Going further with direct visual servoing

Quentin Bateux

ANNÉE 2018

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MathSTIC

présentée par

Quentin BATEUX

préparée à l'unité de recherche IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
Univertisé de Rennes 1

Going further with direct visual servoing

Thèse soutenue à Rennes
le 12 Février 2018

devant le jury composé de :

Christophe DOIGNON
Professeur à l'Université de Strasbourg/ rapporteur

El Mustapha MOUADDIB
Professeur à l'Université de Picardie-Jules Verne/ rapporteur

Céline TEULIÈRE
Maître de conférence à l'Université Clermont Auvergne/ examinatrice

David FILIAT
Professeur à l'ENSTA Paristech/ examinateur

Elisa FROMONT
Professeur à l'Université de Rennes 1/ examinateur

Éric MARCHAND
Professeur à l'Université de Rennes 1/directeur de thèse

CONTENTS

INTRODUCTION

Vision is the primary sense used by humans to perform their daily tasks and move efficiently through the world. Although using our eyes seemingly occurs easily and naturally, whether it is to recognize objects, localize ourselves within a known or unknown environment or interpret the expression of a face: it seems that the images captured through our visual system come already laden with significations. The history of computer vision research throughout many decades proved that meaning does not come from the sensor directly: without the brain's visual cortex to process the eye's input, an image is no more than unstructured data. What is perceived as simple -innate even- for most humans comes in fact from incredible amounts of learning, performed during literally every waking hour since birth.

Developed and popularized in the 19th century, photographic devices are used as a way to preserve memories by fixing visual scenes. With the advancements of electronics and the explosion of the smartphone market, the numerical camera became a cheap and reliable sensor allowing to perceive large amounts of information without acting on the environment. But a camera feed does not come with any prior knowledge of the world, and retrieving context from raw data is the sort of problems computer vision researchers are tackling: how to give to artificial vision systems the ability to infer meaning from images to solve complex problems?

Understanding complex scenes is a key toward autonomous systems such as autonomous robots. Nowadays, robots are heavily used in the industry, to perform repetitive and/or dangerous tasks. The adoption of robots in factories has been fast, as these environments are well-structured, and the tasks well separated from each others. This relative simplicity allowed to use robots even with low to no level of adaptability. On the other hand, outside of factories, robots presence is much more scarce. The underlying cause is the nature of the world itself: shifting and unpredictable. To operate in an environment where endless variability of

objects can be observed, where lighting conditions fluctuate, the ability to adapt is mandatory.

For a robotic system, adaptation is the ability to extract meaningful data representations from an image and modify its behaviour accordingly to solve a given task.

Although large amounts of research are aimed at developing systems able to solve complex problems, such as autonomous driving cars, research concerning ”lower” level tasks is still in progress. This concerns operations such as tracking, positioning, detection, segmentation, localization... that play a critical role as basis of more complex decision and control algorithms. In this thesis, we choose to address the visual servoing problem, a set of methods that consists of positioning a robot using the information provided by a visual sensor.

In the visual servoing literature, classical approaches propose to extract geometrical visual features from the image through a pre-processing step, then use these features (key-points, lines, contours...) to perform the positioning of the robotic system. This separates the problem into two separate tasks, an image processing problem and a control problem.One drawback from this approach is by not using the full image information, measurement errors can arise and later result in control errors. In this thesis, we focus mostly on the direct visual servoing subset of methods. Direct visual servoing is characterized by the use of the entirety of the image information to solve the positioning task. Instead of comparing two sets of geometrical features, the control laws rely directly on a similarity measure between two images without any feature extraction process. This imply the need for a robust way of performing this similarity evaluation, along with efficient and adequate optimization schemes.

This thesis proposes a set of new ways to exploit the full image information in order to solve visual positioning tasks: by providing ways to compare image statistic distributions as visual features, to rely on predictions generated on the fly from the camera views to overcome optimization issues, and to use task-specific prior knowledge modeled through deep learning strategies.

Organization of the manuscript

The first two chapters present the basic knowledge on which our contributions are building on.

- The first chapter provides the background knowledge on computer vision required to solve visual servoing tasks.
- The second chapter details the basics of visual servoing, dealing with both classical and direct methods.

The next chapters of the manuscript present the contributions that we proposed to extend the direct visual servoing state-of-the-art. Following the organization of the manuscript, the list of contributions is such as:

- Chapter 3: We propose a generic framework to consider histograms as a new visual feature to perform direct visual servoing tasks with increased robustness and flexibility. It allows the definition of efficient control laws by allowing to choose from any type of histograms to describe images, from intensity to color histograms, or Histograms of Oriented Gradients. Several control laws derived from these histograms are defined and validated on both 6DOF positioning tasks and 2DOF navigation tasks.
- Chapter 4: A novel direct visual servoing control law is then proposed, based on a particle filter to perform the optimization part of visual servoing tasks, allowing to accomplish tasks associated with highly non-linear and non-convex cost functions. The Particle Filter estimate can be computed in real-time through the use of image transfer techniques to evaluate camera motions associated to suitable displacements of the considered visual features in the image. This new control law has been validated on a 6DOF positioning task.
- Chapter 5: We present a novel way of modeling the visual servoing problem through the use of deep learning and Convolutional Neural Networks to alleviate the difficulty to model non-convex problems through analytic methods. By using image transfer techniques, we propose a method to generate quickly large training datasets in order to fine-tune existing network architectures to solve 6DOF visual servoing tasks. We show that this method can be applied both to model known static scenes, or more generally to model relative pose estimations between couples of viewpoints from arbitrary scenes. These two approaches have been applied to 6DOF visual servoing positioning tasks.

Chapter 1

BACKGROUND ON ROBOT VISION GEOMETRY

This chapter presents the basic mathematical tools used throughout the robot vision field and more specifically the tools required for the elaboration of visual servoing techniques. In order to operate a robot within an environment, these two elements (robot and environment) need to be described in mathematical terms. On top of this model, it is also necessary to describe the digital image acquisition process that is modeled through the pin-hole camera model, a model that allows the projection of visual information from the 3D space onto a 2D plane and ultimately in sensor space. More in-depth details can be found in, for example in [Ma 12]

3D world modeling

To deal with real world robotics applications, the relationship between a given environment and the robotic system needs to be defined. At human scale, the environment is described successfully by the Euclidean geometry, that allows to describe positions and motions of objects with respects to each other. It also extends to define transformations and projections of information between the 3D space to various sensors spaces, including 2D cameras.

Euclidean geometry

Being described around 300BC by Euclid, the Euclidean geometrical model is valid and accurate for most situations occurring at a human scale, and hence has been used extensively in robotics and computer vision.

More specifically, we need Euclidean geometry to be able to express objects coordinates with respects to one another. This is done thanks to the definition of frames. A frame can be seen as an anchor and each object in the world can be defined as being at the origin position of its own frame. From there, it is also possible to define transformations in space to express coordinates from one frame to another. For example it is possible to express coordinates in a given object a space as coordinates in a second object b space as illustrated in 1.1. This allows to compute relative positions and orientation differences between those two objects by actually defining the rigid relationship that exists between the frames of these objects. In a more formal

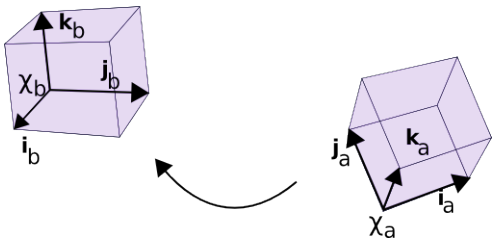


Figure 1.1: *Rigid transformation from object frame to camera frame*

definition, a frame $\mathcal{F}_a$ attached to an object defines a pose with respect to another frame (often a world frame that arbitrarily defines a reference position and orientation), meaning a relative 3D position and an orientation, by using a basis of 3 orthonormal unit vectors $(\mathbf{i_a}, \mathbf{j_a}, \mathbf{k_a})$. The frame origin χ_a is always defined as a point and is often taken as the center of the object for more convenience, although any point in space could be chosen. The notation for the Euclidean space of dimension n is $\mathbb{R}^n$. Here we work mostly with the three-dimensional Euclidean space $\mathbb{R}^3$.

From this definition, any point χ in space can be located in frame by a set of three coordinates, defined in an object's frame ${}^a\mathbf{X} = ({}^a\mathrm{X}, {}^a\mathrm{Y}, {}^a\mathrm{Z})^\top$ such as:

$$\chi = \chi_0 + {}^a\mathrm{X}\mathbf{i}_a + {}^a\mathrm{Y}\mathbf{j}_a + {}^a\mathrm{Z}\mathbf{k}_a \tag{1.1}$$

One typical operation that is performed in robot vision is the change of coordinates from a point defined in a given object frame $\mathcal{F}_a$ to one defined in another frame $\mathcal{F}_b$. This transformation has to model both the change in position and orientation between those two frames. Therefore it is defined by a 3D translation ${}^b\mathbf{t}_a$ that transforms the origin χ_a of the object frame into the origin χ_b of the camera frame, as well as a 3D rotation ${}^b\mathbf{R}_a$ that transforms the first frame's axes into the second's. A point ${}^a\mathbf{X}$ defined with respect to $\mathcal{F}_a$ is expressed as a point ${}^b\mathbf{X}$ defined with respect to $\mathcal{F}_b$ such as:

$$ {}^b\mathbf{X} = {}^b\mathbf{R}_a \, {}^a\mathbf{X} + {}^b\mathbf{t}_a \tag{1.2} $$

${}^b\mathbf{R}_a$ is a 3×3 rotation matrix (it belongs to the SO(3) rotation group, containing all the rotation transformations about the origin of the 3D Euclidean space that can be performed through a composition operation) and ${}^b\mathbf{t}_a$ is a 3×1 translation vector.

Homogeneous matrix transformations

The homogeneous matrix representation of geometrical transformations is commonly used in robotic vision field. It allows to compute easily linear transformations, such as the position of a frame with respect to another, in the form of successive matrix multiplications. To consider the Euclidean model into this formalism, one needs to define the notions presented in the previous section within this matrix representation: from the Euclidean representation $\mathbf{X} = (X, Y, Z)^\top$, one can define this same point in a projective space by using homogeneous coordinates that yield $\tilde{\mathbf{X}} = (\mathbf{X}, 1)^\top = (\lambda\mathbf{X}, \lambda)^\top$ with λ a scalar. This allows us to re-write Eq (1.2) such as:

$$ {}^c\tilde{\mathbf{X}} = {}^c\mathbf{M}_o \, {}^o\tilde{\mathbf{X}} \quad \text{with:} \quad {}^c\mathbf{M}_o = \begin{bmatrix} {}^c\mathbf{R}_o & {}^c\mathbf{t}_o \\ \mathbf{0} & 1 \end{bmatrix} \tag{1.3} $$

This allows the full transformation from object to camera frame to be contained in ${}^c\mathbf{M}_o$.

It is important when one wants a minimal representation for a frame transformations, as several parametrizations can be considered. The translational part is straightforward, as the 3 parameters of the translation matrix fully define the changes in coordinates along each of the 3 axes from the object to the camera frame. For rotations however, several popular representations are commonly used in the computer vision community to represent them, such as Euler angles or θu.

One first representation is the Euler angle representation of the rotations, which consists in considering the 3D rotation as a combination of three consecutive rotations around each three axes of the space. This defines the 3D rotation matrix as the product of three 1D rotation matrices around the x, y and z axes. Each one of these 1D rotations is defined by a single rotation angle (resp. r_x, r_y and r_z). The main advantage of this representation is its simplicity, being very intuitive for human readability, but it suffers from several bad configurations (singularities) which can lead to a loss in controlability of one or more degrees of freedom. This singularity issue can be illustrated by a mechanical system that consists of three gimbals, as illustrated by Fig. 1.2(a). The singular configurations occur when 2 rotation axes are aligned (see Fig. 1.2(b)), making it impossible to transform both independently: a situation called *gimbal lock*. In this thesis we will use the Axis-Angle representation (often denoted

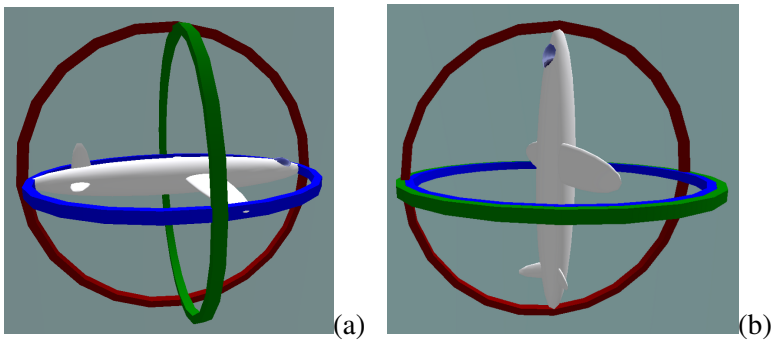

(a) (b)

Figure 1.2: *3 axes-gimbal: the object attached to the inner circle can rotate in 3D. (a) General position: the object has 3 degree of freedom. (b) Singular 'gimbal-lock' position: as two axes are aligned, the object can now be controlled only along 2 degree of freedom, the aligned axes becoming dependents.*

θu representation), which avoids the gimbal-lock singular configurations. This representation consists of a unit vector $\mathbf{u} = (u_x, u_y, u_z)$ that indicates the direction of the rotation axis and an angle θ around this axis. The rotation matrix is then built using the Rodrigues formula:

$$\mathbf{R} = \mathbf{I}_3 + (1 - cos\theta)\mathbf{u}\mathbf{u}^\top + sin\theta[\mathbf{u}]_\times \tag{1.4}$$

with

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ u_y & u_x & 0 \end{bmatrix} \tag{1.5}$$

being an antisymmetric matrix, also called skew-symmetric matrix.

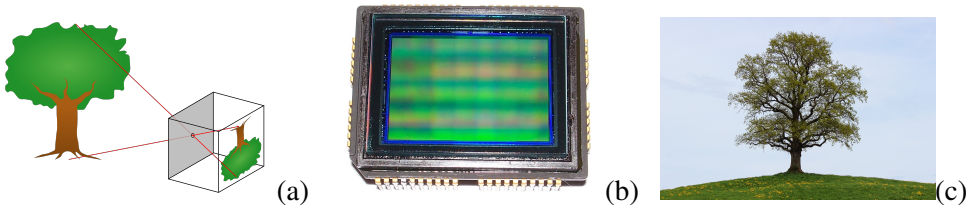
The perspective projection model

The presented work is focused on using monocular cameras as the main sensor to gather information concerning the surroundings of the robot. A monocular camera is a very powerful and versatile sensor, as it gathers a lot of information in the form of 2D images. Although an image contains very detailed information concerning the textures and lighting characteristics of the scene it represents, by definition, it is limited in the perception of the geometry due to the projection of 3D data into a 2D image.

This projection from 3D to 2D can be modeled through the pin-hole (or perspective) camera model, that describes the process of image acquisition for traditional cameras.

From the 3D world to the image plane

The pin-hole camera model can be explained schematically as follows: a bright scene on one side, and a dark room on the other side. The two sides are separated by a wall, pierced by a single small hole. If a planar surface is set vertically in the room in front of the hole, the light rays from the outdoor scene, channeled through the hole will hit the sheet plane and form a reversed image of the outdoor scene, as illustrated by Fig. 1.3(a). By setting an array of light-sensitive sensors (CCD or CMOS, see Fig. 1.3(b)) that coincides with this area, it is then possible to record an image from the light coming by the scene to the sensor in the form of a 2D pixel array, that forms a digital image, such as Fig. 1.3(c). This digital image is made of discrete information pieces, called pixels, reconstituted from the discrete signals received from the individual sensors forming the CCD array.



(a) (b) (c)

Figure 1.3: *Projection camera model: (a) Pinhole model, (b) CCD sensor, (c) Digital image*

From scene to pixel

In the pinhole camera model, the hole in the wall described in the previous section is called center of projection, and the plane where the image is projected is named projection (or image) plane.

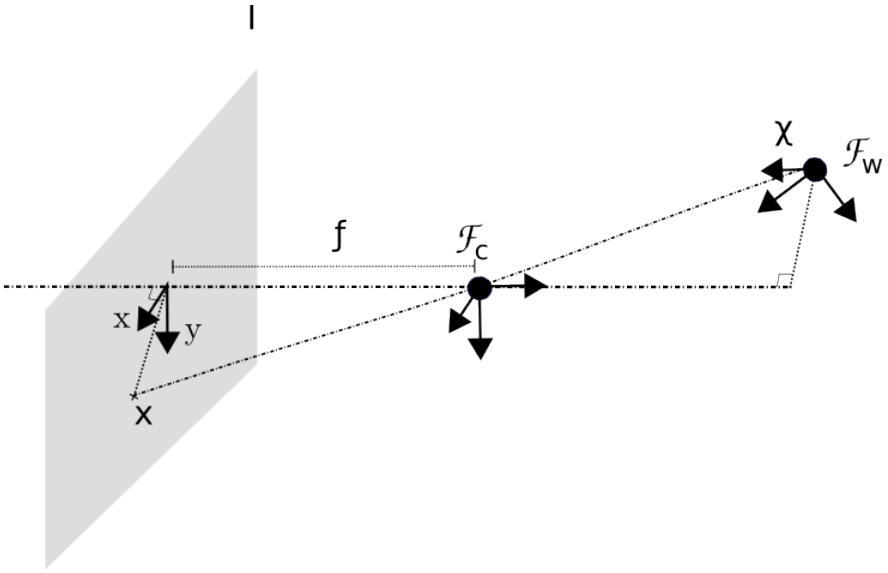


Figure 1.4: *Perspective projection model*

This model is illustrated in Fig. 1.4. The camera frame is chosen such as the z-axis (frontal axis) is facing the scene and is orthogonal to the image plane I.

Let us denote $\mathcal{F}_c$ the camera frame, and ${}^c\mathbf{T}_w$ the transformation that defines the position of the camera frame with respect to the world frame $\mathcal{F}_w$. ${}^c\mathbf{T}_w$ is an homogeneous matrix such as:

$$ {}^c\mathbf{T}_w = \begin{bmatrix} {}^w\mathbf{R}_c & {}^w\mathbf{t}_c \\ \mathbf{0} & 1 \end{bmatrix} \tag{1.6} $$

where ${}^w\mathbf{R}_c$ and ${}^w\mathbf{t}_c$ are respectively the rotation matrix and translation vector defining the position of the camera in the world frame.

From there, the projective projection in normalized metric space $\mathbf{x} = (x, y, 1)^\top$ of a given point ${}^w\mathbf{X} = ({}^wX, {}^wY, {}^wZ, 1)^\top$ is given by:

$$ \mathbf{x} = \Pi \, {}^c\mathbf{M}_w \, {}^w\mathbf{X} \tag{1.7} $$

where Π is the projection matrix, given in the case of a perspective projection model, by:

$$\Pi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.8}$$

As measures performed in the image space depend on the camera intrinsic parameters, we need to define the coordinates of a point in pixel units $\tilde{\mathbf{x}}$.

It is then possible to link $\mathbf{x}$ and $\tilde{\mathbf{x}}$ such as:

$$\mathbf{x} = \mathbf{K}^{-1}\tilde{\mathbf{x}} \tag{1.9}$$

where $\mathbf{K}$ is the camera intrinsic parameters matrix, defined by:

$$\mathbf{K} = \begin{bmatrix} p_x & 0 & u_0 \\ 0 & p_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

with $(u_0, v_0, 1)^\top$ the coordinates of the principal point (intersection of the optical axis with the image plane) and p_x (and p_y resp.) is a ratio between the focal length f of the lens and the pixel width (resp. height) of the sensor such as: $p_x = \frac{f}{l_x}$ (resp. $p_y = \frac{f}{l_y}$). The intrinsic parameters can be obtained through an off-line calibration step ([Zhang 00]).

For convenience, only coordinates in normalized metric space will be used in the rest of this thesis.

We consider here a pure projective model, but any additional model with distortion can be easily considered and handled from this framework.

Generating images from novel viewpoints

In several chapters of this thesis, we will need to generate images as if seen from other camera viewpoints through image transfer techniques.

An image transformation function takes the pixels from a source image and an associated motion model to generates the corresponding modified image (in our use case, an approximation of the scene as seen if the camera had been displaced in the real world). The transformation process consists of two distinct mechanisms: the mapping that makes each source pixel position correspond to a destination pixel's position, and a resampling process that determines the destination pixel' value.

Image transformation function parametrization

In this thesis, no prior depth information is known in the scenes seen by the camera. Hence, we will use the simplifying hypothesis that the scene is planar. To generate images from a given scene as taken from another point of view, we then need to find an appropriate parametrization for the image transformation.

Many image transformation can be defined, depending on the expected type of motion. A general notation can be written as:

$$\mathbf{x}_2 = w(\mathbf{x}_2, \mathbf{h}) \tag{1.11}$$

where $\mathbf{h}$ the set of parameters is the associated image transformation model (it can represent a translation, an affine motion model, a homography, etc.), that transfers a point $\mathbf{x}_1$ in an image $\mathbf{I}_1$ to a point $\mathbf{x}_2$ in an image $\mathbf{I}_2$.

Let us assume that all the points ${}^1\mathbf{X} = ({}^1X, {}^1Y, {}^1Z)$ of the scene belong to a plane $\mathcal{P}({}^1\mathbf{n}, {}^1d)$ where ${}^1\mathbf{n}$ and ${}^1d$ are the normal and origin of the reference plane expressed in the camera frame 1 (${}^1\mathbf{X}\,{}^1\mathbf{n} = {}^1d$).

An accurate way to perform this operation is to use the homography transformation. A homography is a transformation that warps a plane on another plane. In our case, it allows to keep consistent this image generation with the projective image acquisition process detailed before. In computer vision, it is usually used to model the displacement of a camera observing a planar object.

Under the planar scene hypothesis, the coordinates of any point $\mathbf{x}_1$ in an image $\mathbf{I}_1$ are linked to the coordinates $\mathbf{x}_2$ in an image $\mathbf{I}_2$ thanks to a homography ${}^2\mathbf{H}_1$ such that:

$$\mathbf{x}_2 = w(\mathbf{x}_1, \mathbf{h}) = {}^2\mathbf{H}_1\,\mathbf{x}_1 \tag{1.12}$$

with

$$ {}^{1}\mathbf{H}_{2} = \left({}^{2}\mathbf{R}_{1} + \frac{{}^{2}\mathbf{t}_{1}\ {}^{1}\mathbf{n}^{\top}}{{}^{1}d} \right) \tag{1.13} $$

From the viewpoint prediction perspective, this means that a pixel $\mathbf{x}_1$ in the image $\mathbf{I}_1$ will be at coordinates $\mathbf{x}_2$ in an image $\mathbf{I}_2$ considering that the camera undergoes the ${}^{2}\mathbf{H}_{1}$ motion.

Full image transfer

From this image transfer technique, it is then possible to generate approximated images viewed from virtual cameras. One needs only to set the constant parameters $\mathbf{n}$ and d to arbitrary values that represent the initial expected relationship between the recorded scene and the camera, in orientation and distance (depth). As in this thesis no prior information is known about the scene geometry and the relationship between the orientation of the image plane and the camera pose, $\mathbf{n}$ will always be set as equal to the optical axis of the camera. The depth d will depend of the average depth between the camera and the scene, which may vary from one test scene to another, and thus is set empirically.

Then, by specifying the 6DOF pose relationship between the camera and virtual camera, it is possible to generate an approximated image as seen from a virtual viewpoint, as illustrated by Fig. 1.5, by applying for every pixel in the image observed by the camera the following relationship:

$$ \mathbf{I}_2(\mathbf{x}) = \mathbf{I}_1\left(w(\mathbf{x}, \mathbf{h})\right) \tag{1.14} $$

where $\mathbf{h}$ is a representation of the homography transformation. In the two last chapters of this thesis, we will use this image generation technique extensively as a way to predict viewpoints to avoid actually moving the robot to record a particular viewpoint, an action that is time-consuming and not compatible with a real-time solving of visual servoing tasks.

Conclusion

In this chapter, we described the necessary tools to model the acquisition of numerical images and to understand the relationship between the physical position of objects with respect to their associated projection in a 2D plane, and ultimately with their location in an image recorded

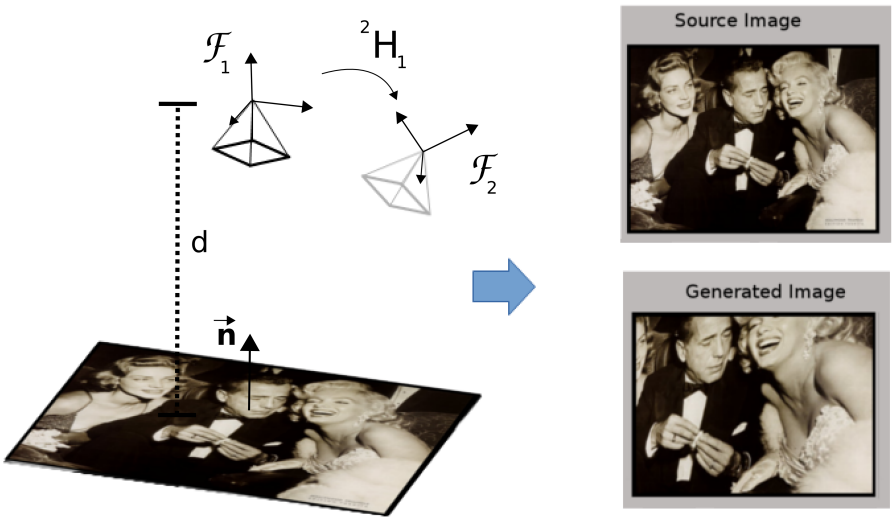


Figure 1.5: *Generating virtual viewpoints through homography*

by a camera. We also detailed a way to perform the generation of images seen from virtual viewpoints as a way to predict sensor input associated to given camera motions.

Chapter 2

BACKGROUND ON VISUAL SERVOING

The goal of the chapter is to present the basics of visual servoing (VS) techniques, then focus more closely on the image-based VS techniques (IBVS).This chapter will also highlight the classical methods for performing IBVS, the underlying challenges that these methods try to address and the state-of-the-art of the field.

Problem statement and applications

Visual servoing techniques are designed to control a dynamic system, such as a robot, by using the information provided by one or multiple cameras. Many modalities of cameras can be used, such as ultrasound probes or omnidirectional cameras.

By essence, it is well suited to solve positioning or tracking tasks, a category of problem that is critical in a wide variety of applications where a robot has to move toward an unknown or moving target position, such as grasping (Fig. 2.1(a)), navigating autonomously (Fig. 2.1(b)), enhancing the quality of an ultrasound image by adjusting its orientation (Fig. 2.1(c)) or controlling robotic arms toward handles for locomotion in low-gravity environments (Fig. 2.1(d)).

From this range of situations, two major configurations can be defined regarding the relation between the sensor and the end-effector. Either the camera is mounted directly on the

end-effector, defining the 'eye-in-hand' visual servoing, or the camera is mounted on the environment and is looking at the end-effector, defining the 'eye-to-hand' visual servoing.

In this thesis, we will consider only the "eye-in-hand" configuration, meaning that control is performed in the camera frame. Compared to the "eye-to-hand" configuration, where the camera observes the effector from a remote point of view, this configuration benefits from an increasing access to close details of the scene in the camera field of view, although the more global relationship between the effector and its surroundings is as a result, not available.

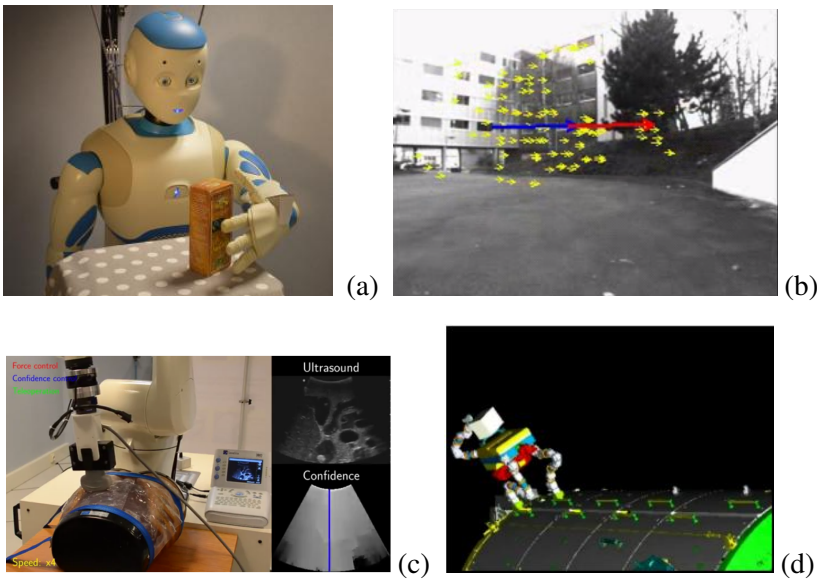


Figure 2.1: *Illustrations of visual servoing applications. (a) The Romeo robot positioning its hand to grasp a box [Petit 13]. (b) An autonomous vehicle following a path by using recorded images of the journey [Cherubini 10]. (c) A Viper robotic arm adapting its position to get optimal image quality from a ultrasound probe [Chatelain 15]. (d) Eurobot walking on the ISS by detecting and grasping ladders.*

Visual servoing techniques

The aim of a positioning task is to reach a desired pose of the camera $\mathbf{r}^*$, starting from an arbitrary initial pose. To achieve that goal, one needs to define a cost function to be minimized that reflects, usually in the image space, this positioning error. Considering the current pose of the camera $\mathbf{r}$ the problem can therefore be written as an optimization process:

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}} \rho(\mathbf{r}, \mathbf{r}^*) \tag{2.1}$$

where ρ is the error function between the desired pose and the current one, and $\widehat{\mathbf{r}}$ the pose reached after the optimization process (servoing process), which is the closest possible to $\mathbf{r}^*$ (optimally $\widehat{\mathbf{r}} = \mathbf{r}^*$).

This cost function $\rho(\mathbf{r}, \mathbf{r}^*)$ is typically defined as a distance between two vectors of visual features $\mathbf{s}(\mathbf{r})$ and $\mathbf{s}^*$. $\mathbf{s}(\mathbf{r})$ is a set of information extracted from the image at pose $\mathbf{r}$, and $\mathbf{s}^*$ the set of information extracted at pose $\mathbf{r}^*$ chosen such as $\mathbf{s}(\mathbf{r}) = \mathbf{s}^*$ when $\mathbf{r} = \mathbf{r}^*$.

Provided that such a suitable distance exists between each vector of features, the task is solved by computing and applying a camera velocity that will make this distance decrease toward the minimum of the cost function.

In order to update the pose of the camera, one needs to apply a velocity $\mathbf{v}$ to it. This velocity $\mathbf{v}$ is a vector containing the motion to be applied to the camera along each of the degrees of freedom (DOF) that needs to be controlled. For example, a velocity applied to control the full 6 DOF will be in the form $\mathbf{v} = (t_x, t_y, t_z, r_x, r_y, r_z)$, where t_x (resp. t_y and t_z) controls the motion to be applied along the x-axis (resp. y and z axes) of the camera frame, and r_x (resp. r_y and r_z) controls the rotation around the x-axis (resp. y and z axes) of the camera frame.

Classically, this error function is expressed through two types of features: either by relying on geometrical features that are extracted and matched for each frame, which defines the classical visual servoing [Chaumette 06] set of techniques, or by relying directly on pixel intensities (and possibly more elaborate descriptors based on these intensities) without any tracking or matching, defining the direct visual servoing methods [Collewet 08a, Dame 11].

Visual servoing on geometrical features

Model of the generic control law

A classical approach for performing visual servoing tasks is to use geometrical information $\mathbf{x}(\mathbf{r})$ extracted from the image taken in the configuration $\mathbf{r}$. Considering a vector of such geometrical features $\mathbf{s}(\mathbf{x}(\mathbf{r}))$, the typical task is to minimize the difference between $\mathbf{s}(\mathbf{x}(\mathbf{r}))$ and the corresponding vector of features observed at their desired configuration $\mathbf{s}(x(\mathbf{r}^*))$. To increase readability, we will simplify $\mathbf{s}(x(\mathbf{r}))$ into $\mathbf{s}(\mathbf{r})$, and $\mathbf{s}(x(\mathbf{r}^*))$ which is constant, in our

case, throughout the task, into $\mathbf{s}^*$. Assuming that a suitable distance between each of the feature couples can be defined, the visual servoing task can be expressed as the following optimization problem:

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}} (\mathbf{s}(\mathbf{r}) - \mathbf{s}^*)^\top (\mathbf{s}(\mathbf{r}) - \mathbf{s}^*) \tag{2.2}$$

We define an error $\mathbf{e}(\mathbf{r})$ between the two sets of information $\mathbf{s}(\mathbf{r})$ and $\mathbf{s}^*$, such as:

$$\mathbf{e}(\mathbf{r}) = \mathbf{s}(\mathbf{r}) - \mathbf{s}^* \tag{2.3}$$

In order to ensure appropriate robot dynamics (fast motion when the error is large, and slower motion as the camera approaches the target pose), we specify an exponential decrease of the measure $\mathbf{e}(\mathbf{r})$, such as:

$$\dot{\mathbf{e}}(\mathbf{r}) = -\lambda \mathbf{e}(\mathbf{r}) \tag{2.4}$$

where λ is a scalar factor used as a gain and since $\mathbf{s}^*$ is constant, then $\dot{\mathbf{e}}(\mathbf{r}) = \dot{\mathbf{s}}(\mathbf{r})$, leading to:

$$\dot{\mathbf{s}}(\mathbf{r}) = -\lambda (\mathbf{s}(\mathbf{r}) - \mathbf{s}^*) \tag{2.5}$$

On the other hand, this visual servoing task is achieved by iteratively applying a velocity to the camera. The notion of interaction matrix $\mathbf{L_s}$ of $\mathbf{s}(\mathbf{r})$ is then introduced to link the motion of $\mathbf{s}(\mathbf{r})$ to the camera velocity. It is expressed as:

$$\dot{\mathbf{s}}(\mathbf{r}) = \mathbf{L_s}\mathbf{v} \tag{2.6}$$

where $\mathbf{v}$ is the camera velocity.

By combining Eq. (2.5) and Eq. (2.6), we obtain:

$$\mathbf{L_s}\mathbf{v} = -\lambda (\mathbf{s}(\mathbf{r}) - \mathbf{s}^*) \tag{2.7}$$

By solving this equation through an iterative least-square approach, the control law becomes:

$$\mathbf{v} = -\lambda \mathbf{L_s^+} (\mathbf{s}(\mathbf{r}) - \mathbf{s}^*) \tag{2.8}$$

where λ is a positive scalar gain defining the convergence rate of the control law and $\mathbf{L_s^+}$ is the pseudo-inverse of the interaction matrix, defined such as:

$$\mathbf{L_s^+} = (\mathbf{L_s}^\top \mathbf{L_s})^{-1} \mathbf{L_s}^\top \tag{2.9}$$

Control laws can be seen as gradient descent problems, meaning that these methods rely on finding the gradient descent direction $\mathbf{d}(\mathbf{r})$ of the cost function to be optimized. From this value, the control law that computes the velocity to apply to the camera is given by:

$$\mathbf{v} = \lambda \mathbf{d}(\mathbf{r}) \tag{2.10}$$

with λ a scalar that determines the amplitude of the motion (also called gain or descent step). To link back with Eq. (2.8), we can express:

$$\mathbf{d}(\mathbf{r}) = -\mathbf{L}_{\mathbf{s}}^{+}(\mathbf{s}(\mathbf{r}) - \mathbf{s}^{*}) \tag{2.11}$$

The computation of the direction of descent can be obtained by a range of methods such as the steepest descent, Newton (as in Eq. (2.11)), Gauss-Newton or Levenberg-Marquardt methods. In this thesis, we will use the last two, as they give a greater flexibility.

It is important to note that all the methods based on a gradient descent share the same underlying characteristics, as they all rely on the assumption that the problem to be solved (reaching the cost function minimum) is locally convex, as illustrated by Fig. 2.2(a). In this case, the cost function is described as well defined (smoothly decreasing toward a clearly defined minimum) and this set of methods will perform well in term of behavior and final precision.

However, for image processing problems, the cost function can have less desirable properties depending on the nature of the feature that is used and the perturbations that can occur during the image acquisition process. The resulting cost function can get ill-defined, making it more difficult for the process to converge toward the global minimum as the cost function gets more and more non-convex and riddled with local minima, as illustrated by Fig. 2.2(b). This can lead to the impossibility to reach the minimum depending on the starting state, as the gradient descent can get stuck in a local minimum or diverge completely in the worst case.

It is also important to note that with most visual features used throughout this thesis, we are dealing almost exclusively with locally convex problems (as opposed to globally convex), as illustrated by Fig. 2.2(b) where the problem becomes intractable for classical gradient-based methods when the starting point is too far from the optimum.

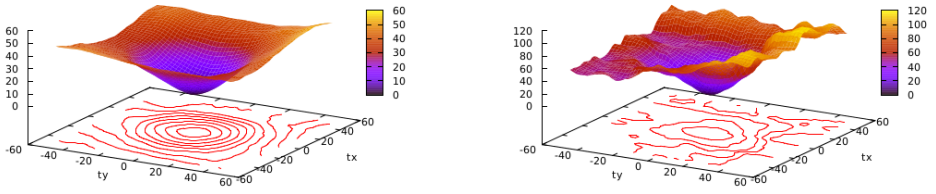


Figure 2.2: *(a) Near optimal convex function. (b) Locally limited convex function*

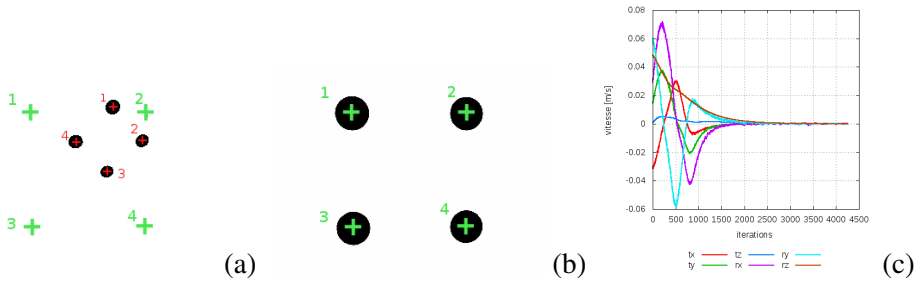


Figure 2.3: *(a) Camera view at initial position. (b) Camera view at the end of the motion. (c) Velocity components applied to the camera*

Example of the point-based visual servoing [Chaumette 06]

For the classical 4 point-based visual servoing problem (illustrated by Fig. 2.3), the goal is to regulate to zero the distance between the position $\mathbf{x}_i = (x_i, y_i)$ of the 4 points detected in the current image (red crosses) and their desired position (x_i^*, y_i^*) (green crosses), with $i \in [0..3]$. In this case, the feature vector $\mathbf{s}$ is defined using the 2D coordinates of the dots, such as:

$$\mathbf{s}(\mathbf{r}) = (\mathbf{x}_0(\mathbf{r}), \mathbf{x}_1(\mathbf{r}), \mathbf{x}_2(\mathbf{r}), \mathbf{x}_3(\mathbf{r}))$$
$$\mathbf{s}^* = (\mathbf{x}_0^*, \mathbf{x}_1^*, \mathbf{x}_2^*, \mathbf{x}_3^*)$$

In order to solve this positioning problem, we need to define an error to minimize, here the 2D error $\mathbf{e}$ between $\mathbf{s}$ and $\mathbf{s}^*$:

$$\mathbf{e}(\mathbf{r}) = \mathbf{s}(\mathbf{r}) - \mathbf{s}^* \tag{2.12}$$

where $\mathbf{r}$ is the pose of the camera.

The classical control law to drive the error down following an exponential decrease is such as:

$$\mathbf{v} = -\lambda \mathbf{L}_\mathbf{s}^+ \mathbf{e}(\mathbf{r}) \tag{2.13}$$

where $\mathbf{L}_\mathbf{s}$ is the interaction matrix that links the displacement of the features $\mathbf{s}$ in the image to the velocity $\mathbf{v}$ of the camera. As $\mathbf{s}$ is a concatenation of all the point features, $\mathbf{L}_\mathbf{s}$ is also a concatenation of all the interaction matrices related to each individual features:

$$\mathbf{L}_\mathbf{s} = \begin{bmatrix} \mathbf{L}_{\mathbf{x}_0} & \mathbf{L}_{\mathbf{x}_1} & \mathbf{L}_{\mathbf{x}_2} & \mathbf{L}_{\mathbf{x}_3} \end{bmatrix}^\top \tag{2.14}$$

with:

$$\mathbf{L}_{\mathbf{x}_i} = \begin{bmatrix} -1/Z & 0 & x_i/Z & x_i y_i & -(1+x_i^2) & y_i \\ 0 & -1/Z & y_i/Z & 1+y_i^2 & -x_i y_i & -x_i \end{bmatrix} \tag{2.15}$$

with Z the depth between the camera plane and the points, it is unknown and thus set as a constant. This expression is obtained thanks to the projective geometry equations. Details can be found in [Chaumette 06].

Photometric visual servoing

To avoid extraction and tracking of geometrical features (such as points, lines, etc) from each image frames, in [Collewet 08b] [Collewet 11] the notion of direct (or photometric) visual servoing where the visual feature is the image considered as a whole has been introduced.

Definition

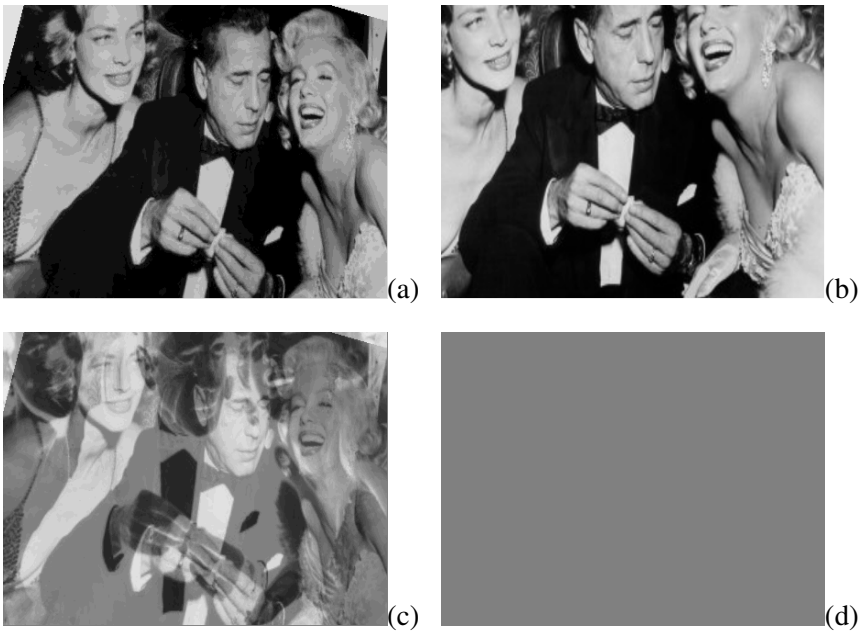


Figure 2.4: *(a) initial position; (b) final position; (c) initial image difference; (d) final image difference*

The goal of the photometric visual servoing is to stay as close as possible to the pixel intensity information in order to eliminate the tracking and matching processes that remain a bottleneck for VS performances. The feature $\mathbf{s}$ associated to this control scheme becomes the entire image ($\mathbf{s}(\mathbf{r}) = \mathbf{I}(\mathbf{r})$). This means that the optimization process becomes:

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}}((\mathbf{I}(\mathbf{r}) - \mathbf{I}^*)^\top (\mathbf{I}(\mathbf{r}) - \mathbf{I}^*)) \tag{2.16}$$

where $\mathbf{I}(\mathbf{r})$ and $\mathbf{I}^*$ are respectively the image seen at the position $\mathbf{r}$ and the reference image (both of N pixels). Each image is represented as a single column vector. The interaction matrix $\mathbf{L_I}$

can be expressed as:

$$\mathbf{L_I} = \begin{bmatrix} \mathbf{L_{I(x_0)}} \\ \vdots \\ \mathbf{L_{I(x_N)}} \end{bmatrix} \tag{2.17}$$

with N the number of pixels in the image, and $\mathbf{L_{I(x)}}$ the interaction matrix associated to each pixel. As described in [Collewet 08a], thanks to the optical flow constraint equation (the hypothesis that the brightness of a physical point will remain constant throughout the experiment), it is possible to link the temporal variation of the luminance $\mathbf{I}$ to the image motion at a given point, leading to the following interaction matrix for each pixel:

$$\mathbf{L_{I(x)}} = -\nabla I^{\top} \mathbf{L_x} \tag{2.18}$$

with $\mathbf{L_x}$ the interaction matrix of a point (seen in the previous section), and $\nabla I = (\frac{\partial \mathbf{I(x)}}{\partial x}, \frac{\partial \mathbf{I(x)}}{\partial y})$ the gradient of the considered pixel.

The resulting control law is inspired by the Levenberg-Marquardt optimization approach. It is given by:

$$\mathbf{v} = -\lambda(\mathbf{H_I} + \mu\, diag(\mathbf{H_I}))^{-1} \mathbf{L_I}^{\top} (\mathbf{I(r)} - \mathbf{I^*}) \tag{2.19}$$

where μ is the Levenberg-Marquardt parameter which is positive scalar that defines the behavior of the control law, from a steepest gradient behavior (μ small) to a Gauss-Newton's behavior (μ large) and $\mathbf{H_I} = \mathbf{L_I}^{\top} \mathbf{L_I}$.

The advantage of this method is its precision (thanks to redundancy in the amount of data), it's main drawbacks being robustness issues with respect to illumination perturbations and limited convergence area (due to a highly non-linear cost function).

