UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Técnica de Telecomunicaciones (Especialidad Imagen y Sonido)

Proyecto Fin de Carrera

Extracción de Características de Imagen para Navegación de Robots Móviles

Antonio Redondo López

Febrero 2011

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Técnica de Telecomunicaciones (Especialidad Imagen y Sonido)

Proyecto Fin de Carrera

Extracción de Características de Imagen para Navegación de Robots Móviles

Autor: Antonio Redondo López
Fdo.:
Tutor.: Pedro M. Núñez Trujillo
Fdo.:

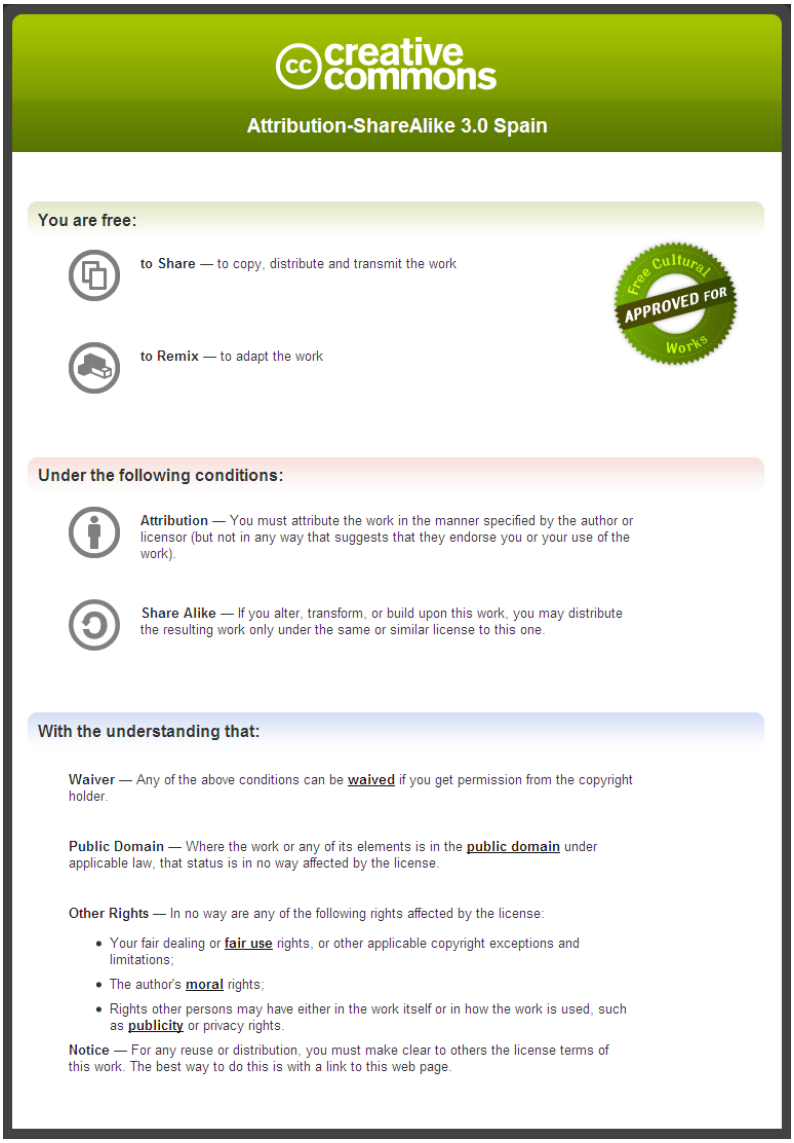
Tribunal calificador

Presidente: José Vicente Crespo
Fdo.:
Secretario: Valentín de la Rubia Hernández
Fdo.:
Vocal: Pablo Bustos García de Castro
Fdo.:

Calificación:
Fecha:

Nota: la Memoria está íntegramente en inglés, aunque el prólogo también está en español. En la sección Sobre la Memoria del Prólogo está explicado porqué.

Note: the Memory is entirely in English, although the prologue is also in Spanish. In About the Memory section of the Preface it is explained why.



Visit the project website on Google Code:
http://code.google.com/p/image-feature-detector

Contents

Prólogo

Éste es mi proyecto final de carrera que se convertirá en la guinda del pastel de una Ingeniería Técnica de Telecomunicaciones llena de excitantes e inesperadas aventuras. Su finalidad es dar un vistazo general al mundo de la visión por computador como una introducción a la memoria y a partir de este punto, elaborar un resumen de los principales detectores contemporáneos de características de imagen, principal finalidad del proyecto y disciplina de visión por computador que es fundamental en campos como biometría, industria, telecomunicaciones o videojuegos. A su vez, estos detectores de características de imágenes son implementados en un programa de ordenador para Linux con interfaz gráfica de usuario llamado Image Feature Detector, segundo objetivo del proyecto y ampliamente explicado en el Capítulo 4.

Cada capítulo de la memoria presenta un paquete de información en lo referente a extracción de características en procesamiento de imágenes y visión por computador. En el Capítulo 1 veremos el principal conocimiento en el cual este proyecto es desarrollado: la visión por computador y el procesamiento de imágenes. Será una visión global de los puntos claves de estos campos y una recopilación de los principales usos aplicados a la vida real, y posteriormente nos concentraremos en dos subtemas relacionados con el proyecto: odometría visual y emparejamiento por plantilla.

En el Capítulo 2 profundizaremos en el tema principal del proyecto, las características de imagen. Este capítulo está desarrollado desde sus origines y posteriormente referenciado a material más actual. Será un desarrollo teórico anterior a la complementación en el Capítulo 4.

En la mitad de la memoria, Capítulo 3, compararemos los principales detectores vistos en el Capítulo 2. Puesto que no todos los detectores funcionan de la misma manera, la comparativa está dividida basándonos en el tipo de detector, entre si son escalarmente variables (Harris y FAST) o escalarmente invariables (SIFT y SURF). Veremos cómo de potente y rápidos son los detectores y si están preparados para ser implementados en un sistema de video en tiempo real.

Posteriormente, en el Capítulo 4 explicaremos la implementación práctica de los detectores de características en un programa de ordenador para Linux. En este capítulo

hay implementaciones totalmente funcionales de la mayoría de las técnicas descritas en el Capítulo 2, y aplicadas para procesar imágenes.

Aunque el objetivo de este proyecto ha estado más enfocado en analizar imágenes en tiempo real para la aplicación posterior de odometría visual, las técnicas son generales y pueden migrar a otros ámbitos de aplicación. Además, hay una cierta cantidad de matemáticas en la memoria. Y es que la visión por computador puede considerarse como una rama de las matemáticas aplicadas, aunque esto no siempre se aplica a todas las áreas dentro del campo.

Yo mismo también me he beneficiado mucho escribiendo esta memoria. Gracias a ella he sido capaz de iniciarme en una manera bastante decente en la abarcante disciplina de la visión por computador, y especialmente en sus subdisciplinas de detención de características y emparejamiento por plantilla.

Gracias a esta memoria también he conocido el mundo de la literatura científica, algo que, aunque las fuentes de conocimiento (libros, apuntes de universidad, noticias de prensa y artículos de Internet que cada día están con nosotros) están basados en gran medida en estos trabajos, en mi humilde existencia solo los vi como esas pequeñas referencias que están al final de libros y artículos de Wikipedia.

Sobre la Memoria

He intentado elaborar la memoria en el modo más claro y lógico posible. Para hacerlo he puesto especial atención en la apariencia y orden de textos y figuras. Esto es esencial para obtener claridad e incrementar la velocidad de lectura y comprensión, aspecto importante en largos trabajos académicos y científicos como éste. Además, con una buena selección de fuentes, colores y orden la memoria es visualmente más atractiva. Al final del prologo está explicado cuales programas y fuentes han sido usadas en la memoria.

Absolutamente todas las fuentes y referencias usadas en la creación de esta memoria, ya sean libros, literatura científica, documentación on-line o artículos de Wikipedia, están enumerados en la Bibliografía. Para añadir claridad a la importancia de cada fuente, hay dos tipos de referencias, explicadas en el siguiente párrafo. Durante los capítulos, fragmentos resumidos de estas fuentes son usadas, y normalmente los capítulos y párrafos de un determinado tema empezarán con una referencia a la fuente original. También, todas las imágenes que aparecen a lo largo de la memoria tienen la fuente original enumerada en la Bibliografía.

A lo largo de la memoria hay referencias ([x]) a la Bibliografía. Estas referencias están clasificadas en 2 categorías: en azul ([x]) están las referencias principales, referencias que son básicas para la elaboración y comprensión del proyecto y las cuales he cogido información para escribir la memoria o comprender algún elemento relacionado. Estas referencias en la mayoría de los casos no solo aparecen sino que

además son explicadas en el párrafo o capítulo con texto e imágenes. Por otra parte, en verde ([x]) están las referencias secundarias. Estas referencias no son decisivas para la elaboración y comprensión del proyecto y muchas veces son referencias que solo son mencionadas sin mayor explicación, a diferencia de las referencias azules. Pero deben aparecer durante las explicaciones para poder crear una red de conceptos e ideas. Además, si el lector quiere, permite investigación adicional.

¿Por qué he decidido hacer el proyecto (memoria y programa de ordenador) en inglés? He tomado esta decisión, primero de todo, porque tengo un nivel de inglés escrito suficiente para escribir más o menos decentemente la memoria. No voy a hacer ninguna presentación en inglés, así que escribir la memoria en inglés es una tarea asumible puesto que no requiere hablar inglés, la parte dura de aprender idiomas. Segundo, el inglés es el idioma del conocimiento; casi toda la literatura científica y muchos libros técnicos, son, en su origen, en inglés. Y tercero y como consecuencia de las razones anteriores, si me comunico en inglés mi trabajo podrá llegar a más audiencia científica en vez de si lo hubiera escrito en español. Y como *casi* ingeniero, debo saber inglés, y he hecho uso de él. Por supuesto, he usado herramientas que me han ayudado a escribir un inglés mejor; empezando por el corrector gramatical de OpenOffice.org Writer, y posteriormente, todo tipo de herramientas para averiguar si lo que escribía era correcto: Google Translate, SYSTRANet, WordReference.com y English Wiktionary. La memoria está escrita en inglés americano.

Para escribir la memoria he usado OpenOffice.org Writer 3.2; me permite cubrir todas las necesidades que pueden aparecer en un trabajo académico como éste, y además, es multiplataforma y software libre y de código abierto. La edición de imagen no ha sido una tarea importante en el proyecto, pero para algunas imágenes y algún icono de Image Feature Detector he usado Corel Photo Paint X5 para editar y crearlos. La fuente usada para el cuerpo del texto es URWClassico, para los títulos Myriad Pro y para las fórmulas Minion Pro.

Preface

This is my end of degree project that will become the icing on the cake of a Telecommunications Degree full of exciting and unexpected adventures. Its scope is to give a general view of the computer vision world as an introduction to the memory and thereupon, to elaborate a summarize of the main contemporaneous image feature detectors, main scope of the project and computer vision discipline that is fundamental in fields as biometrics, industry, telecommunications or video games. In turn, these image feature detectors are implemented in a computer program with GUI for Linux called Image Feature Detector, and second scope of the project and widely explained in Chapter 4.

Each chapter of the memory presents a particular package of information concerning feature extraction in image processing and computer vision. In Chapter 1 we shall view the main knowledge in which this project is developed: the computer vision and the image processing. It will be a global vision of the key points of these fields and a collection of main uses applied to the real world, and we will afterwards focus in two subsubjects related with the project: visual odometry and template matching.

In Chapter 2 we will deepen in the main subject of the project, the image features. This chapter is developed from its origins and later referenced to more recent material. It will be a theoretical development prior to implementation in Chapter 4.

In the middle of the memory, Chapter 3, we will compare the leading detectors viewed in Chapter 2. Since not all detectors work in the same way, the comparative is divided by basing on the detector kind, whether scale-variant (Harris and FAST) or scale-invariant (SIFT and SURF). We will see how powerful and fast detectors are and if they are ready to be implemented in an real-time video system.

Later, in Chapter 4 we explain the practical implementation of features detector in a Linux computer program. In this chapter there are full working implementations of most of the major techniques described in Chapter 2, and applied them to process imagery.

Although the target of this project has been more focused in analyzing live video

imagery to the later application of visual odometry, the techniques are general and can migrate to other application domains. Furthermore, there is a certain amount of mathematics in the memory. And is that computer vision can be thought of as a branch of applied mathematics, although this does not always apply to all areas within the field.

I myself have already benefited much by writing this memory. Thanks to it I have been able to introduce me in fairly grade on the comprehensive discipline of computer vision and specially in its subdisciplines of feature detection and template matching.

Thanks to this memory I have also met the world of scientific literature, something that, although knowledge sources (books, university notes, media news and Internet articles that every day are with us) are based in great extend on these works, in my humble existence I only viewed them as those little references that are at the end of books and Wikipedia articles. I have learned to deepen on everything that is going to be taken into account, and that most times it will be easier than it could be at first. And as final lesson, I have learned to know what I want, what I need and seeking it in right places in a right way. And all this being as self-sufficient as possible.

About the Memory

I have tried to elaborate the memory in the clearest and most logical way. To do so I have put special attention in the appearance and arrangement of text and figures. This is essential to obtain clarity and increase the reading and understanding speed, important aspect in academic and scientific long works like this. In addition, with a good selection of fonts, colors and arrangements the memory is visually more attractive. At the end of the prefaced is explained which programs and typefaces have been used in the memory.

Absolutely all sources and references used in the creation of this memory, whether books, scientific literature, on-line documentation or Wikipedia articles, are listed in the Bibliography. To add clarity to the importance of each source, there are two kinds of references, explained in the next paragraph. Among chapters, summarized fragments of these sources are used, and usually chapters and paragraphs of a determinate subject will begin with a reference to the original source. Also, all the images than appear thorough the memory have the original source listed in the Bibliography.

Throughout all the memory there are references ([x]) to the Bibliography. These references are classified in 2 categories: in blue ([x]) are the main references, references that are basic for the elaboration and understanding of the project and of which I have took information to write the memory or to understand some element related. This references in most cases does not only appear but are furthermore explained in the paragraph or chapter with text and images. On the other hand, in

green ([x]) there are the secondary references. These references are not decisive for the elaboration and understanding of the project and many times are references that only are mentioned without further explanation, unlike blue references. But they must appear during explanations to can creating a concepts and ideas net. In addition, if the reader wants, they allow additional investigation.

Why have I decided to make all the project (memory and computer program) in English? I have taken this decision, first of all, because I have a level of written English enough to write fairly decent the memory. I am not going to do any presentation in English, so writing the memory in English is an acceptable task since it does not require to speak English, the hard part of learning languages. Second, the English is the language of the knowledge; almost all scientific literature and many technical books are, in its origin, in English. And third and as consequence of the former reasons, if I communicate in English my works will be able to arrive to more scientific audience than if I would write them in Spanish. As an *almost* engineer, I must know English, and hence, I have made use of it. Of course, I have used tools that have helped me to write a better English; beginning by the grammatical corrector of OpenOffice.org Writer, and afterward, all kind of tools to find out if what I was writing was correct: Google Translate, SYSTRANet, WordReference.com and English Wiktionary. The memory is written in American English.

To write the memory I have used OpenOffice.org Writer 3.2; it allow me to fulfill all the needs that can appear in an academic work like this, and in addition, it is cross-platform and free and open source software. Image edition has not been a important task in this project, but for some pictures and some Image Feature Detector icons I have used Corel Photo Paint X5 to edit and create them. The typeface used for the text body is URWClassico, for tittles Myriad Pro and for formulas Minion Pro.

Chapter 1
Introduction

Across this chapter we shall view the basic concepts in which this project is based. First, we shall view what is computer vision and a list with uses of this science applied to the real world. Subsequently, it will explain the digital image processing field, the *brother* discipline of computer vision and *mother* of image features detectors, the main subject of the project. Later, we shall view a brief explication about robotics and an introduction to Robolab, the University of Extremadura's robotics laboratory where this project has been developed. And finally, we shall view how the detection of image features is a valuable tool with which we can treat many problems in robotics, like to solve the localization of a robot, that is, visual odometry.

What Is Computer Vision

As humans, we perceive the three-dimensional structure of the world around us with apparent ease [1]. Think of how vivid the three-dimensional perception is when you look at a vase of flowers sitting on the table next to you. You can depict the shape and translucency of each petal through the subtle patterns of light and shading that play across its surface and effortlessly segment each flower from the background of the scene. Looking at a framed group portrait, you can easily count all of the people in the picture and even guess at their emotions from their facial appearance. Perceptual psychologists have spent decades trying to understand how the visual system works and, even though they can devise optical illusions to confirm some of its principles, a complete solution to this puzzle remains elusive.

Researchers in computer vision have been developing, in parallel, mathematical techniques for recovering the three-dimensional shape and appearance of objects in imagery. We now have reliable techniques for accurately computing a partial 3D model of an environment from thousands of partially overlapping photographs (Figure 1.1(a)). Given a large enough set of views of a particular object or facade, we can create accurate dense 3D surface models using stereo matching (Figure 1.1(b)). We can track a person moving against a complex background (Figure 1.1(c)). We can even, with moderate success, attempt to find and name all of the people in a photograph using a combination of face, clothing, and hair detection and recognition (Figure

1.1(d)). However, despite all of these advances, the dream of having a computer interpret an image at the same level as a two-year old (for example, counting all of the animals in a picture) remains elusive. Why is vision so difficult? In part, it is because vision is an *inverse problem*, in which we seek to recover some unknowns given insufficient information to fully specify the solution. We must therefore resort to physics-based and probabilistic *models* to disambiguate between potential solutions. However, modeling the visual world in all of its rich complexity is far more difficult than, say, modeling the vocal tract that produces spoken sounds.

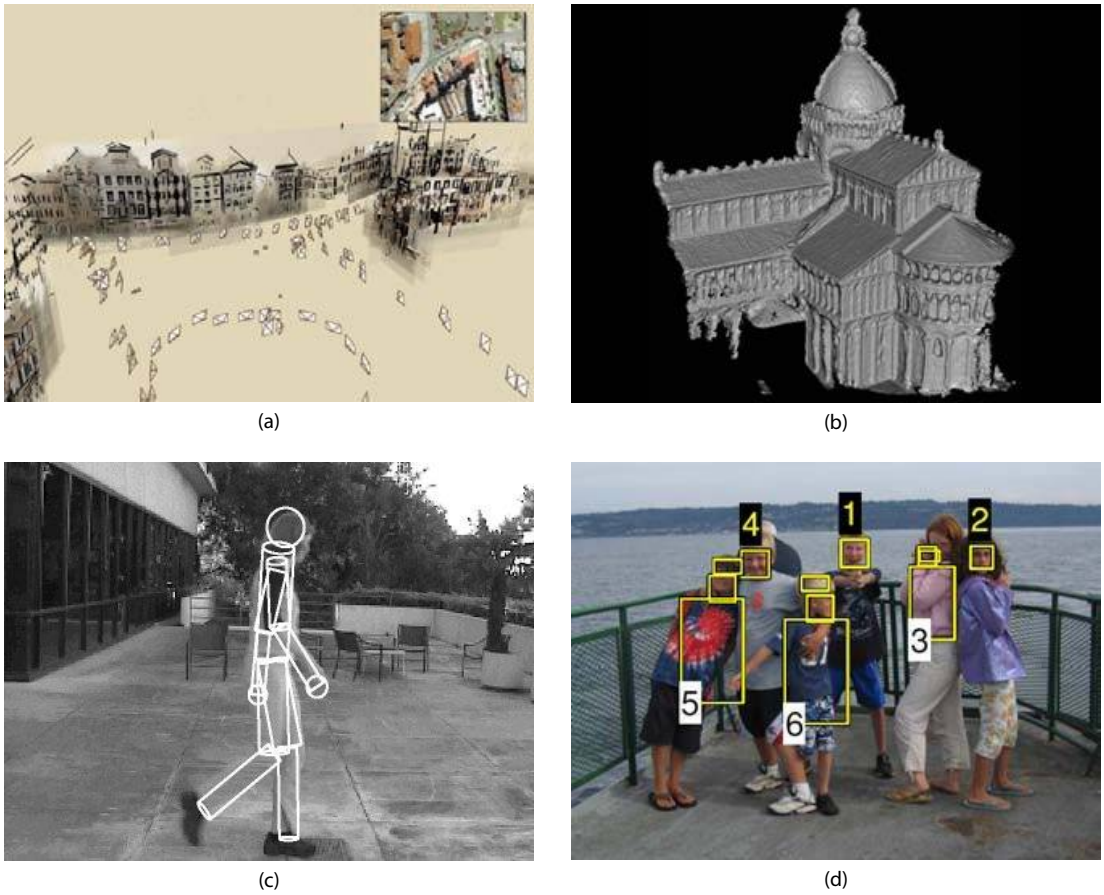


Figure 1.1: Some examples of computer vision algorithms and applications.. (a) *Structure from motion* algorithms can reconstruct a sparse 3D point model of a large complex scene from hundreds of partially overlapping photographs (Snavely, Seitz, and Szeliski 2006) © 2006 ACM. (b) *Stereo matching* algorithms can build a detailed 3D model of a building façade from hundreds of differently exposed photographs taken from the Internet (Goesele, Snavely, Curless et al. 2007) © 2007 IEEE. (c) *Person tracking* algorithms can track a person walking in front of a cluttered background (Sidenbladh, Black, and Fleet 2000). © 2000 Springer. (d) Face detection algorithms, coupled with color-based clothing and hair detection algorithms, can locate and recognize the individuals in this image (Sivic, Zitnick, and Szeliski 2006) © 2006 Springer.

