Projecte Final de Carrera

Solving Correspondences for Non-Rigid Deformations

Author: Anna Tamarit Sariol
Advisor: Francesc Moreno Noguer

Universitat Politècnica de Catalunya
Academic year: 2009-2010

Contents

Chapter 1

Introduction

Solving the correspondence problem between images is one of the main topics in computer vision. The problem of recognizing objects in images is very challenging because images may be acquired at different scales and perspectives or even because objects may vary their shape. Our project is focused on this last topic, the problem of solving correspondences for non-rigid deformations.

In computer vision, one of the most used methodologies to solve identifications between images has consisted on the study of the objects' local features. Working with local properties, problems such as partial occlusion, background clutter or illumination changes can be solved or partially solved. Despite of the fact that most of the current works try to exclusively solve the matches where only rotation, illumination, scale and perspectives changes are considered, we have proposed a method which also deals with non-rigid deformations. More specifically, we have tackled the problem of finding correspondences for initially planar meshes which can be deformed as if they were some tissue. As it will be seen, this problem has been very few addressed in the literature and our approach is different to the ones we have found in this field.

Our method relies in a well-known method, the ferns evaluation. The principal advantage of this tool is that it is based on a training process that makes detection a lot faster than other methods. Precisely, multiple views of the most important keypoints of the object are used to create non-hierarchical structures, ferns, which discriminate among the different keypoints. Each fern consists of a small set of binary tests and returns the probability that a patch belongs to any one of the classes that have been learned during training. The responses to different ferns are then combined in a simple way. The main difference between the traditional methods using ferns and ours is that we train ferns for deformed views of the objects.

We have implemented the method and shown that it represents indeed a great improvement with respect to one of the most used techniques for local features, SIFT. When dealing with deformations, we have found that detection rate is increased on almost a 20% by our method. Figure 1.1 shows an example of the results given by our method.

The core structure of the report consist on three chapters, where we respectively discuss the state of the art, present the method and analyze the results obtained. In Chapter 2, we comment the work done in the fields of detecting and describing local features. We also pay special attention to the methods developed for non-rigid deformation. Learning and redacting

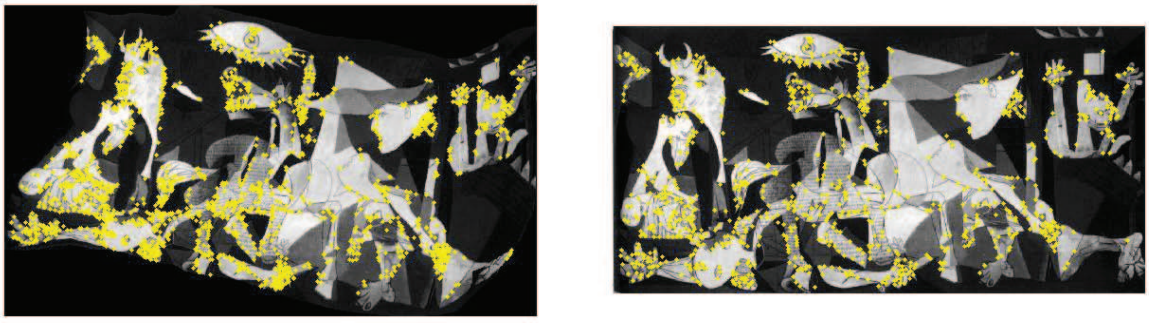

Figure 1.1: At left, points detected by our method in a deformed image of the Guernica are shown. At right, same results on the reference image. In this deformed image, a total of 2154 points have been positively detected among an initial set of 3000 points.

the state of the art has been very enlightening and amusing. Despite the length chapter, we hope readers, instead of getting bored, will enjoy learning general procedures in the computer vision. We also hope mathematics notation and figures will help clarify ideas introduced. In advance, vocabulary may be somehow affected by one or the other papers in which this chapter is based. Nevertheless, there has been a work homogenizing it.

Chapter 3 explains the fundamental parts of our method in an extensive way. After a global overview of all the procedure, we examine each of the stages taking place in the training process and in the evaluation of a new input image. In other words, we describe how we decide which local features are kept for the training process, we explain how the different deformed views are created to build the ferns' structure and finally we explain the ferns' training and evaluation processes.

Finally, in Chapter 4, we give some results when applying the method we have implemented. In order to make them comprehensive, we first present the way we have evaluated our method and also give the parameters used when using another method, SIFT, so that we can compare our results with others. Then, comparisons are held and, moreover, since our method depends on a selection of various parameters, we execute some experiments to study the effect of each parameter in the results obtained. This last part allows us to use the method in an optimal way.

Chapter 2

State of the Art

In this chapter, we present the state of the art techniques which have somehow relations to the method that we introduce in next chapter. We already start focusing on the techniques relying on local features. We describe detection procedures and also descriptors being used. Finally, we insist in the methods which pay attention to the non-rigid deformations. These ones are still very few and there is not much works willing to deal with them.

