UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria dell'Informazione

MULTI–OBJECT AND MULTI–CAMERA ROBOTIC VISUAL SERVOING

VINCENZO LIPPIELLO

Tesi di Dottorato di Ricerca

Novembre 2003

Il Tutore
Prof. Luigi Villani

Il Coordinatore del Dottorato
Prof. Luigi P. Cordella

Dipartimento di
Informatica e Sistemistica

Dipartimento di Ingegneria
Elettronica e delle Telecomunicazioni

via Claudio, 21- I-80125 Napoli - ☎ *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

to Simona

ACKNOWLEDGMENT

The author wishes to thank professors Bruno Siciliano and Luigi Villani for their valuable tutoring during these last three years and for the estimate they have always demonstrated.

Very important has been the guide of Luigi Villani for the evolution of my Ph.D. formation, for the assistance during the development of the experiments at the PRISMA Lab, and for the writing of this thesis.

A special word of thanks is for Bruno Siciliano, who made this Ph.D. experience possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

During the last decade the use of visual systems for robotics applications has become a viable solution, which is often capable to offer a good cost/performance tradeoff. In fact, the effectiveness and autonomy of a robotic system operating in unstructured environments can be significantly enhanced if a visual system is adopted to achieve direct measurements of the state of the environment and of the task in progress.

The use of visual sensors may have high impact in applications where it is required to measure the pose (position and orientation) and the visual features of objects moving in unstructured environments. Typical industrial applications are, e.g., assembling of mechanical parts, edge/path following, object grasping; non-industrial applications are, e.g., automotive guidance, spatial and underwater robotics. In particular, visual sensors offer the possibility to extract multiple information from a workspace in a noninvasive manner. Moreover, in the last decade, high performance visual systems are becoming less and less expensive. This scenario is quite appealing for industries and researchers which are stimulated at developing innovative strategies that actual technology is able to support.

This thesis considers the problem of the real-time reconstruction of the pose of 3D objects moving within unstructured environments, using the information provided by a multi-camera visual system. The aim is to study in depth the main issues that are involved in the pose reconstruction process, and to develop a new and efficient pose reconstruction technique for the multi-object and multi-camera visual servoing.

The contents of the thesis have been organized into eight chapters and two appendices.

In Chapter 1 an introduction to the meaning of *visual servoing* for robotic applications is presented. Some examples of application of this control strategy are

illustrated to define the actual boundaries of the state of the art. The main features of visual servoing systems are introduced to provide a basic knowledge of the most important elements characterizing this field of research.

Chapter 2 presents the main elements to understand the rule and the operation of each element composing a *visual system*. In this thesis, the term "visual system" is used to indicate the set of hardware and software modules that make possible the acquisition of quantitative and qualitative information about the observed workspace. Without loss of generality, four modules are considered: the camera, the frame grabber, the image processing unit, and the image processing library.

Chapter 3 is devoted to the presentation of the mathematical models that are used for the cameras and for the 3D objects. In particular, the camera model is based on the *pin-hole* model and includes some distortion effects; the model used for the 3D objects is based on a specific and very efficient data structure, known as *BSP tree*, which may be directly derived starting from a more intuitive CAD representation (*boundary representation*).

In Chapter 4 two of the most important areas of *computer vision* are described; namely, the *image elaboration* and the *image interpretation*. An introduction to the most popular techniques of image elaboration and image interpretation are presented, with particular attention to those used in the proposed visual tracking algorithm. Further, the so-called *windowing* technique, used to reduce the image region to be elaborated, is described.

The issue of Chapter 5 is the problem of the *pose reconstruction* of a known moving object using image measurements. The geometric formulation of the pose reconstruction problem is presented with reference to the case of a fixed mono-camera system and to the more general case of a fixed multi-camera system, both making

use of local image features, e.g. corners and holes, as image measurements. The proposed solutions are based on the *Extended Kalman Filter* to resolve the complex and nonlinear equations resulting from the geometric formulation, both in the case of a mono and of a multi-camera system. Further, to reduce the computational complexity, a recursive formulation of the filter is proposed. Finally, a new adaptive formulation of the Extended Kalman Filter is presented to enhance the robustness with respect to the light condition.

In Chapter 6 the problem of the management of highly *redundant information* provided by a multi-camera system is presented. A new optimal technique for managing redundant visual information is proposed, that is based on a *pre-selection algorithm* and an *optimal selection algorithm*. It is shown that this two-step procedure allows a sensible reduction of the time spent for the redundancy management process. In fact, with this approach the image area to elaborate may be limited to a constant value, independently from the number of the employed cameras, providing an efficient and flexible tool for real time applications based on multi-camera systems.

In Chapter 7 the proposed reconstruction technique for simultaneous visual tracking of multiple objects using information provided by a multi-camera system is presented, while Chapter 8 shows the results of some experiments realized on a robotic system equipped with a stereo visual system.

Finally, Appendices A and B offer some implementation details of the BSP Trees structure and of the Extended Kalman Filter, respectively.

CHAPTER 1

INTRODUCTION

An introduction to the meaning of *visual servoing* for robotic applications is presented in this chapter. Some examples of applications of this control strategy are illustrated to define the actual boundaries of the state of the art. Moreover, the main features of visual servoing systems are introduced to provide a basic knowledge of the most important elements in this field of research. Finally, the specific topics covered in this thesis are briefly introduced.

1.1 Visual servoing

One of the most challenging ambitions of human kind has been the realization of machines mimicking the human capability to gather and elaborate complex information on the environment so as to interact with it in an autonomous manner. During this endeavor humans have supplied such machines with a large sensorial and elaboration capacity, and have designed their structure so as to replicate human limbs like arms and hands.

The two most important human senses providing sufficient information on an unknown environment for the execution of generic interaction tasks are the tactile sense and the *visual sense*. Devices able to partially imitate these human senses are

the force sensors and the *visual sensors*, respectively. In fact, tasks like parts mating or localization and grasping of moving objects may appear to be very simple for humans thanks to their capacity of interaction provided from such senses.

The visual sense, that is probably the most powerful among the human senses, is often lacking in many human-made devices. In fact, without visual information, manipulating devices can operate only in well-structured environments, where every object is known and can be found in a well-known pose (position and orientation). Thanks to the use of visual sensors, these devices could be used for applications where the geometry and the pose of the objects to be manipulated or avoided cannot be known in advance.

The integration of different types of interaction and field sensors has today become an indispensable step to increment the versatility and the application domain of modern automation devices. Until a few years ago, the cost of many kinds of these sensors was actually too high. Only recently, the integration of a visual system into the sensorial equipment has become affordable for many kinds of applications.

A visual system is a passive, remote and distributed sensor that captures information from a large region of the workspace in a non-invasive manner. Nowadays, it provides a partial imitation of the visual human sense in a relatively economical way. In fact, the cost of a CCD camera and of the relative image processing hardware has become quite competitive, with respect to the quality and the wealth of the provided information. With the increasing of real-time capabilities of such systems, vision is beginning to be utilized in the automatic control community as a powerful and versatile sensor to measure the geometric characteristics of the workspace. Moreover, the use of efficient real-time algorithms for visual applications, running on powerful and economic hardware, make it possible the use of visual information directly in the control loop, giving origin to what is known as *visual servoing*.

Visual servoing is a control strategy based on various kinds of information extracted from a sequence of images provided by a visual system. In general, this system may be composed by one or more cameras, computational system, and specific image processing algorithms. Like human sense during the execution of a specific task, the visual system may be specialized for measuring a specific features of interest, e.g. the presence of objects and obstacles, the presence of a target object, etc. This information, once available, may be used by a specific control unit to perform the assigned task. Probably, the main difficulty in the study and application of this technique resides in its intrinsic multidisciplinarity from image processing and sensor fusion, to real-time computation and control theory.

1.2 Applications of visual servoing

The use of visual systems in automation and robotics has found large diffusion especially during the last decade (see [17] and [5]). In fact, the reduction of the cost of the CCD camera sensors combined with the exceptional growth of the computational capacity of specialized hardware and common PC processors, have made possible the use of sophisticated image processing real-time algorithms at a very limited cost. The main application fields of this type of sensors are humanoid robots, robot manipulators, autonomous mobile vehicles, medical applications, etc.

The realization of robots similar to humans has been the goal of many research groups around the world. The replication of the human visual sense is one of the most important aspects of such goal. Many attempts have been made to construct robots with functional and somatic characteristics similar to human (see Fig. 1.1) and some important results have been obtained. For this kind of applications, visual servoing is used to provide some important capacities to the robot, as that of reconstructing a local map, following a path, avoiding an obstacle, climbing the stairs, playing with

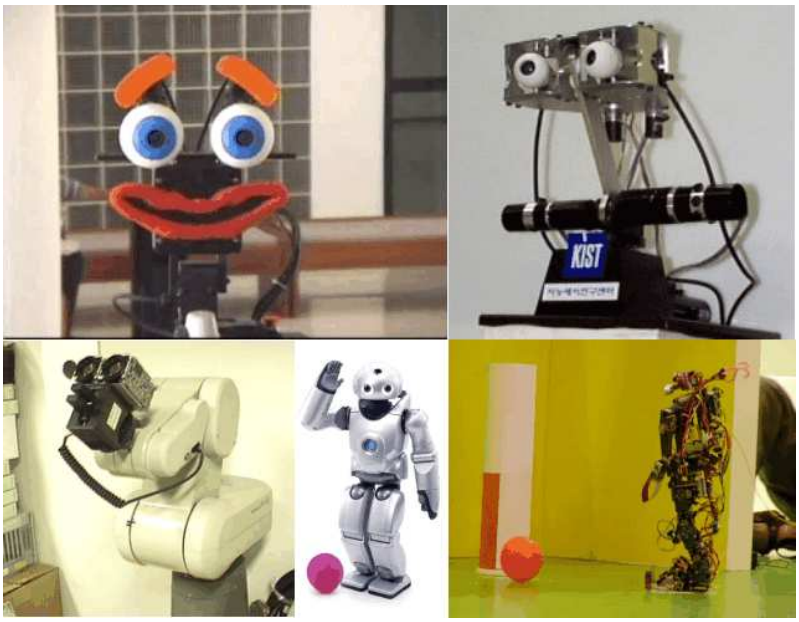

Figure 1.1. Visual servoing applications: human visual sense imitation

a ball, driving vehicle, etc. The execution of all these tasks requires a sophisticated visual capacity, which has to be provided in real time by a stereo visual system. Even though significant progress has been made in the latest years, the degree of autonomy comparable to human autonomy is still far to be reached.

Visual servoing of robot manipulators is an important and prolific field of research. Operations such as object recognition, localization and grasping in unstructured environments have been developed and tested by many research and industrial groups. In Fig. 1.2 some examples of manipulators controlled through a visual system are shown. The significant economic benefits achievable with the introduction of visual systems in industrial applications have determined a quick development of this technology. In fact, the introduction of visual servoing in industrial processes may relax the mechanical constraints supporting the work process of the robot, and may permit the realization of more complex tasks. Moreover, a multi-camera visual system may operate at the service of a whole robotic workcell, and thus its cost can

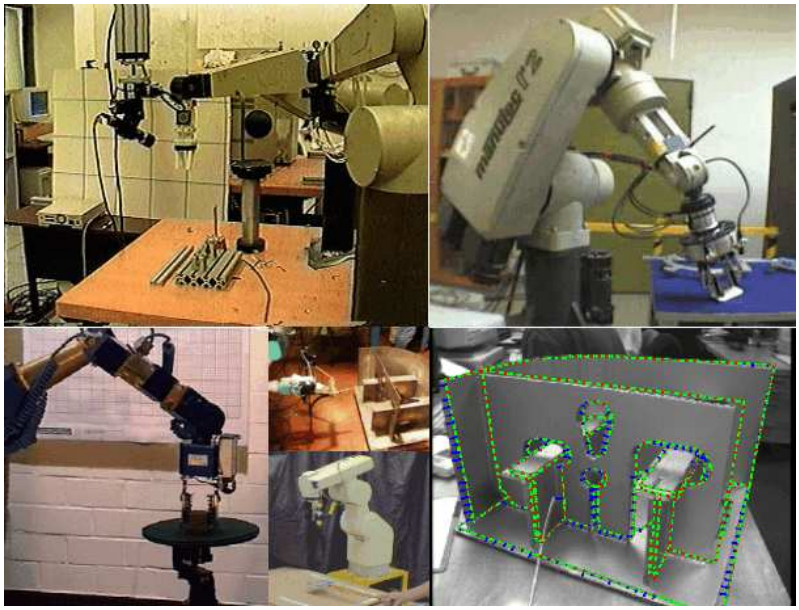

Figure 1.2. Visual servoing applications: parts mating and object manipulation

be actually shared among many robots.

The fusion of force and visual measurements may represent a powerful solution to perform tasks which require interaction with unknown environments. In these applications visual servoing can be used to localize the target and to guide the robot to an appropriate contact configuration; then, the force sensor may provide the information required for completion of the task.

Mobile robots make use of visual systems to achieve autonomy of motion in unknown environments. Thanks to the visual system, it is possible to gather and store information on the local environment that can be used to build a local digital map for navigation and obstacles avoidance. More sophisticated mobile robots are endowed with robotic arms, as shown in Fig. 1.3; they are used, e.g., for housekeeping or post delivering. Imagination is often the sole limitation to the utilization of such machines.

Figure 1.3. Visual servoing applications: autonomous mobile robot

Besides the previously mentioned applications, many other common processes have been the subject of feasibility studies on the employment of visual sensors. Some of the most successful applications can be found , e.g., in surgery, robotic games, and virtual reality (see Fig. 1.4). In surgery the use of specific robots equipped with a visual systems allows incrementing the precision and sensibility of a surgeon and eventually realizing operations remotely. Robotic pets are some of the recent commercial products using visual servoing.

1.3 Visual servoing of industrial robots

The main application of visual servoing in industrial robotics concerns with the control of the end-effector pose with respect to the pose of one or more objects and/or obstacles fixed or moving in the workspace. Many examples of common industrial tasks belong to this category, e.g., positioning or moving some objects, assembling

Figure 1.4. Visual servoing applications: surgery, games, virtual reality, etc

and disassembling mechanical parts, surface treatment (like painting), etc.

For robots supplied with a visual servoing system the control of the pose is realized using synthetic information extracted from a sequence of images, known as *image features*. The images are provided by a visual system, composed of one or more cameras mounted on the end effector of the robot or in a fixed pose with respect to the workspace. An image is a two dimensional (2D) projection of the observed scene, made by the couple lens system/CCD sensor of a camera. The process of reconstruction of the original three-dimensional (3D) pose of the observed object is known as *pose reconstruction* or *pose estimation*, and it is one of the most important processes involved in visual control. Often, when the object is moving in the workspace, the process of pose reconstruction is known as *visual tracking* to underline the dynamic context.

However, to realize visual servoing, it is not strictly necessary to explicitly reconstruct the 3D pose of the target. If a suitable control algorithm based on image features is available, visual servoing may be realized directly in the image feature

space without estimating the pose. The schemes of visual servoing based on visual tracking and image-space visual servoing belong to two of the main categories of visual servoing systems that will be presented with more details in the next section.

The integration of visual systems with force measurements is a promising field of research for industrial applications (see [2]). The potential of this solution is really promising because it ensures the robot capability of interacting with unstructured and unknown environments. Typically, pure visual servoing is used to guide the manipulator in a suitable initial pose with respect to the target and the programmed task. Then, the robot is taken in contact with the target using a suitable force control low. For some applications, like mechanical parts mating, visual servoing remains active also during the insertion to control the motion components that are not interested by the contact constraints in order to realize a specific relative motion trajectory.

For some industrial applications, budgetary limits may still constitute a problem for the implementation of a visual servoing system. Even though the cost of this type of hardware is becoming very competitive, the application tasks are becoming more and more complex and may require visual processing algorithms with significant costs. In fact, this control technique is still young and not completely settled; moreover, the complexity and variety of application tasks does not allow for a complete standardization of these algorithms.

1.4 Characterization of visual servoing systems

The incredible variety of control structures making use of visual feedback presented in the scientific literature does not allow a comprehensive presentation of all the aspects and details of visual servoing. To understand this variety, the main parameters used to classify and distinguish visual servoing systems are listed and described in the

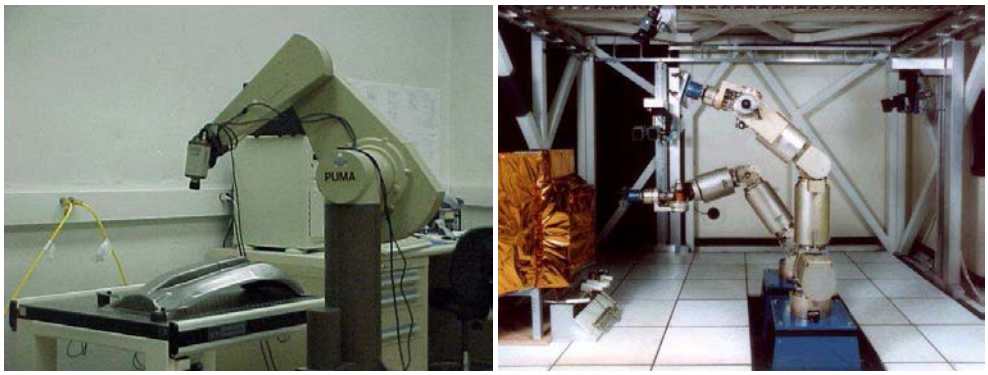

Figure 1.5. Visual servoing classification based on the camera(s) configuration. On the left (right) is shown an example of eyes-in-hand (eyes-out-hand) configuration

following:

- camera(s) configuration,
- observation point,
- control architecture,
- control error,
- model and calibration dependence.

1.4.1 Camera(s) configuration

Visual servoing systems may be classified with respect to its camera(s) configuration. Two main possibilities exist: *eye(s)-in-hand* configuration and *eye(s)-out-hand* configuration.

The first configuration corresponds to the case of one or more cameras mounted on the robot's end effector. In this case there exists a fixed relationship between the pose of the end effector and the poses of the cameras. Moreover, the images of the target objects depend on the robot pose.

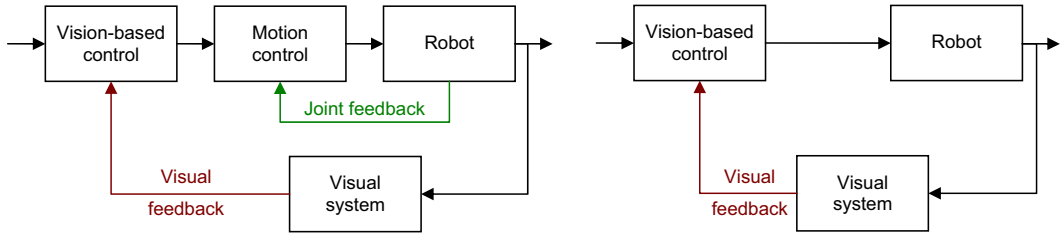

Figure 1.6. Visual servoing classification based on the control architecture. Left: indirect visual control scheme; right: direct visual control scheme

The second configuration describes the case of one or more cameras mounted on fixed or mobile bases in the workspace. In this case the images of the target objects are independent of the robot pose and the camera(s) may be fixed or moving with respect to the robot base frame. In Fig. 1.5 two examples of robotic systems that adopt different cameras configuration are shown. A hybrid configuration is also possible (but less usual), where one or more cameras are mounted in-hand and other are out-hand.

1.4.2 Observation point

This classification depends on whether or not the visual system observes the end effector of the visually guided robot, apart from the target objects. If the visual system observes only the target objects an *endpoint-open-loop* type of visual control is realized; if both the targets and the end effector are observed, then an *endpoint-closed-loop* is realized. In the first case, the accuracy in positioning the end effector with respect to the target object depends on the accuracy of robot kinematic calibration and camera calibration. In the second case, the positioning accuracy does not depend on robot and camera calibration.

1.4.3 Control architecture

A visual servoing control loop may be realized using two different architectures (see Fig. 1.6). In the first one, known as *direct visual control*, the visual feedback is directly

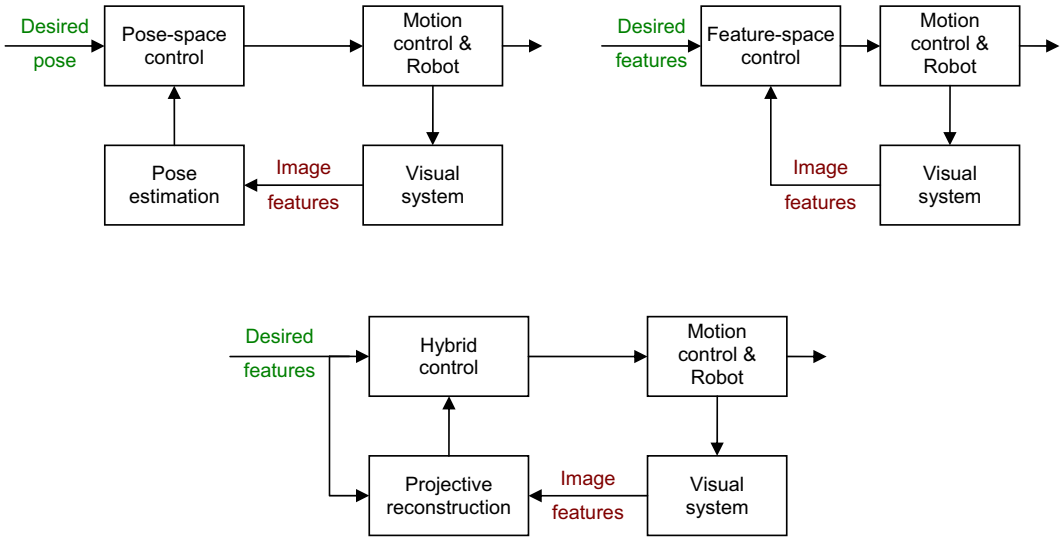

Figure 1.7. Visual servoing classification based on the control error. On the top-left (top-right) is shown a position-based 3D (image-based 2D) visual servoing. On the bottom is shown a hybrid $2\frac{1}{2}$D visual servoing

used to evaluate the joint control law. For this scheme, very fast visual systems are required because the rate of the visual system coincides with the control rate. In the second control architecture, instead, visual feedback is used to provide the set point of an inner motion controller. This type of control scheme, known as *indirect visual control*, can be further divided into two categories: *dynamic look-and-move* and *static look-and-move*. In both cases, the external visual control loop may be slower than the inner motion control loop. In the dynamic scheme the visual system continuously evaluates and updates the motion set point, while in the static scheme the visual system is used only to localize the target position before the motion starts.

1.4.4 Control error

Depending on the type of control error, three different types of visual servoing strategies may be identified: *position-based* (3D), *image-based* (2D), and *hybrid* ($2\frac{1}{2}$D) visual servoing. Position-based visual servoing requires the evaluation of the 3D pose of the target objects through specific pose estimation algorithms. The reconstructed

poses are used by a standard pose controller as any different type of pose measurement. The image-based visual servoing, instead, makes use of the 2D image features (e.g. the object contours) as feedback measurements and of the desired aspect of these features directly into a suitable feature-space controller. Finally, in hybrid visual servoing a position-based approach is used only for some pose components, while an image-based solution is used to reconstruct the remaining components. Figure 1.7 illustrates the three types of visual servoing strategy.

1.4.5 Model and calibration dependence

The last type of classification is based on the dependence of the visual servoing system on the CAD model of the target objects and/or on the camera calibration.

Typically, the dependence on the models of the target objects is determinate by the need to reconstruct 3D pose information from the available 2D image measurements. This type of dependence is often present in such visual servoing systems that employ a single camera systems.

In general, the dependence on the calibration of the camera system is determinate by the request to reconstruct the absolute pose of the target objects with respect to the base frame. For this reason it is more frequent for the position based visual servoing schemes and less frequent for the image based visual servoing schemes.

1.5 Proposed visual servoing system

The main research result presented in this thesis is a new visual tracking algorithm for the simultaneously dynamic estimation of the absolute pose of one or many known objects, moving in an unstructured environment, using images provided from a multi-camera system. It represents a very flexible solution to the realization of a generic visual servoing system. In fact, thanks to its structure based on processing of local

image areas, the algorithm may use a generic number of cameras, in any configuration, at no additional in computational cost. Moreover, it is able to simultaneously track the pose of many known objects.

With respect to previously cited visual servoing schemes, the algorithm presented here may be used both for an eye(s)-in-hand and for an eye(s)-out-hand configuration. In fact, the cameras may be positioned in any configuration and the algorithm is capable to configure itself in an automatic manner. Being a multi-object algorithm, both the endpoint-open-loop visual control and the endpoint-closed-loop visual control may be realized. Moreover, considering its high computational efficiency, it may be used both in a direct-visual-control scheme and in a dynamic-look-and-move scheme. Finally, it is a position-based and calibration-dependent algorithm, because the 3D absolute pose is estimated, and it is model-based, since the CAD model of the target objects is required.

However, the required object knowledge consists in a simple description of the object surfaces. In the case of polyhedral objects, each surfaces is described as an ordered sequence of points describing its contour (boundary representation). This CAD model is further elaborated by the algorithm to achieve a very efficient data representation based on a binary tree structure known as Binary Space Partitioning tree (BSP tree).

This data structure is the base to an efficient algorithm for managing the image features (e.g. corners and holes) which supports the feature extraction process. For each camera, this algorithm is able to recognize and select an optimal and minimal set of local image features to be extracted —through a local elaboration on the image— and used by the pose estimation algorithm.

The core of the pose estimation process is based on a suitable dynamic formu-

lation of the Kalman Filter, able to estimate the pose of the observed moving object. Moreover, the filter provides a prediction of the pose of the object, at the next sampling time, that is keenly used by the algorithm for feature selection to increase the accuracy and computational efficiency of the visual tracking system.

CHAPTER 2

VISUAL SYSTEM

The term *visual system* has been used, in the literature, to indicate a wide variety of systems in function of the specific field of application where they are employed. For the purpose of this work, the term visual system will indicate the set of hardware and software modules, that makes possible the acquisition of quantitative and qualitative information about the observed workspace. Without loss of generality, the main elements composing a visual system are essentially four: the camera with its optical sub-system, the frame-grabber, the image processing unit (specialized or generic), and the image processing library. This chapter presents the main elements to understand the rule and operation of each of these elements.

2.1 Camera

A camera is a passive, remote and distributed device that captures information on a wide region of the observed space in a non-invasive manner. It is a complex system composed of heterogenous sub-systems, as shown in Fig. 2.1, summarized as follows:

- solid state photo-sensible sensor,
- optical sub-system,

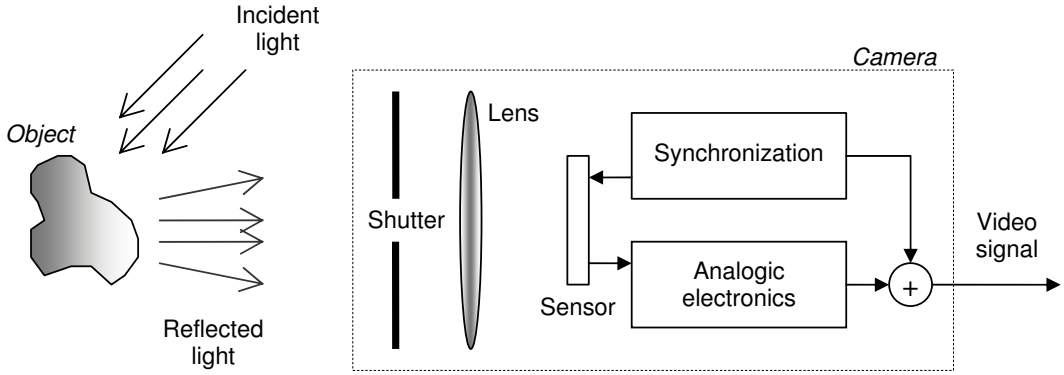

Figure 2.1. Scheme of the main elements of a camera

- analogic electronics for signal conditioning.

2.1.1 Solid state photo-sensible sensor

A solid state sensor is composed of a large number of photo-sensible elements, named *pixels*, able to transform light energy into electrical energy. Each pixel accumulates an electrical charge, that is proportional to the incident light intensity, integrated on the whole period of exposition. The *line scan* cameras use a single line of pixels to build up seamless two-dimensional images of moving objects (a fax machine is a low-end example). They are employed for applications where the linear motion of a part may be used to reconstruct a two dimensional image during time. The *area scan* cameras, instead, use a two-dimensional array of sensible elements and they are the base for the cameras employed in *machine vision*[1]. For this reason, only this type of sensor will be considered in the following.

The mean and most diffused photo-sensible solid state sensors with superficial scansion are the *CCD* and *CMOS* sensors, which are based on the photo-electrical effect of semi-conductor materials.

[1]Term used here to indicate the use of visual information to realize the assigned task.

CCD. A *Charge Coupled Device sensor* is composed of a rectangular matrix of pixels. By virtue of the photo-electrical effect, when a photon strikes the surface of the semi-conductor it creates some free electrons. Therefore, each element accumulates a charge depending on the time integral of the incident light. At the end of the *exposition period*, this charge is passed, via a suitable transporting mechanism like an analogic shift-register, to an output amplifier. Simultaneously, the pixel is discharged. The signal produced by the amplifier is further elaborated in order to produce a *standard video signal*.

CMOS. A *Complementary Metal Oxide Semi-conductor* sensor is composed of a rectangular matrix of photo-diodes. The junctions of each photo-diode are precharged, and they progressively discharge when are struck by the photons strike. An amplifier, which is integrated in each element, transforms this charge into a current or voltage level.

Differently from a CCD sensors, the pixels of a CMOS sensor are not integrating elements because, after they have been activated, they measure a quantity instead of a volume. In this way, a saturated pixel can not "overflow" and influence a close pixel. This property prevents the blurring of the image, that instead distresses the CCD sensors.

Another important difference between this type of sensor is that the CCD sensors sample all the pixels simultaneously, when the photo-sensible elements are transferred through the shift-registers, while in the case of the CMOS sensors the pixels positioned at the opposite sides of the sensor are characterized by different exposition time. This delay may cause problems for images of fast moving objects. In fact, if the motion of an observed object is very fast with respect to the camera, then the recorded images may be blurred. The pixels are sensible to the time integral of the light intensity over the exposition period, and thus the observed object may appear

blurred and lengthened along the motion direction. The use of image sequences with this characteristics as field measurements may produce coarse errors, especially for applications like visual servoing.

A conventional camera based on the use of a film makes use of a mechanical *shutter* to expose the film for a time period whose duration is small with respect to the velocity of the motion. For CCD sensors, an electronic shutter may be derived by discharging the pixels until a time instant just before the beginning of the desired exposition time. Only the charge integrated in the remaining period, properly called exposition time, will be transferred via the shift-register. A disadvantage of this technique is that the accumulated charge decreases with the duration of the exposition time, and causes a reduction of the signal/noise ratio of the image. However, this effect may be contained by increasing the aperture of the camera and/or increasing the illumination of the scene.

The visual information has to be elaborated by a numerical processor, so that the measurement of the light intensity of each points of the image has to be transformed into a finite number of digital codes. It is clear that a *spatial sampling* has to be adopted, because the image is composed of infinite points, as well as a *temporal sampling*, because the image varies with time. By considering the previous functional description, it is obvious that the CCD and CMOS sensors realize the task of spatial samplers, while the electrical shutters assume the role of time samplers.

2.1.2 Optical sub-system

The optical sub-system is composed of one or more lenses[2], which have the task to

[2]A lens is a centered dioptric system, that is a portion of transparent material, with a fixed refraction index, delimited by two spherical surfaces, that may not have the same curving radius (e.g. one of the two may be a plane). The curving centers of these surfaces determine a straight line called *optical axis*.

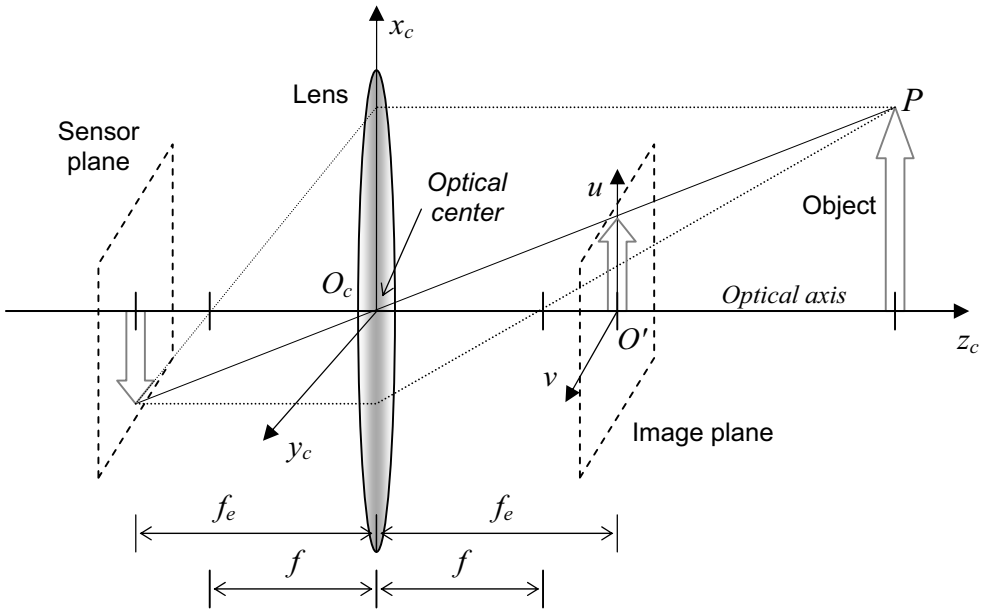

Figure 2.2. Elementary image creation and the referred frames according to the pin-hole model

focus the light reflected by the observed objects on the sensor plane. To understand its functioning for the simple case of one *thin lens*[3], the classic *pin-hole* model will be used. This model, shown in Fig. 2.2, defines a reference frame O_c-$x_c y_c z_c$ fixed with respect to the camera (*camera frame*), with the z-axis corresponding to the *optical axis*, i.e. the axis orthogonal to the lens plane, which contains the two centers of curvature of the spherical surfaces delimiting the lens. The center of this frame corresponds to the *optical center*, also known as "center of the projections", that is the geometric center of the lens for the case of thin lenses. The lens plane corresponds to the plane (x_c, y_c) of the camera. The sensor plane is parallel to the lens plane and it is positioned at a distance f_e, i.e. the *effective focal length*, different with respect to the nominal focal length f.

Due to the projective effect, as shown in Fig. 2.2, the image projected on the

[3]A lens is said thin when its thickness (the distance between the two dioptrics of the lens) is small with respect to the radius of curvature of its two spherical surfaces.

sensor plane is reversed with respect to the original. To obtain a non-reversed image, a fictitious plane is considered, named *image plane*, that is positioned on the other side of the lens with respect to the optical center, at a distance f_e. On the image plane a new reference frame O'-uv is defined, called *image coordinate frame*, where O' corresponds to the intersection of the image plane with the optical axis and is called *principal optical point*, while the u-axis and v-axis are chosen parallel to the x_c-axis and y_c-axis, respectively.

The relationship between the coordinates (x^c, y^c, z^c) of a generic point P in the camera frame and the corresponding image coordinates (u, v) in the image frame of its projection on the image plane[4] may be easily derived using the rules of *perspective projection*. Exploiting the property of similarity between the triangle individuated by point P, its normal projection on the optical axis, and the point O_c, and the triangle defined by the points O', O_c, and the projection of P on the image plane, the following relationships can be derived:

$$\frac{u}{f_e} = \frac{x^c}{z^c} \tag{2.1a}$$

$$\frac{v}{f_e} = \frac{y^c}{z^c} \tag{2.1b}$$

also known as *perspective transformation*. It is worth remarking that the image coordinates are not the output coordinates of the camera, but they do not take into account spatial discretization realized by the solid state sensors, that will be considered in the next chapter. Finally, notice that (2.1) is valid only in theory, because in reality the lenses and the assembly of the optical system are always affected by

[4]The projection of a point P on the image plane is the 2D point determined by the intersection of the image plane with the *optical ray* (the straight line joining the point P and the optical center). Notice that the image projection process transforms a 3D point of the observed space into a 2D point, determining a loss of information. Therefore, this transformation is not reversible because the image coordinates of a point determine only a spacial optical ray.

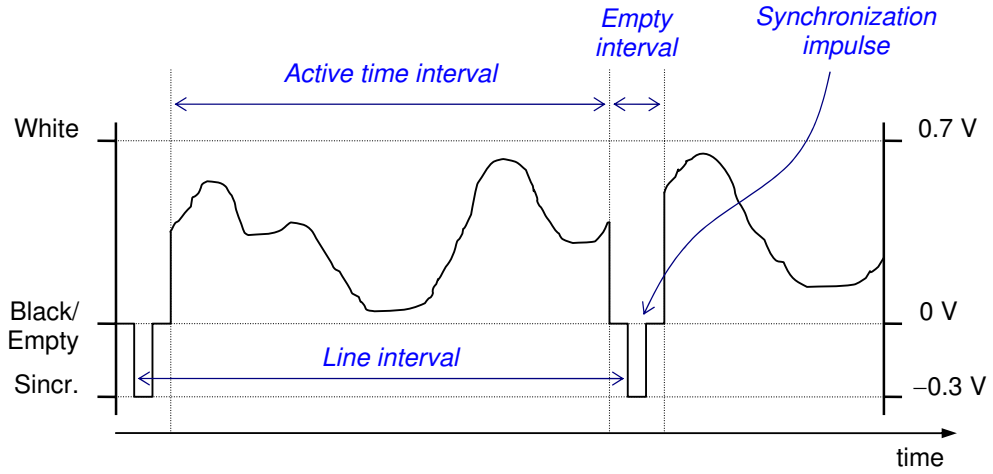

Figure 2.3. Typical standard video signal

various imperfections, which worsen the quality of the image.

2.1.3 Analogic electronics for signal conditioning

In the case of a black and white camera[5] with a CCD sensor, the output amplifier of the sensor produces a signal that is elaborated by an analogic electronic circuit, which is developed to generate one of the standard video signals: the *CIRR* standard, which is adopted in Europe and Australia, or the *RS170* standard, which is adopted in the US and in Japan. The video signal has a peak-peak voltage of 1V, that represents the light intensity in a sequential manner for each scanned line, as shown in Fig. 2.3.

The whole image is divided into a number of lines (625 for the CIRR standard and 525 for the RS170 standard) which have to be scanned sequentially. The scanning path proceeds horizontally along each line, from the top to the bottom, considering first the even lines and then the odd lines. Therefore, the whole image is obtained as a composition of two successive half-frames. This technique is called *interlacing*

[5]The color cameras are supplied with special CCD sensors sensible to the three fundamental color: red, blue, and green. Some professional models use three different sensors for each fundamental color.

as it allows the image to be refreshed at either the frame frequency or the half-frame frequency. In the first case the refresh frequency is 25Hz for the CIRR standard and 30Hz for the RS170 standard, while in the second case the frequency is doubled, but with half of the vertical resolution.

The signals are of analogic type (for this reason the corresponding cameras are known as *analogic cameras*) and then they cannot be input directly to a numerical elaboration system. Special A/D converters, known as *frame grabbers*, are used to convert the video signal into a digital format. A frame grabber performs sampling and quantization of the analogic video signal, and then it stores the values in a 2D array (*frame store*). These values represent the spatial samples of the image and the whole array can be refreshed at full or half-frame rate.

In the case of cameras with CMOS sensors, thanks to the CMOS technology that allows an A/D converter to be integrated directly into each pixel, the output signal of the camera is directly a 2D digital array. Moreover, the access to each pixel may be realized in a random manner. With respect to the CCD cameras, this characteristic allows a substantial increase of the refreshing frequency, provided that only a limited image area is accessed.

2.2 Frame grabber

The digitized image used by machine vision systems is a representation, sampled with respect to space, of the continuous function $I(u, v)$, which defines the intensity of the incident light at the point of coordinates (u, v) of the image plane. The function I, after sampling and quantization is stored into the frame store. The samples are known as image elements or pixels, and their amplitude is named *grey level* or *grey value*. Each row of the frame store corresponds to a line interval of the analogic video signal, as illustrated in Fig. 2.3.

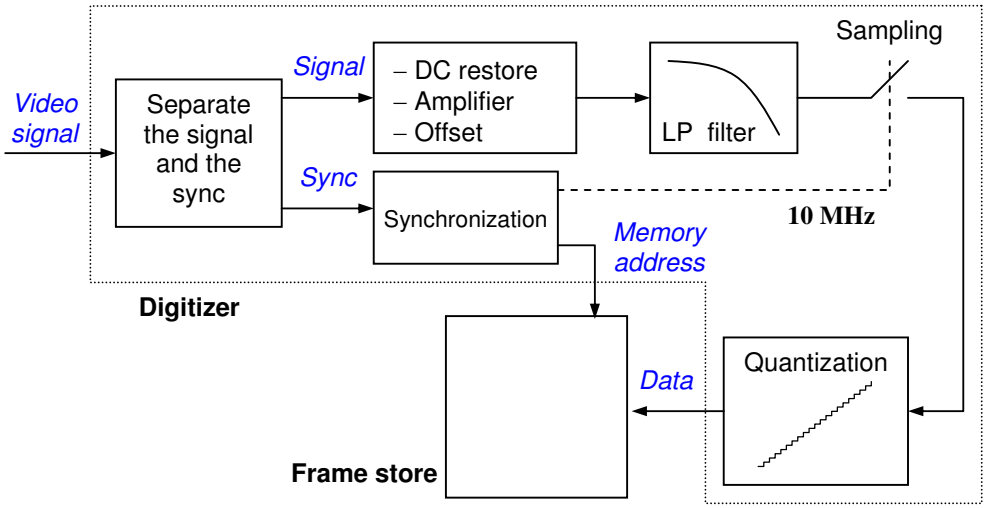

Figure 2.4. Scheme of the digitizing process

A scheme of the digitizing process is shown in Fig. 2.4, and it is characterized by the following elements:

- DC restore and offset,
- signal conditioning,
- sampling and shape ratio,
- quantization.

2.2.1 DC restore and offset

The analogical video signal may accumulate an offset on the DC component during the transmission and the amplification. The "DC restore" process eliminates this offset that, otherwise, may produce an alteration of the lightness level of the image and of the perception of the contrast level.

2.2.2 Signal conditioning

Before digitization, the video signal has to be filtered to reduce the aliasing effects and,

in the case of color cameras, to extract the *chrominance signal* from the composed video signal. This type of filtering has some negative effects like the reduction of contrast and sharpness, and above all the introduction of a delay due to the phase lag of the filter. Also, the conditioning stage of the signal has to introduce a gain, before the digitization, to improve the signal/noise ratio for dark scenes. For this type of image, in fact, the voltage levels of the video signal are very low.

2.2.3 Sampling and shape ratio

Whenever available to the sampler, the portions of the video signal corresponding to the photo-sensible elements of the camera should be constant. Instead, they turn out to be distorted due to the limited band of transmission line and the analogic supporting circuits. This signal has to be sampled to generate the corresponding value $I_f(r, c)$ of the frame-store pixel, where r and c are the row and column indices, respectively. Notice that the pixels of the frame store are not necessarily mapped as the pixels of the camera sensor. As illustrated in a schematic way in Fig. 2.4, to realize in a correct way the sampling of the signal and its storing into the frame store, a synchronization information of the video signal is required.

A digitizer samples the video signal at a frequency f_d, that is typically chosen equal to an integer multiple of the frequency of the synchronization impulse f_h. The sampling does not necessary align the samples to the output of the closest photo-sensible elements. In fact, if the frame store has a dimension of 512×512 pixels[6], then the digitizer has to sample the video signal, so that each line should have 512 samples, regardless of the number of photo-sensible elements of the sensor lines. In this case, the digitized pixels are not accurate samples independent of the entity of the

[6]This format is very popular, especially for low cost applications. Unfortunately, the CIRR signal is composed of 575 active lines for each image and then the last 63 lines (11%) are typically wasted. New digitizers and frame stores are able to capture the whole CIRR images.

incident light; this situation may determine serious problems for some machine vision algorithms, e.g. edge detectors, which operate under the assumption of independence between the pixels.

This "double" sampling, due to the sensor and the frame grabber, modifies the *shape ratio* (the ratio between the vertical and horizontal dimension) of the memorized image. This effect depends on the dimension of the elements of the photo-sensible sensor as well as on the *sampling ratio* β, defined as follows:

$$\beta = \frac{\text{Number of sensor pixels for line}}{\text{Number of frame-store pixels for line}}.$$

This quantity is very important for the definition of a mathematical model of the camera, that will be presented in the next chapter.