The *forward* models that we use in computer vision are usually developed in physics (radiometry, optics, and sensor design) and in computer graphics. Both of these fields model how objects move and animate, how light reflects off their surfaces, is scattered by the atmosphere, refracted through camera lenses (or human eyes), and finally projected onto a flat (or curved) image plane. While computer graphics are not yet perfect (no fully computer-animated movie with human characters has yet

succeeded at crossing the *uncanny valley*[1] that separates real humans from android robots and computer-animated humans), in limited domains, such as rendering a still scene composed of everyday objects or animating extinct creatures such as dinosaurs, the illusion of reality is perfect.

In computer vision, we are trying to do the inverse, i.e., to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions. It is amazing that humans and animals do this so effortlessly, while computer vision algorithms are so error prone. People who have not worked in the field often underestimate the difficulty of the problem. This misperception that vision should be easy dates back to the early days of artificial intelligence, when it was initially believed that the cognitive (logic proving and planning) parts of intelligence were intrinsically more difficult than the perceptual components.

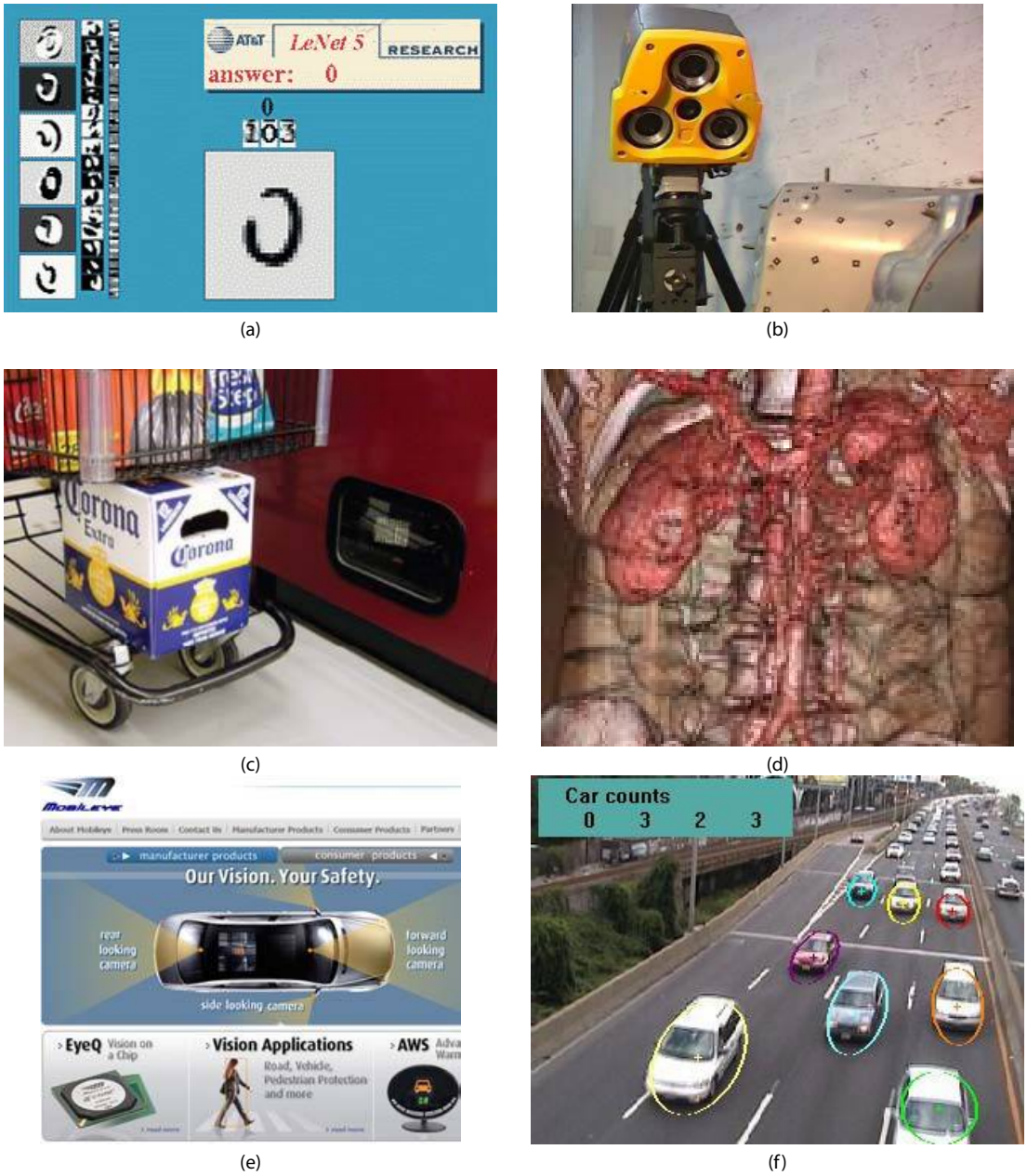
Computer vision is being used today in a wide variety of real-world applications, which include:

- **Optical character recognition (OCR)**: reading handwritten postal codes on letters (Figure 1.2(a)) and automatic number plate recognition (ANPR);
- **Machine inspection**: rapid parts inspection for quality assurance using stereo vision with specialized illumination to measure tolerances on aircraft wings or auto body parts (Figure 1.2(b)) or looking for defects in steel castings using X-ray vision;
- **Retail**: object recognition for automated checkout lanes (Figure 1.2(c));
- **3D model building (photogrammetry)**: fully automated construction of 3D models from aerial photographs used in systems such as Google Maps or Bing Maps;
- **Medical imaging**: registering pre-operative and intra-operative imagery (Figure 1.2(d)) or performing long-term studies of people's brain morphology as they age;
- **Automotive safety**: detecting unexpected obstacles such as pedestrians on the street, under conditions where active vision techniques such as radar or lidar do not work well (Figure 1.2(e); see also Miller, Campbell, Huttenlocher et al. (2008); Montemerlo, Becker, Bhat et al. (2008); Urmson, Anhalt, Bagnell et al. (2008) for examples of fully automated driving);
- **Match move**: merging computer-generated imagery (CGI) with live action footage by tracking feature points in the source video to estimate the 3D camera motion and shape of the environment. Such techniques are widely used in Hollywood (e.g., in movies such as Jurassic Park) (Roble 1999; Roble and Zafar 2009); they also require the use of precise matting to insert new elements between

1The term *uncanny valley* was originally coined by roboticist Masahiro Mori as applied to robotics (Mori 1970). It is also commonly applied to computer-animated films such as *Final Fantasy* (2001) or *Beowulf* (2007) (Geller 2008).

foreground and background elements (Chuang, Agarwala, Curless et al. 2002).

— **Motion capture**: using retro-reflective markers viewed from multiple cameras or other vision-based techniques to capture actors for computer animation;



(a) (b) (c) (d) (e) (f)

Figure 1.2: Some industrial applications of computer vision: (a) optical character recognition (OCR) http://yann.lecun.com/exdb/lenet/; (b) mechanical inspection http://www.cognitens.com/; (c) retail http://www.evoretail.com/; (d) medical imaging http://www.clarontech.com/; (e) automotive safety http://www.mobileye.com/; (f) surveillance and traffic monitoring http://www.honeywellvideo.com/, courtesy of Honeywell International Inc.

— **Surveillance**: monitoring for intruders, analyzing highway traffic (Figure 1.2(f)), and monitoring pools for drowning victims;

— **Fingerprint recognition and biometrics**: for automatic access authentication as well as forensic applications.

Computer vision is at an extraordinary point in its development [2]. The subject itself has been latent since the 1960s, but it is only recently that it has been possible to build useful computer systems using ideas from computer vision. This raising has been driven by several trends: Computers and imaging systems have become very cheap. Not all that long ago, it took tens of thousands of dollars to get good digital color images; now it takes a few hundred, at most. Not all that long ago, a color printer was something one found in few, if any, research labs; now they are in many homes. This means it is easier to do research. It also means that there are many people with problems to which the methods of computer vision apply. Our understanding of the basic geometry and physics underlying vision and, what is more important, what to do about it, has improved significantly. We are beginning to be able to solve problems that lots of people care about, but none of the hard problems have been solved and there are plenty of easy ones that have not been solved either (to keep one intellectually fit while trying to solve hard problems). It is a great time to be studying this subject.

What Is Digital Image Processing

An image may be defined as a two-dimensional function, *f(x,y)*, where *x* and *y* are *spatial* (plane) coordinates, and the amplitude of *f* at any pair of coordinates (*x, y*) is called the *intensity* or *gray level* of the image at that point [3]. When *x, y*, and the intensity values of *f* are all finite, discrete quantities, we call the image a digital image. The field of *digital image processing* refers to processing digital images by means of a digital computer. Note that a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are called *pixels*.

Vision is the most advanced of our senses, so it is not surprising that images play the single most important role in human perception. However, unlike humans, who are limited to the visual band of the electromagnetic (EM) spectrum, imaging machines cover almost the entire EM spectrum, ranging from gamma to radio waves. They can operate on images generated by sources that humans are not accustomed to associating with images. These include ultrasound, electron microscopy, and computer-generated images. Thus, digital image processing encompasses a wide and varied field of applications.

There is no general agreement among authors regarding where image processing stops and other related areas, such as image analysis and computer vision, start. Sometimes a distinction is made by defining image processing as a discipline in which both the input and output of a process are images. We believe this to be a limiting and somewhat artificial boundary. For example, under this definition, even the trivial task of computing the average intensity of an image (which yields a single number) would not be considered an image processing operation. On the other hand, there are fields such as computer vision whose ultimate goal is to use computers to emulate human vision, including learning and being able to make inferences and take actions based on visual inputs. This area itself is a branch of artificial intelligence (AI) whose objective is

to emulate human intelligence. The field of AI is in its earliest stages of infancy in terms of development, with progress having been much slower than originally anticipated. The area of image analysis (also called image understanding) is in between image processing and computer vision.

There are no clear-cut boundaries in the continuum from image processing at one end to computer vision at the other. However, one useful paradigm is to consider three types of computerized processes in this continuum: low-, mid-, and high-level processes. Low-level processes involve primitive operations such as image preprocessing to reduce noise, contrast enhancement, and image sharpening. A low-level process is characterized by the fact that both its inputs and outputs are images. Mid-level processing on images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterized by the fact that its inputs generally are images, but its outputs are attributes extracted from those images (e.g., edges, contours, and the identity of individual objects). Finally, higher-level processing involves “making sense” of an ensemble of recognized objects, as in image analysis, and, at the far end of the continuum, performing the cognitive functions normally associated with vision.

Based on the preceding comments, we see that a logical place of overlap between image processing and image analysis is the area of recognition of individual regions or objects in an image. Thus, what we call *digital image processing* encompasses processes whose inputs and outputs are images and, in addition, encompasses processes that extract attributes from images, up to and including the recognition of individual objects. As an illustration to clarify these concepts, consider the area of automated analysis of text. The processes of acquiring an image of the area containing the text, preprocessing that image, extracting (segmenting) the individual characters, describing the characters in a form suitable for computer processing, and recognizing those individual characters are in the scope of what we call *digital image processing*. Making sense of the content of the page may be viewed as being in the domain of image analysis and even computer vision, depending on the level of complexity implied by the statement “making sense”.

As will become evident shortly, digital image processing, as we have defined it, is used successfully in a broad range of areas of exceptional social and economic value.

Robotics and Robolab

Based on the Robotics Institute of America (RIA) definition: "A robot is a reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks”.

From the engineering point of view [4], robots are complex, versatile devices that contain a mechanical structure, a sensory system, and an automatic control system.

Theoretical fundamentals of robotics rely on the results of research in mechanics, electric, electronics, automatic control, mathematics, and computer sciences.

This project is aimed to introduce and compare image features detectors to be used in visual odometry in mobile robots. But actually, it is a general purpose project which the application of the results is only restricted to any system with Linux, even not being a robot of any kind. In addition, the uses of image feature detectors has many applications more than visual odometry: object recognition, object count or biometrics, among others uses.

Although the subject of the project is not related directly with robotics, the academic context where this project is developing does: Robolab is the laboratory where the project tutor Pedro Núñez currently works and where the same approach explained in the project is used in their robots.

Robolab

The Robolab laboratory [42] is the Robotics and Artificial Vision Laboratory, located at the Polytechnic College of the University of Extremadura, Cáceres. Since its foundation in 1999, it is devoted to conduct research in Intelligent Mobile Robotics and Computer Vision.

Figure 1.3: The three Robolabs robots using the RoboEx platform. The central robot, in addition to the rest of devices, has mounted a forklift

The laboratory has issued diverse researching publications. Recently, they have published two works which are the base for future researching developments: *RobEx: an open-hardware robotics platform* [17] and *RoboComp: a Tool-based Robotics Framework* [18]. These works introduce a global platform for mobile robots, including hardware and software specifications. The first work explains the *RobEx* platform (Figure 1.3), a little metallic base with wheels where a laptop, a head with stereo cameras, and another devices like forklift (like that of the central robot of Figure 1.4),

lasers or claws can be mounted. The second work explains the *RoboComp* component-oriented robotics software framework. This framework boosts distributed software development and collaboration by providing an easy method of integrating components made by different RoboComp users. It also provides a set of tools that makes component creation and management an easy task. It is the software that controls the robots introduced in RoboEx work.

Many of the experiments took out in the laboratory have been realized thanks to the three robots constructed by the own laboratory that uses the RoboEx platform (Figure 1.4). These robots can operate in manual mode throughout a joystick or in automatic mode, where the robots can detect obstacles and dodge them. The skill to dodge obstacles is one of the current researching works.

Visual Odometry

Odometry is the use of data from diverse kind of sensors (usually rotary encoders linked to wheels, lasers or cameras) to estimate change in position over time [50]. It is used by robots, whether they be legged or wheeled, to estimate approximately their position relative to a starting location. This method is very sensitive to errors due to the integration of velocity measurements over time to give position estimates. Rapid and accurate data collection, equipment calibration, and processing are required in most cases for odometry to be used effectively.

When the information comes from cameras and images are used to do the spacial estimation instead of wheels or another devices, we are speaking about *visual odometry*. Visual odometry [16] is the process of determining a visual sensor orientation and position in 3D space from a sequence of images, or simply put, motion estimation using visual information only. To us, humans, as well as many other living beings, our perception system provides the main sensory input for navigation purposes. For example, it has been shown that honey bees [19] use optical flow [20] as an essential navigation aid. It is not only that visual inputs are naturally suitable for navigation, but also visual inputs allows a variety of useful tasks. For example, besides navigating the environment the robot can generate a 3D reconstruction, detect objects of interest, classify the terrain, etc. All of those tasks can performed with low-cost and low-power consumption, which is ideal for robotics platforms.

In Robotics research, the use of visual input for navigation purposes started late in the 1970's ([21]). Among the first uses of cameras for mobile robot navigation can be traced back to Moravec's [22], who used several cameras to navigate a robotic cart in a room. However, the use of vision in mobile robotics has been hindered by the limited computational power. Typical image processing and understanding tasks require much computational power due to the amount of data in images, which was not available until recent advances in hardware Computer Vision algorithms.

Nowadays, visual odometry is attracting much attention in the Robotics and Computer Vision communities. Several real-time visual odometry implementations

have been reported and results are very promising. However, much work remains to be done in this area. Several improvements and enhancements could be added to current systems to allow them to scale for very large terrains. The main issue is the ability to deal with unbounded accumulated error induced by the iterative nature of motion estimation and exaggerated with the large amount of data in images and the associated noise.

One of the main approaches used to acquire images and use them in visual odometry is through stereo cameras, like the robots used by RoboLab (Figure 1.4). In stereo vision [48], two cameras, displaced horizontally from one another are used to obtain differing views on a scene, in a manner similar to human binocular vision. The cameras are then modeled as a perspective view whereby the two cameras will see slightly different projections of the world view. By comparing these two images, the relative depth information can be obtained, in the form of a disparity map, which is inversely proportional to the distance to the object.

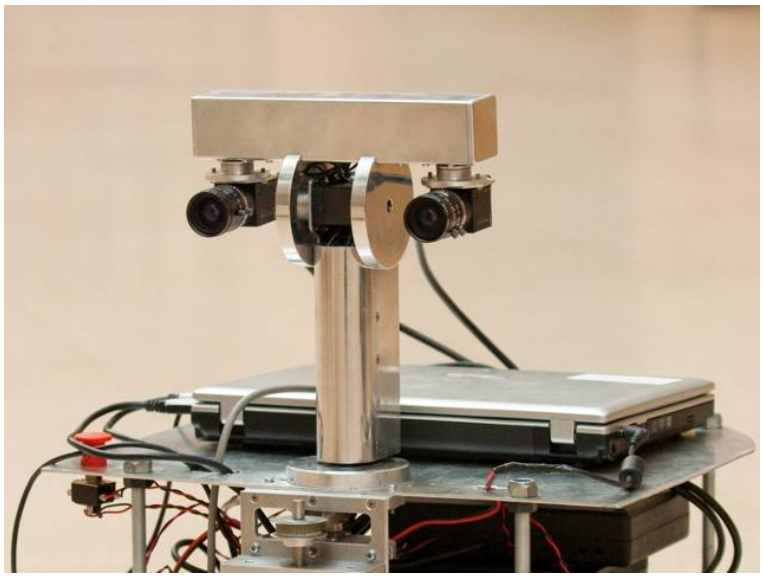


Figure 1.4: RoboLab robots use stereo cameras to estimate its position through visual odometry (among others approaches).

To compare the images, the two views must be transformed as if there were being observed from a common projective camera, this can be achieved, by projecting the right camera to the left camera's position or viceversa, and the relative shifts between the two images can then be seen to be due to parallax. Alternately both camera views may be transformed to an arbitrary location in 3D space, as long as the front face of the images to be compared is visible from this location, and that occlusion or transparency does not interfere with the calculation.

Stereo vision finds many applications in automated systems. Stereo vision is highly important in fields such as robotics, to extract information about the relative position of 3D objects in the vicinity of autonomous systems. Other applications for robotics include object recognition, where depth information allows for the system to separate

occluding image components, such as one chair in front of another, which the robot may otherwise not be able to distinguish as a separate object by any other criteria.

Template matching

Template matching [53] is a technique in digital image processing to find parts of an image which match a template image in another image. It has many uses on computer vision and we are interested in used it to compare detected features in images and use the result as input data for the visual odometry of mobile robots. This use is not implemented in the project since it is out of its main scope, but the SIFT and SURF detectors than are introduced in Chapter 2 and compared in Chapter 3 generate descriptors than are ready to be compared in a template matching way.

Template matching can be subdivided in two approaches: template-based and feature-based matching. The template-based, or global approach, uses the entire template, with generally a sum-comparing metric (using SAD, SSD, cross-correlation, etc.) and determines the best location by testing all or a sample of the candidate locations within the search image. This approach is only useful when the searched object in a image is almost the same than the template since template-base matching is very intolerant to changes. On the other hand, the feature-based approach uses the image features, such as edges or corners, as the primary match-measuring metrics to find the best matching location of the template in the source image. In our case, we are interested in the second approach, in which template matching is used in SIFT and SURF detectors to find similar features in other images by comparing the descriptors of detected features. In the next Chapter 2 is described what is a descriptor.

Chapter 2
Image Features, Descriptors and Detectors

In this chapter we shall see the concept of image feature, an useful piece of information extracted from an image which it let us make understand to the computer how humans see, and then, to make the computer recognize and count objects and process image information in a useful way as a human would do it, but with the speed of computers and some other advantages.

We shall see the different kinds of features and how to obtain and filter between potentials interesting features and residual or not time-perdurable features. Later, we discuss the ways to show extracted features in a graphical way. We can do it whether as a whole transformed image where only interesting parts are visible or outlined, or as an image with marked points with arrows where the arrow's size -the magnitude- indicates the size or importance of that point.

Finally, we shall view what is a feature detector and shall review the most important/used detectors with a shallow mathematical explication. These feature detectors will be compared and tested in the Chapter 3, and they are implemented in the Chapter 4, where the computer program Image Featured Detector is showed and explained.

