conclusions can be extracted. First, the amount of detected keypoints is bigger in SURF, remarking the difference in the *"got-i"* book covers. Probably, this difference is caused by the extensively amount of letters (book title and author) in those covers. Second, the total computational time is lower in SIFT, even when the number of keypoints detected are the same (see *"hp-7"* in the figure). This behaviour is held among the databases used in this report and for this reason extrapolated to other databases. Thus, we can conclude that the previous assumption - the efficiency property is not fulfilled for the SURF implementation provided in OpenCV - is correct.

Other important properties are the ability of the algorithms to singularly describe the images and their aptitude to find elements in common; that is, retrieve images from a database based on their content. To evaluate this property, a cross-matching among all the possible combinations of image pairs within the database BBDD-1 is done. Only the best 200 keypoint matches are considered to avoid the number of detected keypoints dependence. The result is a square matching matrix (see Figure 4.9). The square matrix is supposed to be symmetric but, to briefly test the repeatability of the behaviour, the triangular-transposition method is not used. Thus, there exist two direction matches; that is, for a given pair of images the directions $I_1 - I_2$ and $I_2 - I_1$ are computed.

Analyzing the results in Figure 4.9 we conclude that both algorithms describe the image singularly enough to be able to recognize it. This conclusion is extracted from the garnet diagonal indicating that the number of correct matches for pairs $I_i - I_i$ is high. The main difference between both algorithms resides in the square patterns around the garnet diagonal. These patterns correspond to book covers of the same sage, and consequently with common elements. For example, the *Harry Potter* sage title or the author name in the *Game of Thrones*. It is important to notice that SIFT algorithm (see Figure 4.9b) does not detect those patterns with the

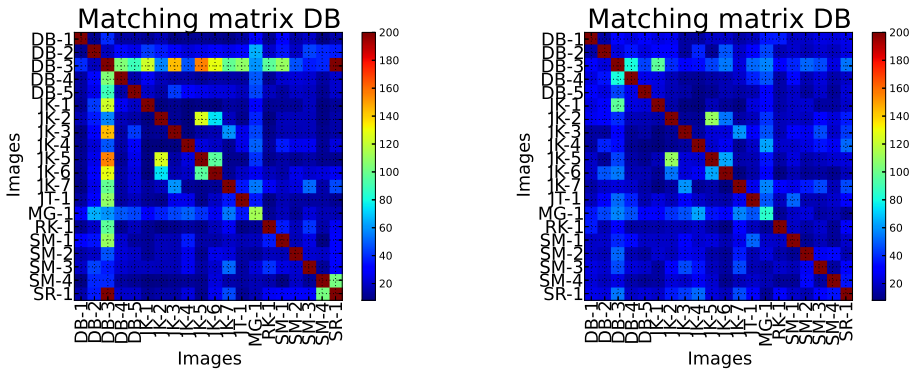


(a) SURF matching-matrix **(b)** SIFT matching-matrix

Figure 4.9: Amount of correct cross matches for all possible pairs within the images in BBDD-1. Highlight the garnet diagonal representing the good recognition performance and the square patterns due to the common elements in the book covers.

same clarity than SURF, consequently SIFT describes the keypoints more singularly. BBDD-1 is built under one assumption, all images have the same size. Usually, this assumption is not held and the available images in the database have different sizes, furthermore huge differences in size can lead the recognition algorithms to fail. To evaluate this property, a cross-matching with the same settings as the previous experiment is done (see Figure 4.10), but using BBDD-2 instead.

Taking a first look to the results, it is easy to observe that both algorithms are able to recognize the input image (garnet diagonal in figures). The number of correct matches for *"MG-1"* is reduced because of the reduced size of the image (143×93); that is, the number of detected keypoints is lower than 200. The main feature to stand out resides in the ability of SIFT to avoid false positive matches (mismatching), even when the images sizes are very different. Opposite, SURF provides a high amount of matches with other non-related covers (see Figure 4.10a). For example the *"DB-3"* book cover or the two garnet cells out the diagonal corresponding to the pair <*DB-3*,*SR-1*>. Note that *"DB-3"* is the largest image and *SR-1* the smallest one. The difference in the size of the images is a problem that can be solved in many ways: modifying the size of the images, using a different measure (i.e. percentage of correct matches) or simply using SIFT are some possible approaches.



(a) SURF matching-matrix **(b)** SIFT matching-matrix

Figure 4.10: Amount of correct cross matches for all possible pairs within the images in BBDD-2. Highlight the bad performance using SURF due to the difference of the size for images DB-3 and MG-1, biggest and smallest images respectively.

To generalize the conclusions extracted in the previous experiments a real database is used. The database contains book covers collected by OpenLibrary [1]. Under the slogan *"one web page for every book ever published"*, they have over 1.000.000 free ebook titles available and even more book cover images. A portion of this database, in particular 1000 book covers, have been used to repeat the previous experiments and thus, check and verify the previously extracted conclusions.