2.2.4 Quantization

The last step of the digitizing process is the quantization of the analogic video signal. Each sample is quantized into a n-bit integer with a value between 0 to $2^n - 1$. Typically the black reference corresponds to 0, while the highest level of white corresponds to $2^n - 1$.

The quantized signal $x_q(t)$ may be expressed as a function of the original signal $x(t)$ as follows:

$$x_q(t) = x(t) + e_q(t)$$

where $e_q(t)$ is the quantization noise, assumed with an uniform distribution in the interval $[-\frac{1}{2}, \frac{1}{2}]$ and squared mean $\overline{e^2} = \frac{1}{12}$. The corresponding signal/noise ratio will be SNR $= \overline{e^2}$.

2.3 Image processing unit

The dimensioning of the image processing unit is certainly one of the most important

issues for real-time application. The elaboration of the image sequences provided by one or more cameras via the corresponding frame grabbers may require a large processing capability, depending on the kind of image elaborations that has to be realized. In fact, for a low resolution black and white camera with 512×512 pixels, the dimension of the data to elaborate is about 2,09 Megabits; for a camera with 782×582 pixels the dimension grows to about 3,64 Megabits, and so on. If the whole image has to be elaborated with complex algorithms, then the required image processing unit has to be properly dimensioned and designed to respect the real-time constraints.

For some application, a personal computer, supplied a frame grabber board for PCI slots, may be used as an image processing unit. The processing capability of modern processors may be often sufficient, especially for mono-camera applications when the time constraints are not severe. This solution is especially suitable for those applications where only a limited area of the image has to be processed or when simple image features have to be extracted. In all these cases, a PC-based solution offers many advantages, apart from the limited cost, as the possibility of using a unique PC platform to implement the whole visual servoing system.

If the simultaneous elaboration of the whole images provided by many cameras is required, or whether the image features are very complex, then a specialized image processing unit may be necessary. At the moment, the cost of specialized units has become more competitive, but they still remain too expensive for many types of application. This specialized hardware is often integrated directly into the frame grabber and is also available on a common PCI board. Typically, it is composed of an external interface for the connection to many types of cameras and to one or more monitors, of one or more specific memory areas, and of one or more dedicated processing nodes such as DSPs and/or common PC processors. In many cases it is

possible to connect two or more cameras to one single image processing unit, which is able to manage them simultaneously. These devices may be programmed to realize a generic image processing algorithm, both for low-level processing and for complex image feature extraction. However, especially in the case of more cameras, a further central processing unit, like a host PC, may be necessary to realize some indispensable centralized operations on the image data provided by each processing unit. A typical case may be the pose reconstruction, where the features extracted from each camera have to be suitably composed and elaborated to estimate the pose of the observed object.

2.4 Image processing library

The image processing library is the collection of the algorithms for low and high-level image processing. The low-level processes are used to realize elementary image processing, such as the evaluation of the histogram of the grey levels, the transformation of the image into a binary format, the image filtering, and so on. The high-level processes are used to realize the image segmentation, the centroid evaluation, the corner detection, the edge detection, and, in general, the feature extraction.

In conjunction with much specialized hardware and the associated drivers, sophisticate image processing libraries are often available. The cost of these algorithms may be very high, and strongly depends on the complexity of the elaboration capability. However, these commercial libraries may be insufficient and limitative for sophisticated use, and often the corresponding source codes are not provided. Therefore, a custom library has to be realized to satisfy the necessity of specific applications.

CHAPTER 3

MODELLING

The *mathematical models* are a fundamental instrument for the development, simulation, realization, and management of experimental and innovative systems. This chapter is devoted to the presentation of the mathematical models of the camera and of the 3D object geometry. In particular, the model of the camera is based on the *pin-hole* model and includes some distortion effects, while the model of the 3D object geometry is based on a specific and efficient data structure, known as *BSP tree*, which is very popular for applications of 3D animation.

3.1 Camera

The types of cameras used for robotic and industrial applications with respect to high precision cameras, have the following characteristics: a) the image resolution depends on the spatial digitalization and is generally low; b) the employed lenses are non-metric standard lenses and present important geometric distortion, especially "wide angle" lenses; c) the assembly is characterized by significative internal mismatch[7]. For

[7]The mechanical assembly of industrial cameras is realized with low tolerances, and thus the image plane may not be orthogonal to the optical axis, the center of the sensor may not correspond to the principal optical point, and so on.

these reasons the camera model has to consider both the geometrical distortions and the mechanical mismatch of the optical structures, in order to allow for an effective compensation of the corresponding errors.

The parameters characterizing a generic model of a camera may be divided into three main categories: *internal parameters*, *external parameters*, and *distortion parameters*. The internal parameters characterize the relationship between the image coordinates of a point and the corresponding spatial coordinates, expressed with respect to the camera frame. The external parameters characterize the geometric relationship between the camera(s) frame and the base frame. Finally, the distortion parameters characterize the model of geometric distortions.

In this section, two different models are considered. The first one is a classic pin-hole model without distortion. The second model, that may be considered as an extension of the first one, considers also the main distortion effects. In the following, a short introduction to geometric distortion and the corresponding models is presented.

3.1.1 Model without distortion

A simple mathematical model representing a generic camera in an ideal condition of absence of any type of distortion will be derived in this section. The classical pin-hole camera model is considered. Consider the camera frame O_c-$x_c y_c z_c$ fixed with respect to the camera, as shown in the pin-hole model of Fig. 3.1. The (fictitious) image plane (u, v) is parallel to the camera's plane (x_c, y_c), and is placed at a distance f_e (effective focal length) from the origin of the camera frame along the z_c-axis in the opposite direction with respect to the sensor.

Consider the vectors $\boldsymbol{p}$ and $\boldsymbol{p}^C$, which represent the position of a generic observed point P with respect to the base frame and the camera frame, respectively. It is known that the coordinate transformation (rotation and translation) between these

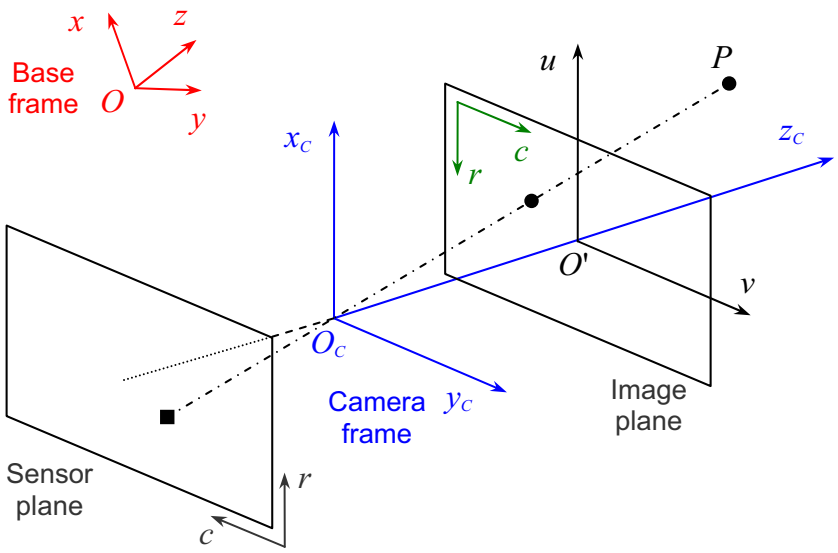

Figure 3.1. Reference frames for the camera using the pin-hole model

two vectors may be represented as follows:

$$\boldsymbol{p}^C = \boldsymbol{o}^C + \boldsymbol{R}^C \boldsymbol{p} \tag{3.1}$$

where $\boldsymbol{o}^C$ and $\boldsymbol{R}^C = \{r_{i,j}\}$, with $i, j = 1, 2, 3$, are the translation vector and the (3×3) rotation matrix defining the position and orientation of the base frame with respect to the camera frame, respectively.

On the image plane, consider the image coordinate frame O'-uv, as shown in Fig. 3.1, where O' corresponds to the principal optical point and the axes u and v are parallel to the corresponding axes x_c and y_c. The image coordinates of the generic observed point P can be computed using the *perspective transformation*, in (2.1) that can be rewritten in a compact form as follows:

$$\begin{bmatrix} u \\ \\ v \end{bmatrix} = f_e \begin{bmatrix} \dfrac{x^C}{z^C} \\ \\ \dfrac{y^C}{z^C} \end{bmatrix} . \tag{3.2}$$

Finally, the address of the pixel on the frame grabber, corresponding to the

digitized image of the projection of point P on the sensor, can be computed as

$$\begin{bmatrix} r \\ \\ c \end{bmatrix} = \begin{bmatrix} r_o \\ \\ c_o \end{bmatrix} + \begin{bmatrix} s_u & 0 \\ \\ 0 & s_v \end{bmatrix} \begin{bmatrix} u \\ \\ v \end{bmatrix} \tag{3.3}$$

where (r_o, c_o) indicates the address of the principal optical point O', and s_u and s_v are the *row* and *column scale factors*, respectively. These latter quantities define the *theoretical shape ratio* and their values depend (see Section 2.1) on the dimension of the photo-sensible elements as well as on the sampling ratio β of the digitizer, according to the following relationships:

$$s_u = \frac{1}{\beta a}$$
$$s_v = \frac{1}{b}$$

where a and b are the dimensions of the photo-sensible elements along the u-axis and the v-axis, respectively. The dimensions of both s_u and s_v are pixel/m. Therefore, an object with size $U \times V$ on the sensor will have a corresponding size $\|s_u\|U \times \|s_v\|V$ expressed in pixel. Notice that, due to the adopted conventions, the quantity s_u is negative, while the quantity s_v will be positive. Further, r and c correspond respectively to the row and column address of the pixel of the digitized image plane hit by the projection ray of point P, because the axes x_c and y_c (and so u and v) have been chosen parallel to the row and column directions, respectively.

In view of (3.2) and (3.3), it is possible to derive the following relationship:

$$\overline{u} = \frac{u}{f_e} = \frac{r - r_o}{f_u} = \frac{x^C}{z^C} \tag{3.4a}$$
$$\overline{v} = \frac{v}{f_e} = \frac{c - c_o}{f_v} = \frac{y^C}{z^C} \tag{3.4b}$$

where $(\overline{u}, \overline{v})$ define the normalized image coordinates in the so-called *normalized image plane*[8] and the new quantities $f_u = s_u f_e$ and $f_v = s_v f_e$ are known as *row focal length*

[8]The normalized image plane is a special fictitious image plane placed at a

and *column focal length*, respectively. The normalization of the image plane allows employing the two quantities f_u and f_v in the place of the three quantities f_e, s_u, and s_v, which always appear as products. Further, it is also possible to derive the relationship between the normalized image coordinates and the spatial coordinates of the observed point expressed in the base frame, using Eq. (3.1).

If the ratio between the row-row and column-column distances must be equal to q, then the ratio $|f_u/f_v| = |s_u/s_v|$ must be equal to q^{-1}. Many of the conventional cameras provide a rectangular image with a ratio of 4/3 between the horizontal and the vertical size.

3.1.2 Geometric distortion

The geometric distortion regards the position of the *image point* (i.e., the projection of the observed point on the image/sensor plane) on the image plane. Because of different imperfections of the shape and of the assembly of the lenses composing the optical sub-system, the expressions (3.4) are no longer valid and have to be replaced with the expressions:

$$u' = u + \delta_u(u, v) \tag{3.5a}$$

$$v' = v + \delta_v(u, v) \tag{3.5b}$$

where u and v are the image coordinates without distortion, that are not observable, while u' and v' are the corresponding distorted coordinates; the position errors δ_u and δ_v depend on the position of the point itself.

To correct the distortion effects in an efficient manner it is necessary to analyze the different causes of distortion and model their effects on the image plane. Three

distance $z^C = 1$, instead of $z^C = f_e$, from O^C. Another definition of normalized image plane may be obtained by performing the normalization with respect to the optical center so that the row-row (column-column) distance in pixel is equal to 1. In this case it is $s_u = 1$ ($s_v = 1$) and $f_u = f_e$ ($f_v = f_e$).

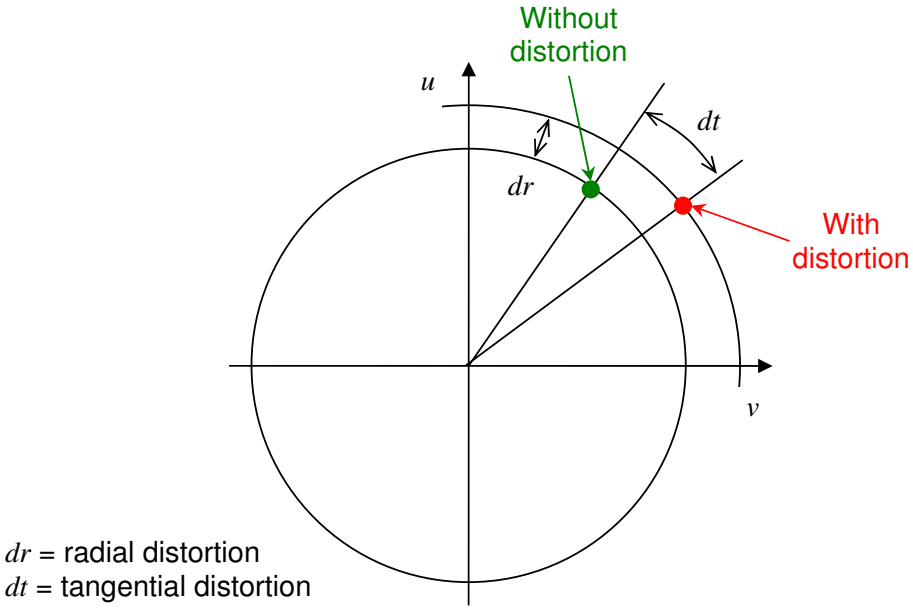

Figure 3.2. Radial and tangential distortion

types of distortion have been considered. The first type depends on the imperfections of the shape of the lenses, which cause a radial error of the image position; the second and third type of distortion depend on the erroneous assembly of the optical subsystem, which generate both a radial and a tangential error of the image position. Figure 3.2 shows both the effects of the radial and of the tangential distortion on the image point.

Radial distortion. The radial distortion causes a shift of the ideal image point towards an internal (negative) or an external (positive) direction, with respect to the principal optical point. Typically, this effect is caused by an imperfect curvature of the shape of the lens. A negative radial shifting is called *cylindric distortion*. It determines the thickening of most external pixels, toward the principal optical point, and a reduction of the scale factors (see Fig. 3.3). A positive radial shifting of the image point is called *pincushion distortion*. It determines the stretching of the most external pixels and an increment of the scale factors.

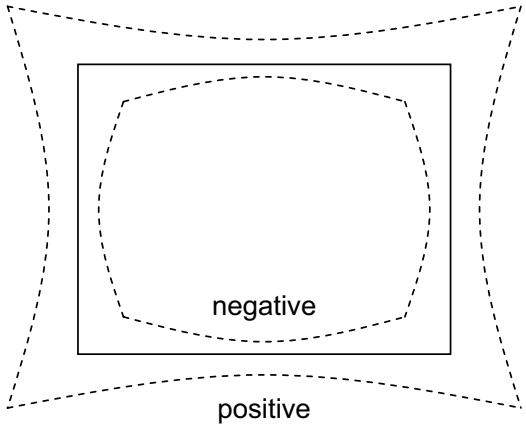

Figure 3.3. Radial distortion. The dotted lines show the effects of a positive and a negative radial distortion

The radial distortion for a centered lens may be modelled as

$$\delta_{\rho r} = k_1\rho^3 + k_2\rho^5 + k_3\rho^7 + \dots$$

where ρ is the radial distance from the principal image point, and $k_1, k_3, k_4, \dots$ are the coefficients of the radial distortion. To each image point of polar coordinates (ρ, φ), it corresponds a radial distortion along each direction that depends only on the radial distance ρ. Therefore, by taking in account the equalities:

$$u = \rho\cos(\varphi)$$
$$v = \rho\sin(\varphi),$$

the radial distortion may be directly expressed with respect to the image coordinates as follows:

$$\delta_{ur} = k_1 u\left(u^2 + v^2\right) + O\left[(u,v)^5\right] \tag{3.6a}$$
$$\delta_{vr} = k_1 v\left(u^2 + v^2\right) + O\left[(u,v)^5\right], \tag{3.6b}$$

where $O(\cdot)$ represents the order of infinite.

Decentralizing distortion. The optical sub-systems are subject to erroneous alignment of the optical center of the assembled lenses. These defects determine the

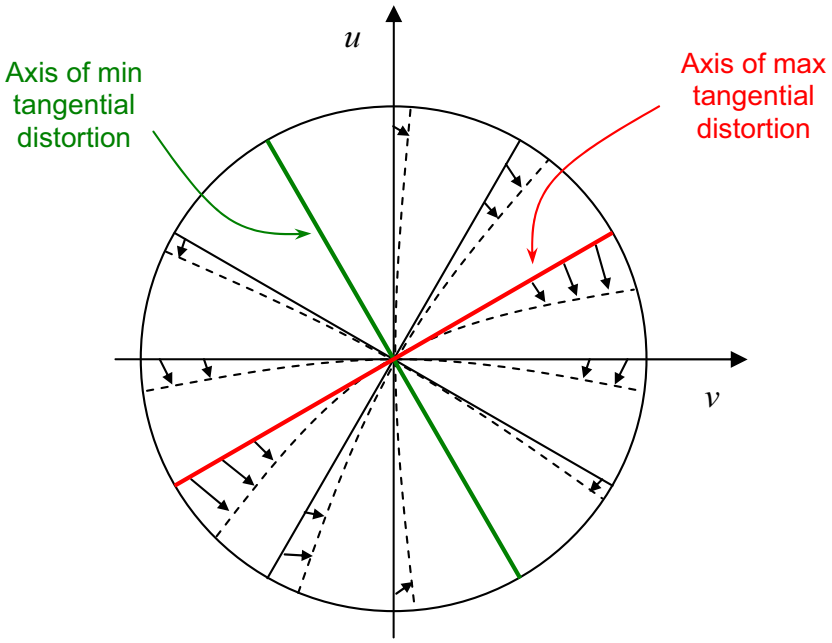

Figure 3.4. Tangential distortion. The dotted lines show the effects of the tangential distortion

so-called *decentralizing distortion*. This type of distortion presents both radial and tangential components, which may be computed as:

$$\delta_{\rho d} = 3\left(j_1\rho^2 + j_2 + \rho^4 + \dots\right)\sin(\varphi - \varphi_{od}) \tag{3.7a}$$

$$\delta_{td} = \left(j_1\rho^2 + j_2 + \rho^4 + \dots\right)\cos(\varphi - \varphi_{od}) \tag{3.7b}$$

where φ_{od} is the angle between the positive axis u and the maximal tangential distortion line, as shown in Fig. 3.4.

The resulting distortion along the u-axis and v-axis may be expressed in terms of $\delta_{\rho d}$ and δ_{td} as follows:

$$\begin{bmatrix} \delta_{ud} \\ \delta_{vd} \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} \delta_{\rho d} \\ \delta_{td} \end{bmatrix}. \tag{3.8}$$

Considering that $\cos(\varphi) = u/\rho$, $\sin(\varphi) = v/\rho$, from (3.7) and (3.8) the following

expression can be obtained:

$$\delta_{ud} = \xi_1 \left(3u^2 + v^2\right) + 2\xi_2 uv + O\left[(u, v,)^4\right] \tag{3.9a}$$

$$\delta_{vd} = 2\xi_1 uv + \xi_2 \left(u^2 + 3v^2\right) + O\left[(u, v,)^4\right] \tag{3.9b}$$

where $\xi_1 = -j_1 \sin(\varphi_o d)$ and $\xi_2 = -j_1 \sin(\varphi_o d)$.

Prismatic distortion. The *prismatic distortion* depends on the imperfections of the shape of the lenses as well as on the camera assembly, e.g. the inclination of the planes of the lenses or of the sensor. This type of distortion may be suitably modelled as the presence of a thin prism into the optical sub-system, that determines an increment of the radial and tangential distortion. The mathematical model is the following:

$$\delta_{\rho p} = 3\left(i_1\rho^2 + i_2 + \rho^4 + \dots\right)\sin(\varphi - \varphi_{op}) \tag{3.10a}$$

$$\delta_{tp} = \left(i_1\rho^2 + i_2 + \rho^4 + \dots\right)\cos(\varphi - \varphi_{op}) \tag{3.10b}$$

where φ_{op} is the angle between the positive axis u and the maximum tangential distortion axis, as shown in Fig. 3.4. Defining $\sigma_1 = -i_1 \sin(\varphi_{op})$ and $\sigma_2 = i_2 \cos(\varphi_{op})$, and using a similar assignment as before the model of the distortion along the u and v axes may be evaluated as

$$\delta_{up} = \sigma_1 \left(u^2 + v^2\right) + O\left[(u, v,)^4\right] \tag{3.11a}$$

$$\delta_{vp} = \sigma_2 \left(u^2 + v^2\right) + O\left[(u, v,)^4\right]. \tag{3.11b}$$

Complete distortion model. The effects of three types of distortion have been considered. It is important to notice that even though the decentralizing and prismatic distortion have similar mathematical models, these correspond to different distortion effects having different maximum distortion axes. When all the distortion effects are present simultaneously, the global distortion may be computed as the sum of the elementary effects. Combining (3.6), (3.9), and (3.11) and ignoring the terms of order

greater than three, the complete distortion model is achieved:

$$\delta_u(u,v) = (g_1+g_2)u^2 + g_4uv + g_1v^2 + k_1u\left(u^2+v^2\right) \tag{3.12a}$$

$$\delta_v(u,v) = g_2u^2 + g_3uv + (g_2+g_4)v^2 + k_1v\left(u^2+v^2\right) \tag{3.12b}$$

where $g_1 = \sigma_1 + \xi_1$, $g_2 = \sigma_2 + \xi_2$, $g_3 = 2\xi_1$, and $g_4 = 2\xi_2$.

3.1.3 Complete model of the camera

Considering the distortion along the u-axis and v-axis according to (3.5), the relationship between the ideal image point (u,v) and its effective pixel position is given by

$$u + \delta_u(u,v) = \frac{r - r_o}{s_u} \tag{3.13a}$$

$$v + \delta_v(u,v) = \frac{c - c_o}{s_v}. \tag{3.13b}$$

By considering the quantities

$$\hat{u} = \frac{r - r_o}{f_u} \tag{3.14a}$$

$$\hat{v} = \frac{c - c_o}{f_v}, \tag{3.14b}$$

which represent the image normalized coordinates with distortion, equations (3.13) can be rewritten in the form

$$\overline{u} = \frac{u}{f_e} = \hat{u} - \frac{\delta_u(u,v)}{f_e} \tag{3.15a}$$

$$\overline{v} = \frac{v}{f_e} = \hat{v} - \frac{\delta_v(u,v)}{f_e}. \tag{3.15b}$$

Notice that the true values of u and v cannot be obtained from the captured points, because they are corrupted by noise and distortion. Therefore, the arguments of the distortion functions are replaced with $\hat{u}$ and $\hat{v}$, i.e.

$$\frac{u}{f_e} \cong \hat{u} - \frac{\hat{\delta}_u(\hat{u},\hat{v})}{f_e} \tag{3.16a}$$

$$\frac{v}{f_e} \cong \hat{v} - \frac{\hat{\delta}_v(\hat{u}, \hat{v})}{f_e}. \tag{3.16b}$$

These approximations are reasonable because the distortion corresponding to the true image projection is about equal to the distortion corresponding to the measured image projection.

By defining the normalized distortion coefficients $\overline{g_1}$, $\overline{g_2}$, $\overline{g_3}$, $\overline{g_4}$ and $\overline{k_1}$ (obtained by dividing the corresponding quantities by f_e), and replacing them into (3.4), (3.12), and (3.16), the following camera model is obtained:

$$\overline{u} = \frac{x_c}{z_c} = \hat{u} + (\overline{g_1} + \overline{g_2})\hat{u}^2 + \overline{g_4}\hat{u}\hat{v} + \overline{g_1}\hat{v}^2 + \overline{k_1}\hat{u}\left(\hat{u}^2 + \hat{v}^2\right) \tag{3.17a}$$

$$\overline{v} = \frac{y_c}{z_c} = \hat{v} + \overline{g_2}\hat{u}^2 + \overline{g_3}\hat{u}\hat{v} + (\overline{g_2} + \overline{g_4})\hat{v}^2 + \overline{k_1}\hat{v}\left(\hat{u}^2 + \hat{v}^2\right). \tag{3.17b}$$

Notice that those equations are linear with respect to the distortion coefficients and this property may be used to derive a camera calibration procedure (see [43] and [41]).

This model may be used to compensate for the distortion effects, e.g. to estimate the optical ray of each observed point. The pixel coordinates r and c of an observed point determine the coordinates $\hat{u}$ and $\hat{v}$ by virtue of (3.14). These values are then used into (3.17) to compute the compensated normalized image coordinates and, finally, the optical ray.

The described distortion model considers only some types of distortion but this does not mean that it is not able to represent any type of distortion. In fact, this model, that has been derived considering the most important type of distortion, has been formulated in a polynomial form. Therefore, the distortion parameters computed during camera calibration will represent not only the distortion effects considered in the model but, through its interpolation capability, they will capture also other types of distortion, if at all present.

3.2 Three-dimensional object

Modelling the geometry of the objects is a crucial step involved into an object-oriented machine vision process. The realization of a task requiring tracking of target objects requires the knowledge of the geometry of the objects and their relative poses in the observed workspace. If such information has to be provided by a visual system, then a model of the geometry of the target objects, suitably defined for the kind of involved image elaboration process, is required. Unfortunately, in most cases a Cartesian CAD model (*boundary representation* or *B-reps*) of the objects is available (or derivable), but it cannot be directly employed by sophisticated real-time image processing algorithms, while a more efficient representation of the object geometry is necessary to reduce computational complexity. As a matter of fact, two different requirements have to be satisfied: it is important to have an *easily derivable representation* that allows a manual description of the geometry of the object; also, it is necessary to have an *efficient representation* that allows the implementation of real-time visual servoing algorithms.

A good trade-off between these demands may be obtained using a simple boundary representation to describe the object geometry, and a more complex and computationally efficient object representation that may be automatically derived from the previous one. In the next sections, a possible boundary representation of *manmade*[9] objects and a specific data structure representing a recursive and hierarchical partition of an n-dimensional space into convex subspaces, known as *BSP tree*, are presented. In particular, the chosen boundary representation is very simple

[9]The so called *manmade* objects belong to an object class that contains all objects which may be represented (or approximated) as a union of planar surfaces, e.g. polyhedral objects. Further, the contours of each surface of these objects have to be representable (or approximable) with a polygonal determined by an ordered sequence of points. This means that a curved contour has to be approximated as a polygonal contour via a suitable spatial sampling.

and accessible for a manual geometric description of the object; the BSP tree data structure is presented, both in a general version and in a custom version, that may be automatically derived from boundary representation.

3.2.1 Boundary representation

This section is not aimed at presenting in general the Cartesian CAD modelling of three-dimensional objects, also known as *boundary representations* or *B-reps*, but is focused on the presentation of a specific model used to implement the visual tracking algorithm, proposed in this work. In particular, the considered boundary representation has three specific properties:

- it is suitable for representing manmade objects;
- it has to be easily constructed manually;
- it has to describe the object with two hierarchical levels of representation: the top level describes the set of the *feature points* of the object, while the bottom level describes the object surfaces using the information of the first level.

In this work, the so-called *feature points* are all the points which allow the contours of the object to be fully described. Generally, they are found by the discontinuities in the direction of the contours and by the intersection of multiple contours. In the case of curved contours, a suitable spatial sampling is required to approximate the contour with a polygonal line.

The considered B-reps may be constructed in two steps. During the first step all the feature points of the object are individuated and measured with respect to a chosen object frame O_O-$x_O y_O z_O$, fixed with the object. During the second step an easy description of the object surfaces is achieved by choosing the sequences of feature points delimiting the considered surface. The sequences of feature points are ordered

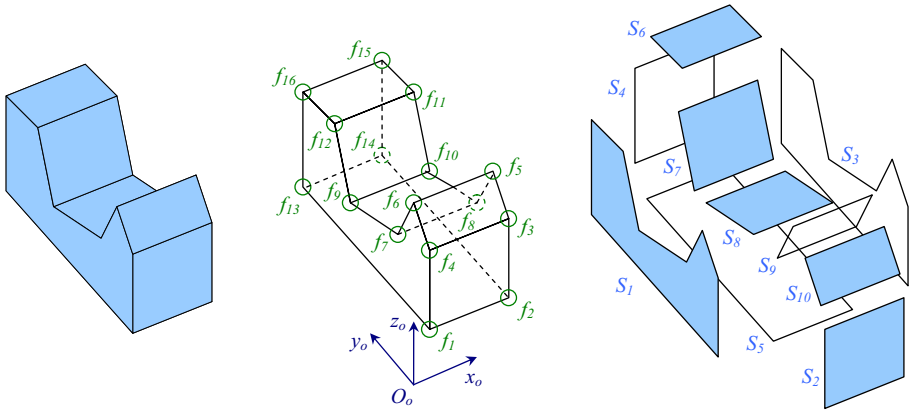

Figure 3.5. Example of Cartesian CAD model of a 3D manmade object. Left: object; center: object feature points (top level of representation); right: object surfaces (bottom level of representation)

in anticlockwise direction with respect to the outgoing unit vector orthogonal to the object surface. With this choice, it will be possible to recognize the external side of a surface.

In Fig. 3.5 an example of manmade object is shown, along with its feature points and its surfaces. In the first level of representation, 16 feature points have been found, which are sufficient to represent all the contours of the object. Each point (f_i, with $i = 1, \ldots, 16$) is represented by its position with respect to the object frame. In the second level of representation 10 surfaces are found and described using the previous feature points. Table 3.1 reports the representation of the surfaces with the corresponding ordered sequences of feature points, as described above. Notice that it is not important which is the first point of the sequence but only its order.

3.2.2 Binary space partitioning trees

In machine vision and in most applications involving computation of 3D geometric models, the task of generating images of objects, depending on their geometry, their relative pose, and the selected point of view, is of fundamental importance. Performing this task requires determining, for each image of a video sequence, the spatial

Surface	**Feature points**
S_1	$\{f_1 f_4 f_6 f_7 f_9 f_{12} f_{16} f_{13}\}$
S_2	$\{f_1 f_2 f_3 f_4\}$
S_3	$\{f_2 f_{14} f_{15} f_{11} f_{10} f_8 f_5 f_3\}$
S_4	$\{f_{16} f_{15} f_{14} f_{13}\}$
S_5	$\{f_1 f_{13} f_{14} f_2\}$
S_6	$\{f_{12} f_{11} f_{15} f_{16}\}$
S_7	$\{f_9 f_{10} f_{11} f_{12}\}$
S_8	$\{f_9 f_7 f_5 f_{10}\}$
S_9	$\{f_6 f_5 f_8 f_7\}$
S_{10}	$\{f_6 f_4 f_3 f_5\}$

Table 3.1. Boundary representation. Ordered sequences of feature points corresponding to the surfaces of the object of Fig. 3.5

relations between objects: how they might *intersect* each other, and how they may *occlude* each other. The manmade objects, rather than being one piece, are often composed by many pieces (or they may be so "represented"), e.g. by many polygons forming the faces of polyhedra. The number of pieces depends on the number and the complexity of the target objects. Computing spatial relations between n polygons by brute force entails comparing every pair of polygons, and then it would require $O(n^2)$ operations. For the case of 1,000 polygons, this would mean 10^6 operations, which is much more than necessary.

The number of operations can be substantially reduced to anything from $O(n \log_2(n))$ when the objects interpenetrate (in the previous example this corresponds to less than 10^4), to as little as constant time, $O(1)$, when they are somewhat separated from each other. This can be accomplished by using *binary space partitioning trees*, also called *BSP Trees* or *partitioning trees* (see [40]). They constitute a computational representation of space that simultaneously provides a *search structure* and a *representation* of geometry. The reduction in number of operations occurs because BSP Trees provide a kind of "spatial sorting". In fact, they are a generalization

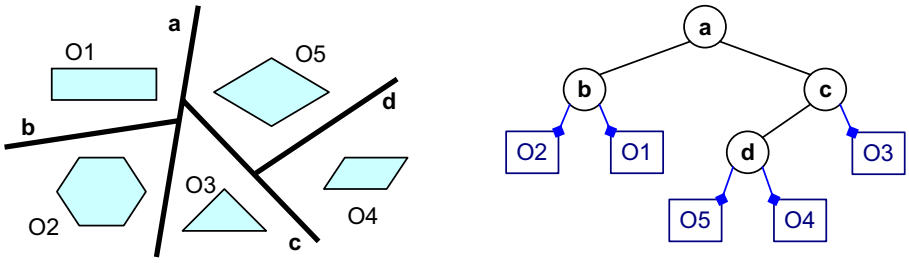

Figure 3.6. BSP tree representation of inter-object spatial relations for a 2D case. Left: spatial partitioning; right: binary tree

to dimensions grater than unity of binary search trees, which have been widely used for representing sorted lists.

Figure 3.7 gives an introductory example showing how a binary tree of lines, instead of points, can be used to "sort" five geometric objects, as opposed to sorting symbolic objects such as names.

Constructing a BSP tree representation of one or more manmade objects involves computing the spatial relations between polygonal faces once and encoding these relations in a binary tree (see Fig. 3.6). This tree can then be transformed and merged with other trees to quickly compute the spatial relations (for visibility and intersections) between the polygons of two moving objects.

BSP Trees achieve an elegant solution to a number of important problems in geometric computation by exploiting two very simple properties occurring whenever a single plane separates (lies between) two or more objects:

1. any object on one side of the plane cannot intersect any object on the other side,

2. given a viewing position, objects on the same side as the viewer can have their images drawn on top of the images of objects on the opposite side (*Painter's Algorithm*).

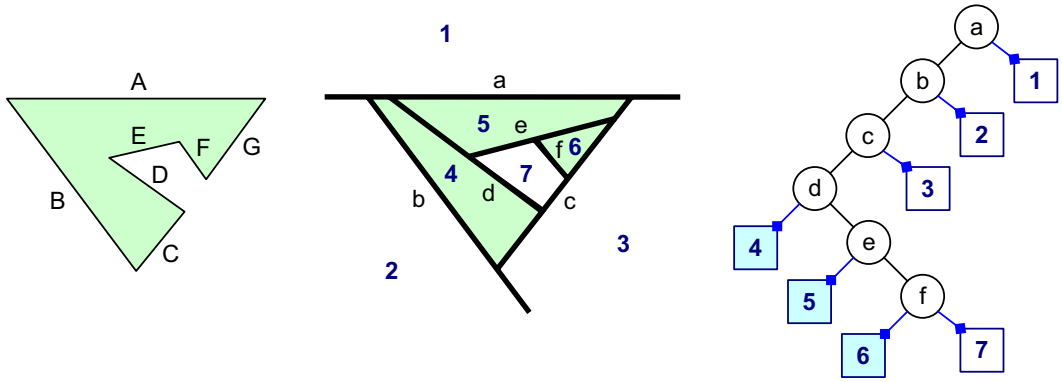

Figure 3.7. BSP tree representation of intra-object spatial relations for a 2D case. Left: original boundary representation; center: spatial partitioning; right: binary tree

These properties can be made independent of dimension if we use the term *hyperplane* to refer to planes in 3D, lines in 2D, and in general for n-space, to an $(n-1)$-dimensional subspace defined by a single linear equation. An example of the second property is showed in Fig. 3.8.

To determine visibility, all that is required is choosing at each tree node which one of the two branches to draw first based solely on which branch contains the viewer. No other single representation of geometry inherently answers questions of intersection and visibility for a scene of 3D moving objects. And this is accomplished in a computationally efficient and parallelizable manner.

A partitioning tree is also a program for performing intersections between the hyperplane half-spaces and any other geometric entity. Since subdivision generates increasingly smaller regions of space, the order of the hyperplanes is chosen so that following a path deeper into the tree corresponds to adding more detail, yielding a multi-resolution representation. This leads to efficient intersection computations.

BSP tree data structures have been used for many other different applications earning significant success [3]. In this work their capacity to make visibility orderings

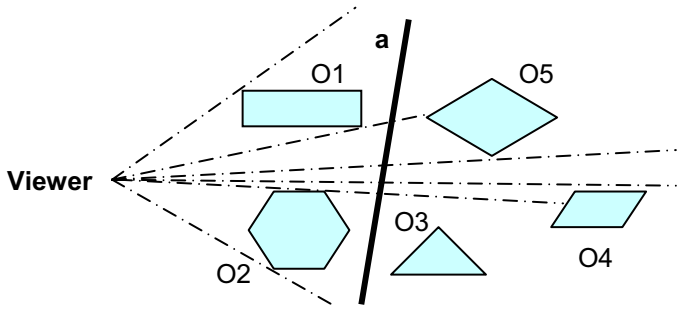

Figure 3.8. Property of object sorting of a hyperplane

in an efficient manner is exploited, but they also may be advantageously employed for the following applications:

- hidden surfaces removal,
- analytic visibility computing,
- ray tracing accelerating,
- boolean operations performing on polytopes,
- collision detection performing,
- dynamic scenes performing,
- shadows computing,
- connectivity information extraction,
- robot motion planning.

Notice that as long as the relations encoded by a tree remain valid, which for a rigid body is forever, one can exploit the benefits of having generated this tree structure every time the tree is used in subsequent operations. The return on investment leads to substantially faster algorithms for computing intersections and

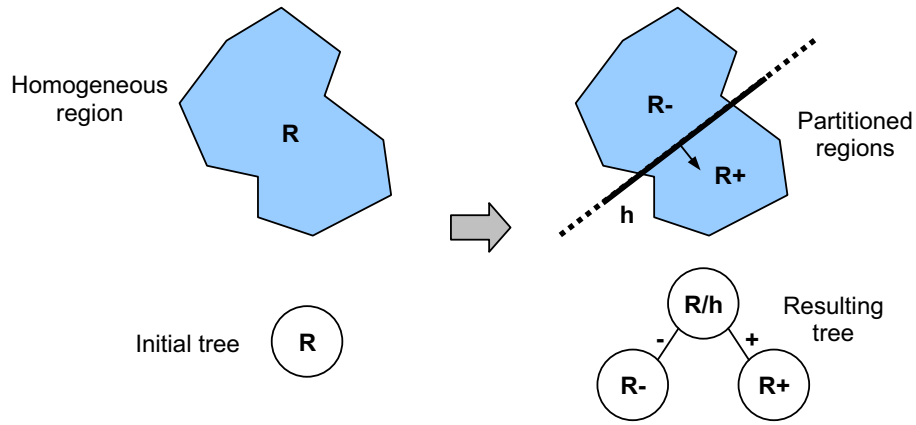

Figure 3.9. Elementary operation used to construct BSP Trees

visibility orderings. And for applications of image sequences processing in real-time, this savings is greater than hundreds of thousands of frames.

Moreover, affine and perspective transformations can be applied without having to modify the structure of the tree itself, but rather by modifying the linear equations representing each hyperplane (with a vector-matrix product as one does with points).

Building partitioning trees

BSP Trees exploit the properties of separating planes by using one very simple but powerful technique to represent any object or collection of objects: *recursive subdivision by hyperplanes*. A partitioning tree is the recording of this process of recursive subdivision in the form of a binary tree of hyperplanes [36]. The only operation necessary for constructing BSP Trees is the partitioning of a convex region by a single hyperplane into two child regions, both of which are also convex as a result (see Fig. 3.9).

Given a set of polygons in 3D space, which define the target objects, the goal is to build a BSP tree tree which contains all of the polygons. For now, the question

of how the resulting tree is going to be used is ignored. The algorithm to build a BSP tree is very simple:

1. select a partition plane,
2. partition the set of polygons with the plane,
3. recurse with each of the two new sets.

The choice of partition plane depends on how the tree will be used, and what sort of efficiency criteria is adopted for the construction. For the purpose of this work, it is appropriate to choose the partition plane from the input set of polygons (called an *auto-partition*). Other applications may benefit more from axis-aligned orthogonal partitions.

Partitioning a set of polygons with a plane is done by classifying each member of the set with respect to the plane. If a polygon lies entirely into one side or the other of the plane, then it is not modified, and is added to the partition set for the side to which it belongs. If a polygon spans the plane, it is split into two or more pieces and the resulting parts are added to the sets associated with either side as appropriate.

The decision to terminate tree construction is, again, a matter of the specific application. Some methods terminate when the number of polygons in a leaf node is below a maximum value. Other methods, like that used in this work, continue until every polygon is placed in an internal node. Another criteria is a maximum tree depth.

Here is an example of pseudo C++ code for the construction of a partitioning tree:

```
struct  BSP_tree {
```

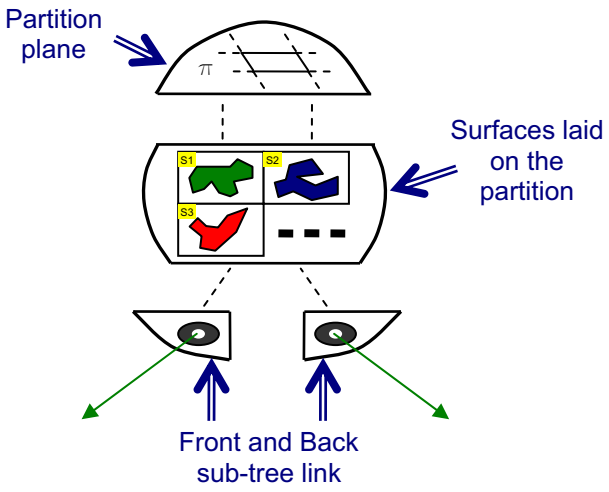

Figure 3.10. Structure of a node of the partitioning tree

```
    plane       partition;
    list        polygons;
    BSP_tree    *front,
                *back;
};
```

This structure definition will be used for all subsequent example code. It stores pointers to its children, the partitioning plane for the node, and a list of polygons coincident with the partition plane (see Fig. 3.10). For this example, there will always be at least one polygon in the coincident list: the polygon used to determine the partition plane (auto-partition). A constructor method for this structure should initialize the child pointers to NULL.

```
void    Build_BSP_Tree (BSP_tree *tree, list polygons) {
    polygon   *root = polygons.Get_From_List ();
    tree->partition = root->Get_Plane ();
    tree->polygons.Add_To_List (root);
```

```
list      front_list,
          back_list;
polygon   *poly;
while ((poly = polygons.Get_From_List ()) != 0) {
    int   result = tree->partition.Classify_Polygon (poly);
    switch (result) {
        case COINCIDENT:
            tree->polygons.Add_To_List (poly);
            break;
        case IN_BACK_OF:
            back_list.Add_To_List (poly);
            break;
        case IN_FRONT_OF:
            front_list.Add_To_List (poly);
            break;
        case SPANNING:
            polygon   *front_piece, *back_piece;
            Split_Polygon (poly, tree->partition,
                front_piece, back_piece);
            back_list.Add_To_List (back_piece);
            front_list.Add_To_List (front_piece);
            break;
    }
}
if ( ! front_list.Is_Empty_List ()) {
    tree->front = new BSP_tree;
    Build_BSP_Tree (tree->front, front_list);
```

```
        }
        if ( ! back_list.Is_Empty_List ()) {
            tree->back = new BSP_tree;
            Build_BSP_Tree (tree->back, back_list);
        }
    }
```

A detailed description on how to split a polygon with a plane may be found in Appendix A. This routine recursively constructs a BSP tree using the above definition. It takes the first polygon from the input list which is then used to partition the remainder of the set. The routine then calls itself recursively with each of the two partitions. This implementation assumes that all the input polygons are convex. Notice that an iterative formulation may be realized in lieu of the previous recursive algorithm, using an explicit stack.

Converting boundary representation to partitioning tree

Since humans do not see physical objects in terms of binary trees, it is important to know how such a tree be constructed from something which is more intuitive. The most common method is to convert a boundary representation, which corresponds more closely to how humans see the world, into a tree [32]. In order for a BSP tree to represent a solid object, each cell of the tree must be classified as being either entirely inside or outside of the object; thus, each leaf node corresponds to either an *in-cell* or an *out-cell*. The boundary of the set then lies between in-cells and out-cells; since the cells are bounded by the partitioning hyperplanes, it is necessary for all of the boundaries to lie in the partitioning hyperplanes.

Therefore, it can convert from a boundary representation to a tree simply by using all the face hyperplanes as partitioning hyperplanes (*auto-partition*), as shown

in Fig. 3.7 [33]. The face hyperplanes can be chosen in any order and the resulting tree will always generate a convex decomposition of the internal side and the external side. If the hyperplane normals to the boundary representation faces are consistently oriented to point to the external side, then all *back* leaves will be in-cells and all *front* leaves will be out-cells.