What Is a Feature

When we have an image, that is, a set of pixels with a given intensity value, the concept of feature is used to denote a set of pixels which is relevant in some way from a human view point. They describe the elementary characteristics of the image such as the shape, the color, the texture or the motion, although we will be mainly interested in the shape. Edges, borders and other points which are non-or-little variant features to be tracked in an image sequence will be the worth features for us. Then, these extracted features, as processed data that has become useful information, can be used for solving different tasks or problems. Most of these problems where image features are used to solve them are object recognition by template matching and object count.

Let's fix in the image of Figure 2.1. On it we can see buildings, cars, a cross shape monument in the middle, two traffic lights at both sides of the image and a red bus at

the right. The most of the shapes that we can see in the image are edges -mainly straight lines and some curves-, corners and some objects with a non very well defined shape or very complex one -trees-. This delimiters shapes are filled by surfaces with textures -the asphalt and the wall of the buildings-. These edges, corners and, in lesser extent, textures are the basic pieces that we can use to depict the image in a objective way. Hence, the features of the image are all these shapes described above that we shall consider with useful information to define an image. The color is another property that we can take into account to extract features, but due to the computational cost of using color images, in most cases by using grayscale images is enough to extract a good feature sample set.

Figure 2.1: Illustrative image in which we can see different shapes.

The complexity of these features can vary from just simple points –the result of a general neighborhood operation– to more elaborated shapes where different basic points are used to create a new feature –specific structures in the image itself such as borders or edges or more complex structures such as objects–.

The feature concept is very general and the choice of a right feature detector in a particular computer vision system may be highly dependent on the specific problem at hand. Likewise, the correct adjusts of feature parameters can be a critical issue. In some cases, a higher level of detail in the description of a feature may be necessary for solving the problem, but this comes at the cost of having to deal with more data and more demanding processing.

What Is a Descriptor

Some features detectors are ready to use the output information that generate to searching similar detected features in other images. This is the case of SIFT and SURF detectors commented in the next subchapter, but also that of other detectors less efficient like LESH (Local Energy based Shape Histogram, [40]) or GLOH (Gradient Location and Orientation Histogram, [41]).

When we works with detectors that generate descriptors [1], after detecting features, we must match them, i.e., we must determine which features come from corresponding locations in different images. In some situations, as for video sequences or for stereo pairs that have been rectified, the local motion around each feature point may be mostly translational. In this case, simple error metrics, such as the sum of squared differences or normalized cross-correlation can be used to directly compare the intensities in small patches around each feature point. (The comparative study by Mikolajczyk and Schmid (2005), uses cross-correlation). Because feature points may not be exactly located, a more accurate matching score can be computed by performing incremental motion refinement, but this can be time consuming and can sometimes even decrease performance (Brown, Szeliski, and Winder 2005).

In most cases, however, the local appearance of features will change in orientation and scale, and sometimes even undergo affine deformations. Extracting a local scale, orientation, or affine frame estimate and then using this to resample the patch before forming the feature descriptor is thus usually preferable.

Even after compensating for these changes, the local appearance of image patches will usually still vary from image to image. The accuracy of each detector matching descriptors will determinate their effectiveness and efficiency. We will compare this efficiency with SIFT and SURF descriptors in Chapter 3.

Visual descriptors

When we speak about descriptor we can also refer to the visual representation of the data output of what the detector has considered useful information basing on the parameters we have set.

Features detectors carry out local neighborhood operations into an image. Then, we can differentiate detectors depend on whether the feature detector produces as output: they can return a new image with the same dimensions (Figure 2.2(a)) that can acts as a mask, or they produce a set of pointers with information –usually vectors– (Figure 2.2(b)) situated in determinate pixels of the image, instead of a new image.

The first ones are called *feature image descriptors* because the output shows the detected features like an image mask (this is the case of the OpenCV Harris implementation used in the computer program of Chapter 4). And the second ones are called *vector descriptors* or simply *descriptors* since they are a set of points indicating and interesting region into the image that additionally some kinds of detectors add a vector with their corresponding magnitude and a angle.

In the case of vector descriptors, the feature detector as a result produces *interesting points* situated in determinate pixels of the image. Although local decisions are made, the output from a feature detection step does not need to be a binary image. The result is often represented in coordinates sets (connected or unconnected) of the image points where features have been detected. In addition, some detectors calculate extra

information in form of vector with start point the coordinates of the feature point. The FAST detector generates a set of coordinates without any additional information.

Another use of vectors in feature descriptors is to indicate the confidence of the detected feature. As they are marked with an *importance label*, in later data processing we could choice the features with high confidence value. This enables that a new feature descriptor being computed from several descriptors, computed at the same image point but at different scales, or from different but neighboring points, in terms of a weighted average where the weights are derived from the corresponding confidences. These kind of feature descriptors are use by the SIFT and SURF detectors.

(a) (b)

Figure2.2 (a) Output image showing detected corners by the Harris OpenCV implementation. (b). Image with superimposed vector descriptors calculated by the OpenCV SURF detector implementation indicating the magnitude and direction of the feature points.

Main Detectors Overview

The last main concept that appears in this chapter is that of feature detector. As its name indicates, the detector is a mathematical algorithm that in a discrete image signal finds and isolates some determinate feature as it could be edges, corners or time-invariable zones to be tracked in image sequences. The response of the algorithm will give a new image or a set of points indicating where the detected features are situated into the image, as we saw in the previous section. We will see 6 feature detectors: Robert cross, Sobel, Harris (and Shi and Tomasi), FAST, SIFT and SURF. They are chosen by its efficacy or by its base use in others detectors. Although the Image Feature Detector computer program that we shall view in Chapter 4 only implements

the last 4 detectors, the Robert cross and Sobel ones are the base of Harris detector (and edge detection in general); for that reason are explained in the chapter.

Throughout the evolution of the computer vision science several approaches to detect features have been developed. Essentially, the boundary of an object is a step-change in the intensity levels of each pixel. By basing on this promise, the simplest feature detector that we could develop would be a edge detector based on this intensity level change between pixels. We will consider the edge is at the position of the step-change. Then, to detect the edge position we can use first-order differentiation since as we are searching changes in the intensity, first-order differentiation gives a proportional response to the change and no response when the signal does not alter.

A change in intensity can be revealed by differencing adjacent points. Differencing horizontally adjacent points will detect vertical changes in intensity and is often called a horizontal edge detector. From now, as we will define different differencing mapping from one vector space to another, we will refer to these transformations as operators. A horizontal operator will not show horizontal changes in intensity since the difference is zero. As such, the gradient along the line normal to the edge slope can be computed in terms of the derivatives along orthogonal axes:

$$G(x,y)=\frac{\partial P(x,y)}{\partial x}\cos\theta+\frac{\partial P(x,y)}{\partial y}\sin\theta \tag{2.1}$$

The equation above describes the generation of an edge gradient in terms of a row gradient and a column gradient. This expressed as a discrete signal and applied to an image P the action of the horizontal edge detector gives the difference between two horizontally adjacent points, detecting the vertical edges Ex as:

$$\mathbf{Ex}_{x,y} = \left|\mathbf{P}_{x,y} - \mathbf{P}_{x+1,y}\right| \ \forall x \in 1, N-1; y \in 1, N \tag{2.2}$$

In order to detect horizontal edges we need a vertical edge detector which differences vertically adjacent points. This will determine horizontal intensity changes, but not vertical ones, so the vertical edge detector detects the horizontal edges, Ey, according to:

$$\mathbf{Ey}_{x,y} = \left|\mathbf{P}_{x,y} - \mathbf{P}_{x,y+1}\right| \ \forall x \in 1, N; \mathrm{y} \in 1, N-1 \tag{2.3}$$

Figures 2.3(b) and (c) show the application of the vertical and horizontal operators to the image of the square in Figure 2.3(a). Note that in Figure 2.3(b) the left-hand vertical edge appears to be beside the square as result of the differencing process. This is thus because we consider the edge is at the position of the step-change. Then, the result is showed in the previous pixel before the step-change. Likewise, the upper edge in Figure 2.3(b) appears above the original square.

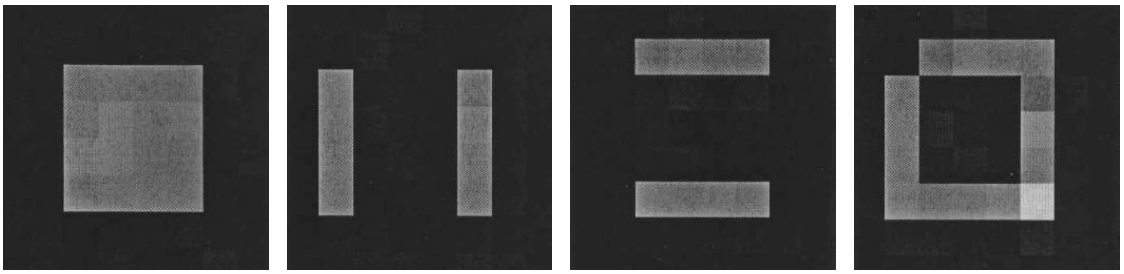


Figure2.3: First-order edge detection.

Combining the two detectors gives an operator E that can detect both vertical and horizontal edges:

$$\mathbf{E}_{x,y} = \left|\mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x,y+1}\right| \; \forall x, y \in 1, N-1 \Leftrightarrow$$
$$\mathbf{E}_{x,y} = \left|2 \times \mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y+1}\right| \; \forall x, y \in 1, N-1 \qquad (2.4)$$

Equation 2.3 gives the coefficients of the differencing matrix that appears in Figure 2.4, which can be convolved with an image to detect all the edge points, although in this case the algebraic approach is easier. Note that the bright point in the lower right corner of the edges of the square in Figure 2.3(d) is brighter than the other points. This is because it is the only point to be detected as an edge by both the vertical and the horizontal operators and is therefore brighter than the other edge points. In contrast, the top left hand corner point is detected by neither operator and so does not appear in the final image.

$$M = \begin{bmatrix} -2 & -1 \\ -1 & 0 \end{bmatrix}$$

Figure2.4: Matrix for first-order operator.

There are several edge operators based on first-order difference that often appear on literature. First, as an introduction mode we shall slightly view the Robert cross and Sobel operators. Afterwards, we shall view the interesting detectors for the project. We will begin with Harris; it is based in the Sobel operator. FAST, SIFT and SURF detectors use another techniques furthermore than differentiation. For that, they are a little different in relation to Harris and previous detectors.

The Roberts cross and Sobel detectors mainly detect edges. Harris, in addition, detects corners. FAST only detect corners and SIFT and SURF detect any feature in an image susceptible of being a good point to be searched in another scales or variations of the image.

Roberts Cross Operator

The Roberts cross operator, released by Lawrence G. Roberts in 1963 in the work *Machine Perception Of Three-Dimensional Solids* [7], was one of the earliest edge detection operators. It implements a version of basic first-order edge detection and uses two matrices which difference pixel values in a diagonal manner, as opposed to along the axes directions. The two matrices are called M^+ and M^- and are given in Figure 2.5.

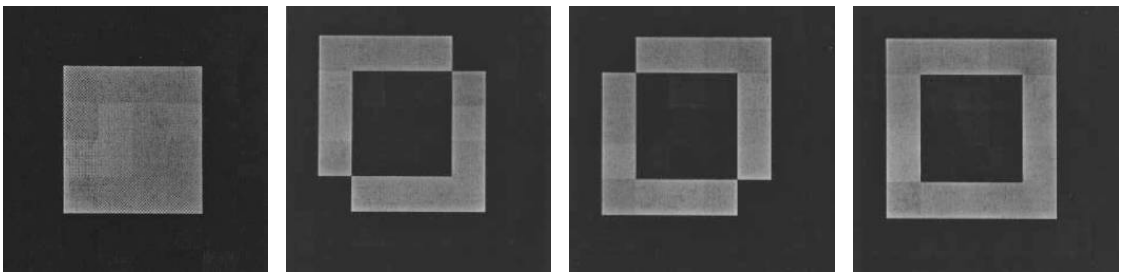
$$\mathbf{E}_{x,y} = \max\{|M^+ * \mathbf{P}_{x,y}|, |M^- * \mathbf{P}_{x,y}|\} \; \forall x, y \in 1, N-1 \tag{2.5}$$

In implementation, the maximum value delivered by application of these matrices is stored as the value of the edge at that point. The edge point $E_{x,y}$ is then the maximum of the two values derived by convolving the two matrices at an image point $P_{x,y}$:

$$M^+ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad M^- = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Figure2.5: Matrices for Roberts cross operator.

The application of the Roberts cross operator to the image of the square is shown in Figure 2.6. The two matrices provide the results in Figures 2.6(b) and (c) and the result delivered by the Roberts operator is shown in Figure 2.6(d). Note that the corners of the square now appear in the edge image, as result of the diagonal differencing operation, whereas they were less visible in Figure 2.6(d) (where the top left corner did not appear).



(a) (b) (c) (d)

Figure2.6: Steps of applying Roberts cross operator.

An alternative to taking the maximum is simply to add the results of the two matrices together to combine horizontal and vertical edges. There are, of course, more varieties of edges and it is often better to consider the two matrices as providing components of an edge vector: the strength of the edge along the horizontal and vertical axes. These give components of a vector and can be added in a vectorial manner (which is perhaps more usual for the Roberts operator). The edge magnitude is

the length of the vector and the edge direction is the vector's orientation, as shown in Figure 2.7.

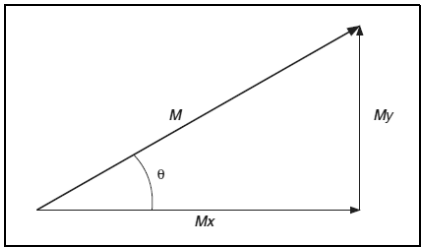


Figure 2.7: Roberts cross operator represented as a vector

Sobel Operator

Everytime we are going to carry out any edge detection we will have to do differentiation operations. Since they detects any change, they will also detect noise changes, as well as any kind of step-like changes in image intensity. It is hence a good idea to add averaging within the edge detection operations. We can then extend the vertical matrix, M_x, along three rows, and the horizontal matrix, M_y, along three columns, as it is showed in Figure 2.8. This first step of doing a larger differentiation is called Prewitt edge detection operator [8] and we will use it as a base to the Sobel operator, operator released by Irwin Sobel in 1968 [9].

$$M_x=\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix} \qquad M_y=\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix}$$

Figure2.8: Matrices for first order differentiation in 3×3 windows.

With this we get the rate of intensity change along each axis. As you can remember, this is the vector illustrated in Figure 2.7: the edge magnitude M is the length of the vector and the edge direction θ is the angle of the vector:

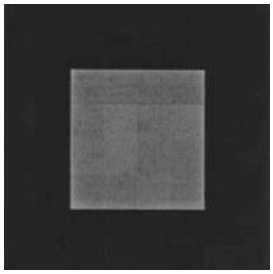
$$M=\sqrt{M_x(x,y)^2+M_y(x,y)^2} \tag{2.6}$$

$$\theta(x,y)=\arctan\left(\frac{M_y(x,y)}{M_x(x,y)}\right) \tag{2.7}$$

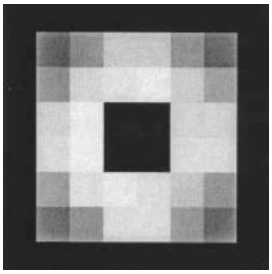
When applied to the image of the square Figure 2.9(a), we obtain the edge magnitude and direction, Figures 2.9(b) and (d), respectively (where (d) does not include the border points, only the edge direction at processed points). The edge direction in Figure 2.9(d) is shown measured in degrees where 0° and 360° are

horizontal, to the right, and 90° is vertical, upwards. Though the regions of edge points are wider due to the operator's averaging properties, the edge data is clearer than the earlier first-order operator, highlighting the regions where intensity changed in a more reliable form (compare, for example, the upper left corner of the square which was not revealed earlier). The vector representation of the Figure 2.9(c) shows that the edge direction data is clearly less well defined at the corners of the square (as expected, since the first-order derivative is discontinuous at these points).

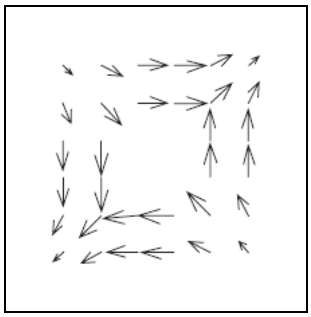
If we double the weight at the central pixels for both horizontal and vertical Prewitt matrices, it gives the Sobel edge detection operator which, again, consists of two masks to determine the edge in vector form. The Sobel operator was the most popular edge detection operator until the development of edge detection techniques with a theoretical basis. It become popular because it gave, overall, a better performance than other contemporaneous edge detection operators, such as the Prewitt operator.



(a)



(b)



(c)

dir =

313	331	3	3	24	47
298	315	1	2	42	63
273	276	13	43	88	88
269	268	199	117	91	92
242	225	181	178	133	116
225	210	183	179	155	132

(d)

Figure2.9: Result (b) of applying Prewitt operator to the original image (a) and the vector representation (c) with its angle value in degrees (d).

The coefficients for the Sobel operator for a window of 3×3 are those of the Figure 2.10:

$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure2.10: Matrices for Sobel operator for 3×3 windows.

The window size of 3×3 is the standard formulation for the Sobel matrices, although is also possible to form larger window sizes of 5×5 or 7×7, but it requires further calculations to find out the values of the new matrices. In the bibliography there are books where you could find a thoroughly explanation.

A larger edge detection matrix involves more smoothing and reduce noise but as a drawback edges loss its definition and become blurry. If we are not interested in the edge definition, the estimate of edge direction can be improved with more smoothing, since this one is particularly sensitive to noise.

Since the Prewitt operator allows that edge direction data can be arranged to point in different ways, we can do the same with the Sobel operator. If the matrices are inverted, the edge direction will be inverted in both axes. If only one of the matrices is inverted, then the measured edge direction will be inverted in such axis. This gives four possible directions for measurement of the edge direction provided by the Sobel operator. The figures illustrated in Figures 2.11(a) and (b) show the vectors with the matrices inverted. On the other hand, by swapping the Sobel templates, the measured edge direction can be arranged to be normal to the edge itself as opposed to tangential data along the edge, as Figures 2.11(c) and (d) show.

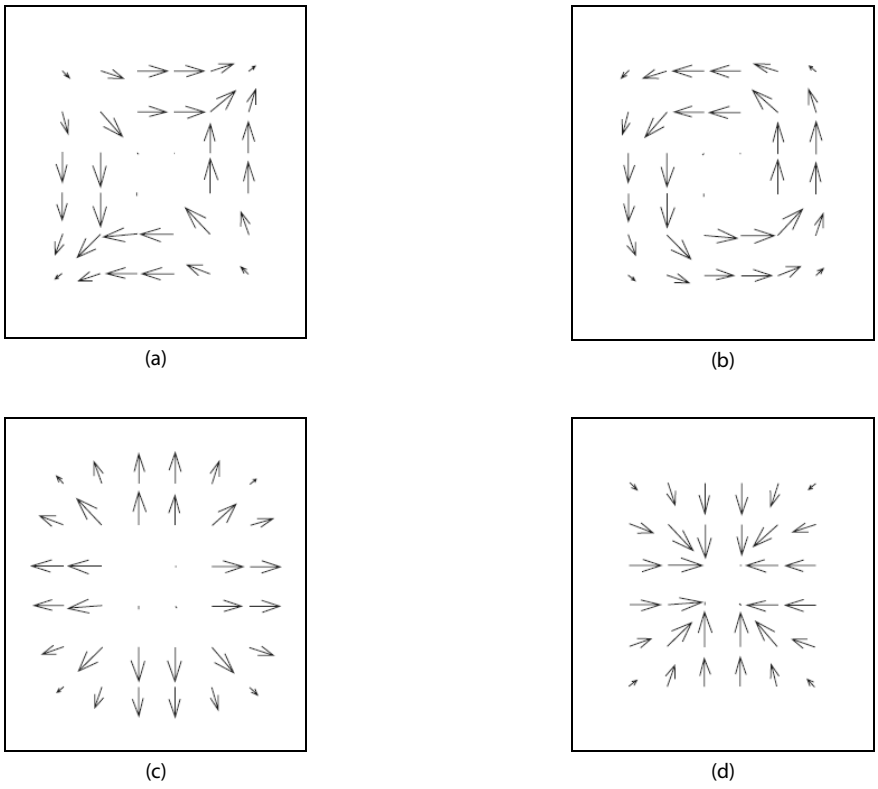


Figure2.11: Different arrangements of the edge directions.

Harris Detector

Harris is an edge and corner detector released by Chris Harris and Mike Stephens in 1988 in the work *A Combined Corner And Edge Detector* [10]. The detector tries to solve link problems between points into textures and surfaces where edge limits are not well defined, as it happens with another detector as Canny [39] or previous ones. Therefore, it proposes not consider all the image features as edges and treating problematic surfaces as features points.