Other methods can be derived from this technique and conserve the advantage of eliminating the need for tracking or matching. The idea is to keep the pixel intensities as the underlying source of information, and to add some higher level computation on those pixels, by exploiting the statistics of those intensities in order to create more powerful descriptors than the strict pixel-wise comparisons of images. This has been done for example in [Dame 09, Dame 11], where a direct visual servoing scheme is created by maximizing the mutual information between the two images as a descriptor in the control law. [Bakthavatchalam 13] proposed a new global feature by using photometric moments to perform visual servoing positioning tasks without traking or matching processes. In [Teuliere 12], the authors proposed an approach relying a control law exploiting directly a 3D point-cloud to perform a positioning task. More

recently, [Crombez 15] proposed a method to exploit a full image information by creating a new feature computed as a Gaussian mixture from the input pixels to increase the convergence domain of the traditional photometric VS.

Common image perturbations

Due to the wide spectrum of applications and possible settings, direct visual servoing methods face a wide array of challenges in terms of variations coming from the type of sensor employed: various resolutions (low resolution vs high resolution, that affect strongly the nature of the available information), various image quality (with various types of noises), or even different modalities (standard CCD cameras, ultrasound cameras, or even event cameras).

The first common type of perturbation consists of illumination changes that affect the pixel intensities, either globally or locally, by shifting the intensity value of the pixels, making them darker or brighter, but often conserving the relative relationship (direction of the gradients, not necessary its norm) between adjacent pixels.

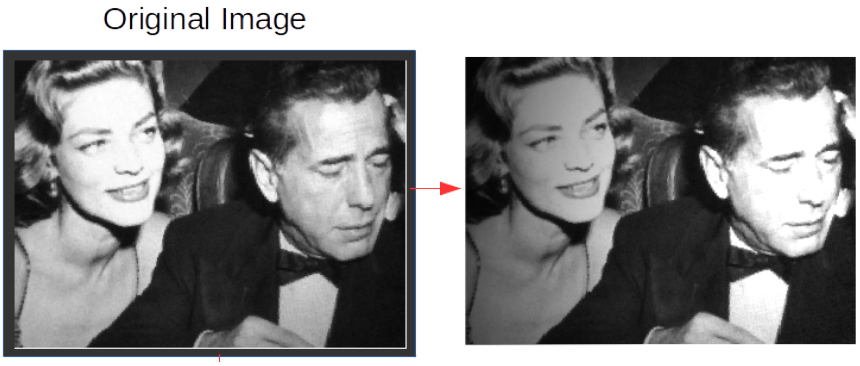


Figure 2.5: *Illumination perturbations*

The second main category of perturbations is the occlusion. It is characterized by a loss of information across an area of the image, often created by an external object that comes between the scene and the sensor. In most cases, all the information on this area is altered independently from the intensities that would be recorded under nominal conditions. Managing this perturbation thus often requires special care in order to prevent the integration of this additional, non-relevant information into the control scheme to avoid instabilities.

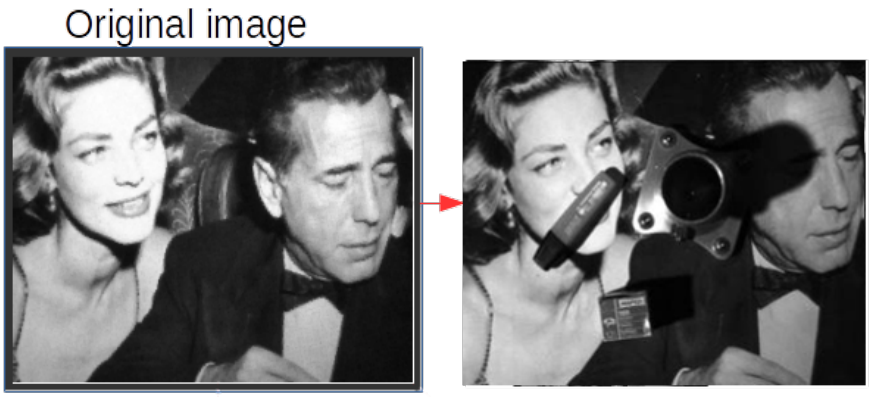


Figure 2.6: *Occlusion perturbations*

Conclusion

In this chapter we gave a general overview of the visual servoing approaches. It is important to underline that traditional geometric-based methods are very powerful and stable, as long as the geometric features involved in the control law can be adequately and precisely extracted and matched from two successive frames. On the other hand, the direct visual servoing methods do not rely on such tracking and matching scheme and exhibit very fine positioning accuracy, although being so far more limited in term of convergence area and robustness, motivating the works presented in this thesis to overcome these issues.

Chapter 3

DESIGN OF A NOVEL VISUAL SERVOING FEATURE: HISTOGRAMS

As seen in the previous chapter, the photometric VS approach [Collewet 11] yields very good results if the lighting conditions are kept constant. A later approach [Dame 09] showed that it was possible to extend this method in order to use the entropy probability distribution in order to perform a positioning task.

In this chapter we propose a new way for achieving visual servoing tasks by considering a new statistical descriptor: the histogram. This descriptor consists of an estimate of the distribution of a given variable (in our case the distributions stemming from the organization of pixels). As an image is a rich representation of a scene, many levels of information can be considered (gray-scale intensity values, color information, directional gradients...), and as a result, histograms can be computed to focus specifically on one or several of those characteristics, making it a very flexible way to represent an image and describe it in a global way.

The proposed VS control laws task are designed in order to minimize the error between two (sets of) histograms. By describing the image in a global way, no feature tracking or extracting is necessary between frames, similarly to the direct VS method [Collewet 11], leading to good task precision by exploiting the high redundancy of the information. In this sense, the proposed method is an extension of the previously described photometric visual servoing as well as the mutual information-based VS.

Histograms basics

In this section, after detailing some statistics basics, we recall the formal definition of histograms using probability distributions and the classical distances that are used to compare two histograms.

Statistics basics

This section presents some elements in statistics that are required to understand the choice of going from pure photometric information to probability distribution (histograms) to perform visual servoing. We present first the notion of discrete random variable used to define a probability distribution, allowing us to define the generic and formal definition of histograms.

Random variables

Random variables have been introduced in order to describe phenomenons that have uncertain results, such as dice rolls. A random variable is described by statistical properties, allowing to expect the probabilities of results that can occur. Taking the example of the dice roll, let X be the random variable that reflects the result of a roll. All the possible values of X are $\Omega_X = \{1,2,3,4,5,6\}$.

Probability distribution

A probability distribution $p_X(x) = \Pr(X = x)$ represents the proportion of times where the random variable X is expected to take the value x. With the dice roll, if the game is fair, the probability distribution is denoted as uniform, meaning that any of the possible values have the same chance of occurring. The dice could also be loaded to increase the chance of getting sixes (incidentally reducing the chance of getting ones), resulting in a change in the probability distribution, as illustrated in Fig. 3.1.

Formally, a probability distribution will always possess these two properties: any value

not defined in the set of possible values Ω_X has no defined probability, and the sum of all the probabilities of the possible values is one:

$$\sum_{x \in \Omega_X} p_X(x) = 1 \tag{3.1}$$

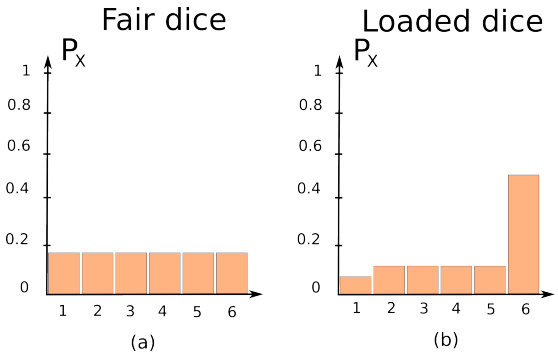


Figure 3.1: *Probability distribution function of a throw of 6-sided dice*

Analyzing an image through pixel probability distributions: uses of histograms

Our goal in this chapter is to use histograms to compare the probability distributions between the pixel arrays of images. In this section, we define the histogram in its generic form, as well as in the specific forms that we use, namely the histogram of gray-levels intensities, the color histogram and the histogram of oriented gradients.

Definition of an histogram: the gray-levels histogram example

A gray-levels histogram represents the statistical distribution of the pixel values in the image by associating each of those pixels to a given 'bin'. For example, the intensity histogram is expressed as:

$$\mathbf{p_I}(i) = \frac{1}{N_\mathbf{x}} \sum_{\mathbf{x}}^{N_\mathbf{x}} \delta(\mathbf{I}(\mathbf{x}) - i) \tag{3.2}$$

where $\mathbf{x}$ is a 2D pixel in the image plane, the pixel intensity $i \in [0,255]$ (Ω_X here) if we use images with 256 gray-levels, $N_\mathbf{x}$ the number of pixels in the image $\mathbf{I}(\mathbf{x})$, and $\delta(\mathbf{I}(\mathbf{x}) - i)$ the Kronecker's function defined such as:

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

$\mathbf{p_I}(i)$ is nothing but the probability for a pixel of the image $\mathbf{I}$ to have the intensity i.

According to the number of bins ($\text{card}(\Omega_X)$) selected to build the histogram, one can vary the resolution of the histogram (meaning here the granularity of the information conveyed). This is illustrated by Fig. 3.2, where the input image is Fig. 3.4(a) and two resulting histograms are generated from its distribution of gray-levels pixels: Fig. 3.2(b) is a representation of an histogram considering each pixel possibility (between 0 and 255) as an individual bin, whereas Fig. 3.2(c) shows an histogram with only 10 bins, meaning that all the pixel intensities of the image have been clustered into 10 categories. In this particular case, the bins has been quantified linearly, each bins containing a 1/10th of the pixel possible values. Performing this binning operation is a way to reduce the impact of local variations, making the representation less noisy, at the potential expense of ignoring important, less global information.

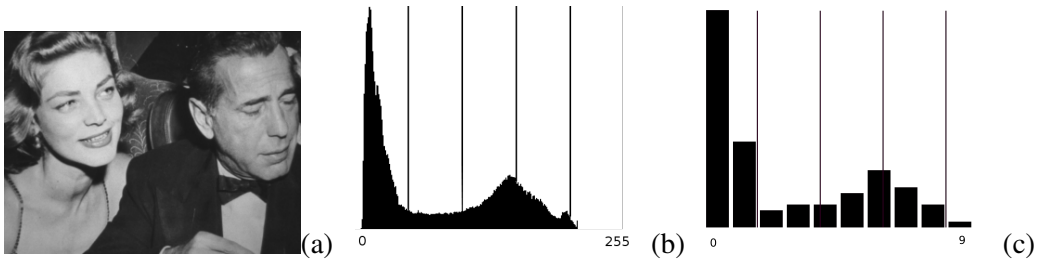


Figure 3.2: *Histogram representation: varying resolution through binning. (a) Raw data. Histogram representations: (b)* $\Omega_X = [0,255]$*, (c)* $\Omega_X = [0,9]$

Color Histogram

Since the intensity image is basically a reduced form of the color image, using a color histogram should yield more descriptive information of a scene than a gray-levels one. This descriptor has been widely used in several computer vision domains, such as indexing [Swain 92] [Jeong 04] or tracking [Comaniciu 03a] [Pérez 02]. The main interest of using color histograms over simple intensity histograms is that they capture more information and can therefore prove to be more reliable and versatile in some conditions. In particular, a way to exploit this additional layer of information is to define an histogram formulation that exploit solely the colorimetric information of an image and let out completely the illumination information. Indeed, in the case of VS tasks, i.e, controlling a camera in a real world environment, the illumination of the scene is often uncontrolled. In this case, shaping a descriptor to be insensitive to it is a

good way to gain additional robustness for the execution of the task.

3.1.2.2.1 Color space: from RGB to HSV. Traditionally in computer vision, the color information is represented through various representations, such as RGB, L*a*b or HSV. RGB is a very popular model, based on the three following components: the Red, Green and Blue. It is an additive color model, as mixing these three primary components in various respective amounts allows to create any color, as illustrated in Fig. 3.3(a). Although this model is perfectly adapted to the current hardware, one major drawback is that the geometry of this color space mingles many aspects of color that are easily interpreted by humans, such as the hue or the intensity. In order to get a more intuitive representation, several models where created, including the one we use here: the HSV color space.

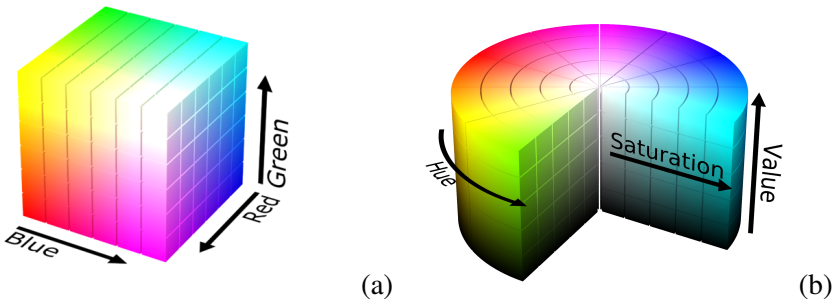


Figure 3.3: *Visual representations of color spaces. (a) RGB. (b) HSV*

As the RGB color model, the HSV color space is able to generate any color from 3 distinct components, but this time, the components have a more tangible interpretations: colors are defined by there Hue, Saturation and Value, as illustrated by Fig. 3.3(b). Two of these three components are linked to the color itself, the Hue and Saturation, the third component being associated only to the intensity of the color. In real environment situations, a uniformly colored, non-reflective texture under white light illumination of variable strength exhibits constant Hue and Saturation values, independently of the illumination changes, the latter affecting the only Value component.

3.1.2.2.2 HS Histogram In order to reduce the sensibility of the visual feature associated to color histograms regarding global illumination variations, we can choose here to focus more

on the color information and choose to set aside the intensity part of the image, as proposed in [Pérez 02]. We perform a change in the color space, from RGB (Red/Green/Blue) to HSV (Hue/Saturation/Value) which separates the color information (Hue and Saturation) from the pure intensity (Value). Since in this case the histogram has to be computed not only from one but two image planes, the expression becomes (as proposed in [Pérez 02], but taking out the 'Value' component):

$$\boldsymbol{p_I}(i) = \sum_{x}^{N_x} \delta(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i) \tag{3.4}$$

where $\mathbf{I}_H(\mathbf{x})$ and $\mathbf{I}_S(\mathbf{x})$ are respectively the Hue and Saturation image planes of the HSV color image from the camera, and $\delta(.)$ is the Kronecker's function.

As this histogram is computed from 2 distinct image planes, $card(\Omega_X)$ will be the squared value than for an histogram computed on 1 plane. Indeed, for an equal reduction of complexity on each plane (if they are reduced to n values possibilities), then for the color histogram $\Omega_X = [0, n * n]$, when for the single-plane image $\Omega_X = [0, n]$.

3.1.2.2.3 Generating a lightweight color histogram by combining H and S color planes.

In order to keep complexity low, an alternative method to consider color is to artificially create a single intensity plane that is based on color information instead of the intensity information. The control law associated to this feature would be the similar as the one used for the intensity histogram (Eq. (3.2)). Nevertheless it gains interesting properties in terms of increased robustness regarding global illumination changes, without the increase in computational cost induced by the feature described in the previous paragraph. To do so, we exploit the properties of the HSV color-space as in the previous part, and we exclude the Value component from the image. Following the same reasoning as in the design of the HS histogram, we choose here to use only the Hue and Saturation information, by creating the synthetic image following this protocol [Zhang 13]:

- computing the new image values as an average from the H and S planes, such as:

$$\mathbf{I}_{HS-Synth} = \frac{I_H(\mathbf{x}) + I_S(\mathbf{x})}{2} \tag{3.5}$$

- normalizing the new image values between 0 and 255.

We can then use this generated image as a standard intensity image and compute a gray-levels histogram on it, following Eq. (3.2), which becomes:

$$\mathbf{p_I}(i) = \frac{1}{N_\mathbf{x}} \sum_{\mathbf{x}}^{N_\mathbf{x}} \delta(\mathbf{I}_{HS-Synth}(\mathbf{x}) - i) \tag{3.6}$$

The difference being that this synthetic gray-levels image of a scene remains constant when recorded with two different lighting strength (provided that the scene is non-reflective and lit by a white light source), while the standard gray-levels histogram would see all its pixel values shifted by a change in illumination.

Histograms of Oriented Gradients

The histogram of oriented gradients (HOG) differs from the intensity histogram in the sense that it does not classify directly the values of the raw image pixels but relies on the computation of the norm and orientation of the gradient for each pixel. HOG has been introduced first by [Dalal 05] in order to define a new descriptor to detect humans in images. This descriptor proved very powerful and is now widely used for recognition purposes such as human faces [Déniz 11]. Relying solely on gradients can prove to be very beneficial since gradient orientation is invariant to illumination changes, either global or local, this property was previously used successfully in [Lowe 99].

The idea behind HOG is that any contour in an image can be described as having an orientation. Instead of generating the distribution of the pixels values, the HOG consists instead of a distribution of the orientation of all the image contours, weighted by the strength of the contour. This generates a descriptor that is only related to the texture of the image, mostly invariant to changes in illuminations, global or local, without white light assumption. Figure 3.4 illustrates the underlying idea behind the Histogram of Oriented Gradients. In order to get a gradient orientation for a given pixel $\mathbf{x}$, the following definition is used:

$$\theta(\mathbf{I}(\mathbf{x})) = \text{atan}\left(\frac{\nabla_y \mathbf{I}(\mathbf{x})}{\nabla_x \mathbf{I}(\mathbf{x})}\right) \tag{3.7}$$

where $\nabla_x \mathbf{I}(\mathbf{x})$ and $\nabla_y \mathbf{I}(\mathbf{x})$ are respectively the horizontal and vertical gradient values. The histogram of oriented gradient is then defined such as:

$$\boldsymbol{p}(i) = \frac{1}{N_x} \sum_{\mathbf{x}}^{N_x} \parallel \nabla \mathbf{I}(\mathbf{x}) \parallel \delta\left(\theta(\mathbf{I}(\mathbf{x})) - i\right) \tag{3.8}$$

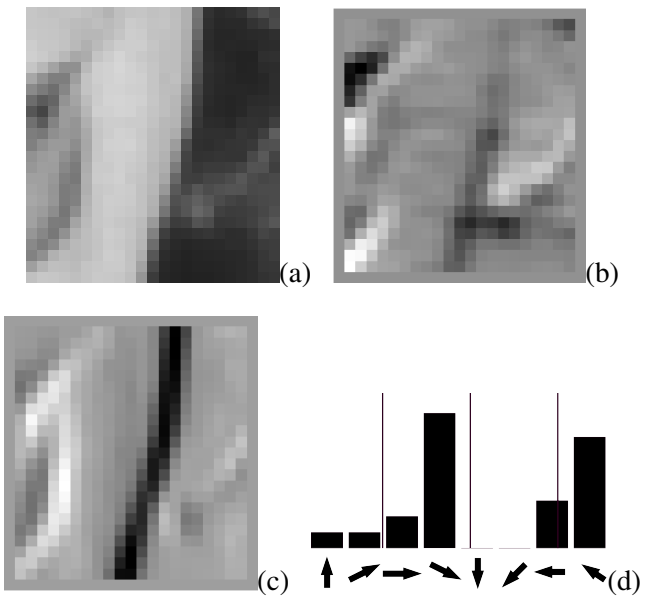


Figure 3.4: *From pixels to HOG. (a) Image sample. (b,c) Computing the image directional gradients: (b) horizontal component ∇_x ; (c) vertical component ∇_y. (d) Classifying the gradient orientations into categories creates the HOG (here with 8 bins)*

where $\| \nabla \mathbf{I}(\mathbf{x}) \|$ is the norm of the gradient, weighting out the contribution of the pixels of weaker gradient that are more prone to possess a less defined information (for example an uniform texture with some image noise will be a weak, randomly oriented gradient field).

Histograms as visual features

Defining a distance between histograms

As stated, visual servoing is the determination of the robot motion that allows to minimize an error between visual features in a current view and desired views of a scene. In order to perform this operation, it is first mandatory to be able to compare these two sets of visual features by defining a distance $\rho(.)$ between them. Since, here, images are described through histograms, we need to define a suitable distance to compare them. Classically, histograms can be compared bin-wise through the Bhattacharyya coefficient [Bhattacharyya 43] that measure

the similarity between two probability distributions. It is defined such as:

$$\rho_{\text{Bhattacharyya}}(\mathbf{I}, \mathbf{I}^*) = \sum_{i}^{N_c} \left(\sqrt{\mathbf{p}_{\mathbf{I}}(i) \mathbf{p}_{\mathbf{I}^*}(i)} \right) \tag{3.9}$$

where N_c is the number of bins considered in the histograms.

One undesirable property of this coefficient is that it does not represent a distance that would be null when the histograms are similar. Alternatively, a distance denoted as Matusita distance [Matusita 55] has been derived from the Bhattacharyya coefficient. It is expressed such as:

$$\rho_{\text{Matusita}}(\mathbf{I}, \mathbf{I}^*) = \sum_{i}^{N_c} \left(\sqrt{\mathbf{p}_{\mathbf{I}}(i)} - \sqrt{\mathbf{p}_{\mathbf{I}^*}(i)} \right)^2 \tag{3.10}$$

With this formulation, we are now able to compare two histograms such that driving the camera toward a position where the current camera view and the desired view will drive the Matusita distance to zero at the global minimum.

Improving the performance of the histogram distance through a multi-histogram strategy

As expected from previous works in computer vision using intensity histograms distances, eg, [Comaniciu 03b], preliminary testings showed that processing a single histogram on the whole image fails to have good properties since most of the spatial information is lost. In the tracking field, [Hager 04, Teuliere 11] proposes to extend the sensibility of histogram-based method to more DOF through the use of multiple histograms throughout the image. Here we adapt this idea by equally dividing the image in multiple areas and by associating a histogram to each area. Regarding the effect of the multiple histograms approach on the cost function, Fig. 3.6(a) shows the mapping of the cost function in the case where we use only one histogram on the whole image to calculate our cost function. It can be seen that even for the two DOF t_x and t_y, there is no clear minimum, preventing the use of optimization methods to find the desired pose. On the other hand, if we compute 25 histograms by separating evenly the images, compute for each respective histogram couples a distance and sum each of those distances into a global one, this total cost function features a clear minimum, with a large convex area, as seen on Fig. 3.6(b).

Figure 3.5: *Illustration of multi-histogram approach with 9 areas*

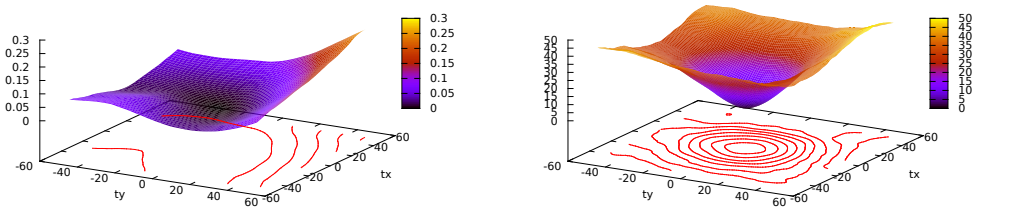


Figure 3.6: *(a) Cost function with a single histogram. (b) Cost function with 25 histograms*

Empirical visualization and evaluation of the histogram distance as cost function

In this section, we propose a process to evaluate the quality of histogram visual features as an alignment function.

Methodology

This evaluation process is performed by considering a nominal image $\mathbf{I}^*$, and creating from it a set of warped images $\mathbf{I}$, each representing the same scene as $\mathbf{I}^*$ but warped along the vertical and horizontal axis. These translational motions are applied such as to create a regular grid of shifted images, farther and farther from the reference image along both axis. This warp transformation is defined such as each pixel of coordinates $\mathbf{x}_1$ in the reference image $\mathbf{I}^*$ is transferred at coordinates $\mathbf{x}_2$ in the resulting image $\mathbf{I}_n$ such as:

$$\mathbf{x}_2 = w(\mathbf{x}_1, \mathbf{h}) \tag{3.11}$$

where $\mathbf{h}$ is a set of transformation parameters that can represent a simple translation, an affine motion model, an homography... In our case, since we work on a planar image, it is possible to link the coordinates of $\mathbf{x}_1$ and $\mathbf{x}_2$ thanks to a homography ${}^2\mathbf{H}_1$ such that

$$\mathbf{x}_2 = w(\mathbf{x}_1, \mathbf{h}) = {}^2\mathbf{H}_1 \mathbf{x}_1 \tag{3.12}$$

with

$${}^2\mathbf{H}_1 = ({}^2\mathbf{R}_1 + \frac{{}^2\mathbf{t}_1}{{}^1d} {}^1\mathbf{n}^\top) \tag{3.13}$$

where ${}^1\mathbf{n}$ and ${}^1d$ are the normal and distance to the origin of the reference plane expressed in the reference camera frame such as ${}^1\mathbf{n}\mathbf{X} = {}^1d$.

From this set of warped images, we compute for each image the value of the distance between the histogram computed on this image and the histogram computed on I^*. We can then display this set of values on a 3D graph, with the position shift values on the X and Y axis, and the value of the distance (effectively a cost function here) on the Z axis.

Fig. 3.7 illustrates the generation of such a cost function visualization. The generated cost function represents the ability for the corresponding descriptor to link the amount of shared information between two images according to the considered camera motion (approximated

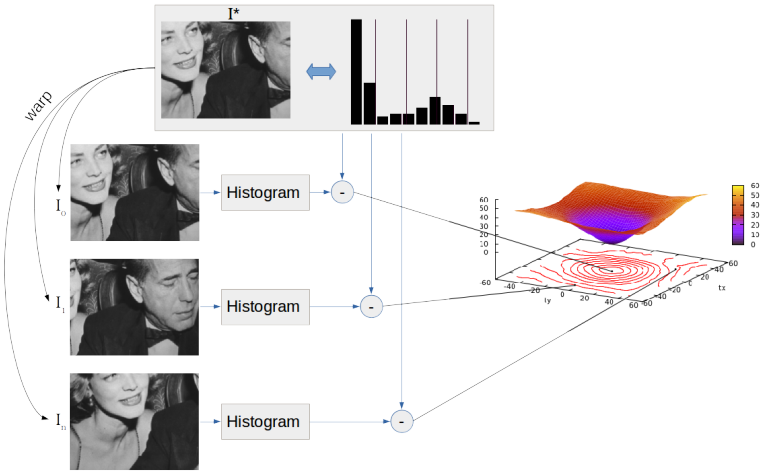


Figure 3.7: *Visualizing the cost function associated to an histogram distance along 2 DOF (here the 2 translations tx/ty in the image plane). Here, we show the computation of three cost function values, associated with different shift of the current image, compared to the reference image.*

here through the warp function). In VS terms, we want a cost function that is correlated as linearly as possible to the camera motion, in order to apply successfully the minimization scheme presented in the previous chapter. Visually, desirable properties for a cost function are translated as a wide convex area that represents the working space of the VS task, as well as a smooth surface in that convex space, showing no local minima.

Comparison of the histogram distances and parameter analysis

Using the methodology presented above, we compare empirically the cost functions generated by the following visual features: the raw intensity array (compared through the SSD, as performance baseline), the gray-levels histogram, the HS color histogram, the gray-levels histogram computed on an image synthesized from the H and S planes of a color image, and the Histogram of Oriented Gradients (HOG). First in nominal conditions, with fixed illumination and constant image quality, then with various perturbations. This will give us insights of how we can expect the features to perform in a robotic positioning task in a real environment (nominal conditions give the best expected results, and perturbed conditions highlight the potential robustness to external changes in the environment). It will also help us determine the impact of the various parameters involved, such as the number of bins per histogram, as well as the

number of histograms per image.

It is important to note that the resulting cost function visualizations will be affected by the choice of the reference image's appearance (performed here on a single test image). Hence, the analysis performed here is purely qualitative, in order to visualize potential trends in the cost function changes. This information will ease the parameter selection that will have to performed when setting up experiments in real scenes and leads to better performances.

3.2.3.2.1 Comparison of the methods in nominal conditions According to the following figures (Figs. 3.8, 3.9, 3.10, 3.11, 3.12, corresponding to the cost function visualization associated to, respectively, the SSD, the gray-levels histogram, the HS histogram and the HOG), we can see that the cost function shapes are greatly affected by the variation of the two parameters (number of bins and number of histograms), and a trend can be observed: the increase in the number of bins tends to sharpen the cost function making it more precise around the global minimum (also increasing the computational complexity), and the increase in the number of histograms tends to improve the convexity and sharpness, although in many cases, it also decreases the radius of the convex zone.

A resulting general rule for choosing an initial set of parameters is the following: for the number of bins, a noticeable improvement is seen by increasing it to at least 32, increasing it further yielding diminishing returns comparing to the increase in computational cost. For the number of histograms, choosing 4x4 histograms allows generally for a good trade-off between a large convex area and a good definition of the minimum, a lower number degrading the minimum noticeably, and a higher number reducing the convex area radius and increasing the computational cost. While comparing the 4 methods with each others, we can see that with a

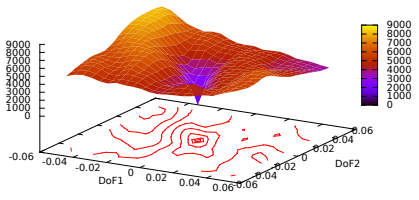


Figure 3.8: *SSD-based cost function*

proper choice of parameters, all 4 proposed methods can exhibit a wider convergence radius

for these two DOF (tx and ty) than the classical SSD-based direct VS method. This confirms the preliminary intuition leading to the development of this class of methods that considering globally the image statistical properties could yield a better representation. It can also be seen that the HOG-based method seems to exhibit a cost function of slightly less quality, showing a more restricted convex area than the other three.

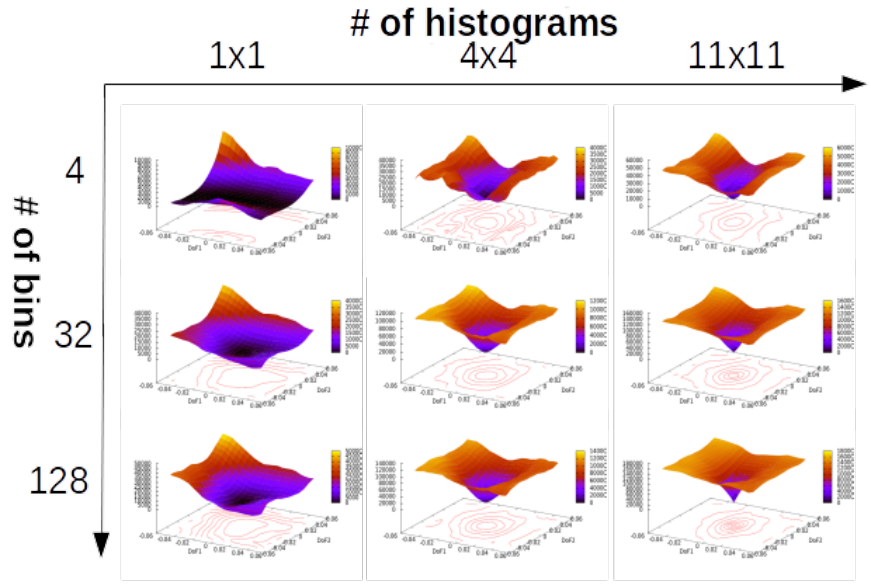


Figure 3.9: *Gray-levels histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

3.2.3.2.2 Comparison of the methods in perturbed conditions

3.2.3.2.2.1 Illumination perturbation. In this section, the illumination in the reference image is altered before computing the cost functions, simulating a change in the scene illumination (the hypothesis of white light source illumination is kept, as no chromatic changes are introduced). We can see that this perturbation has severe impacts on our cost functions. Fig. 3.13 displays the SSD cost function, and we see that this direct pixel-wise comparison is not robust to this type of perturbation, as the convex area is reduced to a very limited motion range. The gray-levels histogram cost function (Fig. 3.14) is also strongly affected, the global minimum disappearing as well as the convexity property, independently of the set of parameters chosen. It is interesting to compare it to the cost function generated by the synthesized gray-levels plane (Fig. 3.16): through minimal computational overhead, the visual

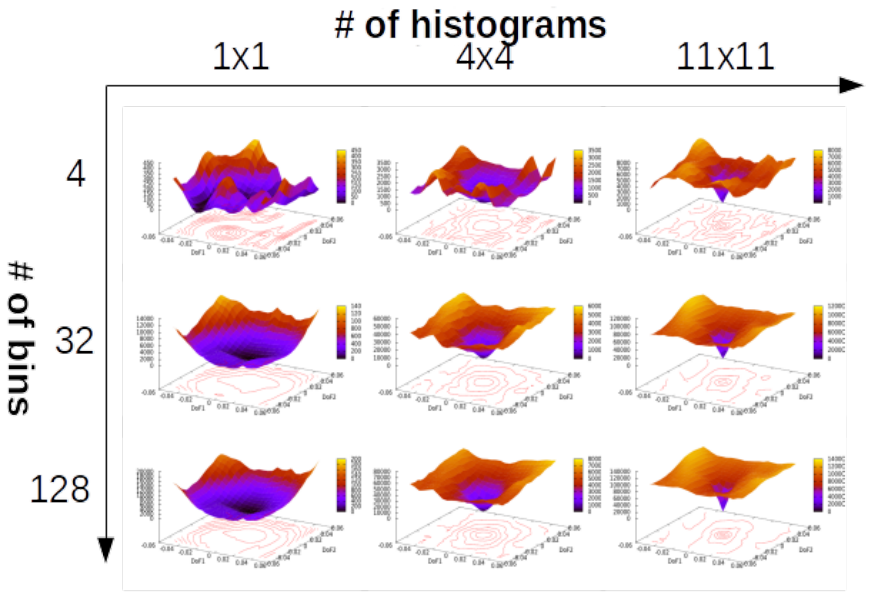


Figure 3.10: *HS histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

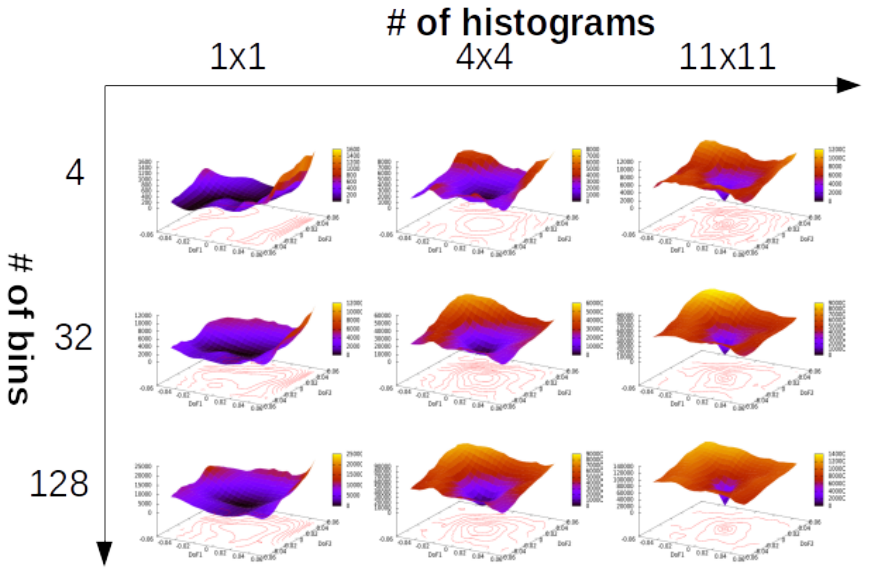


Figure 3.11: *Gray-levels on HS-synthesized plane histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

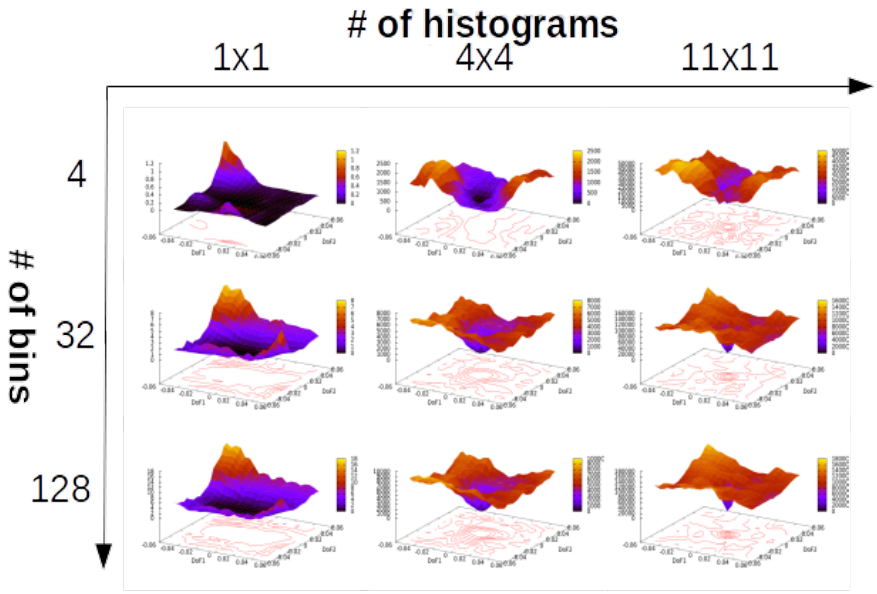


Figure 3.12: *HOG-based cost function visualization with varying number of bins and varying number of histograms used in the image*

feature exhibits improved characteristics, since a clear global minimum exists, as well as a limited convex area. The cost function with the most desirable properties are achieved by the HS-based method (Fig. 3.15) that shows good convex areas and sharp global minimum for at least a given set of parameters. The HOG method (Fig. 3.17) performs second, keeping good properties overall, but exhibiting a smaller convex area than the HS-based method.

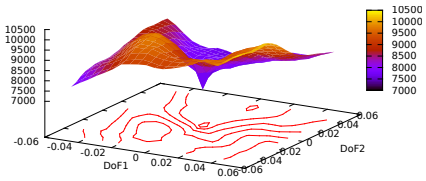


Figure 3.13: *SSD-based cost function*

3.2.3.2.2.2 Occlusion perturbation. In this part, we are computing the cost function in the same way, while modifying the current image in order to add an external object to the current image, creating a discrepancy with respect to the reference image. We can see on Fig. 3.18 that the SSD-based method is somehow robust to this perturbation.

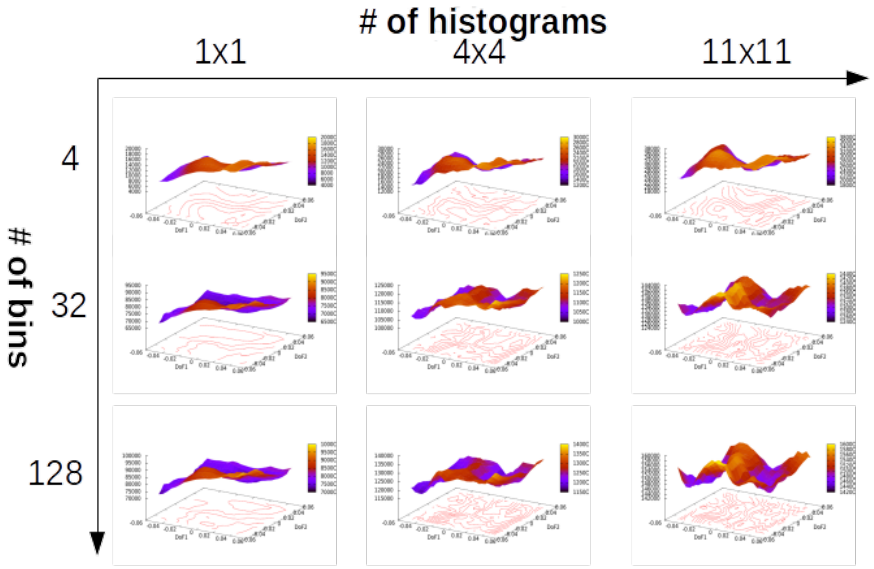


Figure 3.14: *Gray-levels histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

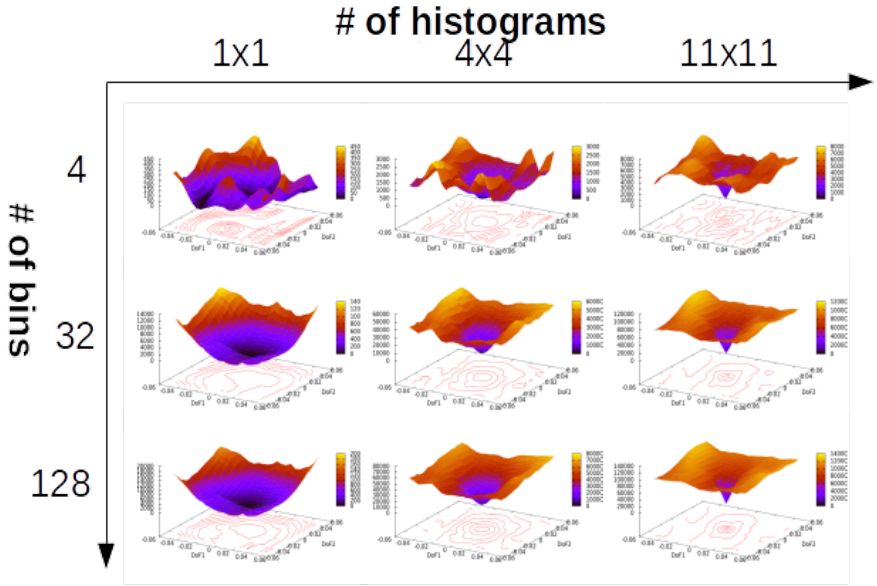


Figure 3.15: *HS histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

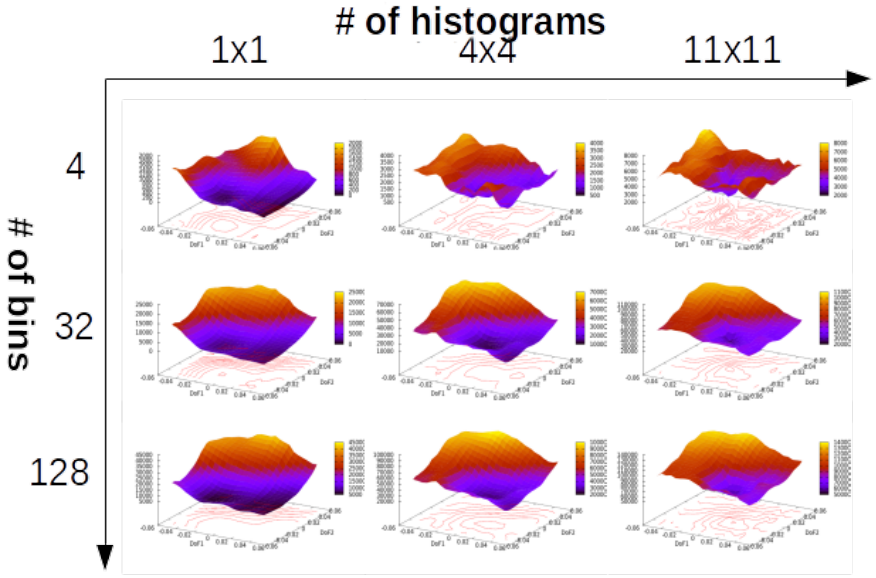


Figure 3.16: *Gray-levels on HS-synthesized plane histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

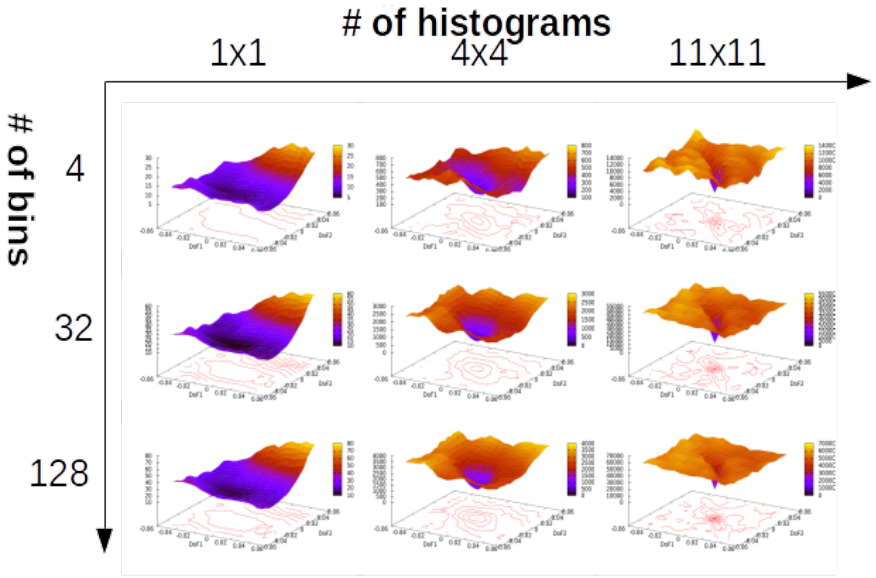


Figure 3.17: *HOG-based cost function visualization with varying number of bins and varying number of histograms used in the image*

The gray-levels and color-based methods (Fig. 3.19, 3.20 and 3.21) all exhibit a clear degradation when the number of histograms is low, but are robust to the perturbation when this parameter is higher.

The HOG-based method (Fig. 3.22) shows a lack of robustness to this perturbation, the convex area shrinking in almost every settings.