However, the choice of partition plane will strongly affect the results. For some applications, e.g. multi-dimensional searching, it is desirable to have a balanced tree, where each leaf contains roughly the same number of polygons. However, there is some cost in achieving this. If a polygon happens to span the partition plane, it will be split into two or more pieces. A poor choice of the partition plane can result in many such splits, and a marked increase in the number of polygons. Usually there will be some trade off between a well-balanced tree and a large number of splits.

Tree balancing is important when performing spatial classification of points, lines, and surfaces. This includes ray tracing and solid modelling. Tree balancing is important for these applications because the time complexity for classification is based on the depth of the tree. Unbalanced trees have deeper subtrees, and therefore have a worse worst case. For the application of this work, instead, the balancing of the treeis not so crucial, while it is more important to reduce the number of resulting polygons. In fact, for the hidden surface problem balancing does not significantly affect runtime. This is because the expected time complexity for tree traversal is linear with respect to the number of polygons in the tree, rather than the depth of the tree.

The problem is that polygons get split during the construction phase, which may give rise to a larger number of polygons. Larger numbers of polygons translate into larger storage requirements and longer tree traversal times. This is undesirable in all applications of BSP Trees, and thus some scheme for minimizing splitting will

improve tree performance. If should be considered that minimization of splitting requires pre-existing knowledge about all the polygons that will be inserted into the tree. This knowledge may not be available for interactive use such as solid modelling, but it is available for the application considered in this work. The easiest strategy to minimize splitting is to choose a partition plane, if it may be extracted from the list of the remaining polygons, that does not intersect with any other polygon. This strategy has a visibility on the recursive steps of only one step, and thus it does not assure that the obtained tree contains the minimal number of polygons. However, some other more sophisticated strategy may be applied to impact this problem.

Whenever balancing is also of concern for the application of interest, it will be necessary to trade off some balance for reduced splitting. So, if the hyperplanes are chosen from the polygon candidates, then one way to optimize these two factors is to randomly select a small number of candidates. These new candidates are tested against the full list for splitting and balancing efficiency. A linear combination of the two efficiencies is used to rank the candidates, and the best one is chosen.

Boundary representations and BSP Trees can be thought of as competing alternatives or as complementary representations expressing difference aspects of geometry, the former being topological, the latter expressing hierarchical set membership. B-reps are well suited for interactive specification of geometry, expressing topological deformations, and scan conversion. BSP Trees are well suited for intersection and visibility calculations.

The most often asked question on this argument is what is the size of a BSP tree representation of a polyhedron vs. the size of its boundary representation. This, of course, ignores the fact that expected cost, measured over the suite of operations for which the representation will be used, is the appropriate metric. Also, boundary representations must be supplemented with other devices, such as *octrees*, *bounding*

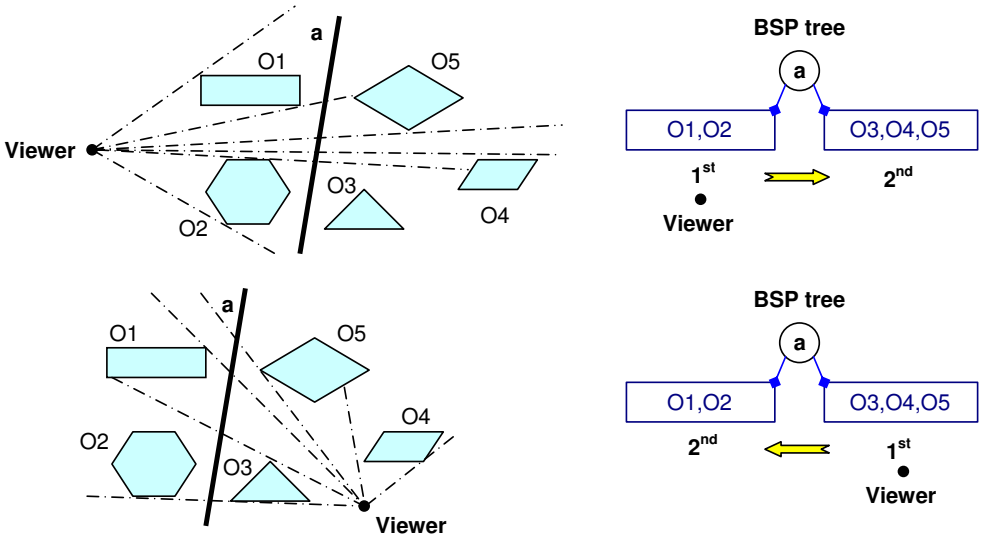

Figure 3.11. Visibility ordering using a single hyperplane. Top: left side has priority over right side; bottom: right side has priority over right side

volume hierarchies, and *z-buffers*, in order to achieve an efficient system; and then the cost of creating and maintaining these structures should be brought into the picture.

Visibility orderings

Visibility orderings are used in image synthesis for visible surface determination (hidden surface removal), shadow computations, ray tracing, beam tracing, and radiosity. For a given center of projection, such as the position of a viewer or a light source, they provide an ordering of geometric entities, such as objects or faces of objects, consistent with the order in which any ray originating at the center might intersect the entities. Loosely speaking, a visibility ordering assigns a priority to each object or face so that closer objects have priority over objects further away. Any ray emanating from the center or projection that intersects two objects or faces, will always intersect the surface with higher priority first. The simplest use of visibility orderings is with the *Painter's Algorithm* for solving the hidden surface problem [11]. Faces are drawn into a frame buffer in far-to-near order (low-to-high priority), so that the image of closer objects/polygons over writes those of more distant ones.

A visibility ordering can be generated using a single hyperplane; however, each geometric entity or "object" (polyhedron, polygon, line, point) must lie completely on one side of the hyperplane, i.e. no objects are allowed to cross the hyperplane. This requirement can always be induced by partitioning objects by the desired hyperplane into two "halves". The objects on the side containing the viewer are said to have visibility priority over objects on the opposite side; that is, any ray emanating from the viewer that intersects two objects on opposite sides of the hyperplane will always intersect the near-side object before it intersects the far-side object (see Fig. 3.11).

However a single hyperplane cannot order objects lying on the same side, and thus it cannot provide a total visibility ordering. Consequently, in order to exploit this idea, it must be extended somehow so that a visibility ordering for the entire set of objects can be generated. One way to do this would be to create a unique separating hyperplane for every pair of objects. However, for n objects this would require n^2 hyperplanes, which is too many.

The required number of separating hyperplanes can be reduced to as little as n by using the geometric version of recursive subdivision (*divide and conquer*). Whether the subdivision is performed using hyperplanes whose position and orientation is unrestricted, then the result is a BSP tree. The objects are first separated into two groups by some appropriately chosen hyperplane (as above). Then each of the two groups are independently partitioned into two sub-groups. The recursive subdivision continues in a similar fashion until each object, or piece of an object, is in a separate cell of the partitioning, as shown in Fig. 3.6. This process of partitioning space by hyperplanes is naturally represented as a binary tree.

BSP Trees may be efficiently used to generate a visibility ordering on the collection of objects [36]. For any given viewing position, it is first necessary to determine on which side of the root hyperplane the viewer lies. From this, all objects

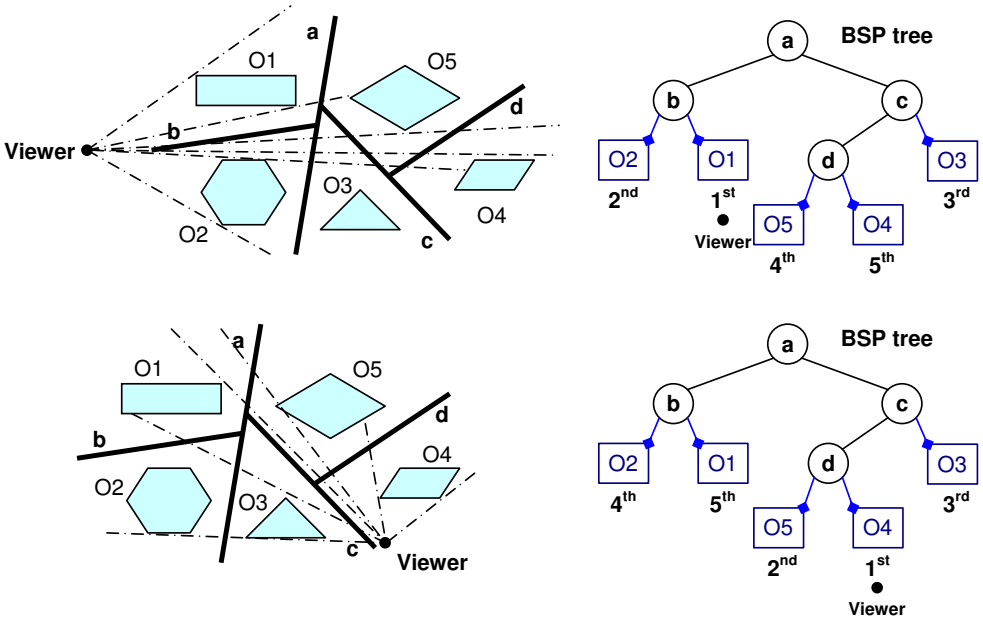

Figure 3.12. Visibility ordering on a collection of objects

in the near-side subtree have higher priority than all objects in the far-side subtree; and this determination can be made with only a constant amount of computation (in fact, only a dot product). At this point the near-side objects have to be ordered, followed by an ordering of the far-side objects. Since the structure bas been recursively defined, any subtree has the same computational form as that of the whole tree. Therefore, this technique can be simply applied for ordering subtrees recursively, going back or front first at each node, depending upon which side of the node hyperplane the viewer lies. This corresponds to a traversal of the entire tree, in near-to-far order, using only $O(n)$ operations, which is optimal (this analysis is correct only if no objects have been split; otherwise it is $> n$), as shown in Fig. 3.12.

The scheme described above can be adopted only for inter-object visibility, i.e. between individual objects. Moreover, only when the objects are both convex and separable by a hyperplane, the scheme is a complete method for determining visibility. To address the general unrestricted case, it is required to solve intra-object

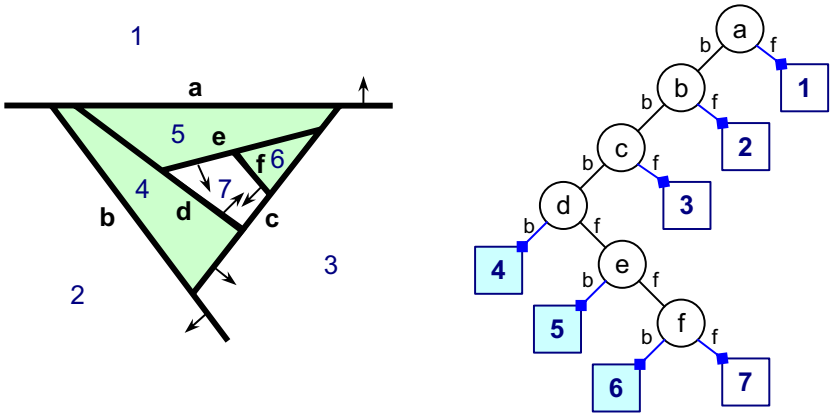

Figure 3.13. Visibility ordering on intra-objects. Left: spatial partitioning; right: binary tree (b=back, f=front)

visibility, i.e. correctly ordering the faces of a single object. BSP Trees can solve this problem as well. To accomplish this, it is required to change the focus from convex cells containing objects to the idea of hyperplanes containing faces and to consider again the analysis of visibility with respect to the hyperplane. If instead of ordering objects, it is wished to order faces, it can exploit the fact that not only can faces lie on each side of a hyperplane as objects do, but they can also lie on the hyperplane itself. This gives a 3-way ordering as: *in front of the face*, *coincident with the face*, and *in back of the face*.

Whether hyperplanes by which to partition space that always contain a face of an object are chosen, then a BSP tree can be built by applying this scheme recursively as above, until every face lies in some partitioning hyperplane contained in the tree. An example of intra-object BSP tree is shown in Fig. 3.13. To generate a visibility ordering of the faces in this intra-object tree, the method above may be used with one extension: faces lying on hyperplanes are included in the ordering, i.e. at each node, so as to generate the visibility ordering of `front-subtree` ⇒ `on-faces` ⇒ `back-subtree`.

The idea behind the Painter's Algorithm in the case of inter-object visibility

is to draw polygons far away from the viewer first, followed by drawing those that are close to the viewer. Hidden surfaces will be written over in the image as the surfaces obscuring them are drawn. One condition for a successful painter's algorithm is that there is a single plane which separates any two objects. This means that it might be necessary to split polygons in certain configurations. One reason that BSP Trees are so elegant for the painter's algorithm is that the splitting of difficult polygons is an automatic part of tree construction. Notice that only one of these two polygons needs to be split in order to solve the problem. To draw the contents of the tree, a back to front tree traversal has to be performed. One has to begin at the root node and classify the viewer with respect to its partition plane. The subtree is drawn at the far child from the eye, then the polygons in this node, then the near subtree. This procedure shall be repeated recursively for each subtree.

When building a BSP tree specifically for visibility ordering (sometimes called *hidden surface removal*), the partition planes are usually chosen from the input polygon set. However, any arbitrary plane can be used if there are no intersecting or concave polygons. Using the `BSP_tree` structure defined in the previous section, here is a simple pseudo C++ code of a back to front tree traversal:

```
void    Draw_BSP_Tree (BSP_tree *tree, point eye) {
   real   result = tree->partition.Classify_Point (eye);
   if (result > 0) {
      Draw_BSP_Tree (tree->back, eye);
      tree->polygons.Draw_Polygon_List ();
      Draw_BSP_Tree (tree->front, eye);
   }
   else if (result < 0) {
      Draw_BSP_Tree (tree->front, eye);
```

```
      tree->polygons.Draw_Polygon_List ();
      Draw_BSP_Tree (tree->back, eye);
   }
   else // result is 0 {
      // the eye point is on the partition plane...
      Draw_BSP_Tree (tree->front, eye);
      Draw_BSP_Tree (tree->back, eye);
   }
}
```

If the viewer is classified as being on the partition plane, the drawing order is unclear. This is not a problem if the `Draw_Polygon_List` routine is smart enough not to draw polygons that are not within the viewing frustum. The coincident polygon list does not need to be drawn in this case, because those polygons will not be visible to the user.

It is possible to substantially improve the quality of this example by including the viewing direction vector in the computation. You can determine that entire subtrees are behind the viewer by comparing the view vector to the partition plane normal vector. This test can also make a better decision about tree drawing when the eye point lies on the partition plane.

Using visibility orderings provides an alternative to z-buffer based algorithms. They obviate the need for computing and comparing z-values, which is very susceptible to numerical error because of the perspective projection. In addition, they eliminate the need for z-buffer memory itself, which can be substantial if used at a sub-pixel resolution of 4×4 to provide anti-aliasing. More importantly, visibility orderings permit unlimited use of (non-refractive) transparency with no additional computational effort, since the visibility ordering gives the correct order for compos-

ing faces using alpha blending.

Finally, noting that it is just as easy to traverse the BSP tree in front to back order as it is for back to front, this can be conveniently used in a *scan-line* method by using a write mask which will prevent pixels from being written more than once. This will represent significant speedups if a complex lighting model is evaluated for each pixel, because the painter's algorithm will blindly evaluate the same pixel many times. The trick to making a scan-line approach successful is to have an efficient method for masking pixels. One way to do this is to maintain a list of pixel spans which have not yet been written for each scan line. For each converted polygon scan, only pixels in the available spans are written, and the spans are updated accordingly. The scan-line spans can be represented as binary trees, which are just one dimensional BSP trees. This technique can be expanded to a 2D screen coverage algorithm using a 2D BSP tree to represent the masked regions. Any convex partitioning scheme, such as a quad-tree, can be used with similar effect.

CHAPTER 4

IMAGE PROCESSING

The image information, differently from the standard information provided by the sensors which are typically employed in automation, may be very rich and various, but it needs of complex and computationally expensive elaboration. The *image elaboration* and the *image interpretation* are two of the most important areas of *computer vision*, that is the application of computational system to the elaboration of visual information. The target of image elaboration is the manipulation of the visual information to obtain data formats and structures to be converted into synthetic numerical information (called *image features*), while the image interpretation consists in the extraction of a few number of specific image features from the scene, that are of some interest for the specific application. In this chapter an introduction to the most diffused techniques of image elaboration and interpretation will be presented, with a particular attention to those that will be used in the proposed visual tracking algorithm. Further, the so-called *windowing* technique, for reduction of the image region to elaborate, will be discussed.

4.1 Image elaboration

The image interpretation consists in the description of the main interesting charac-

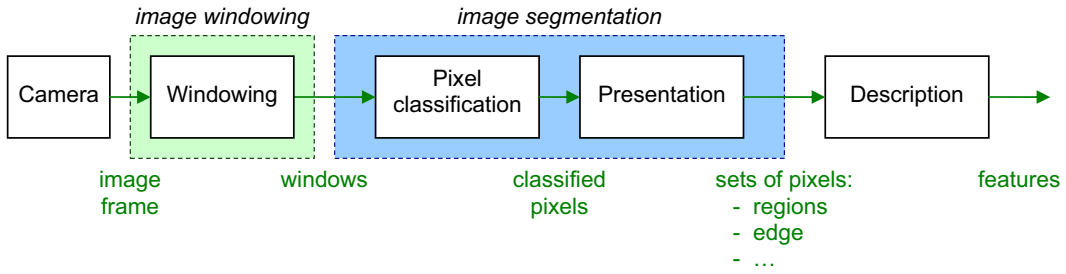

Figure 4.1. Image elaboration process

teristics of the observed structures and objects present in the scene. Typically, this description is made using numerical characteristics (features), which measure the visual attributes of concern for the task execution or for the evaluation of the task quality. Unfortunately, the complexity of the image information and the time constraints for the real-time applications do not allow the direct extraction of the image features from the image frame provided by the camera. In this condition the image elaboration becomes fundamental, which is aimed at transforming the image frame into suitable and efficient structures, for the extraction of the desired features.

The main operation realized during image elaboration is the *image segmentation*. It is the process of image subdivision into homogenous segments with respect to some characteristics. In the literature many kind of techniques have been presented to achieve a robust image segmentation. Unfortunately, not all those techniques may be applied in real-time applications, because they are computational very expensive. So, a trade-off between robustness and computational efficiency is often necessary.

In Fig. 4.1 the main elaboration steps involved in the image elaboration process are shown:

1. windowing,

2. image segmentation

- pixel classification
- presentation,

3. description.

The windowing step extracts a small number of windows from the image frame, centered around the desired features, to reduce the dimension of the total elaboration area (more details on this optional step will be presented in the next section). The image segmentation can be divided into two sub-steps: *pixel classification* and *presentation*. During the first sub-step the pixel of the image windows (or of the whole image, if the windowing step is absent) are classified into spatial subsets according to the low level characteristics of the pixels. During the second sub-step these subsets are transformed into data structure, which are more efficient and suitable to the successive elaboration. Finally, in the description step these subsets are described using scalar and/or vectorial values, which are associated to their characteristics.

The values of a pixel may be scalars or vectors, and they can represent the light luminosity, color, velocity, or a generic measurable property of the scene. The classification may also consider the proximity of the pixels, global statistics of the pixels, and temporal variation of the pixel values.

Once the relevant pixels have been found, they may be represented in many formats which allow determining some characteristics, such as shape and position of an object in an efficient manner. Two main representations are used for the image segments: *2D regions* and *boundary regions*. The first representation considers 2D regions of close pixels having similar characteristics, while the second representation considers only the edges of the image. The edges are discontinuities of the pixel characteristics, which often correspond to the boundary regions of the objects of the scene.

These representations are "equivalent" and can be converted one into the other, even though the two data structures are very different. The advantage of the boundary regions is the reduced requirement of memory, but the 2D regions turn out to be more flexible and efficient for a wide class of feature extraction algorithms. Moreover, the edge detection may begin after that the whole image frame is available in the elaboration memory, requiring random access to the memory, and four memory access on average to establish the location of the next pixel of the edge. On the other hand, the boundary regions are very efficient and suitable for the extraction of boundary features, e.g. contours, lines, and corners.

In the next section the most used kinds of segmentation, i.e. binary segmentation and edge detection, will be presented. In particular, the general concepts at the base of the edge detection will be described and the Canny edge detector will be presented.

4.1.1 Binary segmentation

The simplest segmentation technique is *binary segmentation*, that considers only two regions. Typically, the segmentation process consists in a comparison of the pixels with a *threshold* T. Suppose that P_{ij} is the value of the pixel with coordinates (i, j) and S_B and S_F are the sets of the background and foreground pixels, respectively. The binary segmentation process may be represented as follows:

$$P_{ij} = \begin{cases} S_B & \text{if} \quad I_{ij} < T \\ \\ S_F & \text{if} \quad I_{ij} \geq T \end{cases}.$$

This technique is widely employed for laboratory applications, where it is possible to control the illumination and the environment, e.g. using a black object against a white background, to increment the contrast and accurately isolate the image region corresponding to the object. The most important advantage of this technique is the

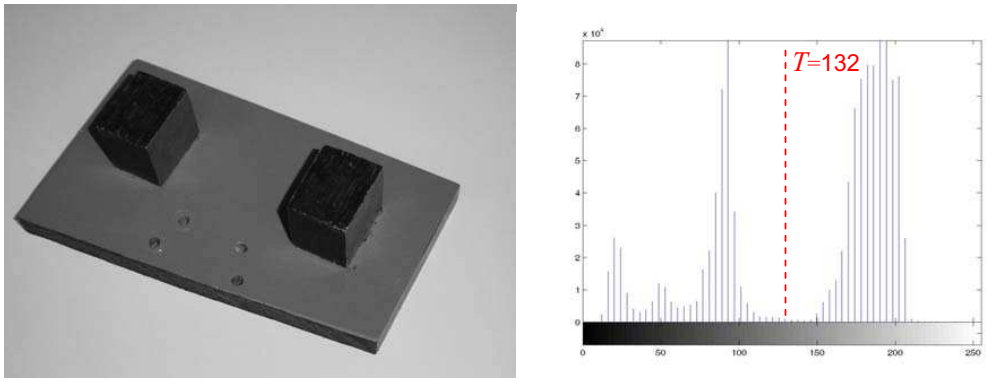

Figure 4.2. Binary segmentation. An example of gray level histogram of the image of an object. The threshold $T = 132$ has been automatic evaluated exploiting the presence of a concavity in the histogram

reduced computational complexity, that makes it suitable for real-time applications.

The crucial step of this approach is the choice of a suitable threshold. In the literature many approaches to the automatic threshold selection have been presented. Some adaptive techniques have also been proposed, but they still have problems with the determination of effective quality indexes of the binary segmentation. The most employed solution is that based on the *image histogram*, also called *histogram of the grey levels*, that provides the number of pixels for each possible pixel's values. In a controlled environments, e.g. in a laboratory, the histogram should present two peaks, corresponding to the object background and foreground, which are separated by a concavity. The threshold may be automatically chosen in the middle of this concavity, but this situation is purely ideal. In Fig. 4.2 an example of grey level histogram of a real image is shown, for which the threshold $T = 132$ has been evaluated by exploiting the presence of a concavity.

For scenes with 3D objects the operating conditions are more complicated, especially because of image noise. This is experienced in various manners, but in particular it modifies the grey levels of each pixel in a different way. Some other problems are caused by the illumination and by the light reflection of the object.

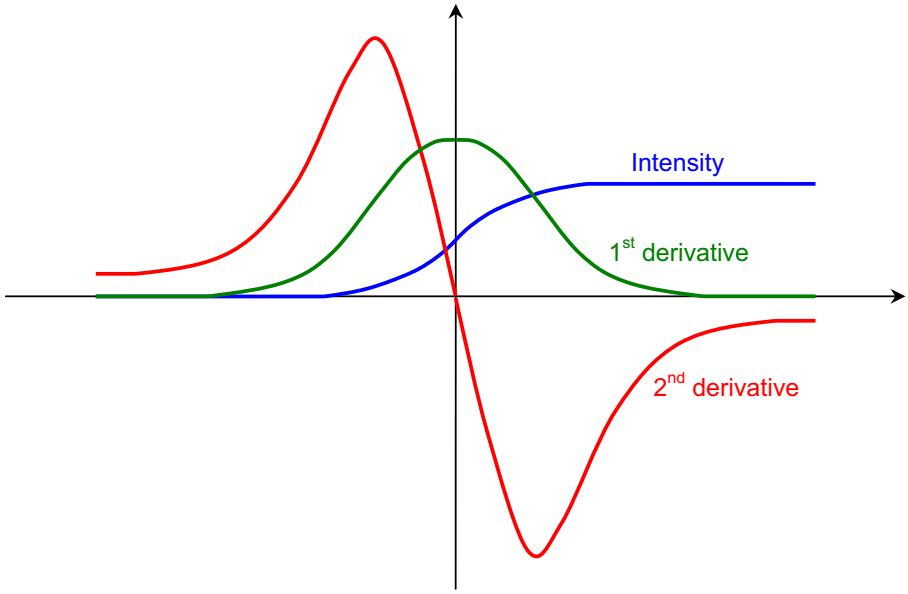

Figure 4.3. Image intensity shape and its first and second derivative of an image edge in one-dimensional case

Finally, different dimensions between the region of the object and the background may generate problems, e.g. the peak corresponding to the smallest region may be confused with the noisily base of the largest peak.

4.1.2 Edge detection

Edge detection is a problem of fundamental importance in image analysis. In typical images, edges characterize object boundaries and are therefore useful for segmentation, registration, and identification of objects in a scene. Further, detecting the edge of an image significantly reduces the amount of data and filters out useless information, while preserving the important structural properties of an image.

There are many ways to perform edge detection. However, the majority of the methods may be grouped into two categories, *gradient* and *Laplacian*. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. The Laplacian method searches for zero crossings in the second derivative of the image to find edges.

An edge has the one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location. Suppose that the intensity signal is that shown in Fig. 4.3, with an edge shown by the jump in intensity below. By taking the gradient of this signal (which, in one dimension, is just the first derivative) the signal marked as "1$^{\text{st}}$ derivative" is achieved. Clearly, the derivative shows a maximum located at the center of the edge in the original signal. This method of locating an edge is characteristic of the *gradient filter* family of edge detection filters and includes the *Sobel* and the *Roberts* method. A pixel location is declared an edge location if the value of the gradient exceeds some threshold. As mentioned before, edges will have higher pixel intensity values than those surrounding it. Therefore, once a threshold is set, the gradient value can be compared to the threshold value and an edge can be detected whenever the threshold is exceeded.

Furthermore, when the first derivative is at a maximum, the second derivative is zero. As a result, another alternative to finding the location of an edge is to locate the zeros in the second derivative. This method is known as the *Laplacian* (or *zero-crossing edge detectors*) and the second derivative of the signal is shown if Fig. 4.3 and it is marked as "2$^{\text{nd}}$ derivative". Notice that the zero crossings are independent of the steepness of the transition, while the gradient magnitude is directly related to the edge slope.

As it has been said, an edge is a jump in intensity of the image and the cross section of an edge has the shape of a ramp (an ideal edge is a discontinuity, i.e. a ramp with an infinite slope), and the first derivative of the image assumes a local maximum at an edge. For a continuous image $I(r, c)$, where r and c are the row and column coordinates respectively, the two directional derivatives $\partial_r I(r, c)$ and $\partial_c I(r, c)$ are considered. Of particular interest in edge detection are two functions that can be expressed in terms of these directional derivatives: the *gradient magnitude* and the

gradient orientation. The gradient magnitude is defined as

$$|\nabla I(r,c)| = \sqrt{(\partial_r I(r,c))^2 + (\partial_c I(r,c))^2},$$

and the gradient orientation is given by

$$\underline{/\nabla I(r,c)} = \arctan\left(\frac{\partial_r I(r,c)}{\partial_c I(r,c)}\right).$$

Local maxima of the gradient magnitude identify edges in $I(r,c)$. When the first derivative achieves a maximum, the second derivative is zero. For this reason, an alternative edge-detection strategy is to locate zeros of the second derivatives of $I(r,c)$. The differential operator used in these so-called zero-crossing edge detectors is the Laplacian

$$\nabla I(r,c) = \partial_{\{r,2\}} I(r,c) + \partial_{\{c,2\}} I(r,c).$$

In practice, finite difference approximations of first-order directional derivatives are used. These are represented by a pair of masks, say h_r and h_c. Formally these are linear-phase FIR filters. A convolution of the image with h_r and h_c gives two directional derivative images g_r and g_c respectively. The gradient image is traditionally calculated as $\nabla I = \sqrt{g_r^2 + g_c^2}$, or alternatively using $\nabla I = |g_r| + |g_c|$. A pixel location is declared an edge location if the value of this gradient (at point r, c) exceeds some threshold. The locations of all edge points constitute an edge map (it is similar to the binary segmentation described in the previous section). Hence, to recover the edges, the gradient image must be segmented using a global or local (i.e., adaptive) threshold operator. The choice of a threshold value determines the resulting segmentation and, therefore, the perceived quality of the edge detector. The selection of a threshold value is an important design decision that depends on a number of factors, such as image brightness, contrast, level of noise, and even edge direction. It is useful to consider the cumulative histogram of the gradient image in selecting an appropriate threshold value.

In the remainder of the section the Sobel, Prewitt, Roberts, and Laplacian edge detectors will be briefly presented, with particular reference to the masks used for the convolution operator, while in the last subsection a more sophisticated edge detector, due to Canny [4], will be presented.

4.1.2.1 Sobel operator

The Sobel operator performs a 2D spatial gradient measurement with built-in smoothing on the image. In Fig. 4.4 is shown an example of edge detection with the Sobel operator. Typically it is used to find the approximate absolute gradient magnitude at each point in an input gray-scale image. The Sobel edge detector uses a pair of (3×3) convolution masks, one estimating the gradient in the r-direction (rows) and the other estimating the gradient in the c-direction (columns). A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The actual Sobel masks are shown below:

$$h_r = \begin{array}{|c|c|c|} \hline -0.25 & -0.50 & -0.25 \\ \hline 0 & 0 & 0 \\ \hline +0.25 & +0.50 & +0.25 \\ \hline \end{array} \qquad h_c = \begin{array}{|c|c|c|} \hline -0.25 & 0 & +0.25 \\ \hline -0.50 & 0 & +0.50 \\ \hline -0.25 & 0 & +0.25 \\ \hline \end{array}$$

The mask is slid over an area of the input image, changes that pixel value and then shifts one pixel to the right and continues to the right until it reaches the end of a row. It then starts at the beginning of the next row. The example below shows how a (3×3) mask slides over the input image frame. Considering the $(s \times h)$ image frame

$$\begin{array}{|c|c|c|c|c|} \hline a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,h} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,h} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,h} \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline a_{s,1} & a_{s,2} & a_{s,3} & \cdots & a_{s,h} \\ \hline \end{array}$$

where $a_{i,j}$ represents the value of the pixel of coordinates (i, j), with $1 \leq i \leq s$ and

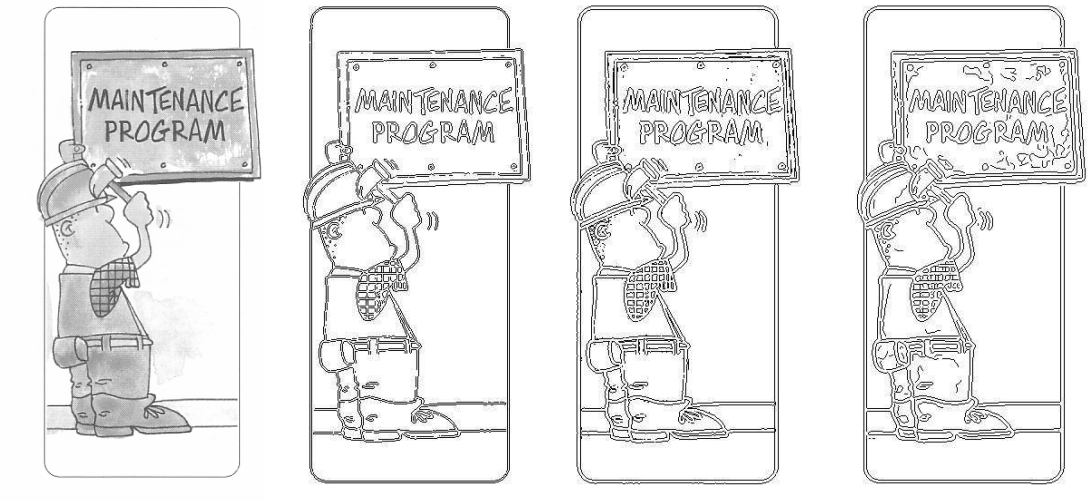

Figure 4.4. Examples of edge detection. From the left the original image and the results of the Sobel, Laplacian, and Canny edge detector are shown

$1 \leq j \leq h$, and the convolution (3×3) mask

$m_{1,1}$	$m_{1,2}$	$m_{1,3}$
$m_{2,1}$	$m_{2,2}$	$m_{2,3}$
$m_{3,1}$	$m_{3,2}$	$m_{3,3}$

then the frame result of their convolution assumes the following shape

$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$\cdots$	$b_{1,h}$
$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$\cdots$	$b_{2,h}$
$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$\cdots$	$b_{3,h}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$b_{s,1}$	$b_{s,2}$	$b_{s,3}$	$\cdots$	$b_{s,h}$

where the pixel of coordinates (i, j) is evaluated as follows

$$\begin{aligned} b_{i,j} = {} & a_{i-1,j-1}m_{1,1} + a_{i-1,j}m_{1,2} + a_{i-1,j+1}m_{1,3} \\ & + a_{i,j-1}m_{2,1} + a_{i,j}m_{2,2} + a_{i,j+1}m_{2,3} \\ & + a_{i+1,j-1}m_{3,1} + a_{i+1,j}m_{3,2} + a_{i+1,j+1}m_{3,3}, \end{aligned}$$

for $i = 2, \ldots, s-1$ and $j = 2, \ldots, h-1$. The center of the mask is placed over the pixel being manipulated in the image, and the i and j values are used to move the file pointer so that, for example, pixel $a_{2,2}$ can be multiplied by the corresponding mask

value $m_{2,2}$. It is important to notice that pixels in the first and last rows, as well as the first and last columns, cannot be manipulated by a (3×3) mask. This is because when placing the center of the mask over a pixel, say in the first row, the mask will be outside the image boundaries.

The g_c mask highlights the edges in the horizontal direction while the g_r mask highlights the edges in the vertical direction. After taking the magnitude of both, the resulting output detects edges in both directions.

4.1.2.2 Prewitt operator

The Prewitt operator performs a 2D spatial gradient measurement on the image in a similar manner to that of the Sobel operator. The Prewitt edge detector also uses a pair of (3×3) convolution masks, one estimating the gradient in the r-direction (rows) and the other estimating the gradient in the c-direction (columns). The actual Prewitt masks are shown below:

$$h_r = \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline +1 & +1 & +1 \\ \hline \end{array} \quad h_c = \begin{array}{|c|c|c|} \hline -1 & 0 & +1 \\ \hline -1 & 0 & +1 \\ \hline -1 & 0 & +1 \\ \hline \end{array}$$

4.1.2.3 Roberts operator

The Roberts operator performs a 2D cross difference on the image. Differently from the previous operators, the Roberts edge detector uses a pair of (2×2) convolution masks, one estimating the gradient in the r-direction (rows) and the other estimating the gradient in the c-direction (columns). The actual Roberts masks are shown below:

$$h_r = \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array} \quad h_c = \begin{array}{|c|c|} \hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array}$$

4.1.2.4 Laplacian

The commonly used (5×5) Laplacian is a convoluted mask to approximate the second

derivative, unlike the Sobel method which approximates the gradient. And instead of two (3×3) Sobel masks, one for the r and c direction, Laplace uses one (5×5) mask for the second derivative in both the r and c directions.

$$h_{rc} = \begin{array}{|c|c|c|c|c|} \hline -1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & 24 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 \\ \hline \end{array}$$

However, because these simple masks are approximating a second derivative measurement on the image, they are very sensitive to noise.

A zero-crossing edge operator was originally proposed in [29], where it is suggested that in order to effectively detect intensity changes (edges), the operator needs to have two characteristics. First, it must be a differential operator, taking either a first or second spatial derivative of the image. Second, it should be capable of being tuned to act at any desired scale so that large filters can be used to detect blurry shadow edges, and small ones can be used to detect sharply focused fine details in the image. This led to the so-called *Laplacian-of-Gaussian* edge operator. This is a compound operator that combines a smoothing operation, using a Gaussian-shaped, linear-phase FIR filter, with a differentiation operation, using a discrete Laplacian. The edges are identified by the location of zero crossings (recall that the second derivative changes sign in the vicinity of maxima of the first derivative). In fact, in order to mitigate the increase in pixel noise due to differentiation, the image may be filtered with a low-pass filter, which is accomplished by a Gaussian-shaped, linear-phase FIR filter.

Since convolution is associative and commutative, the two-step sequence can be reduced to one step by constructing a compound operator. In the following three

common (3×3) FIR filter approximations to the Laplacian operator are shown:

$$h_{rc}^{\alpha=0} = \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}$$

$$h_{rc}^{\alpha=0.5} = \begin{array}{|c|c|c|} \hline 0.333 & 0.333 & 0.333 \\ \hline 0.333 & -2.67 & 0.333 \\ \hline 0.333 & 0.333 & 0.333 \\ \hline \end{array}$$

$$h_{rc}^{\alpha=2} = \begin{array}{|c|c|c|} \hline 0.667 & -0.333 & 0.667 \\ \hline -0.333 & -1.33 & -0.333 \\ \hline 0.667 & -0.333 & 0.667 \\ \hline \end{array}$$

where the parameter α determines the effectiveness of the smoothing operation ($\alpha = 0$, when no smoothing is realized). The value ($\alpha = 2$) returns the minimum-variance discrete Laplacian subject to the conditions that pixel noise is uncorrelated and has uniform variance. Notice that the Laplacian is the lowest-order isotropic (i.e., orientation-independent) operator. In Fig. 4.4 an example of edge detection with the Laplacian operator is shown.

4.1.2.5 Canny edge detection

The Canny edge detection algorithm is known to many as the optimal edge detector. Canny's intentions were to enhance the many edge detectors already available at the time he started his work (see Fig. 4.4). He was very successful in achieving his goal and his ideas and methods can be found in his work [4], where he followed a list of criteria to improve current methods of edge detection. The first and most obvious is *low error rate*. It is important that edges occurring in images should not be missed and that there be no responses to non-edges. The second criterion is that the *edge points be well localized*. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum. A third criterion is to have *only one response to a single edge*. This was implemented because the first

two criteria were not substantial enough to completely eliminate the possibility of multiple responses to an edge.

Based on these criteria, the Canny edge detector first smoothes the image to eliminate the noise. It then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum (non-maximum suppression). The gradient array is now further reduced by *hysteresis* to track along the remaining pixels that have not been suppressed. Hysteresis uses two thresholds and if the magnitude is below the low threshold, it is set to zero (made a non-edge). If the magnitude is above the high threshold, it is made an edge. And if the magnitude is between the two thresholds, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above the high threshold.

In order to implement the Canny edge detector algorithm, a series of steps must be followed.

Step 1. The first step is to filter out any noise in the original image before trying to locate and detect any edges. And because the Gaussian filter can be computed using a simple mask, it is used exclusively in the Canny algorithm. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The larger the width of the Gaussian mask, the lower the detector sensitivity to noise. The localization error in the detected edges also increases slightly as the Gaussian width is increased. A mask approximating a Gaussian function with a

standard deviation equal to 1.4 is shown below.

$$\frac{1}{115}\begin{array}{|c|c|c|c|c|}\hline 2 & 4 & 5 & 4 & 2 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 5 & 12 & 15 & 12 & 5 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 2 & 4 & 5 & 4 & 2 \\ \hline \end{array}$$

Step 2. After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of (3×3) convolution masks, one estimating the gradient in the c-direction (columns) and the other estimating the gradient in the r-direction (rows). They are shown below:

$$h_r = \begin{array}{|c|c|c|}\hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \qquad h_c = \begin{array}{|c|c|c|}\hline -1 & 0 & +1 \\ \hline -2 & 0 & +2 \\ \hline -1 & 0 & +1 \\ \hline \end{array}$$

The magnitude, or *edge strength*, of the gradient is then approximated using the formula:

$$|\nabla I| = |g_r| + |g_c|.$$

Step 3. Finding the edge direction is trivial once the gradient in the r and c directions are known. However, it must generate an error whenever $g_c(r, c)$ is equal to zero. In such a case, the edge direction has to be equal to 90 degrees or 0 degrees, depending on the value of the gradient in the r-direction. If $g_r(r, c)$ has a zero value, the edge direction will be equal to 0 degrees. Otherwise the edge direction will be equal to 90 degrees. The formula for finding the edge direction when $g_c(r, c)$ is not equal to zero is just

$$\underline{/\nabla I} = \arctan\left(\frac{g_r}{g_c}\right)$$

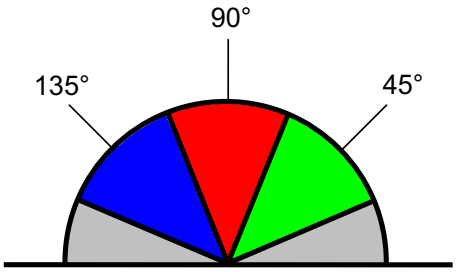

Figure 4.5. Canny edge detector. Step 4: possible edge direction

Step 4. Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So if the pixels of a (5×5) image are aligned as follows:

×	×	×	×	×
×	×	×	×	×
×	×	$\wp$	×	×
×	×	×	×	×
×	×	×	×	×

then, it can be seen by looking at pixel $\wp$, there are only four possible directions when describing the surrounding pixels: 0 degrees (in the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (in the vertical direction), or 135 degrees (along the negative diagonal). So now the edge orientation has to be resolved into one of these four directions depending on which direction it is closest to (e.g. if the orientation angle is found to be 3 degrees, it is approximate to zero degrees). To understand this a semicircle can be considered which is divide into 5 regions, like in Fig. 4.5. Therefore, any edge direction falling within the range 0 to 22.5 and 157.5 to 180 degrees is set to 0 degrees. Any edge direction falling in the range 22.5 to 67.5 degrees is set to 45 degrees. Any edge direction falling in the range 67.5 to 112.5 degrees is set to 90 degrees. And finally, any edge direction falling within the range 112.5 to 157.5 degrees is set to 135 degrees.

Step 5. After the edge directions are known, non-maximum suppression has to

be applied. Non-maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (set to 0) that is not considered to be an edge. This will give a thin line in the output image.

Step 6. Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold T is applied to an image, and an edge has an average strength equal to T, then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line. To avoid this, hysteresis uses two thresholds: one high and one low. Any pixel in the image that has a value greater than T_H is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T_L are also selected as edge pixels. It is like when following an edge, a gradient of T_L is needed to start and then it comes on until a gradient below T_H is hit.

4.2 Image interpretation

The image interpretation consists in the determination and extraction of the main image features, which are of some interest for the specific application. During the image processing, as explained in the previous sections, the image frame has to be converted into more efficient structures suitable for the specific image features to extract.

Actually, the image features of major interest for visual servoing applications are *centroids*, *corners*, and *contours*. The centroid is the simplest and most computational efficient image feature actually used. It is particulary indicated for planar tasks where both the environment and target objects are highly structured. The corner is a very robust and flexible image feature, that assures a good trade-off between

computational efficiency, wide usability, and robustness easily for the extraction in non-structured environments. Moreover, its use is adaptable to applications based on geometric object models. Finally, the contour is the most sophisticated image feature actually employed for visual servoing applications. It requires a significant computational power for the feature extraction process itself and often for the related algorithm that makes use of this feature.

In the next sections all these image features will be presented, with particulary attention to the corners, which will be used as main image features by the proposed visual tracking algorithm. The general definition of an image feature may be given by the following expression:

$$f = \iint_{\text{Image}} \Im\left(u, v, I(u,v)\right) du dv \tag{4.1}$$

where $I(u,v)$ is the *light intensity*[10] of the pixel of coordinates (u,v). The function $\Im(.,.,.)$ may express a linear a non-linear relation, in function of the specific image feature.

4.2.1 Centroids

The centroids are a class of image features very popular for planar applications in highly structured environments. They are easy and fast to evaluate also with non-dedicated hardware. Typically, they are used to construct numerical quantities, known as *invariants*, which allow reconstructing the position and orientation of the observed object.