As said, we start commenting methods based on local features. Local features consist on properties that can easily distinguish limited areas of the image. In computer vision, local features have shown to be well suited to wide baseline matching, object and texture recognition or even classification, image retrieval and many other applications. Their value relies on the fact that they are intrinsically robust to occlusion, background clutter and other image changes. Taking into account their virtues, the effort has been centered in creating local features which not only stand for their distinctiveness power but can hold more and more properties such as, for example, robustness to scale, viewpoint changes or image blur.

The most common procedure when dealing with local features is:

(i) **Region detection**. Interest points are detected to select matchable points between different images. As we will see, depending on the method, the region selection can be prefixed or automatically computed in order to make the region invariant to some changes such as scale or affine transformations.

(ii) **Local features description**. They are used to represent the previous detected region. The goal is to be as discriminative as possible.

(iii) **Similarity measures or methods**. Once the local features have been created, the comparison between local features of different images has to be done. In order to accomplish our objective, some metrics have to be defined such that the descriptors of same regions in different images get closer than others.

(iv) **Matching**. Matching is primarily done assigning the nearest feature in the other image according to the similarity measures. Typically, additional constraints are incorporated, for example, global location between local features.

Following these steps, we will next give some highlights on the methods currently being used at each stage and also some ideas about pros and cons of each one. Since structure in local feature methods is pretty much clear, chapter is divided according to the different steps we presented above. Nevertheless, since most of the current algorithms involve all the parts, we may move forward and backward of the main explanation sequence. In Section 2.4, we will comment the state of the art in dealing with image matching with non-rigid objects. As it will be seen, very few things have been done in detection of non-rigid objects.

2.1 Local Detectors

In this section, detectors are described on base of their benefits and it is somehow also an historical overview about detectors creation. In Figure 2.1, an schematic graph gives clarity to relations between them and also a general perspective on detectors. In addition, Table 2.1 summarizes pros and cons of the methods explained.