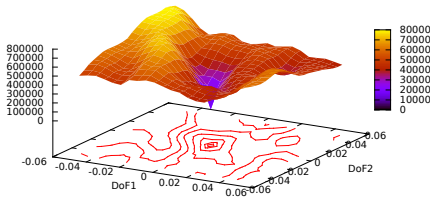


Figure 3.18: *SSD-based cost function*

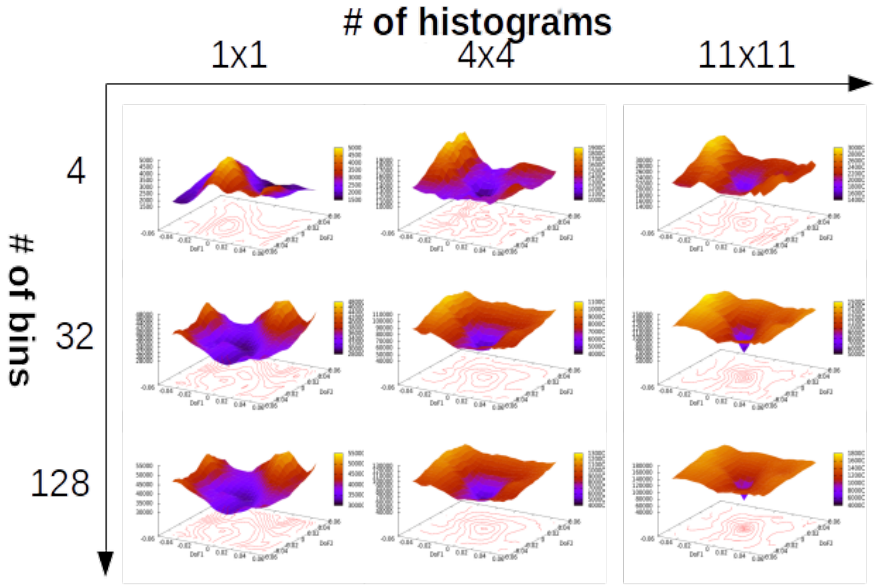


Figure 3.19: *Gray-levels histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

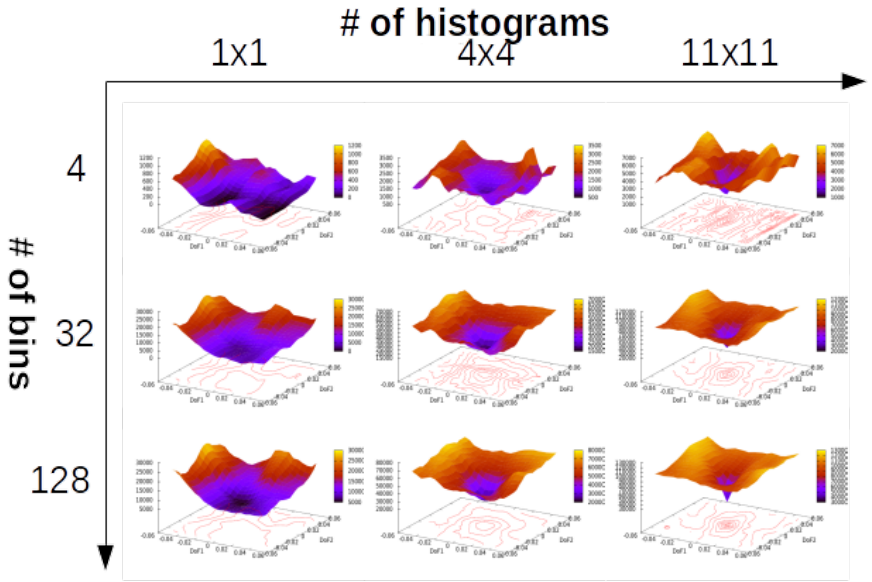


Figure 3.20: *HS histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

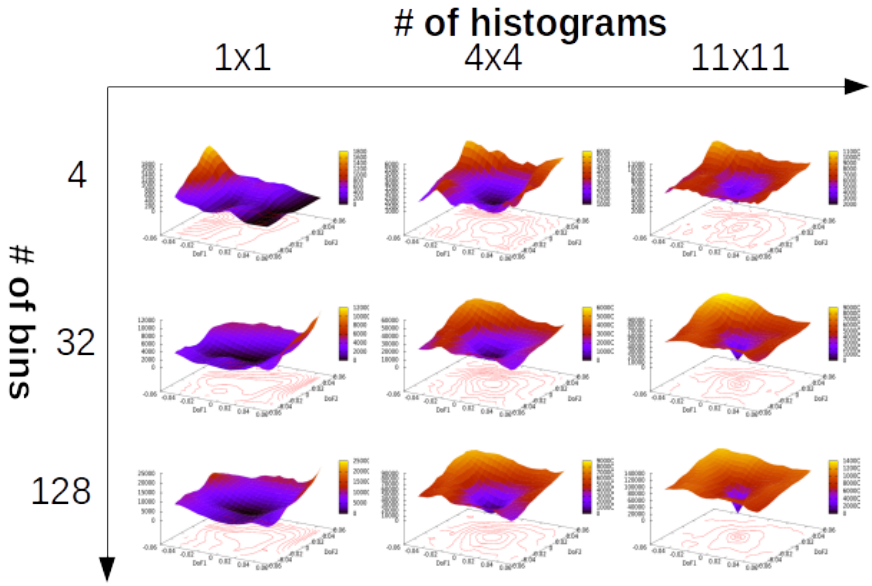


Figure 3.21: *Gray-levels on HS-synthesized plane histogram-based cost function visualization with varying number of bins and varying number of histograms used in the image*

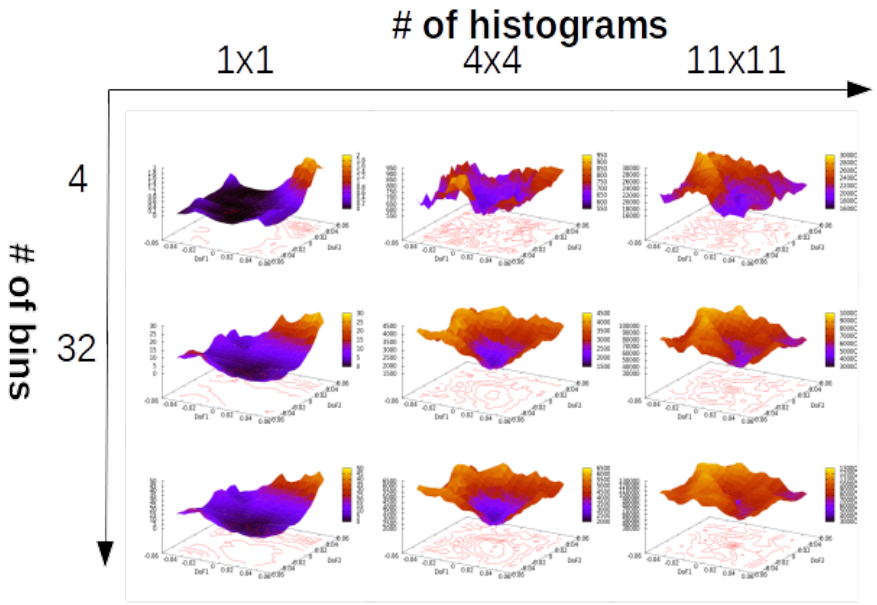


Figure 3.22: *HOG-based cost function visualization with varying number of bins and varying number of histograms used in the image*

Conclusion

After analyzing the shapes of the various cost functions associated to the proposed methods, all of them feature improved properties (provided that the parameters are set appropriately) over the SSD-based method on nominal conditions. Under lighting perturbations the color and HOG-based methods exhibit similar qualities. Under occlusions, all methods but the HOG-based method keep exhibiting a clear improvement with respect to the SSD-based method.

This preliminary analysis seems to confirm that the ability of using histograms to compare images can allow us to outperform the classical direct visual servoing method, and give us multiple options to choose from, depending on the perturbations that can be expected to occur in the target environment.

Designing a histogram-based control law

In this section, we propose a generic analytic formulation for a VS control law based on the Matusita distance. From there, we will apply this generic method to all four methods studied in the previous part.

Control law and generic interaction matrix for histograms

Based on Eq. (2.8), the control law based on the Matusita distance is expressed as:

$$\mathbf{v} = -\lambda \mathbf{L}_H^+ \, \rho(\mathbf{I}, \mathbf{I}^*) \tag{3.14}$$

where $\rho(\mathbf{I}, \mathbf{I}^*)$ is the cost function associated with the Matusita distance between histograms computed in the reference image and a current image (see Eq.(3.10)), and $\mathbf{L}_H$ is the interaction matrix that links the variation of the visual features to the camera motion [Chaumette 06]. This definition leads to the generic expression of the interaction matrix, $\mathbf{L_s} = \frac{\partial \boldsymbol{s}}{\partial \mathbf{r}}$, where $\mathbf{r}$ is the camera position. In our context, it links the variation of the cost function ρ to the camera motion. It is then defined by:

$$\mathbf{L}_H = \frac{\partial \rho(\mathbf{I}, \mathbf{I}^*)}{\partial \mathbf{r}} \tag{3.15}$$

As $\rho(\mathbf{I}, \mathbf{I}^*)$ obviously depends on the nature of the histogram considered, each being defined by its probability density descriptor $\mathbf{p_I}(i)$. We can then develop this expression and obtain, using the expression defined in Eq. (3.10):

$$\begin{aligned} \frac{\partial \rho(\mathbf{I}, \mathbf{I}^*)}{\partial \mathbf{r}} &= \frac{\partial}{\partial \mathbf{r}} \left[\sum_i^{N_c} \left(\sqrt{\boldsymbol{p}_\mathbf{I}(i)} - \sqrt{\boldsymbol{p}_{\mathbf{I}^*}(i)} \right)^2 \right] \\ &= 2N_c \sum_i^{N_c} \left(\frac{\partial}{\partial \mathbf{r}} \sqrt{\boldsymbol{p}_\mathbf{I}(i)} \left(\sqrt{\boldsymbol{p}_\mathbf{I}(i)} - \sqrt{\boldsymbol{p}_{\mathbf{I}^*}(i)} \right) \right) \end{aligned} \tag{3.16}$$

with

$$\frac{\partial}{\partial \mathbf{r}} \sqrt{\boldsymbol{p}_\mathbf{I}(i)} = \frac{1}{2\sqrt{\boldsymbol{p}_\mathbf{I}(i)}} \frac{\partial \boldsymbol{p}_\mathbf{I}(i)}{\partial \mathbf{r}} \tag{3.17}$$

The expression then becomes:

$$\frac{\partial \rho(\mathbf{I}, \mathbf{I}^*)}{\partial \mathbf{r}} = 2N_c \sum_i^{N_c} \left(\frac{\partial \boldsymbol{p}_\mathbf{I}(i)}{\partial \mathbf{r}} \left(\frac{\sqrt{\boldsymbol{p}_\mathbf{I}(i)} - \sqrt{\boldsymbol{p}_{\mathbf{I}^*}(i)}}{2\sqrt{\boldsymbol{p}_\mathbf{I}(i)}} \right) \right) \tag{3.18}$$

Finally, we get this generic expression of the interaction matrix required to design histogram-based control laws:

$$\mathbf{L}_H = N_c \sum_{i}^{N_c} \left(\frac{\partial \boldsymbol{p}_\mathbf{I}(i)}{\partial \mathbf{r}} \left(1 - \frac{\sqrt{\boldsymbol{p}_{\mathbf{I}^*}(i)}}{\sqrt{\boldsymbol{p}_\mathbf{I}(i)}} \right) \right) \tag{3.19}$$

Solving the histogram differentiability issue with B-spline approximation

From the definition of the interaction matrix in Eq. (3.19), the Kronecker's function $\delta(.)$ used in Eq. (3.2) poses an issue because of its non-differentiability. In order to compute the derivative of the error function, the histograms are approximated through second-order centered cardinal B-Splines, as in [Dame 11]. This is illustrated in Fig. 3.23(a)

A thorough description of B-Splines is given in [Unser 93]. For our purpose, the most interesting properties are the following: the integral of a B-Spline function is 1, so that the result does not have to be re-normalized, and the computation of its derivatives is easily obtained. As a B-Spline of order n is $n-1$ times differentiable and we need to derive our expression once, so we select a B-Spline of second order, defined such as:

$$\phi_2(t) = \begin{cases} t+1 & \text{if } t \in [-1,0[\\ -t+1 & \text{if } t \in [0,1[\\ 0 & \text{else} \end{cases} \tag{3.20}$$

The derivative of a B-Spline of n-th order is given by the following expression:

$$\frac{\partial \phi_n(t)}{\partial t} = \phi_{n-1}(t+\frac{1}{2}) - \phi_{n-1}(t-\frac{1}{2}) \tag{3.21}$$

We now only need the expression of the first order B-Spline, given by:

$$\phi_1(t) = \begin{cases} 1 & \text{if } t \in [-0.5,0.5[\\ 0 & \text{else} \end{cases} \tag{3.22}$$

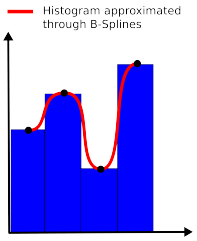


Figure 3.23: *Smoothing and approximating a histogram by using a B-spline*

Based on Eq. (3.2), that is:

$$\mathbf{p}_{\mathbf{I}}(i) = \frac{1}{N_{\mathbf{x}}} \sum_{\mathbf{x}}^{N_{\mathbf{x}}} \delta(\mathbf{I}(\mathbf{x}) - i)$$

this technique yields the following expression for the intensity histogram :

$$\mathbf{p}_{\mathbf{I}}(i) = \frac{1}{N_{\mathbf{x}}} \sum_{\mathbf{x}}^{N_{\mathbf{x}}} \phi(\bar{\mathbf{I}}(\mathbf{x}) - i) \tag{3.23}$$

where $N_{\mathbf{x}}$ is the number of pixels, $\mathbf{x} = \mathbf{x}, \mathbf{y}$ a single image pixel, $\phi(.)$ a B-spline differentiable once and $\bar{\mathbf{I}}$ the image reduced to N_c intensity values (each histogram then contains N_c bins). For images originally expressed with 255 possible intensity levels, we have:

$$\bar{\mathbf{I}}(\mathbf{x}) = \frac{N_c}{255} \mathbf{I}(\mathbf{x}) \tag{3.24}$$

Integrating the multi-histogram strategy into the interaction matrix

It is crucial to note that $\mathbf{L}_H$, defined in Eq. (3.19), being of size 1×6 (the error is based one single distance), it allows us to control only 1 DOF when integrated into a control scheme because the minimization problem is under-constrained. This poses an important issue since our goal is to control at least 6 DoF. Therefore an extension to this method had to be developed in order to extend the control to more DOF, through the use of a multi-histogram strategy.

Building on the Matusita cost function $\rho_{Matusita}(\mathbf{I}, \mathbf{I}^*)$ defined by Eq. (3.10), the cost function corresponding to a distance computed over n histograms in the image is given by:

$$\rho_{multiple}(\mathbf{I}, \mathbf{I}^*) = \begin{bmatrix} \rho_{Matusita}(\mathbf{I}(1), \mathbf{I}^*(1)) & \rho_{Matusita}(\mathbf{I}(2), \mathbf{I}^*(2)) & \dots & \rho_{Matusita}(\mathbf{I}(n), \mathbf{I}^*(n)) \end{bmatrix}^\top \tag{3.25}$$

where $\mathbf{I}(n)$ and $\mathbf{I}^*(n)$ are respectively the current subdivision of $\mathbf{I}$ and $\mathbf{I}^*$ on which the n-th couple of histograms is computed.

As expressed on Eq. (3.14), the control law is originally defined such as:

$$\mathbf{v} = -\lambda \mathbf{L}_H^+ \rho(\mathbf{I}, \mathbf{I}^*)$$

By replacing the error term $\rho(\mathbf{I}, \mathbf{I}^*)$ by $\rho_{multiple}(\mathbf{I}, \mathbf{I}^*)$ of dimensions $n \times 1$. To keep coherent the dimensionality, the interaction matrix also needs to be of size $n \times 6$, defined such as:

$$\mathbf{L}_H = \begin{bmatrix} \mathbf{L}_{H_1} & \mathbf{L}_{H_2} & \dots & \mathbf{L}_{H_n} \end{bmatrix}^\top \tag{3.26}$$

where $\mathbf{L}_{H_n}$ is the interaction matrix given by the n-th histogram distance.

By using 6 or more histograms on the image, it then becomes possible to control every six degrees of freedom of the robot, the minimization problem of Eq. (3.14) no longer being under-constrained.

Computing the full interaction matrices

From the generic expression Eq. (3.19) of the interaction matrix corresponding to a single histogram, we can now instantiate the control laws of the presented histograms.

Single plane histogram interaction matrix

As shown in the previous part, the interaction matrix can be seen as the Jacobian of the error vector according to the variation of the camera pose $\mathbf{r}$. This definition leads to the following expression: $\mathbf{L}_\rho = \frac{\partial \rho(\mathbf{I}, \mathbf{I}^*)}{\partial \mathbf{r}}$ where $\mathbf{I}^*$ and $\mathbf{I}$ are the images recorded at the desired pose and at the current pose respectively, and $\rho(\mathbf{I}, \mathbf{I}^*)$ the Matusita distance between the two histograms.

Using a distance between gray-levels histograms as cost function, we recall that in this case:

$$\rho(\mathbf{I}, \mathbf{I}^*) = \sum_i^{N_c} \left(\sqrt{\mathbf{p}_\mathbf{I}(i)} - \sqrt{\mathbf{p}_{\mathbf{I}^*}(i)} \right)^2$$

(see Eq. (3.10)), with $\mathbf{p_I}(i)$ defined by Eq. (3.23), such as:

$$\mathbf{p_I}(i) = \frac{1}{N_\mathbf{x}} \sum_{\mathbf{x}}^{N_\mathbf{x}} \phi(\bar{\mathbf{I}}(\mathbf{x}) - i)$$

Using chain rule derivation, we can obtain the following expressions:

$$\mathbf{L}_\rho = \frac{\partial}{\partial \mathbf{r}} \left[\sum_{\mathbf{i}}^{\mathbf{N_c}} \left(\sqrt{\mathbf{p}_{\bar{\mathbf{I}}}(\mathbf{i})} - \sqrt{\mathbf{p}_{\bar{\mathbf{I}}^*}(\mathbf{i})} \right)^2 \right] = 2 \sum_{\mathbf{i}}^{\mathbf{N_c}} \left(\frac{\partial}{\partial \mathbf{r}} \sqrt{\mathbf{p}_{\bar{\mathbf{I}}}(\mathbf{i})} \left(\sqrt{\mathbf{p}_{\bar{\mathbf{I}}}(\mathbf{i})} - \sqrt{\mathbf{p}_{\bar{\mathbf{I}}^*}(\mathbf{i})} \right) \right) \tag{3.27}$$

with

$$\frac{\partial}{\partial \mathbf{r}} \sqrt{\mathbf{p}_{\bar{\mathbf{I}}}(i)} = \frac{1}{2\sqrt{\mathbf{p}_{\bar{\mathbf{I}}}(i)}} \frac{\partial \mathbf{p}_{\bar{\mathbf{I}}}(i)}{\partial \mathbf{r}} \tag{3.28}$$

where

$$\frac{\partial \mathbf{p}_{\bar{\mathbf{I}}}(i)}{\partial \mathbf{r}} = \frac{\partial}{\partial \mathbf{r}} \left(\sum_{\mathbf{x}}^{N_\mathbf{x}} \left(\phi(\bar{\mathbf{I}}(\mathbf{r}, \mathbf{x}) - i) \right) \right) = \sum_{\mathbf{x}}^{N_\mathbf{x}} \left(\frac{\partial}{\partial \mathbf{r}} \left(\phi(\bar{\mathbf{I}}(\mathbf{r}, \mathbf{x}) - i) \right) \right) \tag{3.29}$$

and from [Dame 11],

$$\begin{aligned} \frac{\partial}{\partial \mathbf{r}} \left(\phi \left(\bar{\mathbf{I}}(\mathbf{r}, \mathbf{x}) - i \right) \right) &= \frac{\partial}{\partial i} \left(\phi \left(\bar{\mathbf{I}}(\mathbf{r}, \mathbf{x}) - i \right) \right) \frac{\partial \bar{\mathbf{I}}(\mathbf{r}, \mathbf{x})}{\partial \mathbf{r}} \\ &= \frac{\partial}{\partial i} \left(\phi \left(\bar{\mathbf{I}}(\mathbf{r}, \mathbf{x}) - i \right) \right) \mathbf{L}_{\bar{\mathbf{I}}} \end{aligned} \tag{3.30}$$

with, from [Collewet 08b], the interaction matrix that links the variation of the image intensity to the camera velocity is given by:

$$\mathbf{L}_{\bar{\mathbf{I}}} = -\left(\nabla_\mathrm{x} \bar{\mathbf{I}} \mathbf{L}_\mathrm{x} + \nabla_\mathrm{y} \bar{\mathbf{I}} \mathbf{L}_\mathrm{y} \right) \tag{3.31}$$

where $\mathbf{L}_\mathrm{x}$ and $\mathbf{L}_\mathrm{y}$ are the lines corresponding to the x and y-coordinates of $\mathbf{L_x} = \begin{pmatrix} \mathbf{L}_\mathrm{x} \\ \mathbf{L}_\mathrm{y} \end{pmatrix}$ is the interaction matrix corresponding to a single 2D point $\mathbf{x} = (x, y)$ in the image, and is defined in [Chaumette 06] as:

$$\mathbf{L_x} = \begin{bmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix} \tag{3.32}$$

Finally, after simplifying Eq. (3.27), we get the following $\mathbf{L}_\rho$ interaction matrix for our control law:

$$\mathbf{L}_\rho = \sum_{\mathbf{i}}^{\mathbf{N_c}} \left(\frac{\mathbf{1}}{\mathbf{N_x}} \left(\mathbf{1} - \frac{\sqrt{\mathbf{p}_{\bar{\mathbf{I}}^*}(\mathbf{i})}}{\sqrt{\mathbf{p}_{\bar{\mathbf{I}}}(\mathbf{i})}} \right) \sum_{\mathbf{x}}^{\mathbf{N_x}} \left(\frac{\partial}{\partial \mathbf{i}} \left(\phi(\bar{\mathbf{I}}(\mathbf{r}, \mathbf{x}) - \mathbf{i}) \right) \mathbf{L}_{\bar{\mathbf{I}}} \right) \right) \tag{3.33}$$

This resulting interaction matrix can be applied on any gray-levels image, meaning that this interaction matrix can be applied both to the standard gray-levels images, as well as to an image computed from a combination of Hue and Saturation values, using Eq. (3.5).

HS histogram interaction matrix

From the expression of the histogram-based interaction matrix of Eq. (3.19), we only need to compute the derivative of expression of Eq. (3.4) to obtain an expression that can be computed directly from the image values, as:

$$\boldsymbol{p_I}(i) = \sum_{\mathrm{x}}^{N_\mathrm{x}} \left(\phi(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i)\right)$$

To do so, we apply derivation chain rules:

$$\frac{\partial}{\partial \boldsymbol{r}}\left(\boldsymbol{p_I}(i)\right) = \sum_{\mathrm{x}}^{N_\mathrm{x}} \left(\frac{\partial}{\partial \boldsymbol{r}} \left(\phi(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i)\right)\right) \tag{3.34}$$

As in the previous section, we apply the following development:

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{r}} \left(\phi(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i)\right) \\
&= \frac{\partial}{\partial i} \left(\phi(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i)\right) \frac{\partial}{\partial \boldsymbol{r}} \left(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x})\right) \\
&= \frac{\partial}{\partial i} \left(\phi(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i)\right) \left(\frac{\partial}{\partial \boldsymbol{r}} \left(\mathbf{I}_H(\mathbf{x})\right) \mathbf{I}_S(\mathbf{x}) + \mathbf{I}_H(\mathbf{x}) \frac{\partial}{\partial \boldsymbol{r}} \left(\mathbf{I}_S(\mathbf{x})\right)\right) \\
&= \frac{\partial}{\partial i} \left(\phi(\mathbf{I}_H(\mathbf{x})\mathbf{I}_S(\mathbf{x}) - i)\right) \left(\mathbf{L_H}\mathbf{I}_S(\mathbf{x}) + \mathbf{I}_H(\mathbf{x})\mathbf{L_S}\right)
\end{aligned} \tag{3.35}$$

where $\mathbf{L_H}$ and $\mathbf{L_S}$ are (from [Collewet 08b]):

$$\begin{aligned}
\mathbf{L_H} &= -(\nabla_x \mathbf{I_H}(\mathbf{x})\mathbf{L}_x + \nabla_y \mathbf{I_H}(\mathbf{x})\mathbf{L}_y) \\
\mathbf{L_S} &= -(\nabla_x \mathbf{I_S}(\mathbf{x})\mathbf{L}_x + \nabla_y \mathbf{I_S}(\mathbf{x})\mathbf{L}_y)
\end{aligned} \tag{3.36}$$

Finally:

$$\frac{\partial}{\partial \boldsymbol{r}}\left(\boldsymbol{p_I}(i)\right) = \sum_{\mathrm{x}}^{N_\mathrm{x}} \left(\frac{\partial}{\partial i} \left(\phi(\mathbf{I_H}(\mathbf{x})\mathbf{I_S}(\mathbf{x}) - i)\right) \left(\mathbf{L_H}(\mathbf{x})\mathbf{I_S}(\mathbf{x}) + \mathbf{I_H}(\mathbf{x})\mathbf{L_S}(\mathbf{x})\right)\right) \tag{3.37}$$

HOG interaction matrix

In the same way as the previous color histogram method, we start directly by computing the derivative of the cost function of Eq. (3.8):

$$\boldsymbol{p}(i) = \frac{1}{N_x} \sum_{\mathbf{x}}^{N_x} \| \nabla \mathbf{I}(\mathbf{x}) \| \, \delta \left(\theta(\mathbf{I}(\mathbf{x})) - i \right)$$

which leads to:

$$\begin{aligned} \frac{\partial \boldsymbol{p}_{\mathbf{I}}(i)}{\partial \mathbf{r}} &= \frac{\partial}{\partial \mathbf{r}} \left(\frac{1}{N_x} \sum_{\mathbf{x}}^{N_x} \| \nabla \mathbf{I}(\mathbf{x}) \| \, \phi \left(\theta(\mathbf{I}(\mathbf{x})) - i \right) \right) \\ &= \frac{1}{N_x} \sum_{\mathbf{x}}^{N_x} \left(\frac{\partial}{\partial \mathbf{r}} \left(\| \nabla \mathbf{I}(\mathbf{x}) \| \right) \phi(\theta(\mathbf{I}(\mathbf{x})) - i) + \| \nabla \mathbf{I}(\mathbf{x}) \| \, \frac{\partial}{\partial \mathbf{r}} \left(\phi(\theta(\mathbf{I}(\mathbf{x})) - i) \right) \right) \end{aligned} \tag{3.38}$$

where

$$\begin{aligned} \frac{\partial}{\partial \mathbf{r}} \left(\| \nabla \mathbf{I}(\mathbf{x}) \| \right) &= \frac{\partial}{\partial \mathbf{r}} \left(\sqrt{(\nabla_x \mathbf{I}(\mathbf{x}))^2 + (\nabla_y \mathbf{I}(\mathbf{x}))^2} \right) \\ &= \frac{\left(2 \nabla_x \mathbf{I}(\mathbf{x}) \frac{\partial}{\partial \mathbf{r}} \left(\nabla_x \mathbf{I}(\mathbf{x}) \right) + 2 \nabla_y \mathbf{I}(\mathbf{x}) \frac{\partial}{\partial \mathbf{r}} \left(\nabla_y \mathbf{I}(\mathbf{x}) \right) \right)}{2 \sqrt{(\nabla_x \mathbf{I}(\mathbf{x}))^2 + (\nabla_y \mathbf{I}(\mathbf{x}))^2}} \\ &= \frac{\nabla_x \mathbf{I}(\mathbf{x}) \frac{\partial}{\partial \mathbf{r}} \left(\nabla_x \mathbf{I}(\mathbf{x}) \right) + \nabla_y \mathbf{I}(\mathbf{x}) \frac{\partial}{\partial \mathbf{r}} \left(\nabla_y \mathbf{I}(\mathbf{x}) \right)}{\| \nabla \mathbf{I}(\mathbf{x}) \|} \end{aligned} \tag{3.39}$$

and with,

$$\begin{aligned} \frac{\partial}{\partial \mathbf{r}} \left(\phi(\theta(\mathbf{I}(\mathbf{x})) - i) \right) &= \frac{\partial}{\partial i} \phi \left(\theta(\mathbf{I}(\mathbf{x})) - i \right) \frac{\partial}{\partial \mathbf{r}} \left(\operatorname{atan} \left(\frac{\nabla_y \mathbf{I}(\mathbf{x})}{\nabla_x \mathbf{I}(\mathbf{x})} \right) \right) \\ = \frac{\partial}{\partial i} \phi \left(\theta(\mathbf{I}(\mathbf{x})) - i \right) & \frac{1}{1 + \left(\frac{\nabla_y \mathbf{I}(\mathbf{x})}{\nabla_x \mathbf{I}(\mathbf{x})} \right)^2} \frac{\partial}{\partial \mathbf{r}} \left(\frac{\nabla_y \mathbf{I}(\mathbf{x})}{\nabla_x \mathbf{I}(\mathbf{x})} \right) \end{aligned} \tag{3.40}$$

with

$$\frac{\partial}{\partial \mathbf{r}} \left(\frac{\nabla_y \mathbf{I}(\mathbf{x})}{\nabla_x \mathbf{I}(\mathbf{x})} \right) = \frac{\frac{\partial}{\partial \mathbf{r}} \left(\nabla_y \mathbf{I}(\mathbf{x}) \right) \nabla_x \mathbf{I}(\mathbf{x}) - \nabla_y \mathbf{I}(\mathbf{x}) \frac{\partial}{\partial \mathbf{r}} \left(\nabla_x \mathbf{I}(\mathbf{x}) \right)}{(\nabla_x \mathbf{I}(\mathbf{x}))^2} \tag{3.41}$$

We now only needs to determine the values of $\frac{\partial}{\partial \mathbf{r}} \left(\nabla_x \mathbf{I}(\mathbf{x}) \right)$ and $\frac{\partial}{\partial \mathbf{r}} \left(\nabla_y \mathbf{I}(\mathbf{x}) \right)$.

from [Marchand 10], we can obtain the following expression:

$$\mathbf{L}_{\boldsymbol{s}} = \nabla_x \mathbf{s}(\mathbf{r}) \mathbf{L}_x + \nabla_y \mathbf{s}(\mathbf{r}) \mathbf{L}_y \tag{3.42}$$

We can thereby define $\mathbf{L}_{\nabla_x \mathbf{I}(\mathbf{x})}$ such as:

$$\begin{aligned}\mathbf{L}_{\nabla_x \mathbf{I}(\mathbf{x})} &= \frac{\partial}{\partial x}\left(\nabla_x \mathbf{I}(\mathbf{x})\right)\mathbf{L}_x + \frac{\partial}{\partial y}\left(\nabla_x \mathbf{I}(\mathbf{x})\right)\mathbf{L}_y \\ &= \nabla_x^2 \mathbf{I}(\mathbf{x})\mathbf{L}_x + \nabla_{xy}\mathbf{I}(\mathbf{x})\mathbf{L}_y \end{aligned} \tag{3.43}$$

And in the same way:

$$\begin{aligned}\mathbf{L}_{\nabla_y \mathbf{I}(\mathbf{x})} &= \frac{\partial}{\partial x}\left(\nabla_y \mathbf{I}(\mathbf{x})\right)\mathbf{L}_x + \frac{\partial}{\partial y}\left(\nabla_y \mathbf{I}(\mathbf{x})\right)\mathbf{L}_y \\ &= \nabla_{yx}\mathbf{I}(\mathbf{x})\mathbf{L}_x + \nabla_y^2 \mathbf{I}(\mathbf{x})\mathbf{L}_y \end{aligned} \tag{3.44}$$

where $\mathbf{L}_x$ and $\mathbf{L}_y$ are defined in Eq. (3.32)

In this section, we instantiated the interaction matrices and control laws corresponding to our target histograms. The resulting expressions can be directly implemented and used to control a robotic system.

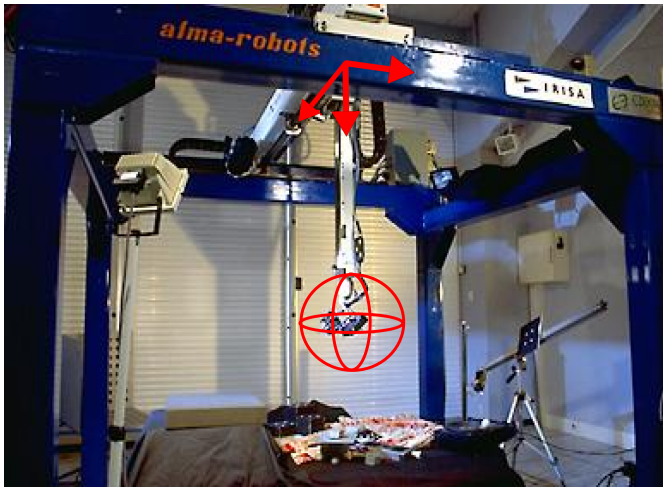
Experimental results and comparisons

6 DOF positioning task on a gantry robot

In order to validate these approaches, we perform a positioning task on a 6DOF gantry robot using the 4 features designed previously: gray-levels, HOG, color and synthesized gray level from color planes.

The robotic system used can be seen on Fig. 3.24(a). It consists of a 6DOF gantry robot with a camera attached on the end-effector. This system can be positioned withing tenth of a millimeter accuracy, allowing to experiment control laws on fine motions within a large work space. The implementation of the control laws is performed in C++, through the use of the ViSP library [Marchand 05], developed by the Lagadic team, which provides an API to control the robotic system in velocity mode directly in the camera frame. The images acquired by the camera have a resolution of 320×240 pixels. As introduced in Chapter 2, instead of the classical Newton optimization method (Eq. (2.8)), we chose here (as in [Collewet 08b]) to perform the optimization by using a Levenberg-Marquardt optimization scheme, which we recall is defined by Eq. (2.19), such as:

$$\mathbf{v} = -\lambda(\mathbf{H}_\mathbf{I} + \mu\, diag(\mathbf{H}_\mathbf{I}))^{-1}{\mathbf{L}_\mathbf{I}}^\top(\mathbf{I}(\mathbf{r}) - \mathbf{I}^*)$$



(a)

(b)

Figure 3.24: *(a) Afma robot used in the positioning experiments. (b) Scene recorded by the camera.*

where μ is the Levenberg-Marquardt parameter which is positive scalar that defines the behavior of the control law, from a steepest gradient behavior (μ small) to a Gauss-Newton's behavior (μ large) and $\mathbf{H_I} = \mathbf{L_I}^\top \mathbf{L_I}$. This optimization scheme allows for more flexibility while dealing with complex cost functions in order to have a suitable convergence rate. The μ parameter of the LM minimization scheme is set constant at $10e^{-3}$ during all the experiments. This value was picked empirically through preliminary test runs.

The test scene can be seen in Fig 3.24(b). In order to test the robustness of the methods, this scene contains a variety of textures, from homogeneously textured-wood to specular surfaces and scattered 3D objects.

Initial positioning error is such that $\Delta\mathbf{r}$ = (14.4cm, -17.7cm, 1.3cm, -28°, -18°, -2.4°) with respect to the target position, with a mean depth of 80cm. The only variable parameter in these experiments is the number of bins used in the computation of the histograms (selected empirically for each method, based on the preliminary cost function visualizations of Section 3.2.3, as well as test runs): 32 for the intensity histogram and synthesized one plane color histogram, 10 for the HOG and 8 for the Hue-Saturation histogram. The number of histograms used throughout the image (see Section 3.2.2) is set constant at 6x6. Concerning computational complexity, the gray-levels, HOG and synthesized color histogram perform equivalently, the HS method being twice as slow due to the increased complexity to compute the associated interaction matrix (8 bins for each color plane, amounting to 64 possible values). On a i7-4600 processor, we obtain an iteration time of 70ms for the 3 first methods and 140ms for the HS method.

The first experiment can be seen on Fig. 3.25, and uses the intensity histogram, computed

from Eq. (3.2). Although the process eventually converges with a good accuracy at the end of the motion (less than 5 millimeter offset), the first part of the motion exhibits a very sub-optimal motion (Figs. 3.25(c)(d)(i)). It is interesting to compare it with the evolution of the cost function optimized throughout the task, and showed on Fig. 3.25(b), which decreases steadily. This mismatch between the evolution of the positioning error and the cost function shows that for a large part of the positioning, this cost function is not optimal and may lead to unwanted motions, as shown on the trajectory graph at Fig. 3.25(i). The unwanted motion in this case is a strong overshoot of the rotation around the y-axis and its coupled translation along the x-axis. During the second part of the motion however, all error components decrease steadily, meaning that the cost function possess better convex properties closer to the target position.

The second experiment displayed by Fig. 3.26 uses histograms computed on an image that is a combination of the Hue and Saturation color planes recorded by a color camera, making an artificial intensity image, following Eq. (3.5). By comparing this experiment with the previous one, we can see that the resulting cost function exhibits better properties, as no large unwanted motion is present at the start of the experiment. On the other hand, the decrease of the positioning error is still not perfectly following an exponential decrease. This can be explained by the presence of both specular surfaces and non-planar areas in the scene, with each of those elements not taken into account in the modeling of the control law. The final precision is on par with the previous method with less than 5 millimeter accumulated error at the end of motion.

The third run (Fig. 3.27) uses the control law based on both the Hue and Saturation planes of a color image captured from the scene, with histograms computed following Eq. (3.6). We can see that the performance of this control law, in addition of being more computationally expensive, are worse than the two first experiments. On one hand, unwanted motions are present, and the end precision is twice as bad, with an error over 1 centimeter at the end of the motion. This can be caused by the strong underlying approximation that the scene is lit only by white light sources. Indeed, if the light sources affect the colorimetry of the scene, with the non-planar object on the scene, this will create a slight discrepancy between the shade of color of the visible side of the object with respect to the color that is expected from the target image. A second cause could be linked to the additional complexity of this feature: as many more histogram bins are computed, the resulting cost function can be more sensitive to noises, degrading the performances of the associated control law.

The last run is described by Fig. 3.28 and is based on the HOG feature, computed from Eq. (3.8). We can see that this method performs better than the other three on every aspect, as the end precision is less than 1 millimeter, with a steady decrease of all error components. We can see that this steady decrease of all error components is by no means a guaranty of linear trajectory, as Fig. 3.28(i) shows. The only undesirable property that can be seen in this run is the slower convergence rate of the rotations around the x and y axes. One can hypothesize than this descriptor perform better than the intensity-based ones because of the noise-filtering

property of using the gradient of the image. This alleviates the impact of the hypothesis such as the lighting stability, should these be violated during real world experiments.

By comparing the trajectories followed by the camera on each experiment it can be noted that although the camera is steadily positioned toward the desired pose, this does not constraint the evolution of the 3D trajectory to any kind of optimal path.

Comparing convergence area in simulation

Increasing initial position distance

Since all 4 methods are validated by real robot experiments, further testing is required in order to compare them in a more detailed way. In terms of convergence area: random initial positions are computed, with an increasing spatial noise from the desired position. For each step of increasing spatial noise, multiple trials are run. By determining for each run if the method successfully converges (spatial error below a given threshold), it gives us a percentage of successful convergence. By repeating this operation with all the 4 control laws, it is then possible to compare them quantitatively in terms of convergence area. For this test, 10 increases of the spatial noises are executed. For each step, 40 runs are performed to get the percentage of convergence. The mean depth is of $10cm$ and the spatial Gaussian noise is applied such as :

- from 0 to $1cm$ in the mean standard deviation for the x/y translations;
- from 0 to $5cm$ for the depth;
- from 0 to 10°for the rotations around the x/y axis;
- from 0 to 20°for the rotations around the z axis.

The result of this test can be seen in Fig. 3.29(a). We can see that all four methods perform rather similarly under these nominal conditions, as expected from the analysis of the cost functions of a previous section, where a couple of parameters could always be found to obtain a suitable locally convex area, and these area where roughly of similar sizes.

This experiment confirms that the important performance gap between the 4 methods seen in the real-robot experiments arises from noises introduced by unmodeled phenomenons in the scene, as proposed earlier.