4.2. Graph-based Representation Analysis

The databases used in previous experiments were characterized by the existence of one single canonical view for the object (i.e book covers in this report). Most of the objects in the real world have several canonical views due to their 3D structure. The proposed graph-based approach drops the single-view assumption into a more general recognition problem. In this project the objects to classify/recognize are shoes. The first multi-view database, denoted by BBDD-3, is composed by generic images of shoes taken from different views. It is called generic because the same number of canonical views is not required for each shoe. Intuitively, not all the views for an object are equally important and there could even be views

[1] Link OpenLibrary: http://openlibrary.org

with no information at all. To perform this experiment a new database, denoted by BBDD-4, with five different views for each object is used. The experiments related to the graph-based representation are performed and commented in this section.

4.2.1. Model Selection

The aim of these experiments is to train the classifier with the samples available in the database to select that model providing the best performance. The classifiers selected for this project are the *Support Vector Machines* and their theoretical background was explained in Subsection 2.2.1. To estimate the values of the free parameters providing better performance in completely unseen samples *Grid-Search* and *Cross-Validation* are used. The combination of these techniques was explained thoroughly in Subsection 2.2.2

The set of samples in the database are normalized before performing the search process and cross validation. The number of samples for each class is not the same and a weight term, inversely proportional to the amount of samples available, is used to overcome the difference. Let us define two classes $C_1 = \{(x_i, y_i)\}_{i=1}^{n=100}$ and $C_2 = \{(x_i, y_i)\}_{i=1}^{n=10}$, where n is the amount of samples for each class. The weight associated with the misclassification of a sample belonging to C_i are 0.01 and 0.1 respectively; that is, a C_2 misclassification is more penalized than a C_1 misclassification. The influence of outliers can lead the classifier to find very complex, maybe just wrong, hyperplanes. To palliate the presence of outliers, another weighted term is used. Let us define the sample $(\mathbf{x}_S, y_S)$, where $\mathbf{x}_S$ is the feature vector and y_S the assigned class, and assume that the sample belongs to the previously defined class C_1 (indicated by its label y_S). The centroid is calculated as the average of all the samples belonging to the class C_1. The weight term for the sample is stated in Equation 4.1, where N is the amount of elements in C_k and Φ is any function inversely proportional to the distance between the sample and the centroid. For example, the functions $\Phi(d) = \frac{1}{d}$ or $\Phi(d) = \frac{1}{exp(d)}$ can be used, as the distance d is always positive the behaviour of the function for negative d values are not considered. The result is a more general hyperplane that seems to generalize better.

$$weight(\mathbf{x}_S, C_k) = \Phi\left(\|\mathbf{x}_S - \frac{1}{N}\sum_{\mathbf{x}\in C_k}\mathbf{x}\|\right) \tag{4.1}$$

The different trained models and their scores are shown in Figure 4.11. The models are represented sorted by their accuracy in the x axis and the y axis shows the accuracy value. The kernel is indicated by the color of the circle; red for linear kernel, green for polynomial kernel and blue for the radial basis function kernel. Regarding the free parameters: the degree d in polynomial kernels is inside the circle and the parameters C and γ are at the bottom and top of the circle respectively. To easily identify the value of these parameters a range of colors is used. Blue represents low values while pink represents high values.

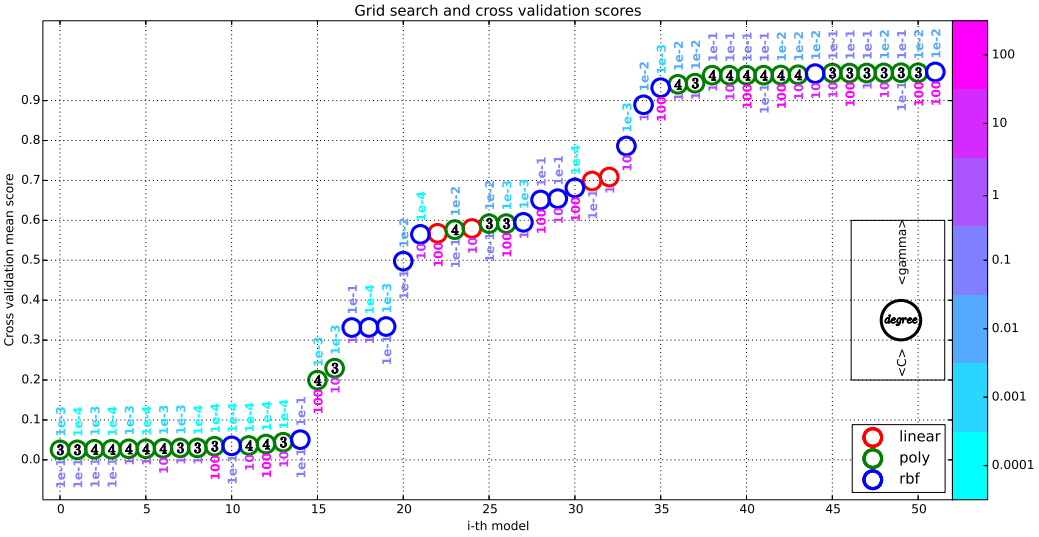


Figure 4.11: Models sorted in ascending order by their accuracy. The training set represents the 70 % of the samples in the database. Grid-Search and Cross-Validation with $v = 5$ folds was used.