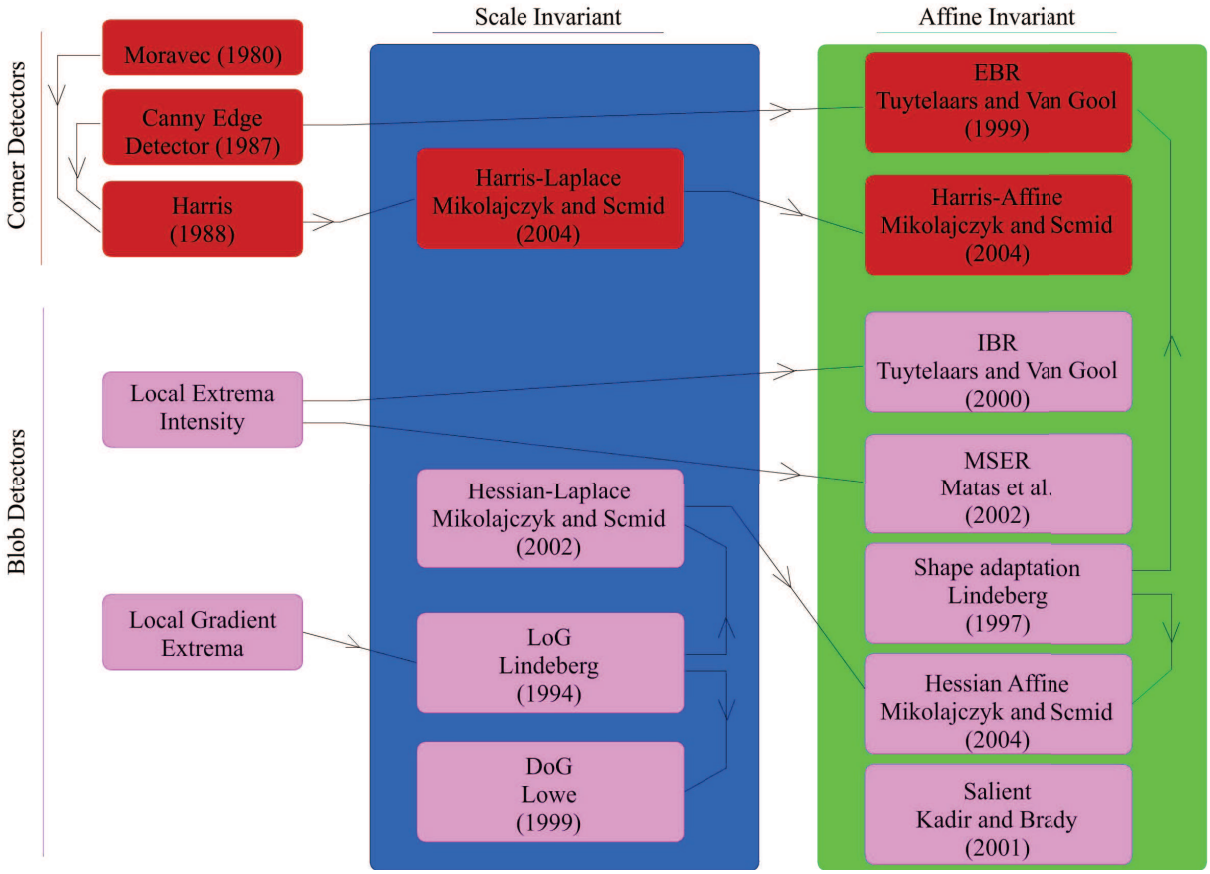

Figure 2.1: Overview of detectors used in computer vision to extract regions of interest.

Detector	Pros	Cons
Intensity Extrema [9]	Resists very well monotonic intensity transformations Less likely to lie close an edge so planar assumption is more realistic	Since a Gaussian smoothing is initially applied, point location is not accurate
Harris [8]	Contains a large amount of information → distinctive power Very well localized in terms of accuracy	As it detects corners, planar assumption is weak
Scale Invariant		
Harris-Laplace [22][20]	Improvement of the Harris detector so that regions are scale-invariant	Still, affine invariance has to be added
DoG [18]	Even not affine designed, it can handle relatively well viewpoint changes up to 50 degrees Faster than Harris-Laplace	If not carefully applied, points can be detected at different scales and therefore a posteriori matching can be confusing
Affine Invariant		
Harris-Affine [22][20]	It detects more regions than others so results are better for occlusion or clutter	Still, based on corners so planar assumption is weak
EBR [32]	Edges are very stable features	Weak planar assumption
IBR[32]	Robust to inexact location of the interest point	Since ellipse size is doubled, there is some lost of the local character of the region and a certain non planar assumption risk even they are not a corner interest point.
MSER [19][33]	While relying on intensity extrema, location accuracy is not lost as there isn't image presmoothing The most efficient one	It does not respond very well to blur
Salient [11]	Interest points are not defined as an extremum for a filter response. It is a more general approach	The less efficient one

Table 2.1: Pros and Cons summary of some of the detectors explained

The Simplest Ones

In order to find interest points, the techniques more obvious would be the ones that rely on detecting local maxima of the intensity values along the image pixels or on detecting points with maximum variation of the intensity (local maxima of the intensity derivative).

Some of the simplest ones are described below. Figure 2.2 shows the results when being applied to an image.

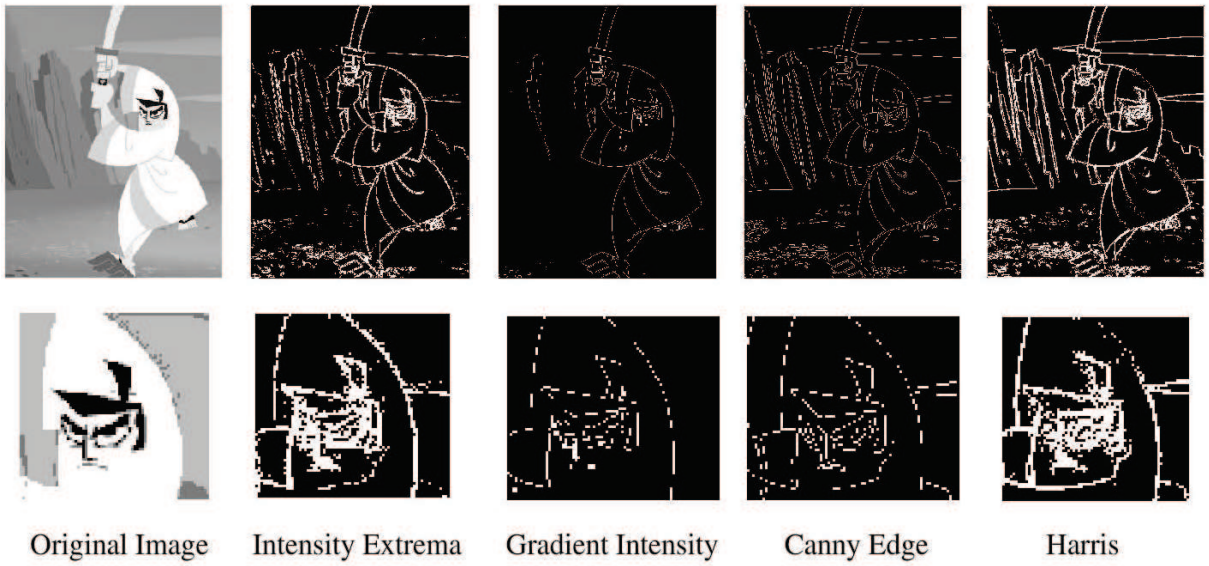

Figure 2.2: From left to right: original image, responses to the intensity extrema detector, gradient intensity detector, Canny edge detector and Harris detector with $k = 0.1$ respectively. At second row a part of image is enlarge so differences between detectors can be better appreciated