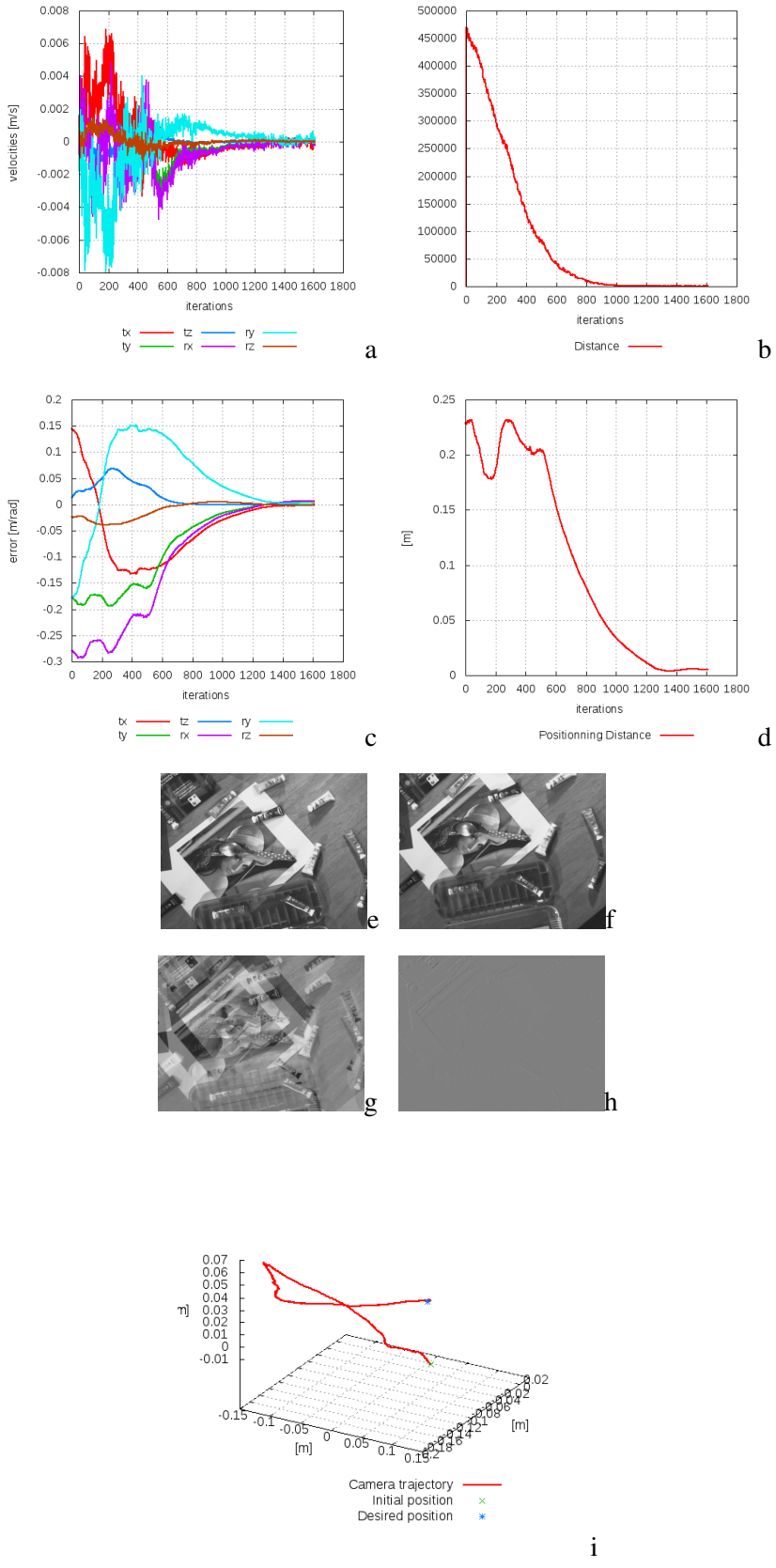


Figure 3.25: *Gray-levels histogram-based servoing. Camera velocities in m/s and rad/s in (a). (b) Matusita distance. (c) Positioning error per DOF. (d) Total positioning error. (e) Initial image. (f) Desired image. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D trajectory.*

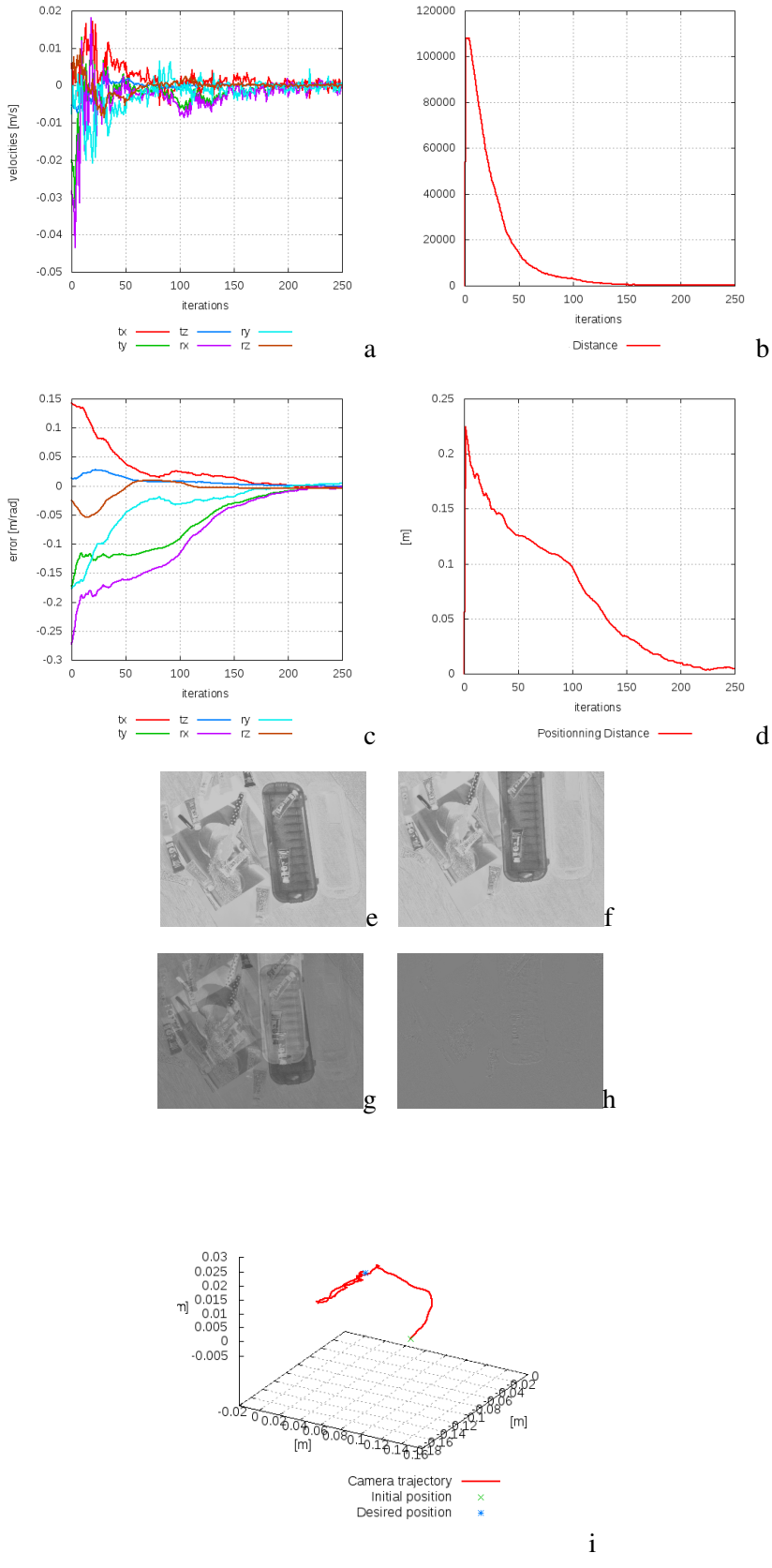


Figure 3.26: *Synthesized gray level from Hue-Saturation histogram-based servoing. Camera velocities in m/s and rad/s in (a). (b) Matusita distance. (c) Positioning error per DOF. (d) Total positioning error. (e) Initial image. (f) Desired image. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D trajectory.*

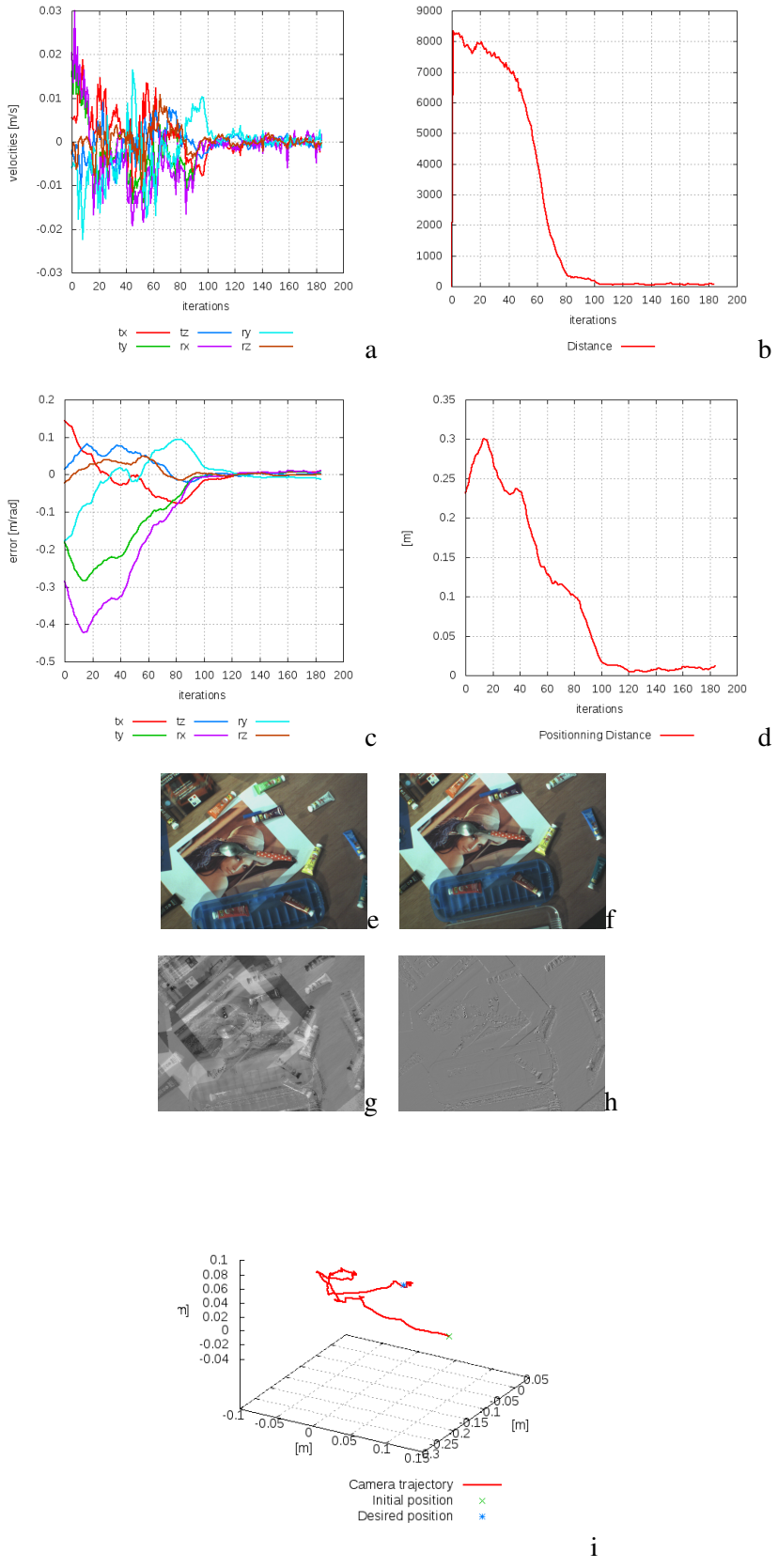


Figure 3.27: *Hue-Saturation histogram-based servoing. Camera velocities in m/s and rad/s in (a). (b) Matusita distance. (c) Positioning error per DOF. (d) Total positioning error. (e) Initial image. (f) Desired image. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D trajectory.*

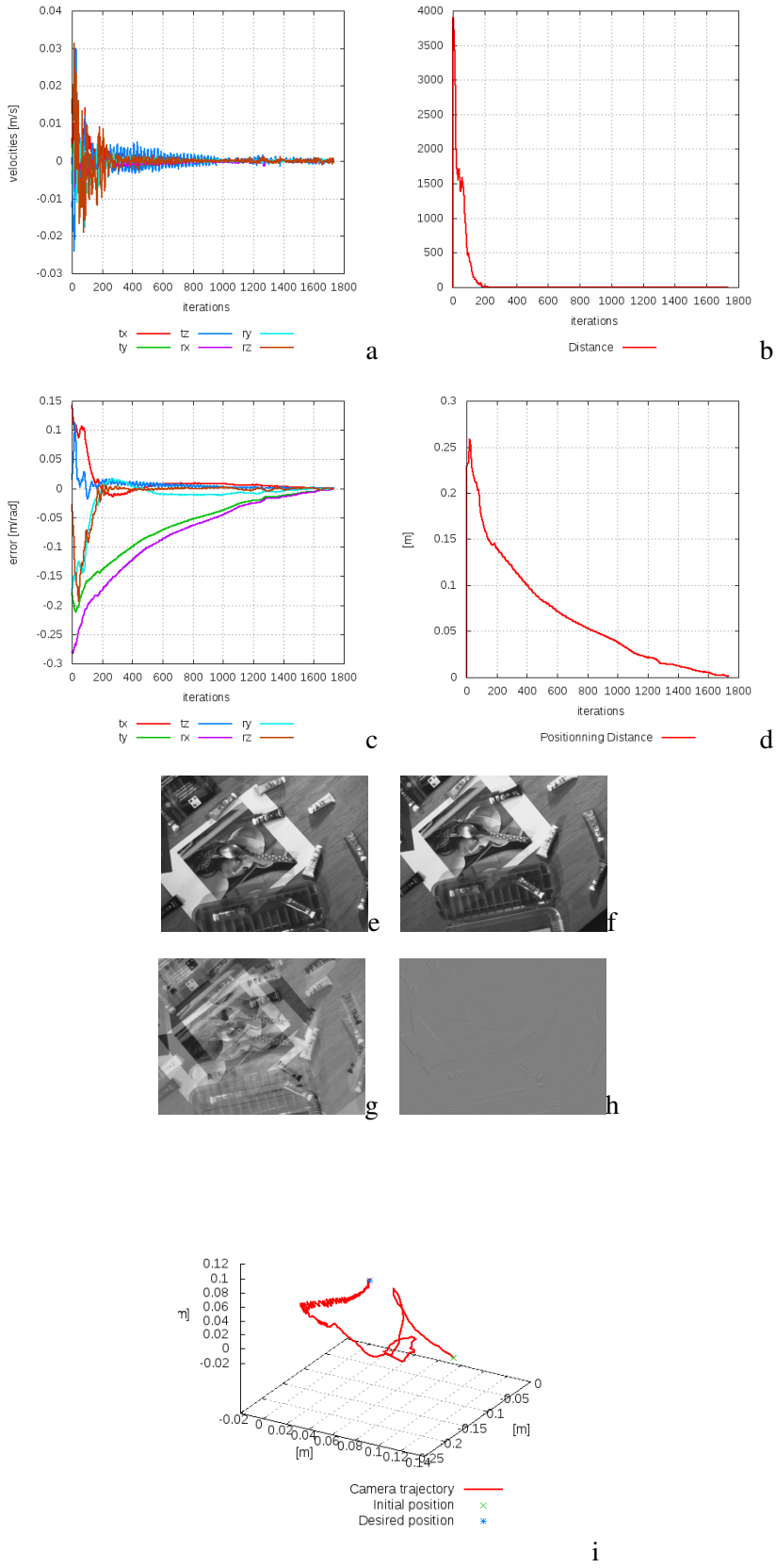


Figure 3.28: *HOG-based servoing. Camera velocities in m/s and rad/s in (a). (b) Matusita distance. (c) Positioning error per DOF. (d) Total positioning error. (e) Initial image. (f) Desired image. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D trajectory.*

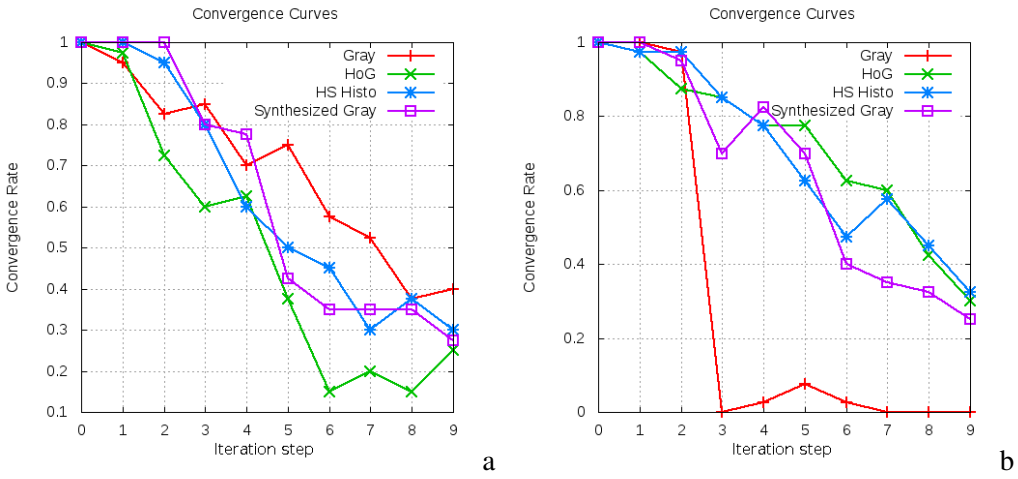


Figure 3.29: *Convergence areas. (a) With increasing spatial noise on initial position; (b) With increasing spatial noise on initial position and diminishing illumination*

Decreasing global luminosity

The same test is performed with an addition: at each step, in top of the increase in spatial noise, the global illumination of the scene is reduced linearly (increasingly as the iteration number increase). It allows us to compare the methods in terms of sensibility to global illumination changes. The same noise as in the previous experiment is applied at each step, but additionally, the illumination of the image plane goes from 100% to 10% (as illustrated by Fig. 3.30).

Figure 3.30: *(a) Image with 100% illumination. (b) Image with 10% illumination*

The result of this test can be seen in Fig. 3.29(b).

The main information provided by this experiment is that the two variations of color-based histograms and the HOG-based methods keep performing at the same level despite the increasing illumination change. This emphasizes the fact that these methods do possess an invariance to illumination changes as hinted by the nature of the chosen histograms and the analysis of their cost function visualization. It is interesting to note that this invariance do not apply in the same way for all three methods, since the color-based ones requires a change in illumination that do not alter the coloration of the scene (a change in a white light illumination, such as the sun during the main part of the day), whereas the HOG-based method does not suffer from this problem, being based purely on the orientation of the gradients: neither the change in intensity, nor color alters this property. One interesting comment concerning the analysis of these figures is that the HOG method performs better under mild illumination decrease than under full illumination. This is due to the shape of its cost function that becomes smoother, with less local minima, when the illumination in the image is low. This is caused by an induced quantification in the pixels intensities that homogenize large texture-less areas with randomly oriented gradients that provide non-informative data in the histogram.

Navigation by visual path

Navigation by visual path consists of the autonomous navigation of a mobile robot based on the video recording of a predetermined path [Blanc 05] [Diosi 07] [Dame 13]. From this recording, key-frames are extracted and the robot navigates in order to minimize the difference between its current camera view and the first key-frame. The method then proceeds iteratively, going through all the key-frames, therefore following the same spatial path as the one corresponding to where the key-frames where acquired (as illustrated in the Fig. 3.31). In the first experiment, the robot navigates at first around 20 meters inside corridors lit by both artificial and natural lighting through large windows. In the second experiment, it navigates around 20 meters in an outdoor environment. Some pedestrian activity occurs during the experiment, creating important occlusions. The robot is a non-holonomic Pioneer robot (illustrated in Fig. 3.32) with 2 controllable DOF: the rotation and the forward velocity. Here, only the rotation is controlled by visual servoing, the forward translation being fixed as constant. Since the goal of this experience is to test the robustness of the histogram-based method, no elaborate scheme of key-frames selection is performed: the key-frames are acquired at a fixed rate (1.5Hz). In order to benefit from the maximal invariance with respect to light variation, we choose here to navigate using the HOG-based visual servoing. It is interesting to comment the fact that we do not propose a quantitative measure concerning the match between the 3D path recorded and the path undertaken by the robot because our goal is to follow a path in the image-space, and then, ground-truth is unknown. As such, our goal is only to go from a starting point to a destination, independently of a possible minor rerouting that can occur due to partial occlusions or light changes. As we can see in the Fig. 3.33, the proposed method succeeds in navigating indoor along the visual path, even in presence of important discrepancies due to turns in the path that

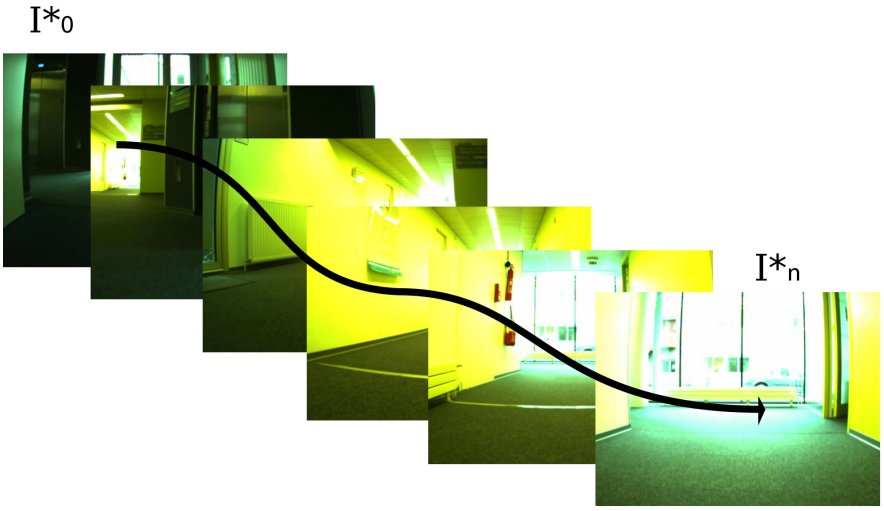


Figure 3.31: *Illustration of the visual path approach*

Figure 3.32: *Pioneer robot used in the navigation experiment*

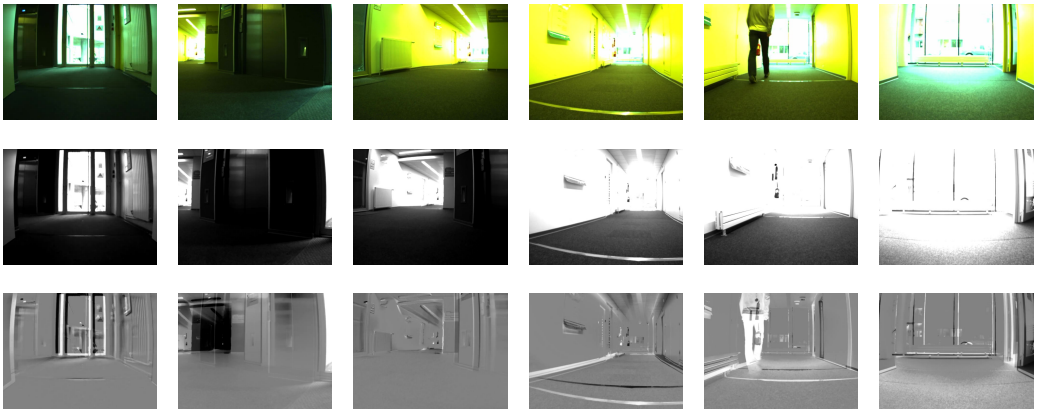


Figure 3.33: *Samples of the navigation experiment in the indoor scene: first line is the current key-frame, second is the actual view of the robot and third line is the difference between the two previous lines, this SSD is given to outline the difference.*

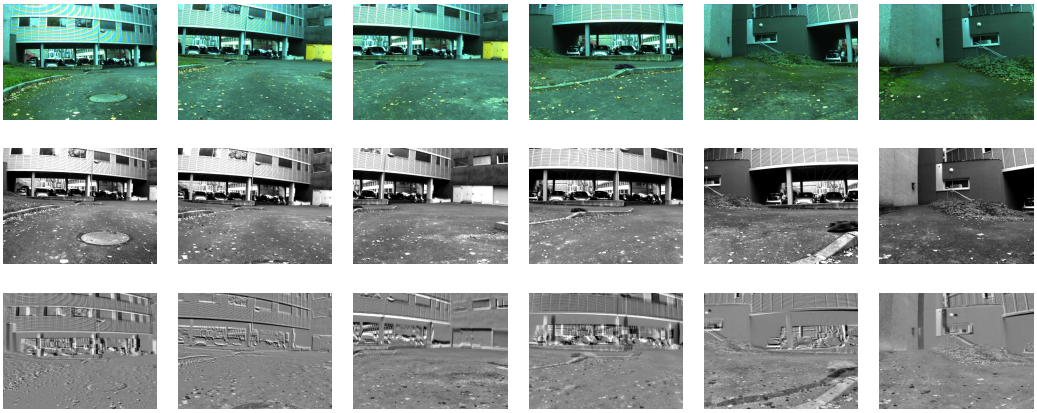


Figure 3.34: *Samples of the navigation experiment, in the outdoor scene: first line is the current key-frame, second is the actual view of the robot and third line is the difference between the two previous lines, this SSD is given to outline the difference.*

can be seen for example in 3.33(n) or the presence of a pedestrian that was recorded in the reference visual path, as seen in Fig. 3.33(e). The second experiment depicted in Fig. 3.34 has been performed outdoor in order to validate the invariance to mild illumination changes due to meteorological conditions (variably cloudy conditions) and shows similar good performances.

Conclusion

In this chapter, we wanted to develop a new visual servoing framework to exploit the descriptive properties of histograms applied to images. We proposed a methodology based on the use of multiple histograms throughout the images combined with a strategy that exploit B-Splines to compute VS interaction matrices analytically. We showed that this framework can be applied to several histogram formulations, such as gray-levels, color or oriented gradient histograms.

The analysis of the resulting cost functions showed that the choice of the probability distributions parameters allows for increased flexibility in order to improve the properties of the cost function before performing optimization.

A benchmark in simulation has been performed to compare the performances of the proposed methods in both nominal and perturbed condition to evaluate the individual properties of the resulting control laws. Experiments demonstrated that the resulting control law were suitable to solve positioning tasks in eye-in-hand configurations, dealing with perturbed conditions such as illumination changes and occlusions.

These works have been published in [Bateux 15, Bateux 17a].

Chapter 4

PARTICLE FILTER-BASED VISUAL SERVOING

Motivations

In the previous chapter, we showed that broadening the direct visual servoing methods by defining new visual features can allow us to find more suitable cost functions, with better properties and ultimately to design more efficient control laws.

Still, even with better features, finding ways of exploiting more fully and efficiently the information provided by the chosen cost functions remains a challenge. In this chapter, we propose a new method for designing direct (but not only) visual servoing control schemes, by replacing one of the core block of this process: the control law that can be seen as an optimization process. Indeed, for now, nearly all VS control law rely for the optimization of their cost functions on methods that perform various forms of gradient descents (such as the Gauss-Newton method in [Hutchinson 96, Chaumette 06] or Levenberg-Marquardt method in [Collewet 08b]).

Such gradient descent methods are known to perform extremely well in case where the cost function to optimize displays clear convex properties, but may fail when these properties disappear and the optimization can then get stuck on local minima or diverge altogether [Chaumette 98a, Chaumette 07].

In this chapter, we propose to replace the control law based on a classical gradient descent-style optimization process by another method, based on a Particle Filter, to perform a stochastic optimization. We show that this method is able to converge toward a global minimum, even when the initialization state is outside of a convex area, by relying on a cost function estimation process that allows to sample the search space and discover a convex region unreachable by classical gradient descent methods.

Particle Filter overview

Statistic estimation in the Bayesian context

In this chapter, since we aim to replace the classical gradient descent approach, we need to replace the control law associated to analytic formulation of the cost function that we want to optimize. Instead of solving analytically the optimization problem, we can instead rely on a statistical estimation of the cost function that we want to optimize. From a valid estimation of our cost function, it can be possible to find a global minimum in non-convex cost function situations.

To do so, we are looking to estimate the state of our system through time, depending on observations or measures. In the case of the control of a camera, this state often contains 2D or 3D relative pose parameters $\mathbf{r}_k$ of the camera, between the current pose and a target pose. If we suppose that we possess a model f of the evolution of the state $\mathbf{x}_k$ of the system, then it enables, from the previous state $\mathbf{x}_{k-1}$, to give a prediction of the new state $\mathbf{x}_k$. This model is an approximation of the behavior of the system, with the uncertainty is modeled by a noise $\mathbf{n}_k$.

Without prior information on the relative motion between the scene and the camera, it is possible to model this relative motion in a simplified way as a constant position model, such as:

$$f(\mathbf{x}_{k-1}, \mathbf{n}_k) = \mathbf{x}_{k-1} + \mathbf{n}_k \tag{4.1}$$

For a positioning task, the state $\mathbf{x}_k$ corresponds to the pose parameters vector $\mathbf{r}_k = (t_{xk}, t_{yk}, t_{zk}, r_{xk}, r_{yk}, r_{zk})$ defining the relative pose between the current camera pose and the desired camera pose.

We also suppose that we possess an observation model (which represents a measure of the system at time k)

$$\mathbf{z}_k = h(\mathbf{x}_k, \mathbf{w}_k) \tag{4.2}$$

where $\mathbf{w}_k$ is the noise related to this model.

In the statistic framework of this section, the state and the measures are considered as noisy and these noises are taken into the estimation process. State and measure are then considered as random variables in the statistical sense. Estimating the state $\mathbf{x}_k$ of the system from measures $\mathbf{z}_{1:k} = (\mathbf{z}_1, \ldots, \mathbf{z}_k)$ taken at instant k is then equivalent to determining the probability density of the state $\mathbf{x}_k$ knowing the measures $\mathbf{z}_{1:k}$.

In the next sections, we will present the formalism of the generic Bayesian filtering, then instantiate it through two classical approximations of the generic filter, namely the Kalman Filter and Particle Filter.

Bayesian filtering

Bayesian filtering stems from the Bayes law on conditional probabilities. This law states that for every random variable A, B:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.3}$$

where $P(A|B)$ is the probability of observing event A given the knowledge of event B.

To link this expression to our context, it is possible to say that, if the measures $\mathbf{z}_{1:k-1}$ are conditionally independent between themselves given the states, then the Bayes law allows the following recursive relation:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{4.4}$$

It can be noted that $p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})d\mathbf{x}_k$ is a normalization factor, independent of the state. Thus, Eq. (4.4) shows that the posterior density of the state is proportional to the product of the prior probability density of the state $p(\mathbf{x}_k|\mathbf{z}_{1:k-1}$ and the likelihood of the measure $p(\mathbf{z}_k|\mathbf{x}_k)$.

Supposing that the measured process is a Markovian process, for which the state $\mathbf{x}_k$ depends only of the previous state $\mathbf{x}_{k-1}$, then Eq. (4.4) becomes:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \propto p(\mathbf{z}_k|\mathbf{x}_k) \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1}. \tag{4.5}$$

In a Bayesian framework, an optimal recursive solution to this problem can be found based on two steps:

$$p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) \overset{prediction}{\rightarrow} p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) \overset{correction}{\rightarrow} p(\mathbf{x}_k|\mathbf{z}_{1:k}) \tag{4.6}$$

- the prediction step uses the equation of the system's dynamic and previous probability density $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ to obtain the prior probability density $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1}; \tag{4.7}$$

- the correction step allows to compute the posterior probability density $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ from the likelihood function $p(\mathbf{z}_k|\mathbf{x}_k)$ thanks to the recursive Bayes formula Eq. (4.5).

This recursive solution is true for any Markov process for which the observations are conditionally independent given the states. This resulting filter is thus generic since no hypothesis is made on the shape of the probability density that we aim to estimate.

On the other hand, it is not always possible to solve Eq (4.5) and (4.7) explicitly. Depending on the problem, various simplifications are used to approach the optimal solution. In the case of a linear Gaussian, an analytic solution is given by the Kalman filter. In the general case, approximation methods are used:

- linearization of the system around the current estimate (Extended Kalman Filter (EKF))
- numerical approximation through discretization of the state space
- representation of the filter through samples (Particle Filters)

The following section presents the Particle Filter, commonly used in computer vision and robotic problems.

Particle Filter

In this section, we show how the Particle Filter can provide an approximation of the optimal filter presented earlier. Unlike the Kalman Filter, no hypothesis is made concerning the shape of the probability densities involved or the linearity of the system [Arulampalam 02, VanDer-Merwe 01]. The theory behind the Particle Filter is based on the particle approximation described as follow.

4.2.1.2.1 Particle approximation and Monte Carlo principle The Monte Carlo methods are derived from the impossibility to compute the integral values in Eq. (4.5) and (4.7). The idea of particle approximation is to try to estimate a value through a set of samples. These samples, called particles, are supposed to be independent and identically distributed according to the probability density to be estimated. This means that more samples will be present around the peaks of the function to be estimated.

The idea of particle approximation is based on the strong law of large numbers, according to which the expected value computed on samples is an estimator of the actual probability density. More formally, if we note $p(\mathbf{x})$ the probability density to be estimated and $\{s^i\}_{i=1}^N$ a set of N independent and identically distributed samples according to p, then for any continuous and bounded function ϕ, we have:

$$\frac{1}{N}\sum_{i=1}^{N}\phi(s^{(i)}) \underset{n\to\infty}{\to} E_p[\phi(\mathbf{x})] = \int \phi(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x} \tag{4.8}$$

where E_p is the expected value with respect to the density p.

Let $p^N(\mathbf{x})$ be defined as:

$$p^N(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\pi^{(i)}\delta_{s^{(i)}}(\mathbf{x}) \tag{4.9}$$

with $\pi^{(i)} \geq 0$ and $\sum_{i=1}^{N}\pi^{(i)} = 1$.

This formulation allows to give a weight $\pi^{(i)}$ to each particle $s^{(i)}$. For any ϕ, we have then:

$$E_p[\phi(\mathbf{x})] = \int \phi(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x} \approx \int \phi(\mathbf{x})\,p^N(\mathbf{x})\,d\mathbf{x} = \sum_{i=1}^{N}\pi^{(i)}\phi(s^{(i)}) \tag{4.10}$$

To obtain such a sampling when the density p is not directly accessible, one must rely on a sampling technique, such as the Importance Sampling method described in the next paragraph.

4.2.1.2.2 Importance sampling The objective is to get a representation such as Eq. (4.9) of a probability density p, with p not directly measurable. Let us define a distribution q, called proposal distribution, such as $p = 0 \Rightarrow q = 0$, and it is possible to obtain N samples $s^{(i)}$ of q. Then:

$$\int \phi(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x} = \int \phi(\mathbf{x})\,\frac{p(\mathbf{x})}{q(\mathbf{x})}\,q(\mathbf{x})\,d\mathbf{x} \tag{4.11}$$

$$\int \phi(\mathbf{x}) p^N(\mathbf{x}) d\mathbf{x} \frac{1}{N} \sum_{i=1}^{N} \frac{p(s^{(i)})}{q(s^{(i)})} \phi(s^{(i)}) \tag{4.12}$$

where $p^N(\mathbf{x})$ is given by Eq. (4.8). The terms $\pi^{(i)} = \frac{p(s^{(i)})}{q(s^{(i)})}$ are importance weights. In practice, it is usual to normalize them, to get back to the form of Eq. (4.9), with $\sum_{i=1}^{N} \pi^{(i)} = 1$.

4.2.1.2.3 Particle Filter The goal behind a Particle Filter is to determine approximations from particles for the recursive Bayes filter. We then want to obtain an approximation of the density $p(\mathbf{x}_k|\mathbf{x}_{1:k-1})$ such as:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx p^N(\mathbf{x}_k|\mathbf{z}_{1:k}) = \sum_{i=1}^{N} \pi_k^{(i)} \delta_{s_k^{(i)}}(\mathbf{x}_k) \tag{4.13}$$

Supposing that at time $k-1$, we have such an approximation:

$$p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) \approx p^N(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) = \sum_{i=1}^{N} \pi_{k-1}^{(i)} \delta_{s_{k-1}^{(i)}}(\mathbf{x}_{k-1}) \tag{4.14}$$

The equations of the recursive Bayes filter Eq. (4.5) and (4.7) become:

- Prediction:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) \approx p^N(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \sum_{i=1}^{N} \pi_{k-1}^{(i)} \delta_{s_k^{(i)}}(\mathbf{x}_k) \tag{4.15}$$

  where the $s_k^{(i)}$ are sampled according to $p(\mathbf{x}_k|\mathbf{x}_{k-1} = s_{k-1}^{(i)})$.

- Correction: from Eq. (4.5), we obtain:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) \approx \sum_{i=1}^{N} \pi_k^{(i)} \delta_{s_k^{(i)}}(\mathbf{x}_k) \tag{4.16}$$

  with

$$\pi_k^{(i)} = \frac{\pi_{k-1}^{(i)} p(\mathbf{z}_k|\mathbf{x}_k = s_k^{(i)})}{\sum_{i=1}^{N} \pi_{k-1}^{(i)} p(\mathbf{z}_k|\mathbf{x}_k = s_k^{(i)})} \tag{4.17}$$

We obtain an explicit expression of the particle approximation at time k from its expression at time $k-1$. In this section, we restrict ourselves to the usual case where the particles are

propagated according to the law $p(\mathbf{x}_k|\mathbf{x}_{k-1})$. In the general case where we use a proposal function to control the evolution of the particles, and apply the importance sampling principle presented in the previous paragraph (see [Arulampalam 02] for example).

In practice, the particle filtering is based on the following principle: a set of particles is created, each particle representing a possible state of the system. These particles then move through the state space according to a prediction model. From this, we obtain a new set of particles, representing the probability density of the prediction. In a second time, the likelihood of each particle is evaluated through a measure. A weight is given to each particle from its likelihood. This new set of weighted particles represents the new posterior probability density. This method is summarized by the following algorithm, named Sequential Importance Sampling (SIS):

- Initialization: generate N particles $s_0^{(i)}{}_{i=1}^{N}$ and set $\pi_0^{(i)} = \frac{1}{N}$
- For $k = 1, \ldots, T$:
  1. Prediction: generate N particles $s_k^{(i)}$ according to $p(\mathbf{x}_k|\mathbf{x}_{k-1} = s_k^{(i)})$
  2. Correction: update the weights of the predicted particles using Eq. (4.17)
- Expectation is given by:

$$E[\mathbf{x}_k] = \sum_{i=1}^{N} \pi_k^{(i)} s_k^{(i)} \tag{4.18}$$

4.2.1.2.4 Degeneration and resampling The SIS algorithm, based only on those two steps of prediction/correction is not usable in practice. Indeed, after a few iterations, most of the particles are given a weight close to zero, and very small amount of the particle set are actually estimating the probability density. To alleviate this degeneration issue, classical particle filters use an additional resampling step that redistribute the particles toward the high likelihood areas of the state space.

The resampling technique favors the particles of high likelihood by duplicating them, removing particles with negligible weights ([Gordon 93]). The resulting algorithm is known as SIR (Sequential Importance Resampling). It is illustrated in Fig. 4.1 and summarized as follow:

- Initialization: generate N particles $\left\{s_0^{(i)}\right\}_{i=1}^{N}$ and set $\pi_0^{(i)} = \frac{1}{N}$
- For $k = 1, \ldots, T$:

1. Prediction: generate N particles $s_k^{(i)}$ according to $p(\mathbf{x}_k|\mathbf{x}_{k-1} = s_k^{(i)})$
2. Correction: update the weights of the predicted particles using Eq. (4.17)
3. Resampling: weighting random draw of N particles from the set $\left\{s_0^{\prime(i)}, \pi_{k-1}^{(i)}\right\}_{i=1}^{N}$: considering that every resampled particle has the same importance (actual measure unknown initially), we obtain $\left\{s_0^{(i)}, \frac{1}{N}\right\}_{i=1}^{N}$ (so that all particle weights are equal and sum to 1)

- Expectation is given by:

$$E[\mathbf{x}_k] = \sum_{i=1}^{N} \pi_k^{(i)} s_k^{\prime(i)} \tag{4.19}$$

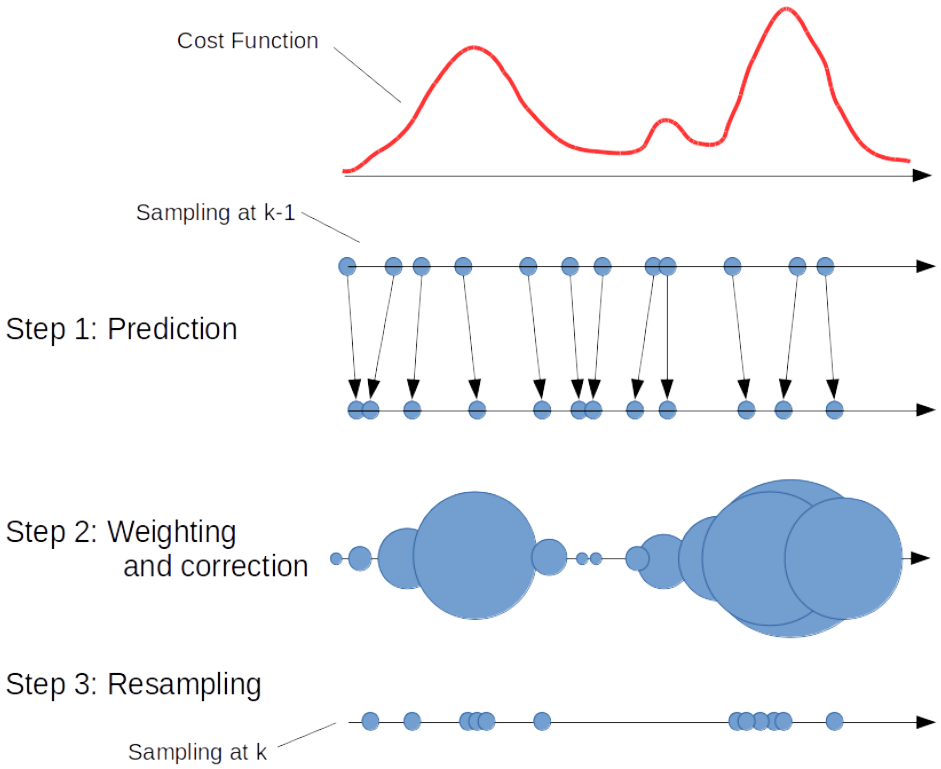


Figure 4.1: *Illustration of the SIR algorithm*

The Particle Filter is widely used in computer vision, due to its flexibility. As we saw, the PF tends to the optimal Bayesian Filter when the number of particles tends to infinity. Practically, this means that the precision of the estimate will depend directly on the number of particles used. The greater in dimension the search space, the higher the necessary number of particles. This can prove an issue as each particle, at every iteration, imply a computation of its likelihood

by making a measure. The use a large number of particles can lead to prohibitive computational time, especially for a task requiring real-time performances. Finding a way to make this measure efficiently is then paramount to maintain a good estimation between a moving physical system and the state estimate.

Particle Filter based visual servoing control law

In this section we will see how PF can be considered to replace the classical control law (similar to a Gauss-Newton optimization technique). In this section, we present a generic approach to perform PF-based visual servoing, independently of the descriptor used. Hence, it can be applied to a variety of contexts, such as feature-based visual servoing or the direct visual servoing.

We have seen earlier that the performances of a VS control law are linked in most case to the convexity of the cost function that is optimized between the initial camera pose and the optimum that corresponds to the camera target pose. This cost function appears to be, in most of the cases, highly non-linear. The previous chapter works show that the choice of cost function with better properties is key to improve the converging area of a control scheme. However, according to the starting pose of the camera, starting outside the convex area of the cost function leads to failure to find the global optimum.

The Particle Filter has been designed to make no hypothesis on the shape of the cost function that it estimates. Here, this property is used to overcome the limitation of having to start in an initial state within a convex area of the cost function. By estimating the cost function, it is possible to find an estimation of a global minimum without relying on gradient descent. The camera can then be positioned toward this estimated minimum independently of the local cost function variations.

This idea is illustrated by Fig. 4.2 on a 2-dimensional search space. The desired position of the system is located at the optimum of the cost function inside a convex area. The initial state of the system is outside of the convex area, prohibiting the use of gradient-based methods to solve the problem. Using the SIR algorithm, the PF will iteratively build an estimate of the cost function. As a large chunk of the cost function's state space estimation may be required, we will use initially a large number of particles, scattered over a large state space to perform what we call the Search Phase.

Once we are able to use the estimate to find the convex area surrounding the optimum, we switch to the Servoing Phase: the goal here is to drive the robotic system toward the convex

area in real-time (meaning in our case that the difference between the actual camera view and the image used to compute the cost function estimate is kept small enough not to affect performances and keep the system from oscillating). The number of particles is reduced to reduce computational costs, and to keep the estimation of the cost function accurate enough to drive the system, the spread of the particle set over the state space is reduced around the convex area. This allows to keep a good estimate of the cost function around the estimate and stir the system towards it. By applying this technique iteratively, the system can be driven toward the optimum, ultimately into the convex area. As the precision of the estimate is linked to the number of particles used, it can be more efficient to switch back to the classical VS control law to perform the final control steps, instead of increasing the number of particles, to reach the desired position as precisely as possible.

In the real setting, the state-space is the full SE(3) space, as we aim to control the 6DOF of the camera.

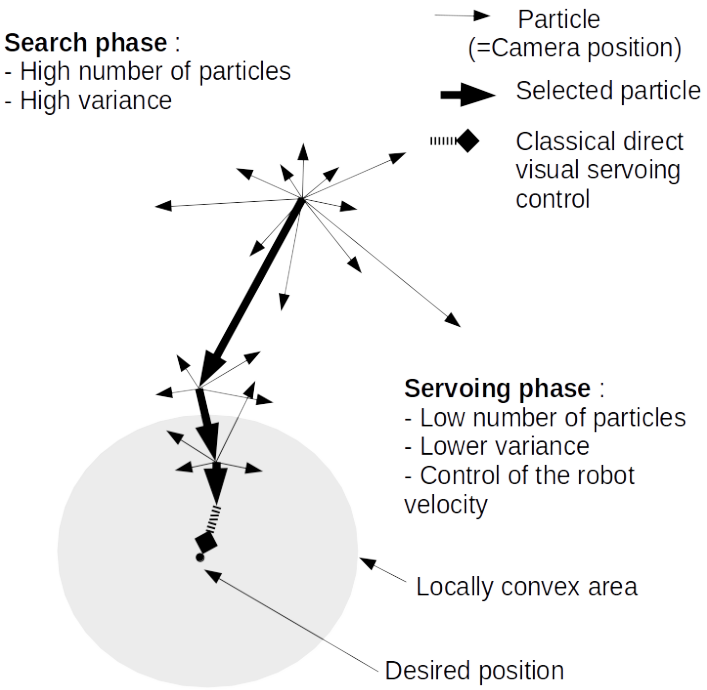


Figure 4.2: *Illustration of the 2-phases PF-based control scheme*

In order to implement such a control scheme, the following points have to be defined:

1. the search space and how to generate particles within it

2. the evaluation of the quality of each particle (i.e., estimation of $\pi_k^{(i)}$ from Eq. (4.17))
3. the camera velocity computation

Let us now examine more precisely these three parts.

Search space and particles generation

Particle definition for 6DOF task

We want to control the 6DOF of the robot; it seems natural then to define the state space as the full set of 3D pose. Let us define $\mathcal{F}_k$ as the camera frame at iteration k of the positioning process. Each generated particle $s_k^{(i)}$ represents a candidate virtual camera pose that is expressed in $\mathcal{F}_k$. We have then $s_k^{(i)} = \mathbf{r}_k(i)$, with $\mathbf{r}_k^{(i)}$ a pose vector $\mathbf{r}_k(i) = (\mathbf{t}, \theta\mathbf{u})$ that contains both the translational and the rotational positions of the candidate camera position expressed in the camera frame. Alternately, one can denote this position by a homogeneous matrix ${}^{k(i)}\mathbf{T}_k$ defined as:

$$ {}^{k(i)}\mathbf{T}_k = \begin{pmatrix} {}^{k(i)}\mathbf{R}_k & {}^{k(i)}\mathbf{t}_k \\ \mathbf{0}_{3\times 1} & 1 \end{pmatrix}, \tag{4.20} $$

where ${}^{k(i)}\mathbf{R}_k$ and ${}^{k(i)}\mathbf{t}_k$ are the rotation matrix and translation vector respectively that define the position of the particle in the current camera frame $\mathcal{F}_k$.

The set of particle is then a set of potential camera positions, meaning that each of them has to be evaluated and weighted (see Section 4.3.2) to obtain a preferred camera position. The system can then be driven toward this candidate position to minimize the cost function.

Particle generation

In order to explore the search space properly (both the local and farther neighborhood of the current camera pose), we generate each particle' associated pose component through the following Gaussian function f_k, expressed as:

$$ f(a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(a-\mu)^2}{2\sigma^2}} \tag{4.21} $$

With σ^2 the variance, μ the expected value and a is a pose component of the particle of highest likelihood at time k (at $k = 0$, $\mathbf{r}_0$ is set as the zero displacement from the camera frame, as no

prior information is available). This same function is used through the SIR algorithm whenever a particle is discarded and resampled. It has to be noted that as this function's role is to diffuse each particle around its own position, we always have $\mu = 0$.

To generate a given camera pose (expressed as ${}^{k(i)}\mathbf{T}_k$), we draw through $f(.)$ all 6 pose parameters $(t_x, t_y, t_z, r_x, r_y, r_z)$. The variances σ_i are defined empirically from the prior knowledge of the estimated task difficulty (small or large motions expected).

Particle evaluation

In this section, we detail ways to obtain the individual weights of each particle, corresponding to Eq. (4.17). This particle evaluation scheme is the most critical part of the algorithm, as it is linked directly to the cost function estimation process that drives the optimization scheme later on. The possibility to define multiple ways of evaluating a particle (equivalent to being able to choose among several cost functions in the previous chapter) is a strength of the proposed method by providing a generic optimization scheme that can evolve as state-of-the-art visual features are developed.

From feature selection to cost function

As described in Chapter 2, a positioning task can be performed using various features extracted from the image data. These features, which can consist of a set of geometrical points easily detected and tracked, a pixel vector or a statistic representation of the pixel intensities, are then compared in the current image recorded by the camera and the image at desired camera pose. This comparison is performed by defining a distance between the current and desired feature sets, creating a cost function that the control scheme will try to reduce to zero by computing a velocity to be applied to the camera.

To integrate this notion in the context of Particle Filtering, this cost function (independently of the features used to build it) is the target of the estimation process. This means that, according to Eq. (4.19), it is necessary to be able to compute, for each particle $s_k^{(i)}$, its associated weight $\pi_k^{(i)}$.