The training set is divided into $v = 5$ folds with the same size. For each fold, the accuracy of the classifier trained with $v - 1$ fold is evaluated in the remaining fold. The accuracy shown in Figure 4.11 is the average of the accuracies obtained for all the folds. The best rated model is a Support Vector Machine with a radial basis function kernel, where the free parameters are $C = 100$ and $\gamma = 0{,}01$. The performance provided by this model, using the evaluation set with unseen samples by the classifier, is shown in Table 4.1. The precision, recall, f1-score and support are shown individually for each class. We conclude that classification using SVMs performs with an average accuracy of 97 %. Recall that the images of the

shoes in the database were taken from different positions not restricting ourselves to a single canonical view.

class	precision	recall	f1-score	support
DC	0.88	0.85	0.87	696
achilles	0.94	0.96	0.95	671
ankle	0.97	0.96	0.97	1066
button	1.00	1.00	1.00	58
converse	1.00	1.00	1.00	351
cord	0.99	1.00	0.99	3665
cue	1.00	1.00	1.00	678
dots	0.98	0.93	0.96	212
double-sewing	0.98	1.00	0.99	711
elastics	1.00	0.99	0.99	310
lacoste	1.00	1.00	1.00	338
lecoq	1.00	0.97	0.99	160
line	1.00	0.99	0.99	376
nike	0.99	0.99	0.99	812
relief	1.00	0.99	1.00	636
ring	0.98	0.97	0.97	490
running	1.00	1.00	1.00	160
single-sewing	0.76	0.78	0.77	398
spring	1.00	0.99	1.00	107
straight	0.99	0.97	0.98	151
triang-sewing	1.00	1.00	1.00	389
avg / total	**0.97**	**0.97**	**0.97**	**12435**

Table 4.1: Table with the scores obtained for each class and average score. Note that the observations used have not been previously seen by the classifier and support indicates the number of samples of that class in the overall evaluation.

4.2.2. View-relevance and performance

As mentioned before, the object can have several canonical views and not all of them are equally important. A matrix form representation (see Figure 4.12) is used to show the relevance of each of the views when training the classifier. In the y axis the views used to train the classifier are represented (we use 70 % of the training samples) and the evaluation views are placed in the x axis. The accuracy obtained within the range [0,1] is shown by the color of the cell. Note that the database used to create the

Figure 4.12 has five different views for each object, considered by us as canonical views.

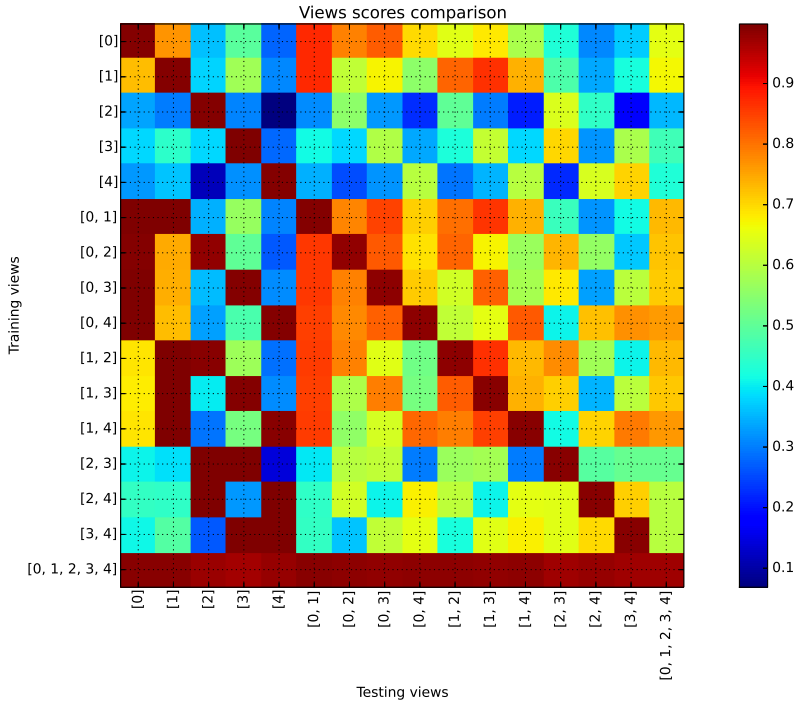


Figure 4.12: Accuracy matrix obtained for different views. The views [0] and [1] are closely related and both contains the most of the relevant information. Note the garnet diagonal as an indicator for the good performance of the SVMs.

The most important conclusions we draw from the result obtained when using the BBDD-4 database to train the classifier are:

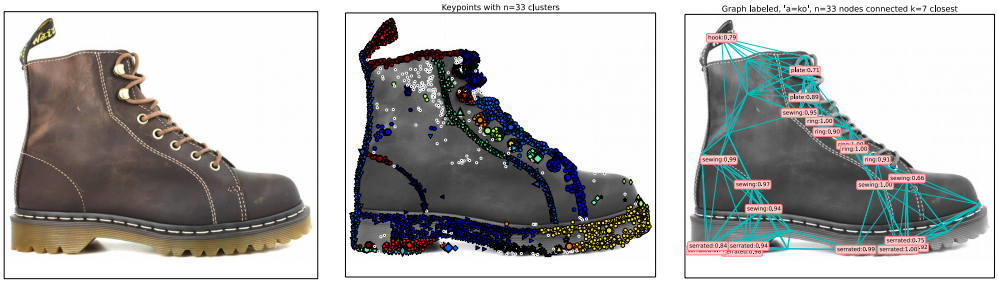
- If the training data adequately represents all views, the accuracy obtained for each view separately or any combination of them performs good. If we look at the last row (where the training set is [0, 1, 2, 3, 4]) the accuracy obtained reaches values over 0.95 for any combination of views in the x axis.
- For any view, trained with 70 % of its available samples, the *Support Vector Machines* are able to detect the patterns needed to achieve good prediction results in the evaluation set (30 % remaining samples). This fact is represented by the garnet diagonal.