---

[10] *Light intensity* is an informal terminology that may refer to two different attributes: a) the *irradiance* or *image luminosity* is referred to the light energy flow that affects on the image plane (it depends on the quantity of the light); b) the *radiance* or *scene luminosity* is referred to the light flow emitted from a surfaces (it depends on the light source and on the reflection capacity of the object). In this work the first definition is always considered.

The image processing required for the evaluation of the centroid is a simple binarization, that is used to highlight the target object with respect to the background of the scene. To reach this purpose it is necessary that the object and the background are easy distinguishable, e.g. a white object against a black background or viceversa. For a digitized image, the centroid of order $(p+q)$ is evaluated as follows:

$$m_{pq} = \sum_r \sum_c r^p c^q I(r,c) \tag{4.2}$$

where (r,c) are the coordinates of the rows and columns of the pixel into the digitized image frame.

The physical interpretation of the centroids is very clear if the image function I as a mass distribution of a planar object is considered. Then m_{00} represents the total mass of the object region, while the coordinates of the center of mass are

$$\begin{bmatrix} r_c \\ \\ c_c \end{bmatrix} = \begin{bmatrix} \dfrac{m_{10}}{m_{00}} \\ \\ \dfrac{m_{01}}{m_{00}} \end{bmatrix}. \tag{4.3}$$

The so-called *principal centroids* (or *central centroids*) are defined around the center of mass as follows:

$$\mu_{pq} = \sum_r \sum_c (r - r_c)^p (c - c_c)^q I(r,c), \tag{4.4}$$

and they are invariant with respect to the translation. They may be evaluated from the centroids m_{pq} as follows:

$$\begin{aligned} \mu_{20} &= m_{20} - \frac{m_{10}^2}{m_{00}} \\ \mu_{02} &= m_{02} - \frac{m_{01}^2}{m_{00}} \\ \mu_{11} &= m_{11} - \frac{m_{10} m_{01}}{m_{00}}. \end{aligned}$$

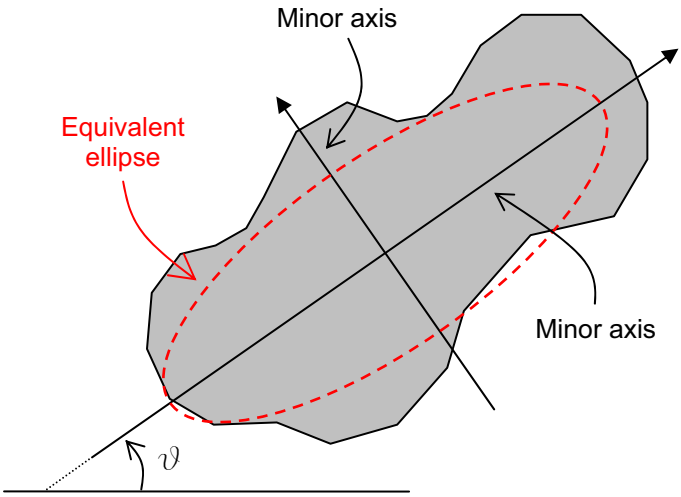

Figure 4.6. Centroids. Equivalent ellipse of an image region

A very common measure of the shape of the object region is the so-called *circularity*, that is defined by the following expression:

$$\rho = \frac{4\pi m_{00}}{p^2}$$

where ρ is the perimeter of the region. The value of the circularity is less than 1, while it is equal to $\pi/4$ for a square region.

The principal centroids of second order μ_{20}, μ_{02}, and μ_{11} may be considered as the inertial moments about the center of mass:

$$\boldsymbol{I} = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \\ \mu_{11} & \mu_{02} \end{bmatrix}.$$

The eigenvalues of this matrix are the principal moments of the region, while the corresponding eigenvectors are the principal axes of the region (the directions of the minimum and maximum inertial moment). From the eigenvector corresponding to the maximum eigenvalue it is possible to evaluate the orientation of the principal axis by the inversion of the following relation:

$$\tan 2\vartheta = \frac{2\mu_{11}}{\mu_{20} - \mu_{02}}$$

Statistics	
Dimension	1280 × 960 pixel
Area	467,518 pixel
Centroid	[596.35 520.62]
Bounding box	top-left-corner: [24 141] width: [1114 758]
Major axis length	1079.6 pixel
Minor axis length	576.05 pixel
Circularity	0.8458
Orientation	-16.48 deg
Equivalent diameter	771.53 pixel

Table 4.1. Centroids. Principal statistics of the digitized image, with the threshold $T = 132$, of the image of Fig. 4.2

where ϑ is the angle indicated in Fig. 4.6. To evaluate the orientation of an asymmetric object moving on a plane, its axis of asymmetry may be employed. This axis may be evaluated using the so-called *equivalent ellipse* of the region (see Fig. 4.6). This ellipse has an area equal to the area of the corresponding image region, and its minor and major axes correspond to the principal axes of the region. Table 4.1 collects the principal statistics of the corresponding digitized image.

4.2.2 Corners

The *corner* is one of the most used image feature in visual sensing application. It represents a good trade-off between computational complexity and quality and robustness of a synthetic image information. Corners have been used in very different manner for various types of applications, and in particular they are well indicated for those based on manmade models. Typically they are used to match the corners of a known model to the corners extracted from the image to determinate the presence or the pose of target objects in the scene.

The image processing required for the evaluation of the corners depends on

kind of algorithm employed. For example, the algorithms based on the search for significant turnings at the boundary require, given a digital image, the segmentation of the sharp and the extraction of its boundary as *chain code*[11], while other techniques directly process the gray level frame without any further elaboration. Corner detection algorithms should satisfy a number of important criteria:

- all the true corners should be detected,
- no false corners should be detected,
- corner points should be well localized,
- corner detector should be robust with respect to noise,
- corner detector should be efficient.

The Plessey corner detector. Harris and Stephens [12] proposed what has become to be known as the Plessey corner detector. The algorithm is based on the following matrix:

$$\boldsymbol{M} = \begin{bmatrix} \left(\frac{\partial I}{\partial u}\right)^2 & \left(\frac{\partial I}{\partial u}\right)\left(\frac{\partial I}{\partial v}\right) \\ \left(\frac{\partial I}{\partial u}\right)\left(\frac{\partial I}{\partial v}\right) & \left(\frac{\partial I}{\partial v}\right)^2 \end{bmatrix}$$

where $I(u, v)$ is the grey level intensity. If at a certain point the two eigenvalues of the matrix $\boldsymbol{M}$ are large, then a small motion in any direction will cause an important change of grey level. This indicates that the point is a corner. The corner response function is given by:

$$\rho = \det(\boldsymbol{M}) - k\left(\mathrm{trace}(\boldsymbol{M})\right)^2,$$

[11]The chain code is a particular representation of the edge of an observed object. It describes the edge with a "chain" of small integers which define the position of the next pixel of the edge indicating the corresponding discrete direction.

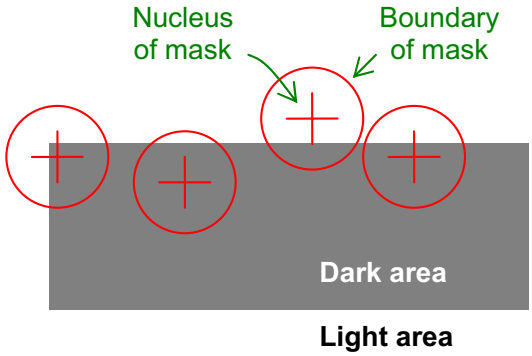

Figure 4.7. SUSAN corner detector. Some circular masks at different places on a simple image

where k is a parameter set to 0.04 (as per Harris suggestion). Corners are defined as local maxima of the corner-ness function. Sub-pixel precision is achieved through a quadratic approximation of the neighborhood of the local maxima. To avoid corners due to image noise, it can be useful to smooth the images with a Gaussian filter. This should however not be done on the input images, but on images containing the squared image derivatives (i.e. $\left(\frac{\partial I}{\partial u}\right)^2$, $\left(\frac{\partial I}{\partial v}\right)^2$, $\left(\frac{\partial I}{\partial u}\right)\left(\frac{\partial I}{\partial v}\right)$). In practice, often far too many corners are extracted. In this case it is useful to first restrict the numbers of corners before trying to match. One possibility consists of selecting only the corners with a value ρ above a certain threshold. This threshold can be tuned to yield the desired number of features. Since for some scenes most of the strongest corners are located in the same area, this scheme can be further refined to ensure that in every part of the image a sufficient number of corners are found.

The SUSAN corner detector. *SUSAN* (Smallest Univalue Segment Assimilating Nucleus) is based on an entirely different approach to low-level image processing compared to all preexisting algorithms [38]. It provides corner detection as well as edge detection and is more resistant to image noise although no noise reduction (filtering) is needed.

The concept of each image point associated with a local area of similar bright-

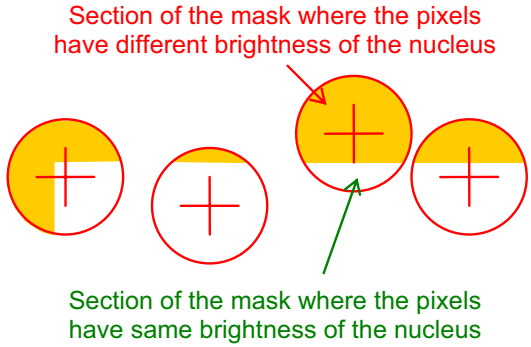

Figure 4.8. SUSAN corner detector. Some circular masks with similarity coloring; USANs are shown as the white parts of the masks

ness is the basis for the SUSAN principle. If the brightness of each pixel within a mask is compared to the brightness of that mask's nucleus then an area of the mask can be defined which has the same (or similar) brightness as such of the nucleus. This area of the mask shall be known as the "USAN", an acronym standing for "Univalue Segment Assimilating Nucleus".

In Fig. 4.8 each mask from Fig. 4.7 is depicted with its USAN shown in white. Computing USAN for every pixel in the digital image provides a way to determine the edges inside it. The value of USAN gets smaller on both sides of an edge and becomes even smaller on each side of a corner. Hence one can look for the Smallest USAN (or SUSAN for short). The local minima of the USAN map represents corners in the image. The reason that this method stays resistant to noise is the lack of computing spatial derivatives of the image intensity.

The Curvature Scale Space corner detector. The curvature scale space (CSS) technique is suitable for recovering invariant geometric features (curvature zero-crossing points and/or extreme) of a planar curve at multiple scales. The CSS corner detector works as follows [30]:

1. Extract the edge contours from the input image using any good edge detector

such as Canny.

2. Fill small gaps in edge contours. When the gap forms a T-junction, mark it as a T-corner.

3. Compute curvature on the edge contours at a high scale.

4. The corner points are defined as the maxima of absolute curvature that are above a threshold value.

5. Track the corners through multiple lower scales to improve localization.

6. Compare T-corners to the corners found using the CSS procedure and remove very close corners.

Experimental results show this algorithm spends most of its time (80%) detecting the edges in the image, then faster edge detectors may be used. The local maxima of absolute curvature are the possible candidates for corner points. A local maximum is either a corner, the top value of a rounded corner or a peak due to noise. The latter two should not be detected as corners. The curvature of a real corner point has a higher value than that of a rounded corner or noise. The corner points are also compared to the two neighboring local minima. The curvature of a corner should be twice that of one of the neighboring local minima. This is because when the shape of the contour is very round, contour curvature can be above the threshold.

It is interesting to see that in this method two types of corners are detected: T shaped and corners on the curve. Then one has to avoid assigning two corners to pixels which are close together. Figure 4.9 illustrates when this type of problem arises. This explains the reason for the last step in the CSS algorithm above.

Although the notion of corner seems to be intuitively clear, no generally accepted mathematical definition exists, at least for digital curves. In a sense, different

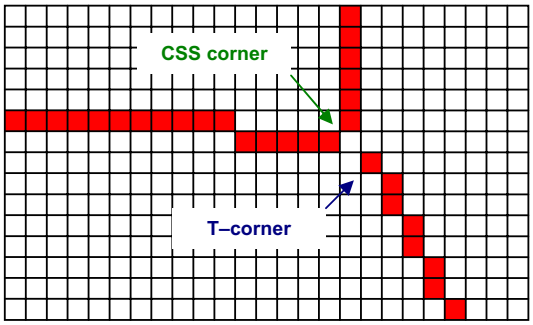

Figure 4.9. The CSS corner detector. Case where one corner is marked twice

approaches give different —but conceptually related— computational definitions to a visual phenomenon. Hence the detection of high curvature points in planar curves cannot be unique. In this section a brief summary of four alternative CSS corner detection algorithms is proposed. Each algorithm inputs a *chain-coded* curve that is converted into a connected sequence of grid points

$$\boldsymbol{p}_i = \begin{bmatrix} r_i \\ \\ c_i \end{bmatrix}$$

with $i = 1, 2, \ldots, N$. A measure of corner strength (*cornerity*) is assigned to each point, then corner points are selected based on this measure. For each approach, these two main steps are summarized and the parameters of the algorithm and their default ("best" with respect to the specific literature) values are listed.

When processing a point $\boldsymbol{p}_i$, the algorithms consider a number of subsequent and previous points in the sequence, as candidates for the arms of a potential corner in $\boldsymbol{p}_i$. For a positive integer k, the forward and backward k-vectors at point $\boldsymbol{p}_i$ are defined as

$$\boldsymbol{a}_{ik} = \begin{bmatrix} r_i - r_{i+k} \\ \\ c_i - c_{i+k} \end{bmatrix} = \begin{bmatrix} R_i^+ \\ \\ C_i^+ \end{bmatrix} \tag{4.5}$$

$$\boldsymbol{b}_{ik} = \begin{bmatrix} r_i - r_{i-k} \\ \\ c_i - c_{i-k} \end{bmatrix} = \begin{bmatrix} R_i^- \\ \\ C_i^- \end{bmatrix} \tag{4.6}$$

where R_i^+, C_i^+, and R_i^-, C_i^- are the components of $\boldsymbol{a}_{ik}$ and $\boldsymbol{b}_{ik}$, respectively.

Algorithm#1

Corner strength. The k-cosine of the angle between the k-vectors is used, which is defined as

$$\gamma_{ik} = \frac{(\boldsymbol{a}_{ik} \cdot \boldsymbol{b}_{ik})}{|\boldsymbol{a}_{ik}||\boldsymbol{b}_{ik}|} \tag{4.7}$$

Selection procedure. Starting from $m = \kappa N$, k is decreased until γ_{ik} stops to increase: $\gamma_{im} < \gamma_{i,m-1} < \cdots < \gamma_{in} \geq \gamma_{i,n-1}$; $k = n$ is then selected as the best value for the point i. A corner is indicated in i if $\gamma_{in} > \gamma_{jp}$ for all j such that $|i - j| \leq n/2$, where p is the best value of k for the point j.

Parameter. The single parameter κ specifies the maximum considered value of k as a fraction of the total number of curve points N. This limits the length of an arm at κN. The default value is $\kappa = 0.05$.

Algorithm#2

Corner strength. The averaged k-cosine of the angle between the k-vectors is used, which is defined as

$$\overline{\gamma}_{ik} = \begin{cases} \frac{2}{k+2} \sum_{f=k/2}^{k} & \text{if } k \text{ is even} \\ \\ \frac{2}{k+3} \sum_{f=(k-1)/2}^{k} & \text{if } k \text{ is odd} \end{cases}$$

where γ_{if} are given by (4.7).

Selection procedure. Same as in Algorithm #1, but for $\overline{\gamma}_{ik}$.

Parameter. Same as in Algorithm #1, with the same default $k = 0.05$.

Algorithm#3

Corner strength. At a point i, the angle between the r-axis and the backward k-vector defined in (4.6) is given as

$$\theta_{ik} = \begin{cases} \tan^{-1}\left(\frac{C^-_{ik}}{R^-_{ik}}\right) & \text{if } |R^-_{ik}| > |C^-_{ik}| \\ \coth^{-1}\left(\frac{R^-_{ik}}{C^-_{ik}}\right) & \text{otherwise.} \end{cases}$$

The incremental curvature is then defined as

$$\delta_{ik} = \theta_{i+1,k} - \theta_{i-1,k}. \tag{4.8}$$

Finally, the k-strength in i is computed as

$$S_{ik} = \ln(t_1)\ln(t_2)\sum_{j=i}^{i+k}\delta_{jk}, \tag{4.9}$$

where $t_1 = \max\{t : \delta_{i-\nu,k} \in (-\Delta, \Delta), \forall 1 \leq \nu \leq t\}$ and $t_2 = \max\{t : \delta_{i+k+\nu,k} \in (-\Delta, \Delta), \forall 1 \leq \nu \leq t\}$ account for the effect of the forward and backward arms as the maximum spacings (numbers of steps from i) that still keep the incremental curvature δ_{ik} within the limit $\pm\Delta$. The latter is set as

$$\Delta = \arctan\left(\frac{1}{k-1}\right). \tag{4.10}$$

Selection procedure. A point i is selected as a corner if S_{ik} exceeds a given threshold S and individual corners are separated by a spacing of at least $k + 1$ steps.

Parameter. The two parameters are the spacing k and the corner strength threshold S. The default values are $k = 5$ and $S = 1500$.

Algorithm#4

Corner strength. Similar to Algorithm #3, with the following modifications. The arm cutoff parameter τ is introduced to specify the upper limit for t_1 and t_2 as a fraction of N: $t_1 = \max\{t : \delta_{i-\nu,k} \in (-\Delta, \Delta), \forall 1 \leq \nu \leq t \text{and} t < \tau N\}$ and $t_2 = \max\{t : \delta_{i+k+\nu,k} \in (-\Delta, \Delta), \forall 1 \leq \nu \leq t \text{and} t < \tau N\}$, where δ_{ik} and Δ are given by (4.8) and (4.10), respectively. The corner strength is obtained by averaging (4.9) between two values k_1 and k_2:

$$S_i = \frac{1}{k_2 - k_1 + 1} \sum_{k=k_1}^{k_2} S_{ik}.$$

Selection procedure. Same as in Algorithm #3.

Parameter. The four parameters are the averaging limits k_1 and k_2, the arm cutoff parameter τ, and the corner strength threshold S. The default values are $k_1 = 4$, $k_2 = 7$, $\tau = 0.05$, and $S = 1500$.

4.2.3 Contours

Lines and curves are important features in the field of computer vision because they define the contour or shape of objects and hence enable computer recognition. The area of *shape representation* is concerned with finding ways of describing shape that are sufficiently general to be useful for a range of objects, whilst at the same time allowing their computation from image data, and facilitating comparisons of similar shapes. While line and curve fitting can be directly done using templates, these methods usually run into problems because of the requirement of a large number of data sets.

The algorithms that are used for detecting binary points lying on the same line and simple curves are the *Hough transform*, *robust fitting method*, *Euclidean fitting*

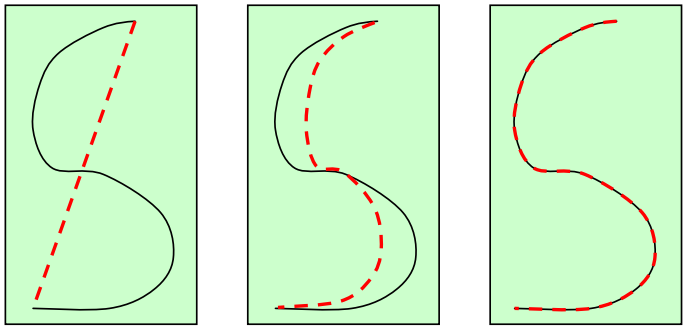

Figure 4.10. Snake. Left: initial snake position (dotted) interactively defined near the true contour; center and right: interaction steps of snake energy minimization (the snake is pulled toward the true contour)

and the *algebraic distance fitting*. The algorithm usually employed for deformable contours (which do not have a fixed shape) is the energy functional also called the *Greedy algorithm* or the *Snake*[12]. For reasons of brevity, the Greedy algorithm (also called *active contour model*) is the unique technique that will be presented in this section [21], [44].

The snake is defined as an energy minimizing spline: the snake energy depends on its shape and location within the image. Local minima of this energy then correspond to desired image properties. Snakes may be understood as a special case of a more general technique of matching a deformable model to an image by means of energy minimization. However, the snakes do not solve the entire problem of finding contours in images, but rather they depend on other mechanisms like interaction with a user, interaction with some higher level image understanding process, or information from image data adjacent in time or space. This interaction must specify an approximate shape and starting position for the snake somewhere near the desired contour. A-priori information is then used to push the snake toward an appropriate solution (see Fig. 4.10).

[12]Snake is a set of points where the energy is defined.

The energy functional which is minimized is a weighted combination of internal and external forces. The internal forces emanate from the shape of the snake, while the external forces come from the image and/or from higher level image understanding processes. The snake is parametrically defined as

$$\nu(s) = \begin{bmatrix} r(s) \\ \\ c(s) \end{bmatrix}$$

where $r(s)$, $c(s)$ are coordinates along the contour and s is from [0,1]. The energy functional to be minimized is

$$E^*_{Snake} = \int_0^1 E_{Snake}(\nu(s))ds = \int_0^1 \left[E_{int}(\nu(s)) + E_{image}(\nu(s)) + E_{con}(\nu(s))\right] ds. \quad (4.11)$$

where E_{int} is the internal spline energy caused by stretching and bending, E_{image} is the measure of the attraction of image features such as contours, and E_{con} is the measure of external constraints either from higher level shape information or user applied energy.

The internal spline energy can be written as

$$E_{int} = \alpha(s) \left|\frac{d\nu}{ds}\right|^2 + \beta(s) \left|\frac{d^2\nu}{ds^2}\right|, \quad (4.12)$$

where $\alpha(s)$ and $\beta(s)$ specify the *elasticity* and *stiffness* of the snake. The first-order term makes the snake act like a membrane; the constant $\alpha(s)$ controls the tension along the spine (stretching a balloon or elastic band). The second order term makes the snake act like a thin plate; the constant $\beta(s)$ controls the rigidity of the spine (bending a thin plate or wire). If $\beta(s) = 0$ then the function is discontinuous in its tangent, i.e. it may develop a corner at that point. If $\alpha(s) = \beta(s) = 0$, then this also allows a break in the contour, i.e. a positional discontinuity.

The second term of the energy integral is derived from the image data over which the snake lies. As an example, a weighted combination of three different func-

tionals is presented which attracts the snake to lines, edges, and terminations

$$E_{image} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term}, \tag{4.13}$$

where the weights w_{line}, w_{edge}, and w_{term} control the corresponding influence of each energy term in the functional and varies along the path of the curve.

The line-based functional may be very simple

$$E_{line} = I(r, c),$$

where $I(r, c)$ denotes image gray levels at image location (r, c). As a consequence, if w_{line} is largely positive, then the spline is attracted to light lines (or areas); on the other hand, if it is largely negative, then it is attracted to dark lines (or areas).

The edge-based functional attracts the snake to contours with large image gradients

$$E_{edge} = -|\nabla I(r, c)|^2,$$

where the gradient represents the intensity gradient along the curve computed at each snake point. Clearly E_{edge} becomes very small (negative) whenever the norm of the spatial gradient is large (which it would be if the contour are an edge).

Finally, the termination functional allows terminations (i.e. free ends of lines) or corners to attract the snake.

The constraint energy E_{con} comes from external constraints imposed either by a user, e.g. in the form of a spring attached to the snake in a specific position, or some other higher level process which may force the snake toward or away from particular features.

The Greedy algorithm is based on the minimization of the energy functional by varying the values of w_{line}, w_{edge}, and w_{term} which in turn vary the values of the

terms in the energy functional. The algorithm makes a series of choices at the local level so that an optimum solution can be found at a global level. The core part of the detection of a deformable contour can be explained in two steps. First, at each iteration each point of the contour is moved in a small neighborhood and a local minimum is found; the contour point is then moved to this minimum position to achieve energy functional minimization. Secondly, the corners or edge points on the contour are found out and a variance in the w_{edge} coefficients of the E_{edge} term is made correspondingly. These steps in detail are as follows.

1. **Greedy minimization**: The neighborhood chosen for a particular contour point is small, typically (3×3) or a (5×5) region in order to keep the computational intensity small. The complexity varies linearly with the increase in size of the neighborhood. The local minimization is then performed by keeping the energy functional minimum at each location.

2. **Corner elimination**: The second step involves the detection of the curvature maximum along the contour. The aim of the stiffness term is to avoid oscillations of the deformable contour. Thus, if the gradient is very high, then $\beta(s_k)$ is made zero so as to encourage equally spaced points on the contour.

The iterations stop when a predefined fraction of all the contour points reach there local minimum. However the Greedy algorithm does not guarantee that the local minimum will be same as the global minimum.

The Greedy algorithm takes an initial snake and iteratively refines the location of each of its points by looking at a "neighborhood" of pixels surrounding each pixel and selecting the location in that neighborhood where error is minimized. Larger neighborhoods should naturally be expected to result in fewer iterations before convergence and less likelihood to fall into small local minima, at the expense of becoming

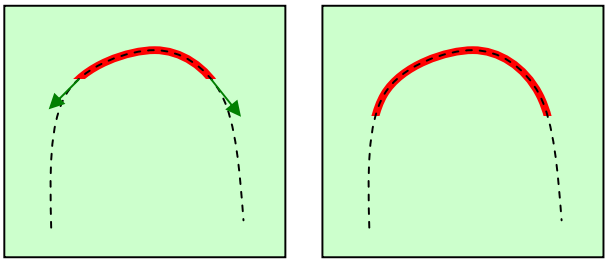

Figure 4.11. Snake growing. Left: lengthening in tangent direction; right: energy minimization after a growing step

linearly more time-consuming with neighborhood size.

Since this algorithm does not calculate actual gradients over the error surface but only takes the Greedy approach of minimizing over a relatively small neighborhood, it is easily prone to falling into local minima which may be caused by noise or small, non-salient image features. One work-around for this drawback is to blur out noise and small features, allowing the snake to more easily detect more substantial features, then refine its estimation as the blur is reduced. This implementation includes a multi-scale blurring approach, in which the user specifies how many times the initial image be blurred; the algorithm is run on the most blurred copy, then repeated on each "blurring scale" until it is run on the original image with a (hopefully) reasonable initial estimate.

Originally, a resolution minimization method was proposed; partial derivatives in s and t were estimated by the finite differences method. Later, a dynamic programming approach was proposed which allows "hard" constraints to be added to the snake. Further, a requirement that the internal snake energy must be a continuous function may thus be eliminated and some snake configurations may be prohibited (that is, having infinite energy) allowing more a-priori knowledge to be incorporated.

Difficulties with the numerical instability of the original method were overcome by incorporating an idea of *snake growing*. A single primary snake may begin which

later divides itself into pieces. The pieces of very low energy are allowed to grow in directions of their tangents while higher energy pieces are eliminated, as shown in Fig. 4.11. After each growing step, the energy of each snake piece is minimized (the ends are pulled to the true contour and the snake growing process is repeated). Further, the snake growing method may overcome the initialization problem. The primary snake may fall into an unlikely local minimum but parts of the snake may still lie on salient features. The very low energy parts (the probable pieces) of the primary snake are used to initialize the snake growing in later steps. This iterative snake growing always converges and the numerical solution is therefore stable. However, the robustness of the method is paid for by an increase in the processing cost of the algorithm. In the remainder of the section the main steps of the snake growing algorithm are described.

1. Based on a priori knowledge, estimate the desired contour position and shape as a curve S_0.

2. Use this curve S_0 to initialize the conventional snake algorithm. This yields a contour C_0.

3. Segment the contour C_0 to eliminate high-energy segments, resulting in a number of initial shorter low-energy contour segments C_0^i, where i is a segment identifier.

4. Repeat steps 5 and 6 while lengthening is needed.

5. Each contour segment C_k^i is allowed to grow in the direction of tangents (see Fig. 4.11). This yields a new estimate S_{k+1}^i for each contour segments C_k^i.

6. For each contour segment C_k^i, run the conventional snake algorithm using S_{k+1}^i as an initial estimate to get a new (longer) contour segment C_{k+1}^i.

A different approach to the energy integral minimization is based on a Galerkin solution of the finite-element method and has the advantage of greater numerical stability and better efficiency. This approach is especially useful in the case of closed or nearly closed contours. In particular, an additional *pressure force* is added to the contour interior by considering the curve as a *balloon* which is inflated. This allows the snake to overcome isolated energy valleys resulting from spurious edge points giving better results.

It should be pointed out that with active contour models, in general, the results of this error minimization are highly prone to the placement of the initial snake. By design, this is a mechanism to refine awareness of an object border, which is being attended to, good results are achieved by placing the initial snake near the object of interest. Conversely, the blurring approach could be taken to the extreme and the initial snake placed arbitrarily, but the result will have no particular significance beyond being an arbitrarily discovered contour in the image with certain continuity properties. In many implementations, the initial contour is a closed circle with location and radius specified by the user; closure is maintained by treating the first and last points in the snake as adjoining.

4.3 Windowing

The artificial vision in unstructured environments is typically realized by the determination of the variation lines of the contrast of the observed geometric shapes, e.g. angles and contours. The elaboration of the whole image in real time to extract this kind of features may require the use of sophisticated and powerful hardware, if the complete elaboration of the camera image sequence has to be realized. In fact, the use of software algorithms in place of dedicate hardware increases processing time of one or two orders of magnitude, and this may not be in accordance to the real-time

constraints.

However, not all the pixels of the image are necessary for the extraction of a chosen set of feature. Then, the computational time may be significantly reduced if only little *image windows* of the image frame are processed. Positioning of these windows into the frame have to be "predicted" via suitable dynamic filters to avoid that significant portion of the image may be lost. This strategy is called *window-based tracking technique*, or more easily *windowing*. This family of techniques present many advantages: computational efficiency, flexibility, and hardware and task independency. However, the advantages provided by the adoption of a windowing technique are directly proportional to the ratio between the total area of the elaborated image windows and the area of the whole image. So the best results may be reached for local feature based algorithms, e.g. corners and hole.

When feature searching has to be limited to very little image windows because of time constraints, then the prediction of the positioning of these windows become a crucial question. This problem is known as *feature following*, and it requires a filtering process to generate an estimation and a prediction of the actual and future positions, respectively, on the basis of noisy measurements of a suitable objects representation, as well as of the model of the objects motion.

4.3.1 Image window addressing

The pixels of the image frame are addressed by a 2D image coordinate frame. The notation used to indicate the value of the pixel at the coordinates

$$\boldsymbol{x} = \begin{bmatrix} r \\ c \end{bmatrix}$$

of an image captured at time t is $I(\boldsymbol{x}, t)$. An image window is a 2D set of pixels subject to an invertible function that relates the coordinates of a point of the window

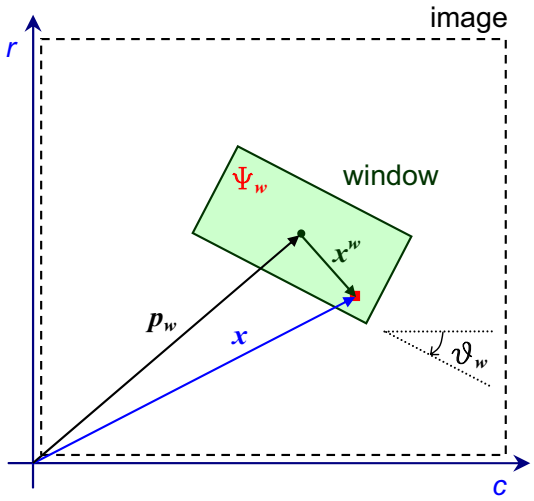

Figure 4.12. Windowing. Coordinate frames of the digitized image plane and of an image window

to the coordinates of the same point into the original image frame. In particular, this function is a rigid transformation, that is established by the translation vector

$$\boldsymbol{p}_w = \begin{bmatrix} r_w \\ c_w \end{bmatrix}$$

and the rotation angle ϑ_w. Hence, the value of a pixel with coordinates

$$\boldsymbol{x}^w = \begin{bmatrix} r^w \\ c^w \end{bmatrix},$$

with respect to the window frame, is expressed as follows

$$W(\boldsymbol{x}^w, t, \boldsymbol{p}_w, \vartheta_w) = W(\boldsymbol{p}_w + \boldsymbol{R}(\vartheta_w)\boldsymbol{x}^w, t),$$

with respect to the image frame, where $\boldsymbol{R}$ is a 2D rotation matrix (see Fig. 4.12). In this work it will be assumed that $\boldsymbol{x}^w = \boldsymbol{0}$ is the center of the window, and Ψ_w is the set of all possible values of $\boldsymbol{x}^w$ internal to the window.

The feature following algorithms generally operate in two steps. With the first step, all the current target windows are extracted from the image frame using the

nominal parameters of each window $\{\boldsymbol{p}_{w1}, \theta_{w1}, \ldots, \boldsymbol{p}_{wn}, \theta_{wn}, \}$. All the pixels of each window have copied into 2D arrays, that will be treated as normal image frames. In some applications, when dedicated hardware is available, the window's coping is not realized and the rigid transformations are used to address directly the relative portion of the image frame as well as distinct windows. During the second step the windows are elaborated to localize the searched features and to predict the next window parameters, using suitably filters. In the literature different filters have been proposed to cope with the feature following for visual applications. Among the most employed there are the $\alpha - \beta$ *filter*, the *Kalman filter*, the *auto-regressive filter* (AR), and the *auto-regressive filter with heterogeneous inputs* (ARX).

4.3.2 Local feature extraction

The windowing technique is particulary suitable to work together with a local image feature extraction, e.g. corners and holes. In fact, the small dimension of the image portion needed to extract an image feature like a corner or little holes allows setting up very small windows. Further, the sizing and placing of the windows may be realized so as to contain only one corner/hole for each window, and thus very simple algorithms for the corner/hole extraction may be used. Finally, another important goal of windowing is to check the feasibility of the target feature(s) for windowing and to automatically select windows that can be robustly tracked on the image plane in order to provide robust visual servoing.

If each window has to contain only one feature, the windows have to be centered on the "predicted" positions of the feature points on the image plane, so as predicted by one of the dynamic filters cited in the previous subsection. Further, considering that both the corner and the hole are image features developing around the respectively centers, the rotation angle ϑ_w may be always set to zero, leaving the window sizing with the task to contain an adequate portion of the image frame.

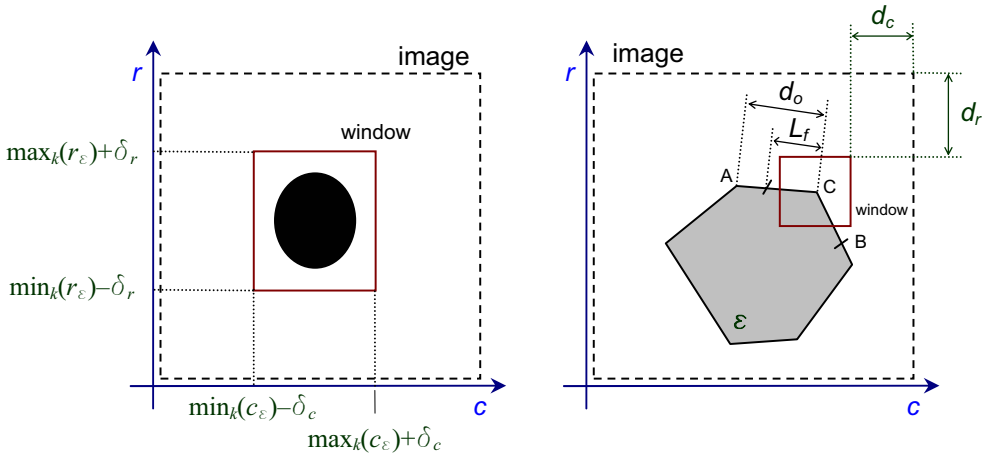

Figure 4.13. Window sizing. Left: window around a hole feature; right: window around a corner feature

The size of selected windows around the feature(s) should be between the maximum and minimum allowable window sizes, the windows must have enough clearance from the image plane boundaries, and they should not overlap other windows or portions of other features.

Given a polygonal approximation of a hole feature ε (see the left side of Fig. 4.13), the window will be centered at

$$r_w = \frac{1}{2}\left(\max_k(r_\varepsilon)_k + \min_k(r_\varepsilon)_k\right)$$
$$c_w = \frac{1}{2}\left(\max_k(c_\varepsilon)_k + \min_k(c_\varepsilon)_k\right)$$

with

$$W_r = \max_k(r_\varepsilon)_k - \min_k(r_\varepsilon)_k + 2\delta_r$$
$$W_c = \max_k(c_\varepsilon)_k - \min_k(c_\varepsilon)_k + 2\delta_c$$

where W_r and W_c are the sizes of window in the r and c directions, respectively, r_w and c_w denote the image coordinates of its center, $(r_\varepsilon)_k$ and $(c_\varepsilon)_k$ are the image coordinates of the k-th vertex of the feature ε in image frame, and δ_r, and δ_c are clearance factors in the r and c directions of the image frame, respectively. It should

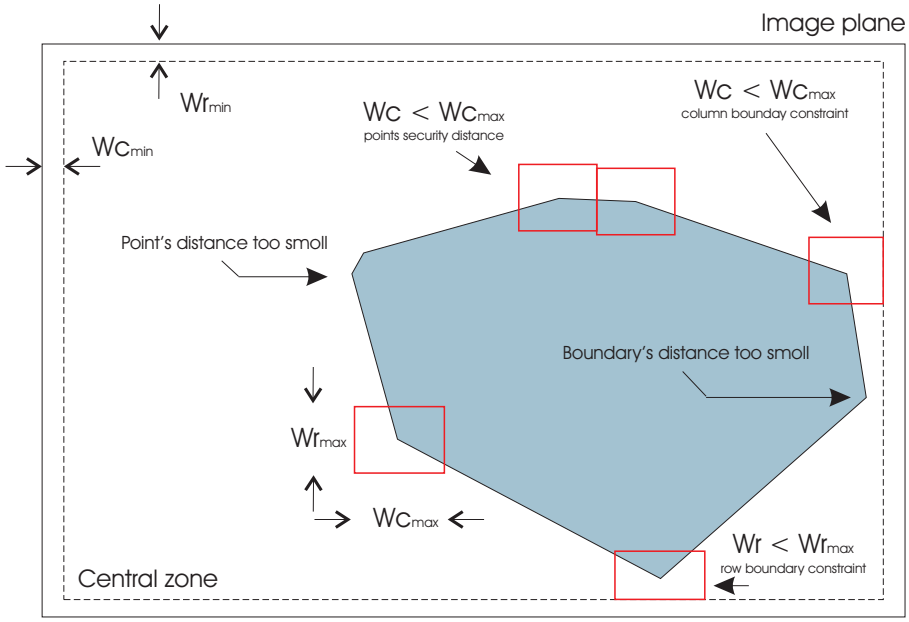

Figure 4.14. Windowing. Significant examples of windowing with corner features

be noted that the window for a hole feature, due to its geometry and feature extraction process, will hardly overlap other windows or portions of other features.

For a corner feature, however, the closest feature to the target feature in the image plane has to be determined. The image distance of the closest feature from the target feature ε_k is denoted by d_o (see the right side of Fig. 4.13), i.e., $d_o = \min_j \|\varepsilon_k - \varepsilon_j\|_2 \ \forall \varepsilon_j \in \Upsilon$ and $k \neq j$, where Υ is the set of candidate features. First, a secure distance is calculated by $L_f = d_o/C_f$, where $C_f > 1$ is a clearance factor. If a window is bounded by this distance, it will not interfere with other features. Let A, B be the points on the image plane along two edges of the corner C with distance L_f from the corner. Then, Δ can be defined as

$$\Delta = \max\left(|r_C - r_A|, |c_C - c_A|, |r_C - r_B|, |c_C - c_B|\right)$$

In addition, to provide clearance from image frame boundaries, d_r and d_c are defined

as

$$d_r = \frac{D_r}{2} - |r_C|$$
$$d_c = \frac{D_c}{2} - |c_C|$$

where D_r and D_c are the dimension of the image frame along the r and c direction, respectively.

Finally, the window will be centered at the corner with

$$W_r = \min\left(2\Delta, 2d_r, W_r^{max}\right)$$
$$W_c = \min\left(2\Delta, 2d_c, W_c^{max}\right)$$

where W_r^{max} and W_c^{max} are the maximum allowable window sizes along r and c directions, respectively. In Fig. 4.14 some significant cases for the sizing and placing of windows for corner features are shown.

Notice that a more sophisticated windowing approach may be implemented if the limit of using only one window for each feature is removed. Using two or more windows to cover the portion of the image frame of interest represents the evolution of the windowing technique described before. With more windows, in fact, the dimension of the image area to elaborate may be further reduced with the consequent reduction of computational time.

CHAPTER 5

POSE RECONSTRUCTION

The problem of *pose reconstruction* of known moving objects using image measurements is the issue of this chapter. The geometric formulation of the pose reconstruction problem is first presented for the case of a fixed mono-camera system using local image features, e.g. corners and holes, as image measurements. Then, the problem is extended to the general case of a fixed multi-camera system.

An important problem to consider is that the visual measurements are usually affected by significant noise and disturbances due to temporal and spatial sampling and quantization of the image signal, lens distortion, etc.. Hence, the use of visual measurements requires the adoption of suitable algorithms with good disturbance rejection capability.

The proposed solutions make use of the Extended Kalman Filter to solve the complex and nonlinear equations resulting from the geometric formulation, both in the case of a mono and of a multi-camera system fixed in the workspace. Further, to reduce computational complexity, an iterative formulation of the filter is proposed. Finally, an adaptive formulation of the Extended Kalman filter is introduced to enhance robustness with respect to light conditions.

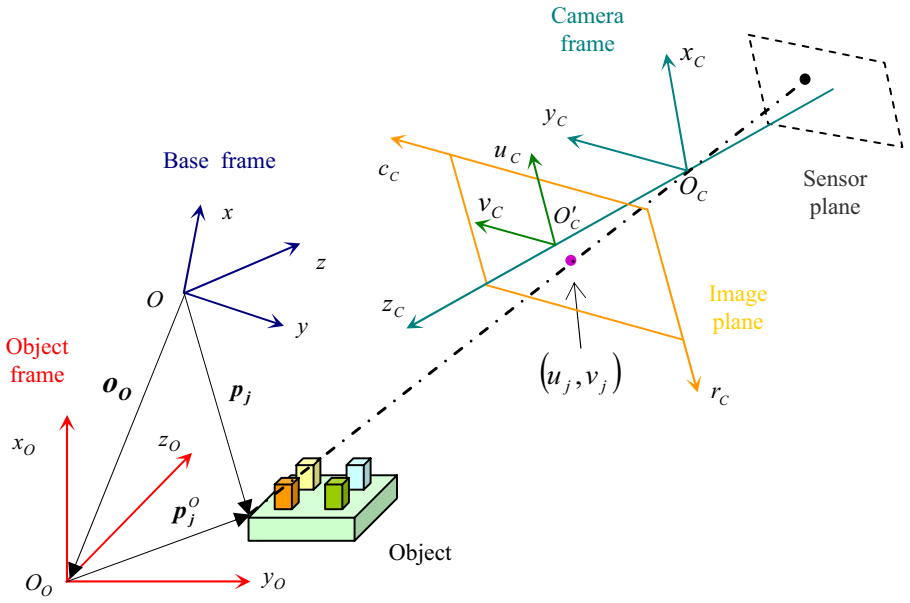

Figure 5.1. Fixed mono-camera system. Reference frames for the camera and the object using the pinhole model

5.1 Pose reconstruction from image measurements

In this section the geometric formulation of the problem of the pose reconstruction of an object of known geometry will be presented. The reconstruction is based on image measurements of the object *feature points*, i.e. the corners.

Two cases will be studied in depth: the case of a fixed mono-camera system and the case of a fixed multi-camera system.

5.1.1 Fixed mono-camera system

The geometry of a system with a single video camera fixed in the workspace can be characterized, as described in the Section 3.1, using the classical pin-hole model shown in Fig. 5.1.

A frame O_c–$x_c y_c z_c$ attached to the camera (*camera frame*), with the z_c-axis aligned to the optical axis and the origin in the optical center, is considered. The sensor plane is parallel to the $x_c y_c$-plane at a distance $-f_e$ along the z_c-axis, where f_e

is the effective focal length of the camera lens, which may be different from the focal length f. The image plane is parallel to the $x_c y_c$-plane at a distance f_e along the z_c-axis. The intersection of the optical axis with the image plane defines the principal optic point O'_c. It is the origin of the image frame O'_c–$u_c v_c$ whose axes u_c and v_c are taken parallel to the axes x_c and y_c, respectively.