- ***Local Extrema of Intensity.***

Given an $m \times n$ image $\mathbf{I}$, it can be represented as a matrix:

$$\mathbf{I} = I(u,v) = \begin{pmatrix} I_{11} = I(u_1, v_1) & \cdots & I_{1n} \\ \vdots & \ddots & \vdots \\ I_{m1} & \cdots & I_{mn} \end{pmatrix}, \text{ where } (u,v) \text{ are the pixel coordinates.}$$

Note that, since we are associating a single intensity value to each pixel, we are assuming we are working with a grayscale image. If not said the opposed, we will always assume it[1]. We can consider a "raw" way of extracting the local extrema by inspecting, for each pixel, its eight neighbors (eight at most). So we would take as set of interest points,

$$S = \{(u,v) : \underset{(u',v') \in [u-1,u+1] \times [v-1,v+1]}{\operatorname{argmin}} I(u',v') = (u,v)\}.$$

However, this is not an efficient way to explore the whole image. Since convolution[2] can be

[1] There are many ways to convert an RGB image to grayscale, one of the most common is the conversion $I = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B$.

[2] Let $\mathbf{A}$, $\mathbf{B}$ be matrices, with size $n_a \times m_a$ and $n_b \times m_b$ respectively, the convolution matrix $\mathbf{M} = \mathbf{A} * \mathbf{B}$ is a $(n_a - n_b + 1) \times (n_a - n_b + 1)$ matrix, assuming $\dim \mathbf{A} \geq \dim \mathbf{B}$. Nevertheless, it is usual to consider as the convolution result of $\mathbf{A} * \mathbf{B}$ only its central part, $\mathbf{M}'$, of the latter matrix $\mathbf{M}$ of size the same as $\mathbf{A}$. Last definition is the one considered in all the text.

very efficiently done, the previous operation can be expressed as looking for local maxima in

$$\mathbf{M} = |\mathbf{I} * \mathbf{A}|, \quad \text{where} \quad \mathbf{A} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Note that, assuming intensity values in a $[0, 1]$ range, pixels in an uniform intensity region would get a 0 after the convolution while pixels with intensities values the most different to their neighborhood intensity values could get up to 16. Then, it is only needed to fix a threshold and keep the pixels above the threshold.

- ***Local Extrema of Intensity Gradient.***

In order to find local extrema of the intensity gradient, the method used is to convolve the image matrix $\mathbf{I}$ with some functions which act as simple directional derivatives. By defining

$$\mathbf{G_x} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{G_y} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix},$$

we can obtain $\mathbf{I_x} \doteq \mathbf{I} * \mathbf{G_x}$ and $\mathbf{I_y} \doteq \mathbf{I} * \mathbf{G_y}$ which are first numerical approximations of the derivatives along X and Y axis respectively. $\mathbf{G_x}$ and $\mathbf{G_y}$ are the so called *Sobel operators*. Once gradient intensities have been obtained, the most relevant ones are picked, meaning those whose modulus is beyond a threshold. As we will discuss later, it is often important to retain the value $\theta(u, v) = \arctan\left(\frac{I_y(u,v)}{I_x(u,v)}\right)$, which indicates the gradient orientation at each pixel. That way, it is possible to align features so that comparisons are always done with the same known orientations.

- ***Canny Edge Detector.***

The Canny edge detector is, as the name shows, not an interest point detector but an edge detector. Nevertheless, it is sometimes used by local feature algorithms, such as in the EBR detector (see Section 2.1.2 or [32]) or the Harris detector as well. We will briefly explain it. The Canny edge detector was one of the earliest detectors. Developed in 1986 by John F. Canny, it is computed as follows:

  - *Gaussian smoothing.*

    A Gaussian mask $\mathbf{g}$, meaning a square matrix whose values are the discretization of a certain gaussian $\mathbf{g} \sim \mathcal{N}(0, \sigma^2)$, is used to obtain $\mathbf{I'} = \mathbf{I} * \mathbf{g}$ so that $\mathbf{I'}$ is almost like $\mathbf{I}$ but the intensity values of their pixels receive certain influence, depending on σ of the intensity values around the neighborhood. Smoothing $\mathbf{I}$ provides a way to control influence of noise in the detector results. The larger the variance is, the more smoothness and therefore the lower the detector's sensitivity to noise is, but also, the precise location of the edges decreases. Thus, there is a trade-off. For instance, Figure 2.3 shows the result of convolving