- If two views of the object are very similar, the information provided by one of them is enough to get a good prediction result. In the first two rows of the Figure 4.12 the accuracies obtained by training with the view [0] and [1] are very similar.
- Some views are more representative than others. In Figure 4.12 any combination where view [0] or [1] is used to train provides acceptable accuracy values. In contrast, those combinations in which both views [0] and [1] are not used provide a low accuracy result. In particular, they are not able to obtain high accuracy for the following evaluation sets [0], [1], [0,1] and [0,1,2,3,4].

From the results obtained and explained in this section we can state that SVMs are able to extract the patterns needed to get a good classification performance in new completely unseen samples. It is also interesting to analyze the possible views available within the database and thus avoiding redundant information.

4.2.3. *Graphic Results*

In this subsection some classification and recognition graphic results are shown to the understand previously explained concepts graphically. No further conclusions are extracted from the following images. Figure 4.13 and Figure 4.14 show the original image, cluster analysis result and labelled graph-based representation for a boot and several views of a casual shoe respectively (more examples in Appendix C).



(a) Original image **(b)** Boot clustering **(c)** Boot labelled graph

Figure 4.13: Boot example: (a) original image; (b) cluster analysis result; (c) labelled graph-based ($k = 7$ closest nodes) representation for the object. Note the clustering uses k-means on $\mathbf{v}$ and optics on $\mathbf{p}$ recursively.

Figure 4.14: Examples of recursive cluster analysis, graph-based representation and predicted labels for different views of the same shoe.

5

CONCLUSIONS

In previous chapters the context of the project, the related background and the proposed approach have been explained. Furthermore, some experiments have been performed to evaluate the robustness of feature extraction algorithms, to select the model (classifier) with the best performance and to evaluate it in different views. This last chapter presents the main extracted conclusions and suggests future lines of work.

5.1. Conclusions

The main conclusion following the completion of this final project is, without doubt, that computer vision is a research field growing constantly that requires huge amount of knowledge. This knowledge can vary from basic concepts in image processing, such as lighting changes or transformations to other spaces (Fourier transform, Hough transform) to advanced concepts of segmentation, graph theory and feature extraction, to name a few. Below are some of the particular conclusions extracted after the completion of this project.

- Choosing the right (maybe ideal) feature extraction algorithm is a difficult task. Based on the results, SIFT performs generally better than SURF and its computational time is smaller in the OpenCV implementation. Furthermore, keypoints detected and described with SIFT are valid to tackle classification and recognition.
- The selection of the free parameters for the model (classifier) using Grid-Search and Cross-Validation has a direct impact on the quality and accuracy of the classifier for new unseen samples.
- Support Vector Machines are able to extract the necessary patters within the data to provide good results in multi-view object recognition and classification.

- The cluster analysis and the graph-based representation of the object open a new range of possibilities. The introduction of new features to an observation $\mathbf{x}$ (i.e. correlation between samples within a clusters), the application of structured learning and prediction algorithms (i.e. SO-SVM) and the use of graph-theory algorithms to create, search and compare graphs are some examples.

5.2. Future Work

The improvements that can be added to the algorithms are countless and in fact, because this is a current research field, many of them are likely to be developed in the future. Among the improvements considered relevant should be highlighted:

A) Increase of the robustness of the algorithms against geometric affine transformations, because they are one of the most common transformations when capturing an image with the mobile device. The method developed by Jean Michel Morel and Yu Guoshen called ASIFT [24], may be effective but also the increase in the number of false positives due to the increase (perhaps excessive) in the number of keypoints can lead to errors.

B) A mathematical measure for the clustering performance could help to choose the number of recursions and the dimension to be applied (descriptor vector or position) for each particular image, providing better results. This measure should penalize both, very low and very high amount of clusters.

C) Store a larger amount of information for each node in the graph. Since each node represents a set of points, makes sense to store in addition to the descriptor vectors information about the distribution of the keypoints in the cluster (i.e. the correlation).

D) Use of dimensionality reduction techniques to select the features that really contains the relevant information.

E) The use of complex graph theory could help to create a better graph-based representation of the object. For example, a mask obtained with the segmentation of the image in background and foreground could be used to filter those edges connecting two non-related nodes; that

is, using searching algorithms only arcs lying on the foreground can be used to create the graph.

F) Since there are multiple canonical views of the object available in the database, a similar idea to the one proposed by Rothganger et al [38] could be used. Their approach represents the object in 3D and subsequently divide it into smaller and invariant patches. The detector and descriptor is computed on these patches and the 3D information is stored in addition to the descriptor vectors. Instead of dividing the object into patches, a 3D graph representing the object by its most important elements can be created. Furthermore, methods evaluated in this project could be applied.