Moravec Revisited

Harris operator takes advantage of Moravec corner detector [22]. This last detector works by considering a local window in the image, and determining the average changes of image intensity that result from shifting the window by a small amount in various directions. Mathematically, we can express this change E as the sum of squared differences (SSD) between the original window and the shifted by (x,y) one:

$$E_{x,y} = \sum_{u,v} w_{u,v} \left| I_{x+u,y+v} - I_{u,v} \right|^2$$

where w is the a rectangular image window (1 into the window and 0 outside). The shifts (x,y) that are evaluated are (1,0), (1,1), (0,1) and (-1,1), that is, horizontal and vertical axis and its respective diagonals. When we have calculate all SSD of a point in the 4 directions we will find a feature of interest if the minimal calculated SSD of that point is a local maximum in the image above some threshold value.

Based on this premise we will carry out some changes to improve the detector:

1. The response is anisotropic because only a discrete set of shifts at every 45 degrees is considered. By performing an analytic expansion about the shift origin we can cover all possible shifts:

$$E_{x,y} = \sum_{u,v} w_{u,v} \left| I_{x+u,y+v} - I_{u,v} \right|^2 = \sum_{u,v} w_{u,v} \left| xX + yY + O(x^2, y^2) \right|^2$$

where the first gradients are approximated by

$$X = I * (-1,0,1) \approx \partial I / \partial x = I_{\partial x}$$

$$Y = I * (-1,0,1)^T \approx \partial I / \partial y = I_{\partial y}$$

Hence, for small shifts, E can be written

$$E(x,y) = Ax^2 + 2\mathrm{C}xy + By^2$$

where

$$A = X^2 * w$$

$$B = Y^2 * w$$

$$C = (XY) * w$$

2. The response is noisy because the window is binary and rectangular. Using a smooth circular window, for example a Gaussian, we can improve the response:

$$w_{u,v} = \exp -(u^2 + v^2)/2\sigma^2$$

3. The operator responds too early to edges because only the minimum of E is taken into account. Reformulating the corner measure to make use of the variation of E with the direction of shift resolves this.

We can express the change E for the shift (x,y) as a matrix multiplication:

$$E(x, y) = \begin{bmatrix} x & y \end{bmatrix} M^T$$

where the 2×2 symmetric matrix M is

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} X^2 * w & (XY) * w \\ (XY) * w & Y^2 * w \end{bmatrix}$$

Note that E is closely related to the local autocorrelation function, with M describing its shape at the origin (explicitly, the quadratic terms in the Taylor expansion). Let α, β be the eigenvalues of M. α and β will be proportional to the principal curvatures of the local auto-correlation function, and form a rotationally invariant description of M. There are three cases to be considered:

A. If both curvatures are small, that is, $\alpha \approx 0$ and $\beta \approx 0$, so that the local autocorrelation function is flat, then the windowed image region is of approximately constant intensity (ie. arbitrary shifts of the image patch cause little change in E);

B. If one curvature is high and the other low, that is, $\alpha \approx 0$ or $\beta \approx 0$ and the other is high, so that the local auto-correlation function is ridge shaped, then only shifts along the ridge (ie. along the edge) cause little change in E: this indicates an edge;

C. If both curvatures are high, so that the local autocorrelation function is sharply peaked, then shifts in any direction will increase E: this indicates a corner.

Corner/edge Response Function

We do not only need corner and edge classification regions, but also a measure of corner and edge quality or response. The size of the response will be used to select isolated corner pixels and to thin the edge pixels.

Let us first consider the measure of corner response, R, which we require to be a function of α and β alone, on grounds of rotational invariance. It is attractive to use trace of M, Tr(M), and determinant of M, Det(M), in the formulation as this avoids the explicit eigenvalue decomposition of M, thus

$$Tr(M) = \alpha + \beta = A + B$$

$$Det(M) = \alpha\beta = AB - C^2$$

Consider the following formulation for the corner response

$$R = \alpha\beta - k(\alpha + \beta)^2 = Det(M) - k\ Tr^2(M)$$

Contours of constant R are shown by the fine lines in Figure 2.12. k is an arbitrary value that usually takes 0.04. R is positive in the corner region, negative in the edge regions, and small in hit flat region. Note that increasing the contrast (i.e. moving radially away from the origin) in all cases increases the magnitude of the response. The flat region is specified by Tr falling below some selected threshold.

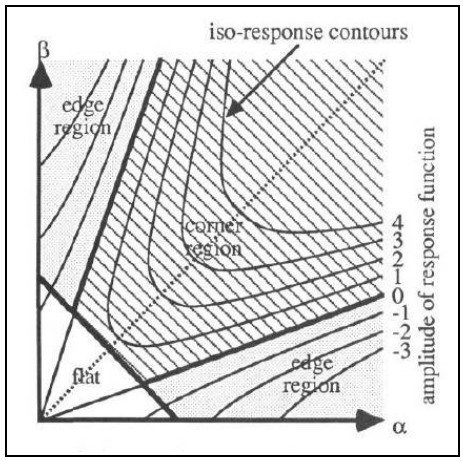


Figure 2.12. Auto-correlation principal curvature spaceheavy lines give corner/edge/flat classification, fine lines are equiresponse contours.

R is the value that the Harris implementation of the Image Feature Detector will show when the Harris operator is applied to an image.

Shi and Tomasi Detector

The Shi and Tomasi corner detector is similar to Harris but with one difference that in some cases can be improve the results. It was release by Jianbo Shi and Carlo Tomasi in 1994 in the work *Good Features to Track* [11]. The authors showed that is sometimes more reliable to use the smallest eigenvalue of M, that is, min(α, β), as the corner strength function than the R corner response.

FAST

FAST (Features from Accelerated Segment Test) is a corner detection algorithm released by Edward Rosten and Tom Drummond in 2006 in the work *Machine learning for high-speed corner detection* [14]. This detector is reported to produce very stable features, as we will view below.

FAST is a kind of corner detector that is arrange within the AST (Accelerated Segment Test) category [49]. AST is a modified version of the SUSAN corner criterion [23]. This last one states that in an image we can find features by segmenting image regions in circles of radio *r* around a determinate pixel *p* and evaluating whether it is a feature depending on the intensity *I* of the adjacent pixels in comparison with the intensity I_c of the center one. AST states that the pixels to be evaluated are only the content pixels in a Bresenham circle, unlike the SUSAN detector in which all the pixels of the circle are evaluated.

Therefore, FAST considers a feature in a determinate pixel *p* if *n* contiguous pixels are all brighter than the nucleus by at least *t* or all darker than the nucleus by *t*. Larger radiuses of the test mask tend to be more noise and blur resistant while they are more computationally expensive. The choice of the order in which each pixels of the circumference are tested is decisive. This order is a so called Twenty Questions problem (name taken from an American TV parlor game), that is, by arranging the pixel kind by building short decision trees in a way that will split the number of kinds roughly in half or third each time. In this way, FAST in the most computationally efficient corner detector available so far.

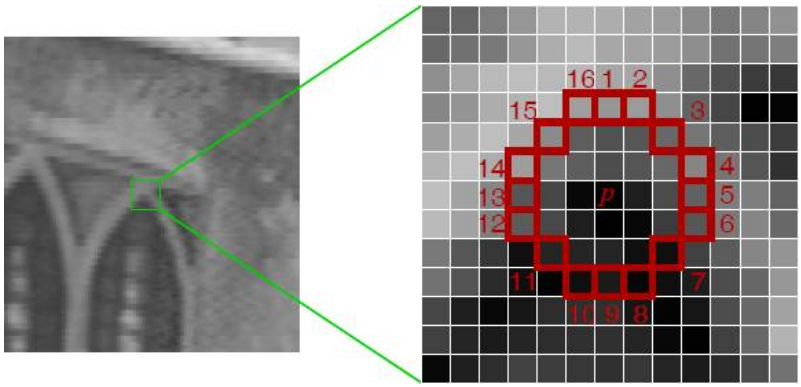


Figure 2.13. 16 pixels Bresenham circumference. The highlighted squares are the pixels used in the corner detection.

Although the value in pixels of the radio, *r*, can in principle take any value, FAST uses only a value of 3 (corresponding to a circle of 16 pixels circumference as that of Figure 2.13), and tests show that the best results are achieved with a number of highlighted contiguous pixels, *n*, being 9. This value of *n* is the lowest one at which edges are not detected, since we are not interested in detect edges, only corners. The order in which pixels are tested is determined by a ID3 algorithm from a training set of images. ID3 (Iterative Dichotomiser 3) is an algorithm used to generate decision trees, decision support tools that use a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

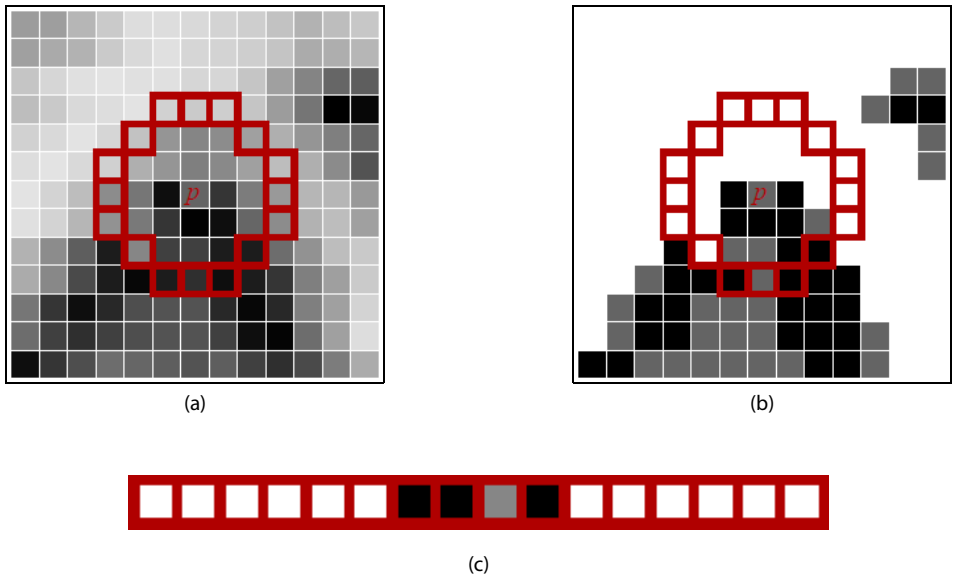


Figure 2.14. Over a candidate pixel *p* (a), the pixels of the Bresenham circumference are clasified in one of the three sorts: darker (in black), similar (in gray) and brighter (in white) (b). In (c) the pixels of the circumference are showed as a row.

For each location on the circle $x \in \{1..16\}$, the state, S, of the pixel at that position relative to *p* (denoted by $p \rightarrow x$) can be:

$$S_{p\rightarrow x} = \begin{cases} \text{d (darker)}, & I_{p\rightarrow x} \leq I_p - t \\ \text{s (similar)}, & I_p - t < I_{p\rightarrow x} < I_p + t \\ \text{b (brighter)}, & I_p + t \leq I_{p\rightarrow x} \end{cases}$$

Choosing an *x* and computing $S_{p\rightarrow x}$ for all $p \in P$ (the set of all pixels in a image) arranges *P* into three subsets, P_d, P_s, P_b, where each *p* is assigned to $P_{s_{p\rightarrow x}}$. The sort of the pixels are visually expressed in the Figure 2.14.

Let K_p be a boolean variable which is true if *p* is a corner and false otherwise. After to have classified the kind of the 16 pixels, the ID3 algorithm begins by selecting the *x* which yields the most information about whether the candidate pixel is a corner, measured by the entropy of K_p.

The entropy of K for the set P is:

$$H(P)=(c+\bar{c})\log_2(c+\bar{c})-c\log_2 c-\bar{c}\log_2\bar{c} \tag{2.8}$$

where $c=|\{p|K_p \, is \, true\}|$ (number of corners)

and $c=|\{p|K_p \, is \, false\}|$ (number of non corners)

The choice of x then yields the information gain:

$$H(P)-H(P_d)-H(P_s)-H(P_b) \tag{2.9}$$

Having selected the x which yields the most information, the process is applied recursively on all three subsets i.e. x_b is selected to partition P_b in to $P_{b,d}$, $P_{b,s}$, $P_{b,b}$, x_s is selected to partition P_s in to $P_{s,d}$, $P_{s,s}$, $P_{s,b}$ and so on, where each x is chosen to yield maximum information about the set it is applied to. The process terminates when the entropy of a subset is zero. This means that all p in this subset have the same value of K_p, i.e. they are either all corners or all non-corners. This is guaranteed to occur since K is an exact function of the learning data.

This creates a decision tree which can correctly classify all corners seen in the training set and therefore (to a close approximation) correctly embodies the rules of the chosen FAST corner detector. This decision tree is then converted into C-code, creating a long string of nested if-then-else statements which is compiled and used as a corner detector.

Non-maximal suppression

Since the segment test does not compute a corner response function, non maximal suppression can not be applied directly to the resulting features. Consequently, a score function, V, must be computed for each detected corner, and non-maximal suppression applied to this to remove corners which have an adjacent corner with higher V. Although there are several intuitive defnitions for V, due to speed of computation we assum that V is the sum of the absolute difference between the pixels in the contiguous arc and the centre pixel:

$$V=max\left(\sum_{x\in S_{bright}}|I_{p\to x}-I_p|-t \;\;,\;\; \sum_{x\in S_{dark}}|I_p-I_{p\to x}|-t\right) \tag{2.10}$$

where

$$\begin{aligned} S_{bright}&=\{x|I_{p\to x}\geqslant I_p+t\} \\ S_{dark}&=\{x|I_{p\to x}\geqslant I_p-t\} \end{aligned} \tag{2.11}$$

The non-maximal suppression stage is optional and in the Image Feature Detector

implementation of the FAST detector it can be deactivate, although we will view many not interesting points around a one corner.

SIFT

Scale-invariant feature transform (SIFT) is an algorithm in computer vision to detect and describe local features in images. The algorithm was published by David Lowe in 1999 in the work *Object recognition from local scale-invariant features* [12] and from 2000 it is patented in the US by the University of British Columbia [45].

Unlike the previous seen detectors, SIFT works in a different way and has more elavorated steps. The features detected by SIFT are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. The features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. Scale-space extrema detection: The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.

2. Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.

3. Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

4. Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

Detection of scale-space extrema

As described in the introduction, we will detect keypoints using a cascade filtering approach that uses efficient algorithms to identify candidate locations. The first stage of keypoint detection is to identify locations and scales that can be repeatably assigned under differing views of the same object. Detecting locations that are invariant to scale

change of the image can be accomplished by searching for stable features across all possible scales, using a continuous function of scale known as scale space (Witkin, 1983, [24]).

The scale space of an image is defined as a function, $L(x, y, \sigma)$, that is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$, with an input image, $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

where $*$ is the convolution operation in x and y, and

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

It has been shown by Koenderink (1984, [25]) and Lindeberg (1994, [26]) that under a variety of reasonable assumptions the only possible scale-space kernel is the Gaussian function.

To efficiently detect stable keypoint locations in scale space, we will use scale-space extrema in the difference-of-Gaussian function convolved with the image, $D(x, y, \sigma)$, which can be computed from the difference of two nearby scales separated by a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{2.12}$$

There are a number of reasons for choosing this function. First, it is a particularly efficient function to compute, as the smoothed images, L, need to be computed in any case for scale space feature description, and D can therefore be computed by simple image subtraction.

In addition, the difference-of-Gaussian function provides a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma\nabla^2 G$, as studied by Lindeberg. Lindeberg showed that the normalization of the Laplacian with the factor σ^2 is required for true scale invariance. In detailed experimental comparisons, Mikolajczyk (2002, [27]) found that the maxima and minima of $\sigma\nabla^2 G$ produce the most stable image features compared to a range of other possible image functions, such as the gradient, Hessian, or Harris corner function.

The relationship between D and $\sigma\nabla^2 G$ can be understood from the heat diffusion equation (parametrized in terms of σ rather than the more usual $t = \sigma^2$):

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G$$

From this, we see that $\nabla^2 G$ can be computed from the finite difference approximation to $\partial G/\partial\sigma$, using the difference of nearby scales at $k\sigma$ and σ:

$$\sigma\nabla^2 = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

and therefore,

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$$

This shows that when the difference-of-Gaussian function has scales differing by a constant factor it already incorporates the σ^2 scale normalization required for the scale-invariant Laplacian. The factor $(k-1)$ in the equation is a constant over all scales and therefore does not influence extrema location. The approximation error will go to zero as k goes to 1, but in practice we have found that the approximation has almost no impact on the stability of extrema detection or localization for even significant differences in scale, such as $k = \sqrt{2}$.

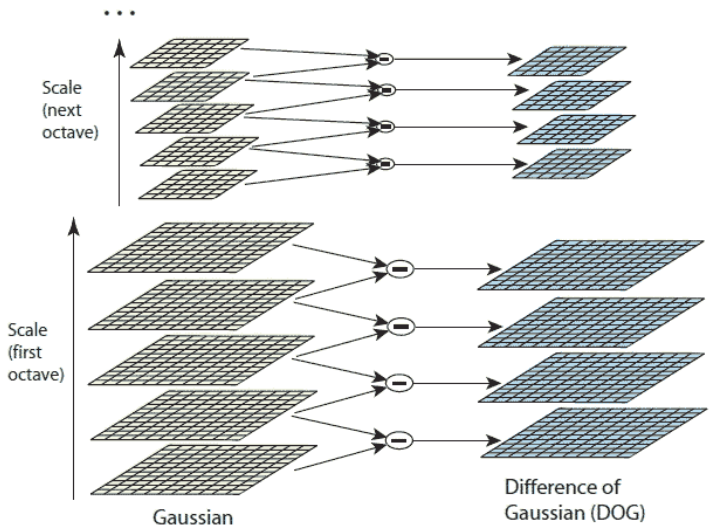


Figure 2.15. For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

An efficient approach to construction of $D(x, y, \sigma)$ is shown in Figure 2.15. The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale space (i.e., doubling of σ) into an integer number, s, of intervals, so $k = 2^{1/s}$. We must produce $s+3$ images in the stack of blurred images for each octave, so that final extrema detection covers a complete octave. Adjacent image scales are subtracted to produce the difference-of-Gaussian images shown on the right.

Once a complete octave has been processed, we resample the Gaussian image that has twice the initial value of σ (it will be 2 images from the top of the stack) by taking every second pixel in each row and column. The accuracy of sampling relative to σ is no different than for the start of the previous octave, while computation is greatly reduced.

In order to detect the local maxima and minima of $D(x, y, \sigma)$, each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below (see Figure 2.16). It is selected only if it is larger than all of these neighbors or smaller than all of them. The cost of this check is reasonably low due to the fact that most sample points will be eliminated following the first few checks.

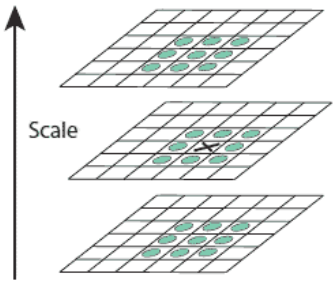


Figure 2.16. Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

An important issue is to determine the frequency of sampling in the image and scale domains that is needed to reliably detect the extrema. Unfortunately, it turns out that there is no minimum spacing of samples that will detect all extrema, as the extrema can be arbitrarily close together. We can determinate the best settings experimentally, by studding a range of sample frequencies *y* checking which of them give us the best results within a realistic simulation.

Accurate keypoint localization

Once a keypoint candidate has been found by comparing a pixel to its neighbors, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

We will use the Taylor expansion (up to the quadratic terms) of the scale-space function, $D(x, y, \sigma)$, shifted so that the origin is at the sample point:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \qquad (2.13)$$

where D and its derivatives are evaluated at the sample point and $x = (x, y, \sigma)^T$ is the

offset from this point. The location of the extremum, $\hat{x}$, is determined by taking the derivative of this function with respect to x and setting it to zero, giving

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{2.14}$$

The Hessian and derivative of D are approximated by using differences of neighboring sample points. The resulting 3x3 linear system can be solved with minimal cost. If the offset $\hat{x}$ is larger than 0.5 in any dimension, then it means that the extremum lies closer to a different sample point. In this case, the sample point is changed and the interpolation performed instead about that point. The final offset $\hat{x}$ is added to the location of its sample point to get the interpolated estimate for the location of the extremum.

The function value at the extremum, $D(\hat{x})$, is useful for rejecting unstable extrema with low contrast. This can be obtained by substituting Ecuation 2.14 into 2.13, giving

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

All extrema with a value of $|D(\hat{x})|$ less than 0.03 are discarded (as before, we assume image pixel values in the range [0,1]).

For stability, it is not sufficient to reject keypoints with low contrast. The difference-of-Gaussian function will have a strong response along edges, even if the location along the edge is poorly determined and therefore unstable to small amounts of noise.