The *perspective transformation* of the j-th object feature point[13] P_j, whose position components are described by the vector

$$\boldsymbol{p}_j^O = \begin{bmatrix} x_j^O \\ y_j^O \\ z_j^O \end{bmatrix}$$

with respect to the *object frame* (a frame attached to the object), onto the image plane is defined by (3.2). The corresponding projection geometry is showm in Fig. 5.1. The positions of all the feature points with respect to the object frame are assumed to be known, i.e. they are provided by a boundary representation of the object, by direct measurements, or by a generic CAD model.

The pose of the object frame with respect to the base frame O-xyz is described by the translation vector

$$\boldsymbol{o}_O = \begin{bmatrix} x_O \\ y_O \\ z_O \end{bmatrix}$$

[13]The object feature points considered in this section are local features, e.g. corners and holes.

and a minimal representation of the orientation

$$\boldsymbol{\phi}_O = \begin{bmatrix} \varphi_O \\ \vartheta_O \\ \psi_O \end{bmatrix}$$

based, for example, on the *roll*, *pitch*, and *yaw* angle [37]. These quantities are the *unknown variables* of the pose reconstruction problem. Notice that, for simplicity of notation, the indication of the superscript for the vectors referred to the base frame is omitted. Then, the corresponding coordinate transformation of the j-th feature point from the object frame to the base frame is defined as follows:

$$\boldsymbol{p}_j = \boldsymbol{o}_O + \boldsymbol{R}(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O, \tag{5.1}$$

where the matrix $\boldsymbol{R}(\boldsymbol{\phi}_O)$ is the rotation matrix depending from the roll, pitch, and yaw angle via the following expression:

$$\boldsymbol{R}(\boldsymbol{\phi}_O) = \begin{bmatrix} c_{\varphi O}c_{\vartheta O} & c_{\varphi O}s_{\vartheta O}s_{\psi O} - s_{\varphi O}c_{\psi O} & c_{\varphi O}s_{\vartheta O}c_{\psi O} + s_{\varphi O}s_{\psi O} \\ s_{\varphi O}c_{\vartheta O} & s_{\varphi O}s_{\vartheta O}s_{\psi O} + c_{\varphi O}c_{\psi O} & s_{\varphi O}s_{\vartheta O}c_{\psi O} - c_{\varphi O}s_{\psi O} \\ -s_{\varphi O} & c_{\vartheta O}s_{\psi O} & c_{\vartheta O}c_{\psi O} \end{bmatrix}.$$

Notice that the compact notations $c_{\varphi O}$, $s_{\varphi O}$, ... have been used to indicate $\cos(\varphi_O)$, $\sin(\varphi_O)$, ..., respectively.

Consider the translation vector $\boldsymbol{o}^C$ and the rotation matrix $\boldsymbol{R}^C$ of the base frame with respect to the camera frame. Then, the coordinates of the j-th object point with respect to camera frame may be evaluated with the following expression:

$$\boldsymbol{p}_j^C = \boldsymbol{o}^C + \boldsymbol{R}^C\boldsymbol{p}_j. \tag{5.2}$$

Notice that both the vector $\boldsymbol{o}^C$ and the matrix $\boldsymbol{R}^C$ are constant because the camera is fixed with respect to the base frame. These quantities, known as *extrinsic camera parameters*, may be evaluated via a suitable calibration procedure of the camera.

Starting from (3.2), the coordinates of the j-th feature points with respect to the normalized image frame can be computed as follows:

$$\begin{bmatrix} \overline{u}_j \\ \\ \overline{v}_j \end{bmatrix} = \begin{bmatrix} \dfrac{x_j^C}{z_j^C} \\ \\ \dfrac{y^C}{z^C} \end{bmatrix}, \tag{5.3}$$

where

$$\boldsymbol{p}_j^C = \begin{bmatrix} x_j^C \\ \\ y_j^C \\ \\ z_j^C \end{bmatrix}.$$

These quantities may also be derived directly from the corresponding measured pixel coordinates, according to (3.4). The pixel coordinates of the feature points, the position of the corresponding object points with respect to the object frame, and the camera calibration parameters are the *known variables* of the pose reconstruction problem. Notice that using the expressions (3.17) it is possible to compensate the distortion effects on the image measurements of the feature points.

Consider an object with m feature points. Substituting (5.1) into (5.2), for $j = 1, \ldots, m$, the expression of the coordinates of the m feature points with respect to the camera frame are obtained depending on the camera (known) pose and on the object (unknown) pose, as well as on the position of the feature points with respect to the object frame

$$\boldsymbol{p}_j^C = \boldsymbol{o}^C + \boldsymbol{R}^C(\boldsymbol{o}_O + \boldsymbol{R}(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O). \tag{5.4}$$

These quantities may be substituted into the m equations (5.3), which depend only on the digitized image coordinates (the image measurements), i.e.:

$$\begin{bmatrix} r_j \\ \\ c_j \end{bmatrix} = \begin{bmatrix} r_o \\ \\ c_o \end{bmatrix} + \begin{bmatrix} f_u & 0 \\ \\ 0 & f_v \end{bmatrix} \begin{bmatrix} \dfrac{x_j^C}{z_j^C} \\ \\ \dfrac{y^C}{z^C} \end{bmatrix}, \tag{5.5}$$

where the quantities $\{r_o, c_o, f_u, f_v\}$ are the so called *intrinsic camera parameters*, they are constant and may be evaluated via a suitable calibration procedure of the camera.

Therefore, a system of $2m$ nonlinear scalar equations is achieved, which depend on the measurements of the m feature points in the image plane of the camera, whereas the six components of the vectors $\boldsymbol{o}_O$ and $\boldsymbol{\phi}_O$, expressed in the base frame, are the unknown variables. To solve these equation at least six independent equations are required, which can be achieved with the measurement of at least three non-colinear feature points, although additional feature points may enhance the accuracy of the estimate in the presence of noise. Notice that the computation of the solution is nontrivial and, for visual servoing applications, it has to be repeated at a high sampling rate.

5.1.2 Fixed multi-camera system

The geometric formulation of the pose reconstruction problem presented in the previous section may be naturally extended to the general case of a fixed multi-camera system. The geometry of a system of n video cameras can still be characterized using the pinhole model. In Fig. 5.2 the corresponding referring frames have represented.

Consider the translation vector $\boldsymbol{p}^{Ci}$ and the rotation matrix $\boldsymbol{R}^{Ci}$ of the base frame with respect to the i-th camera frame (the frame attached to the i-th camera). Then the coordinate transformation for the j-th feature points from the base frame to the i-th camera frame is described by the following equation:

$$\boldsymbol{p}_j^{Ci} = \boldsymbol{o}^{Ci} + \boldsymbol{R}^{Ci}\boldsymbol{p}_j, \tag{5.6}$$

where the quantities $\boldsymbol{p}^{Ci}$ and $\boldsymbol{R}^{Ci}$, for $i = 1, \ldots, n$, are the extrinsic calibration parameters od the n cameras.

The normalized image coordinates of the i-th camera corresponding to the

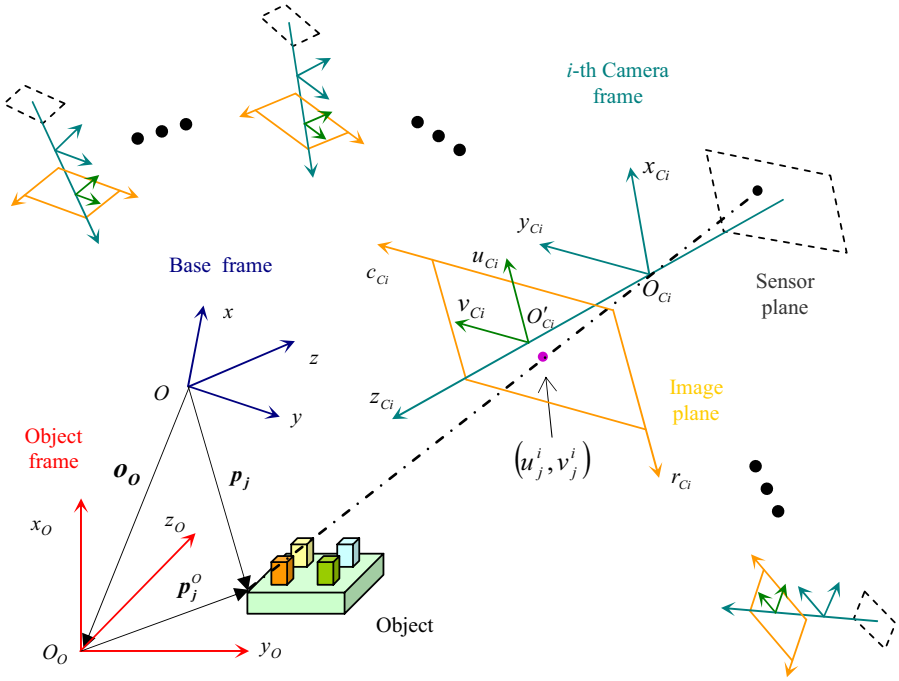

Figure 5.2. Fixed multi-camera system. Reference frames for the i-th camera and the object using the pinhole model

j-th feature points:

$$\begin{bmatrix} \overline{u}_j^i \\ \\ \overline{v}_j^i \end{bmatrix} = \begin{bmatrix} \dfrac{x_j^{Ci}}{z_j^{Ci}} \\ \\ \dfrac{y^{Ci}}{z^{Ci}} \end{bmatrix}, \tag{5.7}$$

where

$$\boldsymbol{p}_j^{Ci} = \begin{bmatrix} x_j^{Ci} \\ \\ y_j^{Ci} \\ \\ z_j^{Ci} \end{bmatrix}.$$

These quantities may also be derived directly from the corresponding measured pixel coordinates, via (3.4).

Assuming that the object has m feature points and substituting (5.1) into the corresponding n equations (5.6), the expression of the coordinates of the feature

points with respect to the n camera frames is achieved

$$\boldsymbol{p}_j^{Ci} = \boldsymbol{o}^{Ci} + \boldsymbol{R}^{Ci}(\boldsymbol{o}_O + \boldsymbol{R}(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O), \tag{5.8}$$

where

$$\boldsymbol{p}_j^{Ci} = \begin{bmatrix} x_j^{Ci} \\ y_j^{Ci} \\ z_j^{Ci} \end{bmatrix}.$$

Substituting (5.8) for each of the n cameras into the m equations (5.3), a system of $2nm$ nonlinear scalar equations may be achieved, which depend on the measurements of the m feature points in the n image planes of the cameras, whereas the six components of the vectors $\boldsymbol{o}_O$ and $\boldsymbol{\phi}_O$, expressed in the base frame, are the unknown variables. The corresponding system of equations may be described as follows:

$$\begin{bmatrix} r_j^i \\ c_j^i \end{bmatrix} = \begin{bmatrix} r_o^i \\ c_o^i \end{bmatrix} + \begin{bmatrix} f_u^i & 0 \\ 0 & f_v^i \end{bmatrix} \begin{bmatrix} \dfrac{x_j^{Ci}}{z_j^{Ci}} \\ \dfrac{y^{Ci}}{z^{Ci}} \end{bmatrix}, \tag{5.9}$$

for $j = 1, \ldots, m$ and $i = 1, \ldots, n$, where the quantities $\{r_o^i, c_o^i, f_u^i, f_v^i\}$ are the intrinsic camera parameters of the n cameras.

To find a solution, at least six independent equations are required, which can be achieved with the measurement of at least three non-colinear feature points, anyhow distributed between the n cameras. The effects of triangulation, due to the presence of a stereo camera system, should reduce and make of the same order of magnitude the components of the estimation errors of the pose. However, additional feature points may still enhance accuracy in the presence of noise.

Notice that, both a the case of a mono and a multi-camera system, the solution of the corresponding system of equations provides the pose of the object directly with

respect to the base frame, without the necessity of further coordinate transformations.

5.2 Extended Kalman Filter

The iterative Extended Kalman Filter formulation described in the following sections provides a computationally tractable solution to the previous system of equations, which can also incorporate redundant measurement information [24], [27]. Actually, Kalman filtering offers many advantages over other pose estimation methods [1], [7], [46], [15], [47], e.g., implicit solution of photogrammetric equations with iterative implementation, temporal filtering, ability to change the measurement set during the operation. Moreover, the statistical properties of Kalman Filter may be tuned to those of the image measurements noise of the particular vision system. Last but not least, the prediction capability of the filter allows setting up a dynamic windowing technique of the image plane which may sensibly reduce image processing time. Applications of Kalman Filter in machine vision range from visual tracking of objects with many internal degrees of freedom [34], to automatic grasp planning [19] and [14] as well as pose and size parameters estimation of objects with partially known geometry [22].

The accuracy of the provided solution depends on the accuracy of the model of the dynamic system representing the object motion (*discrete-time state space model* of the filter), on the accuracy of the model of the image projections (*output equations* of the filter), and on the validity of the assumption that the disturbance and measurement noises are well represented by a Gaussian noise, with zero mean and fixed covariances[14] (*statistics* of the filer) [35], [45], [19]. If this hypothesis is satisfied, the provided solution is the optimal solution; otherwise the solution represents an approximation to the optimal solution.

---

[14]In the case of the proposed Adaptive Extended Kalman filter the statistics of the filter (means and covariances) will be self-tuning.

5.2.1 Iterative formulation

In order to estimate the pose of the object using the image measurements of the object feature points, a discrete-time state space model of the object motion has to be considered [23], [42]. The state vector of the model is chosen as the (12×1) vector

$$\boldsymbol{w} = \begin{bmatrix} x_o & \dot{x}_o & y_o & \dot{y}_o & z_o & \dot{z}_o & \varphi_o & \dot{\varphi}_o & \vartheta_o & \dot{\vartheta}_o & \psi_o & \dot{\psi}_o \end{bmatrix}^{\mathrm{T}}. \tag{5.10}$$

For simplicity, the object velocity is assumed to be constant over a sample period T. This approximation is reasonable in under the assumption that T is sufficiently small with respect to the variation of the object pose. The corresponding dynamic modelling error can be considered as an input disturbance $\boldsymbol{\gamma}_k$. The discrete-time dynamic model can be written as

$$\boldsymbol{w}_k = \boldsymbol{A}\boldsymbol{w}_{k-1} + \boldsymbol{\gamma}_k \tag{5.11}$$

where the state transition matrix $\boldsymbol{A}$ is a constant (12×12) block diagonal matrix of the form

$$\boldsymbol{A} = \begin{bmatrix} \begin{matrix} 1 & T \\ 0 & 1 \end{matrix} & & & \boldsymbol{0} \\ & \ddots & & \\ \boldsymbol{0} & & & \begin{matrix} 1 & T \\ 0 & 1 \end{matrix} \end{bmatrix}.$$

Without loss of generality, suppose that all the m feature points of the object are visible from each camera[15]. The outputs of the Kalman Filter are chosen as the two vectors of the normalized coordinates of the feature points in the image plane of

[15]This hypothesis is not necessary and it will be removed in the next sections, when the issue of dynamic loss of the visible feature points will be discussed.

the n cameras

$$\boldsymbol{\zeta}_{u,k} = \left[\left(\boldsymbol{\zeta}_{u,k}^1\right)^{\mathrm{T}} \quad \cdots \quad \left(\boldsymbol{\zeta}_{u,k}^n\right)^{\mathrm{T}} \right]_k^{\mathrm{T}} \tag{5.12a}$$

$$\boldsymbol{\zeta}_{v,k} = \left[\left(\boldsymbol{\zeta}_{v,k}^1\right)^{\mathrm{T}} \quad \cdots \quad \left(\boldsymbol{\zeta}_{v,k}^n\right)^{\mathrm{T}} \right]_k^{\mathrm{T}}, \tag{5.12b}$$

where the vectors $\boldsymbol{\zeta}_{u,k}^i$ and $\boldsymbol{\zeta}_{v,k}^i$, for $i = 1, \ldots, n$, represent the normalized coordinates of the feature points for the i-th camera

$$\boldsymbol{\zeta}_{u,k}^i = \left[\frac{u_1^i}{f_e^i} \quad \cdots \quad \frac{u_m^i}{f_e^i} \right]_k^{\mathrm{T}} \tag{5.13a}$$

$$\boldsymbol{\zeta}_{v,k}^i = \left[\frac{v_1^i}{f_e^i} \quad \cdots \quad \frac{v_m^i}{f_e^i} \right]_k^{\mathrm{T}}. \tag{5.13b}$$

In view of (5.7), the corresponding output model can be written in the form

$$\boldsymbol{\zeta}_{u,k} = \boldsymbol{g}_u(\boldsymbol{w}_k) + \boldsymbol{\nu}_{u,k} \tag{5.14a}$$

$$\boldsymbol{\zeta}_{v,k} = \boldsymbol{g}_v(\boldsymbol{w}_k) + \boldsymbol{\nu}_{v,k} \tag{5.14b}$$

where $\boldsymbol{\nu}_{u,k}$ and $\boldsymbol{\nu}_{v,k}$ are the observation noise vectors for the u and v components of the normalized image planes of each camera, whereas the vector functions $\boldsymbol{g}_u(\boldsymbol{w}_k)$ and $\boldsymbol{g}_v(\boldsymbol{w}_k)$ are defined as

$$\boldsymbol{g}_u(\boldsymbol{w}_k) = \left[\boldsymbol{g}_u^1(\boldsymbol{w}_k)^{\mathrm{T}} \quad \cdots \quad \boldsymbol{g}_u^n(\boldsymbol{w}_k)^{\mathrm{T}} \right]_k^{\mathrm{T}} \tag{5.15a}$$

$$\boldsymbol{g}_v(\boldsymbol{w}_k) = \left[\boldsymbol{g}_v^1(\boldsymbol{w}_k)^{\mathrm{T}} \quad \cdots \quad \boldsymbol{g}_v^n(\boldsymbol{w}_k)^{\mathrm{T}} \right]_k^{\mathrm{T}}, \tag{5.15b}$$

where the i-th vectors $\boldsymbol{g}_u^i(\boldsymbol{w}_k)$ and $\boldsymbol{g}_v^i(\boldsymbol{w}_k)$, for $i = 1, \ldots, n$, are defined as follows:

$$\boldsymbol{g}_u^i(\boldsymbol{w}_k) = \left[\frac{x_1^{Ci}}{z_1^{Ci}} \quad \cdots \quad \frac{x_m^{Ci}}{z_m^{Ci}} \right]_k^{\mathrm{T}} \tag{5.16a}$$

$$\boldsymbol{g}_v^i(\boldsymbol{w}_k) = \left[\frac{y_1^{Ci}}{z_1^{Ci}} \quad \cdots \quad \frac{y_m^{Ci}}{z_m^{Ci}} \right]_k^{\mathrm{T}}. \tag{5.16b}$$

The coordinates x_j^{Ci}, y_j^{Ci}, and z_j^{Ci} of the j-th feature points with respect to the i-th camera in equations (5.15) are computed from the state vector $\boldsymbol{w}_k$ via equation (5.6).

The components of the disturbance quantities $\boldsymbol{\gamma}_k$, $\boldsymbol{\nu}_{u,k}$ and $\boldsymbol{\nu}_{v,k}$ are considered as independent, non-stationary, Gaussian, white noise sequences with the statistical properties

$$\mathrm{E}[\boldsymbol{\gamma}_k] = \boldsymbol{q}_k \tag{5.17a}$$

$$\mathrm{E}[\boldsymbol{\nu}_{u,k}] = \boldsymbol{r}_{u,k} \tag{5.17b}$$

$$\mathrm{E}[\boldsymbol{\nu}_{v,k}] = \boldsymbol{r}_{v,k} \tag{5.17c}$$

$$\mathrm{E}[(\boldsymbol{\gamma}_k - \boldsymbol{q}_k)(\boldsymbol{\gamma}_l - \boldsymbol{q}_l)^{\mathrm{T}}] = \boldsymbol{Q}_k \delta_{kl} \tag{5.17d}$$

$$\mathrm{E}[(\boldsymbol{\nu}_{u,k} - \boldsymbol{r}_{u,k})(\boldsymbol{\nu}_{u,l} - \boldsymbol{r}_{u,l})] = \boldsymbol{R}_{u,k} \delta_{kl} \tag{5.17e}$$

$$\mathrm{E}[(\boldsymbol{\nu}_{v,k} - \boldsymbol{r}_{v,k})(\boldsymbol{\nu}_{v,l} - \boldsymbol{r}_{v,l})] = \boldsymbol{R}_{v,k} \delta_{kl} \tag{5.17f}$$

where $\mathrm{E}[\cdot]$ indicates the statistical mean operator applied to the components of a vector or matrix, and δ is the *Kroneker symbol*. Notice that the two matrixes $\boldsymbol{R}_{u,k}$ and $\boldsymbol{R}_{v,k}$ become block diagonal matrices if the measurements of the n cameras are independent, where the i-th block represents the observation covariance matrix of the i-th camera.

Since the output model is nonlinear in the system state, it is required to linearize the output equations about the current state estimate at each sample time. This leads to the so-called Extended Kalman Filter (EKF).

The following iterative algorithm provides the best linear, minimum variance, unbiased estimate of the state vector for the system defined by (5.11) and (5.14).

The *update step* improves the previous estimate by using the input measurements according to the equations

$$\boldsymbol{w}_{k,k} = \boldsymbol{w}_{k,k-1} + \begin{bmatrix} \boldsymbol{K}_{u,k} & \boldsymbol{K}_{v,k} \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_{u,k} - \boldsymbol{g}_u(\boldsymbol{w}_{k,k-1}) - \boldsymbol{r}_{u,k} \\ \boldsymbol{\zeta}_{v,k} - \boldsymbol{g}_v(\boldsymbol{w}_{k,k-1}) - \boldsymbol{r}_{v,k} \end{bmatrix} \tag{5.18a}$$

$$\boldsymbol{P}_{k,k} = \boldsymbol{P}_{k,k-1} - \begin{bmatrix} \boldsymbol{K}_{u,k} & \boldsymbol{K}_{v,k} \end{bmatrix} \begin{bmatrix} \boldsymbol{H}_{u,k} \\ \\ \boldsymbol{H}_{v,k} \end{bmatrix} \boldsymbol{P}_{k,k-1}, \tag{5.18b}$$

where $\boldsymbol{w}_{k,k-1}$ is the propagated state vector, $\boldsymbol{P}_{k,k-1}$ is the (12×12) covariance matrix conditioned on observations prior to time k, and $\boldsymbol{K}_{u,k}$ and $\boldsymbol{K}_{v,k}$ are the $(12 \times nm)$ Kalman matrix gains

$$\boldsymbol{K}_{u,k} = \boldsymbol{P}_{k,k-1}\boldsymbol{H}_{u,k}{}^{\mathrm{T}}(\boldsymbol{R}_{u,k} + \boldsymbol{\Gamma}_{u,k})^{-1} \tag{5.19a}$$

$$\boldsymbol{K}_{v,k} = \boldsymbol{P}_{k,k-1}\boldsymbol{H}_{v,k}{}^{\mathrm{T}}(\boldsymbol{R}_{v,k} + \boldsymbol{\Gamma}_{v,k})^{-1}, \tag{5.19b}$$

being $\boldsymbol{H}_{u,k}$ and $\boldsymbol{H}_{v,k}$ the $(nm \times 12)$ Jacobian matrices of the output vector functions

$$\boldsymbol{H}_{u,k} = \left.\frac{\partial \boldsymbol{g}_u(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}_{k,k-1}} \tag{5.20a}$$

$$\boldsymbol{H}_{v,k} = \left.\frac{\partial \boldsymbol{g}_v(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}_{k,k-1}}. \tag{5.20b}$$

The analytic expressions of $\boldsymbol{H}_{u,k}$ and $\boldsymbol{H}_{v,k}$ can be found in Appendix B, while $\boldsymbol{\Gamma}_{u,k}$ and $\boldsymbol{\Gamma}_{v,k}$ are defined as

$$\boldsymbol{\Gamma}_{u,k} = \boldsymbol{H}_{u,k}\boldsymbol{P}_{k,k-1}\boldsymbol{H}_{u,k}{}^{\mathrm{T}} \tag{5.21a}$$

$$\boldsymbol{\Gamma}_{v,k} = \boldsymbol{H}_{v,k}\boldsymbol{P}_{k,k-1}\boldsymbol{H}_{v,k}{}^{\mathrm{T}}. \tag{5.21b}$$

The *prediction step* of the algorithm provides an optimal estimate of the state at the next sample time according to the iterative equations

$$\boldsymbol{w}_{k+1,k} = \boldsymbol{A}\boldsymbol{w}_{k,k} + \boldsymbol{q}_k \tag{5.22a}$$

$$\boldsymbol{P}_{k+1,k} = \boldsymbol{A}\boldsymbol{P}_{k,k}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{Q}_k. \tag{5.22b}$$

Notice that a-priori estimate of the state $\boldsymbol{w}_0$ and of the state covariance $\boldsymbol{P}_0$ and a-priori statistical information represented by (5.17) are required. Further, the

covariance matrix $\boldsymbol{Q}_k$ of the state noise and the covariance matrixes $\boldsymbol{R}_{u,k}$ and $\boldsymbol{R}_{v,k}$ of the observation noise have to be set. Typically, they are fixed to constant values, which are achieved by some special measurements and empirical experiments. In particular, matrices $\boldsymbol{R}_{u,k}$ and $\boldsymbol{R}_{v,k}$ may be measured during the calibration procedure of the cameras via a suitable statistical analysis of the residuals of the calibration errors. The choice of the matrix $\boldsymbol{Q}_k$ is more difficult, because it strongly depends on the motion parameters of the object, and in particular from the velocity. Typically, it is the result of empirical experiments.

5.2.2 Scalar iterative formulation

The computational complexity of the proposed formulation of the Kalman filter may represent a crucial issue for real time applications, e.g. visual servoing. The iterative structure already offers good performance in terms of computational complexity; nevertheless, it may be further enhanced if a *scalar update algorithm* is used to evaluate the Kalman gains, the state covariance matrix, and the state vector, during the upgrade step. This algorithm may be used when the observation noise matrixes may be approximated by diagonal matrices[16]. In this case, the computationally expensive evaluation of the inverse matrixes $(\boldsymbol{R}_{u,k}+\boldsymbol{\Gamma}_{u,k})^{-1}$ and $(\boldsymbol{R}_{v,k}+\boldsymbol{\Gamma}_{v,k})^{-1}$ may be substituted with a scalar iterative algorithm. In this case, Equations (5.18) and (5.19) are replaced with the following equations:

$$\overline{\boldsymbol{P}}_k^0 = \boldsymbol{P}_{k,k-1} \tag{5.23a}$$

`for` $j = 1$ `to` nm,

$$\boldsymbol{k}_{uk,j} = \overline{\boldsymbol{P}}_k^{j-1}\boldsymbol{h}_{uk,j}{}^{\mathrm{T}}(R_{u,k}(j,j) + \boldsymbol{h}_{uk,j}\overline{\boldsymbol{P}}_k^{j-1}\boldsymbol{h}_{uk,j}^{\mathrm{T}})^{-1} \tag{5.23b}$$

[16]The hypothesis that the covariance matrices of the observation noise are diagonal is very reasonable because the correlation from the measurements of different feature points may depend only on the position in the image plane, which depends on the illumination condition, and not much on the intrinsic correlation between each pair of feature points.

$$\boldsymbol{k}_{vk,j} = \overline{\boldsymbol{P}}_k^{j-1} \boldsymbol{h}_{vk,j}{}^{\mathrm{T}} (R_{v,k}(j,j) + \boldsymbol{h}_{vk,j} \overline{\boldsymbol{P}}_k^{j-1} \boldsymbol{h}_{vk,j}^{\mathrm{T}})^{-1} \tag{5.23c}$$

$$\overline{\boldsymbol{P}}_k^{j} = \overline{\boldsymbol{P}}_k^{j-1} - \begin{bmatrix} \boldsymbol{k}_{uk,j} & \boldsymbol{k}_{vk,j} \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_{uk,j} \\ \\ \boldsymbol{h}_{vk,j} \end{bmatrix} \overline{\boldsymbol{P}}_k^{j-1} \tag{5.23d}$$

`end`

$$\boldsymbol{P}_{k,k} = \overline{\boldsymbol{P}}_k^{m_k} \tag{5.23e}$$

$$\boldsymbol{w}_{k,k} = \boldsymbol{w}_{k,k-1} + \boldsymbol{P}_{k,k} \begin{bmatrix} \boldsymbol{H}_{u,k}^{\mathrm{T}} & \boldsymbol{H}_{v,k}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_{u,k}^{-1} \\ \\ \boldsymbol{R}_{v,k}^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_{u,k} - \boldsymbol{g}_u(\boldsymbol{w}_{k,k-1}) - \boldsymbol{r}_{u,k} \\ \\ \boldsymbol{\zeta}_{v,k} - \boldsymbol{g}_v(\boldsymbol{w}_{k,k-1}) - \boldsymbol{r}_{v,k} \end{bmatrix} \tag{5.23f}$$

where $R_{u,k}(j,j)$ and $R_{v,k}(j,j)$ are the j-th elements of the diagonals of the corresponding matrices, and $\boldsymbol{h}_{uk,j}$ and $\boldsymbol{h}_{vk,j}$ are the j-th rows of the matrixes $\boldsymbol{H}_{u,k}$ and $\boldsymbol{H}_{v,k}$, respectively.

The same algorithm, with few modifications, may have a parallel formulation to be executed by a multi-processor system [16].

5.2.3 Copying with dynamic loss of image features

In this section the non-realistic hypothesis that all the object feature points must be always visible and extractable from the image frame of each camera will be removed and the general case of dynamic loss of the available feature points will be considered. The loss of a feature point may have two main causes: occlusion and extraction failure.

The occlusion is the loss of visibility of a feature point from the current observation point, and it may be of two kinds: auto-occlusion and self-occlusion. The former occurs when the feature point is hidden by the observable surfaces of the object, while the laster is caused by the alignment of other objects, obstacles, or parts of the environment between the camera and the feature points.

The extraction failure may have many reasons. Typically, the two most fre-

quent causes are the uncertain extraction and the windowing mismatch. The former occurs when the processed piece of the image is very noisy, because of a poor illumination condition. In this case, during the feature extraction, it may not be possible to identify the feature point, or the standard deviation of the distance between the predicted feature point position and its real position may be too high with respect to the mean value achieved in the extraction of the other points[17].

The windowing mismatch happens when the predicted position and size of the window of the windowing algorithm does not allow the feature extraction. Typically, this situation occurs when the object has a strong acceleration or when, for reasons of computational limits of the employed hardware, very small windows have to be used.

To cope with the dynamic loss of available feature points, the case of variable and dynamic subsets of feature points provided by each camera has to be considered. Consider the whole set of the object feature points $\Omega = \{P_1, \ldots, P_m\}$ and its subset $\Omega_k^i = \{P_{j_{1,k}^i}, \ldots, P_{j_{m_k^i,k}^i}\}_k$, for $i = 1, \ldots, n$, where $j_{h,k}^i$ is the h-th feature points of the subset of the effectively available feature points of the i-th camera, and m_k^i is the number of points of Ω_k^i, at the time k, with $m_k^i \leq m$. The total number of visible points at time k from all the n cameras is $m_k = \sum_{i=1}^{n} m_k^i$, with $m_k \leq nm$.

The outputs of the Kalman Filter have to be dynamically composed, at each sampling time, using the effectively available feature points. Therefore the component vectors $\boldsymbol{\zeta}_{u,k}^i$ and $\boldsymbol{\zeta}_{v,k}^i$ of (5.13) have to be dynamically defined for each camera as follows:

$$\boldsymbol{\zeta}_{u,k}^i = \left[\frac{u_{j_{1,k}^i}^i}{f_e^i} \quad \ldots \quad \frac{u_{j_{m_k^i,k}^i}^i}{f_e^i} \right]_k^{\mathrm{T}} \tag{5.24a}$$

[17]This is only one of the possible statistical checks that may be used to verify the reliability of the results of the extraction process.

$$\boldsymbol{\zeta}^i_{v,k} = \left[\frac{v^i_{j^i_{1,k}}}{f^i_e} \quad \dots \quad \frac{v^i_{j^i_{m^i_k,k}}}{f^i_e} \right]^{\mathrm{T}}_k . \tag{5.24b}$$

Analogously, the component vectors $\boldsymbol{g}^i_u(\boldsymbol{w}_k)$ and $\boldsymbol{g}^i_v(\boldsymbol{w}_k)$ of the corresponding output model, described by (5.16), have to be dynamically defined for each camera as follows:

$$\boldsymbol{g}^i_u(\boldsymbol{w}_k) = \left[\frac{x^{Ci}_{j^i_{1,k}}}{z^{Ci}_{j^i_{1,k}}} \quad \cdots \quad \frac{x^{Ci}_{j^i_{m^i_k,k}}}{z^{Ci}_{j^i_{m^i_k,k}}} \right]^{\mathrm{T}}_k \tag{5.25a}$$

$$\boldsymbol{g}^i_v(\boldsymbol{w}_k) = \left[\frac{y^{Ci}_{j^i_{1,k}}}{z^{Ci}_{j^i_{1,k}}} \quad \cdots \quad \frac{y^{Ci}_{j^i_{m^i_k,k}}}{z^{Ci}_{j^i_{m^i_k,k}}} \right]^{\mathrm{T}}_k . \tag{5.25b}$$

Finally, also the corresponding observation covariance matrixes $\boldsymbol{R}_{u,k}$ and $\boldsymbol{R}_{u,k}$ have to be dynamically composed, with the right statistics corresponding to the effective available feature points. The effective dimension of these matrices will depend on the number of extracted points and is equal to $(m_k \times m_k)$ at the time k. In particular, under the reasonable assumption of independence of the image measurements between the n cameras, each matrix will be a block diagonal matrix. The i-th block, corresponding to the i-th camera, will be a $(m^i_k \times m^i_k)$ matrix at time k.

An alternative approach to the treatment of the dynamic loss of the object feature points exists, but is less efficient. The idea is to consider always all the object feature points in the equation of the Extended Kalman Filter, and to act only on the observation covariance matrices. When a feature point is lost, the corresponding values of the matrices $\boldsymbol{R}_{u,k}$ and $\boldsymbol{R}_{u,k}$ are increased to very large values. In this way, the effects of these measurements on the pose reconstruction will not be influenced. To consider again the measurements of this point, it will be sufficient to assign the original values to the corresponding elements of the covariance matrixes. It is clear that this method is very simple to implement but is not computationally efficient because it requires also unused measurements to be processed.

5.3 Adaptive Extended Kalman Filter

If a high-quality camera sensor is used, the illumination of the scene is stable, and the velocity of the tracked object is nearly constant; then it is possible to use constant statistical parameters with optimal results. On the other hand, if these conditions are not satisfied, it may be convenient to update in real time the statistical parameters $\{\boldsymbol{q}_h, \boldsymbol{Q}_h, \boldsymbol{r}_{u,h}, \boldsymbol{r}_{v,h}, \boldsymbol{R}_{u,h}, \boldsymbol{R}_{v,h}\}$. This leads to the Adaptive Extended Kalman Filter (AEKF).

It is known that an optimal estimator of the statistical parameters does not exist, but many sub-optimal schemes can be derived. In this section the intuitive approach proposed in [31], and reviewed in [10] and [20], is first redefined for a visual tracking problem based on local features, e.g. corners and holes. Then, the adaptive algorithm is formulated in an iterative, limited memory format.

5.3.1 Adaptive algorithm

The basic hypothesis of the proposed approach is the constant value of the statistical parameters over N sample times.

Since not all the visual features are always available during the motion and their location into the scene is strongly variable, it may be reasonable to assume the statistics of the observation noise to be equal for all the measurements of the feature points in the scene of each camera. Hence the quantities $\{\boldsymbol{r}_{u,h}, \boldsymbol{r}_{v,h}, \boldsymbol{R}_{u,h}, \boldsymbol{R}_{v,h}\}$ are replaced with the quantities

$$\boldsymbol{r}_{u,h} = \begin{bmatrix} r^1_{u,h}\boldsymbol{\imath}_{m^1_k} & \cdots & r^n_{u,h}\boldsymbol{\imath}_{m^n_k} \end{bmatrix}^{\mathrm{T}} \tag{5.26a}$$

$$\boldsymbol{r}_{v,h} = \begin{bmatrix} r^1_{v,h}\boldsymbol{\imath}_{m^1_k} & \cdots & r^n_{v,h}\boldsymbol{\imath}_{m^n_k} \end{bmatrix}^{\mathrm{T}} \tag{5.26b}$$

$$\boldsymbol{R}_{u,h} = \mathrm{diag}\left\{ (\sigma^1_{u,h})^2 \boldsymbol{I}_{m^1_k}, \ldots, (\sigma^n_{u,h})^2 \boldsymbol{I}_{m^n_k} \right\} \tag{5.26c}$$

$$\boldsymbol{R}_{v,h} = \operatorname{diag}\left\{(\sigma_{v,h}^1)^2\boldsymbol{I}_{m_k^1}, \ldots, (\sigma_{v,h}^n)^2\boldsymbol{I}_{m_k^n}\right\}, \tag{5.26d}$$

where $\boldsymbol{\imath}_s$ indicates a $(s \times 1)$ vector of components equal to 1, $\boldsymbol{I}_s$ indicates the $(s \times s)$ identity matrix, and the notation $\operatorname{diag}(\cdot, \ldots, \cdot)$ indicates a (block) diagonal matrix defined by its arguments. Moreover, considering the i-th camera, the samples of the observation noise sequences $\boldsymbol{\nu}_{u,h}^i$ and $\boldsymbol{\nu}_{v,h}^i$ are independent for $h = 1, \ldots, N$ and have a Gaussian distribution with means $r_u^i\boldsymbol{\imath}_{m_h^i}$ and $r_v^i\boldsymbol{\imath}_{m_h^i}$, and variance $(\sigma_u^i)^2\boldsymbol{I}_{m_h^i}$ and $(\sigma_v^i)^2\boldsymbol{I}_{m_h^i}$, respectively, where the parameters r_u^i, r_v^i, σ_u^i and σ_v^i are constant over N sample times, for $i = 1, \ldots, n$.

In view of the nonlinear relation (5.14), an intuitive approximation of the observation noise sample vectors at time h is given by the quantities

$$\boldsymbol{\rho}_{u,h} = \begin{bmatrix}\boldsymbol{\rho}_{u,h}^1 & \cdots & \boldsymbol{\rho}_{u,h}^n\end{bmatrix}^{\mathrm{T}} = \boldsymbol{\zeta}_{u,h} - \boldsymbol{g}_u(\boldsymbol{w}_{h,h-1}) \tag{5.27a}$$

$$\boldsymbol{\rho}_{v,h} = \begin{bmatrix}\boldsymbol{\rho}_{v,h}^1 & \cdots & \boldsymbol{\rho}_{v,h}^n\end{bmatrix}^{\mathrm{T}} = \boldsymbol{\zeta}_{v,h} - \boldsymbol{g}_v(\boldsymbol{w}_{h,h-1}) \tag{5.27b}$$

which can be considered as independent and identically distributed over N samples. It can be shown that an unbiased estimator for r_u^i and r_v^i can be taken as

$$\hat{r}_u^i = \frac{1}{N}\sum_{h=1}^{N}\overline{\rho}_{u,h}^i \tag{5.28a}$$

$$\hat{r}_v^i = \frac{1}{N}\sum_{h=1}^{N}\overline{\rho}_{v,h}^i, \tag{5.28b}$$

where $\overline{\rho}_{u,h}^i$ and $\overline{\rho}_{v,h}^i$ are scalar quantities equal to the mean values of the components of the vectors $\boldsymbol{\rho}_{u,h}^i$ and $\boldsymbol{\rho}_{v,h}^i$, respectively, for $i = 1, \ldots, n$. Moreover, an unbiased estimator for $(\sigma_u^i)^2$ and $(\sigma_v^i)^2$ may be obtained as

$$(\hat{\sigma}_u^i)^2 = \frac{1}{N-1}\sum_{h=1}^{N}\frac{1}{m_h^i}\left(\|\boldsymbol{\rho}_{u,h}^i - \hat{r}_u^i\boldsymbol{\imath}_m\|^2 - \frac{N-1}{N}\operatorname{tr}\left(\boldsymbol{\Gamma}_{u,h}^i\right)\right) \tag{5.29a}$$

$$(\hat{\sigma}_v^i)^2 = \frac{1}{N-1}\sum_{h=1}^{N}\frac{1}{m_h^i}\left(\|\boldsymbol{\rho}_{v,h}^i - \hat{r}_v^i\boldsymbol{\imath}_m\|^2 - \frac{N-1}{N}\operatorname{tr}\left(\boldsymbol{\Gamma}_{v,h}^i\right)\right). \tag{5.29b}$$

where the function $\mathrm{tr}(\cdot)$ is the trace of the input matrix, and the matrixes $\boldsymbol{\Gamma}^i_{u,h}$ and $\boldsymbol{\Gamma}^i_{v,h}$ are the diagonal block of the matrixes $\boldsymbol{\Gamma}^i_{u,h}$ and $\boldsymbol{\Gamma}_{v,h}$ corresponding to the i-th camera, respectively.

For the state noise statistics, in view of the linear dynamic state relation at time h given by (5.11), an intuitive approximation for the state noise vector at time h is

$$\boldsymbol{\varrho}_h = \boldsymbol{w}_h - \boldsymbol{A}\boldsymbol{w}_{h,h-1}, \tag{5.30}$$

which may be considered independent and identically distributed over N samples. As before, it can be shown that an unbiased estimator for the mean value $\boldsymbol{q}$ of the state noise may be obtained as

$$\hat{\boldsymbol{q}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\varrho}_i, \tag{5.31}$$

while an unbiased estimator for the covariance matrix $\boldsymbol{Q}$ is given by

$$\hat{\boldsymbol{Q}} = \frac{1}{N-1}\sum_{i=1}^{N}\left((\boldsymbol{\varrho}_i - \hat{\boldsymbol{q}})(\boldsymbol{\varrho}_i - \hat{\boldsymbol{q}})^{\mathrm{T}} - \frac{N-1}{N}\boldsymbol{\Delta}_i\right), \tag{5.32}$$

where $\boldsymbol{\Delta}_i = \boldsymbol{A}\boldsymbol{P}_{i,i-1}\boldsymbol{A}^{\mathrm{T}} - \boldsymbol{P}_{i,i}$.

In sum, Equations (5.27)–(5.32) provide a heuristic unbiased estimator for the statistical parameters of an EKF used for visual tracking, on the assumption that the last N samples are statistically independent and identically distributed.

5.3.2 Iterative formulation

Using the previous results, an iterative limited memory formulation of the AEKF may be designed. The required prior knowledge information is represented by an initial estimate of the quantities $\boldsymbol{w}_{1,0}$, $\boldsymbol{P}_{1,0}$, $\hat{\boldsymbol{q}}_0$, $\hat{\boldsymbol{Q}}_0$, and of $\hat{r}^i_{u,0}$, $\hat{r}^i_{v,0}$, $\hat{\sigma}^i_{u,0}$ and $\hat{\sigma}^i_{v,0}$, for $i = 1, \ldots, n$.

Without loss of generality, it is assumed that the observation noise statistical parameters are constant over N_r time samples while the state noise statistical

parameters are constant over N_q time samples.

The first step of the iterative algorithm is the linearization of the output Kalman model around the last predicted value of the state $\boldsymbol{w}_{k,k-1}$, according to (5.20), and the computation of the matrices $\boldsymbol{\Gamma}_{u,k}$ and $\boldsymbol{\Gamma}_{v,k}$ in (5.21).