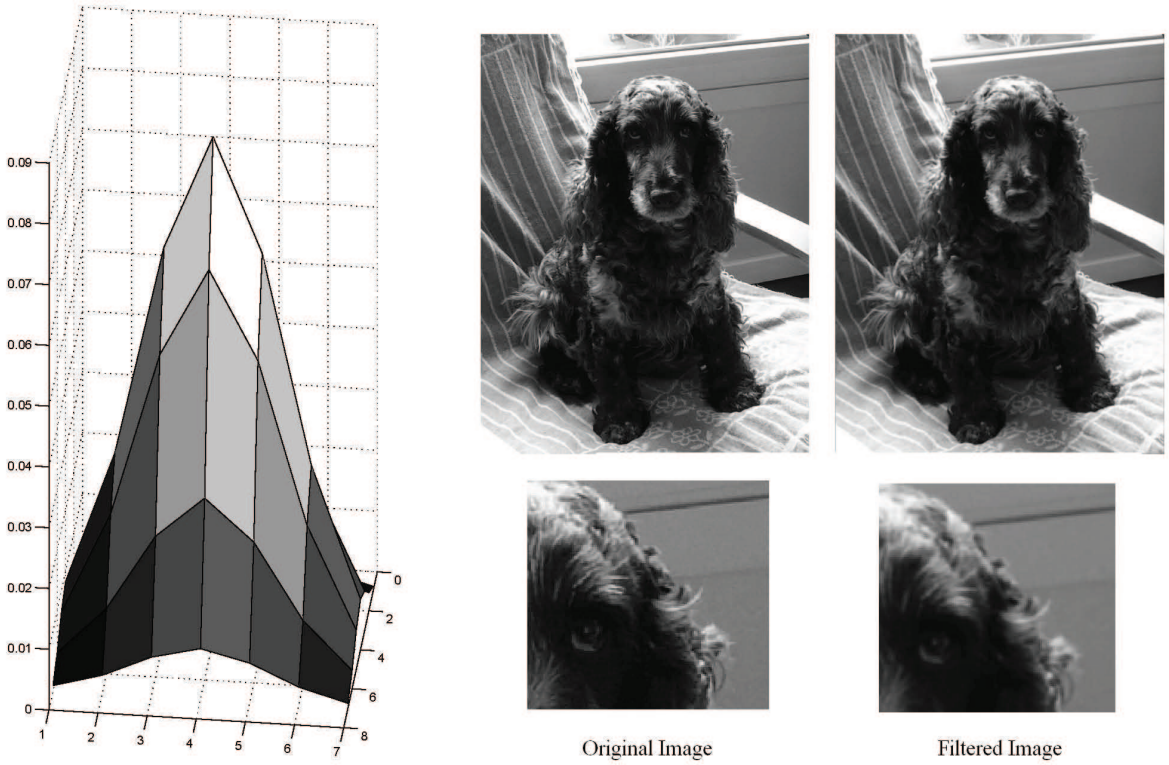

Figure 2.3: From left to right: The gaussian mask, the original image and the image after filtering. In the enlarged images it is very clear that filtered image has more blur and also contrast between white and black is reduced

an image $\mathbf{I}$ with a 7×7 Gaussian filter with variance $\sigma = 1.4$. It can be observed that contrast in edges gets reduced and intensity changes are more gradual along pixels.

– *Finding edges.*

Edge candidates are found applying Sobel operators in order to locate gradient local extrema (as discussed before). Then, gradient modulus at each pixel is taken as an edge strength measure. Similarly, gradient direction is computed for each pixel. Usually, direction is given only within a range $\theta \in [0, \pi]$ *rad*. Moreover, only four directions are considered: $\theta \in \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\}$ *rad*.

– *Non maxima suppression.*

After having calculated the image gradients, pixels are inspected to verify if they reach local maxima in the gradient direction. That is, we compare their modulus with the modulus of their neighbors on the normal direction. For example, for pixels marked as being oriented in the direction $\theta = 0$ *rad*, it has to be checked that their gradient modulus exceeds

gradient modulus of their north and south neighbors; pixels oriented in the $\theta = \frac{\pi}{4}\ rad$ have to exceed in modulus the north-west and the south-east neighbors and analogous to the other two directions. Pixels not having passed the test are set to 0.

– *Tracing edges.*

Tracing edges is done using a hysteresis cycle. First, gradient modulus over a high threshold are chosen to be like 'seeds' for finding strong edges. The edge is extended along both directions until edge strength in that directions falls below a low threshold.

The results of this detector are shown in Figure 2.2.

- ***Harris Detector.***

Based on the Canny edge detector and also on Moravec's corner detector [23], Harris and Stephens proposed in [8] a new detector for both corners and edges. The main idea is to study how similar a patch centered at a certain pixel would be from a patch centered at a pixel on its neighborhood. This can be written in terms of the autocorrelation function as follows:

$$c(x,y) = \sum_{u,v} W(u,v) \cdot (I(x,y) - I(u,v))^2, \quad \text{where } \mathbf{W} \text{ is a window around } (x,y)$$

. Approximating $I(u,v)$ by the first two terms of its Taylor series around point (x,y), so

$$I(u,v) \approx I(x,y) + (I_x(x,y), I_y(x,y)) \begin{pmatrix} u-x \\ v-y \end{pmatrix} = (I_x(x,y), I_y(x,y)) \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

where $\mathbf{I_x}$ and $\mathbf{I_y}$ are the partial derivatives of $\mathbf{I}$. This is $\mathbf{I_x} = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} * \mathbf{I}$ and $\mathbf{I_y} = \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}^T * \mathbf{I}$.