On the other hand, as we aim to derive a new control law to alleviate convergence issues for classical VS methods, we can integrate any features for which a distance can already be computed between two feature sets. We are then able to work with both geometrical features such as points (use in the classical VS), or with a direct descriptor such as the full pixel array

of an image (used in the direct VS).

As described in Chapter 2, each VS control law is associated to a cost function $\rho(.)$ that consists on a measure between two sets of chosen features (for example: SSD for direct VS, Euclidean distance for point-based VS). To perform the evaluation of each particle, we keep the commonly used $\rho(.)$ for each feature that we apply the PF control law to.

To summarize, for each particle $s_k^{(i)}$, a set of features is extracted, and a cost function $\rho(.)$ is computed between this feature set and the desired feature set. After all weights are computed, the weight values are normalized to guarantee that they sum to 1.

Predicting the feature positions through image transfer

The main difference compared to the classical gradient-descent approach is the need to perform measures. Indeed, at a given iteration, a single image acquisition allows to find a gradient descent and compute a camera velocity through an analytic control law. By using a PF, to get an estimate of the cost function that will be used to determine the camera velocity, each particle has to be weighted, based on a measure. As each particle represents a relative pose of the camera from its current pose, the most straightforward and most precise way of computing the weight the particle would be to actually move the camera to the relatives poses, record the images associated to it, and from there, extract the features, then compute the cost function and ultimately the weight. As we are dealing with a physical system with physical constraints (there is a maximal motion velocity, the need to avoid motion blur...etc), performing a single measure takes at least a few seconds. On top of that, a precise estimation necessitates the use of a set of particles, from a few tens to hundreds, which makes unpractical this measuring process, as performing a single iteration of the SIR algorithm would take minutes.

To solve this issue, we will take advantage of the computational power available to compute estimations of the measures. Indeed, we saw in the first chapter that image tranfer techniques can be used to predict virtual camera viewpoints, under a set of approximations (such as the planarity of the scene). By applying extensively this technique, it is possible to run a PF such as a single SIR iteration leading to an estimate of the cost function to minimize to be performed in real-time. Fig. 4.3 illustrates this approach by displaying samples from a set of measurements. Each of the samples presented here corresponds to an approximated recording of a virtual camera viewpoint, derived from a real image recording at time k. In this case the tranfer is accurate as we know that the scene is planar and that the distance Z between the camera and plane is known.

It has to be noted that this transfer technique can be applied to a whole image, as in Fig. 4.3,

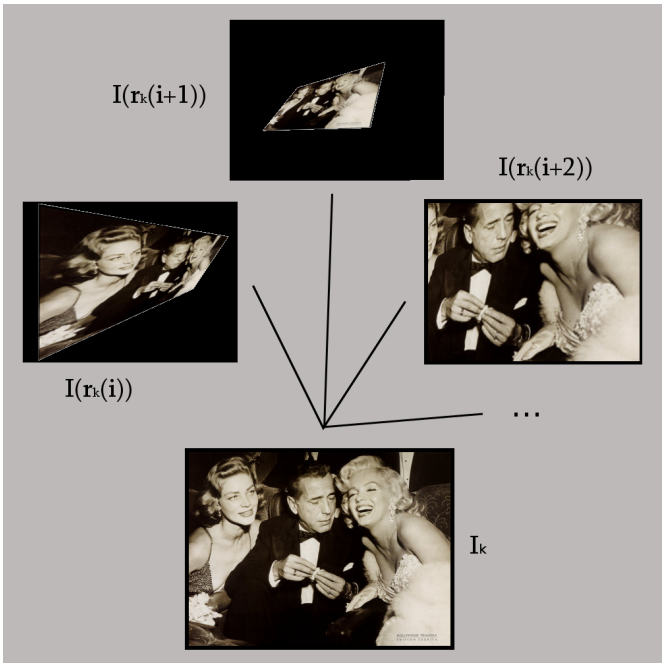


Figure 4.3: *Generation of examples of transferred images through homography to predict views from virtual camera positions*

to compute a cost function based on a dense descriptor, but it can also be used to predict the positions of geometrical features such as points. This latter case being only a simplification of the full image projection, where we only project the considered points instead of the entirety of the pixel grid.

As presented in the first chapter, the image transfer function applied to each element (either pixels or points) is the following:

$$^2\mathbf{x} = \mathbf{H}^1\mathbf{x} \quad \text{with} \quad \mathbf{H} = ^2\mathbf{R}_1 + \frac{^2\mathbf{t}_1\mathbf{n}^\top}{d} \tag{4.22}$$

where $\mathbf{H}$ is a 3×3 homography matrix that links the 3D coordinates of the point from the first to the second frame.

If we consider the transfer of the full image array $\mathbf{I}_1$ into an image $\mathbf{I}_2$, the transformation can be expressed as:

$$\mathbf{I}_2(\mathbf{x}) = \mathbf{I}_1\left(w(\mathbf{x}, \mathbf{h})\right) \tag{4.23}$$

where w is the image transfer function and $\mathbf{h}$ is a representation of the homography transformation.

Alleviating the depth uncertainty impact

While describing the image transfer function, one important point stated that this method rely on two hypotheses: the scene's geometry is unknown, hence, we consider that it is a planar scene, with its normal co-linear with the camera's optical axis, and the depth between this scene and the camera is supposed to be known. Although the projected image (image seen from a virtual viewpoint) will be slightly different than the one seen from an actual camera if the planar hypothesis is violated, this leads often to local discrepancies. On the other hand, if the second hypothesis according to the depth being known is a coarse approximation, then the resulting discrepancy between the real image and the approximated view will be affected globally because of a lever effect induced by the rotational part of the projection. In case of a strong error on the depth estimate, the result can be an image projection that takes out of the camera field of view most of the usable visual information.

This is illustrated in Fig. 4.4, where the black camera represents the positioning that would be reached by the robotic system, whereas the predicted image is simulated as being seen from the blue camera. As illustrated, this offset in position and orientation can lead to taking most of the useful image information out of the camera view when applying the transfer function.

In the case where the predicted viewpoint are to be used to obtain a feature, having a large section of unknown information can render the feature meaningless, lead to the computation of irrelevant particle weights, risk to introduce bias into the cost function estimate, and ultimately lead to a wrong camera velocity that can drive the system to diverge.

Since this projection of the image information outside of the viewpoint is due to the error in depth leading to an error when computing the rotation between the virtual camera and the projected image through the leverage effect, we alter the projection so that this error in depth affects more the translation components than the rotational ones. The result is that a given error in depth will not be amplified as no leverage effect exists on these translation components.

To do so, we perform a change in the transformation matrix ${}^{k(i)}\mathbf{H}_k$ to compute the position of the virtual camera in camera reference frame. The idea is that, in order to limit the offsets in rotation, we perform the rotation part of the transformation in the image plane reference frame. The effect of this change is illustrated in Fig. 4.5 which applies the same magnitude of rotation as in Fig. 4.4, but this time with the change in reference frame. We can see that the leverage effect disappears and that the center of the initial image is still at the center of the transferred image, the offset due to the error in depth being transferred mainly into the translation along the optical axis, inducing a translation along the z-axis. This leads to the error in the depth estimate affecting the accuracy of the motion of the camera along its optical axis, an error

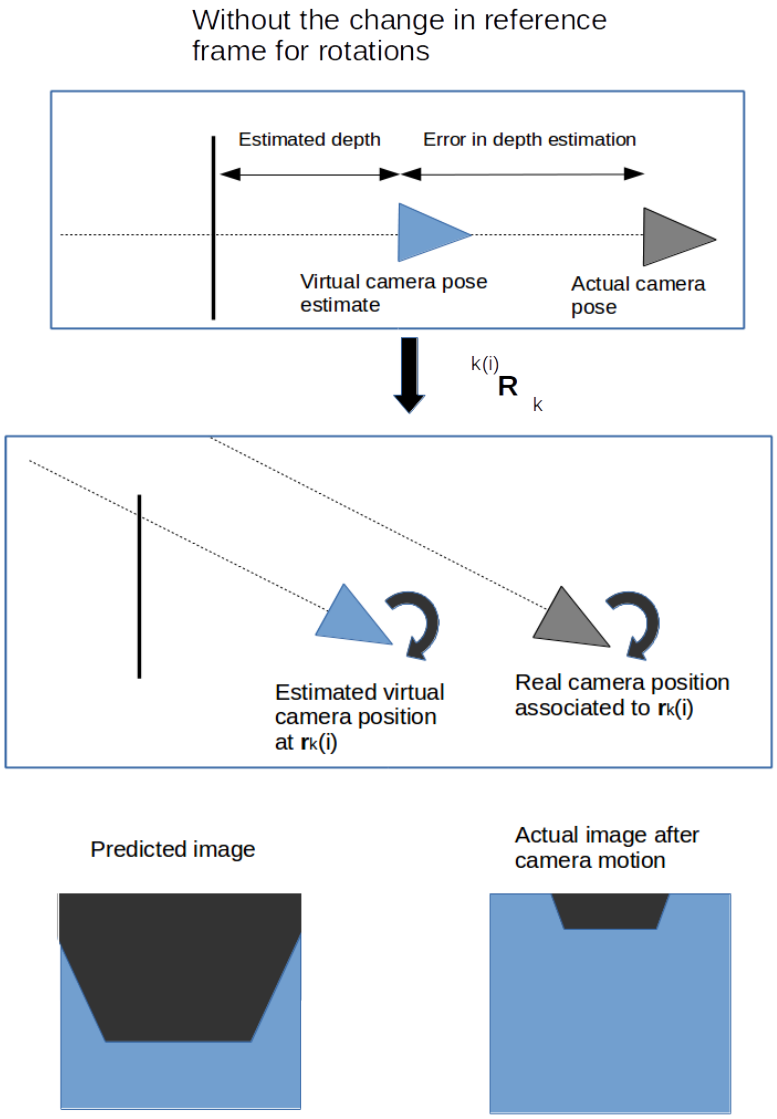


Figure 4.4: *Applying a pure rotation transformation without changing the reference frame with badly estimated depth results in an inadequate camera orientation*

that can be compensated through a closed loop control, as the positioning task will then solve accurately the errors on the other DOF first. Once only the error along the optical axis remains, the projection will be able to determine the direction (not the amplitude, due to the unknown depth) the camera has to move to reduce this depth error.

Building on Eq. (4.22), we denote ${}^{k(i)}\tilde{\mathbf{H}}_k$ the modified image transfer function, expressed such as:

$$ {}^{k(i)}\tilde{\mathbf{H}}_k = \mathbf{T}_2 {}^{k(i)}\mathbf{R}_k \mathbf{T}_1 + \frac{{}^{k(i)}\mathbf{t}_k}{d}\mathbf{n}^\top \tag{4.24} $$

where $\mathbf{n}$ is the unit normal vector of the plane associated to the scene, and d is the distance to the origin of the scene plane expressed in camera frame $\mathcal{F}_{(k)}$ and where $\mathbf{T}_1$ is an intermediary transformation matrix defined such as:

$$ \mathbf{T}_1 = \begin{pmatrix} \mathbf{I}_{3\times3} & -d\mathbf{n} \\ \mathbf{0}_{3\times1} & 1 \end{pmatrix}, \tag{4.25} $$

where $\mathbf{n}_{rotated}$ is the normal vector $\mathbf{n} = (n_x, n_y, 1)^\top$ associated in the image plane reference frame, rotated such as:

$$ \mathbf{n}_{rotated} = {}^{k(i)}\mathbf{R}_k \mathbf{n} \tag{4.26} $$

and $\mathbf{T}_2$ is a second intermediary transformation matrix defined such as:

$$ \mathbf{T}_2 = \begin{pmatrix} \mathbf{I}_{3\times3} & d\mathbf{n}_{rotated} \\ \mathbf{0}_{3\times1} & 1 \end{pmatrix} \tag{4.27} $$

Direct method: penalizing particles associated with poor image data

When applied to dense methods, one additional step have to be considered to evaluate particles. As the cost function for dense methods is computed by defining a distance between a desired image and a current image, one need to ensure that both images contain relevant image data. The desired image is constant, hence, no evaluation is necessary. The current image, for our Particle Filter, is a predicted image, created through an image transfer function. An image transfer function can only output, at most, as much image information as present in the original image. This means that it is also possible to transfer every pixels on this original image outside of the viewpoint of the virtual camera, creating an image with zero information. A cost function estimate computed on such an image would thus be irrelevant and should not be used, as it will only introduce noise in the filter and degrade the overall estimate.

To ensure that enough information is contained in the images resulting from the image transfer function, we use a straightforward method: by measuring the amount of information

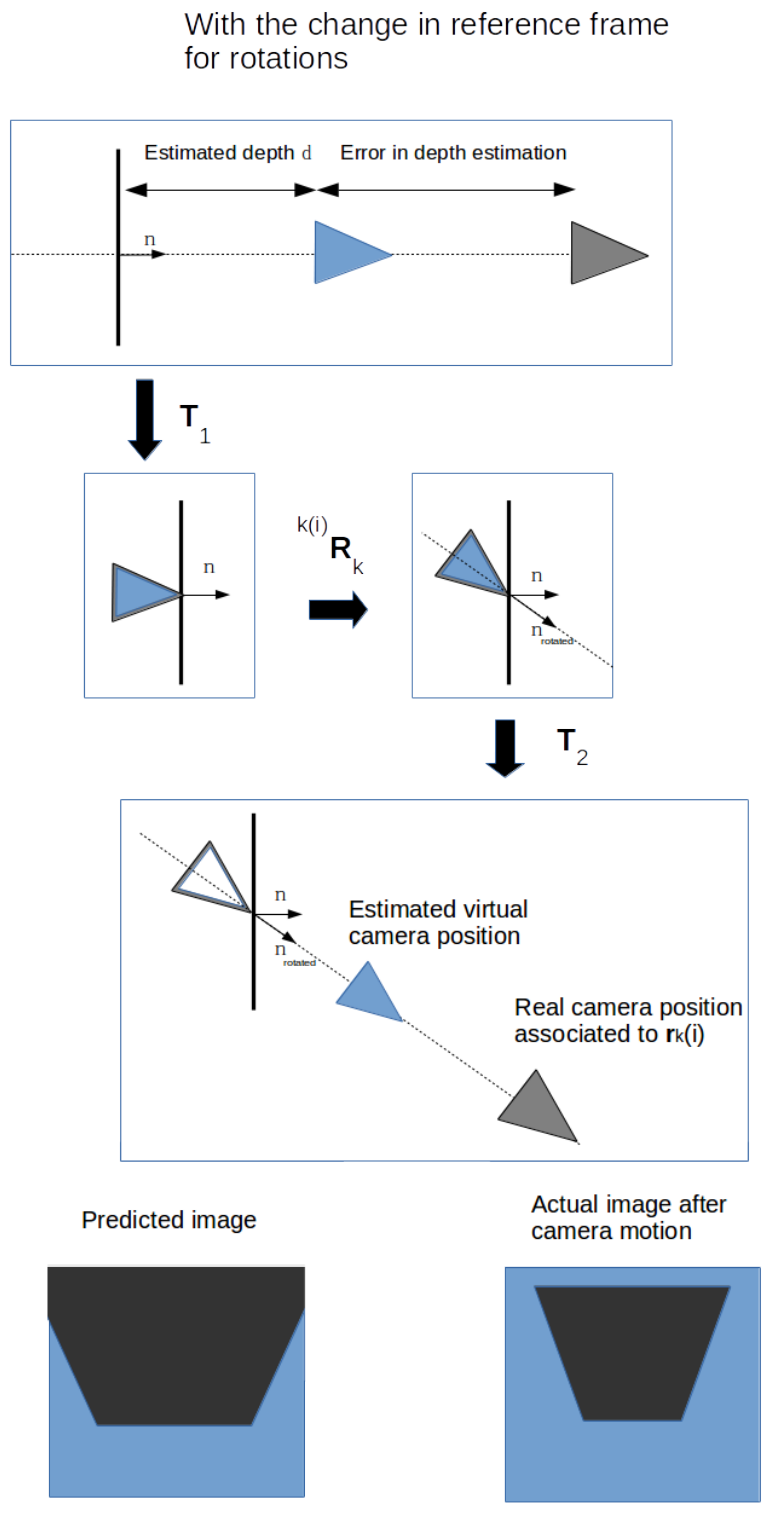


Figure 4.5: *Applying a pure rotation transformation with a change in the reference frame with badly estimated depth conserves the expected camera orientation*

transferred outside of the result image (represented as a black area in Fig. 4.6), it is possible to evaluate the relevance of the generated image.

A threshold is set, and every image with an amount of unknown information above this threshold is considered uninformative, and the corresponding particle is discarded and resampled. For all the following experiments, the threshold as been set empirically to a maximum value of 10% unknown information.

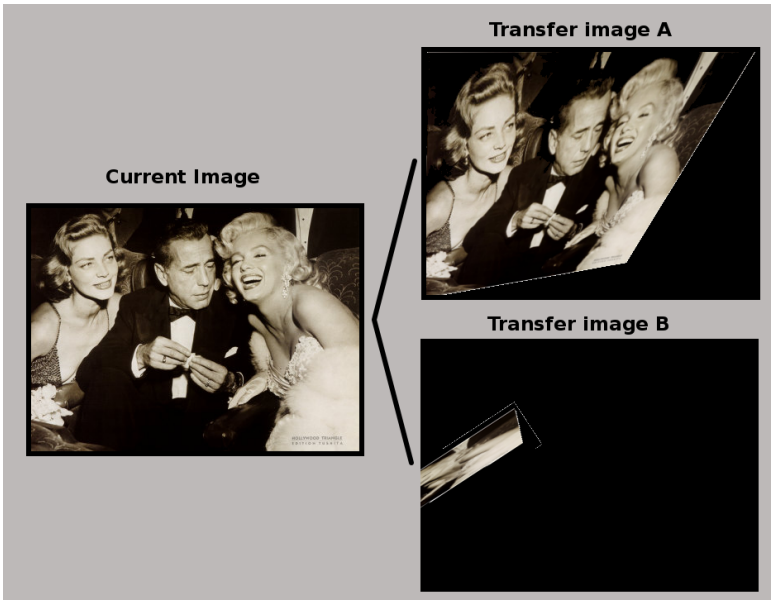


Figure 4.6: *Transferring an image into a suitable image and an uninformative one*

PF-based control law

The general scheme that defines the control law can be expressed as follows:

- Initialization of the PF with a large variance in order to discover the convex zone (Search Phase);
- At the beginning of the control loop (iteration 0), the variance and number of particles is reduced (Servoing Phase):
  - perform an iteration of the PF: draw N_p particles $\mathbf{r}_k(i)$ and compute their weights $\pi_k(i)$ according to the chosen visual features (in our case the accumulated Euclidean distances between 2D points for the point-based VS, or the SSD between

full images for the photometric VS). From these particles, select the one with the highest weight: $\mathbf{r}_{k,best}$

– compute the camera velocity to drive the camera toward the pose defined by $\mathbf{r}_{k,best}$, such as

$$\mathbf{v} = \lambda \, \mathbf{r}_{k,best} \, \rho\big(\mathbf{s}(\mathbf{r}_{current}), \mathbf{s}^*\big) \tag{4.28}$$

where $\mathbf{r}_{k,best}$ is the position of the particle of highest weight in the current camera frame and λ is a gain that determines the amplitude of the system velocity. The weighting by the current measure of the cost function allows the velocity to decrease evenly when approaching the optimal position. $\rho\big(\mathbf{s}(\mathbf{r}_{k,best}), \mathbf{s}^*\big)$ is the cost function measured between the visual features at the current pose, and the visual features at the desired pose. This last term allows the motion to have a better dynamic: high velocities while the camera is far from the desired pose and a slower displacement as it get closer to the target pose.

- When the difference between the estimated cost of the highest-weight particle and the current SSD cost is small enough (a threshold is determined empirically), we consider that we are close enough to the optimal position and we switch to the classical control laws. This provides the benefit of the increased positioning accuracy by allowing us to overcome the discretization error of the PF that would lead to a less precise final position. An alternative solution would be to increase the number of particles by a large amount, but this proves to be prohibitive in terms of computational costs.

From this generic control law, multiple methods can be implemented, depending on the cost function that one wants to estimate through the PF. In this chapter, we present two applications: one using a cost function based on geometrical features, namely points, and a second method using the raw image pixels.

Point-based PF

As seen in Section 2.2.1.2, for this method, the chosen feature is a set of geometrical points $\mathbf{s}(x(\mathbf{r}))$ extracted and tracked in the image through the experiment. These points are defined by their 2D coordinates $x(\mathbf{r})$. The cost function to be estimated is the Euclidean distance between this set of features at current frame, and the same set of features at a desired configuration $\mathbf{s}^*(x(\mathbf{r}))$.

Direct-based PF

In this second method, the chosen feature is a raw pixel array $\mathbf{I}(\mathbf{r})$. The classical cost function defined for this feature is the SSD distance between an array recorded at the current camera position, and an image $\mathbf{I}^*$ recorded from the desired camera configuration.

Experimental validation

In this section, we solve a VS positioning task by using Particle Filter-based control laws. Two methods are presented, one using geometrical points features, and one using a dense approach.

The task to solve in this set of experiments is a positioning task: the system is given features recorded at a desired camera configuration (pose in 6DOF) and, given an arbitrary initial camera pose, the system has to servo the camera toward this desired pose.

PF-based VS positioning task from point features in simulation

When performing point-based visual servoing, under some configurations, singularities can occur and prevent the optimization process to converge properly toward the global minimum. One interesting configuration that leads to a singularity when optimizing directly the Euclidean distances between point positions is to set as initial configuration the points such as they form a square, and set as a target configuration the same square formation, but with a 180° rotation. As illustrated by Fig. 4.7, the gradient descent-based optimization will try to drive each point directly toward its target position, which will lead to the camera moving in a pure translation motion toward the back of the camera, to infinity. This singular configuration and others are presented in [Chaumette 98b].

By running directly this classical control scheme in an experiment, we can confirm this behaviour, as showed in Fig. 4.8: the positioning error is increasing as the camera moves farther and farther away from the desired position, until the experiment is stopped. In order to succeed in this configuration, one has to change the cost function by changing the representation of the points distances by defining the problem in a different geometrical way, like the distance between lines passing through each pair of points. It can be noted that the implementation of the method explains the drift that can be seen on the t_x and t_y translations: the localization of the points in the image are obtained through an algorithm tracking the dots in the camera image. During the experiment, as the dots get smaller, the tracker performance decreases and

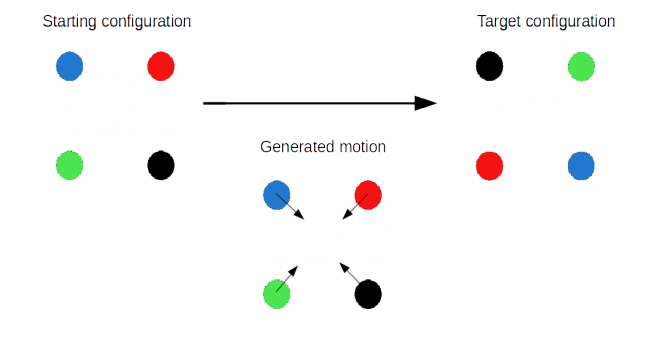


Figure 4.7: *Singular 4-points configuration*

the estimation of the dots positions get less accurate, inducing unwanted translational motions from the control law.

We applied our method of PF-based control to this configuration, and as expected, it was able to solve it as showed in Fig. 4.9 using points as visual features. This is due to the fact that the solution (applying a rotational motion of 180°) is contained in the local state space that is sampled by the PF, unlike the solution followed by the gradient descent. The PF is then able to find a more suitable trajectory in the state space, here the pure rotational motion. The experiment also highlight the limited precision due to the sampling of the search by the PF: instead of computing an ideal motion where the only component with non-zero velocity is on r_z, which should result on a trajectory restricted to a single point. We can see that a noise exists on the other components (which creates a noisy motion trajectory around the initial/desired position), although its amplitude is limited compared to the r_z velocity component. This noise's amplitude is directly linked to the number of particles used (and incidentely to the computational cost of the method).

In a second pair of experiments, we show that our method is able to solve a positioning task with a similar 180°, but with additional positioning errors on all 6 DOF: the initial displacement is such as $\Delta\mathbf{r}$ =(-8cm, -3cm, -18cm, -20°, 20°, 180°), with a final distance to the image plane of 20cm. Although the classical gradient-descent based control law fails to converge, see Fig. 4.11, our method succeeds to reach the desired pose, see Fig. 4.10. We can see that the system exhibits a smooth behaviour in the first part of the run. In the second part however (from iteration 3000), the behaviour becomes more noisy: this is due to the relatively low number of particles used in the PF (here 100 particles), considering the state space to explore. The result is a sparse sampling of the space, that induces a low precision of the cost function estimation around the optimum. As only a single DOF component (r_y) remains with a large error, the other components values are below the precision of cost function estimation, inducing a noisy oscillation around each of the other 5 components' optimum values. This is also highlighted by the trajectory graph, with a very straight motion toward the desired position area in the first

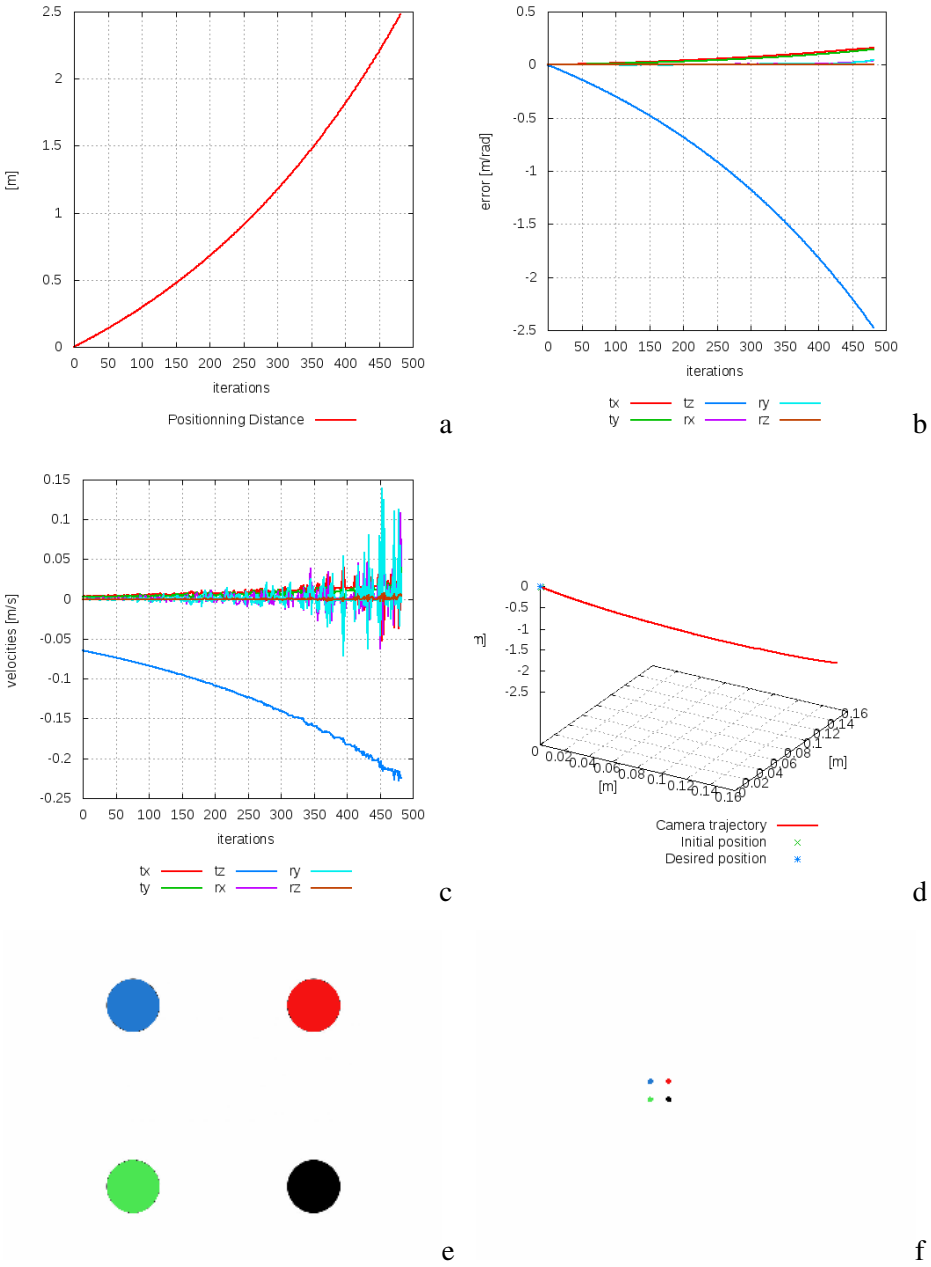


Figure 4.8: *Classical point-based VS control. (a) Positionning error. (b) Translational and rotational errors. (c) Camera velocities. (d) 3D trajectory (e) Initial configuration. (f) Configuration at the end of the motion.*

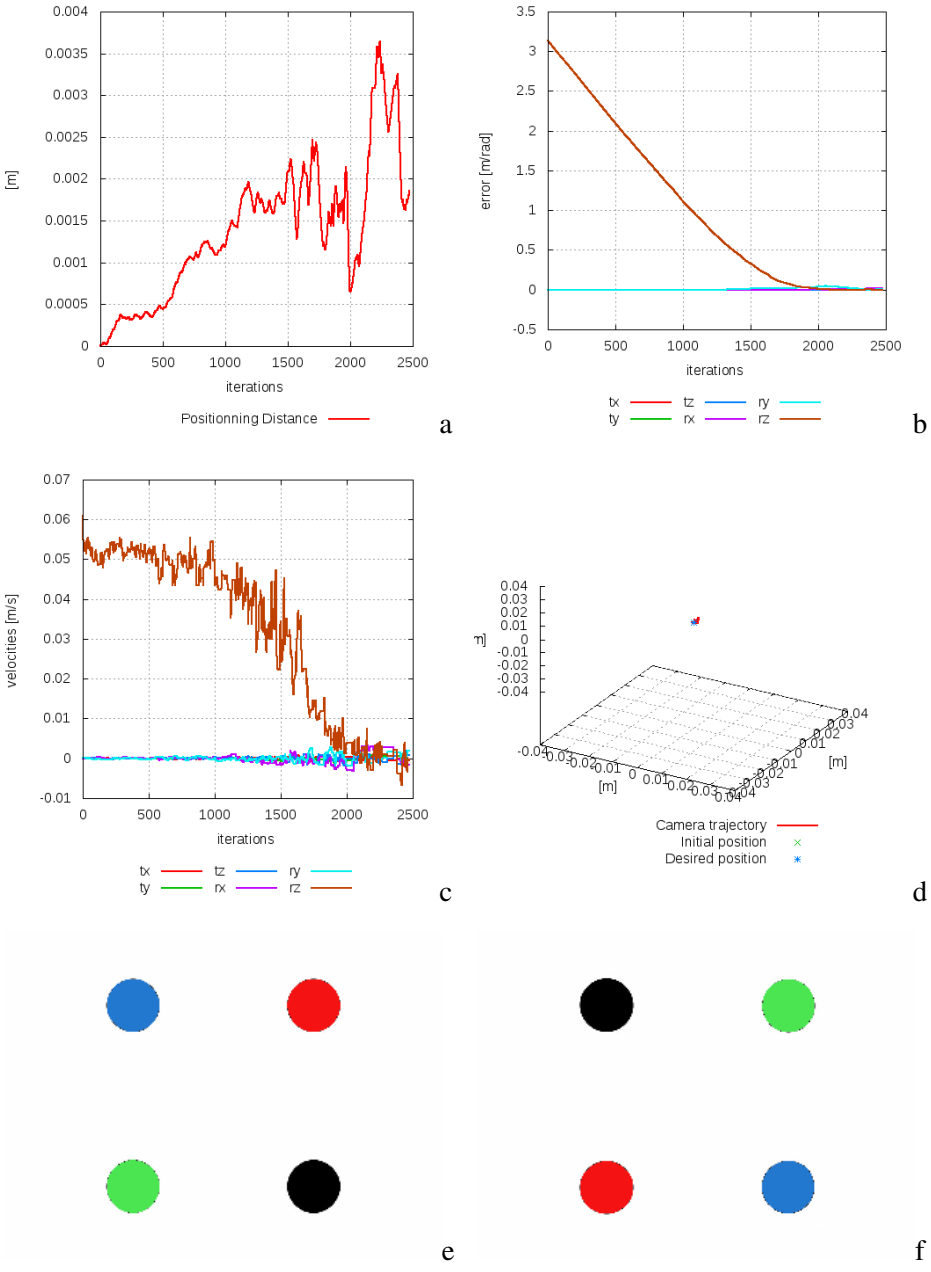


Figure 4.9: *PF-based VS control. (a) Positionning error. (b) Translational and rotational errors. (c) Camera velocities. (d) 3D trajectory (e) Initial configuration. (f) Configuration at the end of the motion.*

part of the run, and a second part with very sub-optimal oscillations as the last rotational component is converging. A way to solve this issue could be to introduce a damping term in order to reduce the variance associated to the particle spread as the component values are decreasing, leading to a better sampling of the state space.

There exist other known difficult cases, presented in [Chaumette 98b]. In the following experiment, the solving of the task involves performing large tx/ry and ty/rx motions. The issue is that for each of these two motions, there exists a coupling in the motion of the features in image space. This coupling makes the cost function associated to this motion ill-defined, with the presence of local minima. The initial positioning error is such as $\Delta\mathbf{r}$ =(-10cm, -10cm, 0cm, -40°, -40°, 0°) , with a final distance to the image plane of 20cm. The result of this experiment can be seen at Fig. 4.12.

PF-based VS positioning task from dense feature on Gantry robot

This set of experiment applies the same PF-based control law, this time with a dense cost function. The images are this time compared as a Sum of Squared Distances (SSD) between the two pixel arrays that represent the current image and a desired image corresponding to a target configuration.

In this experiment, we are interested in dealing with a large initial displacement, especially in terms of rotation, which has a significant effect on the SSD error value and make impossible to solve by the traditional direct SSD-based visual servoing (at this starting position, the cost function is non-convex and highly non-linear and therefore cannot be optimized properly with Gauss-Newton related methods that rely on such properties), as seen in Fig. 4.13 where we try unsuccessfully to solve this positioning task with the direct visual servoing method presented in Chapter 2. The initial pose consists of an offset from the desired pose of $\Delta\mathbf{r}$ =(-10cm, -40cm, 18cm, -52°, 2°, 10°), with a depth value of 80cm at the desired position.

We can see that the proposed control scheme succeeds in converging efficiently toward the desired position, as seen in Fig. 4.14 with a continuous decrease in positioning error. The final recorded position error is less than 1mm. At the end of the motion (here at the iteration 1404), when the offset between the best predicted error and the currently recorded error is small enough, we switch to the direct visual servoing control in order to reach the desired position with a sub-millimetric precision.

A second positioning experiment has been performed to apply the control law to a second scene setting, with a new initial pose, to show the robustness of the proposed method. The initial position error is of $\Delta\mathbf{r}$ =(18cm, -21cm, -8cm, -14°, -13°, -21°). The run of the experiment

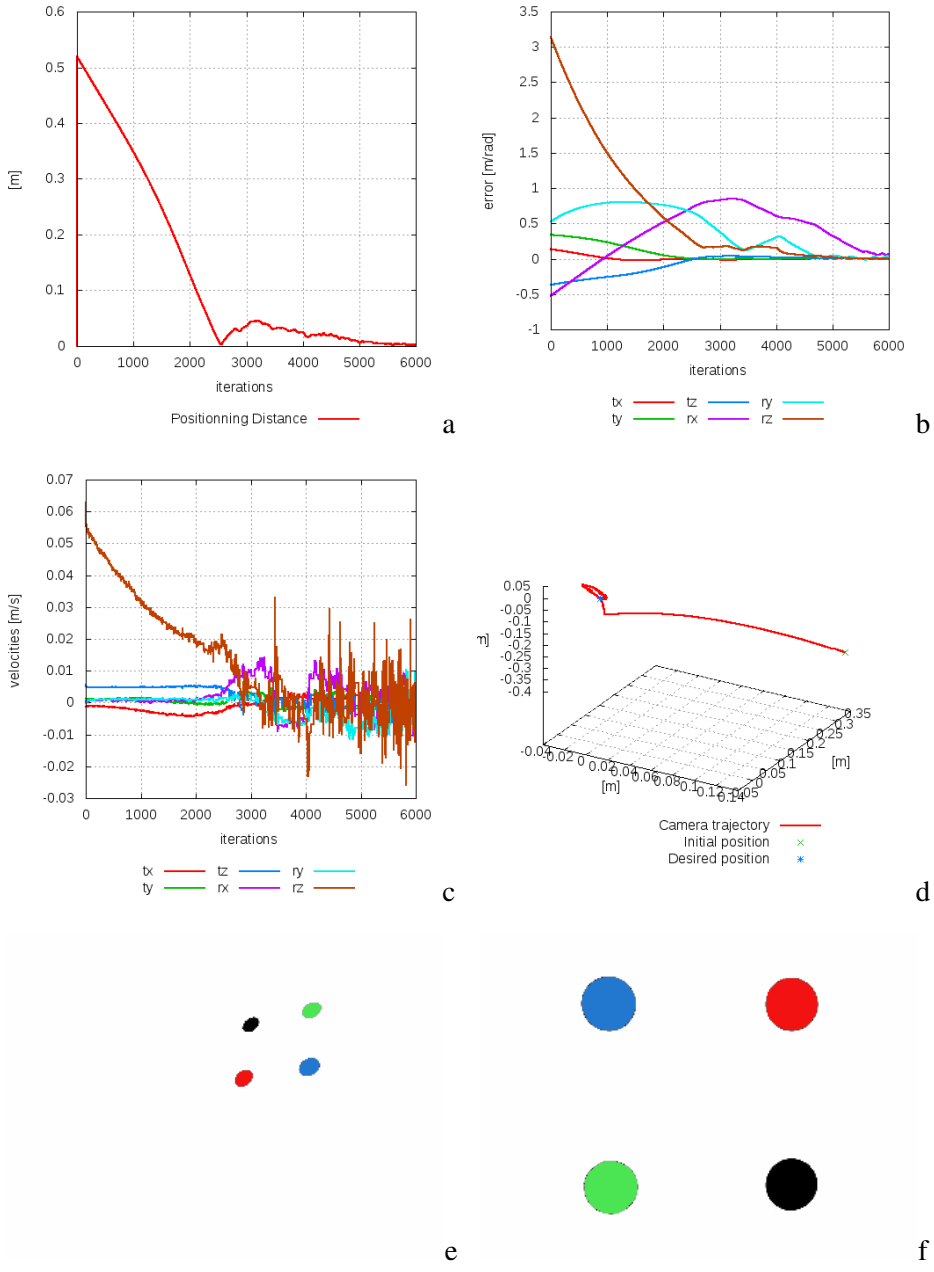


Figure 4.10: *PF-based VS control. (a) Positionning error. (b) Translational and rotational errors. (c) Camera velocities. (d) 3D trajectory (e) Initial configuration. (f) Configuration at the end of the motion.*

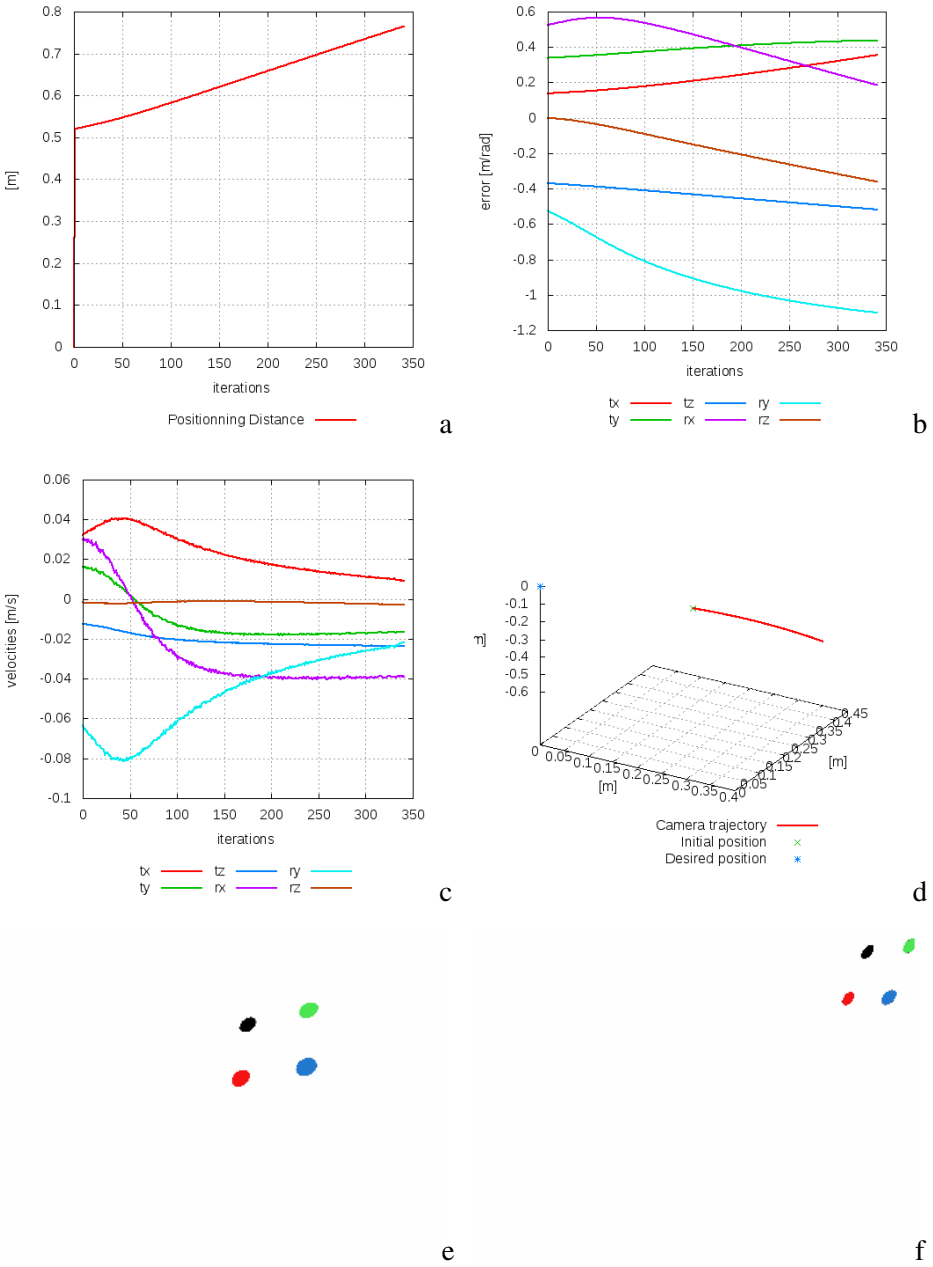


Figure 4.11: *Classical point-based VS control. (a) Positionning error. (b) Translational and rotational errors. (c) Camera velocities. (d) 3D trajectory (e) Initial configuration. (f) Configuration at the end of the motion.*

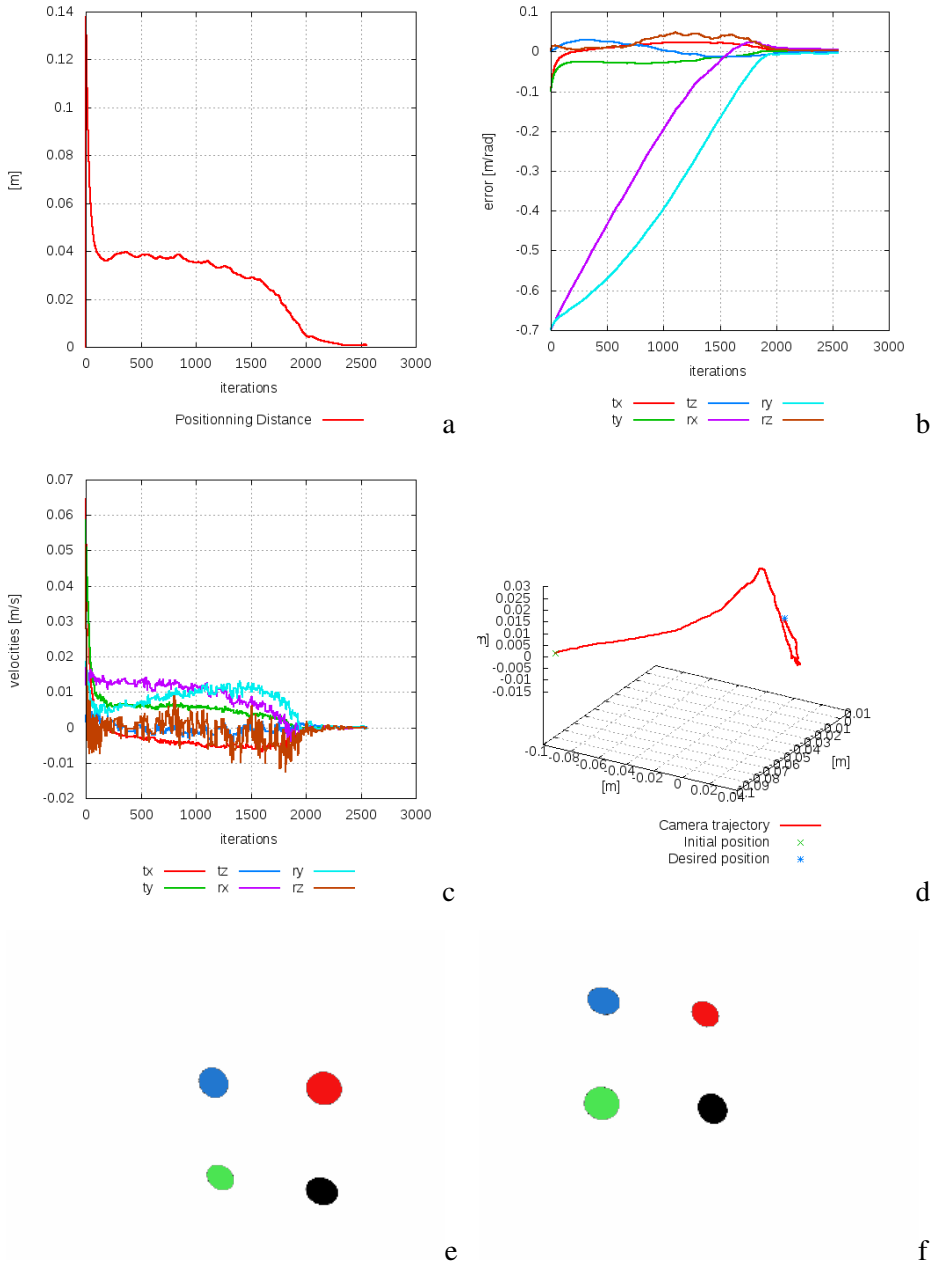


Figure 4.12: *Classical point-based VS control. (a) Positionning error. (b) Translational and rotational errors. (c) Camera velocities. (d) 3D trajectory (e) Initial configuration. (f) Configuration at the end of the motion.*