Starting from time N_r, the second step is the evaluation of the current noise vector as follows

$$\boldsymbol{\rho}_{u,k} = \boldsymbol{\zeta}_{u,k} - \boldsymbol{g}_u(\boldsymbol{w}_{k,k-1}) \tag{5.33a}$$

$$\boldsymbol{\rho}_{v,k} = \boldsymbol{\zeta}_{v,k} - \boldsymbol{g}_v(\boldsymbol{w}_{k,k-1}), \tag{5.33b}$$

and the computation of the estimated observation noise statistics for each camera in the following manner:

$$\hat{r}^i_{u,k} = \hat{r}^i_{u,k-1} + \frac{1}{N_r}(\overline{\rho}^i_{u,k} - \overline{\rho}^i_{u,k-N_r}) \tag{5.34a}$$

$$\hat{r}^i_{v,k} = \hat{r}^i_{v,k-1} + \frac{1}{N_r}(\overline{\rho}^i_{v,k} - \overline{\rho}^i_{v,k-N_r}) \tag{5.34b}$$

$$\begin{aligned}(\hat{\sigma}^i_{u,k})^2 = (\hat{\sigma}^i_{u,k-1})^2 + \frac{1}{m^i_k(N_r-1)}\Bigg(&\|\boldsymbol{\rho}^i_{u,k} - \hat{r}^i_{u,k}\boldsymbol{\imath}_m\|^2 - \|\boldsymbol{\rho}^i_{u,k-N_r} - \hat{r}^i_{u,k}\boldsymbol{\imath}_m\|^2 + \\ &+ \frac{1}{N_r}\|\boldsymbol{\rho}^i_{u,k} - \boldsymbol{\rho}^i_{u,k-N_r}\|^2 + \frac{N_r-1}{N_r}\mathrm{tr}(\boldsymbol{\Gamma}^i_{u,k-N_r} - \boldsymbol{\Gamma}^i_{u,k})\Bigg)\end{aligned} \tag{5.34c}$$

$$\begin{aligned}(\hat{\sigma}^i_{v,k})^2 = (\hat{\sigma}^i_{v,k-1})^2 + \frac{1}{m^i_k(N_r-1)}\Bigg(&\|\boldsymbol{\rho}^i_{v,k} - \hat{r}^i_{v,k}\boldsymbol{\imath}_m\|^2 - \|\boldsymbol{\rho}^i_{v,k-N_r} - \hat{r}^i_{v,k}\boldsymbol{\imath}_m\|^2 + \\ &+ \frac{1}{N_r}\|\boldsymbol{\rho}^i_{v,k} - \boldsymbol{\rho}^i_{v,k-N_r}\|^2 + \frac{N_r-1}{N_r}\mathrm{tr}(\boldsymbol{\Gamma}^i_{v,k-N_r} - \boldsymbol{\Gamma}^i_{v,k})\Bigg),\end{aligned} \tag{5.34d}$$

for $i = 1, \ldots, n$.

The third step consists in the evaluation of the Kalman gains

$$\boldsymbol{K}_{u,k} = \boldsymbol{P}_{k,k-1}\boldsymbol{H}_{u,k}{}^{\mathrm{T}}(\boldsymbol{\Gamma}_{u,k} + \hat{\boldsymbol{R}}_{u,k})^{-1} \tag{5.35a}$$

$$\boldsymbol{K}_{v,k} = \boldsymbol{P}_{k,k-1}\boldsymbol{H}_{v,k}{}^{\mathrm{T}}(\boldsymbol{\Gamma}_{v,k} + \hat{\boldsymbol{R}}_{v,k})^{-1}, \tag{5.35b}$$

while the fourth step is the state correction on the basis of the current measurements

$$\boldsymbol{w}_{k,k} = \boldsymbol{w}_{k,k-1} + \begin{bmatrix} \boldsymbol{K}_{u,k} & \boldsymbol{K}_{v,k} \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}_{u,k} - \hat{\boldsymbol{r}}_{u,k} \\ \boldsymbol{\rho}_{v,k} - \hat{\boldsymbol{r}}_{v,k} \end{bmatrix} \tag{5.36a}$$

$$\boldsymbol{P}_{k,k} = \boldsymbol{P}_{k,k-1} - \begin{bmatrix} \boldsymbol{K}_{u,k} & \boldsymbol{K}_{v,k} \end{bmatrix} \begin{bmatrix} \boldsymbol{H}_{u,k} \\ \boldsymbol{H}_{v,k} \end{bmatrix} \boldsymbol{P}_{k,k-1}. \tag{5.36b}$$

Starting from time N_q, the fifth step is the evaluation of the current state noise vector and the computation of the estimated state noise statistics as follows

$$\boldsymbol{\varrho}_k = \boldsymbol{w}_{k,k} - \boldsymbol{A}\boldsymbol{w}_{k,k-1} \tag{5.37a}$$

$$\boldsymbol{\Delta}_k = \boldsymbol{A}\boldsymbol{P}_{k,k-1}\boldsymbol{A}^{\mathrm{T}} - \boldsymbol{P}_{k,k} \tag{5.37b}$$

$$\hat{\boldsymbol{q}}_k = \hat{\boldsymbol{q}}_{k-1} + \frac{1}{N_q}(\boldsymbol{\varrho}_k - \boldsymbol{\varrho}_{k-N_q}) \tag{5.37c}$$

$$\begin{aligned} \hat{\boldsymbol{Q}}_k = \hat{\boldsymbol{Q}}_{k-1} + \frac{1}{N_q - 1}\Bigg\{ & (\boldsymbol{\varrho}_k - \hat{\boldsymbol{q}}_k)(\boldsymbol{\varrho}_k - \hat{\boldsymbol{q}}_k)^{\mathrm{T}} - (\boldsymbol{\varrho}_{k-N_q} - \hat{\boldsymbol{q}}_k)(\boldsymbol{\varrho}_{k-N_q} - \hat{\boldsymbol{q}}_k)^{\mathrm{T}} + \\ & + \frac{1}{N_q}(\boldsymbol{\varrho}_k - \boldsymbol{\varrho}_{k-N_q})(\boldsymbol{\varrho}_k - \boldsymbol{\varrho}_{k-N_q})^{\mathrm{T}} + \frac{N_q - 1}{N_q}(\boldsymbol{\Delta}_{k-N_q} - \boldsymbol{\Delta}_k) \Bigg\}. \end{aligned} \tag{5.37d}$$

The sixth and last step consists in the evaluation of the predicted state for the next sample time

$$\boldsymbol{w}_{k+1,k} = \boldsymbol{A}\boldsymbol{w}_{k,k} + \hat{\boldsymbol{q}}_k \tag{5.38a}$$

$$\boldsymbol{P}_{k+1,k} = \boldsymbol{A}\boldsymbol{P}_{k,k}\boldsymbol{A}^{\mathrm{T}} + \hat{\boldsymbol{Q}}_k. \tag{5.38b}$$

Notice that the update of the noise statistics starts from time N_r for the observation noise and from time N_q for the state noise. Before such times, those quantities are constant and equal to the initial values.

CHAPTER 6

REDUNDANCY MANAGEMENT

For many kind of applications the use of a system of two or more fixed cameras, suitably located with respect to the robot workspace, is the unique solution which can be practised to guarantee an optimal and robust extraction of visual information. However, the use of a multiple-camera system requires the adoption of intelligent and efficient strategies for the management of the available highly *redundant information*.

In this chapter, an *automatic selection algorithm* for managing redundant visual information is presented, which is based on a *pre-selection algorithm* and an *optimal selection algorithm*.

6.1 Automatic selection algorithm

A vision system for robotic applications is based on eye-in-hand cameras, i.e., one or two cameras mounted on the robot end-effector, and/or a system of multiple fixed cameras. In some cases, the use of eye-in-hand cameras may become problematic and inefficient, e.g., in the presence of occlusions, for the execution of assembly/disassembly tasks in restricted space, or when a very long tool is used. In this case the use of a system of two or more fixed cameras, suitably located with respect to the robot workspace, may guarantee an optimal and robust extraction of visual

information.

On the other hand, the use of a multiple-camera system requires the adoption of intelligent and efficient strategies for the management of highly *redundant information*. This task has to be realized in real time and thus the extraction and interpretation of all the available visual information is not possible. Efficient algorithms must be devised which are able to improve the accuracy and robustness of the visual system by exploiting a minimal set of significant information suitably selected from the initial redundant set.

The accuracy of the pose reconstruction process provided by the presented Kalman Filter depends on the number and on the quality of the available feature points. Inclusion of extra points can improve the estimation accuracy but will increase the computational cost of Kalman filtering and especially that of the feature extraction process. On the other hand, it has been shown that a number of five or six feature points, with suitable properties depending on the geometry of the feature points projections on the image plane of each camera, can ensure the best achievable accuracy for a given camera system [43], [45]. To guarantee the existence of such points for any object pose with respect the cameras, the feature points of the object should be numerous and uniformly distributed or the cameras should be well positioned.

To save computational time, *automatic selection algorithms* should be devised to find the optimal subset of image feature points, which is able to guarantee a pose reconstruction close to the optimal one [8], [9], [18]. It should be pointed out, however, that the complexity of the selection algorithms grows at factorial rate. Hence, in case of objects with a large number of feature points, it is crucial to perform a pre-selection of the points, e.g., by eliminating all the points that are occluded with respect to the camera or that are not extracted with good probability of success [39], [13], [6].

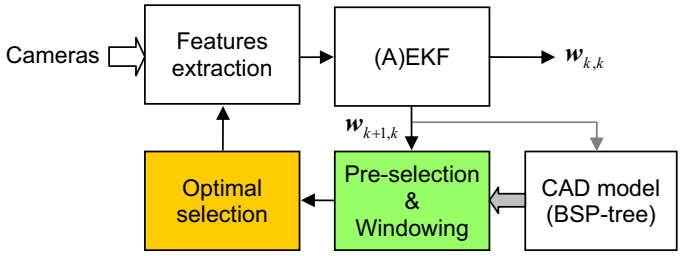

Figure 6.1. Redundancy management. Closed-loop algorithm for the selection of the optimal subset of feature points to consider in the extraction process

In this thesis, an optimal technique for managing redundant visual information is presented, which is based on a *pre-selection algorithm* and an *optimal selection algorithm* [25], [26], and [28]. In particular, in order to reduce computational time, a new pre-selection algorithm of the feature points is proposed, based on the selection of all the points that are visible to each camera and well localizable in the corresponding image plane at a given sample time. This algorithm exhibits a complexity which grows linearly, thanks to the use of the BSP tree for object geometric representation. In detail, shown in Fig. 6.1, the prediction of the object pose provided by the Kalman Filter is used to drive a visit algorithm of the BSP tree which allows identifying all the feature points that are visible at the next sample time for each camera. Moreover, for each visible point, a *dynamic windowing* algorithm is used to verify the feasibility of the extraction of the image feature (*well-localizable* points), and eventually to choose the corresponding optimal extraction window. After the pre-selection, an optimal selection algorithm is adopted to find an optimal subset of feature points to be processed and input to the Kalman Filter. In particular, the selection algorithm exploits the redundancy of the vision system to achieve the best tracking accuracy.

This two-step procedure allows a sensible reduction of the time spent for the whole selection process. In fact, using this approach, the image area to be elaborated may be limited to a constant value, independently at the number of employed cameras, providing an efficient and flexible structure for real-time applications based on multi-

camera systems.

6.2 Pre-selection algorithm

The pre-selection technique is aimed at selecting all the feature points that are visible from each camera, using a geometric model of the observed objects based on BSP tree structures. This technique has to be applied independently to each camera.

When a BSP tree representation of the observed object is available, it is possible to select the feature points of the object that can be visible from any camera position and orientation by implementing a suitable *visit algorithm* of the tree (see Section 3.2.2).

For this specific application, only the image feature points of the object are of concern. Then, a simplified BSP tree model, e.g. based on corners, may be developed, where the data stored into the nodes of the tree are the object surfaces represented by its corners. In fact, if manmade objects are considered, each object planar surface is well described by the corner positions of its contour, which may be represented as an oriented polygonal[18]. In order to build the tree, each object has to be modelled as a set of planar polygons; this means that the curved surfaces have to be approximated. Each polygon is characterized by a set of feature points (the vertices of the polygon) and by the vector normal to the plane leaving from the object. For each node of the tree, a partition plane, characterized by its normal vector and by a point, is chosen according to a specific criterion; the node is defined as the set containing the partition plane and all the polygons lying on it.

Each node is the root of two subtrees: the *front* subtree corresponding to the

[18]An oriented polygonal is a couple composed of a polygonal and a unit vector orthogonal to the plane of the polygonal and outgoing with respect to the internal part of the object

subset of all the polygons lying entirely on the front side of the partition plane (i.e., the side corresponding to the half-space containing the normal vector), and the *back* subtree corresponding to the subset of all the polygons lying entirely on the back side of the partition plane. The construction procedure can be applied recursively to the two subsets by choosing, for each node, a new partition plane among those corresponding to the polygons contained in that subtree. The construction ends when all the polygons are placed in a node of the tree. Further details on BSP trees and an example of construction can be found in the Section 3.2.2.

6.2.1 Recursive visit algorithm

Consider first the case of a single object. The proposed visit algorithm can be applied recursively to all the nodes of the tree, starting from the root node as shown in Fig. 6.2. Initially, the feature points of the surfaces represented by the root node are used to compose the initial set of visible points. After that, the recursive visit algorithm is applied by updating the current set of visible feature points as follows.

For the current node, during the recursive process, classify the camera position with respect to the current partition plane: "Front side", "Back side", "On the plane". Hence, the following operations have to be realized for each specific case:

- **Front side:** Visit the back subtree; process the node; visit the front subtree.
- **Back side:** Visit the front subtree; process the node; visit the back subtree.
- **On the plane:** Visit the front subtree; visit the back subtree.

When the algorithm processes a node, the current set of projections of the visible feature points on the image plane is updated by adding all the projections of the feature points of the polygons of the current node and eliminating all the projections of the feature points that are hidden by the projections of the polygons of the current

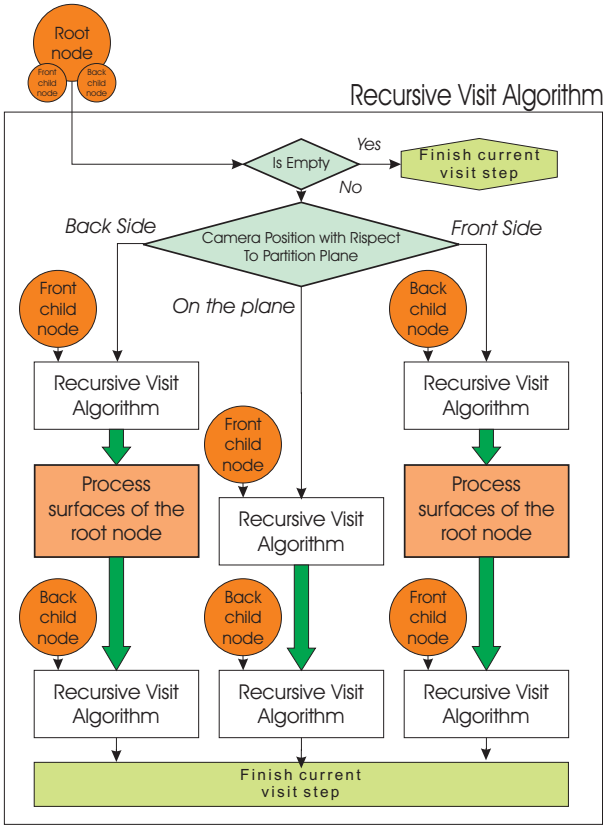

Figure 6.2. Pre-selection algorithm. Block diagram of the recursive visit algorithm

node. If a polygon is hidden from the camera (i.e., the angle between the normal vector to the polygon and the camera z-axis is not in the interval $]-\pi/2, \pi/2[$ or the polygon is behind the camera), the corresponding feature points are not added to the set.

At the end of the visit, the current set will contain all the projections of the feature points visible from the camera, while all the hidden feature points will be discarded. Notice that the visit algorithm updates the set by ordering the polygons with respect to the camera from the background to the foreground (see Section 3.2.2). Moreover, the kind of elaboration is very simple and the corresponding data structures are based only on points, then the computational efficiency is very high.

Consider now the multi-object case. Two main solutions may be employed to evaluate the visible feature points. The first one may be used in those cases when the objects does not have interposing parts, i.e. when it always exists a plane that divides the two objects without intersections. In all these cases, the object has to be ordered from the background to the foreground of the scene, with respect to the distance from the camera point of view. Then, the recursive visit algorithm is applied to each object of this sequence, always working on the current set of visible feature points. At the beginning of the visit of a new object of the sequence, the initial set will contain the visible points of all the previous objects. When the process ends, the set of all the visible points of each object will be available.

In particular, the object ordering may be realized either via a *z-buffering* or making an *ordered BSP tree* structure. The z-buffering simply orders the sequence of objects with respect to the distance of a specific point of the object (e.g. the geometric center) from the camera. Notice that the position of all the objects is provided by the Kalman Filter and it varies at each simple time. If an ordered BSP tree structure is built in real time, a first visit algorithm has to be implemented only to order the object in the right sequence and then the recursive visit algorithm can be applied to evaluate the visibility.

The second solution may be applied in any condition, without limits on the type of observed objects. In this case, a unique BSP tree structure has to be evaluated at each simple time, to represent simultaneously all the objects in the current relative poses, as predicted by the Kalman filter. Then, the recursive visit algorithm is applied directly on this structure to evaluate the visible set of feature points of all the observed objects. This technique is characterized by a greater computational complexity; however, some efficient techniques have been developed to build the unique BSP tree starting from the single BSP tree structures that can be computed off line.

6.2.2 Windowing test

The windowing test is performed in order to recognize all the feature points that are well localizable, i.e., whose positions can be effectively measured with a given accuracy. The recursive visit algorithm presented above recognizes all the feature points that are visible from a camera view point. However, this does not ensure that all the visible points are well localizable. For instance, some points could be out of the field of view of the camera, or they could be too close each other to guarantee absence of ambiguity in the localization.

A *windowing test* can be set up to select all the projections of the feature points that can be well localized. The measurements of the coordinates of the projections of the feature points are obtained by considering suitable rectangular windows of the image plane to be grabbed and processed (see Section 4.3). Each window must contain one feature point to realize a local feature extraction. The windows are centered on the positions of the feature points on the image plane so as predicted by the Kalman Filter, and they are sized as it has been described in Section 4.3.2.

In particular, all the points that are out of the field of view of the camera, or too close to the boundaries of the image plane, are discarded. This is achieved by eliminating all the points whose projections, so as predicted by the Kalman filter, are out of a central window of the image plane. The central window is obtained by reducing the height (base) of the whole image plane of the quantity W_r^{min} (W_c^{min}) from each side, as shown in Fig. 4.14. Then, all the feature points that are too close to each other are discarded, with respect to the chosen clearance factor. All the remaining points are defined well localizable. Finally, the effective dimensions of the corresponding windows are dynamically adapted to the maximum allowable semi-dimension, so as to guarantee an assigned security distance from the other points and from the boundaries of the image plane.

6.3 Optimal selection algorithm

The number of well-localizable feature points after the pre-selection is typically too high with respect to the minimum number sufficient to achieve the best Kalman filter precision. It has been demonstrated that an optimal set of five or six feature points guarantees about the same precision as that of the case when a higher number of feature points is considered. Therefore an automatic selection algorithm to choose an optimal subset of feature points is required to reduce the computational complexity. Moreover, if the number of the selected features is limited, then the area of the image frame to elaborate is fixed, independently of the number of used cameras. This results is very important to achieve robust and flexible multi-camera systems with inexpensive hardware.

6.3.1 Quality indexes

Consider the set of the well-localizable feature points Γ_o, result of the pre-selection and windowing processes, and one of the possible subsets of Γ_o composed by q well-localizable feature points Γ_q ($\dim(\Gamma_q) = q$). Moreover, let $\Gamma^i_{q_i}$ be the subset of q_i feature points of the set Γ_q provided by the i-th camera, for $i = 1, \ldots, n$ ($\dim\left(\Gamma^i_{q_i}\right) = q_i$, and $q = \sum_{i=1}^{n} q_i$).

The optimality of a subset $\Gamma^{opt}_q \subseteq \Gamma_q$ of feature points is valued through the composition of suitably selected *quality indexes* into an optimal cost function. The quality indexes must be able to provide accuracy, robustness and to minimize the oscillations in the pose estimation variables. To achieve this goal it is necessary to ensure an optimal spatial distribution of the projections of the feature points on the image plain, to avoid chattering events between different optimal subsets of feature points chosen during the object motion, and to avoid image features not robust for the extraction process. Moreover, in order to exploit the potentialities of a multi-

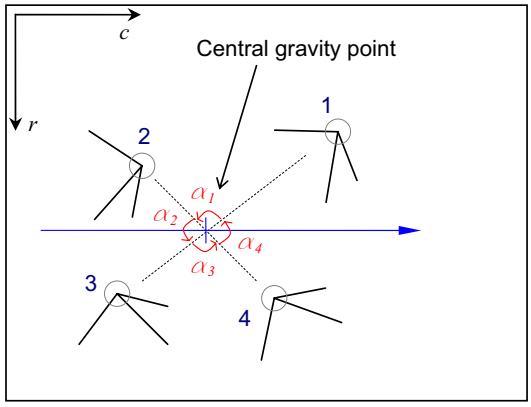

Figure 6.3. Optimal selection algorithm. Angular distribution index: the angles α_k around the central gravity point in the case of four image feature points ($k = 1, \ldots, 4$)

camera system, it is important to achieve an optimal distribution of the feature points among the different cameras. For these reasons, the quality indexes considered in this chapter are of two categories: the first category is based on the evaluation of some characteristics of each subset $\Gamma^i_{q_i}$, while the second category considers some global characteristics of the whole current subset Γ_q.

To achieve accuracy and robustness in both feature extraction and pose estimation, it is convenient to consider separately the features of the subsets $\Gamma^i_{q_i}$ of Γ_q with respect to each camera. The first quality index is a measure of *spatial distribution* of the predicted projections on the image planes of a subset of q_i selected points of the target object for the i-th camera, for $i = 1, \ldots, n$:

$$Q^i_s = \frac{1}{q_i} \sum_{k=1}^{q_i} \min_{\substack{j \in \{1,\ldots,q_i\} \\ j \neq k}} \left\| \boldsymbol{p}_j - \boldsymbol{p}_k \right\| .$$

where $\left\| \boldsymbol{p}_j - \boldsymbol{p}_k \right\|$ denotes the Euclidean distance between the j-th and the k-th point of the considered subset. Notice that a lower threshold on Q^i_s should be adopted to avoid combination of points with inadequate spacial distribution.

To minimize the oscillations in the state prediction error covariance matrix of the extended Kalman filter algorithm and, hence, to improve the quality of pose

estimation, it is desirable that the q_i feature points of $\Gamma^i_{q_i}$ are distributed about the centers of q_i circular sectors of dimension $2\pi/q_i$, for each camera. Therefore, the second quality index is a measure of *angular distribution* of the predicted projections on the image planes of a subset of q_i selected points for the i-th camera, for $i = 1, \ldots, n$:

$$Q^i_a = 1 - \sum_{k=1}^{q_i} \left| \frac{\alpha_k}{2\pi} - \frac{1}{q_i} \right|$$

where α_k is the angle between the vector $\boldsymbol{p}_{k+1} - \boldsymbol{p}_{Ci}$ and the vector $\boldsymbol{p}_k - \boldsymbol{p}_{Ci}$, being $\boldsymbol{p}_{Ci}$ the central gravity point of the whole subset of feature points. The q_i points of the subset are considered in a counter-clockwise ordered sequence with respect to $\boldsymbol{p}_{Ci}$, with $\boldsymbol{p}_{q_i+1} = \boldsymbol{p}_1$ (see Fig. 6.3).

To enhance the efficiency of the feature extraction process of the optimal subset of feature points and, hence, to increase the effective number of available points for the reconstruction, it is desirable to choose only those points with a good percentage of right extraction, with respect to each camera. Therefore, the third quality index measures the current *percentage of success* for the extraction process of a subset of q_i selected points for the i-th camera, for $i = 1, \ldots, n$:

$$Q^i_p = \prod_{k=1}^{q_i} \sigma_k$$

where $0 \leq \sigma_j \leq 1$ is the percentage of success for the extraction of the j-th feature point. These percentages have to be updated during the reconstruction process. At each sampling time and for all the feature points involved in the current extraction process, the percentage of the j-th point, for $j = 1, \ldots, q_i$, is updated by increasing (decreasing) the current value of a quantity $0 < \delta_p < 1$ in the case of a good (failed) extraction. Notice that $q = \sum_{i=1}^{n} q_i$ may be chosen greater or equal to six to handle fault cases during feature extraction.

To avoid frequent changing on the feature set during the servoing, which may cause oscillations in the reconstructed pose, it is desirable to use the same features as

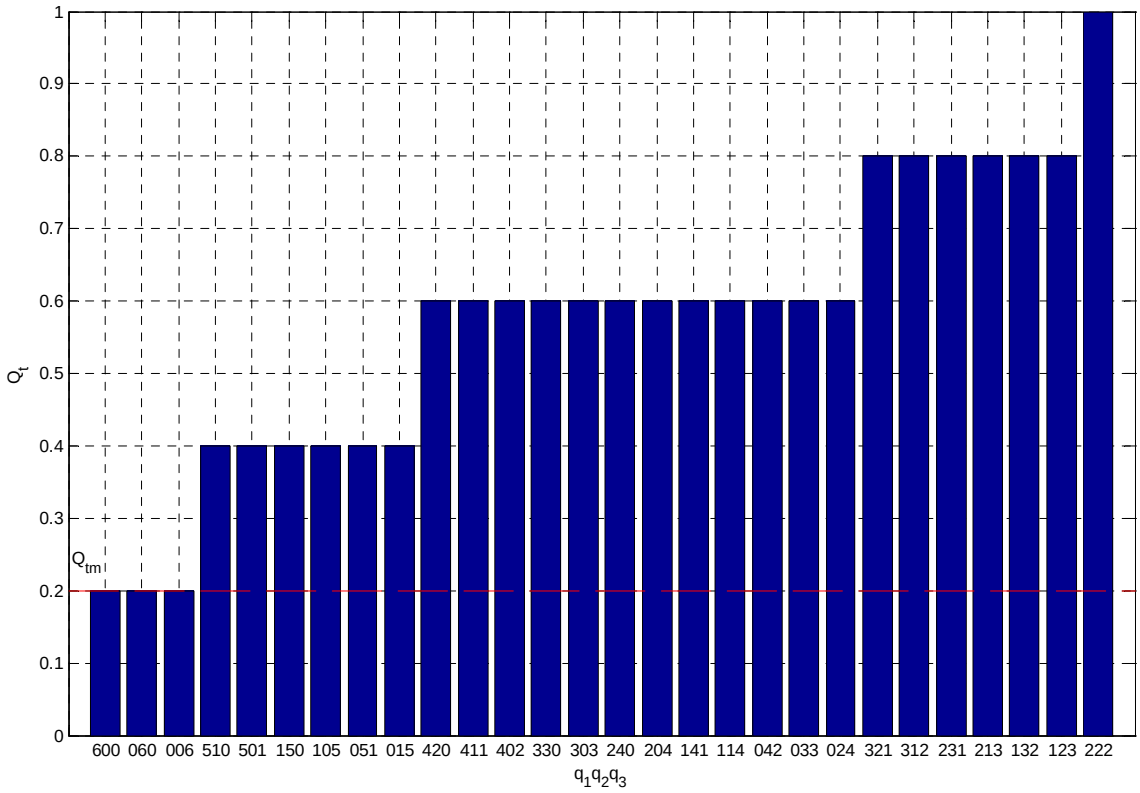

Figure 6.4. Optimal selection algorithm. The repartition index Q_t in the case of six feature points ($q = 6$) and three cameras ($n = 3$), with a minimum value $Q_{tm} = 0.2$

long as possible, depending on the object trajectory. Therefore, the following quality index, which introduces an *hysteresis effect* on the change of the optimal combination of points, is considered:

$$Q_h = \begin{cases} 1+\epsilon & \text{if } \Gamma_q = \Gamma^{opt} \\ 1 & \text{otherwise} \end{cases}$$

where ϵ is a positive constant.

In order to exploit the benefits of triangulation and of different camera positions and resolution, it is desirable to equally distribute the feature points among the cameras. Two effects have to be considered separately: the *triangulation* and the *relative image resolution*.

To increase the benefits of triangulation it is desirable to equally divide the

points among the cameras, independently of their resolution or relative position. To reach this purpose the following repartition index is considered:

$$Q_t = 1 - (1 - Q_{tm}) \frac{n}{2q(n-1)} \sum_{i=1}^{n} \left| q_i - \frac{q}{n} \right|$$

where q_i is the number of points assigned to the i-th camera, and Q_{tm} is the minimum value of the index, for $i = 1, \ldots, n$. In Fig. 6.4 all the possible values of Q_t are reported for a system composed of three cameras, six optimal feature points to be selected, and a minimum value $Q_{tm} = 0.2$.

On the other hand, to take into account the relative camera resolution and the distance of the object from each cameras, a further repartition index has to be considered, that takes into account the distance of the cameras from the object with respect to their effective resolution:

$$Q_r = \frac{1}{q} \max_{k=1,\ldots,n} \left\{ d_k \frac{(s_u s_v)_k}{f_e^k} \right\} \sum_{i=1}^{n} \frac{q_i}{d_i} \frac{f_e^i}{(s_u s_v)_i}$$

where d_i is the distance of the i-th camera form the object, and f_e^i and $(s_u s_v)_i$ are the effective focal length and the product of the row and column scale factors of the i-th camera, respectively, for $i = 1, \ldots, n$. Notice that the values d_i are evaluated using the information provided by the Kalman Filter. Using this index, the selection algorithm becomes capable to manage different resolution zones of different cameras. Therefore, the selection of feature points will be realized also in dependence of the effective resolution of each camera, favoring those cameras that are closer to the object and equipped with the better optical systems and sensors.

The proposed quality indexes represent only some of the possible choices, but guarantee satisfactory performance when used with the pre-selection method and the windowing test presented above. Other examples of quality indexes have been proposed in literature, and some of them can be easily added to the indexes adopted in this work.

6.3.2 Optimal cost function

The quality indexes shown above have to be evaluated for all the possible subsets of q well-localizable feature points, which may be selected from the set $\Gamma_{o,k}$ of all the available well-localizable feature points provided by the n cameras at time k. Therefore, consider the set of all these sets at time k:

$$\Xi_{q,k} = \{\Gamma_{q,k} : \Gamma_{q,k} \subseteq \Gamma_{o,k} \text{ and } \dim(\Gamma_{q,k}) = q\}.$$

The cost function, corresponding to a subset $\Gamma_{q,k}$, can be chosen as

$$\boldsymbol{Q}_{\Gamma_{q,k}} = \frac{1}{q}(Q_{h,k}Q_{t,k}Q_{r,k})_{\Gamma_{q,k}} \sum_{i=1}^{n} q_i \left(Q^i_{s,k}Q^i_{a,k}Q^i_{p,k}\right)_{\Gamma^i_{q_i,k}}$$

where the quality indexes $Q_{h,k}$, $Q_{t,k}$, and $Q_{r,k}$ are evaluated on the whole subset $\Gamma_{q,k}$, while the quality indexes $Q^i_{s,k}$, $Q^i_{a,k}$, and $Q^i_{p,k}$ are evaluated on the subsets $\Gamma^i_{q_i,k} \subseteq \Gamma_{q,k}$ corresponding to the q_i feature points selected for the i-th camera from the set $\Gamma_{q,k}$, for $i = 1, \ldots, n$. The optimal subset $\Gamma^{opt}_{q,k}$ of well-localizable feature points will correspond to the subset of Ξ_q with the minimum value of the cost function

$$\Gamma^{opt}_{q,k} = \arg \min_{\Gamma_{q,k} \in \Xi_q} \boldsymbol{Q}_{\Gamma_{q,k}}.$$

The cost function must be evaluated for all the possible combinations of the visible points of the set on q positions at each sampling time. Then, supposing that l_k is the number of well localizable points of the set $\Gamma_{o,k}$, the number of subsets of $\Xi_{q,k}$ will be

$$\dim(\Xi_{q,k}) = \frac{l_k!}{q!(l_k - q)!}.$$

The factorial dependence may become a problem for real-time applications when l_k is too high, and thus some suboptimal solution should be considered. In fact, in some cases, the number of points resulting at the end of the pre-selection step may be too high to perform the optimal selection in a reasonable time. In such cases, a

computational cheaper solution, based on the optimal set $\Gamma^{opt}_{q,k-1}$ at the previous time step, can be adopted to find a current sub-optimal subset.

The idea proposed here is that of evaluating first off line the initial optimal combination of feature points from the set $\Xi_{q,k}$. Then, only some suitable combinations $\Xi^{sub}_{q,k} \subseteq \Xi_{q,k}$ are tested on line, thus achieving a considerable reduction of processing time (a polynomial complexity instead of a factorial complexity). The subset $\Xi^{sub}_{q,k}$ is the set of combinations that modify at most one point for camera with respect to the current optimal combination

$$\Xi^{sub}_{q,k} = \left\{ \Gamma_{q,k} : \Gamma_{q,k} = \bigcup_{i=1}^{n} \Gamma^{C_i}_{q_i,k} \subseteq \Gamma_{o,k} \text{ and } \sum_{i=1}^{n} \dim \left(\Gamma^{C_i}_{q_i,k} \bigcap \Gamma^{C_i}_{q_i,k-1} \right) \geq q - n \right\}.$$

When the sampling time is sufficiently small, the sub-optimal solution will be very close or coincident to the optimal one.

Notice that, due to the extraction fault cases and to the variations in the visible set of feature points, the previous subset $\Xi^{sub}_{q,k}$ may not exist. In fact, for the cited reasons it may happen that the feature points provided by a camera at times $k-1$ and k may differ in many points, thus making impossible the application of the described simple strategy. In these cases, more complex solutions have to be considered. The best may be that of dynamically relaxing the constraint on the number of acceptable variations in the subset of feature points for these cameras, but only when it is necessary and only for the minimum number of required changes.

CHAPTER 7

VISUAL TRACKING SYSTEM

In this chapter the proposed reconstruction technique for simultaneously visual tracking of more objects using information provided by a multi-camera system is presented. It will be shown how to combine the BSP tree modelling structures, the redundancy management algorithm, the local feature extraction process, and the (adaptive) Extended Kalman Filter to realize a robust and flexible visual tracking system. The main features of the developed system are its intrinsic computational efficiency, the capability to use information provided by a generic number of cameras at a constant extraction cost, the capability to simultaneously reconstruct the poses of multiple objects, and its robustness when working in unstructured environments.

7.1 Overview

The performance of a robotic system during the execution of tasks as grasping, assembling, mating mechanical parts, etc., can be significantly enhanced if visual measurements are available. In fact, visual information can be exploited for planning the end-effector trajectory and, in some cases, can be directly used for visual feedback control. Therefore, a typical problem in robotic vision is the real-time reconstruction of the pose of moving objects of known geometry.

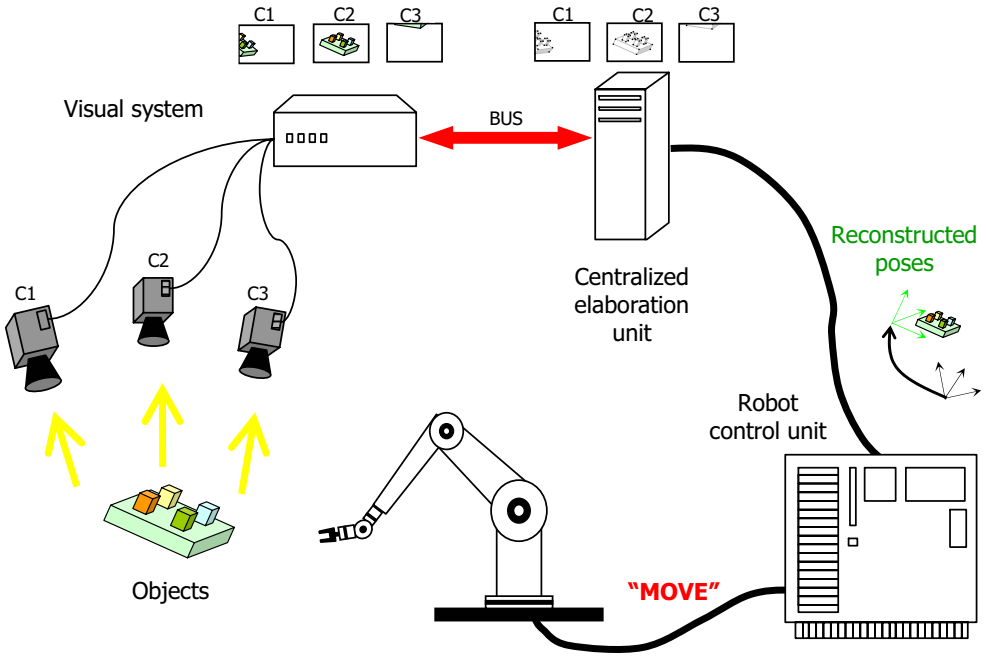

Figure 7.1. Visual tracking system. Typical visual servoing structure with a fixed multi-camera system for robotic applications

In Fig. 7.1 a typical visual servoing structure with a multi-camera system for robotic applications is presented. Depending on the specific application, the visual system may be composed of one, two, or more cameras (fixed in the workspace or on the robot end effector), and of a dedicated elaboration hardware. In the simplest case, only one camera and a frame grabber may be employed to gather visual information on the environment. In more complex cases, instead, a multi-camera system and a dedicated hardware are employed to gather stereo (redundant) visual information and to realize some pre-processing steps on the images, e.g. binarization or edge detection.

In all cases it is necessary to have a *centralized elaboration unit* to complete the image processing, to realize the feature extraction process, and to implement the pose reconstruct algorithm. Usually, a common PC equipped with a frame grabber may be suitable to the purpose, while in more complex situations, in presence of strong real-time constraints, dedicated image processing boards and a multi-processor centralized

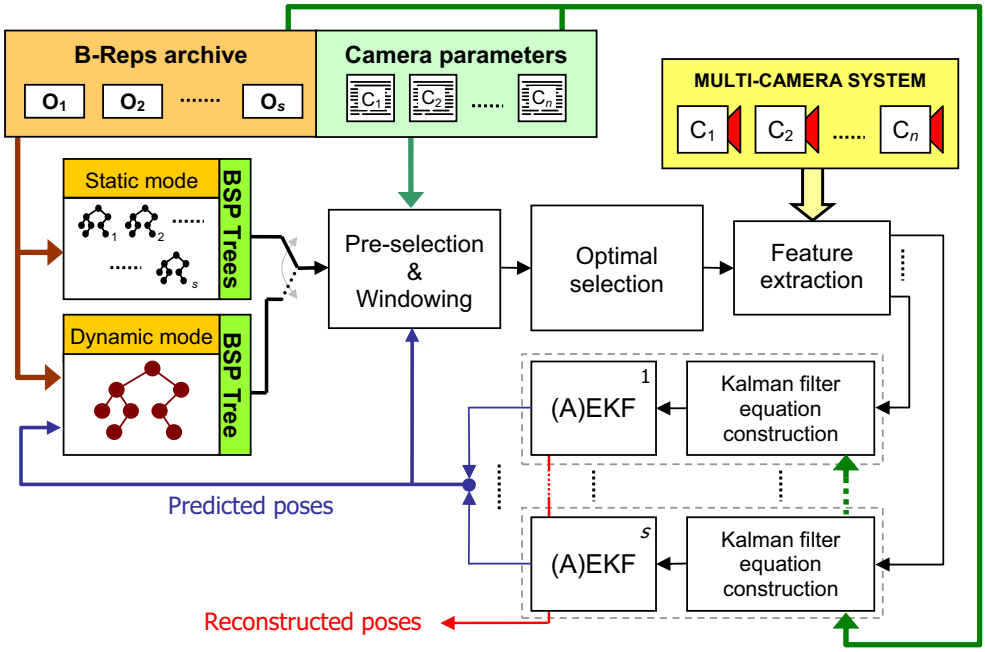

Figure 7.2. Visual tracking system. Overview of the entire visual tracking system

elaboration unit may become indispensable.

For robotic applications, once the pose reconstruction is available, it may be possible to employ this information directly into the *robot control unit* (see Fig. 7.1) to realize a real-time motion planning, a grasp task, an assembly/disassembly task, and so forth. Moreover, if the observed workspace is an entire robotic cell, then the visual system may be also used to coordinate cooperative tasks, or simply to avoid collisions.

To present the developed multi-camera and multi-object visual tracking system, an overview of the entire system will be summarized here, postponing to the next sections the task of presenting in depth each functional module of the reconstruction process, in the light of the material presented in the previous chapters. The block scheme of the developed visual tracking system is shown in Fig. 7.2. It is composed of five functional modules:

1. the system data,
2. the BSP tree management system,
3. the redundancy management system,
4. the feature extraction algorithm,
5. the (adaptive) Extended Kalman Filter algorithm.

The system data holds the prior indispensable knowledge of the system, e.g. the B-Reps archive and the camera calibration parameters. The BSP tree management system is devoted to the construction and to the management in real time of the BSP tree structures used by the pre-selection algorithm. The redundancy management system is composed of two submodules: one to perform the pre-selection and windowing, and one to perform the selection of the optimal subset of feature point. The local feature extraction algorithm is the collection of the processes which make possible the extraction of the image features (corners) from the selected image windows. Finally, a suitable algorithm implement the (adaptive) Extended Kalman Filter for reconstructing the poses of the observed objects.

7.2 Functional modules

In the following all the visual tracking system components shown in Fig. 7.2 will be analyzed in detail with reference to the arguments presented in the previous chapters.

From Fig. 7.2 it is possible to distinguish the functional blocks belonging to each module. The prior knowledge is held in the two blocks on the top-left corner, with the labels "B-Reps archive" (the single object models have the labels "O_k", with $k = 1, \ldots, s$) and "Camera parameters" (the single camera data have the labels "C_i", with $i = 1, \ldots, n$). The blocks belonging to the BSP tree management system are

those with the labels "Static mode/BSP trees" and "Dynamic mode/BSP tree". The redundancy management system has been implemented with the two blocks with the labels "Pre-selection & Windowing" and "Optimal selection". The feature extraction algorithms are executed by the block called "Feature extraction". Finally, the Kalman filter blocks are composed in a stack, at the bottom-right of the figure, with the labels "(A)EKF $^{(1,\ldots,s)}$" and "Kalman filter equation construction".

7.2.1 Prior knowledge

The necessary prior knowledge for the proposed visual tracking system regards two main aspects: the object and the visual system.

All the objects are of known geometry, or their geometric CAD models are known. This knowledge may be provided using the comfortable boundary representation structures (see Section 3.2.1), which may used to represent the main object feature points, e.g. corners and holes. Typically, those models may simply be found into the data sheets of the objects or via a direct measurement. That information is used by the BSP tree management system to generate off line or on line, in dependence of the selected operation mode (static mode for convex objects or dynamic mode for concave objects), the current BSP trees. Moreover, the position of the object features with respect to their object frames are used for the on-line construction of the equations of the Kalman Filter, depending on the effectively extracted feature points, see (5.8).

The camera calibration parameters are the information on the intrinsic, extrinsic, and distortion camera parameters of each available camera. The intrinsic and extrinsic calibration parameters are used to define the camera models as indicated in Section 3.1, while the distortion camera parameters are used to compensate the distortion effects due to imperfections of the optical system and of the camera as-

sembling, as described by (3.17). Those parameters may be evaluated via a suitable camera calibration algorithm, as shown in [43]. Both the pre-selection algorithm and the Kalman filter make use of these parameters: the former one to estimate the projection of the object features on the image plane of each camera (see Section 6.2.1), and the latter to compose the equations of the output model of the Kalman Filter, see (5.9) and (5.15).

7.2.2 BSP tree management system

The BSP trees used during the pre-selection process may be automatically derived from the corresponding B-Reps, as explained in Section 3.2.2. For the specific multi-object based application, the BSP trees building task can be realized in two different ways. Both the case of object without interposing parts (*convex object set*) and the case of object with interposing parts (*concave object set*) can be treated[19]. In the former case, the BSP tree structures may be managed in a *static mode*, while in the latter case a more complex *dynamic mode* has to be employed. In Fig. 7.3 the functional charts for both these modes are shown, representing the static mode on the left and the dynamic mode on the right.

Static mode. In the case of a convex object set, a static operation mode may be employed to perform the pre-selection process. In this modality a BSP tree is built off-line, before starting the reconstruction process, for each observed object (see the left-side of Fig. 7.3). These BSP trees are stored and not further modified during the elaboration. Then, during the pose reconstruction process and for each camera, these BSP trees are simply ordered in a sequence at each sampling time, according to the estimated distance of the objects from the camera, from the farthest to the

[19]A set of objects is convex (concave) when, for any admissible relative poses of the objects of the set, there exists (does not exist) a plane that divides any couple of objects chosen from the set, without making intersections.