Then, we have:

$$\begin{aligned} c(x,y) &= \sum_{u,v} W(u,v) \cdot (I(x,y) - I(u,v))^2 \approx \sum_{u,v} W(u,v) \cdot \left([I_x(x,y), I_y(x,y)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 \\ &\approx \sum_{u,v} W(u,v) \cdot (\Delta x, \Delta y) \begin{pmatrix} I_x(x,y) \\ I_y(x,y) \end{pmatrix} (I_x(x,y), I_y(x,y)) \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \\ &\approx (\Delta x, \Delta y)\, M(x,y) \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \end{aligned}$$

where $\mathbf{M}$ is the second moment matrix, similar to autocorrelation, and it is defined as

$$\mathbf{M} = M(x,y) \doteq \begin{pmatrix} \mathbf{W} * I_x^2(x,y) & \mathbf{W} * (I_x(x,y) \cdot I_y(x,y)) \\ \mathbf{W} * (I_y(x,y) \cdot I_x(x,y)) & \mathbf{W} * I_y^2(x,y) \end{pmatrix} \tag{2.1}$$

In order to make the detector robust to noise, $\mathbf{W}$ is chosen to be a gaussian window $\mathcal{N}(0,\sigma)$. Note that matrix $M(x,y)$ contains all the differential operators describing the geometry of the image surface at a given point (x,y). Let λ_1 and λ_2 be its eigenvalues, then we can study three cases:

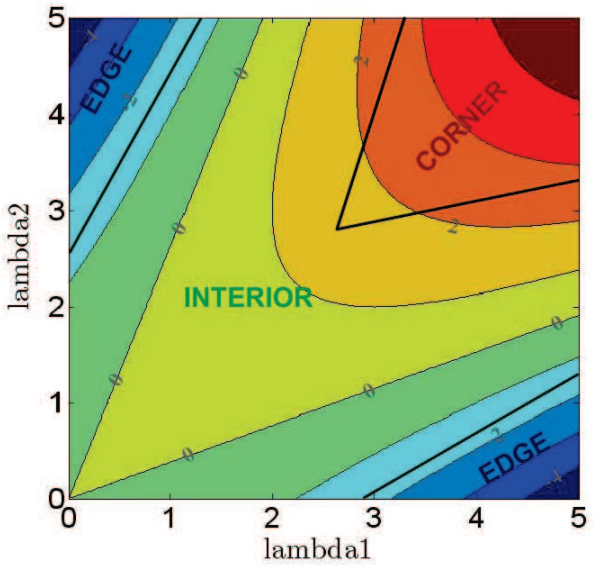

Figure 2.4: Contours of the function $F(x, y)$ in Equation (2.2) with $k = 0.2$. The different cases according to λ values can be observed.

1. Both curvatures are small, that is λ_1 and λ_2 small. Then the autocorrelation function is flat and the windowed image region is of approximately constant intensity.
2. λ_1 and λ_2 are high so that local autocorrelation function is peaked, meaning small moves from (x, y) make patches (one patch centered at (x, y) and the other at (u, v)) very different. We then interpret that patch is centered on a corner ((x, y) is a corner point).
3. If one of the curvatures is high and the other low, that indicates a sharped change of the autocorrelation only in a direction. Hence we have found an edge.

Having observed this casuistics, Harris used the trace and determinant operators to obtain equivalent information without having to explicitly calculate the eigenvalues. Observe that:

1. is equivalent to $\text{trace}(\mathbf{M}) = \lambda_1 + \lambda_2 \approx 0$ and $\det(\mathbf{M}) = \lambda_1 \cdot \lambda_2 \approx 0$,
2. is equivalent to $\text{trace}(\mathbf{M})$ large and $\det(\mathbf{M})$ large, and
3. is equivalent to $\text{trace}(\mathbf{M})$ large and $\det(\mathbf{M}) \approx 0$.

An evaluation function is defined as

$$F(x, y) = \det(\mathbf{M}) - k \cdot \text{trace}(\mathbf{M})^2, \tag{2.2}$$

where k is a tunable parameter that usually moves from 0.04 to 0.15. Also, a positive threshold T is set and corners are points (x, y) such that $F(x, y) \geq T$. This is made more clear from the iso-response contours of Figure 2.4.

It is important to insist that all the methods explained above detect interest points but

don't describe their surrounding region. However, they are used as a first step in more complex descriptors as it will be shown in the following sections.