A poorly defined peak in the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction. The principal curvatures can be computed from a 2×2 Hessian matrix, **H**, computed at the location and scale of the keypoint:

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2.15}$$

The derivatives are estimated by taking differences of neighboring sample points.

The eigenvalues of **H** are proportional to the principal curvatures of D. Borrowing from the approach used in the Harris detector, we can avoid explicitly computing the eigenvalues, as we are only concerned with their ratio. Let α be the eigenvalue with the largest magnitude and be the smaller one. Then, we can compute the sum of the eigenvalues from the trace of **H** and their product from the determinant:

$$Tr(\boldsymbol{H}) = D_{xx} + D_{yy} = \alpha + \beta$$
$$Det(\boldsymbol{H}) = D_{xx} D_{yy} - (D_{xy})^2 = \alpha\beta$$

In the unlikely event that the determinant is negative, the curvatures have different signs so the point is discarded as not being an extremum. Let r be the ratio between the largest magnitude eigenvalue and the smaller one, so that $\alpha = r\beta$. Then,

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

which depends only on the ratio of the eigenvalues rather than their individual values. The quantity $(r+1)^2/r$ is at a minimum when the two eigenvalues are equal and it increases with r. Therefore, to check that the ratio of principal curvatures is below some threshold, r, we only need to check

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

This is very efficient to compute, with less than 20 floating point operations required to test each keypoint. The experiments in this paper use a value of $r = 10$, which eliminates keypoints that have a ratio between the principal curvatures greater than 10. The transition from Figure 2.17(c) to (d) shows the effects of this operation.

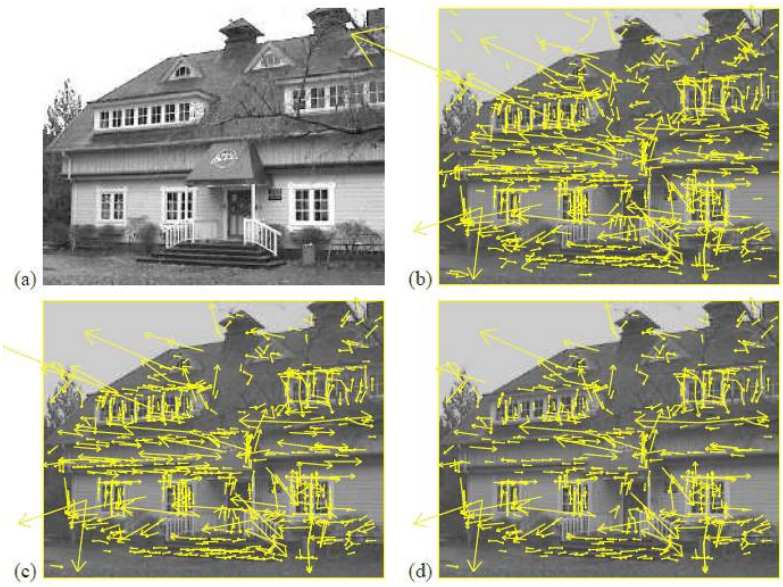


Figure 2.17. This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures.

Orientation assignment

By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation. This approach contrasts with the orientation invariant descriptors of Schmid and Mohr (1997, [28]), in which each image property is based on a rotationally invariant measure. The disadvantage of that approach is that it limits the descriptors that can be used and discards image information by not requiring all measures to be based on a consistent rotation.

The scale of the keypoint is used to select the Gaussian smoothed image, *L*, with the closest scale, so that all computations are performed in a scale-invariant manner. For each image sample, *L(x, y)*, at this scale, the gradient magnitude, *m(x, y)*, and orientation, $\theta(x, y)$, is precomputed using pixel differences:

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$$

$$\theta(x,y)=\arctan\left(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}\right)$$

The magnitude and direction calculations for the gradient are done for every pixel in a neighboring region around the keypoint in the Gaussian-blurred image L. An orientation histogram with 36 bins is formed, with each bin covering 10 degrees. Each sample in the neighboring window added to a histogram bin is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. The peaks in this histogram correspond to dominant orientations. Once the histogram is filled, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peaks are assigned to the keypoint. In the case of multiple orientations being assigned, an additional keypoint is created having the same location and scale as the original keypoint for each additional orientation.

The local image descriptor

The previous operations have assigned an image location, scale, and orientation to each keypoint. These parameters impose a repeatable local 2D coordinate system in which to describe the local image region, and therefore provide invariance to these parameters. The next step is to compute a descriptor for the local image region that is highly distinctive yet is as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.

Figure 2.18 illustrates the computation of the keypoint descriptor. First a set of orientation histograms are created on 4x4 pixel neighborhoods with 8 bins each. These histograms are computed from magnitude and orientation values of samples in a 16×16 region around the keypoint such that each histogram contains samples from a 4×4 subregion of the original neighborhood region. The magnitudes are further

weighted by a Gaussian function with σ equal to one half the width of the descriptor window. The descriptor then becomes a vector of all the values of these histograms. Since there are $4 \times 4 = 16$ histograms each with 8 bins the vector has 128 elements. This vector is then normalized to unit length in order to enhance invariance to affine changes in illumination. To reduce the effects of non-linear illumination a threshold of 0.2 is applied and the vector is again normalized.

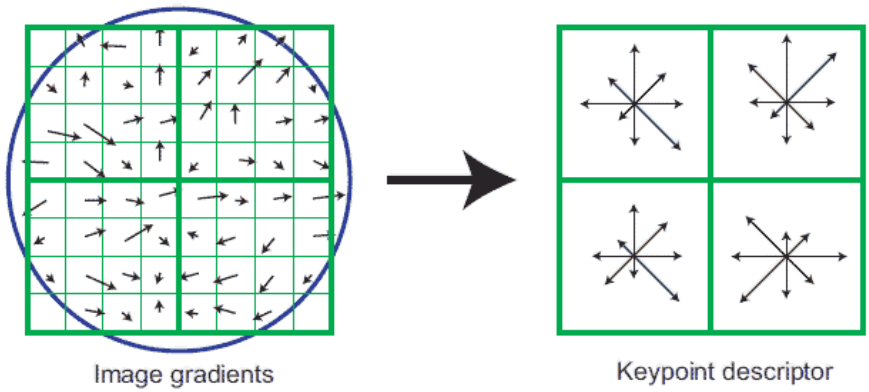


Figure 2.18. A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2×2 descriptor array computed from an 8×8 set of samples, whereas the experiments in this paper use 4×4 descriptors computed from a 16×16 sample array.

Although the dimension of the descriptor, i.e. 128, seems high descriptors with lower dimension than this don't perform as well across the range of matching tasks and the computational cost remains low due to the approximate BBF method used for finding the nearest-neighbor. Longer descriptors continue to do better but not by much and there is an additional danger of increased sensitivity to distortion and occlusion. It is also shown that feature matching accuracy is above 50% for viewpoint changes of up to 50 degrees. Therefore SIFT descriptors are invariant to minor affine changes. To test the distinctiveness of the SIFT descriptors, matching accuracy is also measured against varying number of keypoints in the testing database, and it is shown that matching accuracy decreases only very slightly for very large database sizes, thus indicating that SIFT features are highly distinctive.

SURF

SURF (Speeded Up Robust Features) is a robust image detector and descriptor, first presented by Herbert Bay et al. in 2003 in the work *Speeded-Up Robust Features (SURF)* [13]. It is partly inspired by the SIFT descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT. SURF is based on sums of approximated 2D

Haar wavelet responses and makes an efficient use of integral images. As basic image features it uses a Haar wavelet approximation of the determinant of Hessian blob detector.

The approach for interest point detection uses a very basic Hessian-matrix approximation. This lends itself to the use of integral images, which reduces the computation time drastically. Integral images fit in the more general framework of boxlets.

Integral Images

In order to make the article more self-contained, we briefly discuss the concept of integral images. They allow for fast computation of box type convolution filters. The entry of an integral image $I_\Sigma(x)$ at a location $x=(x, y)^\top$ represents the sum of all pixels in the input image I within a rectangular region formed by the origin and x.

$$I_\Sigma(x)=\sum_{i=o}^{i\leqslant x}\sum_{j=o}^{j\leqslant x} I(i,j) \tag{2.16}$$

Once the integral image has been computed, it takes three additions to calculate the sum of the intensities over any upright, rectangular area (see Figure 2.19). Hence, the calculation time is independent of its size. This is important in our approach, as we use big filter sizes.

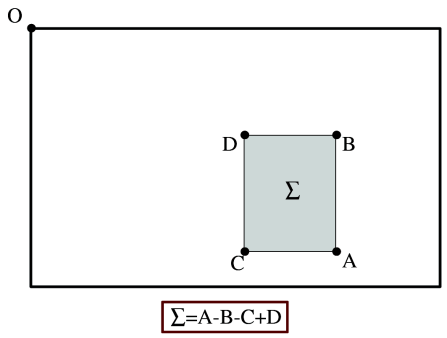


Figure 2.19. Using integral images, it takes only three additions and four memory accesses to calculate the sum of intensities inside a rectangular region of any size.

The SURF detector is based on the Hessian matrix because of its good performance in accuracy. More precisely, we detect blob-like structures at locations where the determinant is maximum. In contrast to the Hessian-Laplace detector by Mikolajczyk and Schmid [33], we rely on the determinant of the Hessian also for the scale selection, as done by Lindeberg [26].

Given a point $x=(x, y)$ in an image I, the Hessian matrix $H(x,\sigma)$ in x at scale σ is defined as follows

$$H(x,\sigma)=\begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{2.17}$$

where $L_{xx}(x, y)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with the image I in point x, and similarly for $L_{xy}(x, \sigma)$ and $L_{yy}(x, \sigma)$.

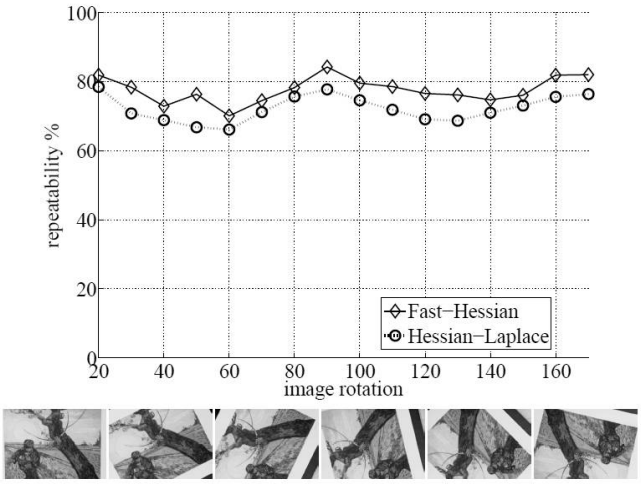


Figure 2.20. Using integral images, it takes only three additions and four memory accesses to calculate the sum of intensities inside a rectangular region of any size.

Gaussians are optimal for scale-space analysis [25] [26], but in practice they have to be discretised and cropped (Figure 2.21(a) and (b)). This leads to a loss in repeatability under image rotations around odd multiples of $\pi/4$. This weakness holds for Hessian-based detectors in general. Figure 2.20 shows the repeatability rate of two detectors based on the Hessian matrix for pure image rotation. The repeatability attains a maximum around multiples of $\pi/2$.

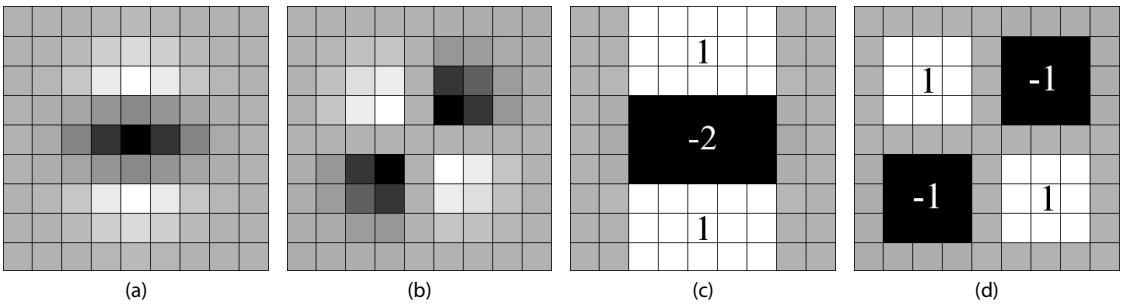


Figure 2.21: The (discretised and cropped) Gaussian second order partial derivative in y- (L_{yy}) (a) and xy-direction (L_{xy}) (b), the SURF approximation for the second order Gaussian partial derivative in y-(D_{yy}) (c) and xy-direction (D_{xy}) (d). The gray regions are equal to zero.

This is due to the square shape of the filter. Nevertheless, the detectors still perform well, and the slight decrease in performance does not outweigh the advantage of fast

convolutions brought by the discretisation and cropping. As real filters are non-ideal in any case, and given Lowe's success with his LoG approximations, we push the approximation for the Hessian matrix even further with box filters (Figure 2.21(c) and (d)). These approximate second order Gaussian derivatives and can be evaluated at a very low computational cost using integral images. The calculation time therefore is independent of the filter size. As shown in the results section and Figure 2.20, the performance is comparable or better than with the discretised and cropped Gaussians.

The 9×9 box filters in Figure 2.21 are approximations of a Gaussian with $\sigma=1.2$ and represent the lowest scale (i.e. highest spatial resolution) for computing the blob response maps. We will denote them by D_{xx}, D_{yy}, and D_{xy}. The weights applied to the rectangular regions are kept simple for computational efficiency. This yields

$$det\left(H_{approxi}\right)=D_{xx}D_{yy}-\left(wD_{xy}\right)^2 \tag{2.18}$$

The relative weight w of the filter responses is used to balance the expression for the Hessian's determinant. This is needed for the energy conservation between the Gaussian kernels and the approximated Gaussian kernels,

$$w=\frac{|L_{xy}(1.2)|_F|D_{yy}(9)|_F}{|L_{yy}(1.2)|_F|D_{xy}(9)|_F}=0.912...\simeq0.9 \tag{2.19}$$

where $|x|_F$ is the Frobenius norm. Notice that for theoretical correctness, the weighting changes depending on the scale. In practice, we keep this factor constant, as this did not have a significant impact on the results in our experiments.

The scale space is divided into octaves. An octave represents a series of filter response maps obtained by convolving the same input image with a filter of increasing size. In total, an octave encompasses a scaling factor of 2 (which implies that one needs to more than double the filter size.

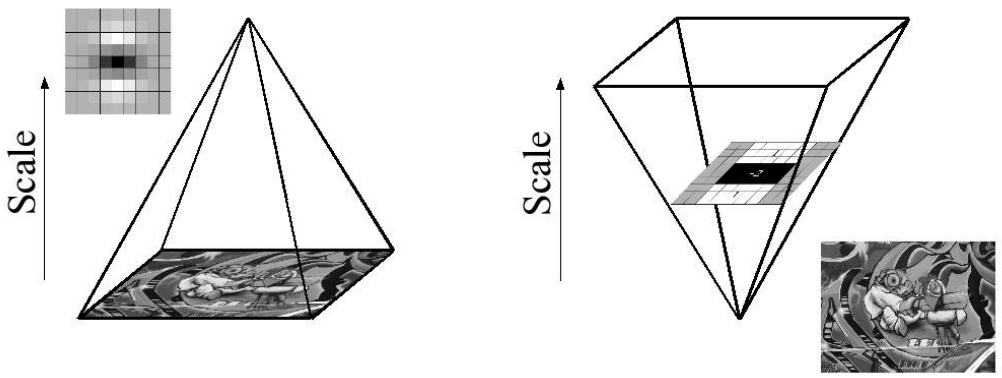


Figure 2.22. Instead of iteratively reducing the image size (left), the use of integral images allows the up-scaling of the filter at constant cost (right).

One of the key differences of the SURF detector compared to SIFT is that the SIFT detector, as it calculates respective scale-spaces, it resamples the original image to a minor size, unlike the SURF detector, that resamples the filter to an higher size, Figure 2.22. In this way, the computational cost of resampling the filter instead of the image is littler and at once more robust in presence of noise or affine transformations.

In order to localize interest points in the image and over scales, a non-maximum suppression in a $3\times3\times3$ neighborhood is applied. Scale space interpolation is especially important in our case, as the difference in scale between the first layers of scale every octave is relatively large. We can see in Figure 2.23 that most of the features can be detected in the first 3 octaves. More octaves may be analyzed, but the number of detected interest points per octave decays very quickly.

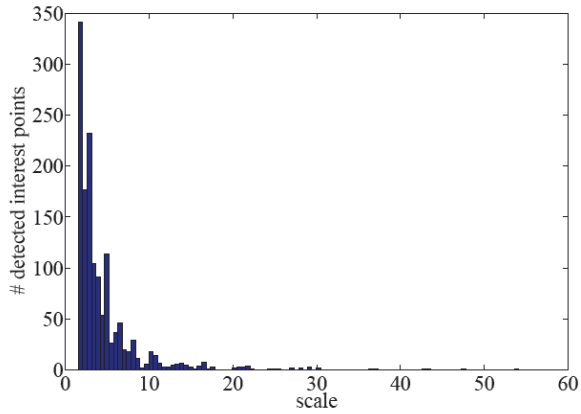


Figure 2.23. Histogram of the detected scales. The number of detected interest points per octave decays quickly.

Orientation Assignment

SURF describes the distribution of the intensity content within the interest point neighborhood, similar to the gradient information extracted by SIFT and its variants. We build on the distribution of first order Haar wavelet responses in x and y direction rather than the gradient, exploit integral images for speed, and use only 64 dimensions. This reduces the time for feature computation and matching, and has proven to simultaneously increase the robustness. Furthermore, we present a new indexing step based on the sign of the Laplacian, which increases not only the robustness of the descriptor, but also the matching speed (by a factor of two in the best case).

In order to be invariant to image rotation, we identify a reproducible orientation for the interest points. For that purpose, we first calculate the Haar wavelet responses in x and y direction within a circular neighborhood of radius $6s$ around the interest point, with s the scale at which the interest point was detected. The sampling step is scale dependent and chosen to be s. In keeping with the rest, also the size of the wavelets are scale dependent and set to a side length of $4s$. Therefore, we can again use integral

images for fast filtering. The used filters are shown in Figure 2.24. Only six operations are needed to compute the response in x or y direction at any scale.

Figure 2.24. Haar wavelet filters to compute the responses in x (a) and y direction (b). The dark parts have the weight -1 and the light parts +1.

Once the wavelet responses are calculated and weighted with a Gaussian ($\sigma = 2s$, with s the scale at which the interest point was detected) centered at the interest point, the responses are represented as points in a space with the horizontal response strength along the abscissa and the vertical response strength along the ordinate. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of size $\pi/3$, Figure 2.25. The horizontal and vertical responses within the window are summed. The two summed responses then yield a local orientation vector. The longest such vector over all windows defines the orientation of the interest point.

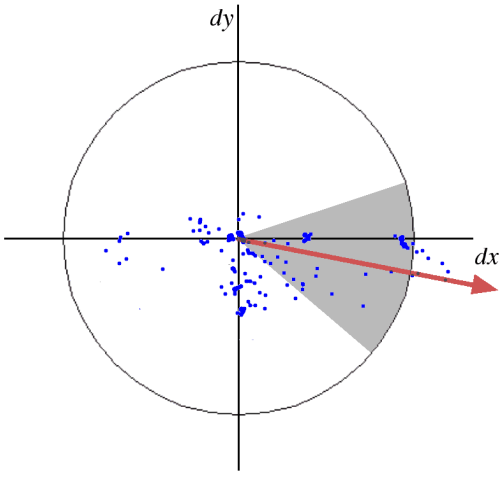


Figure 2.25. Orientation assignment: A sliding orientation window of size $\pi/3$ detects the dominant orientation of the Gaussian weighted Haar wavelet responses at every sample point within a circular neighborhood around the interest point.

Descriptor based on Sum of Haar Wavelet Responses

For the extraction of the descriptor, the first step consists of constructing a square region centered around the interest point and oriented along the orientation selected

in the previous section. The size of this window is $20s$. Examples of such square regions are illustrated in Figure 2.26.

Figure 2.26. Detail of a scene showing the size of the oriented descriptor window at different scales.

The region is split up regularly into smaller 4×4 square sub-regions. This preserves important spatial information. For each sub-region, we compute Haar wavelet responses at 5×5 regularly spaced sample points. For reasons of simplicity, we call d_x the Haar wavelet response in horizontal direction and d_y the Haar wavelet response in vertical direction (filter size $2s$), see Figure 2.24 again. "Horizontal" and "vertical" here is defined in relation to the selected interest point orientation (see Figure 2.27). For efficiency reasons, the Haar wavelets are calculated in the unrotated image and the responses are then interpolated, instead of actually rotating the image. To increase the robustness towards geometric deformations and localization errors, the responses d_x and d_y are first weighted with a Gaussian ($\sigma = 3.3s$) centered at the interest point.