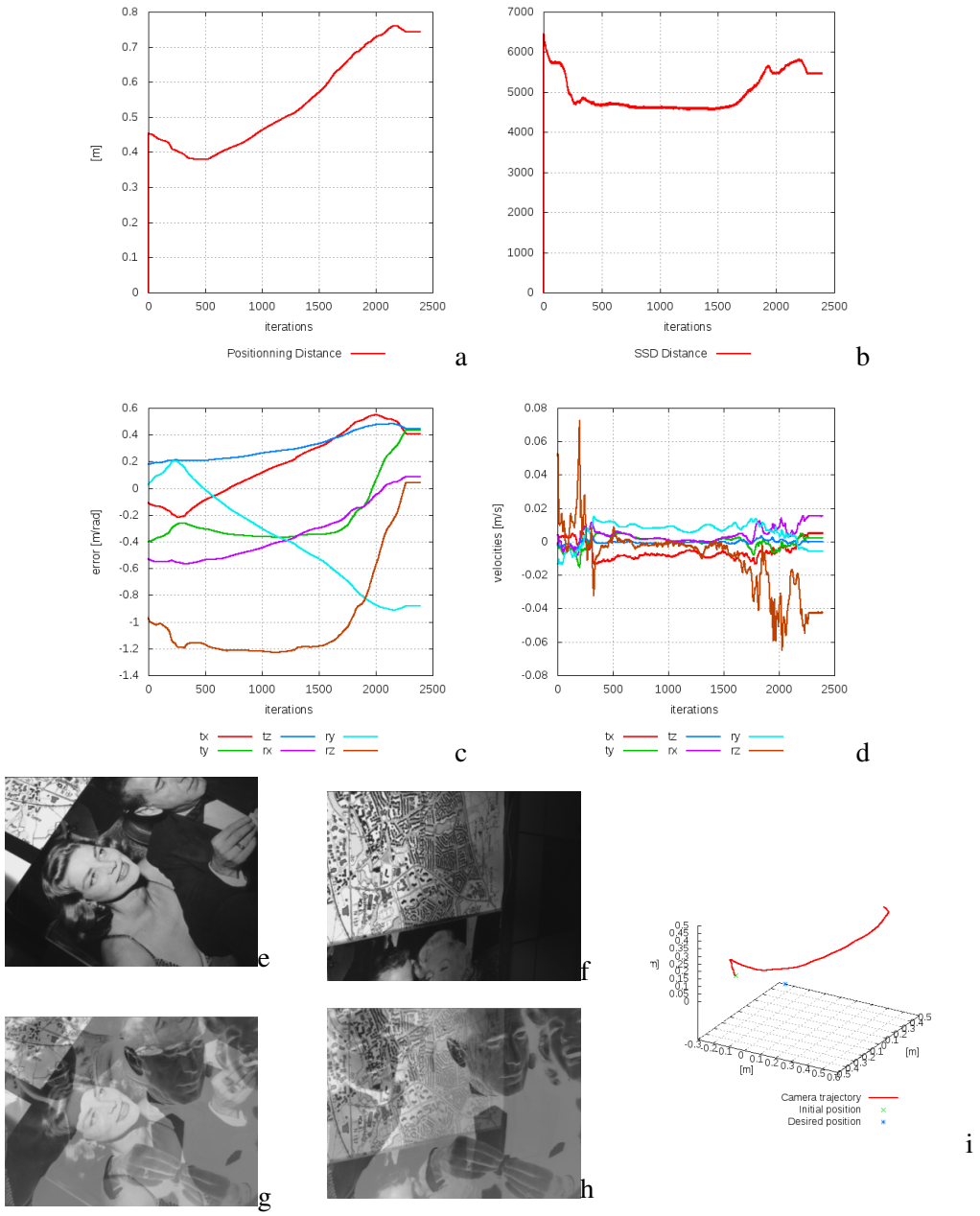


Figure 4.13: *Classical direct visual servoing. (a) Positionning error. (b) SSD distance. (c) Translational and rotational errors. (d) Camera velocities. (e) Initial image. (f) I at the end of the motion. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D tajectory.*

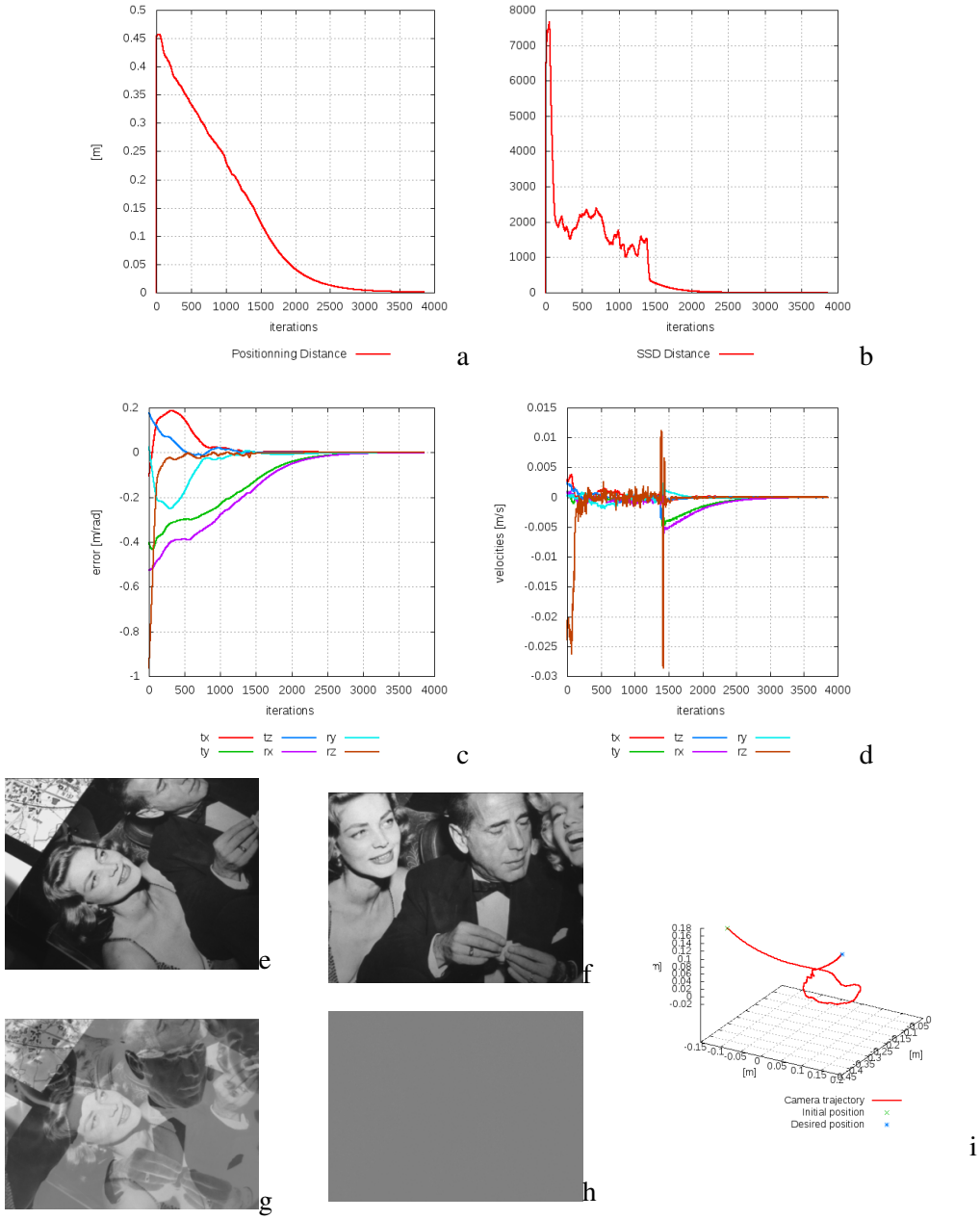


Figure 4.14: *Hybrid visual servoing, first PF-based, then by classical direct VS control (switch at iteration 1404). (a) Positionning error. (b) SSD distance. (c) Translational and rotational errors. (d) Camera velocities. (e) Initial image. (f) I at the end of the motion. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D trajectory.*

can be seen in Fig. 4.15. This scene is challenging, as some materials in the scene are slightly reflective, creating areas where pixel intensities will vary depending of the viewpoint. In the same way as in the previous experiment, once the SSD error is low enough, we switch to the classical direct visual servoing method to achieve a high final precision. This switch here occurs around iteration 700. By comparing the evolution of the positioning distance (Fig. 4.15(a)), and the evolution of the SSD distance (Fig. 4.15(b)), we can see that the cost function is not convex following the camera trajectory, highlighting again the ability of the PF to predict states with low cost function values, independently of the shape of the cost function between this predicted state and the current state of the system.

Statistical comparison between PF-based and direct visual servoing in simulated environment

Since this new method is validated on a real robot experiment, further investigation is required to quantify the advantage in terms of increased convergence area. To this end, we performed a statistical analysis of the convergence successes in a simulated environment. The process is similar to the one performed in the previous chapter to compare the histogram-based control laws. Fig. 4.16 shows the result of this process with 10 increases in the spatial noise variances. 40 runs are performed to get the success rate percentage at each noise iteration. The distance from the camera to the image plane has been set to 20cm and the noise's variances are such as:

- from 0 to 4cm for the x/y translations;
- from 0 to 2cm for the z translation;
- from 0 to 10° for the rotations around the x/y axis;
- from 0 to 50° for the rotation around the z axis.

The rotation transformations being, as usual, the most difficult to handle.

We can clearly see from the resulting curves that the proposed method has much better performances than the direct visual servoing method in terms of convergence radius, especially when strong rotations around the optical axis are involved. It can also be noted that the presented performances could be increased at the cost of additional particles, allowing to sample the state-space farther, with additional computational power.

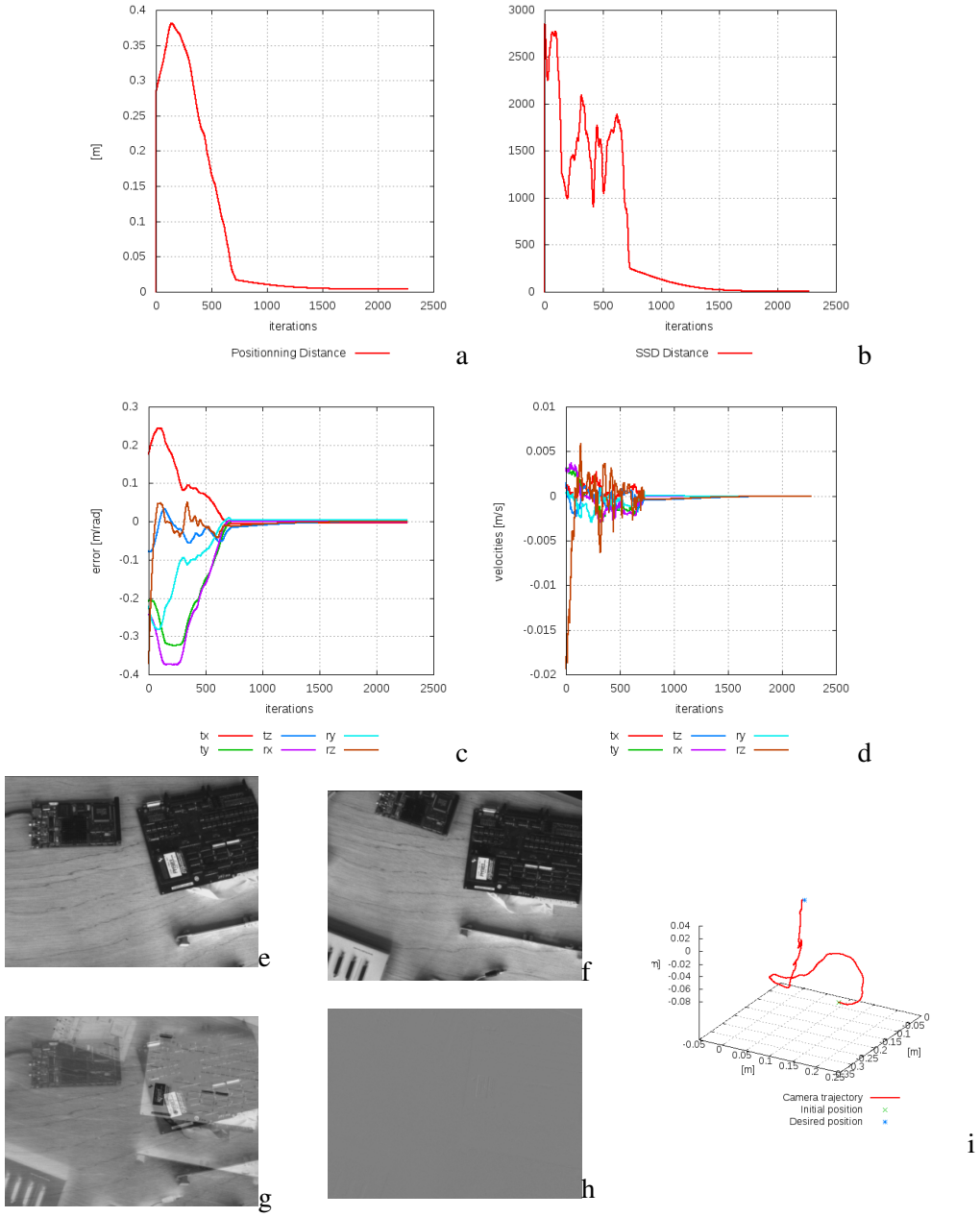


Figure 4.15: *Hybrid visual servoing, first PF-based, then by classical direct VS control (switch at iteration 700). (a) Positionning error. (b) SSD distance. (c) Translational and rotational errors. (d) Camera velocities. (e) Initial image. (f) I at the end of the motion. (g) I - I* at initial position. (h) I - I* at the end of the motion. (i) 3D trajectory*

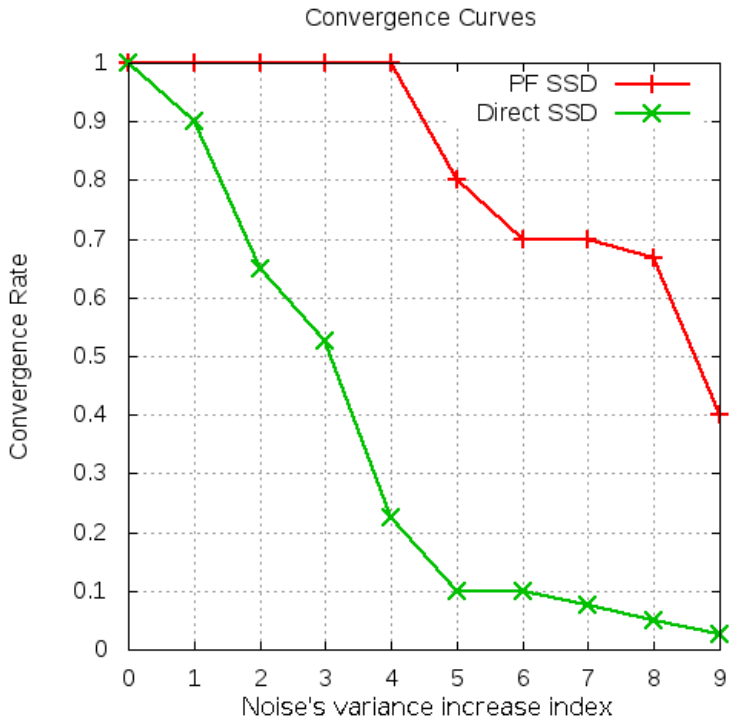


Figure 4.16: *Convergence rates of direct visual servoing methods with PF-based and classic control laws*

Conclusion

In this chapter we presented a new approach to perform optimization within a visual servoing scheme in order to be coupled with the traditional gradient descent. We showed that this optimization scheme, based on a Particle Filter, can be integrated in any visual servoing technique that can be formulated as an optimization problem. This was done by providing a method to represent particles as virtual camera positions, for which predictions of the features can be computed through image transfer techniques. This was validated experimentally on a point-based visual servoing scheme, as well as on a direct visual servoing scheme. These experiments showed that although the computational cost can be important if a high precision is required, it can provide significant improvements and robustness by ignoring local minima as long as the global minimum is within the state space that is sampled. We showed that coupling this method with the classical gradient-based methods in a two-step fashion allows to get the best of both worlds: a large convergence area, as well as a good end-precision.

A simulation benchmark is also provided to compare the performances of the classical and PF-based direct visual servoing methods in term of convergence area, confirming the improvement brought by the PF-based control law.

These works have been published in [Bateux 16] at the IEEE IROS16 conference.

Chapter 5

DEEP NEURAL NETWORKS FOR DIRECT VISUAL SERVOING

Motivations

As seen in the previous chapters, defining both cost functions and optimization strategies is critical for the end performances of VS control schemes. Many VS methods have been proposed by hand-crafting features and cost functions in order to provide control laws with the best trade-off between robustness to noises, optimal behaviour, decoupled DOF, precision and size of the convergence area... etc. These trade-offs exist because the link between the image and the camera motion is not trivial, requiring complex models to achieve the desired levels of performances with a limited amount of computational power. The initial motivation to investigate the application of machine learning techniques to visual servoing is to benefit from the capacity to avoid this manual modelization step. Indeed, the machine learning field provides ways to automatically create models from a set of data. In this way, from images taken from examples tasks, such a system could model adapted image features that would not require additional processing steps.

Over the last years deep neural networks, especially Convolutional Neural Networks (CNNs), have progressed the state-of-the-art in a number of computer vision tasks, for extracting semantic meaning as image classification for object recognition [Krizhevsky 12] or to retrieve geometrical information, like the inference of depth maps from a single RGB image [Eigen 14],

displacement computation through homography estimation [DeTone 16], or camera relocalization [Kendall 15].

Deep learning has also started to become more prominent in robotics. Recent approaches to solve grasping tasks have achieved promising results. Complex positioning tasks can be learned, either without the use of any prior knowledge [Zhang 15], or with the use of deep reinforcement techniques [Levine 16, Finn 16].

These progresses have been fueled by the availability of common deep learning libraries, such as Caffe [Jia 14]. Such tools make possible to construct, train and share deep neural networks and to build on networks already trained on very large datasets of millions of images. It is now possible to re-purpose existing networks (with robust global descriptors already embedded in the lower layers) by using fine-tuning techniques (such as in [Karayev 13]). During fine-tuning, only the task-specific upper layers of the network are re-trained to perform a different task. This allows for a fast training of networks with a limited amount of data (for example a few hours with a few thousand data points), instead of days and millions of data points to train the same network architecture from scratch for the very same end-task.

A hindrance to wide spread use is that these systems require large datasets and long training times. These critical points are mitigated by our proposed approach.

Basics on deep learning

In this section we will present a short history of machine learning and then focus more specifically on convolutional neural networks (CNNs). We will then present an overview of the notion of supervised learning and how this notion is applied to CNNs. This part does not pretend to be exhaustive or go in-depth in every aspects of the design and uses of artificial neural networks. It aims instead to provide enough insights and vocabulary for the reader to be able to go further with more detailed works, such as [LeCun 15].

Machine learning

Machine learning is a field of computer science interested in developing methods for interpreting and predicting data in an automated manner. It originally stems from researches focused on pattern recognition that aimed to allow algorithms to learn automatically from massive quantities of data. Applications of these methods are nowadays used broadly in the internet services companies, for things as varied as e-commerce, content filtering on social networks or web

searches. Detecting patterns in these fields often translates into identifying clusters of similar user profiles, in order to be able to predict the actions or preferences of any particular users and to propose them targeted services such as ads, products or relevant content. This family of method has extended way beyond the scope of pattern recognition.

Over the decades of existence of the machine learning field though, the methods for performing this automatic pattern discovery process were relying a lot on hand-crafted rules and domain-knowledge from human experts were required in order to interpret the data and infer results. This set of methods, such as expert systems, rely on an expertise that is 'transferred' from human(s) to the computer.

The major difference between these early expert systems and the current set of methods based on artificial neural networks is the ability for algorithms to infer high-level representations directly from raw data without relying on hand-crafted human knowledge. Indeed, expert systems are fed data that has been pre-processed into more exploitable 'feature vectors', on which the system can perform pattern recognition more easily as the data is disambiguated into a more tractable representation. This process is similar to the methodology presented earlier in this thesis, where a feature defining a cost function are computed from raw image data in order to be used later by a control algorithm. One major issue with this approach is that the proposed data representation is limited by the breadth and depth of the original human knowledge and is not guaranteed to exploit fully the data. Indeed, the representation may be too simplistic or biased, failing to exploit parts of the data that could prove critical to the solving of the task optimally but that were not identified as important by the human designer. This particular problematic of the limitation of human knowledge has been recently highlighted by a series of articles from Google Deepmind: they proposed initially a new method to train an algorithm to learn the game of Go by using a dataset of games played by top human players. The resulting program succeeded to beat the strongest pro players [Silver 16]. In a further iteration of the method, they modified the training procedure to use only simulated games (meaning no records of human games at all) between various versions of the program [Silver 17]. The results showed that this new iteration was able to surpass the previous version within a shorter training time, with less computational resources.

Recently, the deep learning set of methods has allowed the automatic learning of representations, also called models, from extensive datasets. One prominent example is the nowadays 'ubiquitous' Convolutional Neural Networks (CNNs or ConvNets) approach. The strength of deep learning methods is the ability for those systems to learn the whole chain of recognition (from raw data, directly outputed by the sensor, to the final task action): when fed raw data, it can generate its own representation and then use it to perform the pattern recognition/classification tasks. It does so by using multiple levels of representations that are obtained from simple mathematical operations (described later), the input being fed and transformed successively though those layers, abstracting the data progressively, further and further away from the

input data. These high-level representations serve internally to highlight the 'important' parts of the data related to the task at hand, and diminish the unwanted variability.

This methodology has been applied successfully to various problems related to previously difficult-to-analyze high-dimensional data types, such as raw speech recordings [Hinton 12], unprocessed images [Krizhevsky 12] or particle accelerators data points [Ciodaro 12].

When analyzing such a system trained on image recognition (the goal is then to identify the presence or absence of known classes of objects into a given image), one can notice the following progression of representation throughout the system: the raw pixels input evolves into a set of oriented edges detectors, into highlights of particular edge patterns of the image, then into a representation that detects arrangements of previously detected patterns, which goes then to a layer assembling these arrangements with respect to known objects classes (modeled by the network connections at training time), and finally a pattern detection layer that uses the previous representation to detect the presence and absence of objects in the image.

The important point in this example is that all the described representations are not hand-crafted but are learned directly from a learning procedure.

Supervised learning

In this section, we will concentrate on a specific type of training algorithm, namely the supervised learning methods. This set of methods relies on the possession of a set of annotated data, meaning that each data point of this dataset (for example an image) is associated to a result label (for example the type of object present in this particular image). By feeding this dataset to the system, it can be trained to perform the correct prediction of the labels when presented similar data (i.e. recorded from a similar sensor, with similar variability). A trained system can then be used to predict labels as long as the data show the same degree of variability as the training set (for example lighting changes, various object orientations...). Supervised learning is currently the most common way of training CNNs.

Practically, this consists on feeding the training data to the system iteratively, and comparing the prediction for each data point with the associated desired label, analyzing the potential error between the two, and modifying the internal parameters of the system to improve slightly the prediction. By performing this operation on large quantity of data, the system will converge toward a state where its prediction get more and more accurate. These internal parameters are real numbers called 'weights'. A typical CNN can consist of millions of those weights. In order to modify them, the following operation is performed: given the error between the prediction and the real label, an error gradient is computed for each weight, indicating the amount

of error that would increase if this particular weight where to be modified. Once this gradient is computed, each weight is adjusted slightly in the direction opposite to this gradient. Applying this operation actually optimizes what is called 'objective function', 'cost function' or 'loss', that represent the overall prediction error of the system on average on the whole training set. This training/optimization strategy is denoted as 'backpropagation of the gradient'.

Convolutional neural networks (CNNs)

CNNs are a particular form of 'artificial neural networks', a system inspired loosely by the architecture of brain cells in the visual cortex, where a neuron with a given state is linked to other neurons by synapses that allow for the passing of information. Artificial neurons (illustrated in the top-part of Fig. 5.1) are a simplified neuron representation reduced to these two components: a state, often a real number, and a set of links from and toward other neurons. These links pass on the state of the neurons, altered by a non-linear function, called 'activation function', that performs a signal modification on the state (operations such as $f(x) = max(0, x)$, called Rectified Linear Unit, or ReLU, proposed in [Hahnloser 00]). The links between neurons are also 'weighted', meaning that the importance of a link relative to the future state of the target neuron can be reduced or increased. These weights values are the variable parameter in the training process of CNNs. These neurons can be organized into 'layers', meaning a group of neurons, that can be connected with other layers in various ways: in a feed-forward manner (see Fig. 5.1) where no loop exists in the overall network so that data passes only once into a given neuron (CNNs belong to this category), in a recurrent type of architecture where the data can cycle through the network (useful for treating sequences of data, such as speech or text). Other kinds of architecture exist and discovering new ones is currently a very active research topic.

As neurons are organized into layers, it is also possible to define intermediary layers that contain the associated activation functions. For example, a ReLu layer between two neuron layers (l_1 and l_2) in an architecture means that all the outputs from l_1 are applied the $f(x) = max(0, x)$ operation, the results being fed later into the layer l_2. This also implies that the number of outputs of l_1 must be equal to the number of inputs of l_2.

This notion of layer stems from the fact that CNNs are 'feed-forward' types of networks, meaning that the information passed through the neurons cannot pass a single neuron more than once. This type of network can then be visualized as a succession of consecutive layers, each related to the layer before it as inputs, and the layer right after it that receives the modified output data. It has to be noted that the very first layer that receives the raw data is called 'input layer' and is the deepest layer, the last layer that outputs the predicted label the 'output layer' and is the upper layer, and all the layers in between are referred to as 'hidden layers'.

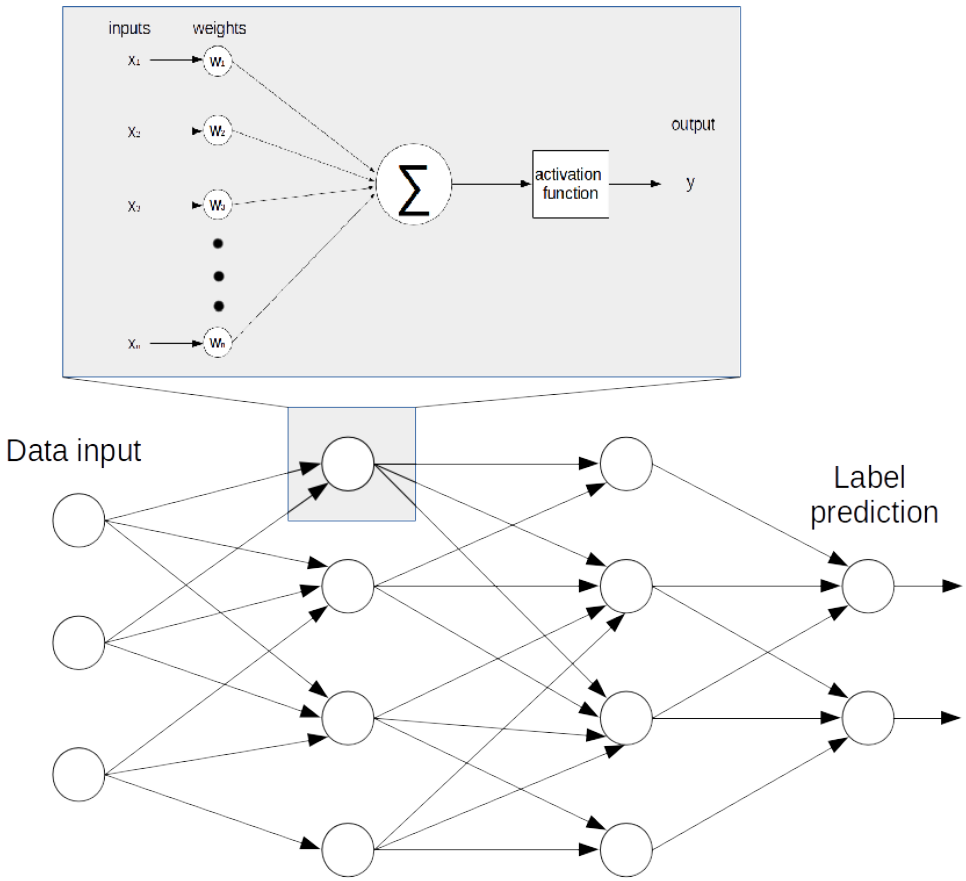


Figure 5.1: *Artificial neural network and neuron model*

CNNs essentially consist of a particular organization of neurons, creating what is commonly called 'convolutional layers'. The name of this type of network stems for the first operation that is performed in a convolutional layer: a convolution. It is defined by a filter size, and a stride value. The filter is then moved throughout the input array, shifted through the array by the stride value. This process is illustrated by Fig. 5.2.

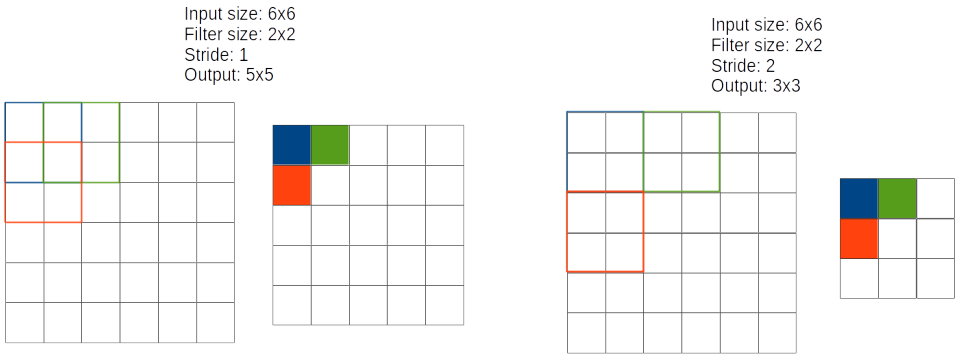


Figure 5.2: *Convolution operation*

Another type of layer commonly used in CNNs is the 'pooling layer'. The role of a pooling layer is to downsample the content of the layer. Several pooling techniques can be employed, the most common one being the 'max pooling'. Knowing the downsizing factor (also called the stride of the pooling layer), the input of the layer will be separated into sets of size equal to the stride, and each set will be converted into a single number, corresponding to the maximum value of the set.

A convolutional layer is the following association of sub-layers:

- a convolution operation;
- a pooling layer;
- a ReLU layer

The output resulting of this layer is then used as input for the next layer.

In CNNs, convolutional layers are used at the bottom of the network, in order to process the raw data and identify structures of higher level. In this part, we will consider one of the more straightforward architecture of CNNs, the architecture that popularized CNNs for computer vision by winning the ImageNet challenge in 2012 by obtaining more than twice the precision of the current state-of-the-art at the time: the AlexNet CNN, presented in [Krizhevsky 12]. This

architecture is composed of 8 layers: the data first pass through five consecutive convolutional layers, and the generated 'feature map' is then fed into three fully-connected layers. These layers simply consists of neurons that are each connected to every neurons of the previous and next layers; they serve to interpret the generated high-level features in order to predict the correct label associated to it. This architecture is illustrated in Fig. 5.3.

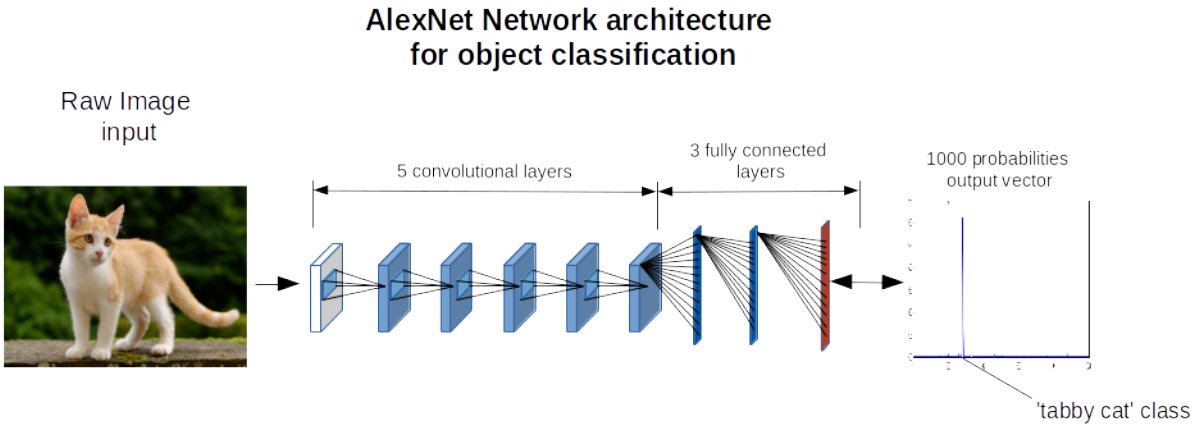


Figure 5.3: *AlexNet network architecture [Krizhevsky 12]*

CNN-based Visual Servoing Control Scheme - Scene specific architecture

This section will focus on detailing the way we propose to integrate the notions of visual servoing with a CNN-based optimization scheme.

A method is proposed to perform eye-in-hand VS from a CNN that estimates the relative pose between two images. This relative pose is then fed into a classical pose-based visual servoing control scheme [Chaumette 06]. The training of the CNN is performed through fine-tuning: taking a pre-trained CNN that is re-purposed to perform a new task, in our case relative pose estimation. The overall method is illustrated by Fig. 5.4. We propose a novel training process, based on a single image (acquired at a desired pose), which includes the fast creation of a large dataset using a simulator allowing for quick fine-tuning of the network for the considered scene. It also enables simulation of lighting variations and occlusions to ensure robustness within the learning phase.

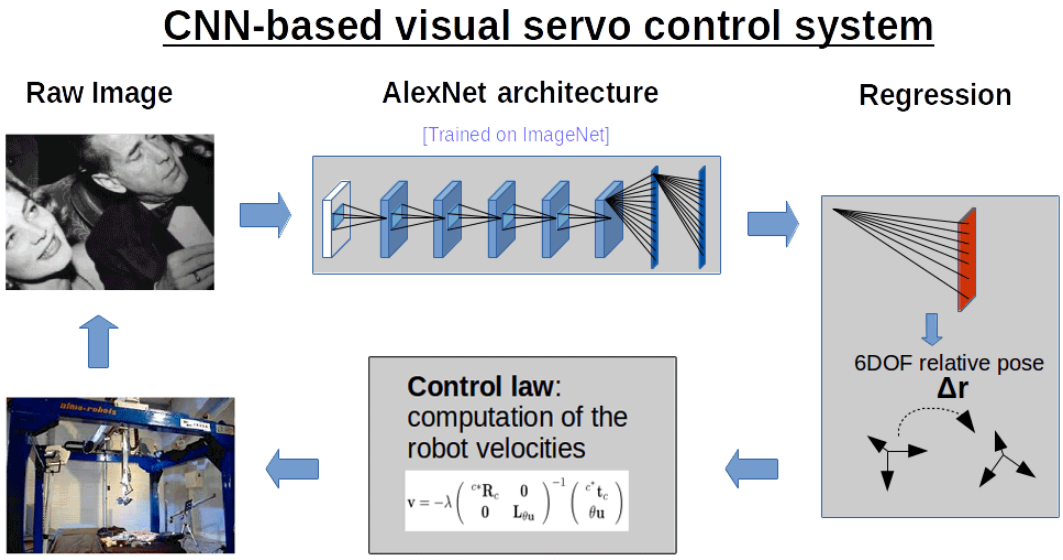


Figure 5.4: *Overview of the proposed CNN-based visual servo control system*

From classical visual servoing to CNN-based visual servoing

From visual servoing to direct visual servoing

As already stated, any VS can always be written as an optimization problem (eg, [Malis 04]). The goal of VS is that, from an initial arbitrary pose, the robot reaches the desired pose $\mathbf{r}^*$ that best satisfies some properties measured in or from the images. If we note ρ, the function that measures the positioning error, then the VS task can be written as:

$$\widehat{\mathbf{r}} \quad = \quad \arg\min_{\mathbf{r}} \rho(\mathbf{r}, \mathbf{r}^*). \tag{5.1}$$

where $\widehat{\mathbf{r}}$ is the pose reached at the end. VS can therefore be considered as an optimization of the function ρ where $\mathbf{r}$ is incrementally updated to reach an optimum of ρ at $\widehat{\mathbf{r}}$. If ρ is correctly chosen, the final pose $\widehat{\mathbf{r}}$ at the end of the minimization should be equal to the desired one $\mathbf{r}^*$.

For example, considering a set of geometrical features $\mathbf{s}$ extracted from the image, the goal is to minimize the error between $\mathbf{s}(\mathbf{r})$ and the desired configuration $\mathbf{s}^*$, which leads, by using as cost function the Euclidean norm of the difference between $\mathbf{s}$ and $\mathbf{s}^*$, to:

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}} \|\mathbf{s}(\mathbf{r}) - \mathbf{s}^*\| \tag{5.2}$$

The visual features $\mathbf{s}$ can be 2D feature leading to image-based VS (IBVS) or 3D features (such as the camera pose $\mathbf{r}$) leading to pose-based VS (PBVS). These visual features (points, lines, moments, contours, pose, etc) have thus to be selected and extracted from the images to control the robot degrees of freedom. In any case, it requires the knowledge or estimation of the interaction matrix $\mathbf{L_s}$ that links the temporal variation of visual features $\dot{\mathbf{s}}$ to the camera velocity $\mathbf{v}$ ($\dot{\mathbf{s}}(\mathbf{r}) = \mathbf{L_s}\mathbf{v}$).

To avoid the classical but error-prone extraction and tracking of geometrical features, direct VS has been introduced. Considering the image as a whole, the set of features $\mathbf{s}$ becomes the image itself [Collewet 11]: $\mathbf{s}(\mathbf{r}) = \mathbf{I}(\mathbf{r})$. The optimization process is then expressed as:

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}} \|\mathbf{I}(\mathbf{r}) - \mathbf{I}^*\| \tag{5.3}$$

where $\mathbf{I}(\mathbf{r})$ and $\mathbf{I}^*$ are respectively the image seen at the pose $\mathbf{r}$ and the desired image (both composed of N pixels). The main issue when dealing with direct VS is that the interaction matrix $\mathbf{L_I}$ is ill-suited for optimization, mainly due to the heavily non-linear nature of the cost function, resulting in a small convergence domain.

From direct visual servoing to CNN-based visual servoing

In this section, we propose to replace the direct VS approach described above by a new scheme based on CNN. Assuming here that a network has been trained to estimate the relative pose between the current camera view and a reference image. Given an image input $\mathbf{I}(\mathbf{r})$ and the reference image $\mathbf{I}_0$, let the output of the network be:

$$\Delta\mathbf{r_0} = net_{\mathbf{I}_0}(\mathbf{I}(\mathbf{r})) \tag{5.4}$$

with $\Delta\mathbf{r_0} = ({}^{c_0}\mathbf{t}_c, \theta\mathbf{u})$ the vector representation of the homogeneous matrix ${}^{c_0}\mathbf{T}_c$ that expresses the current camera frame with respect to the reference frame.

To reach a pose related to a desired image $\mathbf{I}^*$, the CNN is first used to estimate the relative pose ${}^{c_0}\mathbf{T}_{c^*}$ (from $net_{\mathbf{I}_0}(\mathbf{I}^*)$), and then ${}^{c_0}\mathbf{T}_c$ (from $net_{\mathbf{I}_0}(\mathbf{I})$), from which $\Delta^*\mathbf{r}$ is easily obtained using ${}^{c^*}\mathbf{T}_c = {}^{c_0}\mathbf{T}_{c^*}^{-1}\,{}^{c_0}\mathbf{T}_c$.

Expressing the cost function (in the machine learning domain, the term 'loss' is also commonly used instead of 'cost function') ρ as the Euclidean norm of the pose in Eq. (5.1), the minimization problem becomes

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}} \|\Delta^*\mathbf{r}\| \tag{5.5}$$

which is known to present excellent properties [Chaumette 06]. Indeed, the corresponding control scheme belongs to PBVS, which is globally asymptotically stable (ie. the system converges whatever the initial and desired poses are), provided the estimated displacement $\Delta^*\mathbf{r}$

is stable and correct enough. We recall that IBVS, and thus the schemes based on Eq. (5.2) and Eq. (5.5), can only be demonstrated to be locally asymptotically stable for 6 DOF (ie. the system converges only if the initial pose lies in a close neighborhood of the desired pose). With the proposed approach, the stability and convergence issues are thus moved from the control part to the displacement estimation part.

From $\Delta^* \mathbf{r}$ provided by the CNN, it is immediate to compute the camera velocity using a classical control law [Chaumette 06] :

$$\mathbf{v} = -\lambda \begin{pmatrix} {}^{c}\mathbf{R}_{c^*} \, {}^{c^*}\mathbf{t}_c \\ \theta \mathbf{u} \end{pmatrix} \tag{5.6}$$

By computing this velocity command at each iteration, it is then possible to servo the robot towards a desired pose solely from visual inputs.

We now need to build and train a CNN in order to implement Eq. (5.4).

Designing and training a CNN for visual servoing

To implement the approach described above, we need to train a network by feeding it a significant number of images $\mathbb{I}(\mathbf{r})$ for which we know the relative poses $\Delta\mathbf{r}_0 = ({}^{c}\mathbf{t}_o, \theta\mathbf{u})$ with respect to the pose $\mathbf{r}_0$ corresponding to $\mathbf{I}_0$. In order to keep training time and the dataset size low, we present a training method using a pre-trained network. Pre-training is a very efficient and widespread technique to build on CNNs already trained for a specific task. If a new task is similar enough (especially in term of inputs), fine-tuning can be performed on the CNN so it may be employed in a different task. Since we work on natural images in a real-world robotic experiment, a pre-trained AlexNet [Krizhevsky 12] was chosen as a starting point. This network was trained on 1.2 million images from the ImageNet dataset [Deng 09], with the goal of performing object classification (1000 classes). While we are not interested in image classification (instead we want to perform a regression task from an image to a 6DOF pose), works such as [Karayev 13] showed that it is possible to re-purpose a network by using the learned image descriptors embedded in the lower layers of an already trained AlexNet. Indeed, this process is based on the idea that certain parts of the network are useful to the new task and therefore can be transferred, particularly the lower layers (basic image feature extractors) will feature similar functionality in our relative pose estimation. As it is mostly the upper layers that require adaptation, fine-tuning reduces training time (and data requirements).

We substitute the last layer – originally outputting 1000 floats with the highest representing the correct class – by a new layer that outputs 6 numbers, ie. the 6 DOF relative pose components (illustrated by Fig. 5.5). By replacing this layer, learned weights and connections are

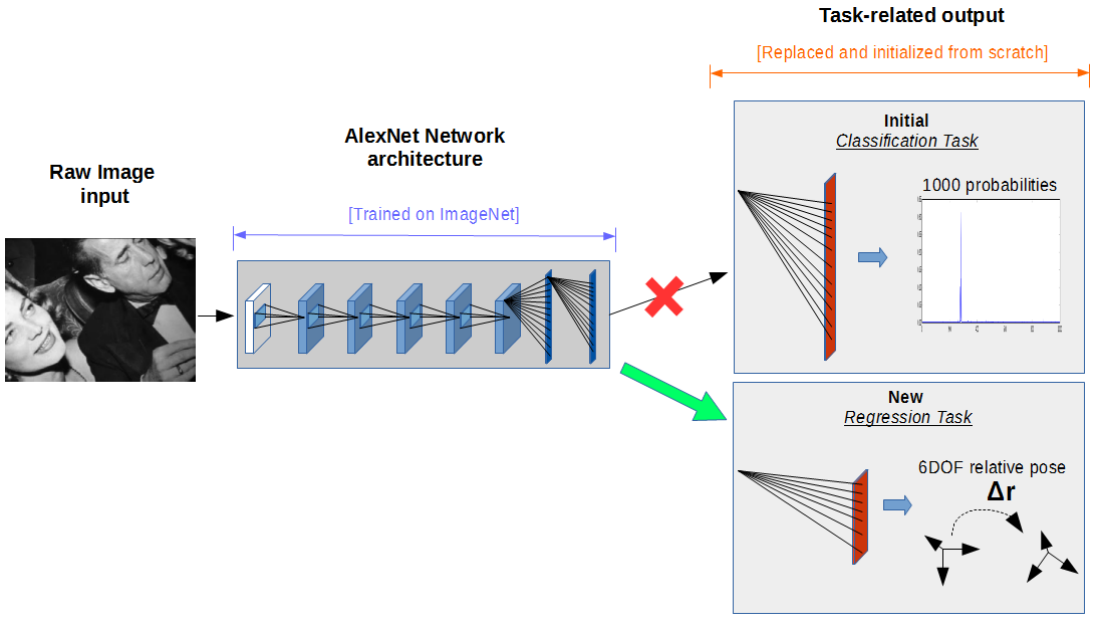


Figure 5.5: *Finetuning an AlexNet network trained for image classification*

discarded and the new links are initialized randomly. The resulting net is trained by being presented examples of the form (**I**, **r**), where **I** is a raw image, and **r** the corresponding relative pose as a training label. Since our training deals with distance between two pose vectors, we choose Euclidean cost function for network training, replacing the commonly used soft-max cost layer for classification, of the following form:

$$loss(\mathbf{I}) = ||\widehat{{}^{c_0}\mathbf{t}_c} - {}^{c_0}\mathbf{t}_c||_2 + \beta||\widehat{\theta\mathbf{u}} - \theta\mathbf{u}||_2 \tag{5.7}$$

where $\widehat{{}^{c_0}\mathbf{t}_c}$ and $\widehat{\theta\mathbf{u}}$ respectively are the estimation of the translation and rotation displacements relatively to the reference pose. $\beta = 0.01$ is a scale factor to harmonize the amplitude of the translation (in m) and rotation (in degrees) displacements for facilitating the learning process convergence. This type of distance was successfully used in previous works, such as [Kendall 15].