G) A probabilistic estimate of the 3D object model [44] could help to determine which object views are most important, and the number of points of interest to be extracted from each view [45]. Inversely, we can estimate the available input view provided as input to the system by using the graph structure.

H) An efficient search of the object to be recognized in the database. One approach to reduce the search time is to use the statistics stored in the database [46].

BIBLIOGRAPHY

[1] M. Šonka, V. Hlaváč, and R. Boyle. *Ise-Imgage Processing, Analysis and Machine Vision*. International student edition. Thomson Learning EMEA, Limited, 2008.

[2] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Always learning. Pearson Education, Limited, 2011.

[3] C. Steger, M. Ulrich, and C. Wiedemann. *Machine Vision Algorithms and Applications*. Wiley-VCH Textbook. Wiley, 2008.

[4] Ra Lauric and Sarah Frisken. Soft segmentation of ct brain data [online], 2007. Last visited on 05/06/2013.

[5] YingLi Tian, R.S. Feris, Haowei Liu, A. Hampapur, and Ming-Ting Sun. Robust detection of abandoned and removed objects in complex surveillance videos. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(5):565–576, 2011.

[6] Vildan Tanriverdi and Robert J. K. Jacob. Interacting with eye movements in virtual environments. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '00, pages 265–272, New York, NY, USA, 2000. ACM.

[7] Robert J. K. Jacob. The use of eye movements in human-computer interaction techniques: what you look at is what you get. *ACM Trans. Inf. Syst.*, 9(2):152–169, April 1991.

[8] Fred Turek. Machine vision fundamentals. how to make robots see. *NASA Tech briefs magazine*, 35(6):60–62, 2011.

[9] K. Lai, Liefeng Bo, Xiaofeng Ren, and D. Fox. Detection-based object labeling in 3d scenes. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1330–1337, 2012.

[10] J. Solem. *Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images*. Oreilly and Associate Series. O'Reilly Media, Incorporated, 2012.

[11] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Andreas Mueller. Pystruct, 2013. http://mloss.org/software/view/466/.

[14] Edward Rosten, R. Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):105–119, 2010.

[15] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, volume 6314 of *Lecture Notes in Computer Science*, pages 778–792. Springer Berlin Heidelberg, 2010.

[16] Motilal Agrawal, Kurt Konolige, and MortenRufus Blas. Censure: Center surround extremas for realtime feature detection and matching. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, volume 5305 of *Lecture Notes in Computer Science*, pages 102–115. Springer Berlin Heidelberg, 2008.

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

[18] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571, 2011.

[19] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, July 2008.

[20] Adam Schmidt, Marek Kraft, Michał Fularz, and Zuzanna Domagała. Comparative assessment of point feature detectors and descriptors in the context of robot navigation. *Journal of Automation, Mobile Robotics and Intelligent Systems*, 7(1):11–20, 2013.

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

[22] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[23] Yan Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–506–II–513 Vol.2, 2004.

[24] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM J. Img. Sci.*, 2(2):438–469, April 2009.

[25] Tony Lindeberg. Scale-space theory: a basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, 21(1-2):225–270, 1994.

[26] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[27] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, March 2002.

[28] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. Artif. Int. Res.*, 2(1):263–286, January 1995.

[29] I. Steinwart and A. Christmann. *Support Vector Machines*. Information science and statistics. Springer, 2008.

[30] S. Abe. *Support Vector Machines for Pattern Classification*. Advances in Computer Vision and Pattern Recognition. Springer, 2005.

[31] Iglesias García F.J. Structured prediction of network data. Master's thesis, July 2013.

[32] S. Nowozin and C.H. Lampert. *Structured Learning and Prediction in Computer Vision*. Foundations and Trends in Computer Graphics and Vision. Now Publishers, 2011.

[33] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[34] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[35] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, June 1999.

[36] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *SIGKDD Explor. Newsl.*, 4(1):65–75, June 2002.

[37] C.M. Cyr and B.B. Kimia. 3d object recognition using shape similiarity-based aspect graph. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 254–261 vol.1, 2001.

[38] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. 3d object modeling and recognition using affine-invariant patches and multi-view spatial constraints. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–272–7 vol.2, 2003.

[39] Jean Ponce, Svetlana Lazebnik, Fredrick Rothganger, and Cordelia Schmid. Toward True 3D Object Recognition. 2010.

[40] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2161–2168, 2006.

[41] Brian Tomasik, Phyo Thiha, and Douglas Turnbull. Tagging products using image classification. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 792–793, New York, NY, USA, 2009. ACM.

[42] N.Y. Khan, B. McCane, and G. Wyvill. Sift and surf performance evaluation against various image deformations on benchmark dataset. In *Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on*, pages 501–506, 2011.

[43] Johannes Bauer, Niko Sünderhauf, and Peter Protzel. Comparing several implementations of two recently published feature detectors. *University of Pennsylvania Law Review*, 154(3):477+, January 2006.

[44] Min Sun, Hao Su, S. Savarese, and Li Fei-Fei. A multi-view probabilistic model for 3d object classes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1247–1254, 2009.