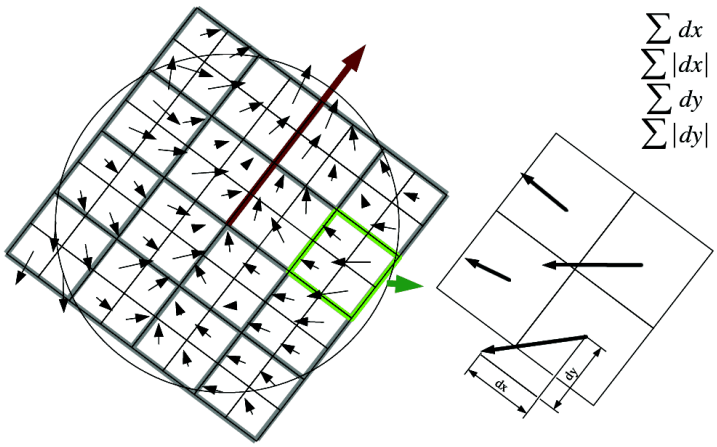


Figure 2.27. To build the descriptor, an oriented quadratic grid with 4×4 square sub-regions is laid over the interest point (left). For each square, the wavelet responses are computed. The 2×2 sub-divisions of each square correspond to the actual fields of the descriptor. These are the sums dx, $|dx|$, dy, and $|dy|$, computed relatively to the orientation of the grid (right).

Then, the wavelet responses dx and dy are summed up over each sub-region and form a first set of entries in the feature vector. In order to bring in information about the polarity of the intensity changes, we also extract the sum of the absolute values of the responses, $|dx|$ and $|dy|$. Hence, each sub-region has a four-dimensional descriptor vector v for its underlying intensity structure $v=\left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right)$, concatenating this for all 4×4 sub-regions, this results in a descriptor vector of length 64. The wavelet responses are invariant to a bias in illumination (offset). Invariance to contrast (a scale factor) is achieved by turning the descriptor into a unit vector.

Figure 2.28 shows the properties of the descriptor for three distinctively different image intensity patterns within a sub-region. One can imagine combinations of such local intensity patterns, resulting in a distinctive descriptor.

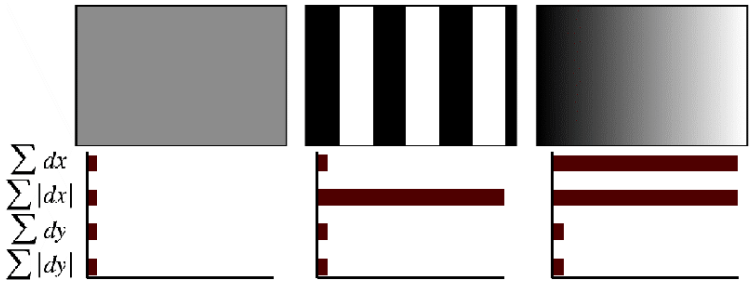


Figure 2.28. The descriptor entries of a sub-region represent the nature of the underlying intensity pattern. Left: In case of a homogeneous region, all values are relatively low. Middle: In presence of frequencies in x direction, the value of Σ|*dx*| is high, but all others remain low. If the intensity is gradually increasing in *x* direction, both values Σ|*dx*| and Σ|*dx*| are high.

Chapter 3

Comparing Image Descriptors: Proofs and Results

After viewing a summary of the detectors working, we shall view the performance of each one. Since the four detectors do not working in the same way, the comparison can not be done with the same conditions for the four detectors.

Harris and FAST operators are aimed to mainly detect corners. They simply shows in an image where are corners. Because of the relative little number of operations, these detectors are computationally little costly. FAST is the faster detector of all, and can even be implemented for real time image sequences in low-end computers.

On the other hand, SIFT and SURF detectors are a little bit different from Harris and FAST operators. SIFT and SURF work with the concept of scale-invariant; they are able to detect and select a good feature point set (not only corners), save the feature parameters of the set in a size-independent format and to find these detected features in another image, although this image is (partially) rotated, scaled, warped, has a different illumination, more noise or there are more objects than the original image. SIFT and SURF are object matching focused and have the necessary tools to run this task, that is, they elaborated descriptors (the *identity card* of each feature that has been founded in an image) that are stored to be compared by the other basic tool of space-invariant detectors, a matching engine that decides whether some feature is the same than another. The logarithm of these detectors looks for features at the original resolution image but also in down-sampled versions, in the so called space-scales. Because the algorithm must rescale the image and have to calculate each feature descriptor, scale-invariant detectors as SIFT or SURF are generally much more computationally expensive than Harris or FAST.

Due to the above explained, the detectors are compared in pairs based on their nature, Harris vs. FAST and SIFT vs. SURF. In the comparative, time performance and number of features detected are tested. In addition, the detectors are put to test with rotated, scaled, point of view changed, blur and noise image sets. In addition, in the SIFT vs. SURF comparative, the quality of the descriptors is tested and the true positive rate is showed in an ROC graphic.

The Harris vs. FAST and SIFT vs. SURF comparatives are a summarize of the FAST [14] and SURF [13] scientific presentation articles, respectively. In the Harris vs. FAST

comparative, the Differential of Gaussian operator, the one used by SIFT and the SUSAN operator (1993, [23]), not explained in this project, also appear. In the second comparative, furthermore than the difference of Gaussians (SIFT) and Fast Harris operators (SURF) also appear the Hessian-Laplace and Harris-Laplace operators, from the Hessian-Affine (2002, [29]) and Harris-Affine (2005, [27]) space-scale detectors, both created by K. Mikolajczyk et al..

Harris vs. FAST

Timing Results

Timing tests were performed on a 2.6GHz Opteron and an 850MHz Pentium III processor. The timing data is taken over 1500 monochrome fields from a PAL video source (with a resolution of 768×288 pixels). The learned FAST detectors for $n = 9$ and 12 have been compared to the FAST autors implementation of the Harris and DoG (difference of Gaussians, the approach used by SIFT) and to the reference implementation of SUSAN [23].

As can be seen in Table 3.1, FAST in general offers considerably higher performance than the other tested feature detectors, and the learned FAST performs up to twice as fast as the handwritten version. Importantly, for $n = 9$ is the most reliable of the FAST detectors. On modern hardware, FAST consumes only a fraction of the time available during video processing, and on low power hardware, it is the only one of the detectors tested which is capable of video rate processing at all.

Detector	Opteron 2.6GHz		Pentium III 850MHz	
	Time (ms)	Time (%)	Time (ms)	Time (%)
FAST $n = 9$ (non-max suppression)	1.33	6.65	5.29	26.5
FAST $n = 9$ (raw)	1.08	5.40	4.34	21.7
FAST $n = 12$ (non-max suppression)	1.34	6.70	4.60	23.0
FAST $n = 12$ (raw)	1.17	5.85	4.31	21.5
Harris	24.0	120	166	830
DoG (SIFT)	60.1	301	345	1280
SUSAN	7.58	37.9	27.5	137.5

Table 3.1. Timing results for a selection of feature detectors run on fields (768x288) of a PAL video sequence in milliseconds, and as a percentage of the processing budget per frame. Note that since PAL and NTSC, DV and 30Hz VGA (common for webcams) have approximately the same pixel rate, the percentages are widely applicable. Approximately 500 features per field are detected.

Repeatability Test Conditions

Although there is a vast body of work on corner detection, there is much less on the subject of comparing detectors. Mohannah and Mokhtarian [31] evaluate performance

by warping test images in an affine manner by a known amount. They define the "consistency of corner numbers" as

$$CCN = 100 \times 1.1^{|-n_w - n_o|} \quad ,$$

where n_w is the number of features in the warped image and n_o is the number of features in the original image. They also define accuracy as

$$ACU = 100 \times \frac{\frac{n_a}{n_o} + \frac{n_a}{n_g}}{2} \quad ,$$

where n_g are the number of "ground truth" corners (marked by humans) and n_a is the number of matched corners compared to the ground truth. This unfortunately relies on subjectively made decisions.

Trajkovic and Hedley (1998, [32]) define stability to be the number of "strong" matches (matches detected over three frames in their tracking algorithm) divided by the total number of corners. This measurement is clearly dependent on both the tracking and matching methods used, but has the advantage that it can be tested on the date used by the system.

When measuring reliability, what is important is if the same real-world features are detected from multiple views [33]. This is the definition which will be used here. For an image pair, a feature is "detected" if it is extracted in one image and appears in the second. It is "repeated" if it is also detected nearby in the second. The repeatability is the ratio of repeated detected features. In [33], the test is performed on images of planar scenes so that the relationship between point positions is a homography (invertible transformation from the real projective plane to the projective plane that maps straight lines to straight lines). Fiducial markers (reference points and lines) are projected on to the planar scene to allow accurate computation of this.

By modeling the surface as planar and using flat textures, this technique tests the feature detectors' ability to deal with mostly affine warps (since image features are small) under realistic conditions. We use a 3D surface model to compute where detected features should appear in other views (illustrated in Figure 3.2). This allows the repeatability of the detectors to be analyzed on features caused by geometry such as corners of polyhedra (geometric solid in three dimensions with flat faces and straight edges), occlusions and T-junctions. We also allow bas-relief textures (projecting image with a shallow overall depth) to be modeled with a flat plane so that the repeatability can be tested under non-affine warping.

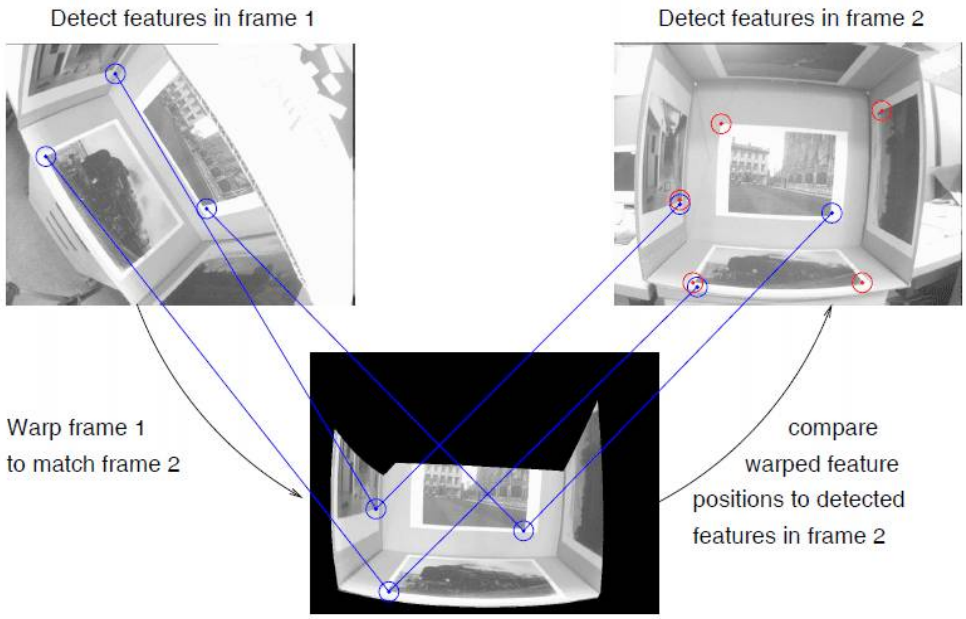


Figure 3.2. Repeatability is tested by checking if the same real-world features are detected in different views. A geometric model is used to compute where the features reproject to.

A margin of error must be allowed because:

— The alignment is not perfect.

— The model is not perfect.

— The camera model (especially regarding radial distortion) is not perfect.

— The detector may find a maximum on a slightly different part of the corner. This becomes more likely as the change in viewpoint and hence change in shape of the corner become large.

Instead of using fiducial markers, the 3D model is aligned to the scene by hand and this is then optimized using a blend of simulated annealing and gradient descent to minimize the SSD between all pairs of frames and reprojections.

To compute the SSD between frame *i* and reprojected frame *j*, the position of all points in frame *j* are found in frame *i*. The images are then bandpass filtered. High frequencies are removed to reduce noise, while low frequencies are removed to reduce the impact of lighting changes. To improve the speed of the system, the SSD is only computed using 1000 random points (as opposed to every point).

The datasets used are shown in Figure 3.3, Figure 3.4 and Figure 3.5. With these datasets, it has tried to capture a wide range of corner types (geometric and textural).

Figure 3.3. Box dataset: photographs taken of a test rig (consisting of photographs pasted to the inside of a cuboid) with strong changes of perspective, changes in scale and large amounts of radial distortion. This tests the corner detectors on planar textures.

Figure 3.4. Maze dataset: photographs taken of a prop used in an augmented reality application. This set consists of textural features undergoing projective warps as well as geometric features. There are also significant changes of scale.

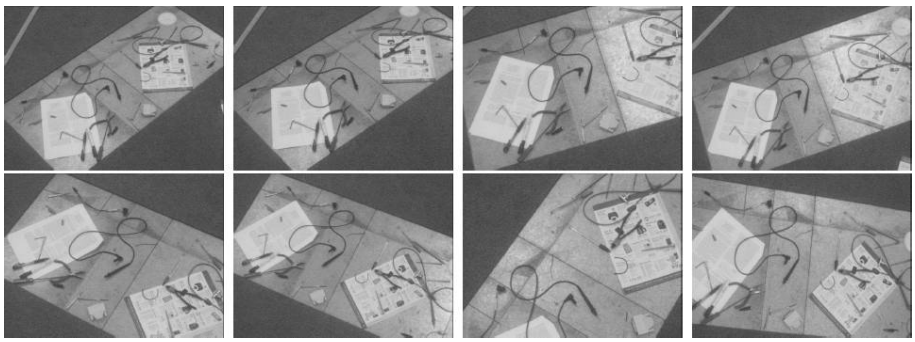


Figure 3.5. Bas-relief dataset: the model is a flat plane, but there are many objects with significant relief. This causes the appearance of features to change in a non affine way from different viewpoints.

The repeatability is computed as the number of corners per frame is varied. For comparison we also include a scattering of random points as a baseline measure, since in the limit if every pixel is detected as a corner, then the repeatability is 100%.

To test robustness to image noise, increasing amounts of Gaussian noise were added to the bas-relief dataset. It should be noted that the noise added is in addition to the significant amounts of camera noise already present.

Results

Shi and Tomasi, derive their result for better feature detection on the assumption that the deformation of the features is affine. In the box (Figure 3.3) and maze (Figure

3.4) datasets, this assumption holds and can be seen in Figure 3.6(a) and Figure 3.6(b) the detector outperforms the Harris detector. In the bas-relief dataset (Figure 3.5), this assumption does not hold, and interestingly, the Harris detector outperforms Shi and Tomasi detector in this case.

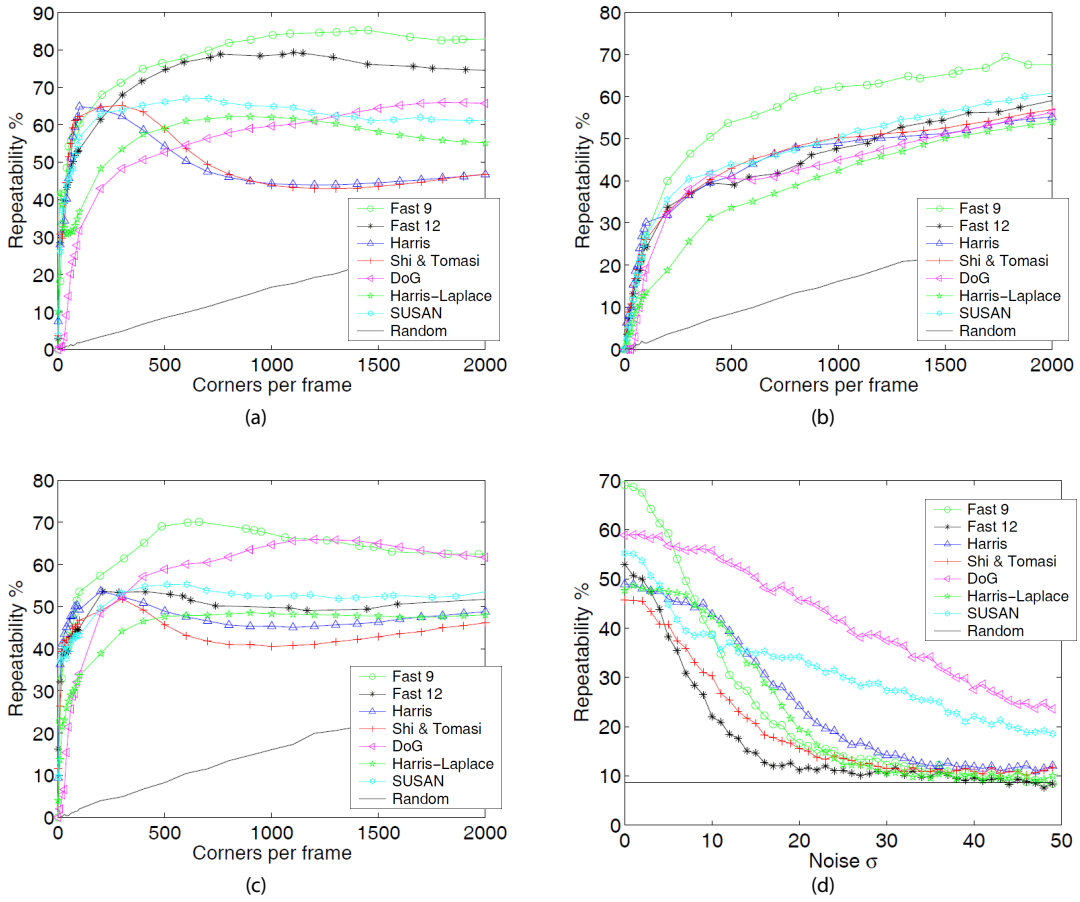


Figure 3.6. (a), (b), (c): Repeatability results for the three datasets as the number of features per frame is varied. (d): repeatability results for the bas-relief data set (500 features per frame) as the amount of Gaussian noise added to the images is varied. For FAST and SUSAN, the number of features can not be chosen arbitrarily; the closest approximation to 500 features per frame achievable is used.

Mikolajczyk and Schmid [34] evaluate the repeatability of the Harris-Laplace detector using the method in [35], where planar scenes are examined. The results show that Harris-Laplace points outperform both DoG points and Harris points in repeatability. For the box dataset, the results verify that this is correct for up to about 1000 points per frame (typical numbers, probably commonly used); the results are somewhat less convincing in the other datasets, where points undergo non-projective changes.

In the sample implementation of SIFT [44], approximately 1000 points are generated on the images from the test sets. We concur that this a good choice for the number of features since this appears to be roughly where the repeatability curve for DoG features starts to flatten off.

Smith and Brady [23] claim that the SUSAN corner detector performs well in the presence of noise since it does not compute image derivatives, and hence, does not amplify noise. We support this claim: although the noise results show that the performance drops quite rapidly with increasing noise to start with, it soon levels off and outperforms all but the DoG detector.

The big surprise of this experiment is that the FAST feature detectors, despite being designed only for speed, outperform the other feature detectors on these images (provided that more than about 200 corners are needed per frame). It can be seen in Figure 3.6(a), that the 9 point detector provides optimal performance, hence only this and the original 12 point detector are considered in the remaining graphs.

The DoG detector is remarkably robust to the presence of noise. Since convolution is linear, the computation of DoG is equivalent to convolution with a DoG kernel. Since this kernel is symmetric, this is equivalent to matched filtering for objects with that shape. The robustness is achieved because matched filtering is optimal in the presence of additive Gaussian noise.

FAST, however, is not very robust to the presence of noise. This is to be expected: Since high speed is achieved by analyzing the fewest pixels possible, the detector's ability to average out noise is reduced.

SIFT vs. SURF

SURF is, up to some point, similar in concept like SIFT, in that they both focus on the spatial distribution of gradient information. Nevertheless, SURF outperforms SIFT in practically all cases, as shown below. This can be due to the fact that SURF integrates the gradient information within a subpatch, whereas SIFT depends on the orientations of the individual gradients. This makes SURF less sensitive to noise, as illustrated in the example of Figure 3.7.