The proposed method can be used with any architecture of neural network trained with natural image inputs, therefore taking advantage of future developments in deep learned image classification. This property is validated later in the manuscript, as a larger CNN (the VGG network [Simonyan 15]) is used to overcome modeling limitations linked to the number of internal parameters of AlexNet.

Designing a Training Dataset

The design of the training dataset is the most critical step in the process of training a CNN, as it affects the ability of the training process to converge, as well as the precision and robustness of the end performances. Gathering real-world data is often cumbersome and sometimes unsuitable depending of the environment where the robot is expected to operate in. Furthermore, it can be difficult to re-create all possible conditions that can be encountered within a real-world environment. Thousands of training configurations should be considered, which requires large amounts of time when an actual robot has to be used. We choose to rely on simulated data in order to avoid those issues.

We now describe how such simulated data allow us to generate a virtually unlimited amount of training configurations. In addition we show how a variety of perturbations can be added, which leads to good results without lengthy real-world data acquisition.

Creating the nominal dataset

The nominal dataset is the basis of the training dataset. It contains all the necessary information for the CNN to learn how to regress from an image input to a 6DOF relative pose. We will be adding various perturbations later on to reduce the sensitivity of the network when confronted with real-world data.

In our design, the nominal dataset is generated from a single real-world image $\mathbf{I_0}$ of the scene. This is possible by relying on image transfer techniques, in order to create images as viewed from virtual cameras (the underlying process is the same as the one used in Chapter 4 for predicting the particles weights). Fig. 5.6 illustrates the image projected on a 3D plane and the varying (virtual) camera poses. On the implementation part, this operation is performed using image warping techniques, as described in the first chapter of this thesis.

Given a set of pose parameters, each component is generated through a Gaussian draw such as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.8}$$

With σ^2 the variance (determined empirically for each pose component in order to get a good sampling for the regression task), μ the expected value. As we are generating around the desired camera pose, the draw is centered around the zero value: so we take $\mu = 0$. Applying this random draw to each of the pose parameters gives us a relative pose associated to our virtual camera: $\mathbf{r} = (\mathbf{t}, \theta\mathbf{u})$.

We recall the theoretical steps to generate a new image from a homography, given a relative pose $\mathbf{r}$. A depth parameter d needs to be defined, this will represent the hypothetical distance between the camera and the scene plane at desired position.

From these two parameters, d and $\mathbf{r}$, the projection of an image $\mathbf{I}_1$ from the desired camera frame $\mathcal{F}_1$ to a projection image $\mathbf{I}_2$ in the virtual camera frame $\mathcal{F}_2$ can be defined as:

$$\mathbf{I}_2(\mathbf{x}) = \mathbf{I}_1\left(w(\mathbf{x}, \mathbf{h})\right) \tag{5.9}$$

where w is the image transfer function, $\mathbf{x}$ is any pixel in the considered image and $\mathbf{h}$ is a representation of the homography transformation (which uses the parameters d and $\mathbf{r}$).

By applying this procedure, it is possible to generate datasets of thousand of images quickly (less than half an hour for 11k images) eliminating the time-consuming step of gathering real-world data. In comparison, 700 robot hours were necessary to gather 50k data points for a single task in [Pinto 16].

The procedure to create the synthetic training dataset is then as follows (see also Fig. 5.7):

- acquire a single image $\mathbf{I_0}$ at pose ${}^{\mathbf{0}}\mathbf{r}$, in order to get the camera characteristics and scene appearance
- create the elements of the training dataset, consisting of tuples $({}^{i}\mathbf{r}, \mathbf{I}_i)$, through virtual camera views created by applying image warping techniques (as illustrated in Fig. 5.6). The first 10,000 virtual camera poses are obtained using a Gaussian draw around the reference pose ${}^{0}\mathbf{r}$, in order to have an appropriate sampling of the parameters space (the 6DOF pose). The scene in the simulator is set up so that the camera-plane depth at ${}^{0}\mathbf{r}$ is equivalent to 20cm, and the variances for the 6DOF Gaussian draw are such as (1cm, 1cm, 1cm, 10°, 10°, 20°), respectively for the $(\mathrm{t}_x, \mathrm{t}_y, \mathrm{t}_z, \mathrm{r}_x, \mathrm{r}_y, \mathrm{r}_z)$ DOF.
- the dataset is appended with 1,000 more elements. These are created by a second Gaussian draw with smaller variances (1/100 of the first draw). The finer sampling around ${}^{\mathbf{0}}\mathbf{r}$ enables the sub-millimeter precision at the end of the robot motion, assuming that ${}^{\mathbf{0}}\mathbf{r}$ is the desired pose.

Adding perturbations to the dataset images

In order to obtain a more robust process, two main perturbations were modeled and integrated in the dataset, namely, lighting changes (both global and local) and occlusions. We assume

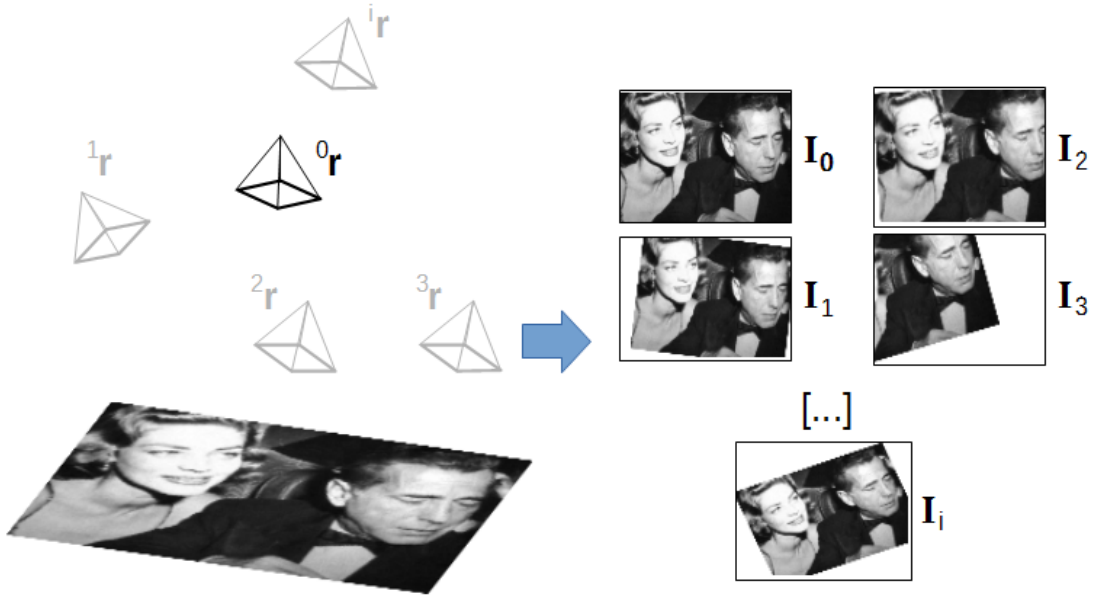


Figure 5.6: *3D plane with the projected reference image and multiple virtual cameras to generate multiple views of the scene.*

the scene to be known under nominal conditions for each experiment (ie. no deformations or temporal changes in the structure).

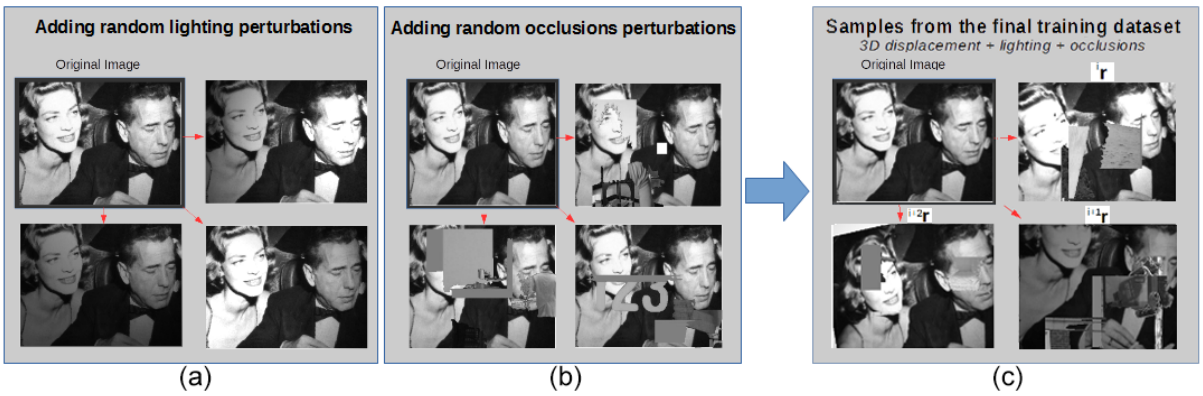


Figure 5.7: *Overall process to create a training set from the original input image: (a) examples after applying local illumination changes; (b) examples after adding super-pixels from random images as occlusions; (c) examples from the final dataset after applying all perturbations.*

5.3.3.2.1 Modeling illumination variations with 2D Gaussian functions Lighting conditions are a common problem when dealing with real-world images and are of global and local

nature. In order to model the global light, one can simply alter the pixel intensities by considering an affine variation of the intensities. Local lighting changes are more challenging and to obtain realistic synthetic images, time-consuming rendering algorithms are required. We alleviate this issue by working with planes in 3D space only, allowing to model lights as local 2D light sources and get realistic results. For each image chosen to be altered, the following mixture of 2D Gaussians is applied at each pixel (x, y):

$$\mathbf{I}_l(x, y) = \sum_{l=1}^{N_{lights}} \mathbf{I}(x, y) g_l(x, y) \tag{5.10}$$

Each 2D Gaussian in turn can be modeled as

$$g_l(x, y) = Ae^{-\left(\frac{(x-x_0)}{2\sigma_x^2} + \frac{(y-y_0)}{2\sigma_y^2}\right)} \tag{5.11}$$

where (x_0, y_0) (in pixel units) corresponds to the projection of the center of the simulated directional light source, gain A to its intensity, and (σ_x, σ_y) to the spread along each axis. An example of the resulting images can be seen in Fig. 5.7(a). We purposely let out the modeling of specularities, as the material and reflection properties are unknown. Our method will handle them as a sub-class of occlusions (see next section).

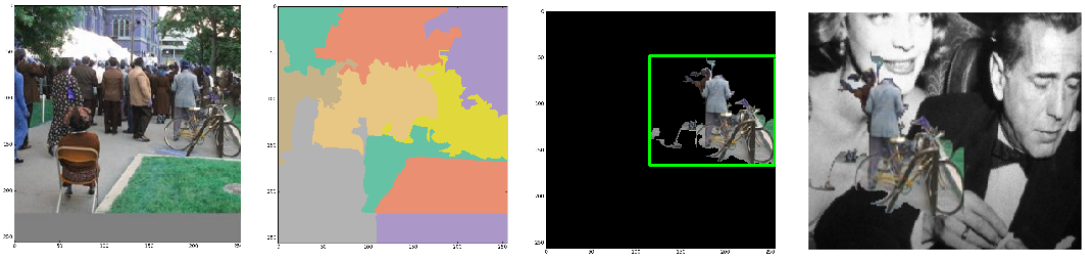


Figure 5.8: *Synthetic occlusion generation: on an arbitrary image in the Label-Me dataset (left) segmentation is performed. A segmented cluster is selected at random. It provides a coherent "occlusion' patch, which is merged with a dataset image and added to the dataset (last image).*

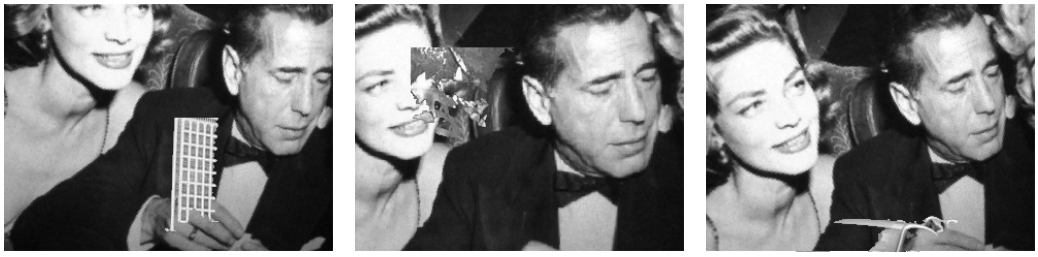


Figure 5.9: *Samples from the dataset with added occlusions*

5.3.3.2.2 Modeling occlusions by superimposing coherent pixel clusters from other image datasets Dealing with occlusions is challenging due to the potential variety in size, shape and appearance that can appear in a real environment. Additionally, when training a CNN, one has to be careful to create the training set with a variety of perturbations included. This is to prevent the network from over-fitting on the presented examples and thus being unable to handle real world occlusions never seen at training time. We present here a procedure to provide the network with a realistic set of occlusions with an adequate range in size, shape and appearance.

Clusters of pixels – representing a coherent part of an image – from other datasets are superimposed on the previously generated images. To create somewhat realistic conditions real world images were preferred over synthetic occlusion images. These images provide a variety of scenes that represent interesting and varied occlusions, rather than those generated from geometrical or statistical methods. Herein the *Label-Me* dataset [Russell 08] containing thousands of outdoor scene images was chosen. The scenes contain a variety of objects in a wide range of lighting conditions. We then applied the following workflow (illustrated in Fig. 5.8) to each of the images in our simulated dataset that we want to alter:

- select randomly one image from the *Label-Me* dataset;
- perform a rough segmentation of the image by applying the SLIC super-pixel [Achanta 12] segmentation algorithm creating coherent pixel groups (implementation available in OpenCV)
- select a random cluster from the previous step, and then insert this cluster into the image to alter at a random position.

This method allows us to get a training dataset with randomly varied occlusions such as illustrated in Fig. 5.7(b). Additional images with added occlusions can be seen on Fig. 5.9, to illustrate the diversity of shape, size and texture that arises from the proposed method. By stacking the two described perturbations on our initial nominal dataset, we are able to generate a final training dataset with all the desired characteristics, as shown in Fig. 5.7(c).

Training the network

Starting from the trained AlexNet available for use with the Caffe library [Jia 14], the network was then fine-tuned. For this a new scene specific dataset of 11000 images with a variety of perturbations is created (as described above). Using Caffe the network was trained with a 'batch size' of 50 images over 50 training 'epochs'. These two terms are explained below.

The notion of batch stems initially from the need to reduce the computational cost of training a CNN, and proved to be a way to increase the stability of the system convergence. Instead of taking every element of the training dataset, compute a forward pass to get the prediction error, then compute through backpropagation the weight adjustments, the following method is used. For a given number of elements, denoted the 'batch size', only the forward pass is performed, and the prediction errors are accumulated. Once the whole batch has been processed, the average prediction error is used to perform the backpropagation. The result is that for this subset of the training dataset, only one backpropagation is performed, and the direction of the gradient is more representative, as multiple data points have been used to compute it, smoothing the training process. The only drawback of this method is that, as the batch size increase in size, more memory in the GPU is used, in order to accumulate the previous forward pass results.

Instead of using as a time metric the number of training iterations (one iteration corresponding to one update of the network's weights), which does not take into account how much of the dataset has been processed due to the batch training strategy, the notion of 'epoch' has been introduced. One epoch represent the number of iterations, for a given batch size, for which the entirety of the dataset has been passed through the network.

Fig. 5.10 displays the evolution of the loss value as the network learns from the dataset. At the end of the 50 epochs of training, the loss value has plateaued. The decision to stop the training at this number of iterations is empirical. For object classification, a second 'validation' dataset is used to compute a separate loss graph. It is used to detect over-fitting on the training dataset, which leads the training loss to be low, but the actual performances of the network to be worse, as the network 'over learned' on the training examples. On our case, regression problems are less prone to over-fitting, hence, simply assessing the moment the loss function starts to plateau is enough to take the decision of stopping the training.

Going further: Toward scene-agnostic CNN-based Visual Servoing

The method presented on the previous section is very powerful in cases where the scene the robot needs to operates on is known, as the network is trained on a given scene. From this hypothesis, using a relatively short offline training time, the method is able to position the system with very fine precision (as seen later in the experimental part).

However, if the scene the robot is expected to operates on is unknown, this approach becomes inapplicable. To address this issue, a variation of this method is proposed in order to train a CNN able to compute a relative pose estimation between two images taken from two viewpoints of an arbitrary scene (in this sense, it is similar to estimating an homography). Such

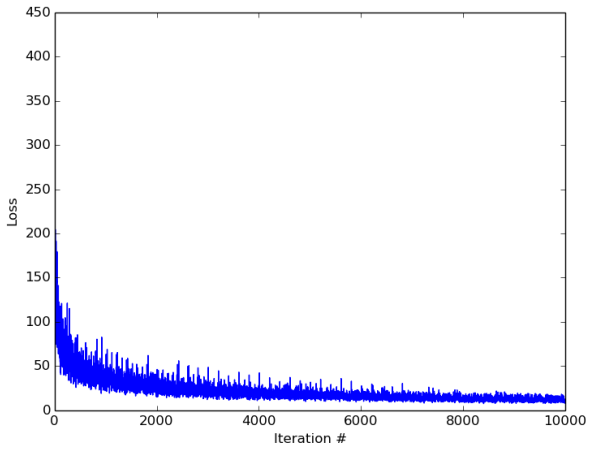


Figure 5.10: *Evolution of the loss function through training iteration*

a method allows for increased flexibility of application.

Building on the first method, this new 'scene-agnostic' network is trained on a dataset generated from simulation tools, using an external dataset for providing scene diversity.

Task definition

To operate within scenes never seen before, we use a similar network architecture as in the previous case, but train it for a new task that consists in computing the relative pose corresponding to two viewpoints of the same scene. This is made possible by training the network to associate a couple of correlated images of the same scene to a given camera displacement.Although the scale of the motion cannot be estimated without depth measurements, the direction of the displacement vector will be accurate and usable within a closed-loop control scheme.

The cost function for the network becomes:

$$loss(\mathbf{I}, \mathbf{I}^*) = ||\widehat{{}^{c_0}\mathbf{t}_c} - {}^{c_0}\mathbf{t}_c||_2 + \beta||\widehat{\theta\mathbf{u}} - \theta\mathbf{u}||_2 \tag{5.12}$$

The important addition from Eq (5.7) is that the optimization process now depends on two image inputs. The only correlation between these two images is that they both need to be a viewpoint of the same known scene.

Network architecture

In order to be able to fine-tune an existing network, one has to keep the shape of the inputs identical in order to maintain the relationship between the first layers. To accommodate this constraint, we make the choice of working on gray-scale image only. Indeed, as the network is originally built to accept as inputs an RGB image, composed of 3 image planes, we cannot input 2 distinct RGB images without changing the topology of the network. Instead, we choose to use the following workaround: each of our input images (current and desired) is recorded as a gray-scale image, and an artificial RGB image is the assembled. An empty (all pixels as black) RGB image is created, then the 'Red' image plane is replaced by the desired image, and the 'Green' image plane is replaced by the current image. The last 'Blue' image plane remains as black. This artificial RGB image then contains all the information necessary to compute a relative pose estimation between two points of view. Although this input is the same shape as the expected network input, interpreting correctly this new data organization is expected to require additional training time.

On top of this additional complexity to interpret the new input, the dataset complexity is higher than for the known-scene one, due to the increased variability. During preliminary experiments, we noticed that fine-tuning the AlexNet network did not lead to satisfactory performances. By varying the complexity of the task (the number of DOF involved in the relative pose estimation), experiments showed that the network was unable to maintain good performances with 6DOF, although giving adequate results for up to 4DOF. The conclusion of this preliminary experiment was that the complexity of the task to solve was higher than the complexity that could be modeled within the AlexNet architecture. Hence, a different, deeper CNN architecture was chosen instead: the VGG-16 network presented in [Simonyan 15]. The overall organization of this network is the same as the AlexNet: several convolutional layers, followed by fully connected layers. The main difference is the number of layers used.

If we abbreviate the layer names such as: convolution layer C, pooling layer P, fully connected layer FC, then the AlexNet would be written $5 \times (C-P) - 3 \times (FC)$, with a number of internal weight parameters of 60M. The VGG-16 network, on the other hand, would be written $2 \times (C-C-P) - 3 \times (C-C-C-P) - 3 \times (FC)$, with 138M internal parameters.

These additional weights parameters make possible the representation of more complex models, such as the task we want to tackle.

Training dataset generation

Using the same set of tools as before, we propose a method to generate a dataset able to solve this new task from simulation tools and readily available datasets. This dataset will be used to fine tune a trained VGG network [Simonyan 15].

Each element of the training set is generated as follow:

- select a random image from a set of natural images, here from the Label-Me classification dataset used previously and set it as texture of the 3D plane in the simulated scene;
- record two distinct viewpoints (chosen randomly) of this scene (network input);
- record the associated relative pose between these two viewpoints (data point label).

By repeating this process, an arbitrary number of annotated couple of viewpoints can be generated, creating an appropriate dataset for this new task. Perturbations can also be added in the same fashion as in the previous method.

Since this new task necessitates a more important re-configuring of the trained VGG network (going from a single image input to a dual image input), we generated a bigger dataset compared to the previous training, going from 10k images to 100k images for training, which ran for 50 epochs.

Experimental Results

In this section, we solve a series of positioning tasks using the two networks presented above, the scene-specific and scene-agnostic networks. These positioning tasks all follow the following plan: a robot arm with a camera attached on the end-effector is positioned so that the camera records the scene according to a given viewpoint (this is the desired pose), then the robot arm is moved at an arbitrary starting position. From this initial pose, the control laws must control autonomously the robotic system toward the desired pose, by making the recording of the camera match as closely as possible the image at desired pose. This task and the associated setup is illustrated in Fig. 5.11. In the nominal cases, the considered scene is planar, with a constant lighting and no occurring occlusions.

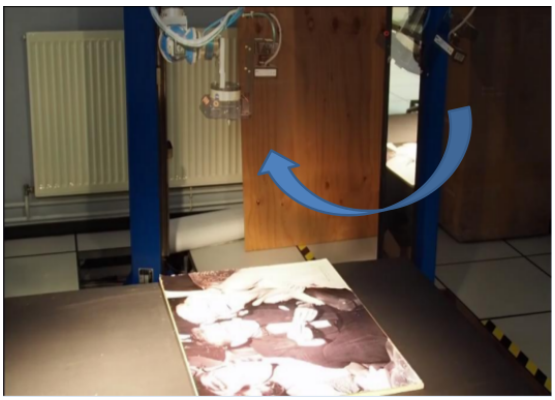


Figure 5.11: *External view from an experimental run: a VS control law drives the system from an initial position toward a desired position*

Scene specific CNN: Experimental Results on a 6DOF Robot

This section describes a set of experiments performed on an Afma 6DOF gantry robot in a typical eye-in-hand configuration. At the beginning of each experiment, the robot is moved to an arbitrary starting pose $\mathbf{r_0}$ and the task is to control the robot toward a position defined by a desired image.

The following experiments can be seen on the following video link:
`https://youtu.be/i_1NZCFdbco`.

Planar scene

In this first set of experiments, the scene that is seen from the hand-mounted camera is planar, allowing for a reduced impact of the planar scene approximation used to train our network. No 3D object within the scene means that no unexpected occlusions will appear during motion, hence, the real scene will be similar to the training dataset.

5.5.1.1.1 Nominal Conditions In terms of pose offset, the robot has to perform a displacement given by $\Delta^{\mathbf{0}}\mathbf{r} = (1\text{cm}, -24\text{cm}, -9\text{cm} - 10°, -16°, -43°)$. with a distance between the camera and the planar scene of 80cm at the desired pose $\mathbf{r}^*$. The training of the network with the 11000 images is performed offline. The overall experiment is shown in Fig. 5.12. Fig. 5.12(a)

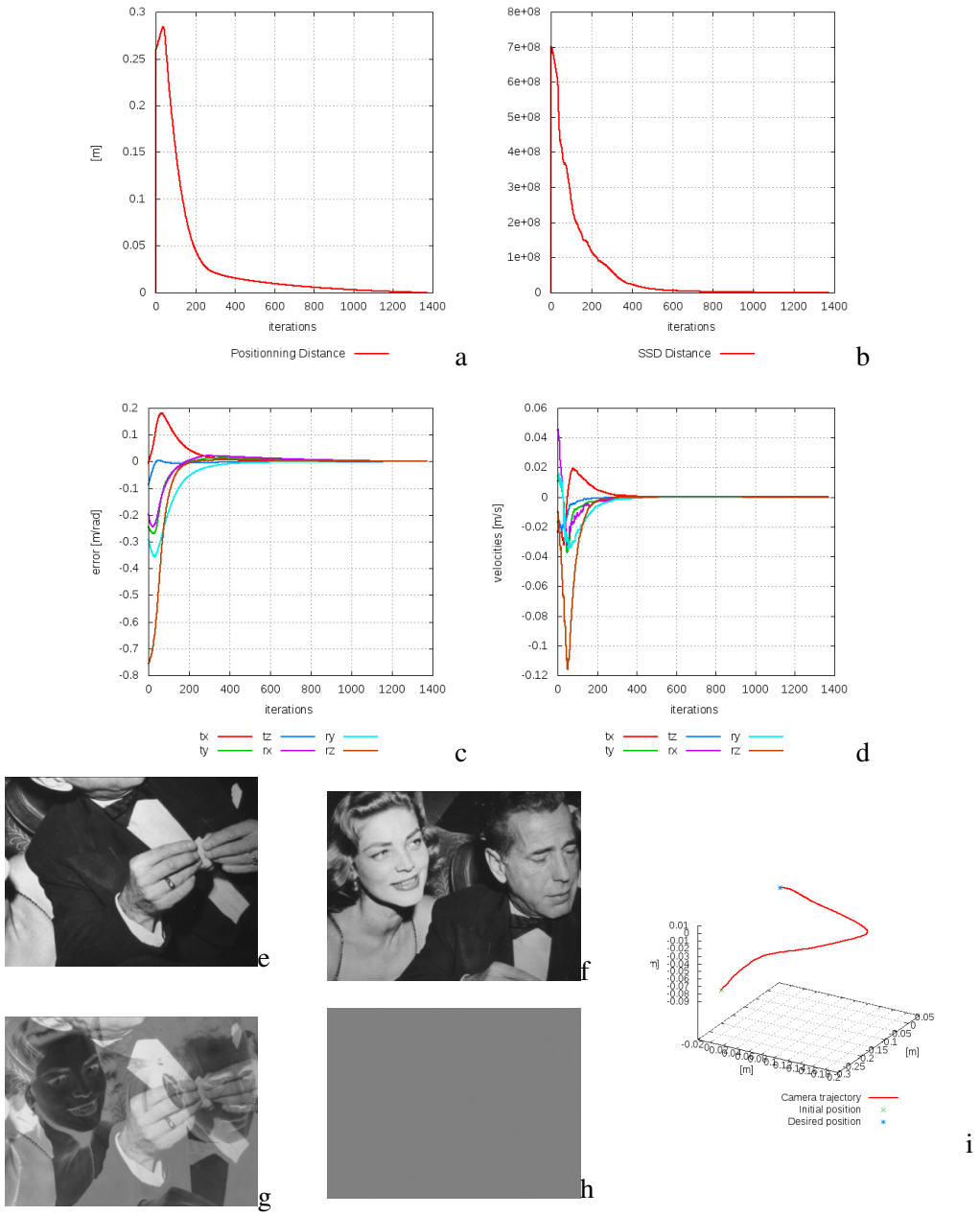


Figure 5.12: *CNN-based visual servoing on a planar scene; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose $\mathbf{I_0}$; (f) Image at final pose $\mathbf{I}(\widehat{\mathbf{r}})$; (g) Image error $\mathbf{I_0} - \mathbf{I}^*$ at initial pose; (h) Image error $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ at the final pose; (i) 3D trajectory.*

to Fig. 5.12(d) and Fig. 5.12(i) show that our CNN-based direct visual servoing approach converges without any noisy nor oscillatory behaviors when performed in a real-world robotic setting. One point that can be noted is that since no constraints are set onto the overall motion throughout the experiment, the trajectory may not be optimal at any point of the motion. This can be seen especially at the beginning of the run: although the dissimilarity between the current and desired images decreases steadily, the positioning error displays a temporary increase as an unnecessary motion along the x-axis is performed. The position of the system at the end of the motion is less than one millimeter from the desired one. No particular efforts were made to have "perfect' lighting conditions, but also no external lighting variations or occlusions were added. These were introduced in the next experiment.

5.5.1.1.2 Dealing with perturbations: Light changes and occlusions Given the same initial conditions as above additional light sources and external occlusions were added to test the robustness of our approach. The overall experiment is detailed in Fig. 5.14. The robot captures a single image at the initial pose, the network is trained again and then our CNN-based direct visual servoing is performed. While the robot is servoing the light coming from 3 lamps is changed independently, resulting in global and local light changes. In addition during the experiment various objects are added, moved and removed from the scene in order to create occlusions. Samples images showing some of these perturbations are depicted in Fig. 5.13.

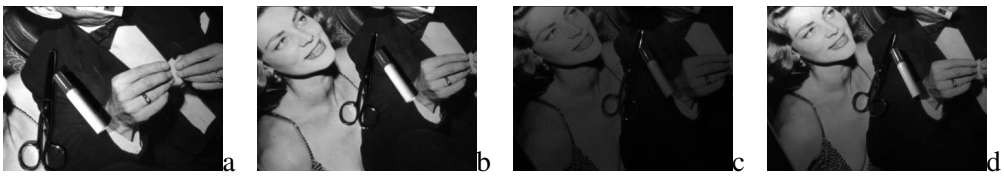


Figure 5.13: *Images collected during real-world experiments on our 6DOF robot. Significant occlusions and variation in the lighting conditions can be seen.*

We can see that despite the variety and severity of the applied perturbations, the control scheme does not diverge, even if the trajectory followed by the camera motion is different than in the nominal conditions. The method instead exhibits a decrease in final precision, ranging from 10cm (in accumulated translation errors) at the worst of the perturbations to less than one millimeter error when back in the nominal conditions. A slower convergence is also observed when compared with the nominal conditions experiment. Additionally, most of the positioning error lies in the coupled translation and rotation degrees-of-freedom tx/ry and ty/rx. This keeps most of the scene in the camera's field of view by keeping the center of the scene aligned with the optical axis. However, as soon as this perturbation vanishes, the method is able to retrieve instantaneously its converging motion. It is important to note that since no tracking is involved, no additional control scheme were introduced to deal with sudden loss of information and re-initialization of the method as it is able to regain its performances as soon as the information

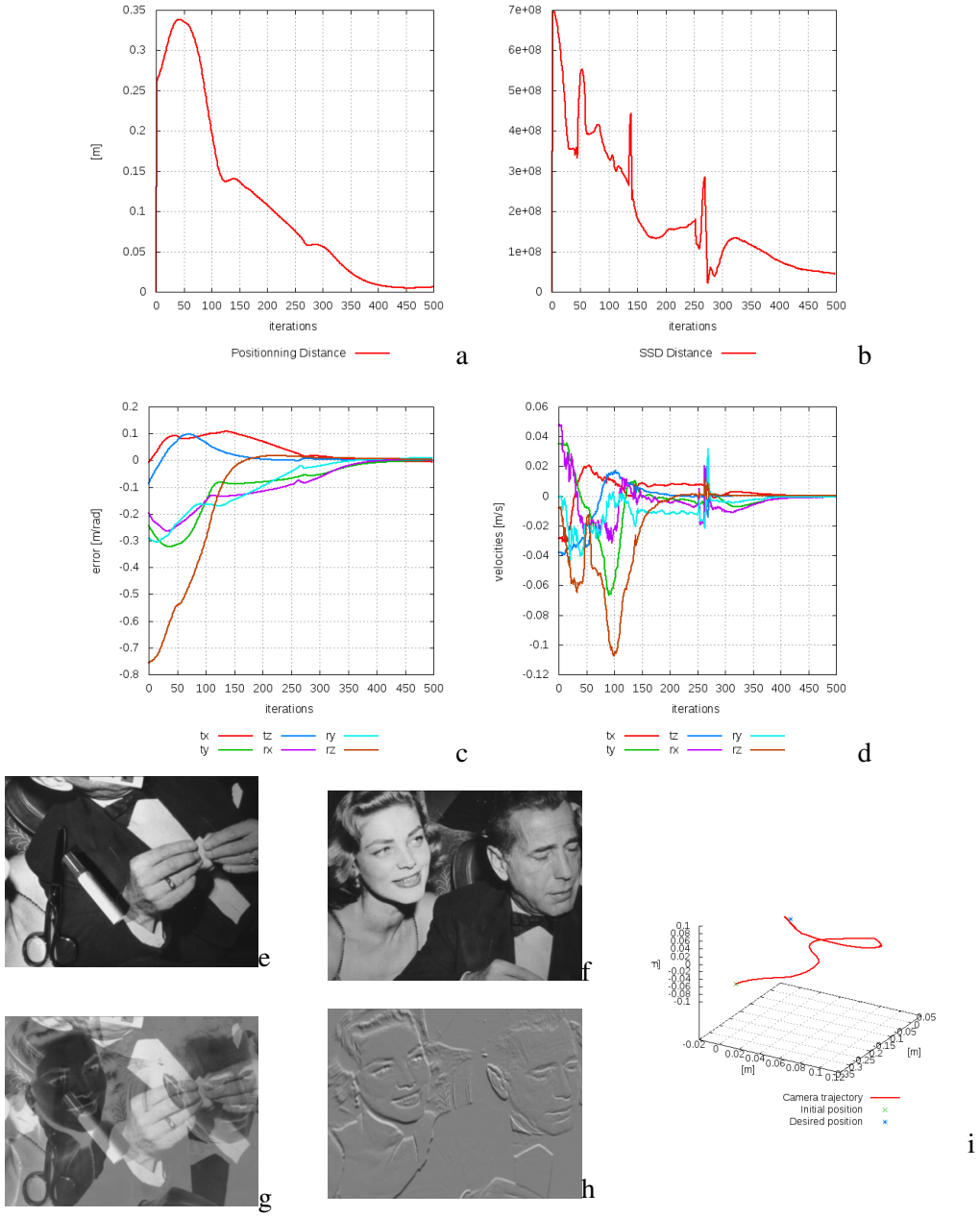
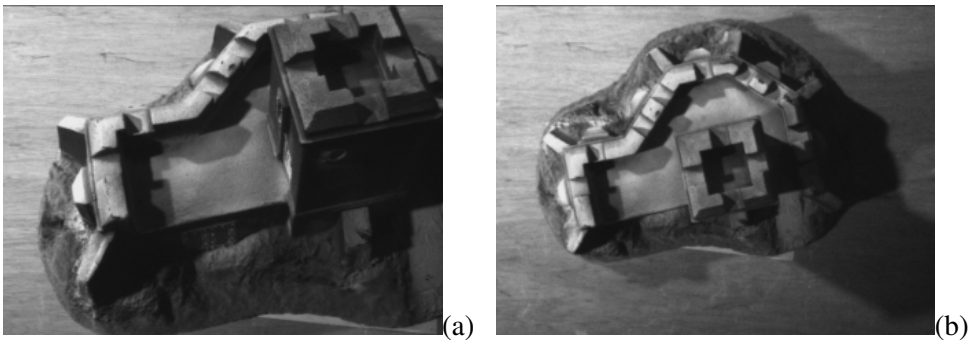


Figure 5.14: *CNN-based visual servoing on a planar scene with various perturbations; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose $\mathbf{I_0}$; (f) Image at final pose $\mathbf{I}(\widehat{\mathbf{r}})$; (g) Image error $\mathbf{I_0} - \mathbf{I}^*$ at initial pose; (h) Image error $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ at the final pose; (i) 3D trajectory.*

becomes available again. It also can be seen that the perturbations observed on the outputs of the network (Fig. 5.14(c)), which are used as inputs of the control scheme, are not synchronous and are less noisy than the perturbations observed in the sum-of-squared-distances (SSD) plot (Fig. 5.14(b)), proving the validity of the internal representation learned by the network.

Dealing with a 3D scene

In this set of experiments, the scene consists of the same 3D plane, with a castle model positioned on top of it. The castle model can be seen on Fig. 5.15. Since for this new scene, the same workflow described as before is applied to train the network: taking a single image of the scene, and generating the training dataset with homographies, this means that the 3D information associated to the scene will be lost. The network will then be trained as if the scene consisted of an image of a castle on a 3D plane, which will create a strong discrepancy at times during the motion, where the camera will feed into the network images of the castle from an unknown angle, revealing parts of the 3D model that were never included in the dataset.



(a) (b)

Figure 5.15: *Model of castle, captured from two viewpoints*

For reference, the following experiments will use a network trained with a dataset generated from the image displayed on Fig. 5.15(b). Comparing it with Fig. 5.15(a) highlights the amount of model parts that will be considered as unknown by the network during motion.

5.5.1.2.1 Nominal conditions. As in the previous set of experiments, the robot has to perform a displacement given by $\Delta^{\mathbf{0}}\mathbf{r} = (1\text{cm}, -24\text{cm}, -9\text{cm}, -10°, -16°, -43°)$. with a distance between the camera and the plane on which the castle rests of 80cm at the desired pose $\mathbf{r}^*$. The training of the network with the 11000 images was performed offline.

This experiment run is described in Fig. 5.16, in the same layout as in the previous section.

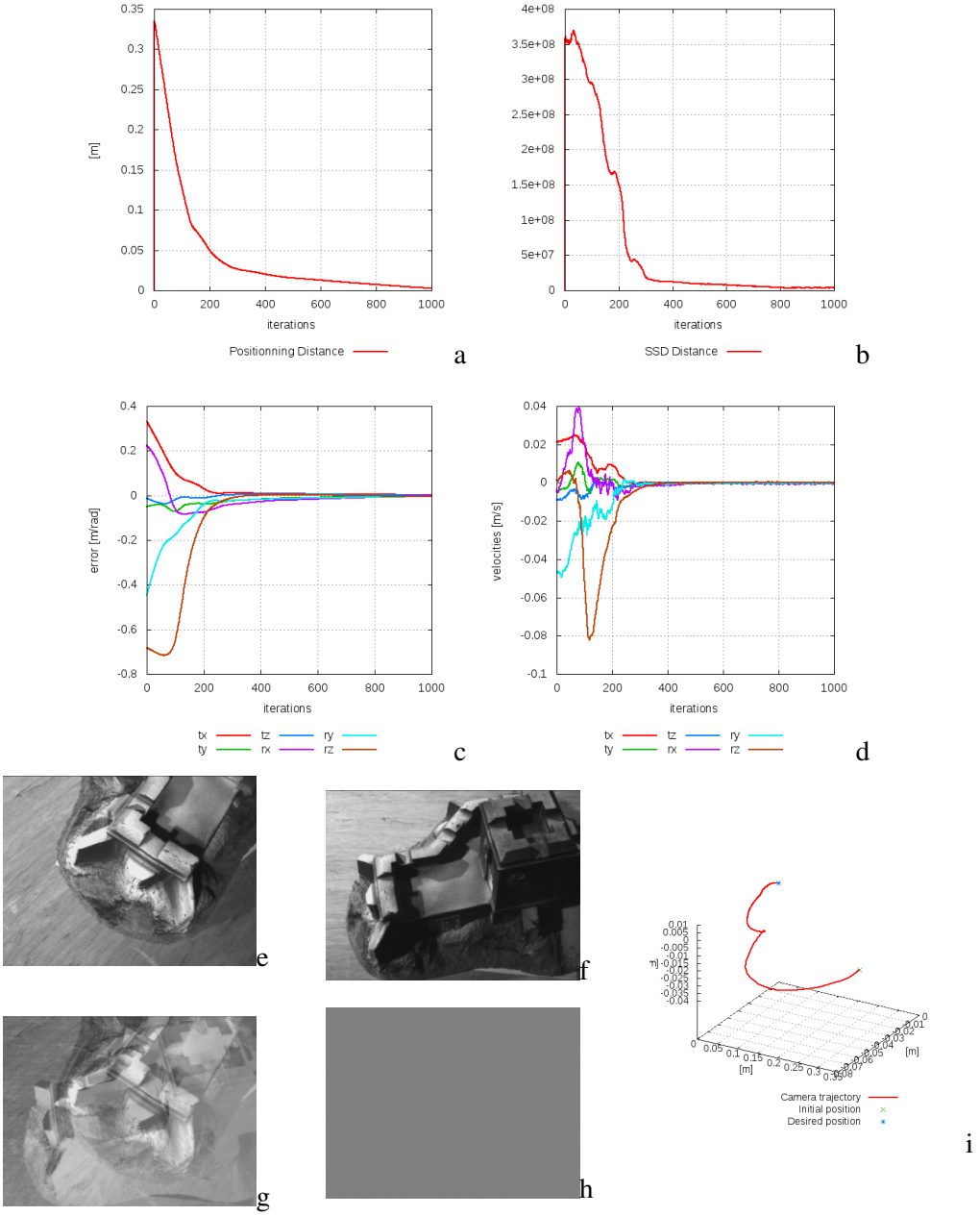


Figure 5.16: *CNN-based visual servoing on a 3D scene; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose* $\mathbf{I_0}$*; (f) Image at final pose* $\mathbf{I}(\widehat{\mathbf{r}})$*; (g) Image error* $\mathbf{I_0} - \mathbf{I}^*$ *at initial pose; (h) Image error* $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ *at the final pose; (i) 3D trajectory.*

We can see that despite the severe discrepancies between the experimental setup and the training set, the control law succeeds in converging successfully toward the desired pose. We can note that even in presence of errors introduced by the unknown 3D elements, but it still manages to decrease at any time and keep exhibiting an exponential decrease of the pose error. As the camera pose gets closer and closer to the target pose, the discrepancy between the current view and the training dataset fades away, allowing the system to converge with a sub-millimeter precision. This highlights the robustness of the control law, and proves that the planar scene hypothesis can be relaxed, given that the learning process is submitted to enough perturbations.

It is interesting to note that at some point on the trajectory, as formerly occluded parts of the castle model enter the field of view of the camera, the network adapts and re-evaluates drastically the relative pose estimate, creating a sharp angle in the trajectory path of the camera on Fig. 5.16(i).

5.5.1.2.2 Perturbed conditions: occlusions. In this experiment we reproduced the previous experiment and added during the course of the experiment an important occluding object on top of the model during motion (see Fig. 5.17). The overall experiment can be visualized in Fig. 5.18.

We can see that despite the severity of the perturbation and unknown 3D parts of the castle, the control is still able to converge properly toward the target pose. Although the pose error is not decreasing anymore in an exponential decrease fashion, it still manages to decrease at any time. It is interesting to point that the SSD cost function (visualized as a reference to compare to the classical direct visual servoing method) during the run is clearly non-convex, exhibiting significant noises although the positioning error decreases steadily. This confirms that the neural net did succeeded in generating its own image descriptor that is more informative than the raw image data.

Final precision is sub-millimetric once the occlusion perturbation has disappeared.

Experimental simulations and comparisons with other direct visual servoing methods

In this section we compare the proposed method with the original direct visual servoing scheme [Collewet 11] and with the direct visual servoing based on a particle filter (PF) control law presented in the previous chapter.

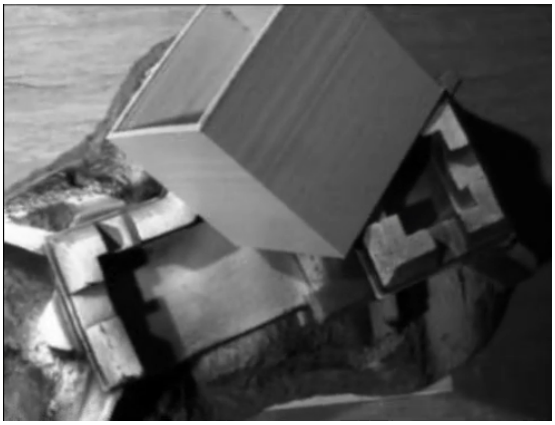


Figure 5.17: *Sample from the experiment*

5.5.1.3.1 Convergence area Each method was evaluated by running a positioning task over 10 batches, each with 80 individual runs. The starting pose for each run was randomly offset (with a Gaussian distribution) from the desired pose $\mathbf{r}^*$. The magnitude of the offset was increased linearly from batch 0 to batch 9. This scene was simulated as illustrated in Fig. 5.6, with the desired image $\mathbf{I}_0$ and the desired pose $\mathbf{r}_0$. The mean depth is 25cm and the offset pose was generated using the following variances (from batch 0 to batch 9): from 0 to 4cm for the X and Y translations, from 0 to 2cm for the depth, from 0 to 20° for rotations around the X and Y axes, from 0 to 50° for rotations around the Z axis.