[45] Yu Xiang and S. Savarese. Estimating the aspect layout of object categories. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3410–3417, 2012.

[46] M. Aly, M. Munich, and P. Perona. Indexing in large scale image collections: Scaling properties and benchmark. In *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*, pages 418–425, 2011.

A

SINGLE-VIEW OBJECT DATABASES

The used database for the experiments with single canonical view object are explained below.

A.1. BBDD-1

The main objective is to evaluate the ability of retrieving images from a database based on content. This database is composed by book covers from 4 different sagas: *Fifty shades of Gray* (3 covers), *Game of thrones* (5 covers), *Harry Potter* (7 covers) and *The hunger games* (3 covers).

Thus, there are 18 covers available in the database, one for each canonical view of the book. All the covers are the same size, particularly 400×262 pixels, and were downloaded from *Google Images*. The complete database is shown in Figure A.1.

(a) fsg-1 **(b)** fsg-2 **(c)** fsg-3 **(d)** got-1 **(e)** got-2 **(f)** got-3 **(g)** got-4 **(h)** got-5 **(i)** hp-1

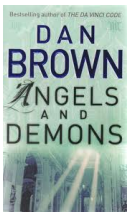
(j) hp-2 **(k)** hp-3 **(l)** hp-4 **(m)** hp-5 **(n)** hp-6 **(ñ)** hp-7 **(o)** thg-1 **(p)** thg-2 **(q)** thg-3

Figure A.1: Book covers in BBDD-1.

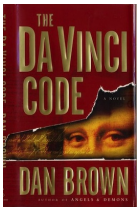
A.2. BBDD-2

The main objective is to evaluate the ability of singularly describing the images to outperform the different size images issue. This database is composed by 19 book covers, each one representing the single canonical view of the book.

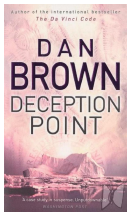
The sizes of the images varies in the database from the smallest image, *"SR-1"*with 143×93 pixels (see Figure A.2m), to the biggest image, *"DB-3"* with 1186×702 pixels (see Figure A.2c). They were downloaded from *Google Images* and the complete database is shown in Figure A.2.



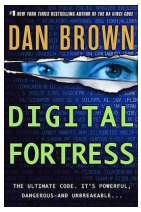
(a) DB-1



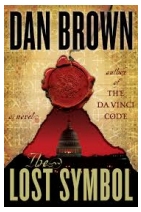
(b) DB-2



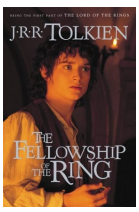
(c) DB-3



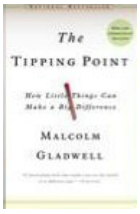
(d) DB-4



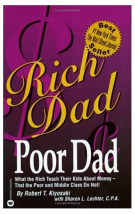
(e) DB-5



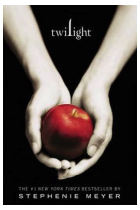
(f) JT-1



(g) MG-1



(h) RK-1

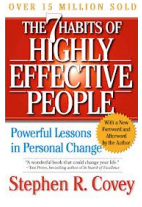


(i) SM-1

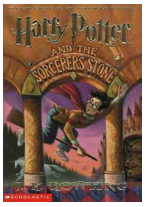
(j) SM-2

(k) SM-3

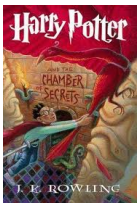
(l) SM-4



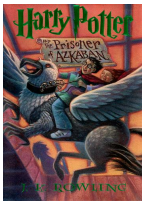
(m) SR-1



(n) JK-1

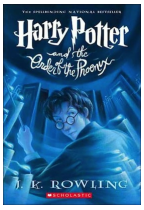


(ñ) JK-2

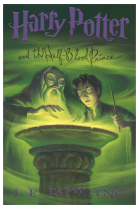


(o) JK-3

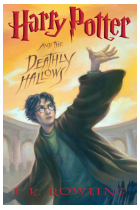
(p) JK-4



(q) JK-5



(r) JK-6



(s) JK-7

Figure A.2: Book covers in BBDD-2.