2.1.1 Scale Invariant Detectors

Scale invariant detectors are those which can accept similarity transformations, that are transformations that preserve the shape of the geometrical objects.

Note that, when demanding scale invariant detectors, what we are exactly saying is that we do not care about scale change as long as it is the same in every direction.

The typical approach applied in order to design scale invariant detectors is to extract interest points at several scales and to use all these points to represent the same interest point, a sort of *pyramidal representation*. More precisely, the technique is referred as **Laplacian of Gaussian detector, LoG**. The major problem with multi-scale approaches is that, in general, a local image structure exists in a range of scales and, as a consequence, there are many points with little variance in location and scale representing the same interest structure. A posteriori, when matching, it can produce high number of false mismatches.

A different approach is given by [11] where scale is determined as the entropy extremum of the local descriptors of the interest region being studied. We will explain it a little bit more in next subsection.

We will now insist on the former technique, LoG. Many modern methods rely on this approach. For example, Harris-Laplace detector, Hessian-Laplace detector and their affine versions and also the Difference-of-Gaussian (DoG), used by SIFT, [18], rely on a LoG strategy.

In order to construct the scale-space point detector, the image is convolved by Gaussian kernels as follows:

Let $I(x,y)$ be our image and $g(x,y,\sigma) = \frac{1}{2\pi\cdot\sigma^2}\cdot \exp^{-\frac{x^2+y^2}{2\sigma^2}}$, the Gaussian kernel. We define the scale space function as:

$$L(x,y,\sigma) = g(x,y,\sigma) * I(x,y) \quad \text{where } \sigma \text{ is related to the image scale.}$$

Then, the Laplacian operator, $\bigtriangledown^2 L(x,y,\sigma) = L_{xx} + L_{yy}$, is applied. Local maxima (minima) of the Laplacian are related to dark (bright) blobs of approximate size $\sqrt{\sigma}$ in the original image. Therefore, a σ sampling is needed and that is where multi-scale appears.

Local extrema are searched in the domain $D = (x,y,\sigma)$, such that the pixel values are not only compared to its eight neighbors at the same convolved image but they are also compared to its nine neighbors obtained at the immediate superior scale level and nine more of the immediate inferior level. That makes a total of 26 comparisons for pixel (at right on figure 2.5).

If we want the process to operate correctly we have to normalize the Laplacian operator by the scale parameter because this operator is sensitive to the variance of the Gaussian kernel, that is $\bar{\bigtriangledown}^2 L(x,y,\sigma) = \sigma\cdot(L_{xx}+L_{yy})$. So, the final objective is to characterize the set of local extrema:

$$S = \{(\hat{x},\hat{y},\hat{\sigma}) : (\hat{x},\hat{y},\hat{\sigma}) = \underset{(x,y,\sigma)\in[\hat{x}-1,\hat{x}+1]\times[\hat{y}-1,\hat{y}+1]\times[\hat{\sigma}-1,\hat{\sigma}+1]}{\text{argmax}} \bar{\bigtriangledown}^2 L(x,y,\sigma) \text{ or}$$

$$(\hat{x},\hat{y},\hat{\sigma}) = \underset{(x,y,\sigma)\in[\hat{x}-1,\hat{x}+1]\times[\hat{y}-1,\hat{y}+1]\times[\hat{\sigma}-1,\hat{\sigma}+1]}{\text{argmin}} \bar{\bigtriangledown}^2 L(x,y,\sigma)\}$$

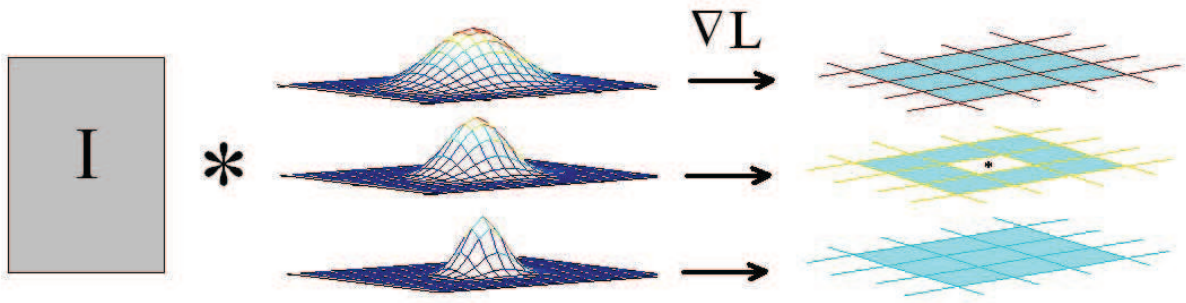

Figure 2.5: At left, image. At center, Gaussians with different scales with whom image is convolved. At right, after having applied normalized Laplacian operator, extrema search over both scale and space is done (the 26 neighbors are colored in blue)

Equivalently, absolute value can be taken and, then, the local extrema set is:

$$S = \left\{ (\hat{x}, \hat{y}, \hat{\sigma}) : (\hat{x}, \hat{y}, \hat{\sigma}) = \underset{(x,y,\sigma)\in[\hat{x}-1,\hat{x}+1]\times[\hat{y}-1,\hat{y}+1]\times[\hat{\sigma}-1,\hat{\sigma}+1]}{\operatorname{argmax}} \left| \bar{\nabla}^2 L(x,y,\sigma) \right| \right\}$$

Figure 2.5 helps understanding the process explained above.