The results of this benchmark are shown in Fig. 5.19. It can be seen that the proposed approach has a larger convergence range with a significantly higher proportion of runs converging to the desired pose than with the other schemes for all starting poses with non-zero offsets.

5.5.1.3.2 Camera trajectory A second property that can be compared between our set of methods is the shape of the trajectory followed by the camera for a given task. In this experiment, we perform a positioning task with an initial offset of
$\Delta^{\mathbf{0}}\mathbf{r} = (8\text{cm}, -21\text{cm}, 13\text{cm}, -13°, -11°, -2°)$. The task is solved with the classical direct VS method (Fig. 5.20(a)), the PF-based method (Fig. 5.20(b)), and the CNN-based method (Fig. 5.20(c)).

While none of the trajectories are close to the ideal straight line, a progression can be seen, as the first method generates a trajectory with no regularity and several sudden orientation changes. The second method associated trajectory is less irregular on the overall motion, but a small oscillatory behaviour is observed along the whole motion. The last method on the other hand displays a smoother trajectory indicating a more steady and accurate control law.

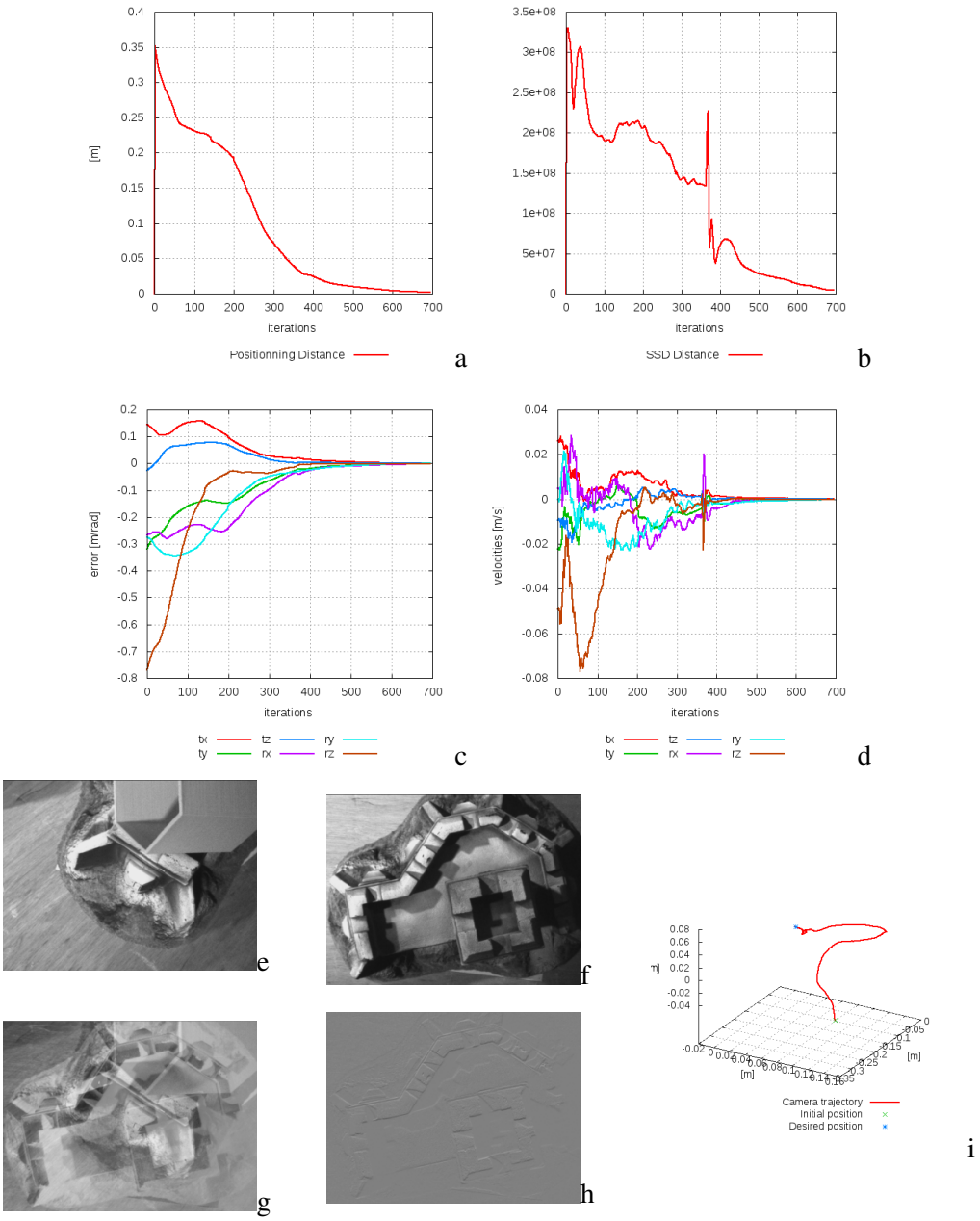


Figure 5.18: *CNN-based visual servoing on a 3D scene; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose* $\mathbf{I_0}$*; (f) Image at final pose* $\mathbf{I}(\widehat{\mathbf{r}})$*; (g) Image error* $\mathbf{I_0} - \mathbf{I}^*$ *at initial pose; (h) Image error* $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ *at the final pose; (i) 3D trajectory.*

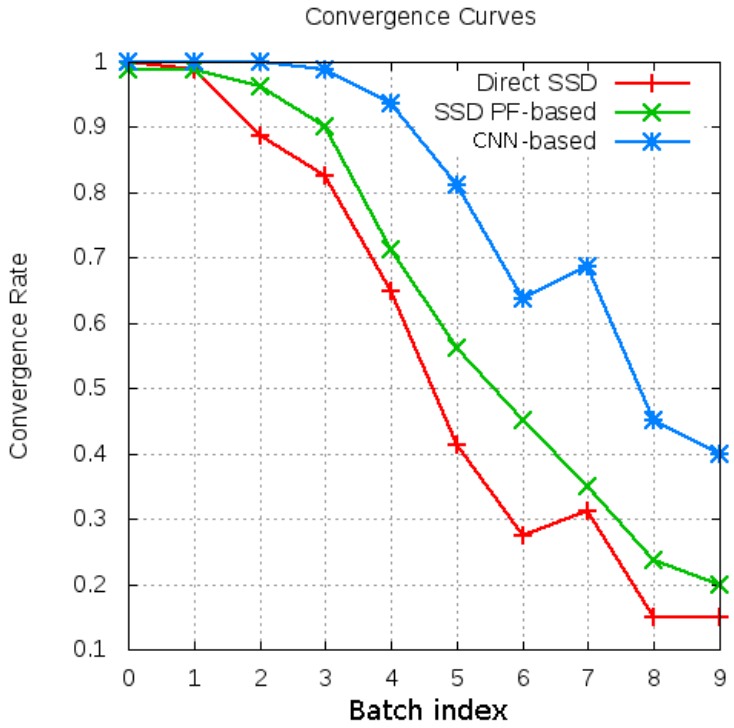


Figure 5.19: *The effects of the initial positioning offset (small offset for batch 0, large one with batch 9) with three DVS methods: DVS using SSD cost function (in red with +), a version using particle filter (green x) and our proposed CNN-based VS (blue *).*

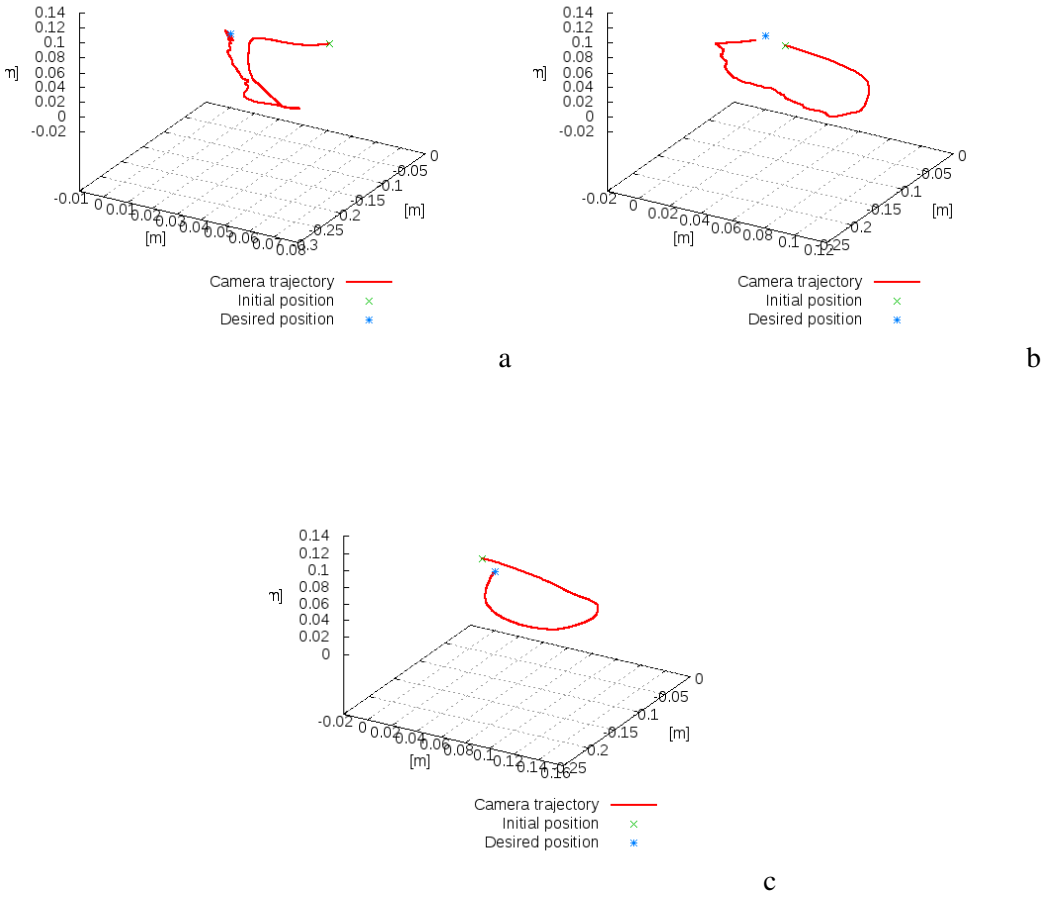


Figure 5.20: *Trajectories followed by the camera while solving an identical task. (a) Classical direct VS. (b) PF-based VS. (c) CNN-based VS.*

Scene agnostic CNN: Experimental Results on a 6DOF Robot

In order to validate the extension toward a scene-agnostic method presented in Section 5.3, we performed the same experiment, this time on a scene the network *has never seen* at training step. These scenes exhibit a large panel of challenging image characteristics, from low textures wooden areas, soft shadows and reflective surfaces on the electronic circuits (Fig. 5.21), highly contrasted with high frequency gradients (Fig. 5.22) to 3D elements Fig. 5.23.

Various initial and desired poses have been tested. The final distance from camera to scene is on average around 50 cm. For each of the runs, an hybrid control is used: the CNN-based control law position the robot close to the desired position, and then a classical direct VS control law performs the final approach phase to ensure a sub-millimetric positioning accuracy.

The first experiment (Fig. 5.21) has an initial positioning offset of
$\Delta^{\mathbf{0}}\mathbf{r} = (33\text{cm}, -5\text{cm}, -1\text{cm}, 13°, -26°, -38°)$. The second experiment (Fig. 5.22) of
$\Delta^{\mathbf{0}}\mathbf{r} = (-2\text{cm}, -30\text{cm}, 30\text{cm}, 28°, -4°, -18°)$. Lastly, the third experiment (Fig. 5.23) has an initial positioning offset of
$\Delta^{\mathbf{0}}\mathbf{r} = (-10\text{cm}, -16\text{cm}, -9\text{cm}, -8°, -1°, -41°)$.

We can see that for each scene the proposed method succeeds to position the robot within a few centimeters of the target pose, performing a large displacement, in both the image space and effector space. From this close pose, we switched to the classical DVS to achieve sub-millimetric accuracy. Using jointly these two methods allows to benefit from the best of each method: a broad convergence radius from the CNN-based method, and a very precise positioning around the optimum with the SSD-based method.

We can also see that the trajectories followed by the camera throughout all three experiments are not straight lines, indicating that the pose estimation provided by the neural network is not perfect. On the other hand, these trajectories are smooth, although the network performs a single - uncorrelated from previous inputs - relative pose estimation at each frame: this highlights the success of the training scheme as the regression from images to 6DOF pose keeps consistent from one image to another, and a small input change induce only a small change in the relative pose estimate. This also opens interesting perspectives as whether it can be possible to improve the trajectories by taking into account the previously estimated values to increase the accuracy of the pose estimate at time k.

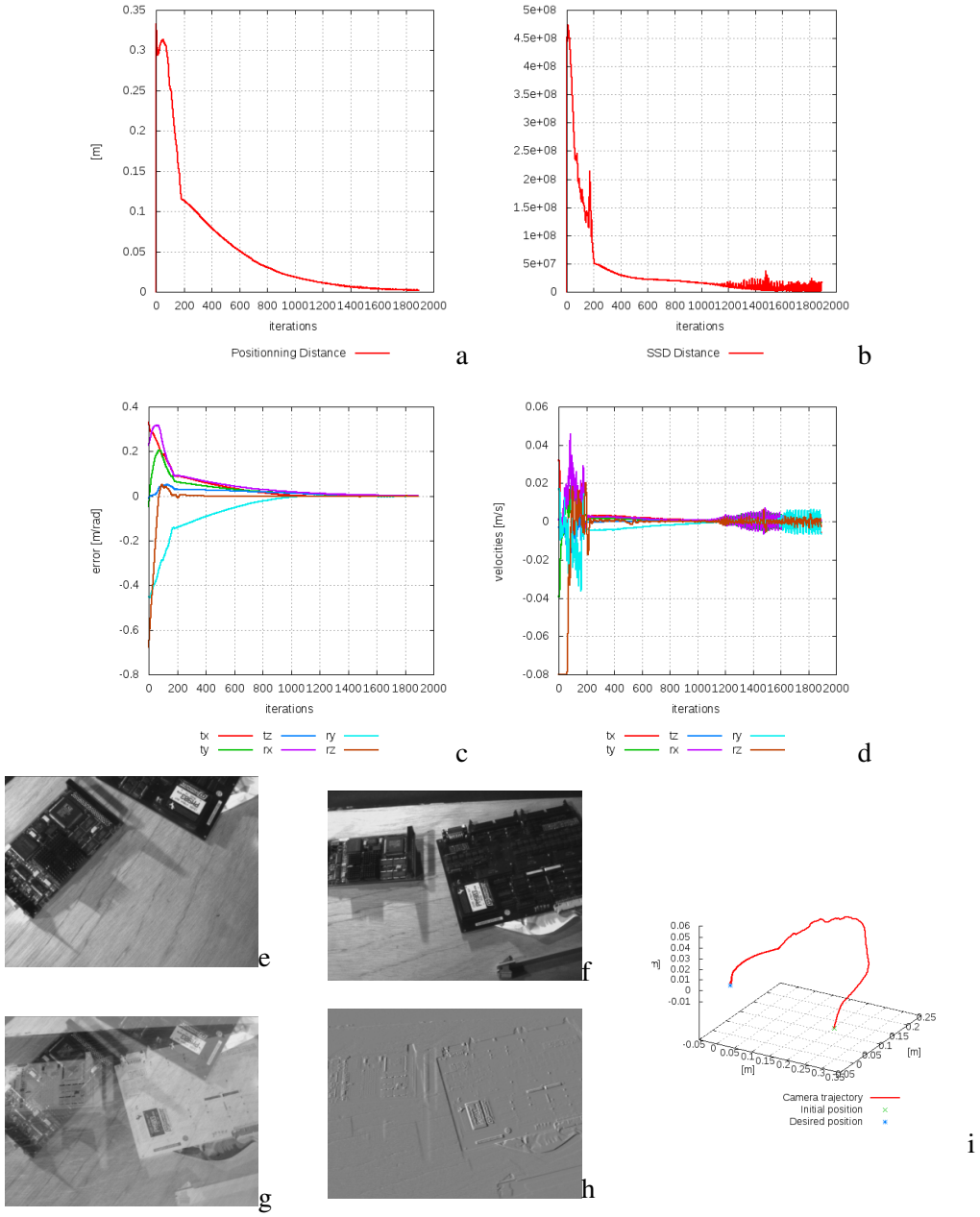


Figure 5.21: *Scene-agnostic CNN-based visual servoing on a weakly 3D scene scene; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose $\mathbf{I_0}$; (f) Image at final pose $\mathbf{I}(\widehat{\mathbf{r}})$; (g) Image error $\mathbf{I_0} - \mathbf{I}^*$ at initial pose; (h) Image error $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ at the final pose; (i) 3D trajectory.*

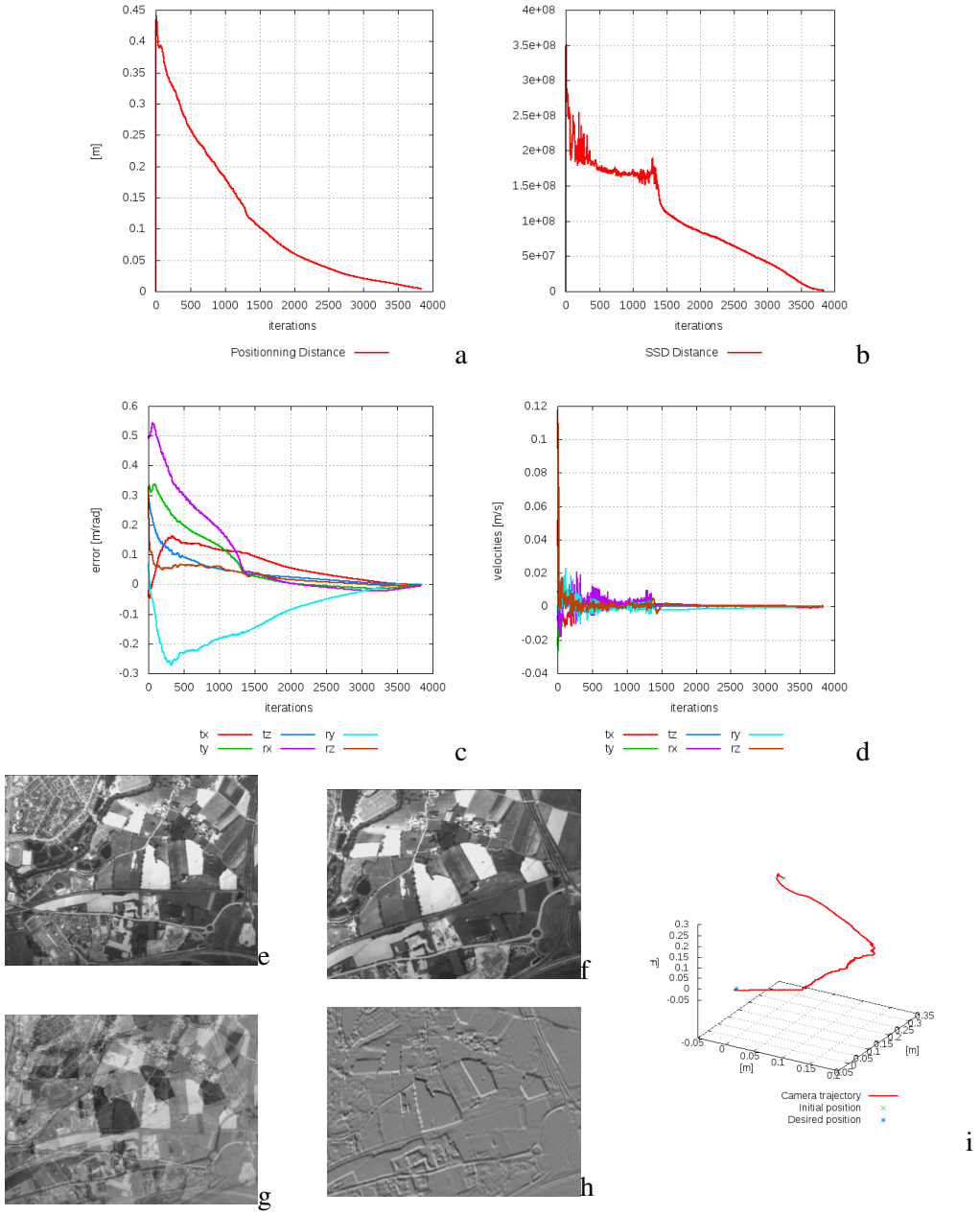


Figure 5.22: *Scene-agnostic CNN-based visual servoing on a planar scene; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose $\mathbf{I_0}$; (f) Image at final pose $\mathbf{I}(\widehat{\mathbf{r}})$; (g) Image error $\mathbf{I_0} - \mathbf{I}^*$ at initial pose; (h) Image error $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ at the final pose; (i) 3D trajectory.*

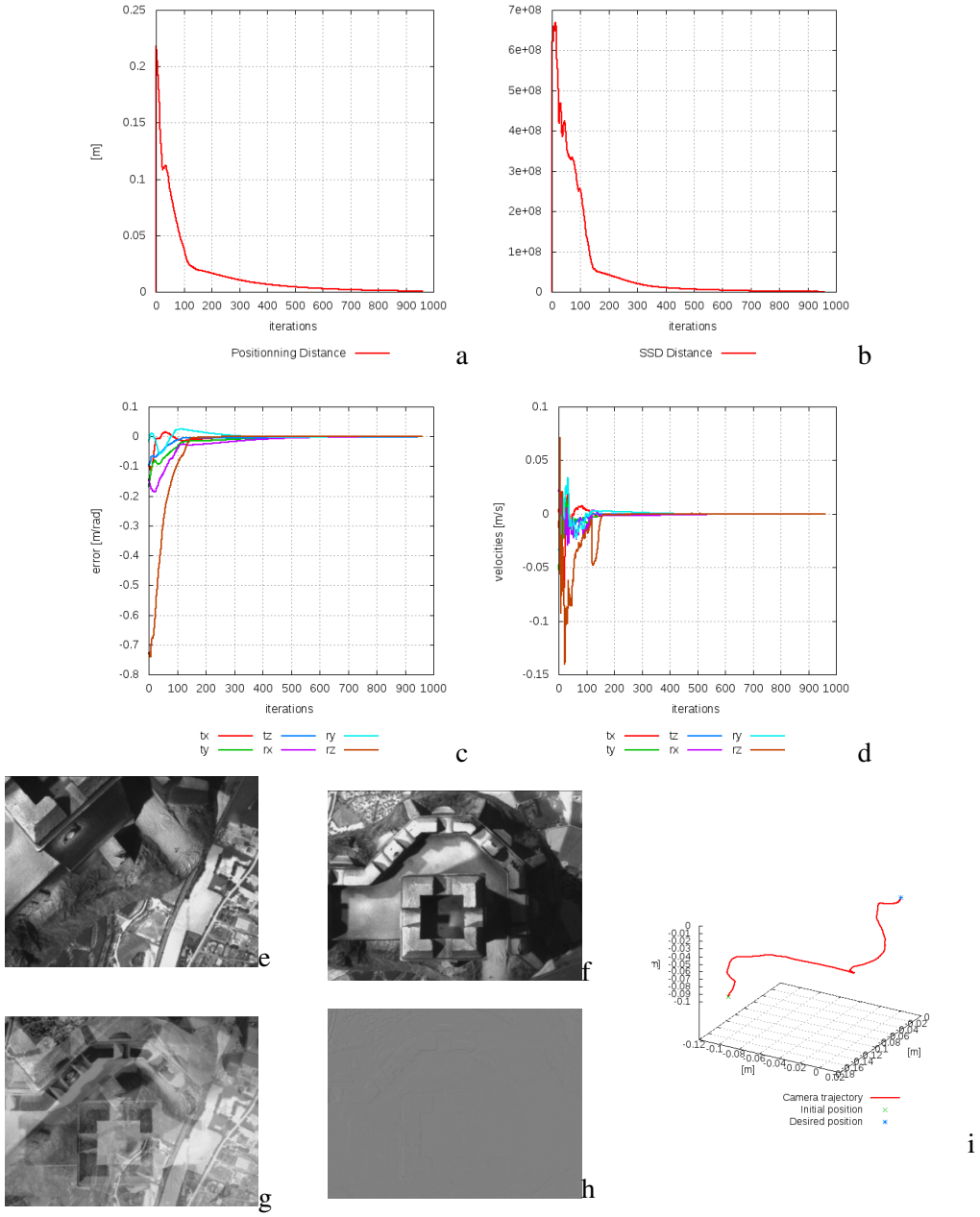


Figure 5.23: *Scene-agnostic CNN-based visual servoing on a 3D scene; (a) Positioning error; (b) SSD distance; (c) Translational and rotational errors; (d) Velocities; (e) Image at initial pose $\mathbf{I_0}$; (f) Image at final pose $\mathbf{I}(\widehat{\mathbf{r}})$; (g) Image error $\mathbf{I_0} - \mathbf{I}^*$ at initial pose; (h) Image error $\mathbf{I}(\widehat{\mathbf{r}}) - \mathbf{I}^*$ at the final pose; (i) 3D trajectory.*

Conclusion

We presented in this chapter a new generic approach for robust VS using deep neural networks. We re-purpose and fine-tune a pre-trained CNNs by substituting the last layer with a new regression layer. The output of the CNN is then used to build a robust control law allowing precise re-positioning tasks. We presented a method to design and collect a synthetic learning dataset for fast fine-tuning of the network. This method allows to be robust to local illumination changes and occlusions. We also show that, with minor modifications, the proposed approach can be scene-agnostic, meaning that it can consider a scene that has never been considered in the learning dataset.

We demonstrated the validity and efficiency of our approach with experiments on a 6DOF robot. The proposed method achieves a positioning task within a millimeter accuracy. Furthermore, we have demonstrated that the proposed approach is robust to strong perturbations, such as lighting variations, occlusions, as well as unknown 3D geometry in the scene.

Lastly, two different networks architectures were used to implement the proposed method, indicating the potential to improve the method as the state-of-the-art of CNN architectures for computer vision progresses.

These works have been published on Arxiv [Bateux 17b] and presented during the RSS17 Workshop: "New Frontiers for Deep Learning in Robotics". A second paper [Bateux 18] has been accepted for submission at ICRA2018.

CONCLUSION AND PERSPECTIVES

As robotics and computer vision systems developments are maturing, expectations are rising from all parts for these technologies to come out in the world. This is currently highlighted by the on-going international race toward the first commercially available autonomous car. This example is only the tree that hides the forest, as robotic systems become more affordable and more safe to interact with through the commercialization of systems for the public, whether it is for entertainment (drones, robotic toys), automatizing daily chore (robot vacuums) or assist customers in stores. As the hardware becomes diversified and available, the need for adequate control systems becomes more pressing. One widespread technique for autonomous navigation or manipulation is to rely on the extraction of geometrical visual features to apply visual servoing techniques. This approach is often very efficient, but does not make use of the whole information present in the images. This means that for many tasks, the accuracy of these systems does not attain maximal precision and progress can be made.

Direct servoing techniques emerged to make advantage of using whole images appearance as a global visual feature can prove to improve the robustness and precision of the associated control scheme. A common component of these techniques is the reliance on cost functions that allow to compare images appearances with each others. The classical approach uses directly the difference between two images as a cost function. One drawback of this cost function is the impossibility to adapt to changing image conditions, as well as the need for a large overlap between the two compared images that restricts the positioning tasks to small displacements.

In this thesis, we presented researches performed in the last three years. The contributions include the definition of new control laws based on histograms, a robust and flexible descriptor. We showed that histograms can be defined to represent various statistical representations of images, such as intensities, color or edges distributions, each able to solve robustly and precisely 6DOF positioning tasks.

As so far the working area of direct visual servoing positioning tasks is limited by the

convexity property of the chosen cost function defining the control law, we presented a new approach to perform the optimization of any similarity criterion between images. This approach, based on a Particle Filter and image transfer techniques was demonstrated to be applicable to both geometrical and direct visual servoing control laws. We showed that the PF-based control can be coupled with the traditional methods to keep the desirable properties of the latter, with the extended working area of the former.

Lastly, we presented an original method to model the visual servoing steps of feature extraction and associated cost function computation, by modeling those through deep learning techniques, allowing the control law to define automatically its own cost function from unprocessed image inputs. This method was proven to successfully solve positioning tasks within known and unknown scenes.

Perspectives

From the proposed contributions, many aspects still need to be studied. First, concerning the proposed framework to perform histogram-based visual servoing, more probability distributions have to be tested and compared. From the methodological side, different measures between histograms need to be evaluated to improve the quality of the resulting cost functions. In the current implementation of the method, the choice of the type of histogram as well as several parameters is empirical. It would be interesting to develop a method to analyze the scene appearance in order to choose the histogram representation (and an associated set of parameters) the most suited to the task to benefit from the flexibility of the method.

Our proposed approach to use a Particle Filter in a VS control law proved to be powerful in term of convergence area and have the potential to be applied with most of the currently used visual features, direct of not. One interesting lead for developing this method further would be to integrate as a prior the camera motion in order to increase the precision of the particle motion from one iteration to another, as in the current version the motion model is considered static.

Finally, the most promising contribution of this thesis is the integration of machine learning techniques into a VS control law. Many more works can be undertaken to extend the current method. One lead would be to integrate reinforcement learning techniques to the control law. Indeed, so far, for each image input, our system starts the computation of a new relative pose estimation, without any constraint regarding the past motion of the robot in space or the evolution of the visual features in the image. Indeed, integrating constraints is a major undertaking, as many works have already been done in the analytic modeling of control laws. Integrating knowledge such as temporal evolution models, prior knowledge on the inputs (dictated by the

physical world properties that define the recorded scenes), as well as various constraints that can be introduced to define the behaviour of the control law, either to generate specific trajectories, increase the amount of exploitable information in the recorded images, avoid perturbations... Comparing different network architectures and learning procedures could also bring more efficiency by determining more precisely the necessary number of parameters required to solve a positioning task under a given set of perturbations.

PUBLICATIONS

Journal articles

- Q. Bateux, E. Marchand. – Histograms-based visual servoing. *IEEE Robotics and Automation Letters*, RA-L, 2(1), pp. 80–87, January 2017.

International conference papers

- Q. Bateux, E. Marchand. – Direct visual servoing based on multiple intensity histograms. In *IEEE Int. Conf. on Robotics and Automation*, ICRA'15, pp. 6019–6024, Seattle, WA, May 2015.
- Q. Bateux, E. Marchand. – Particle filter-based direct visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IROS'16, pp. 4180−4186, October 2016.
- Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, P. Corke. – Training Deep Neural Networks for Visual Servoing. In *IEEE Int. Conf. on Robotics and Automation*, ICRA'18, Brisbane, Australia, 2018.

Workshop presentation papers

- Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, P. Corke. Visual servoing from deep neural networks. In *Robotics: Science and Systems Workshop: "New Frontiers for Deep Learning in Robotics"*, Boston, MA, 2017.

BIBLIOGRAPHY

[Achanta 12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk. – Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[Arulampalam 02] M. S. Arulampalam, S. Maskell, N. Gordon, T. Clapp. – A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[Bakthavatchalam 13] M. Bakthavatchalam, F. Chaumette, E. Marchand. – Photometric moments: New promising candidates for visual servoing. – *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 5241–5246. IEEE, 2013.

[Bateux 15] Q. Bateux, E. Marchand. – Direct visual servoing based on multiple intensity histograms. – *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 6019–6024, Seattle, WA, 2015.

[Bateux 16] Q. Bateux, E. Marchand. – Particle filter-based direct visual servoing. – *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4180–4186, 2016.

[Bateux 17a] Q. Bateux, E. Marchand. – Histograms-based visual servoing. *IEEE Robotics and Automation Letters*, 2(1):80–87, January 2017.

[Bateux 17b] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, P. Corke. – Visual servoing from deep neural networks. *arxiv*, abs/1705.08940, 2017.

[Bateux 18] Quentin Bateux, Eric Marchand, Jürgen Leitner, François Chaumette, Peter I Corke. – Training Deep Neural Networks for Visual Servoing. – *ICRA 2018 - IEEE International Conference on Robotics and Automation*, pp. 1–8, Brisbane, Australia, mai 2018.

[Bhattacharyya 43] A. Bhattacharyya. – On a measure of divergence between two statistical populations defined by their probability distribution. *Bull. Calcutta Math. Soc*, 1943.

[Blanc 05] G. Blanc, Y. Mezouar, P. Martinet. – Indoor navigation of a wheeled mobile robot along visual routes. – *Int. Conf. on Robotics and Automation, 2005. ICRA 2005.*, pp. 3354–3359. IEEE, 2005.

[Chatelain 15] P. Chatelain, A. Krupa, N. Navab. – 3d ultrasound-guided robotic steering of a flexible needle via visual servoing. – *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 2250–2255. IEEE, 2015.

[Chaumette 98a] F. Chaumette. – Potential problems of stability and convergence in image-based and position-based visual servoing. *The Confluence of Vision and Control*, éd. par D. Kriegman, G . Hager, A.S. Morse, pp. 66–78. – LNCIS Series, No 237, Springer-Verlag, 1998.

[Chaumette 98b] F. Chaumette. – Potential problems of stability and convergence in image-based and position-based visual servoing. *The confluence of vision and control*, pp. 66–78, 1998.

[Chaumette 06] F. Chaumette, S. Hutchinson. – Visual servo control, Part I: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, December 2006.

[Chaumette 07] F. Chaumette, S. Hutchinson. – Visual servo control, Part II: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, March 2007.

[Cherubini 10] A. Cherubini, F. Chaumette. – A redundancy-based approach for obstacle avoidance in mobile robot navigation. – *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 5700–5705. IEEE, 2010.

[Ciodaro 12] T. Ciodaro, D. Deva, J-M. De Seixas, D. Damazio. – Online particle detection with neural networks based on topological calorimetry information. – *Journal of physics: conference series*, vol. 368, p. 012030. IOP Publishing, 2012.

[Collewet 08a] C. Collewet, E. Marchand, F. Chaumette. – Asservissement visuel bas sur des informations photomtriques. – *16e congrs francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle, RFIA'08*, Amiens, France, janvier 2008.

[Collewet 08b] C. Collewet, E. Marchand, F. Chaumette. – Visual servoing set free from image processing. – *IEEE Int. Conf. on Robotics and Automation, ICRA'08*, pp. 81–86, Pasadena, CA, May 2008.

[Collewet 11] C. Collewet, E. Marchand. – Photometric visual servoing. *IEEE Trans. on Robotics*, 27(4):828–834, August 2011.

[Comaniciu 03a] D. Comaniciu, V. Ramesh, P. Meer. – Kernel-based object tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):564–577, 2003.

[Comaniciu 03b] D. Comaniciu, V. Ramesh, P. Meer. – Kernel-based object tracking. *IEEE Trans. on PAMI*, 25(5):564–577, May 2003.

[Crombez 15] N. Crombez, G. Caron, E. M. Mouaddib. – Photometric gaussian mixtures based visual servoing. – *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 5486–5491, 2015.

[Dalal 05] N. Dalal, B. Triggs. – Histograms of oriented gradients for human detection. – *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR'05*, pp. 886–893, 2005.

[Dame 09] A. Dame, E. Marchand. – Entropy-based visual servoing. – *IEEE Int. Conf. on Robotics and Automation, ICRA'09*, pp. 707–713, Kobe, Japan, May 2009.

[Dame 11] A. Dame, E. Marchand. – Mutual information-based visual servoing. *IEEE Trans. on Robotics*, 27(5):958–969, octobre 2011.

[Dame 13] A. Dame, E. Marchand. – Using mutual information for appearance-based visual path following. *Robotics and Autonomous Systems*, 61(3):259–270, 2013.

[Deng 09] J. Deng, W. Dong, R. Socher, L-J. Li, K. Li, L. Fei-Fei. – Imagenet: A large-scale hierarchical image database. – *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

[Déniz 11] O. Déniz, G. Bueno, J. Salido, F. De la Torre. – Face recognition using histograms of oriented gradients. *Pattern Recognition Letters*, 32(12):1598–1603, 2011.

[DeTone 16] D. DeTone, T. Malisiewicz, A. Rabinovich. – Deep image homography estimation. – *Robotics: Science and Systems 2016, Workshop on Limits and Potentials of Deep Learning in Robotics*, 2016. – arXiv preprint arXiv:1606.03798.

[Diosi 07] A. Diosi, A. Remazeilles, S. Segvic, F. Chaumette. – Outdoor visual path following experiments. – *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'07*, pp. 4265–4270, San Diego, CA, October 2007.

[Eigen 14] D. Eigen, C. Puhrsch, R. Fergus. – Depth map prediction from a single image using a multi-scale deep network. – *Advances in neural information processing systems*, pp. 2366–2374, 2014.

[Finn 16] C. Finn, Xin Yu Tan, Yan Duan, T. Darrell, S. Levine, P. Abbeel. – Deep spatial autoencoders for visuomotor learning. – *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 512–519, Stockholm, Sweden, 2016.

[Gordon 93] N J Gordon, D J Salmond, A FM Smith. – Novel approach to nonlinear/non-gaussian bayesian state estimation. – *Radar and Signal Processing, IEE Proceedings F*, vol. 140, pp. 107–113. IET, 1993.

[Hager 04] G. Hager, M. Dewan, C. Stewart. – Multiple kernel tracking with ssd. – *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR'04*, pp. 790–797, juin 2004.

[Hahnloser 00] R. H-R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, H. S. Seung. – Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[Hinton 12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et al. – Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[Hutchinson 96] S. Hutchinson, G. Hager, P. Corke. – A tutorial on visual servo control. *IEEE Trans. on Robotics and Automation*, 12(5):651–670, octobre 1996.

[Jeong 04] S. Jeong, C.S. Won, R.M. Gray. – Image retrieval using color histograms generated by gauss mixture vector quantization. *Computer Vision and Image Understanding*, 94(1):44–66, 2004.

[Jia 14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell. – Caffe: Convolutional architecture for fast feature embedding. – *ACM Int. Conf. on Multimedia, MM'14*, pp. 675–678, Orlando, FL, 2014.

[Karayev 13] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, H. Winnemoeller. – Recognizing image style. – *British Machine Vision Conference (BMVC)*, 2013.

[Kendall 15] A. Kendall, M. Grimes, R. Cipolla. – Posenet: A convolutional network for real-time 6-dof camera relocalization. – *IEEE Int. Conf. on Computer Vision, CVPR'2015*, pp. 2938–2946, 2015.

[Krizhevsky 12] A. Krizhevsky, I. Sutskever, G. Hinton. – Imagenet classification with deep convolutional neural networks. – *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[LeCun 15] Yann LeCun, Yoshua Bengio, Geoffrey Hinton. – Deep learning. *Nature*, 521(7553):436–444, 2015.

[Levine 16] S. Levine, C. Finn, T. Darrell, P. Abbeel. – End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[Lowe 99] D. G. Lowe. – Object recognition from local scale-invariant features. – *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157. Ieee, 1999.

[Ma 12] Y. Ma, S. Soatto, J. Kosecka, S. S. Sastry. – *An invitation to 3-d vision: from images to geometric models*. – Springer Science & Business Media, vol. 26, 2012.

[Malis 04] E. Malis. – Improving vision-based control using efficient second-order minimization techniques. – *IEEE Int. Conf. on Robotics and Automation, ICRA'04*, vol. 2, pp. 1843–1848, New Orleans, avril 2004.

[Marchand 05] E. Marchand, F. Spindler, F. Chaumette. – ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12(4):40–52, décembre 2005. – Special Issue on "Software Packages for Vision-Based Control of Motion", P. Oh, D. Burschka (Eds.).

[Marchand 10] E. Marchand, C. Collewet. – Using image gradient as a visual feature for visual servoing. – *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 5687–5692. IEEE, 2010.

[Matusita 55] K. Matusita. – Decision rules, based on the distance, for problems of fit, two samples, and estimation. *The Annals of Mathematical Statistics*, pp. 631–640, 1955.

[Pérez 02] P. Pérez, C Hue, J. Vermaak, M. Gangnet. – Color-based probabilistic tracking. *Computer vision—ECCV 2002*, pp. 661–675. – Springer, 2002.

[Petit 13] A Petit, E Marchand, K Kanani. – A robust model-based tracker combining geometrical and color edge information. – *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3719–3724. IEEE, 2013.

[Pinto 16] L. Pinto, A. Gupta. – Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. – *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 3406–3413, Stockholm, Sweden, May 2016.

[Russell 08] B. Russell, A. Torralba, K. Murphy, W. Freeman. – Labelme: a database and web-based tool for image annotation. *Int. Journal of Computer Vision*, 77(1-3):157–173, 2008.

[Silver 16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al. – Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver 17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al. – Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Simonyan 15] K. Simonyan, A. Zisserman. – Very deep convolutional networks for large-scale image recognition. – *Int. Conf. on Learning Representations, ICLR'15*, San Diego, CA, mai 2015.

[Swain 92] M.J. Swain, D.H. Ballard. – Indexing via color histograms. *Active Perception and Robot Vision*, pp. 261–273. – Springer, 1992.

[Teuliere 11] C. Teuliere, L. Eck, E. Marchand. – Chasing a moving target from a flying uav. – *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 4929–4934. IEEE, 2011.

[Teuliere 12] C. Teuliere, E. Marchand. – Direct 3d servoing using dense depth maps. – *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'12*, pp. 1741–1746, Vilamoura, Portugal, October 2012.

[Unser 93] M. Unser, A. Aldroubi, M. Eden. – B-spline signal processing. i. theory. *IEEE transactions on signal processing*, 41(2):821–833, 1993.

[VanDerMerwe 01] R. Van Der Merwe, A. Doucet, N. De Freitas, E. A. Wan. – The unscented particle filter. – *Advances in neural information processing systems*, pp. 584–590, 2001.

[Zhang 00] Z. Zhang. – A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[Zhang 13] H. Zhang, Y. Wang, X. Jiang. – An improved shot segmentation algorithm based on color histograms for decompressed videos. – *Image and Signal Processing (CISP), 2013 6th International Congress on*, vol. 1, pp. 86–90. IEEE, 2013.

[Zhang 15] F. Zhang, J. Leitner, M. Milford, B. Upcroft, P. Corke. – Towards vision-based deep reinforcement learning for robotic motion control. – *Australasian Conf. on Robotics and Automation (ACRA)*, 2015.

Résumé

Dans cette thèse, nous nous concentrons sur les techniques d'asservissement visuel (AV), critiques pour de nombreuses applications de vision robotique et insistons principalement sur les AV directs.

Afin d'améliorer l'état de l'art des méthodes directes, nous nous intéressons à plusieurs composantes des lois de contrôle d'AV traditionnelles. Nous proposons d'abord un cadre générique pour considérer l'histogramme comme une nouvelle caractéristique visuelle. Cela permet de définir des lois de contrôle efficaces en permettant de choisir parmi n'importe quel type d'histogramme pour décrire des images, depuis l'histogramme d'intensité à l'histogramme couleur, en passant par les histogrammes de Gradients Orientés. Une nouvelle loi d'asservissement visuel direct est ensuite proposée, basée sur un filtre particulaire pour remplacer la partie optimisation des tâches d'AV classiques, permettant d'accomplir des tâches associées à des fonctions de coûts hautement non linéaires et non convexes. L'estimation du filtre particulaire peut être calculée en temps réel à l'aide de techniques de transfert d'images permettant d'évaluer les mouvements de caméra associés aux déplacements des caractéristiques visuelles considérées dans l'image. Enfin, nous présentons une nouvelle manière de modéliser le problème de l'AV en utilisant l'apprentissage profond et les réseaux neuronaux convolutifs pour pallier à la difficulté de modélisation des problèmes non convexes via les méthodes analytiques classiques. En utilisant des techniques de transfert d'images, nous proposons une méthode permettant de générer rapidement des ensembles de données d'apprentissage de grande taille afin d'affiner des architectures de réseau pré-entrainés sur des tâches connexes, et résoudre des tâches d'AV. Nous montrons que cette méthode peut être appliquée à la fois pour modéliser des scènes connues, et plus généralement peut être utilisée pour modéliser des estimations de pose relative entre des couples de points de vue pris de scènes arbitraires.

Abstract

In this thesis we focus on visual servoing (VS) techniques, critical for many robotic vision applications and we focus mainly on direct VS.

In order to improve the state-of-the-art of direct methods, we tackle several components of traditional VS control laws. We first propose a method to consider histograms as a new visual servoing feature. It allows the definition of efficient control laws by allowing to choose from any type of histograms to describe images, from intensity to color histograms, or Histograms of Oriented Gradients. A novel direct visual servoing control law is then proposed, based on a particle filter to perform the optimization part of visual servoing tasks, allowing to accomplish tasks associated with highly non-linear and non-convex cost functions. The Particle Filter estimate can be computed in real-time through the use of image transfer techniques to evaluate camera motions associated to suitable displacements of the considered visual features in the image. Lastly, we present a novel way of modeling the visual servoing problem through the use of deep learning and Convolutional Neural Networks to alleviate the difficulty to model non-convex problems through classical analytic methods. By using image transfer techniques, we propose a method to generate quickly large training datasets in order to fine-tune existing network architectures to solve VS tasks.We shows that this method can be applied both to model known static scenes, or more generally to model relative pose estimations between couples of viewpoints from arbitrary scenes.

Keywords: Robotic Vision, Visual Servoing, Histograms, Particle Filter, Deep Learning.