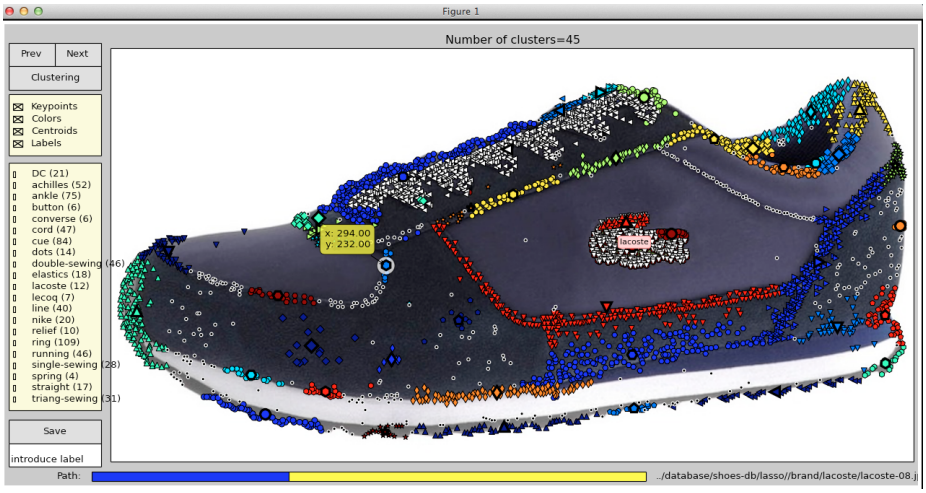
B

LABELLING GRAPHIC INTERFACE

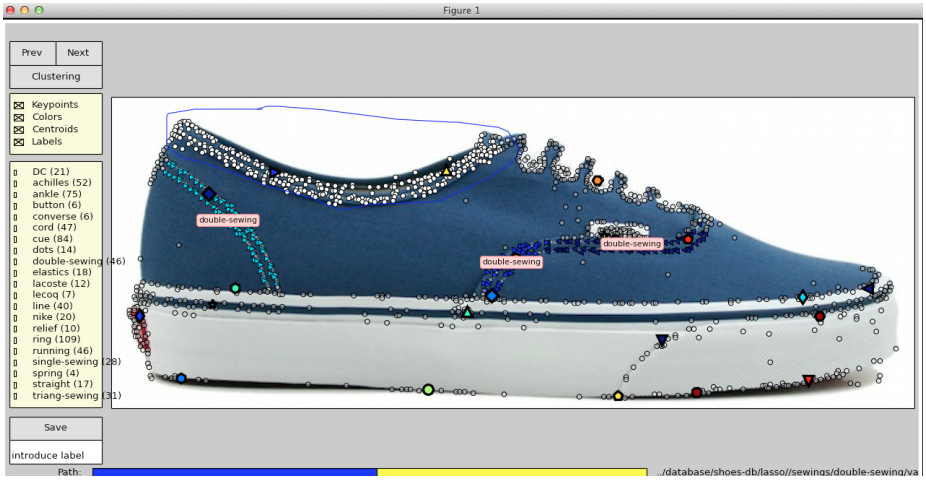
This appendix shows two graphic examples for the labelling graphic interface created to assign human-understandable labels to the objects within the database. The interface provides elements to shift within the set of images to label; *Prev* button and *Next* button to move among the images sequentially and the bottom bar to select the desired image manually. Different display options to select the elements to show are available for the user: keypoints, centroids, labels and background color. The labelling of the keypoints can be done in two different ways: *centroid* mode and *lasso* mode.

In the *Centroid mode* initialization (see Figure B.1a) the algorithm computes the clusters automatically (represented by colors). In addition, for each cluster the centroid (represented with bigger size) is calculated by the average of all its assigned keypoints. Then the user can select one or more clusters by clicking in their centroid. Note that the closest centroid to the mouse position is shown and highlighted automatically (yellow box and white circle). Once the label is introduced, for the centroids it is propagated to all their associated keypoints. This method does not allow an exhaustive keypoint selection and it is dependent of the clustering algorithm. The selected clusters and those already labelled are shown in white.

In the *Lasso mode* initialization (see Figure B.1b) an exhaustive selection of the keypoints is allowed. Previously labelled keypoints are displayed with different colors. The color is different depending on the cluster they were assigned to. Those points that lie within the area selected by the user (delimited by the blue line) are highlighted in white and the rest are transparent. Once the label is introduced, it is propagated to all the selected keypoints. This method allows to group and label the keypoints as they are perceived by the user.



(a) Graphic interface in centroid mode

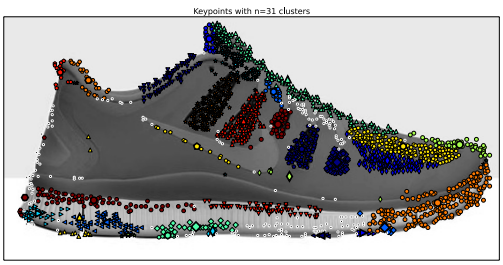


(b) Graphic interface in lasso mode

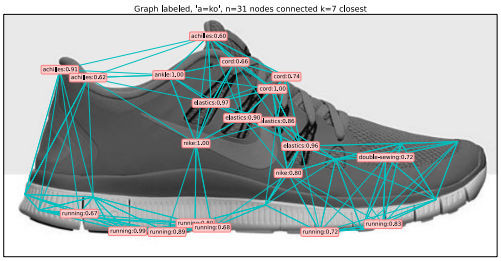
Figure B.1: Graphic interface initiated in different modes: (a) centroid mode; (b) lasso mode.