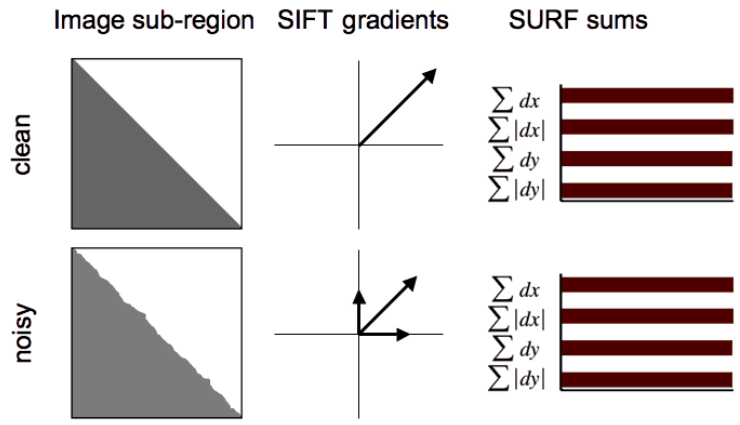
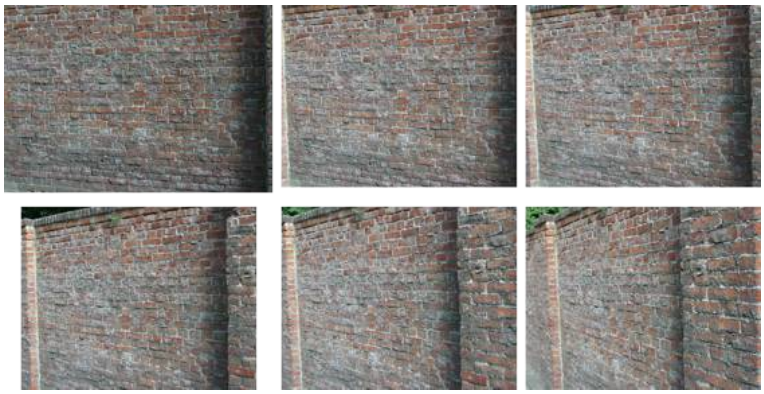


Figure 3.7. Due to the global integration of SURF's descriptor, it stays more robust to various image perturbations than the more locally operating SIFT descriptor.

The following presents both simulated as well as real world results. First, we evaluate the effect of some parameter settings and show the overall performance of the detector and descriptor based on a standard evaluation set.

We tested the detector using the image sequences and testing software provided by Mikolajczyk [43] that they can be seen in Figure 3.8. The evaluation criterion is the *repeatability score* (the percentage of points simultaneously present in two images). The test sequences comprise images of real textured and structured scenes. There are different types of geometric and photometric transformations, like changing viewpoints, zoom and rotation, image blur, lighting changes and JPEG compression.



(a)

(b)

(c)

(d)

Figure 3.8. Four image sequences. With the wall images (a) and the graffiti images (b) the detector repeatability is tested in presence of viewpoint change, with the boat images (c) in the presence of scale changes and with the bikes images (d) in presence of blur.

In all experiments the timings were measured on a standard PC Pentium IV, running at 3 GHz.

Experimental Evaluation and Parameter Settings

We tested two versions of the Fast-Hessian detector, depending on the initial Gaussian derivative filter size. FH-9 stands for the Fast Hessian detector with the initial filter size 9×9, and FH-15 is the 15×15 filter on the double input image size version. Apart from this, for all the experiments shown, the same thresholds and parameters were used.

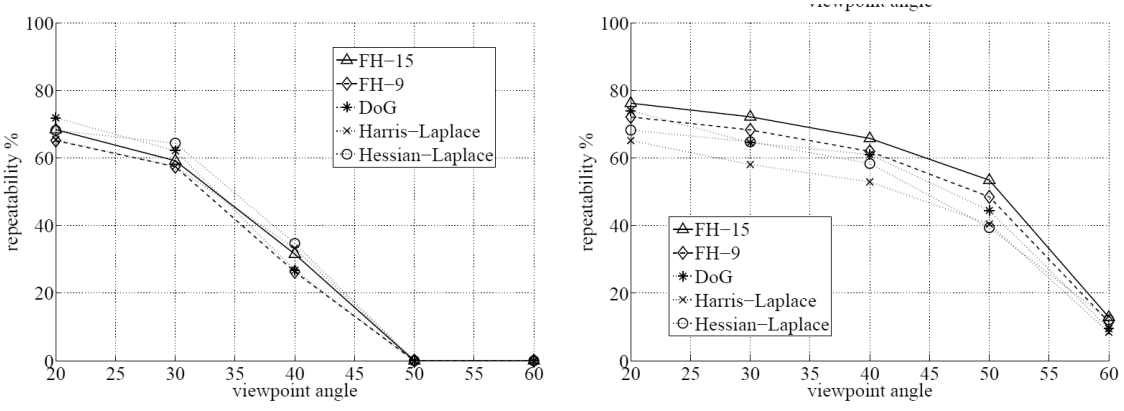
The detector is compared to the Difference of Gaussians (DoG) detector by Lowe (from the SIFT detector), and the Harris Laplace detectors proposed by Mikolajczyk [30]. The number of interest points found is on average very similar for all detectors (see Table 3.9 for an example). The thresholds were adapted according to the number of interest points found with the DoG detector.

Detector	Threshold	Points	Time (ms)
FH-15 (SURF)	60.000	1813	160
FH-9 (SURF)	50.000	1411	70
Hessian-Laplace (Hessian-Affine [x])	1000	1979	700
Harris-Laplace (Harris-Affine [x])	2500	1664	2100
DoG (SIFT)	default	1520	400

Table 3.9. Thresholds, number of detected points and calculation time for the detectors in our comparison. (First image of Gra#ti scene, 800×640).

The FH-9 detector is more than five times faster than DoG and ten times faster than Hessian-Laplace. The FH-15 detector is more than three times faster than DoG and more than four times faster than Hessian-Laplace (see also Table 3.9). At the same time, the repeatability scores for our detectors are comparable or even better than for the competitors.

The repeatability scores for the Graffiti sequence (Figure 3.10(a)) are comparable for all detectors. The repeatability score of the FH-15 detector for the Wall sequence (Figure 3.10(b)) outperforms the competitors. Note that the sequences Graffiti and Wall contain out-of-plane rotation, resulting in affine deformations, while the detectors in the comparison are only invariant to image rotation and scale. Hence, these deformations have to be accounted for by the overall robustness of the features. In the Boat sequence (Figure 3.10(c)), the FH-15 detector shows again a better performance than the others. The FH-9 and FH-15 detectors are outperforming the others in the Bikes sequence (Figure 3.10(d)).



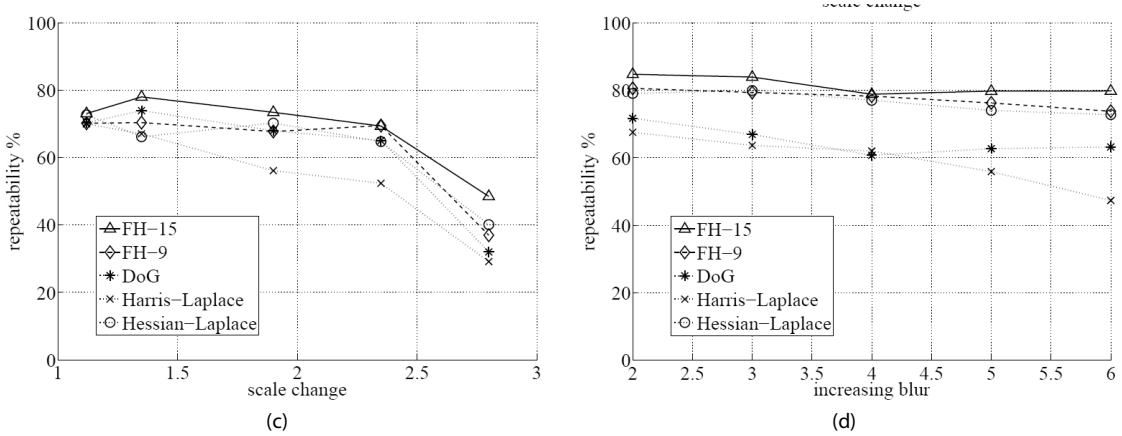


Figure 3.10. Repeatability score for the Graffiti (a) and Wall (b) (viewpoint change), Boat (c) (scale change) and Bikes sequence (d) (image blur).

Descriptors Evaluation

Bay et al. [36] already demonstrated the usefulness of SURF in a simple object detection task. To further illustrate the quality of the descriptor in such a scenario, we present some further experiments. Basis for this was a publicly available implementation of two bag-of-words classifiers [38]. Given an image, the task is to identify whether an object occurs in the image or not. For our comparison, we considered the naive Bayes classifier, which works directly on the bag-of-words representation, as suggested by Dance et al. [37]. This simple classifier was chosen as more complicated methods like pLSA might wash out the actual effect of the descriptor. Similar to [38], we executed our tests on 400 images each from the Caltech background and airplanes set. 50% of the images are used for training, the other 50% for testing. To minimize the influence of the partitioning, the same random permutation of training and test sets was chosen for all descriptors. While this is a rather simple test set for object recognition in general, it definitely serves the purpose of comparing the performance of the actual descriptors.

The framework already provides interest points, chosen randomly along Canny [39] edges to create a very dense sampling. These are then fed to the various descriptors. Additionally, we also consider the use of SURF keypoints, generated with a very low threshold, to ensure good coverage.

Figure 3.11 shows the obtained ROC curves for SURF-128, SIFT and GLOH. Note that for the calculation of SURF, the sign of the Laplacian was removed from the descriptor. For both types of interest points, SURF-128 outperforms its competitors on the majority of the curve significantly.

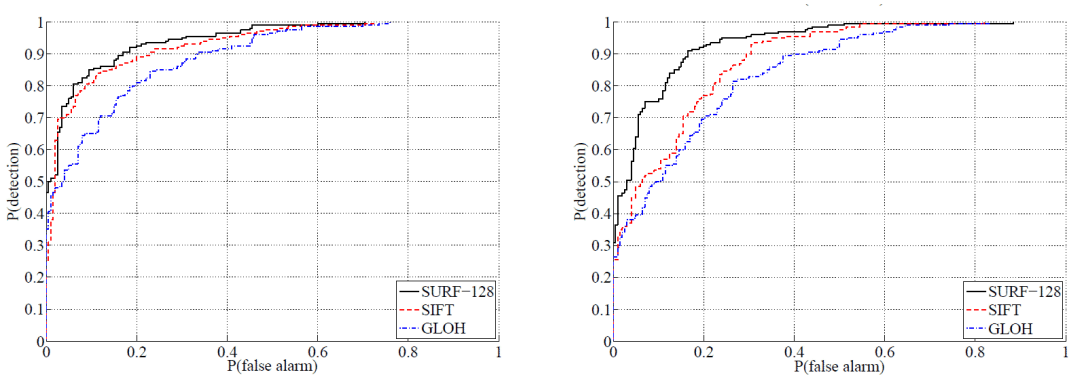


Figure 3.11. Comparison of different descriptor strategies for a naive Bayes classifier working on a bag-of-words representation. Top: descriptors evaluated on random edge pixels. Bottom: on SURF keypoints.

Chapter 4

Image Feature Detector: the Computer Program

Overview

Up to this point we have viewed what is a detector and how they work in detail. We have compared and tested four detectors and now we shall view the last part of the project: the practical implementation.

The practical implementation is where we make reality all what we have reviewed in previous chapters. This practical implementation is a Linux program with graphical user interface where Harris, FAST, SIFT and SURF detectors are applied on images and detected features are showed on screen. The program is called Image Feature Detector and uses the Qt application framework and the OpenCV library to carry out the implementation of the feature detectors.

Qt Application Framework

Qt is a cross-platform application framework used for developing application software [52] with graphical user interface (in whose case Qt is mainly used as a widget toolkit), and also used for developing non-GUI programs such as command-line tools and consoles for servers. It is developed by Nokia from 2008, after the Trolltech acquisition. Distributed under the terms of the GNU Lesser General Public License (among others), Qt is free and open source software.

Figure 4.1. Qt logo and slogan.

Qt uses standard C++ but makes extensive use of a special code generator (called the Meta Object Compiler, or MOC) together with several macros to enrich the language. Qt can also be used in several other programming languages via language bindings. It runs on all major platforms and has extensive internationalization support. Non-GUI features include SQL database access, XML parsing, thread management, network support, and a unified cross-platform API for file handling. All editions support a wide range of compilers, including the GCC C++ compiler and the Visual Studio suite.

The modular Qt C++ class library provides a set of application building blocks, delivering all of the functionality needed to build advanced, cross-platform applications, Figure 4.2.

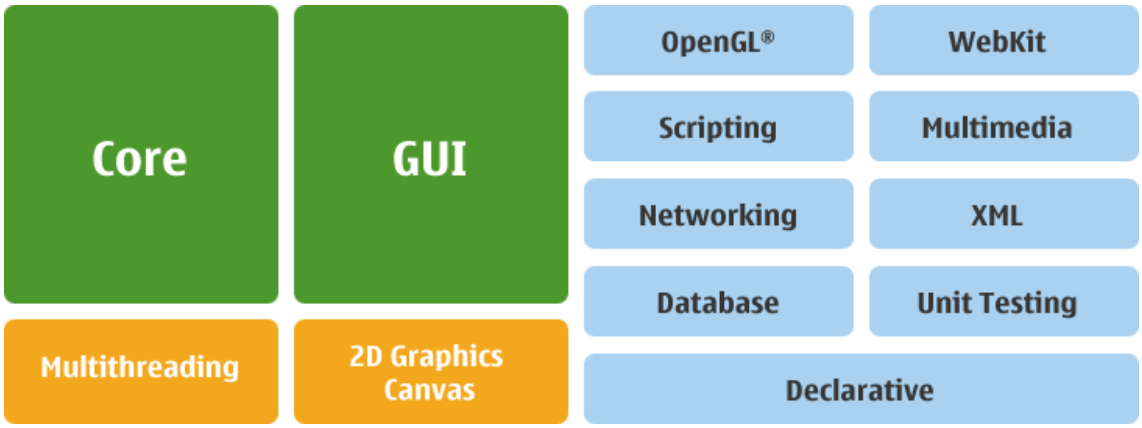
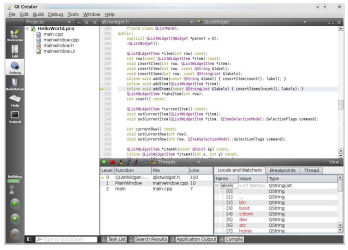


Figure 4.2. Qt module architecture. Qt allow a flexible use of C++ modules that we can choice and add to our project as we need some kind of functionality.

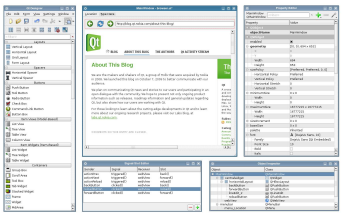
At all times, Qt was available under a commercial license that allows the development of proprietary applications without restrictions on licensing. In addition to that, Qt has been gradually made available under a number of increasingly free licenses. On January 14, 2009, Qt version 4.5 added another option, the LGPL, which should make Qt even more attractive for non-GPL open source projects and for closed applications.

In addition, Qt framework has a set of official tools which makes easier to program applications to developers. The main tools at a glance are:



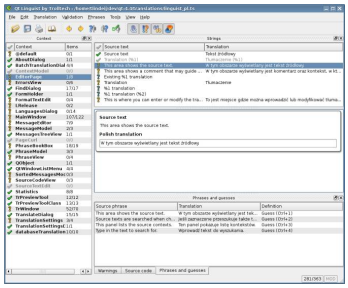
Qt Creator

Qt Creator is a cross-platform integrated development environment (IDE) tailored to the needs of Qt developers. Qt Creator runs on Windows, Linux/X11 and Mac OS X desktop operating systems, and allows developers to create applications for multiple desktop and mobile device platforms.



Qt Designer

Qt Designer is a cross-platform GUI layout and forms builder. It allows to design and build widgets and dialogs using on-screen forms using the same widgets that will be used in the application. Forms created with Qt Designer are fully-functional, and they can be previewed with all the look and feel that will be showed in run time.



Qt Linguist

Qt Linguist provides a set of tools that let the translation and internationalization of applications. Qt supports simultaneous support of multiple languages and writing systems with a single source tree and single application binary.

In Image Feature Detector, besides Qt Creator (Kdevelop has been used instead) Qt Designer and Qt Linguist have been used in the development of the program.

OpenCV Library

OpenCV is a computer vision programming library [47]. It is free for use under the open source BSD license. The library is written in C and C++, and hence, it is cross-platform. There is active development on interfaces for Python, Java, Matlab, and other languages [51].

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative, but from mid 2008 is supported by Willog Garage, a robotics research laboratory with headquarters in Menlo Park (California, USA) devoted to developing hardware and open source software for industrial, academic and personal robotics applications.

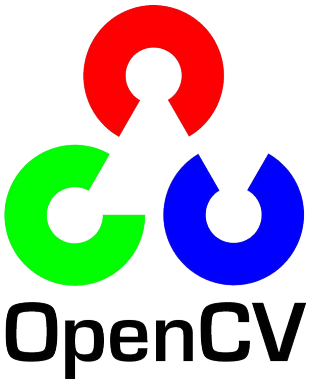


Figure 4.3. OpenCV logo.

OpenCV was designed for computational efficiency and with a strong focus on realtime applications. OpenCV can making use of Intel's Integrated Performance Primitives (IPP): if the library finds these Intel optimized routines on the system, it will use them to accelerate itself in run time. In addition, the library can take advantage of multicore processors.

One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-inhand, OpenCV also contains a full, general-purpose Machine Learning Library (MLL). This sublibrary is focused on statistical pattern recognition and clustering. The MLL is highly useful for the vision tasks that are at the core of OpenCV's mission, but it is general enough to be used for any machine learning problem.

Image Feature Detector mainly makes use of some OpenCV functions to use the Harris, FAST, SIFT and SURF detectors and applying them to images, but also uses other functions to calculate the used time detecting features.

Image Feature Detector

Image Feature Detector has been developed to put into practice the concepts explained in the project in a technical and visual way. Therefore, it has been a little challenger to learn C++ (I only knew a little bit C and Java), to know the Qt framework and to fight (yes, to fight) with the OpenCV libraries.

Learning C++ has not been hard because there are countless of tutorials and books on Internet and I knew Java, similar object-oriented programming language. Knowing the Qt framework has an easy learning curve and its hierarchy class modules, its comprehensive official documentation and tools make the use of these libraries a quick and intuitive task. But unlucky, the use of the OpenCV libraries has been a little harder issue. There are various reasons than have made of the linkage of OpenCV function calls with the Image Feature Detector classes a complicated matter with several headaches:

— First of all, the documentation organization is confused. There are the C and the C++ library, and sometimes you do not know in which library to search some function or structure. The learning curve is high, the classes, structures and functions descriptions are short. In addition, there are three different image formats, CvMat and IplImage C image formats and Mat C++ image format. These formats are different in implementation, but for use are the same, and many times functions do not specificate what image format should be used. For converting from CvMat and IplImage to Mat there are problems and it requires additional computational use.

— The version of OpenCV used in the project is downloaded directly from the program repository. This is so because the last released version as a project date, the 2.1, does not include the SIFT detector, it was only included from the incoming version not released yet. This repository version has underwent a complete module arrangement and, furthermore, the C OpenCV implementation, reportedly, appears to be deprecated and its use is not recommended (although neither forbidden). Because of this and the lack of documentation in this OpenCV trunk version, I have had to guess how the functions worked instead of to call to a function and fixing the parameters.

As main IDE, I have used KDevelop, not for its power, options or versatility but for its simplicity and overall, for its low computer requirements that have allowed run the IDE in my Atom N270 netbook with no so much problems.

The icons are took from the Humanity and Tango free and open source icon packs. The first one is available in Ubuntu Linux distributions and the second one Google helps you to find it.

Nevertheless, thanks to the official documentation, Google and lot of people who posts their doubts on Internet, at last it has been possible to run Qt and OpenCV together in a simple graphical user interface Linux program.

Main Window

The Image Feature Detector GUI has a typical *Main Window* (Figure 4.4) with a *Multiple Document Interface*. The main widget is a window where there are a number of toolbars at the top, a status bar at the bottom and a central area where, as images are opened, the windows are showed in an ordered way.

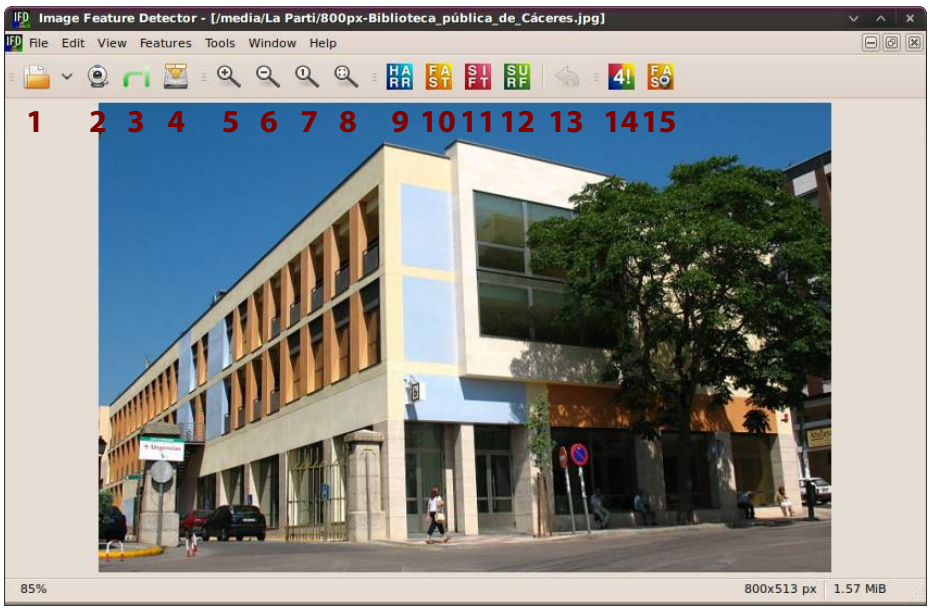


Figure 4.4. Image Feature Detector main window.