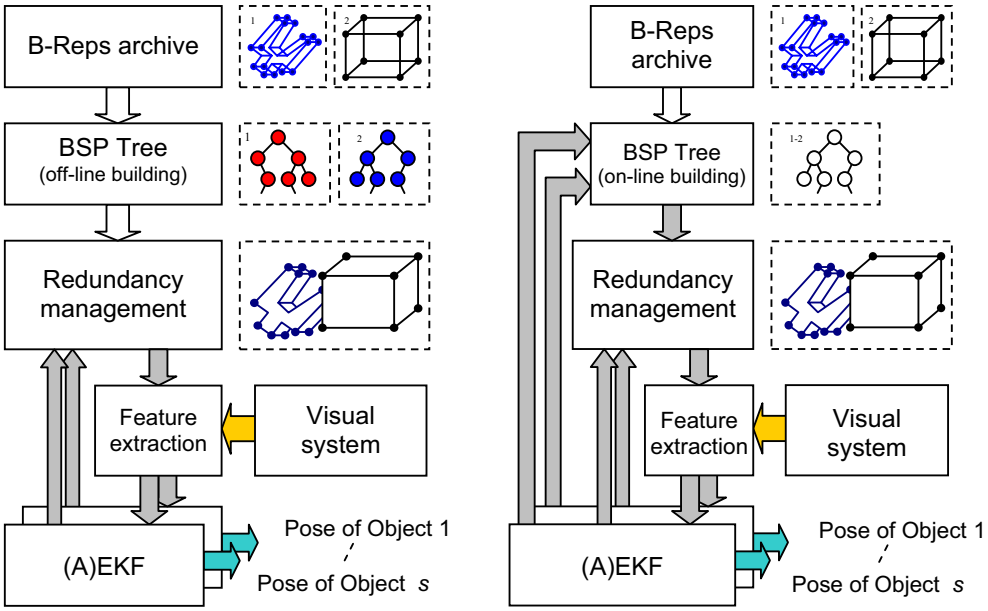

Figure 7.3. Visual tracking system. Block diagrams of the BSP tree management system for the static mode (left) and the dynamic mode (right)

nearest, using suitably techniques, e.g. z-buffering or ordering BSP tree structures (see section 6.2.1). Finally, on this sequence the recursive visit algorithm of the pre-selection process is realized (see Fig. 6.2) to locate all the object feature points which will be visible from each camera.

Dynamic mode. In the case of a concave object set, a dynamic operation mode has to be employed to perform the pre-selection process. In this modality only one BSP tree is built directly on line, during the reconstruction process, to represent all the objects of the set according to the current relative poses, as predicted by the Kalman Filters (see the right-side of Fig. 7.3). This BSP tree simultaneously represents all the objects of the considered set; therefore, for each camera, the recursive visit algorithm of the pre-selection process may be directly applied to estimate the visible feature points of all the objects.

The dynamic mode is the generalization of the static mode, but it requires the on-line building of a complete BSP tree structure that simultaneously represents all

the object, and thus a more powerful centralized elaboration unit may be required.

7.2.3 Redundancy management system

The redundancy management system includes two submodules. The first one performs the pre-selection and the windowing test (see Section 6.2), while the second one performs the optimal subset selection (see Section 6.3).

The pre-selection algorithm makes use of the prediction of the Kalman Filters of the future poses of the observed object, of the calibration parameters of the cameras, and of the BSP tree(s) (with static or dynamic mode) to evaluate all the visible feature points and their projections on the image plane for each camera (see Section 6.2.1). Then, the windowing test is applied to discard all the non-localizable feature points and to dynamic tune the parameters of the potential windows for the image feature extraction process (see Section 6.2.2). The result of these processes, for each camera, is the set of all the well-localizable feature points, their future projections on the image plane, and the corresponding parameters of the feature extraction windows, which will become the input of the optimal feature selection algorithm and of the feature extraction process.

The last step of the process is the selection of the optimal subset of well-localizable feature points, as explained in Section 6.3. The result of this algorithm is the set of the feature extraction windows, for each camera, corresponding to the projection of the feature points of the chosen optimal subset. These windows will be successively elaborated by the feature extraction algorithms to extract the searched image features.

7.2.4 Feature extraction algorithms

The feature extraction algorithms used in the proposed visual tracking system have to extract only one local image feature (a corner or a hole) from the input set of

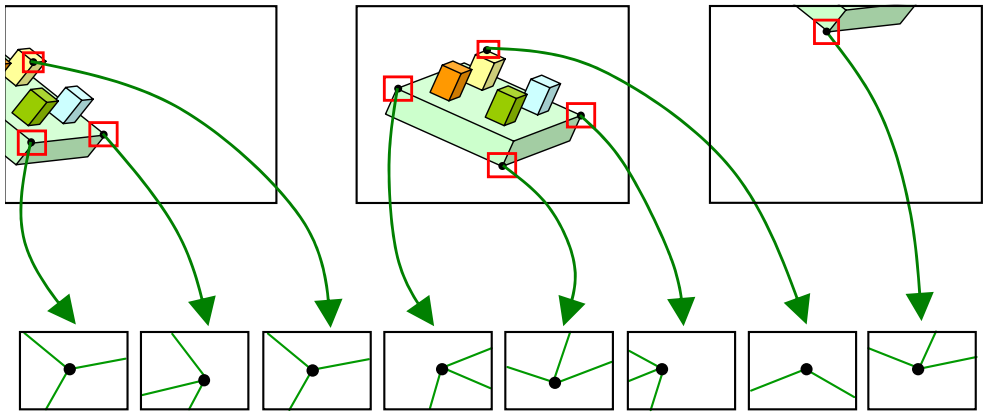

Figure 7.4. Visual tracking system. Example of redundancy management with windowing and feature extraction, in the case of a visual system with three cameras and eight selected feature points

image windows corresponding to the selected set of object feature points. Therefore, the hard problem of the association between the extracted feature points and the object feature points is resolved in an implicit way. In Fig. 7.4 an example of the redundancy management and of the corresponding extraction process of the object feature points, for a case of three cameras and eight selected points, is shown.

The algorithms used to extract the object feature points are those presented in Sections 4.1.2 and 4.2.2. In particular, the Canny edge detector is used to evaluate the edge into each extraction window, while a simplified curvature scale space corner detector is used to extract the corner closer to the center of the each window. Moreover, a statistical test is performed to verify the reliability of the extracted feature points based on a suitable threshold, that is chosen on the base of the standard deviation of the distances of the extracted feature points from the center of the corresponding windows.

7.2.5 Kalman Filter

The (adaptive) Extended Kalman Filter is represented in Fig. 7.2 using two blocks in a stack of s couples (a couple for each tracked object). With reference to a single object,

a first block dynamically builds the equation of the Kalman Filter corresponding to the effectively available measurements provided by the feature extraction algorithms, as explained in Section 5.2.3. The second block, instead, realizes Kalman filtering using the equations described in Sections 5.2 and 5.3.

The output of these blocks are the reconstructed poses of the tracked object at the current sample time and the prediction of the future poses at the next sample time. The latter are used by the BSP tree management system and by the pre-selection algorithm to estimate the visible feature points at the next time.

7.3 Visual tracking process

In this section a summary of the functioning of the presented visual tracking system is presented. For the take of clarity, the case of objects whose parts cannot be interposed (convex object set) and the case of objects with interposing parts (concave object set) are treated separately.

In the first case (see the left-side of Fig. 7.3), it is assumed that a BSP tree representation of each object is built off line from the B-Reps models of the available archive. A different Kalman Filter is required for each object to estimate the corresponding pose with respect to the base frame at the next sample time. The procedure for redundancy management may be summarized as follows:

- **Step 1:** For each camera, all the BSP trees corresponding to the tracked objects are ordered, according to the estimated distance of the objects from the camera, from the farthest to the nearest.
- **Step 2:** For each camera, the recursive visit algorithm is applied to each BSP tree of the sequence to find the set of all the feature points that are visible from the camera. In particular, for each BSP tree of the sequence, a current

set of visible points is updated, by discarding the feature points of the previous objects occluded by the current object and by adding the visible feature points of the current object.

- **Step 3:** For each camera, a dynamic windowing algorithm is applied to discard the feature points not well localizable and to dynamically evaluate the parameters of the image windows to be potentially input to the feature extraction algorithm.

- **Step 4:** For each object, the resulting set of visible points is input to the optimal selection algorithm for the selection of the subset Γ^{opt} of the optimal feature points of the object for the n-camera system.

At this point, all the image windows of the optimal selected points are elaborated using the feature extraction algorithms. The measured coordinates of the points in the image plane, for each object, are input to the corresponding Kalman filter for the dynamical composition of the equation of the output model of the filter, which provides the estimate of the actual object pose and the predicted pose at the next sample time used in Steps 1-2.

In the case of a concave object set, the above procedure may fail because the objects cannot be correctly ordered with respect to the distance from the camera. This problem can be overcome, at expense of computation time, by adopting the solution represented in the functional chart of the right-side of Fig. 7.3. As before, a Kalman filter is used for each object to estimate the corresponding pose with respect to the base frame at the next sample time. Differently form the previous case, a unique BSP tree representing all the objects in the current relative poses is built on line, using the B-Reps models and the estimation provided by the Kalman Filters. Hence, for each camera, the visit algorithm of the tree is applied once to find the set

of all the visible points. Then, Steps 3 and 4 are applied.

Notice that the procedures described above can be applied also to the case of objects moving among obstacles of known geometry; in the case of moving obstacles, the corresponding motion variables can be estimated using Kalman Filters.

CHAPTER 8

EXPERIMENTAL RESULTS

The proposed visual tracking system has been experimentally tested on a robotic system equipped with a stereo visual system. Two fixed cameras, a specialized hardware, and a common personal computer compose the available visual system. Different experimental condition have been considered: the case of one moving object observed by one camera, the case of one moving object observed by two cameras, and the case of two moving objects observed by two cameras. Moreover, the proposed adaptive implementation of the Extended Kalman Filter has also been tested and compared to the non-adaptive implementation. Finally, a visual servoing system has been implemented to realize the visual synchronization of the motion of the two robots: the reconstructed motion of the *master* robot has been used to guide the motion of the *slave* robot.

8.1 Visual tracking

In the following, the results of the experiments realized with the proposed visual tracking system will be presented. First, the employed experimental setup will be described in detail, and then the results of the different case studies will be presented and discussed. In particular, the case of one and two fixed cameras observing one

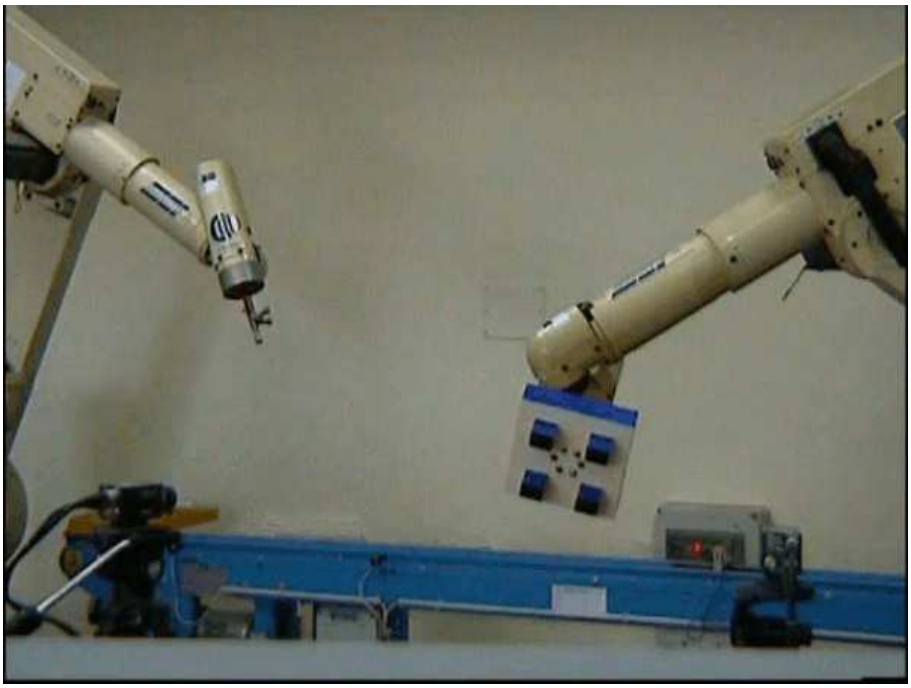

Figure 8.1. Experimental setup composed of two fixed SONY 8500ce cameras, two MATROX GENESIS PCI boards, a host personal computer, two COMAU SMART3-S robots, and a personal computer controlling the robotic cell

moving object are considered, as well as the case of two fixed cameras observing two moving objects.

8.1.1 Experimental setup

The experimental setup composed of a personal computer with Pentium IV 1.7GHz processor equipped with two MATROX GENESIS PCI boards, two SONY 8500ce B/W cameras, and a COMAU SMART3-S robot (Fig. 8.1).

The MATROX boards are used only as frame grabbers and for simple partial image processing (e.g., windows extraction from the image). In fact, the efficiency of the proposed technique does not require the presence of specialized image elaboration boards. The host personal computer, that corresponds to the visual centralized elaboration unit, is used to realize the whole BSP tree structure management, the preselection process, the windowing test, the optimal subset selection algorithm, and the

Camera # 1
$r_o = 187.96$
$c_o = 318.20$
$f_u = -1955.84$
$f_v = 1953.41$
$\boldsymbol{o}_{C_1} = [\ 1.2244 \quad -1.7437 \quad 0.8540\ ]^{\mathrm{T}}$ m
$\boldsymbol{\phi}_{C_1} = [\ -90.234^\circ \quad -1.880^\circ \quad 88.511^\circ\ ]^{\mathrm{T}}$
$\boldsymbol{d}^{C_1} = [\ 0.018 \quad -0.019 \quad -0.024 \quad 0.012 \quad 0.194\ ]^{\mathrm{T}}$
Camera # 2
$r_o = 263.76$
$c_o = 369.64$
$f_u = -1966.35$
$f_v = 1958.10$
$\boldsymbol{o}_{C_2} = [\ 1.6149 \quad -1.6565 \quad 0.8623\ ]^{\mathrm{T}}$ m
$\boldsymbol{\phi}_{C_2} = [\ -92.188^\circ \quad -18.032^\circ \quad -89.111^\circ\ ]^{\mathrm{T}}$
$\boldsymbol{d}^{C_2} = [\ 0.009 \quad 0.0048 \quad -0.0027 \quad -0.0096 \quad 0.1393\ ]^{\mathrm{T}}$

Table 8.1. Camera calibration parameters resulting from the calibration procedure

(adaptive) Extended Kalman Filtering. Simple steps of image processing (e.g. windows extraction) have been parallelized on the MATROX boards and on the personal computer, so as to reduce computational time.

The COMAU robot is merely used to move an object in the visual space of the camera system; thus the object pose with respect to the base frame of the robot can be computed from joint position measurements via the direct kinematic equation.

In order to test the accuracy of the estimation provided by the Kalman Filter, the SONY cameras have been calibrated with respect to the base frame of the robot using a suitable calibration procedure [42], where the robot is exploited to place a calibration pattern in some known pose of the visible space of the camera. The calibration parameters for the two cameras are shown in Table 8.1. The cameras

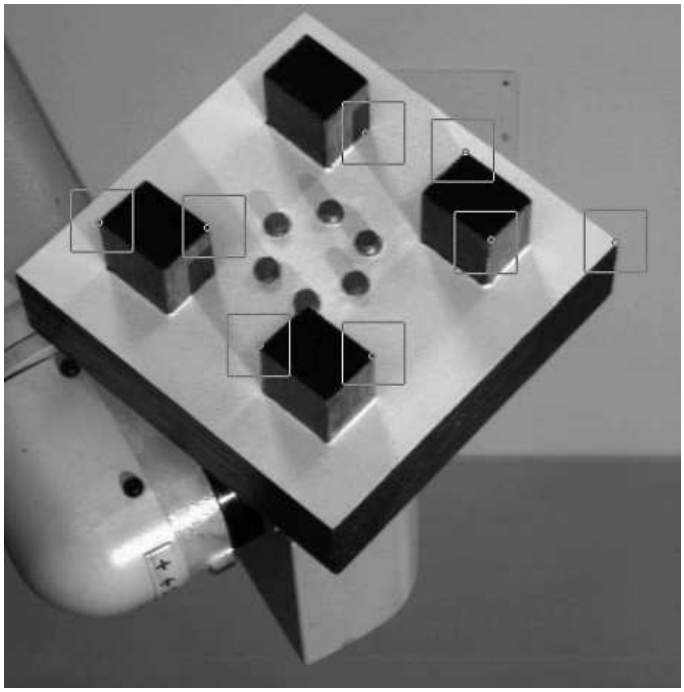

Figure 8.2. Real image seen by the first camera with the corresponding selected windows, which have been used for the feature extraction. Close to the center of each window, the measured position of the corresponding feature point is marked

resolution is 576×763 pixels and the nominal focal length of the lenses is 16 mm. The vector $\boldsymbol{\phi}_{C_i}$ contains the Roll, Pitch and Yaw angles of the i-th camera frame with respect to the base frame corresponding to the matrix $\boldsymbol{R}_{ci}$, while the vector $\boldsymbol{d}^{C_i} = [\ g_1^{C_i} \quad g_2^{C_i} \quad g_3^{C_i} \quad g_4^{C_i} \quad d_1^{C_i}\ ]^{\mathrm{T}}$ contains the parameters used for compensating the distortion effects due to the imperfections of the lens profile and the internal mismatch of the optical system for the i-th camera, as described by (3.17). The estimated value of the *residual mean triangulation error* for the stereo camera system is 1.53 mm. The two cameras are disposed at a distance of about 150 cm from the initial position of the observed object with an angle of about $\pi/6$ between the z_c-axes. No particular illumination equipment has been used, in order to test the robustness of the setup in the case of noisy visual measurements. Notice that the sampling time used for estimation is limited by the camera frame rate, which is about 26 fps.

All the algorithms for BSP structure management, image processing and pose estimation have been implemented in *ANSI C*. The image features are the corners of the object, which can be extracted with high robustness in various environmental conditions. The feature extraction algorithm is based on Canny's method for edge detection and on a custom implementation of the corner detector. In particular, to locate the position of a single corner in the corresponding extraction window, all the straight segments are searched first, using an LSQ interpolator algorithm; then all the intersection points of these segments into the window are evaluated. The intersection points closer than a given threshold are considered as a unique average corner, due to the image noise. In the case of multiple corners detection, all the corners that are at a distance from the center of the window (which corresponds to the position of the corner as predicted by the Kalman filter) greater than the current admissible maximum distance, are considered as fault measurements and are discarded. The maximum distance corresponds to the product of standard deviation with a safety factor of the distance between the measured corner positions and those predicted by the Kalman Filter. In Fig. 8.2 an example of the selected set of extraction windows during the redundancy management process and the results of the image processing algorithms are shown.

The object used in the experiment is shown in Fig. 8.2, as seen from the first camera during the motion, as well as in Fig. 8.1, where the whole experimental setup is presented. The coordinates of the 40 vertices of the object, used as feature points, are reported in Table 8.2.

In the following experiments, constant diagonal covariance matrices $\boldsymbol{Q}$, $\boldsymbol{R}_u$ and $\boldsymbol{R}_v$ have been chosen. The corresponding values of the statistical parameters

#	x^o	y^o	z^o	#	x^o	y^o	z^o
0	0.100	0.100	0.000	**20**	0.070	−0.039	0.092
1	0.100	−0.100	0.000	**21**	0.070	−0.070	0.092
2	−0.100	−0.100	0.000	**22**	0.029	−0.070	0.092
3	−0.100	0.100	0.000	**23**	0.029	−0.039	0.092
4	0.100	0.100	0.051	**24**	−0.029	−0.038	0.051
5	0.100	−0.100	0.051	**25**	−0.029	−0.069	0.051
6	−0.100	−0.100	0.051	**26**	−0.070	−0.070	0.051
7	−0.100	0.100	0.051	**27**	−0.070	−0.039	0.051
8	0.070	0.069	0.051	**28**	−0.029	−0.038	0.092
9	0.070	0.038	0.051	**29**	−0.029	−0.069	0.092
10	0.029	0.038	0.051	**30**	−0.070	−0.070	0.092
11	0.029	0.069	0.051	**31**	−0.070	−0.039	0.092
12	0.070	0.069	0.092	**32**	−0.028	0.069	0.051
13	0.070	0.038	0.092	**33**	−0.028	0.038	0.051
14	0.029	0.038	0.092	**34**	−0.069	0.039	0.051
15	0.029	0.069	0.092	**35**	−0.069	0.069	0.051
16	0.070	−0.039	0.051	**36**	−0.028	0.069	0.092
17	0.070	−0.070	0.051	**37**	−0.028	0.038	0.092
18	0.029	−0.070	0.051	**38**	−0.069	0.039	0.092
19	0.029	−0.039	0.051	**39**	−0.069	0.069	0.092

Table 8.2. Feature points coordinates with respect to the object frame, expressed in meters, used to define the corresponding B-Reps of the object

used for the Extended Kalman Filter are set as follow:

$$\hat{r}^1_{u,0} = \hat{r}^1_{v,0} = \hat{r}^2_{u,0} = \hat{r}^2_{v,0} = 0$$

$$\hat{\sigma}^1_{u,0} = \hat{\sigma}^1_{v,0} = \hat{\sigma}^2_{u,0} = \hat{\sigma}^2_{v,0} = 2$$

$$\hat{\boldsymbol{q}}_0 = \mathbf{0}$$

$$\hat{\boldsymbol{Q}}_0^{(1C)} = \text{diag}\left\{0, 10^{-7}, 0, 10^{-8}, 0, 10^{-7}, 0, 10^{-6}, 0, 10^{-5}, 0, 10^{-6}\right\}$$

$$\hat{\boldsymbol{Q}}_0^{(2C)} = \text{diag}\left\{0, 10^{-7}, 0, 10^{-7}, 0, 10^{-7}, 0, 10^{-5}, 0, 10^{-5}, 0, 10^{-5}\right\},$$

where the physical dimensions are pixel for the mean of the observation noise, and mm, mm/s, rad and rad/s for the components of the mean of the state noise. The matrixes $\hat{\boldsymbol{Q}}_0^{(1C)}$ and $\hat{\boldsymbol{Q}}_0^{(2C)}$ have been used in the case of experiments with one and two cameras, respectively. In fact, the absence of triangulation benefits in the single camera system is also reflected on the different initial values chosen for the elements of the matrix $\hat{\boldsymbol{Q}}_0$. Finally, notice that all the elements of the covariance matrix $\hat{\boldsymbol{Q}}_0$ corresponding to the position components of the state have been considered to be zero.

8.1.2 One object and one camera

Two different experiments have been realized for this case study, that makes use of one camera to observe one moving object. The first experiment reflects a favorable situation where the object moves in the visible space of the camera and most of the feature points that are visible at the initial time remain visible during all the motion. The second experiment reflects an unfortunate situation where the set of the visible points is very variable, and a little portion of the object goes out of the visible space of the camera during the motion.

The time history of the trajectory used for the first experiment is represented in Fig. 8.3. The maximum linear velocity is about 3 cm/s and the maximum angular velocity is about 3 deg/s.

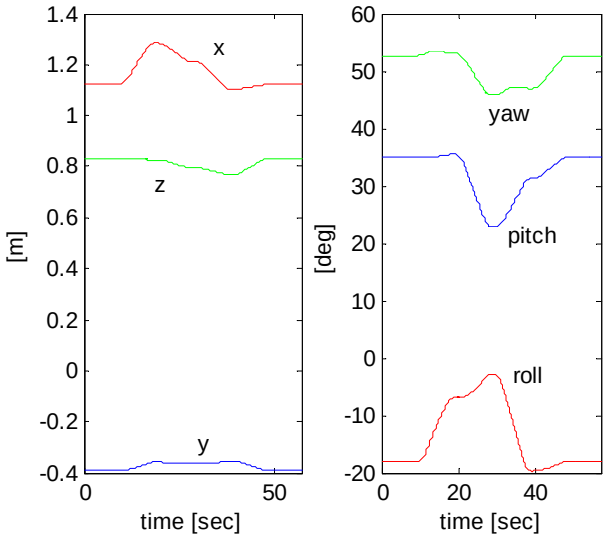

Figure 8.3. First object trajectory with respect to the base frame. Left: position trajectory; right: orientation trajectory

The time history of the estimation errors is shown in the left side of Fig. 8.4. Noticeably, the accuracy of the system reaches the limit allowed by camera's calibration, for almost all the components of the motion. In fact, the calibration errors cause the presence of systematic reconstruction errors, which can be observed by looking at the initial values of the estimation errors. In fact, initially the object is motionless, and thus the corresponding reconstruction errors are due to the camera calibration errors and to the accuracy of the object model. Moreover, as it was expected, the errors for some motion components are larger than others because only 2D information is available in a single camera system and because the reconstructed motion components that are coincident with the object motion components are more affected by estimation errors. In particular, the estimation accuracy is lower along the z_c-axis for the position. Since in the experiment the z_c-axis is almost aligned and opposed to the y axis of the base frame, the estimation errors are larger for the y component of the position. Moreover, the estimation errors for the orientation component about the x_c-axes and the y_c-axis, i.e. the *roll* and *yaw* components, should be larger, but this phenomenon does not occur because the projection of the object shape is well

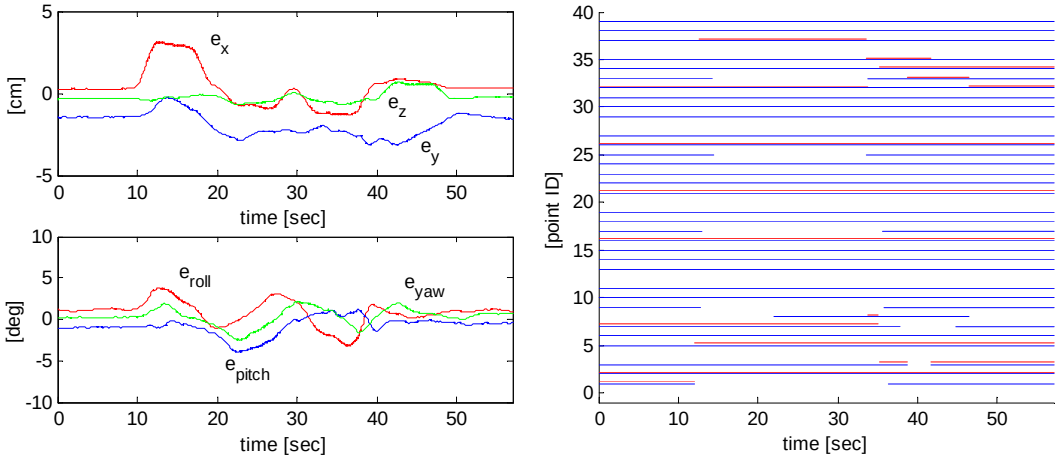

Figure 8.4. Results in the case of one object and one camera with the first object trajectory. Left: time history of the estimation errors (top: position errors, bottom: orientation errors); right: visible points and selected points (for each point, the bottom line indicates when it is visible, the top line indicates when it is selected for feature extraction)

distributed on the image. Finally notice how, during the motion, the object velocity affects the accuracy of the corresponding reconstructed components (e.g. for the x motion component), because of the approximations made for the object motion model used in the Kalman Filter.

On the right side of Fig. 8.4 the output of the whole selection algorithm is reported. For each of the 40 feature points, two horizontal lines are considered: a point of the bottom line indicates that the feature point was classified as visible by the pre-selection algorithm at a particular sample time; a point of the top line indicates that the visible feature point was chosen by the selection algorithm. Notice that 8 feature points are selected at each sample time, in order to guarantee at least five or six measurements in the case of fault of the extraction algorithm for some of the points. Also, some feature points are hidden during all the motion, while others are only visible over some time intervals. Finally, no significant chattering phenomena are present.

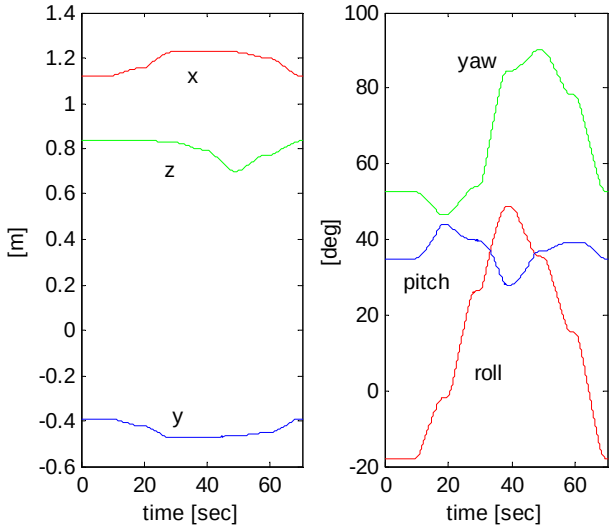

Figure 8.5. Second object trajectory with respect to the base frame. Left: position trajectory.; right: orientation trajectory

The time history of the trajectory used for the second experiment is represented in Fig. 8.5. The maximum linear velocity is about 3 cm/s and the maximum angular velocity is about 7 deg/s.

The time history of the estimation error is shown in Fig. 8.6. It can be observed that the error remains low but is greater than the estimation error of the previous experiment. This is due to the increased velocity of the object motion components and to the fact that from $t = 10$ s to $t = 60$ s the object moves so that it is partially out of the visible space of the camera. This fact penalizes the estimation accuracy and explains how the magnitude of the estimation error components is greater than in the previous experiment, especially for the orientation components due to the increased rotation velocity of the object. The corresponding output of the pre-selection and selection algorithms is reported in Fig. 8.6. It should be pointed out that the pre-selection and selection algorithms are able to provide the optimal set of points independently of the operating condition, although slight chattering phenomena appear in some situation where the elements in the set of localizable points is rapidly changing.

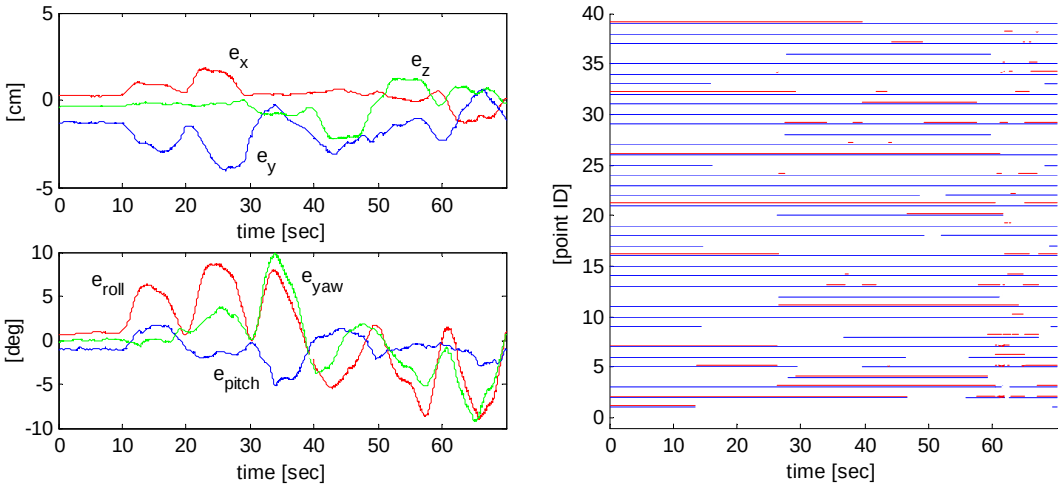

Figure 8.6. Results in the case of one object and one camera with the second object trajectory. Left: time history of the estimation errors (top: position errors, bottom: orientation errors); right: visible points and selected points

8.1.3 One object and two cameras

The trajectory used for the experiment in the case of two cameras is the same represented in Fig. 8.3. The time history of the corresponding estimation errors is shown on the left side of the Fig. 8.7. Noticeably, the accuracy of the system reaches the limit allowed by cameras calibration, for all the components of the motion. As it was expected, the errors for the motion components are of the same order of magnitude, thanks to the use of a stereo camera system.

On the right side of the Fig. 8.7 the output of the whole selection algorithm, for the two cameras, is reported. For each of the 40 feature points, two horizontal lines are considered: a point of the bottom line indicates that the feature point was classified as visible by the pre-selection algorithm at a particular sample time; a point of the top line indicates that the visible feature point was chosen by the selection algorithm. Notice that 8 feature points are selected at each sample time in order to guarantee at least five or six measurements in the case of fault of the extraction algorithm for some of the points. Remarkably, 4 feature points for camera are chosen at each sample

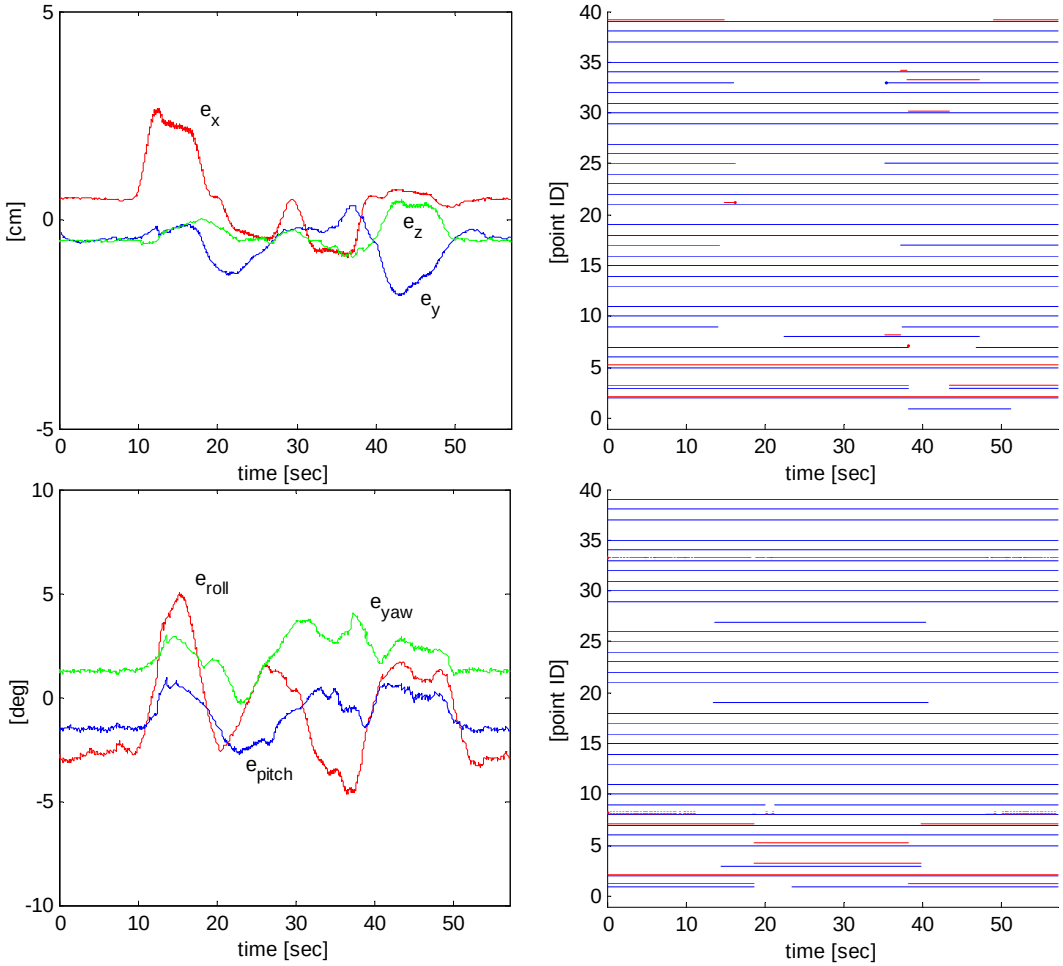

Figure 8.7. Results in the case of one object and two cameras with the first object trajectory. Left: time history of the estimation errors (top: position errors, bottom: orientation errors); right: visible points and selected points for the first (top) and the second (bottom) camera

time, in accordance with the symmetric disposition of the cameras with respect to the object.

8.1.4 Two objects and two cameras

To evaluate the capability of the proposed visual tracking system in coping with multi-object tracking, the object of Fig. 8.1 is moved with the trajectory of the previous experiments, shown in Fig. 8.3, while a second object, that is shown in Fig. 4.2, is placed in a fixed known pose to produce mutual occlusion during the motion. The

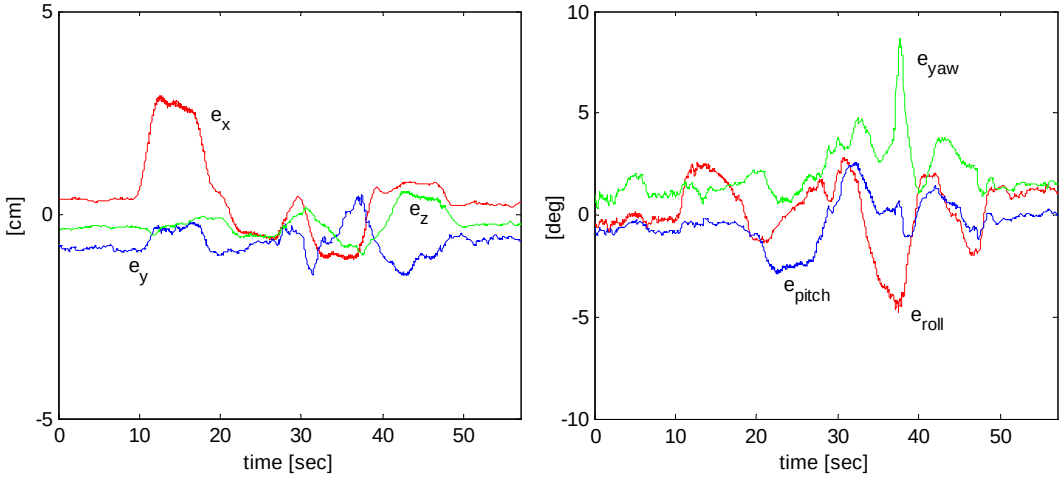

Figure 8.8. Results in the case of two objects and two cameras. Time history of the estimation errors for the motion components of the moving object. Left: position errors; right: orientation errors

fixed object is placed in the pose

$$\boldsymbol{o}_O = \begin{bmatrix} 1.102 & -0.504 & 0.794 \end{bmatrix}^{\mathrm{T}} \mathrm{m}$$

$$\boldsymbol{\phi}_O = \begin{bmatrix} -85.7 & 59.2 & 48.7 \end{bmatrix}^{\mathrm{T}} \mathrm{deg},$$

where it partially and temporarily occludes the moving object during the motion.

The time histories of the estimation errors, reported in Fig. 8.8 for the position and the orientation components, show that the tracking accuracy of the system remains satisfactory also in the presence of occlusions. The occurrence of occlusions is evidenced in Fig. 8.9 where the output of the selection algorithm and the total number of the visible, selected and localized points, for the two cameras, are shown. Notice that the selected feature points are not always equally distributed among the two cameras because, when occlusions occurs, the disposition and the number of the points available for each camera is not always symmetric. Moreover, the light changing condition of the environment causes the faulty of a significant number of feature extractions, due to the image noise, but the system shows a good robustness with respect to both the expected and the unexpected loss of feature points.

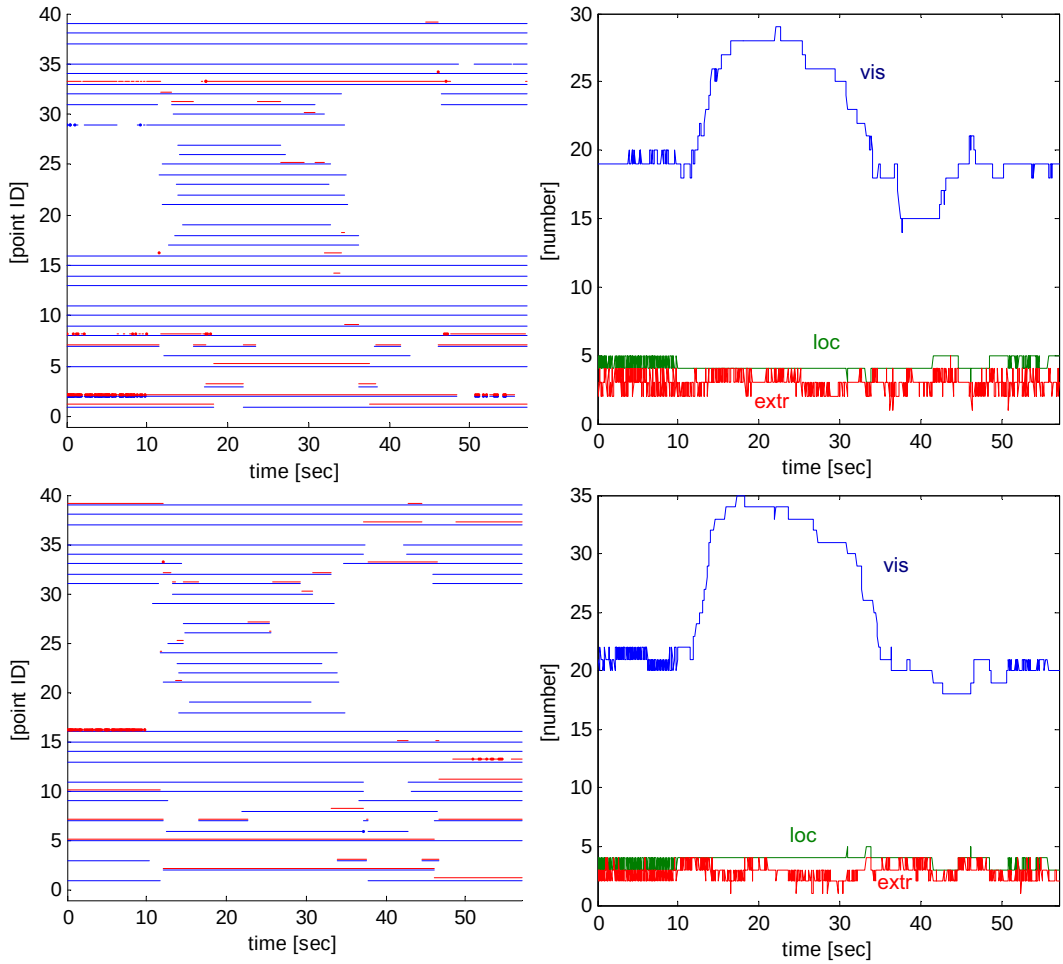

Figure 8.9. Results in the case of two objects and two cameras with the first object trajectory, for the first (top) and the second (bottom) camera. Left: visible points and selected points.; right: the number of visible, localizable and effectively extracted feature points

8.2 Adaptive visual tracking

In this section, the results obtained with the AEKF are compared to the results obtained with a simple EKF, for the case of a single camera system with only one moving object.

In the experiments, a diagonal covariance matrix $\boldsymbol{Q}$ has been chosen, both in the non-adaptive and in the adaptive case. Since a single camera has been used, it is expected that the errors for some components should be larger than others, because

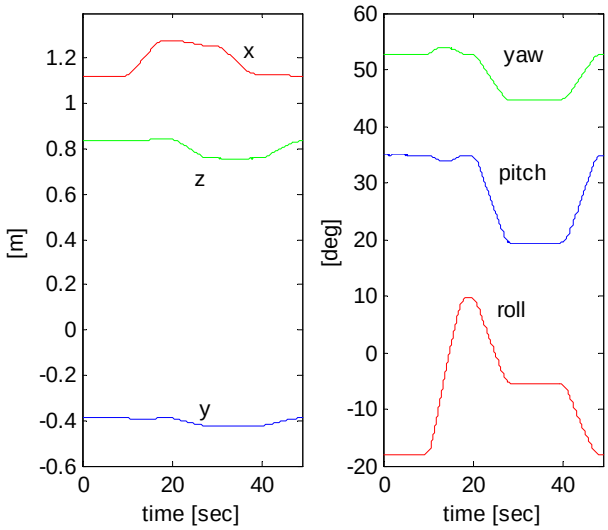

Figure 8.10. Object trajectory with respect to the base frame used for the experiments with the adaptive extended Kalman filter. Left: position trajectory; right: orientation trajectory

only 2D information is available. In particular, the estimation accuracy should be lower along the z_c-axis for the position, and about the x_c-axis and the y_c-axis for the orientation. This asymmetry is also reflected on the different initial values chosen for the elements of the matrix $\hat{\boldsymbol{Q}}_0$. The values of the statistical parameters used for the EKF in the previous experiments are set as initial values for the AEKF. Moreover, the values $N_q = 15$ and $N_r = 15$ have been chosen.

The time history of the object trajectory used for the experiments is represented in Fig. 8.10. The maximum linear velocity and angular velocity for all motion components for this trajectory are about 3.5 cm/s and 6.5 deg/s, respectively.

To prevent some typical implementation problems of Kalman Filters, some of the modifications used in [31] have been adopted.