C

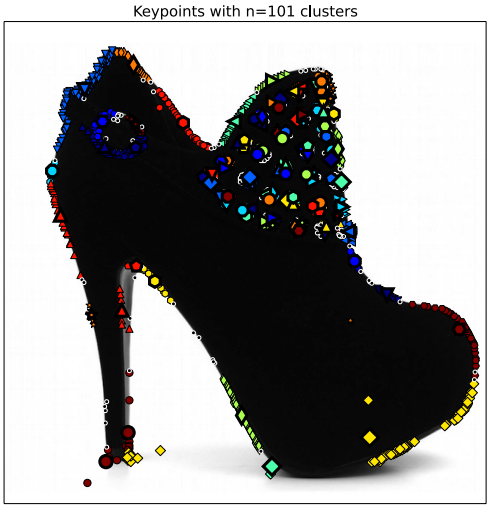
GRAPH-BASED REPRESENTATION EXAMPLES



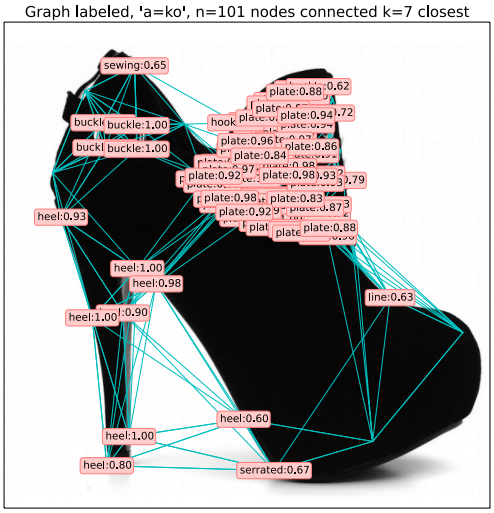
(a) Running clustering



(b) Running labelled graph

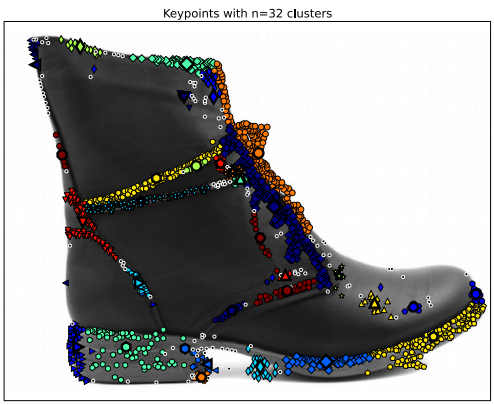


(c) High heel clustering

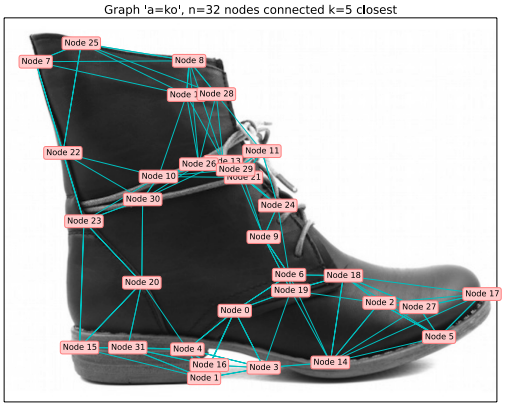


(d) High heel labelled graph

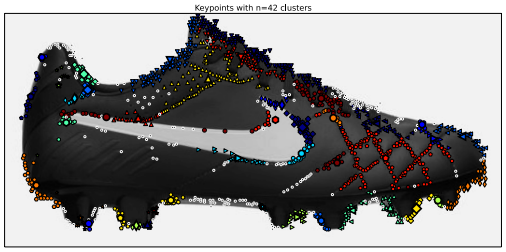
Figure C.1: Examples: (a) and (b) Clustering and graph-based representation ($k = 7$) for a running shoe; (c) and (d) Clustering and graph-based representation ($k = 7$) for a high heel shoe with plates.



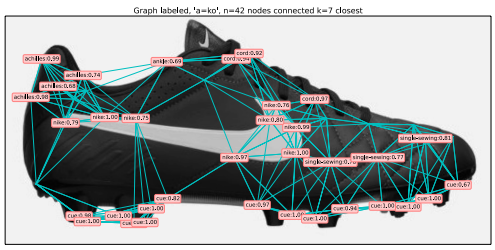
(a) Boot clustering



(b) Boot graph

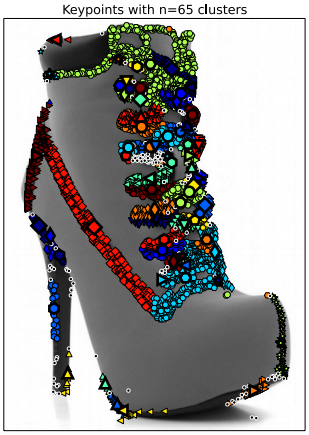


(c) Football boot clustering

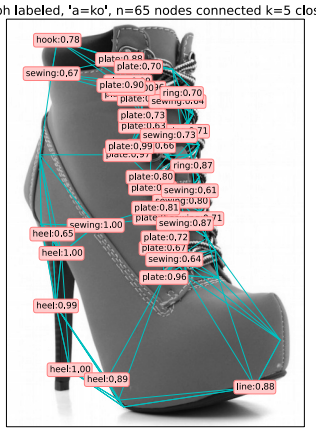


(d) Football boot graph

(e) Original high heel shoe



(f) High heel shoe clustering



(g) High heel shoe graph

Figure C.2: Examples: (a) and (b) Clustering and graph-based representation ($k = 5$) for a boot; (c) and (d) Clustering and graph-based representation ($k = 7$) for a football boot; (e), (f) and (g) Clustering and graph-based representation ($k = 5$) for a high heel shoe.

www.kth.se