Following, the main window options are explained:

1. Opens a stored image.
2. Captures an image from the default computer webcam.
3. Captures an image from the RoboLab webcam.
4. Saves an opened image. Interesting option to save images with detected features.
5. Zooms in the image.
6. Zooms out the image.
7. Adjusts the image size to the window.
8. Shows the image with the original size.
9. Applies Harris detector to the image.
10. Applies FAST detector to the image.
11. Applies SIFT detector to the image.
12. Applies SURF detector to the image.
13. Resets the image to its original state.
14. Do4! Option: opens 3 new windows with the same image than the original one and appliesa detector on each window.
15. FAST Features in Real Time: shows the preview image of the system default camera and captures and detects in real time FAST features.

Through the main window it can access to another secondary windows, preferences or view options. Below is also explained each of the 4 parameter bars that appears when an detector is applied.

Startup Window

This window is aimed to serve as a presentation window. From it, the user can open a stored image, captures a new image from the default system camera or captures a new one from the Robocomp interface camera. In addition, at the bottom there are a *Recent Opened Files* button that when the user press it a menu is opened with a list of recent used images. If the user wants, this window can be avoided on each program starting by unchecking the *Show this dialog on startup* checkbox.

Figure 4.5. Image Feature Detector startup window.

Capture from Webcam Window

It shows the preview image of the system default camera and captures a frame to use it as experimental image.

Figure 4.6. Image Feature Detector Capture from webcam window.

Capture from RoboComp inteface Window

It shows the preview image of the RoboComp Robolab robot camera and captures a frame to use it as experimental image.

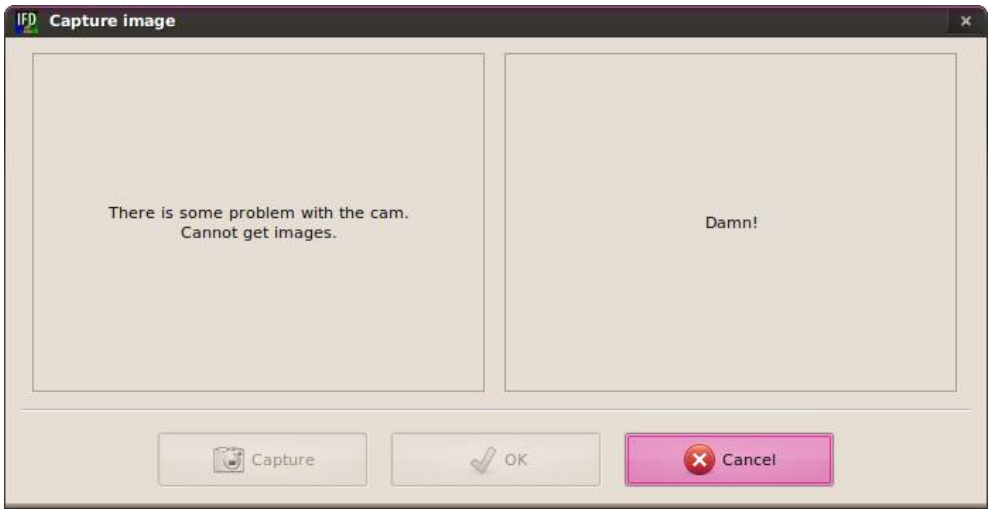


Figure 4.7. Image Feature Detector Capture from RoboComp interface window.

FAST Features in Real Time

It shows the preview image of the system default camera and captures and detects in real time FAST features. The parameters are the same than when features are detected from the main window.

Figure 4.8. Image Feature Detector FAST features in real time window.

Preferences Window

It shows a preferences window where the main program options can be configured. The options to be configured can be:

— To adjust the image size to the window space when windows are tiled or cascaded.

— To save a history with recent opened files. Additionally, the user can clean that list.

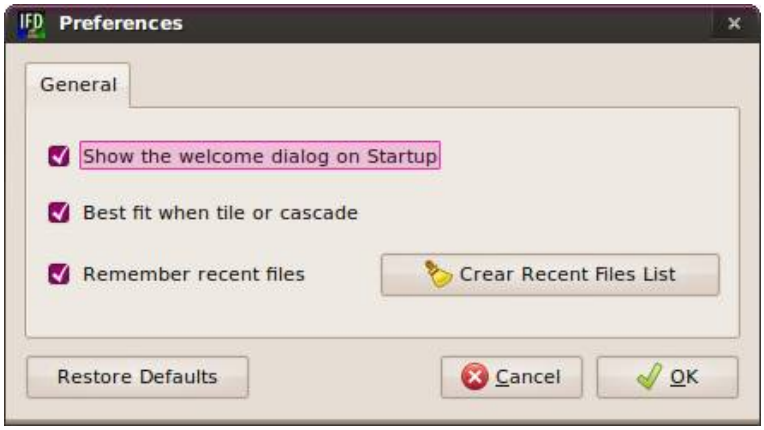


Figure 4.9. Image Feature Detector preferences window.

About Window

It shows a window where information about the license type, used libraries/resources and the author appears.

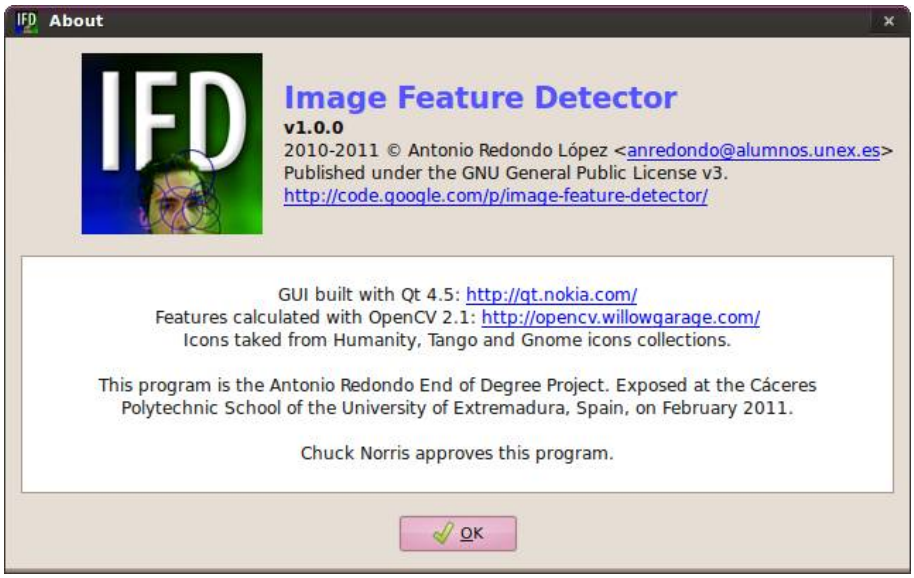


Figure 4.10. Image Feature Detector about window.

Applying Detectors: Parameters

When a detector button is pressed a new toolbar with the detector parameters appears. In that bar it can set the different parameters and options of each detector, and afterwards, applying to the image. It is also possible set the default value of each

parameter pressing the *Reset* button.

Harris parameters

The calculation of the Harris features are made throughout the below OpenCV C function. Corners in the image can be found as the local maxima of the destination image.

```
void cvCornerHarris(CvArr* image,
                    CvArr* harris_dst,
                    int blockSize,
                    int aperture_size=3,
                    double k=0.04)
```

- Sobel Aperture Size: aperture parameter for the Sobel operator. Before apply the Harris operator it is applied the Sobel one. The aperture size is the size of the square window with which doing the convolution. It can be 1, 3, 5 or 7. The IFD default value is 3.
- Harris Aperture Size: after applying the Sobel operator, the Harris covaration matrix M is calculated with an ApertureSize $\times$ ApertureSize window. If the value of the block size is 1 the Harris detector simply is not applied. The IFD default value is 2.
- k Value: it is the arbitrary k value of

$$R = \alpha\beta - k(\alpha+\beta)^2 = Det(M) - k\ Tr^2(M)$$

  The IFD default value is 0.01.

FAST parameters

The calculation of the FAST features are made throughout the following OpenCV C++ function:

```
void FAST(Mat& image,
          vector<KeyPoint>& keypoints,
          int threshold,
          bool nonmaxSupression=true)
```

- Threshold: threshold on difference between intensity of center pixel and pixels on circle around this pixel. The IDF default value is 50.
- Non-max Suppression: if it is true then non-maximum suppression will be applied to detected corners. The highest value of each corner will only be

visualized. By default is activated.

SIFT parameters

To calculate the SIFT features throughout the OpenCV C++ implementation we must first create a SIFT object with tree different structures:

```
SIFT mySIFT(CommonParams mySIFTCommon,
            DetectorParams mySIFTDetector,
            DescriptorParams mySIFTDescriptor);
```

These structures, explained below, let us configure the detector parameters. After create the object, we use the () operator to pass the desired Mat image to the SIFT object to calculate the features:

```
mySIFT(const Mat& img,
       const Mat& mask,
       vector<KeyPoint>& keypoints);
```

The detected keypoypoints are stored in the keypoints vector `keypoints`.

The detector parameters are set by tree structures. These parameters are sorted by kind: with the first structure `CommonParams` we can set the number of octaves and layers per octave. The remaining parameters of this structure set the first octave to take into account and how to measure the angle of each feature, whether the first angle of all or an average of them. The second structure `DetectorParams` sets the thresholds parameters of the detector. And the third structure `DescriptorParams` sets the descriptors parameters, but we are not going to use them. Image Feature Detector only allow to set the main relevant parameters of the detector which are enough in most cases.

```
SIFT::CommonParams mySIFTCommon(
            nOctaves,
            nOctaveLayers,
            SIFT::CommonParams::DEFAULT_FIRST_OCTAVE,
            SIFT::CommonParams::AVERAGE_ANGLE);
SIFT::DetectorParams mySIFTDetector(
            threshold,
            edgeThreshold);
SIFT::DescriptorParams mySIFTDescriptor(
            SIFT::DescriptorParams::GET_DEFAULT_MAGNIFICATION(),
            SIFT::DescriptorParams::DEFAULT_IS_NORMALIZE,
            SIFT::DescriptorParams::DESCRIPTOR_SIZE);
```

- Threshold: features must exceed this threshold to be processed. The IFD default value is 0.014.
- Edge Threshold: threshold of corners. The IFD default value is 10.

- Octaves: the number of octaves to be used for extraction. With each next octave the feature size is doubled. The IFD default value is 3.
- Layers/Octave: the number of layers within each octave. The default value is 1.
- Show Orientation: shows a radius in red inside the circumferences of each feature indicating its orientation. By default is activated.

SURF parameters

The calculation of the SURF features are made throughout the following OpenCV C function:

```
void cvExtractSURF(const CvArr* image,
                   const CvArr* mask,
                   CvSeq** keypoints,
                   CvSeq** descriptors,
                   CvMemStorage* storage,
                   CvSURFParams params);
```

The parameters of the detector are set by the below structure. The parameter *extended* indicates whether a descriptor with 64 or 128 elements, but we are not going to use the descriptors and hence, we are not interested in this parameter.

```
struct CvSURFParams {
  int extended;
  double threshold;
  int nOctaves;
  int nOctaveLayers;
}
```

- Threshold: Hessian threshold that keypoints must exceed to be extracted. The IFD default value is 4000.
- Octaves: the number of octaves to be used for extraction. With each next octave the feature size is doubled. The IFD default value is 3.
- Layers/Octave: the number of layers within each octave. The IFD default value is 1.
- Show Orientation: shows a radius in red inside the circumferences of each feature indicating its orientation. By default is activated.

Chapter 5
Conclusions

In this project we have viewed a sigh introduction to computer vision and its applications to the real world, a presentation to digital image processing and its relation with computer vision, and we have touched the world of robotics and the related Robolab laboratory.

Concerning to image feature detectors, we have view that, depending of the use we are going to do, it is preferable use a detector instead of other:

- If we need to extract features in real time, the solution is FAST without any doubt. If we need to extract scale-invariant features to tracking purposes and we have a high-end computer, then the optimum detector is SURF.
- If we do not need fast processing, FAST is not the most interesting solution, and with SIFT and SURF we can extract features with more information than FAST. Harris extracts similar features than FAST, although is slower.
- If we need to realize recognition tasks we can use SIFT or SURF, but it is demonstrated in Chapter 3 that SURF detector is better in all aspects than SIFT.

We have implemented in C++ with OpenCV and Qt a Linux program with GUI that by applying the four detectors to a same image visually compares them and quickly shows the results on screen. This program, Image Feature Detector, lets configure the different parameters of each detector.

In this End of Degree Project, I have carried out explanations, comparisons and implementation of features detectors and of any other element that was related with the project, and that have had as result this memory and the Linux program. But with any doubt, the increase of my own knowledge and skills related with computer vision science, C++ programming and English language have been the most enriching part of the project.

References

Books

Main references (in About Memory's Preface section this is explained)

[1] **Computer Vision: Algorithms and Applications**
Richard Szeliski – Springer, 2010 September.

[2] **Computer Vision: A Modern Approach**, 1st edition
David A. Forsyth, Jean Ponce – Prentice Hall, 2003.

[3] **Digital Image Processing**, 2nd edition
Rafael C. Gonzalez - Richard E. Woods – Prentice Hall, 2002.

[4] **Theory Of Applied Robots**, 2nd edition
Reza N. Jazar – Springer, 2010.

[5] **Feature Extraction and Image Processing**, 1th edition
Mark S. Nixon, Alberto S Aguado – Academic Press, 2002.

[6] **Learning OpenCV**, 1th edition
Gary Bradski, Adrian Kaehler – O'Really, 2008 September.

Scientific literature

[7] **Machine Perception Of Three-Dimensional Solids**
Lawrence G. Roberts – 1963.

[8] **The Analysis of Cell Images**
Judith M. S. Prewitt, Mortimer L. Mendelsohn – 1966.

[9] **A 3x3 Isotropic Gradient Operator for Image Processing**
Irwin Sobel – 1968.

[10] **A Combined Corner And Edge Detector**

Chris Harris, Mike Stephens – 1988.

[11] **Good Features to Track**
Jianbo Shi, Carlo Tomasi – 1994.

[12] **Distinctive Image Features from Scale-Invariant Keypoints**
David G. Lowe – 1999.

[13] **SURF: Speeded Up Robust Features**
Herbert Bay, Tinne Tuytelaars, Luc Van Gool – 2006.

[14] **Machine learning for high-speed corner detection**
Edward Rosten, Tom Drummond – 2006.

[15] **Faster and better: a machine learning approach to corner detection**
Edward Rosten, Reid Porter, Tom Drummond – 2008.

[16] **Exploring Visual Odometry for Mobile Robots**
Hatem Alismail – 2009.

Secondary references (in About Memory's Preface section this is explained)

[17] **RobEx: an open-hardware robotics platform**
J. Mateos, A. Sánchez, L. J. Manso, P. Bachiller, P. Bustos – 2010.

[18] **RoboComp: a Tool-based Robotics Framework**
Luis Manso, Pilar Bachiller, Pablo Bustos, Pedro Núnez, Ramón Cintas, Luis Calderita – 2010.

[19] **The spectral input to honeybee visual odometry**
L. Chittka, J. Tautz – 2003.

[20] **The computation of optical flow**
S. S. Beauchemin and J. L. Barron – 1995.

[21] **Deriving a 3-d description of a moving rigid object from monocular tv-frame sequence**
T. Bonde and H. H. Nagel – 1979.

[22] **Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover**
Hans P. Moravec – 1980.

[23] **SUSAN - A New Approach to Low Level Image Processing**
Stephen M. Smith and J. Michael Brady – 1993.

[24] **Scale-space filtering**
A. P. Witkin – 1983.

[25] **The structure of images.**
J. J. Koenderink – 1984.

[26] **Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention.**
T. Lindeberg – 1993.

[27] **Detection of local features invariant to affine transformations**
Krystian Mikolajczyk – 2002.

[28] **Local grayvalue invariants for image retrieval. IEEE Trans. on Pattern Analysis and Machine Intelligence**
C. Schmid, R. Mohr – 1997.

[29] **An affine invariant interest point detector**
Krystian Mikolajczyk and Cordelia Schmid – 2002.

[30] **Scale and affine invariant interest point detectors**
Krystian Mikolajczyk and Cordelia Schmid – 2004.

[31] **Performenace Evaluation of Corner Detection Algorithms under Affine and Similarity Transforms**
Farahnaz Mohanna, Farzin Mokhtarian – 2001.

[32] **Fast corner detection**
Trajkovic, M., Hedley, M. – 1998.

[33] **Evaluation of interest point detectors**
Schmid, C., Mohr, R., Bauckhage, C. – 2000.

[34] **Indexing based on scale invariant interest points**
Mikolajczyk, K., Schmid, C. – 2001.

[35] **Comparing and evaluating interest points**
Schmid, C., Mohr, R., Bauckhage, C. – 1998.

[36] **Interactive museum guide: Fast and robust recognition of museum objects**
H. Bay, B. Fasel, and L. van Gool – 2006 May.

[37] **Visual categorization with bags of keypoints**
C. Dance, J. Willamowski, L. Fan, C. Bray, and G. Csurka – 2004.

[38] **R. Fergus, P. Perona, and A. Zisserman**
Object class recognition by unsupervised scale-invariant learning – 2003.

[39] **A Computational Approach to Edge Detection**
John Canny – 1986.

[40] **A performance evaluation of local descriptors**
Krystian Mikolajczyk, Cordelia Schmid – 2005.

[41] **Head Pose Estimation in Face Recognition across Pose Scenarios**
M. Saquib Sarfraz, Olaf Hellwich – 2008.

Websites

[42] http://robolab.unex.es/
[43] http://www.robots.ox.ac.uk/ ~ vgg/research/affine/
[44] http://www.cs.ubc.ca/ ~ lowe/keypoints/

US patent 6,711,293 for SIFT detector:
[45] http://patft.uspto.gov/

The great official Qt 4.x reference documentation and its big on-line community:
[46] http://doc.qt.nokia.com/

OpenCV chaotic website:
[47] http://opencv.willowgarage.com/

Articles on English Wikipedia (not all articles are referenced explicitly):
Blob detection
Canny edge detector
[48] Computer stereo vision
[49] Corner detection
Difference of Gaussians
Edge detection
Feature (computer vision)
Feature detection (computer vision)
Gaussian function
Harris affine region detector
Hessian Affine region detector
[50] Odometry
[51] OpenCV
Optical flow
Prewitt
[52] Qt (framework)
Roberts Cross
Scale space
Scale-invariant feature transform
Sobel operator
SURF
[53] Template matching
Visual descriptors
[54] Visual odometry

A lot of searches on Google:
http://www.google.com/

Image Credits

Chapter 1

Figure 1.1 [1]
Figure 1.2 [1]
Figure 1.3 [42]
Figure 1.4 [42]

Chapter 2

Figure 2.1 Wikimedia Commons – Cáceres
Figure 2.2 Wikimedia Commons – Cáceres
Figure 2.3 [5]
Figure 2.7 [5]
Figure 2.9 [5]
Figure 2.11 [5]
Figure 2.12 [10]
Figure 2.13 [14]
Figure 2.14 [14]
Figure 2.15 [12]
Figure 2.16 [12]
Figure 2.17 [12]
Figure 2.18 [12]
Figure 2.19 [13]
Figure 2.20 [13]
Figure 2.21 [13]
Figure 2.22 [13]
Figure 2.23 [13]
Figure 2.24 [13]
Figure 2.25 [13]
Figure 2.26 [13]
Figure 2.27 [13]
Figure 2.28 [13]

Chapter 3

Figure 3.2 [14]
Figure 3.3 [14]
Figure 3.4 [14]
Figure 3.5 [14]
Figure 3.6 [14]
Figure 3.7 [13]
Figure 3.8 [43]
Figure 3.10 [13]
Figure 3.11 [13]

Chapter 4

Figure 4.1 [46]
Figure 4.2 [46]
Figure 4.3 [47]
Figure 4.4 Own creation
Figure 4.5 Own creation
Figure 4.6 Own creation
Figure 4.7 Own creation
Figure 4.8 Own creation
Figure 4.9 Own creation
Figure 4.10 Own creation