8.2.1 One object and one camera

The time history of the estimation errors for the case of EKF and for the case of AEKF are shown in Fig. 8.11, while the root-mean-square (RMS) and the standard deviation

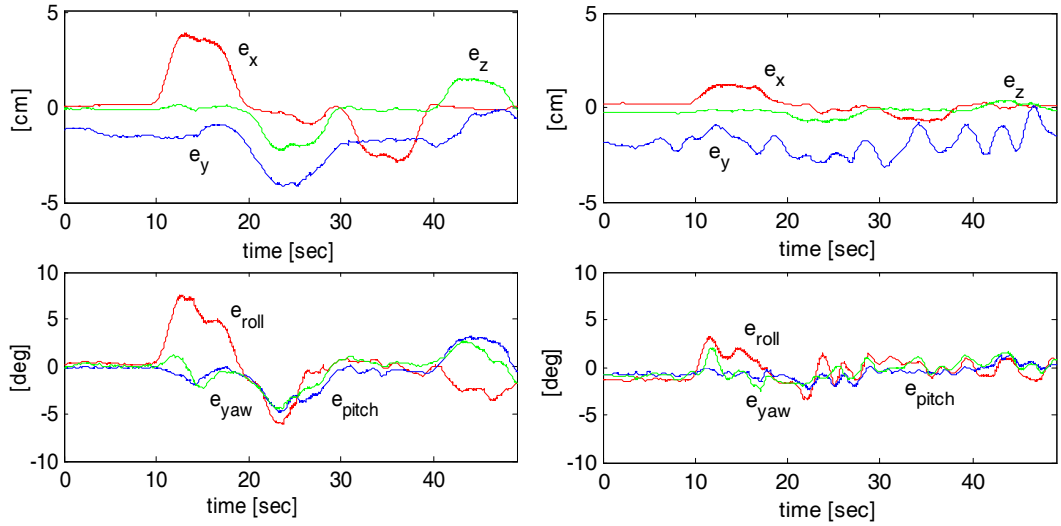

Figure 8.11. Time history of the estimation errors with EKF (left) and AEKF (right)

	EKF		AEKF		Improvement	
Component	RMS	σ	RMS	σ	RMS	σ
e_x	9.34	9.30	6.92	6.79	25.91%	26.99%
e_y	19.36	8.30	20.05	6.59	−3.57%	20.60%
e_z	5.30	5.01	4.15	3.63	21.69%	27.54%
e_φ	1.81	1.81	1.49	1.46	17.66%	19.34%
e_ϑ	1.14	1.03	1.01	0.84	10.99%	18.45%
e_ψ	1.22	1.18	1.05	0.98	14.08%	16.95%

Table 8.3. Comparison of the estimation errors for EKF and AEKF

(σ) of the error components in the two cases are reported in Table 8.3 (the dimensions are mm and deg both for the mean square and for the standard deviation). The results confirm that the adaptive algorithm produces a sensible reduction of the estimation errors. Remarkably, the maximum improvement occurs for those components of the estimation error presenting the highest and the most variable velocity.

The time histories of some of the statistical parameters which are updated on line in the AEKF are also reported in Fig. 8.12. In particular, the time histories of the elements of the (diagonal) covariance matrix of the state noise are shown on the

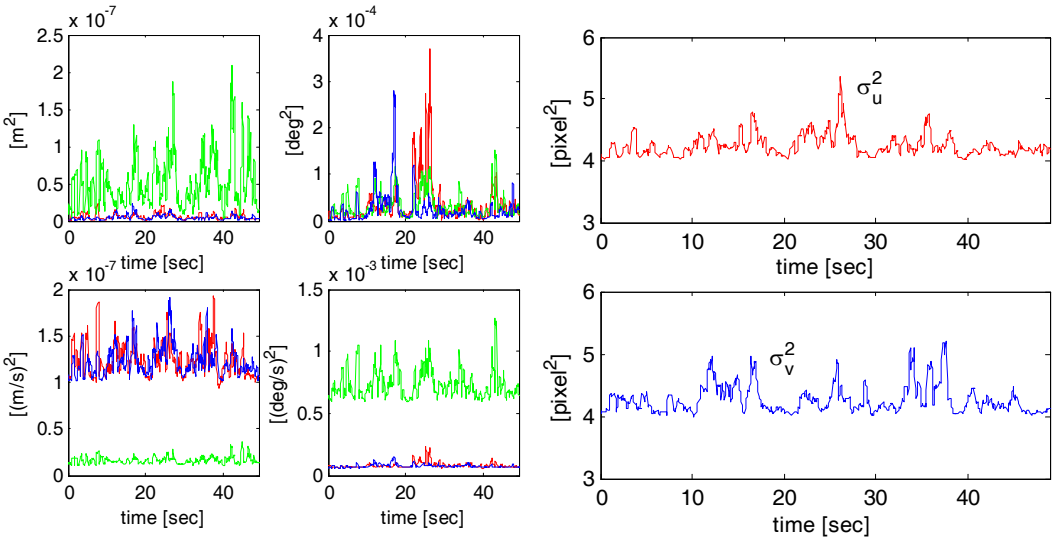

Figure 8.12. Time history of the elements of the state noise covariance matrix (top-left: position; top-middle: orientation; bottom-left: linear velocity; bottom-middle: rotational velocity components) and of the observation noise variance (right)

bottom-left, while the time histories of the variances of the observation noise for the u and v components are shown on the bottom-right. It can be observed that all the updated parameters keep limited values.

8.3 Visual servoing

The proposed visual tracking system has been also employed to realize a visual servoing experiment. The chosen visual task is to realize a synchronized motion between the two COMAU robots, using only the visual information provided by the proposed visual tracking system. In the following, the adopted setup and the corresponding results will be described in detail.

8.3.1 Experimental setup

The setup employed to realize the experiment of visual synchronization between the two COMAU robots is shown in Fig. 8.13. A known object is mounted on the "master" robot, which is guided with a known trajectory during the experiment. As for of the

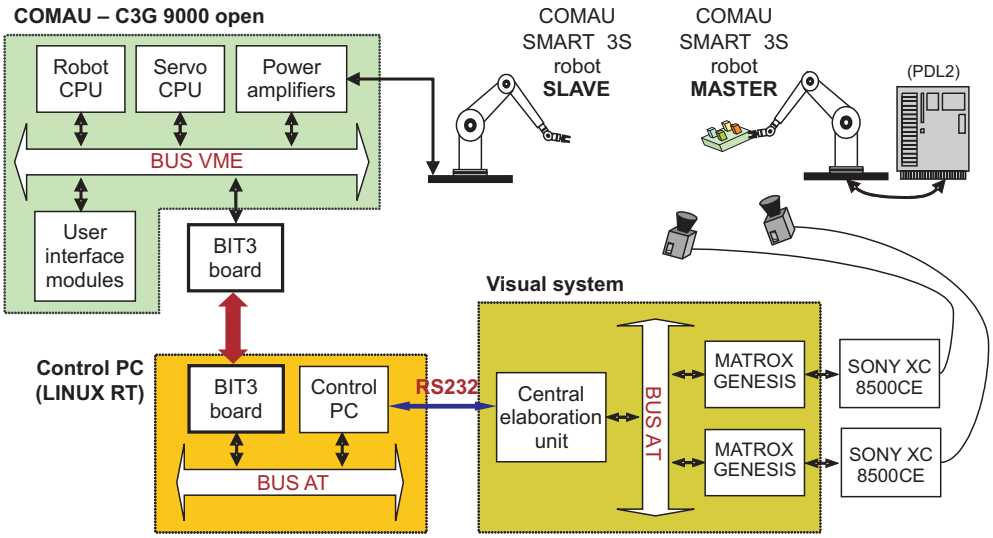

Figure 8.13. Setup employed for the visual servoing experiment

previous experiments, the visual system has used to reconstruct in real time the object pose, i.e. the robot pose.

The main differences with respect to the setup used for the previous visual tracking experiments is the use of a personal computer, that is equipped with a modified version of the LINUX REAL TIME OS, which has been interfaced both with the central elaboration unit of the visual system and with the control unit of the "slave" robot. The reconstructed pose of the master robot is provided by the visual system to the robot control PC, which employs this information to planning the motion of the slave robot in a such way as to realize a synchronized motion between the two robots.

The communication between the control PC and the visual system has been realized using a RS232 serial interface. The control unit of the slave robot employed for the experiment is a COMAU C3G 9000 control unit, which has been modified

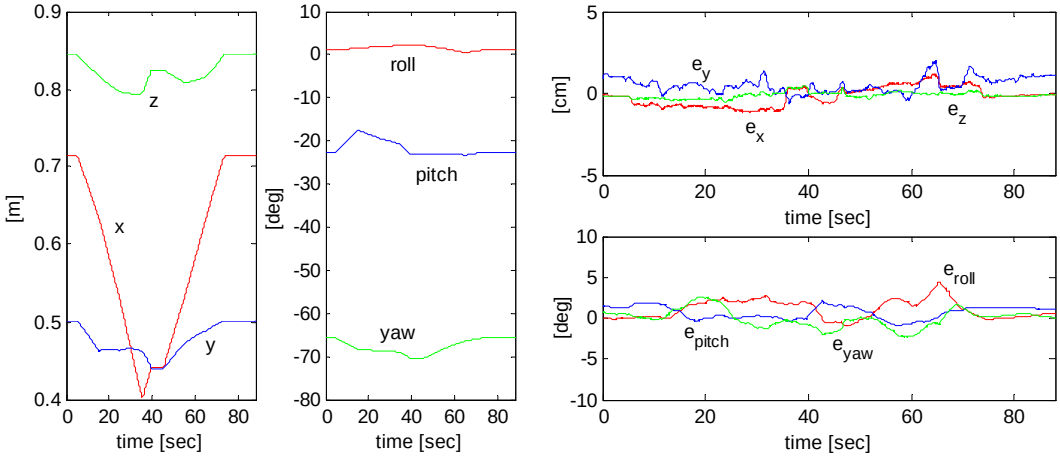

Figure 8.14. Visual servoing experiment. Left: position object trajectory with respect to the base frame; middle: orientation object trajectory with respect to the base frame; top-right: position estimation errors for the reconstructed components of the object pose; bottom-right: orientation estimation errors for the reconstructed components of the object pose

into an "open version" to allow the control of the robot by an external device. The control PC and the C3G 9000 are equipped with two BIT3 boards to make possible the connection between the VME BUS of the C3G 9000 and the AT BUS of the PC.

To increase the observed workspace, the two cameras have been oriented to observe contiguous but different areas. Each camera configuration is managed in a complete automatic manner by the redundancy management process. In particular, the left camera observes the whole object at the start of the experiment, while the right camera does not.

8.3.2 Visual synchronization

The time history of the trajectory of the master robot is represented on the left side of Fig. 8.14. The main motion is realized along the x-axis of the base frame and, in particular, during this motion the object mounted on the master robot goes partially out of the observed space of the first camera and becomes observable by the second camera.

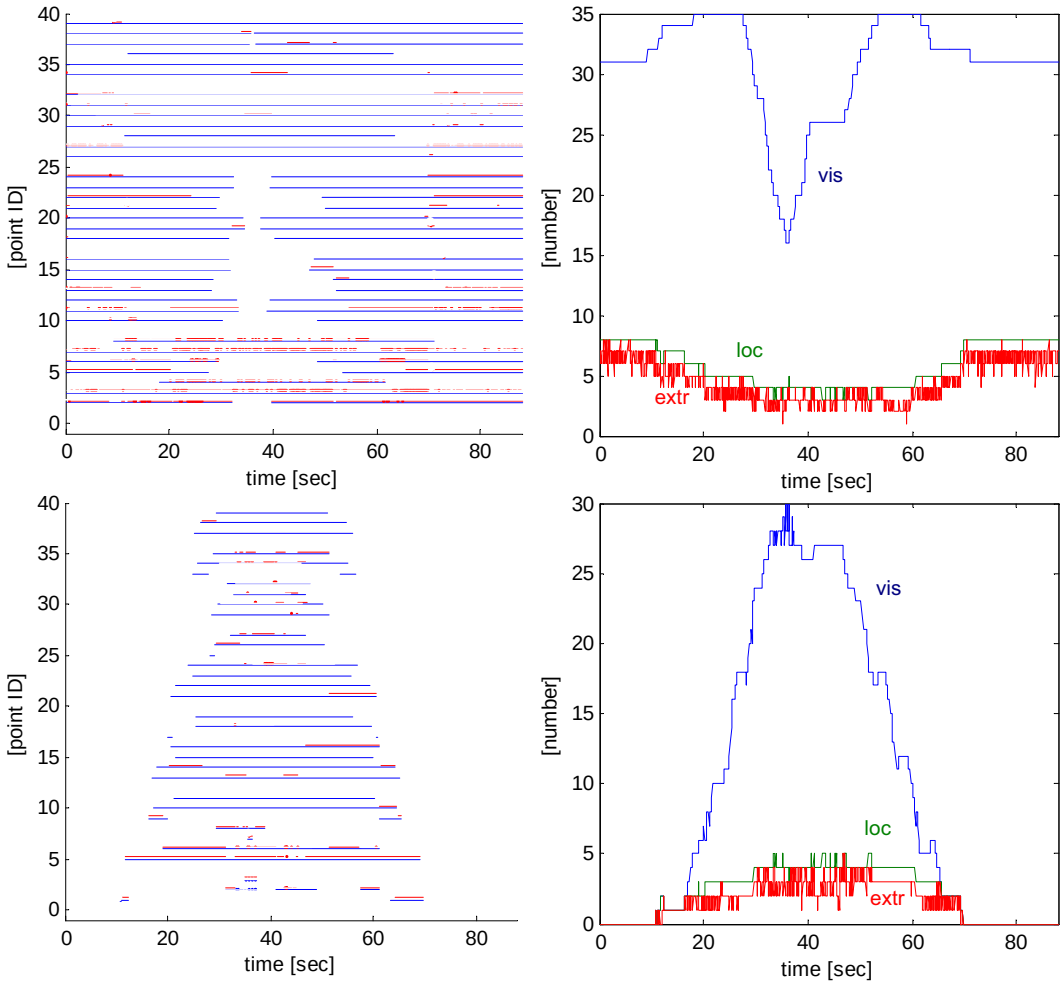

Figure 8.15. Results for the visual servoing experiment, for the first (top) and the second (bottom) camera. Left: visible points and selected points; right: the number of visible, localizable and effectively extracted feature points

The time history of the pose estimation errors are shown on the right side of Fig. 8.14, both for the position and the orientation components, while in Fig. 8.15 the output of the redundancy management process and the total number of the visible, selected and localized points, for the two cameras, are shown. From Fig. 8.15 it is clear that at the start of the motion the right camera does not observe the object. Only from $t = 10$ sec to $t = 70$ sec some parts of the object become observable by the right camera.

It is significant to notice the strong reduction of the pose reconstruction errors

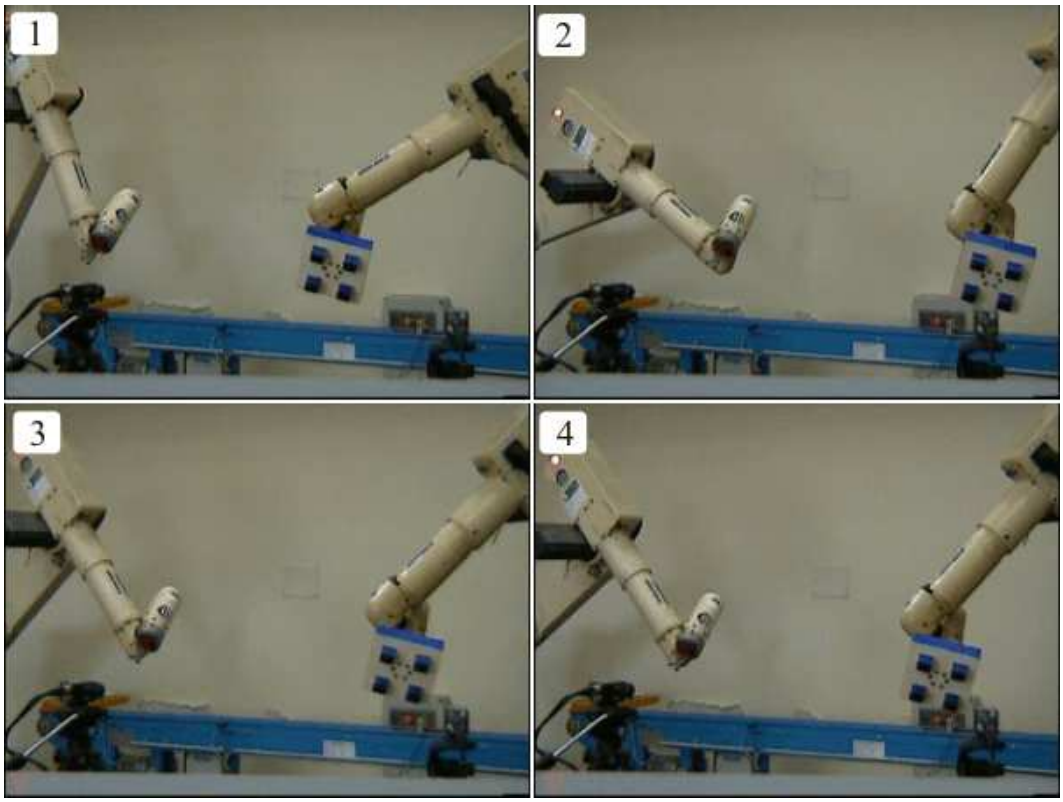

Figure 8.16. Image sequence of the master and slave robots during the experiment of visual synchronization

from $t = 30$ sec to $t = 60$ sec, when the object becomes completely visible by both cameras. In this condition, the benefits of triangulation become important for the reconstruction process.

Another important consideration regards the distribution between the two cameras of the eight points selected for the feature extraction process. As can be observed from Fig. 8.15, the proposed system is able to automatically manage every camera configuration. In fact, the system is able to correctly choose and distribute the feature points between the cameras, thanks to its capacity of estimating the position of each observable feature point on the image plane of each camera. Notice that the selected feature points are not always equally distributed among the two cameras because, according to object visibility, the disposition and the number of

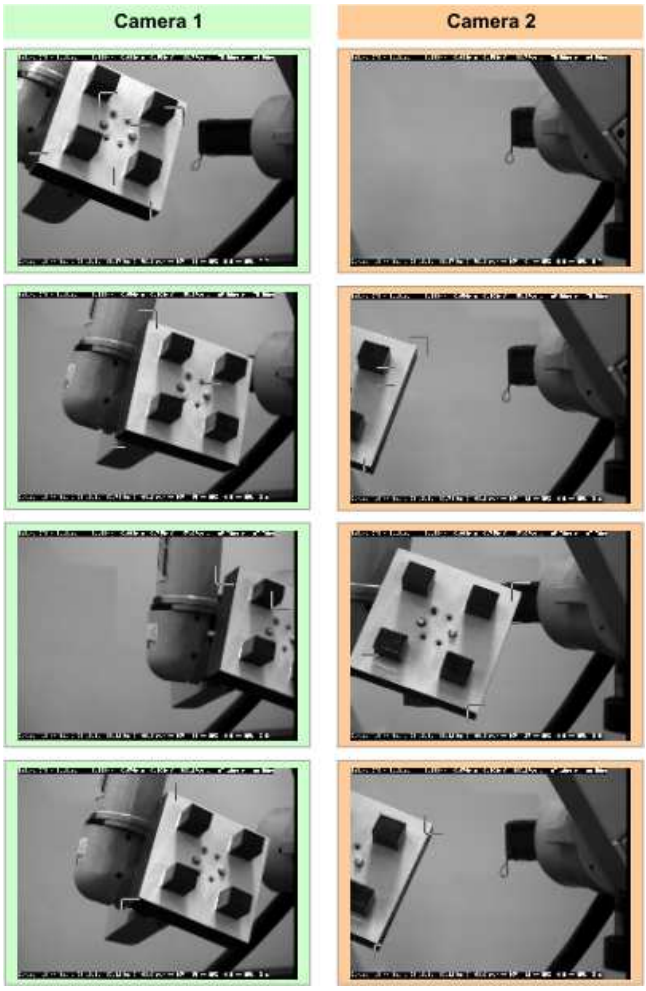

Figure 8.17. Image sequences captured by the two cameras during the experiment of visual synchronization. The rectangles on the images indicate the current image windows, which have been selected during the redundancy management process, while into each window the current extracted corner is marked

the points available for each camera is not always symmetric. Finally, the changing light condition of the environment causes the faulty of a significant number of feature extractions, due to image noise, but the system shows good robustness with respect to both the expected and the unexpected loss of feature points.

In Figs. 8.16 and 8.17 some image shots of the experiment of synchronization are shown. In Fig. 8.16 the two robots during the motion are illustrated, with the master robot on the right side of the image and the slave robot on the left side of the

image. Notice also the orientation and the distances of the cameras with respect to the object trajectory. In Fig. 8.17, instead, the corresponding image shots captured by the two cameras are displayed. In each image, the current image windows selected for the feature extraction process are reported. Moreover, the corresponding extracted corner for each window has been marked.

CHAPTER 9

CONCLUSIONS

The main results and the open problems of the techniques proposed in this thesis are the subject of this chapter. Moreover, a window on some possible future extensions of the approach is presented.

9.1 Main results

The problem of real-time reconstruction of the pose of 3D objects has been considered in this work. The main research result consists of a new visual tracking algorithm for simultaneous dynamic estimation of the absolute pose of a single or multiple objects of known geometry, moving in an unstructured environment, using images provided by a multi-camera system.

It represents a very flexible solution to the realization of a generical visual servoing system. In fact, thanks to its structure based on processing of a limited image area (a set of small image windows selected around an optimal subset of the well-localizable object corners), the algorithm may use a generic number of cameras, in any configuration, without additional computational cost. In fact the processed image area remains limited with respect to the number of employed cameras since it depends only on the desired number of feature points chosen for the feature extraction

process.

This result has been reached thanks to the use of an efficient and innovative redundancy management system of the available image measurements. This system has been realized combining two different modules: the first module realizes a pre-selection which is capable of selecting all the well-localizable image features; the second module realizes a selection of the optimal subset of image features which ensures the best pose reconstruction accuracy.

The proposed technique is able to simultaneously track the pose of many known objects. In fact, thanks to its modularity, simultaneous visual tracking of multiple objects may be realized replicating only the modules devoting to the pose reconstruction, while sharing the modules devoting to the image feature management and extraction. The core of the pose reconstruction process has been based on a suitable dynamic formulation of the Kalman Filter, able to estimate the pose of the observed moving object. Moreover, the filter provides a prediction of the object pose at the next sampling time, which is keenly used by the algorithm for feature selection to increase the accuracy and computational efficiency of the visual tracking system.

In view of its characteristics, this approach may be used both for an eye(s)-in-hand and for an eye(s)-out-hand configuration. In fact, cameras may be positioned in any configuration and the module of image feature management is capable to configure itself in an automatic way. In addition, considering its high computational efficiency, it may be used both in a direct-visual-control scheme and in a dynamic-look-and-move scheme. Finally, it should be pointed out that it is a position-based and calibration-dependent technique, because the estimated 3D absolute pose is model-based, thus requiring the CAD model of the target objects.

The required object knowledge is provided to the system through a simple

and intuitive B-Reps description of the object surfaces. This CAD model is further elaborated by specific modules to achieve a very efficient data representation based on a BSP tree data structure. This structure is the base to the efficient algorithm for managing the image features which supports the feature extraction process. For each camera, this algorithm is able to recognize and select an optimal and minimal set of local image features to be extracted and used by the pose estimation algorithm. On the other hand, the required camera system knowledge is based on an advanced non-linear model, deriving from a standard pin-hole camera model, which considers some of the most important image distortion effects.

The proposed visual tracking system has been experimentally validated on a robotic system equipped with a stereo visual system. Different experimental conditions have been successfully tested: the case of one moving object observed by one camera, the case of one moving object observed by two cameras, and the case of two moving objects observed by two cameras. Moreover, an adaptive implementation of the Extended Kalman Filter has been developed, which has also been successfully tested in a comparison with the non-adaptive implementation. Finally, a visual servoing system has been implemented to realize the visual synchronization of the motion of two robots.

9.2 Open problems

Some aspects of the presented visual tracking technique might be further extended or improved. In particular, the addition of different kinds of image features, which are more robust with respect to the extraction process (e.g. portion of contours and holes), may improve tracking robustness. Such addition does not require any particular modification of the proposed technique as long as the area to elaborate for extraction of these new features is limited, even though under some conditions (e.g. the possibility

of extracting the feature using an image window with a limited extension).

Moreover, the dynamic model of the object motion has been approximated to a first-order model (object velocity is considered constant over each sampling time); therefore the adoption of a more sophisticated model may improve the accuracy of the reconstruction process. Such modification would impact only on the Kalman Filter module, and thus it would not require structural modifications of the proposed approach.

Finally, some implementation aspects may be improved with the adoption of more efficient solutions. For example, the corner extraction algorithm may be replaced with an implementation of the SUSAN algorithm which does not require the edge detection process. However, the effectiveness of all these modifications should be experimentally tested to understand whether or not they represent real improvements.

9.3 Future work

Some aspects of the proposed visual tracking system may be further investigated for future work and research. In particular, the use of the CMOS technology for the construction of camera sensors has opened a new and interesting field of research into the direction of high-rate visual tracking systems. In fact, this kind of technology allows a random access to the two-dimensional pixel array of the sensor in the same way as that of a RAM memory. Therefore, with such kind of sensor it is not necessary to transfer all the images to the frame grabber, but it is possible to transfer only the desired portions of the current image, thus increasing camera rate. The proposed approach is particulary suitable to exploit this characteristics in view of its need to elaborate only a limited portion of the image.

The comparison of the proposed approach with an approach based on the

image Jacobian is an important issue to look at. The results of such comparison may provide important information to directly measure the effectiveness of the proposed approach.

Moreover, the employment of this technique into a visual-force control loop is a fascinating research field, which may provide significant results for application tasks where it interaction with an unknown and unstructured environment is of concern.

Finally, the setup of a hybrid camera configuration with one or more cameras mounted on the robot end-effector and one or more cameras fixed in the workspace represents an important extension of the proposed visual tracking system. Such camera configuration can be managed in an automatic way by the proposed system, providing a significant enhancement of robot task flexibility.

APPENDIX A

BSP TREES

Partitioning a polygon with a plane

Partitioning a polygon with a plane is a matter of determining which side of the plane the polygon belongs to. This is referred to as a front/back test, and is performed by testing each point in the polygon against the plane. If all of the points lie on one side of the plane, then the entire polygon is on that side and does not need to be split. If some points lie on both sides of the plane, then the polygon is split into two or more pieces (Fig. A.1).

The basic algorithm is to loop across all the edges of the polygon and find those for which one vertex (feature point) is on each side of the partition plane. The intersection points of these edges and the plane are computed, and those points are used as new vertices for the resulting pieces.

Classifying a point with respect to a plane is done by plugging the (x, y, z) values of the point into the plane equation, $ax + by + cz + d = 0$. The result of this operation is the distance from the plane to the point along the normal vector to the plane. It will be positive if the point is on the same side of as that of the normal vector, negative otherwise. If the result is 0, the point is on the plane.

For those not familiar with the plane equation, the values a, b, and c are the coordinate values of the normal vector. The value d can be calculated by substituting a point known to be on the plane for x, y, and z.

Convex polygons are generally easier to deal with in BSP Trees construction than concave ones, because splitting them with a plane always results in exactly two convex pieces. Furthermore, the algorithm for splitting convex polygons is straightforward and robust. Splitting of concave polygons, especially self-intersecting ones, is a significant problem in its own right.

Here is an example of pseudo C++ code of a very basic function to split a convex polygon with a plane:

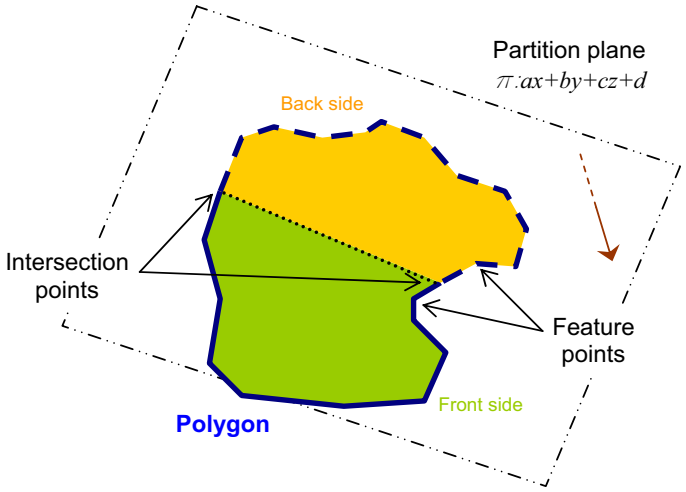

Figure A.1. BSP Trees. Partitioning a polygon with a plane

```
void Split_Polygon (polygon *poly, plane *part, polygon *&front,
polygon *&back) {
    int     count = poly->NumVertices (),
            out_c = 0, in_c = 0;
    point   ptA, ptB,
            outpts[MAXPTS],
            inpts[MAXPTS];
    real    sideA, sideB;
    ptA = poly->Vertex (count - 1);
    sideA = part->Classify_Point (ptA);
    for (short i = -1; ++i < count;) {
        ptB = poly->Vertex (i);
        sideB = part->Classify_Point (ptB);
        if (sideB > 0) {
            if (sideA < 0) {
                // compute the intersection point of the line
```

```
                // from point A to point B with the partition
                // plane. This is a simple ray-plane intersection.
                vector v = ptB - ptA;
                real sect = - part->Classify_Point (ptA) /
                    (part->Normal () | v);
                outpts[out_c++] = inpts[in_c++] = ptA + (v * sect);
            }
            outpts[out_c++] = ptB;
        }
        else if (sideB < 0) {
            if (sideA > 0) {
                // compute the intersection point of the line
                // from point A to point B with the partition
                // plane. This is a simple ray-plane intersection.
                vector v = ptB - ptA;
                real sect = - part->Classify_Point (ptA) /
                    (part->Normal () | v);
                outpts[out_c++] = inpts[in_c++] = ptA + (v * sect);
            }
            inpts[in_c++] = ptB;
        }
        else
            outpts[out_c++] = inpts[in_c++] = ptB;
        ptA = ptB;
        sideA = sideB;
    }
    front = new polygon (outpts, out_c);
```

```
        back = new polygon (inpts, in_c);
    }
}
```

A simple extension to this code that is good for BSP Trees is to combine its functionality with the routine to classify a polygon with respect to a plane. Note that this code is not robust, since numerical stability may cause errors in the classification of a point. The standard solution is to make the plane "thick" by use of an epsilon value.

APPENDIX B

EXTENDED KALMAN FILTER

Jacobian matrix

The computation of the $(m \times 12)$ Jacobian matrices $\boldsymbol{H}_{u,k}$ and $\boldsymbol{H}_{v,k}$ in (5.20) gives

$$\boldsymbol{H}_{u,k} = \left[\frac{\partial \boldsymbol{g}_u}{\partial x_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_u}{\partial y_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_u}{\partial z_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_u}{\partial \varphi_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_u}{\partial \vartheta_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_u}{\partial \psi_O} \;\; \mathbf{0} \right]_k$$

$$\boldsymbol{H}_{v,k} = \left[\frac{\partial \boldsymbol{g}_v}{\partial x_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_v}{\partial y_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_v}{\partial z_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_v}{\partial \varphi_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_v}{\partial \vartheta_O} \;\; \mathbf{0} \;\; \frac{\partial \boldsymbol{g}_v}{\partial \psi_O} \;\; \mathbf{0} \right]_k$$

where $\mathbf{0}$ is a null $(m \times 1)$ vector corresponding to the partial derivatives of $\boldsymbol{g}_u$ and $\boldsymbol{g}_v$ with respect to the velocity variables, which are null because functions $\boldsymbol{g}_u$ and $\boldsymbol{g}_v$ do not depend on the velocity.

Taking into account the expressions of $\boldsymbol{g}_u$ and $\boldsymbol{g}_v$ in (5.15) and (5.16), the non-null elements of the Jacobian matrices $\boldsymbol{H}_{u,k}$ and $\boldsymbol{H}_{v,k}$ have the form:

$$\frac{\partial}{\partial \lambda} \left(\frac{x_j^{Ci}}{z_j^{Ci}} \right) = \left(\frac{\partial\, x_j^{Ci}}{\partial \lambda} z_j^{Ci} - x_j^{Ci} \frac{\partial\, z_j^{Ci}}{\partial \lambda} \right) (z_j^{Ci})^{-2} \tag{B.1a}$$

$$\frac{\partial}{\partial \lambda} \left(\frac{y_j^{Ci}}{z_j^{Ci}} \right) = \left(\frac{\partial\, y_j^{Ci}}{\partial \lambda} z_j^{Ci} - y_j^{Ci} \frac{\partial\, z_j^{Ci}}{\partial \lambda} \right) (z_j^{Ci})^{-2}, \tag{B.1b}$$

respectively, where $\lambda = x_O, y_O, z_O, \varphi_O, \vartheta_O, \psi_O$, and $j = 1, \ldots, m$ and $i = 1, \ldots, n$.

The partial derivatives on the right-hand side of (B.1a) and (B.1b) can be computed as follows.

In view of (5.8), the partial derivatives with respect to the components of the vector

$$\boldsymbol{o}_O = \begin{bmatrix} x_O \\ y_O \\ z_O \end{bmatrix}$$

are the elements of the Jacobian matrix

$$\frac{\partial \boldsymbol{p}_j^{Ci}}{\partial \boldsymbol{o}_O} = \boldsymbol{R}^{Ci}.$$

In order to express in compact form the partial derivatives with respect to the components of the vector

$$\boldsymbol{\phi}_O = \begin{bmatrix} \varphi_O \\ \vartheta_O \\ \psi_O \end{bmatrix},$$

it is useful to consider the following equalities

$$\mathrm{d}\boldsymbol{R}_O(\boldsymbol{\phi}_O) = \boldsymbol{S}(\mathrm{d}\boldsymbol{\omega}_O)\boldsymbol{R}_O(\boldsymbol{\phi}_O) = \boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{S}(\boldsymbol{R}_O^T(\boldsymbol{\phi}_O)\mathrm{d}\boldsymbol{\omega}_O) \tag{B.2a}$$

$$\mathrm{d}\boldsymbol{\omega}_O = \boldsymbol{T}_O(\boldsymbol{\phi}_O)\mathrm{d}\boldsymbol{\phi}_O \tag{B.2b}$$

where $\boldsymbol{S}(\cdot)$ is the skew-symmetric matrix operator, $\boldsymbol{\omega}_O$ is the angular velocity of the object frame with respect to the base frame, and the matrices $\boldsymbol{R}_O$ and $\boldsymbol{T}_O$, in the case of Roll, Pitch and Yaw angles, have the form

$$\begin{aligned} \boldsymbol{R}_O(\boldsymbol{\phi}_O) &= \begin{bmatrix} c_{\varphi_O}c_{\vartheta_O} & c_{\varphi_O}s_{\vartheta_O}s_{\psi_O} - s_{\varphi_O}c_{\psi_O} & c_{\varphi_O}s_{\vartheta_O}c_{\psi_O} + s_{\varphi_O}s_{\psi_O} \\ s_{\varphi_O}c_{\vartheta_O} & s_{\varphi_O}s_{\vartheta_O}s_{\psi_O} + c_{\varphi_O}c_{\psi_O} & s_{\varphi_O}s_{\vartheta_O}c_{\psi_O} - c_{\varphi_O}s_{\psi_O} \\ -s_{\vartheta_O} & c_{\vartheta_O}s_{\psi_O} & c_{\vartheta_O}c_{\psi_O} \end{bmatrix} \\ \boldsymbol{T}_O(\boldsymbol{\phi}_O) &= \begin{bmatrix} 0 & -s_{\varphi_O} & c_{\varphi_O}c_{\vartheta_O} \\ 0 & c_{\varphi_O} & s_{\varphi_O}c_{\vartheta_O} \\ 1 & 0 & -s_{\vartheta_O} \end{bmatrix}, \end{aligned}$$

where $c_\alpha = \cos\alpha$ and $s_\alpha = \sin(\alpha)$, with $\alpha = \varphi_O, \vartheta_O, \psi_O$. By virtue of (B.2a), (B.2b), and the properties of the skew-symmetric matrix operator, the following chain of equalities holds

$$\begin{aligned} \mathrm{d}(\boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O) &= \mathrm{d}(\boldsymbol{R}_O(\boldsymbol{\phi}_O))\boldsymbol{p}_j^O = \boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{S}(\boldsymbol{R}_O^T(\boldsymbol{\phi}_O)\boldsymbol{T}_O(\boldsymbol{\phi}_O)\mathrm{d}\boldsymbol{\phi}_O)\boldsymbol{p}_j^O \\ &= \boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{S}^T(\boldsymbol{p}_j^O)\boldsymbol{R}_O^T(\boldsymbol{\phi}_O)\boldsymbol{T}_O(\boldsymbol{\phi}_O)\mathrm{d}\boldsymbol{\phi}_O \end{aligned}$$

$$= \boldsymbol{S}^T(\boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O)\boldsymbol{T}_O(\boldsymbol{\phi}_O)\mathrm{d}\boldsymbol{\phi}_O,$$

hence

$$\frac{\partial \boldsymbol{R}_O(\boldsymbol{\phi}_O)}{\partial \boldsymbol{\phi}_O}\boldsymbol{p}_j^O = \boldsymbol{S}^T(\boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O)\boldsymbol{T}_O(\boldsymbol{\phi}_O). \tag{B.3}$$

At this point, by virtue of (5.8) and (B.3), the following equality holds

$$\frac{\partial \boldsymbol{p}_j^{Ci}}{\partial \boldsymbol{\phi}_O} = \boldsymbol{R}^{Ci}\frac{\partial \boldsymbol{R}_O(\boldsymbol{\phi}_O)}{\partial \boldsymbol{\phi}_O}\boldsymbol{p}_j^O = \boldsymbol{R}^{Ci}\boldsymbol{S}^T(\boldsymbol{R}_O(\boldsymbol{\phi}_O)\boldsymbol{p}_j^O)\boldsymbol{T}_O(\boldsymbol{\phi}_O),$$

for $j = 1, \ldots, m$ and $i = 1, \ldots, n$.

BIBLIOGRAPHY

[1] K. S. Arun, T. S. Huang, and S. B. Blostein. Least square fitting of two 3D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[2] J. Baeten and J. De Schutter. Hybrid vision/force control at corners in planar robotic-contour following. *IEEE/ASME Transactions on Mechatronics*, 7(2):143–151, 2002.

[3] S. H. Bloomberg. A representation of solid objects for performing Boolean operations. Technical Report 86-006, U.N.C. Computer Science, 1986.

[4] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[5] P. I. Corke. *Visual Control of Robots: High Performance Visual Servoing*. Research Studies Press, Baldock, UK, 1996.

[6] T. W. Drummond and R. Cipolla. Real-time tracking of complex structures with on-line camera calibration. In *Proceedings of the 1999 British Machine Vision Conference*, pp. 574–583, September 1999.

[7] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3):313–326, 1996.

[8] J. T. Feddema, C. S. G. Lee, and O. R. Mitchell. Automatic selection of image features for visual servoing of a robot manipulator. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 832–837, May 1989.

[9] J. T. Feddema, C. S. G. Lee, and O. R. Mitchell. Weighted selection of image features for resolved rate visual feedback control. *IEEE Transactions on Robotics and Automation*, 7(1):31–47, 1991.

[10] A. A. Girgis and W. L. Peterson. Adaptive estimation of power system frequency deviation and its rate of change for calculating sudden power system overloads. *IEEE Transactions on Power Delivery*, 5(2):585–594, 1990.

[11] D. Gordon and S. Chen. Front-to-back display of BSP trees. *IEEE Computer Graphics and Applications*, pp. 79–85, 1991.

[12] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th ALVEY Vision Conference*, pp. 147–151, September 1998.

[13] P. C. Ho and W. Wang. Occlusion culling using minimum occluder set and opacity map. In *Proceedings of the 1999 IEEE International Conference on Information Visualization*, pp. 292–300, July 1999.

[14] R. Horaud, F. Dornaika, and B. Espiau. Visually guided object grasping. *IEEE Transactions on Robotics and Automation*, 14(4):525–532, 1998.

[15] B. K. P. Horn, K. M. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127–1135, 1998.

[16] C. C. W. Hulls and W. J. Wilson. Design of a real-time computer system for relative position robot control. In *Proceedings of the Fourth Euromicro Workshop on Real-Time Systems*, pp. 38–43, June 1990.

[17] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

[18] F. Janabi-Sharifi and W. J. Wilson. Automatic selection of image features for visual servoing. *IEEE Transactions on Robotics and Automation*, 13(6):890–903, 1997.

[19] F. Janabi-Sharifi and W. J. Wilson. Automatic grasp planning for visual-servo controlled robotic maniopulators. *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 28(5):693–711, 1998.

[20] L. Jetto, S. Longhi, and G. Venturini. Development and experimental validation of an adaptive extended Kalman filter for the localization of a mobile robot. *IEEE Transactions on Robotics and Automation*, 5(2):219–229, 1999.

[21] M. Kass, A. Witkin, and D. Terzopoulos. Snake: active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[22] F. Kececi and H. H. Nagel. Machine-vision-based estimation of pose and size parameters from a generic workpiece description. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2159–2164, May 2001.

[23] S. Lee and Y. Kay. An accurate estimation of 3-D position and orientation of a moving object for robot stereo vision: Kalman filter approach. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 414–419, May 1990.

[24] V. Lippiello, B. Siciliano, and L. Villani. Position and orientation estimation based on Kalman filtering of stereo images. In *Proceedings of the 2001 IEEE International Conference on Control Applications*, pp. 702–707, September 2001.

[25] V. Lippiello, B. Siciliano, and L. Villani. 3-D objects motion estimation based on Kalman filter and BSP tree models for robot stereo vision. *Archives of Control Sciences*, 12(2):71–88, 2002.

[26] V. Lippiello, B. Siciliano, and L. Villani. A new method of image features preselection for real-time pose estimation based on Kalman filter. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 1, pp. 372–377, October 2002.

[27] V. Lippiello, B. Siciliano, and L. Villani. Objects motion estimation via BSP tree modeling and Kalman filtering of stereo images. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2968–2973, May 2002.

[28] V. Lippiello and L. Villani. Managing redundant visual measurements for accurate pose tracking. *Robotica*, 21:511–519, 2003.

[29] D. Marr and E. Hildreth. Theory of edge detection. In *Proceedings of the Royal Society London*, vol. B-207, pp. 187–217, 1980.

[30] F. Mokhtarian and R. Suomela. Robust image corner detection through curvature scale space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1376–1381, 1998.

[31] K. A. Myers and B. D. Tapley. Adaptive sequential estimation with unknown noise statistics. *IEEE Transactions on Automatic Control*, 21(4):520–523, 1976.

[32] B. F. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Computer Aided Design*, 22(4):153–162, 1990.

[33] B. F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface*, pp. 201–212, May 1992.

[34] K. Nickels and S. Hutchinson. Weighting observations: The use of kinematic models in object tracking. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pp. 1677–1682, May 1998.

[35] J. N. Pan, Y. Q. Shi, and C. Q. Shu. A Kalman filter in motion analysis from stereo image sequence. In *Proceedings of the 1994 IEEE International Conference on Image Processing*, vol. 3, pp. 63–67, 1994.

[36] M. Paterson and F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete and Computational Geometry*, 5:485–503, 1990.

[37] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. Springer-Verlag, London, UK, 2nd edition, 2000.

[38] S. M. Smith and J. M. Brady. SUSAN: a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[39] K. Tarabanis, R. Y. Tsai, and A. Kaul. Computing occlusion-free viewpoints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):279–292, 1996.

[40] E. Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In *Proceedings of Eurographics'90*, pp. 507–518, September 1990.

[41] R. Y. Tsai. A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lens. *IEEE Journal of Robotics and Automation*, 3(4):323–344, 1987.

[42] J. Wang and W. J. Wilson. 3D relative position and orientation estimation using Kalman filter for robot control. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2638–2645, May 1992.

[43] J. Weng, P. Cohen, and M. Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):965–980, 1992.

[44] D. J. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *CVGIP: Image Understanding*, 55(1):14–26, 1992.

[45] W. J. Wilson, C. C. W. Hulls, and G. S. Bell. Relative end-effector control using Cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12(5):684–696, 1996.

[46] J. W. Yi, T. S. Yang, and J. H. Oh. Estimation of depth and 3D motion parameters of moving objects with multiple stereo images by using Kalman filter. In *Proceedings of the 21st IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, vol. 2, pp. 1225–1230, November 1995.

[47] J. S. C. Yuan. A general photogrammetric method for determining object position and orientation. *IEEE Transactions on Robotics and Automation*, 5(2):129–142, 1999.