2.1.2 Affine Invariant Detectors

Affinity transformations are those which can be written as a linear bijective transformation $\mathbf{y} = \Phi(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, where $\mathbf{A}$ is an invertible matrix and $\mathbf{b}$ is a translation vector. Delimiting regions around interest points that are affine invariant makes detectors robust to viewpoint changes. Viewpoint changes can't be sufficiently well modeled by affinities since projective geometry lays out affine geometry limits. Yet, local features are sufficiently small so that affinity adaptation is usually a good approximation for the viewpoint transformation.

For this purpose, it is clear that regions can not be delimited by a square or a circumference because these forms are not affine invariant: an affine transformation of a square becomes a parallelogram while circumference transforms to an ellipse. The most typical is to use an adaptative ellipse to define the interest region; then, when a comparison is to be made, ellipses can be normalized to a circumference so that comparison is easier to execute.

Moreover, and as usual, the resultant circumferences can be aligned according to orientation and they also can be intensity normalized. It hadn't yet been commented but a typical procedure along most of the algorithms is to normalize intensities values in order to adapt to *photometric changes*. These changes are modeled as an affinity in the intensity values. That is, $\mathbf{I}' = \mathbf{m}\cdot\mathbf{I}+\mathbf{t}$, where $\mathbf{m} = \text{diag}(m_R, m_G, m_B)$ and $\mathbf{t}$ is a translation vector, $\mathbf{t} = (t_R, t_G, t_B)$ and R, G, B refer to red, green and blue colors respectively. Also, very often, color dimensionality is reduced and detection is made in a grayscale. In order to obtain gray intensity from an RGB data

image, some conversion models are used. For example, Matlab standard conversion from RGB to grayscale gives $\mathbf{I}_{gray} = 0.2989 \cdot \mathbf{I}_{red} + 0.5870 \cdot \mathbf{I}_{green} + 0.1140 \cdot \mathbf{I}_{blue}$.

[20] provides an exhaustive discussion and comparison between these affine invariant detectors. We include a brief description of some of the most cited methods below as the appropriate end for a state of the art in local detectors. Also, in Table 2.2, we give some comments about efficiency for these methods.

Detectors	**Complexity**	**Comments**
Harris-Affine	$\mathcal{O}(n)$	n = #pixels. For Harris detector.
	$\mathcal{O}((m+k)p)$	p = #initial points, m = #preselected scales, k = #iterations. For the LoG and shape adaptation.
Edge Based Regions (EBR)	$\mathcal{O}(n)$	n = #pixels. For Harris detector.
	$\mathcal{O}(pd)$	p = #corners, d = average number of edges per corner. To construct the regions.
Intensity Based Regions (IBR)	$\mathcal{O}(n)$	n = #pixels. For local intensity extrema detector.
	$\mathcal{O}(p)$	p = #intensity extrema. To construct the regions.
Maximally Stable Extremal Regions (MSER)	$\mathcal{O}(n(1+\log(\log(n))))$	n = #pixels. To find local extrema.
Salient	$\mathcal{O}(nl)$	n = #pixels, l = #ellipses investigated per pixel. For local intensity extrema detector.
	$\mathcal{O}(p)$	p = #extrema find in the previous step. To construct the regions.

Table 2.2: Complexity and required computation time for affine detectors

- ***Harris-Affine Detector [22]***

The main ideas of this affine regions detector are:

1. *Multi-scale Harris step.*
   First of all, interest points are detected by a Harris detector specially adapted to be scale invariant. The second moment matrix, $M(x,y)$, defining the local structure used by Harris (recall Equation (2.1)), is modified so that scale-space kernels appear:

$$\mu(x,y,\sigma_I,\sigma_D) = \sigma_D^2 \cdot g(x,y,\sigma_I) * \begin{pmatrix} L_x^2(x,y,\sigma_D) & L_xL_y(x,y,\sigma_D) \\ L_yL_x(x,y,\sigma_D) & L_y^2(x,y,\sigma_D) \end{pmatrix} \quad \text{with}$$

$$L_x(x,y,\sigma_D) = g(x,y,\sigma_D) * I_x(x,y),\ I_y(x,y,\sigma_D) = g(x,y,\sigma_D)) * I_y(x,y),$$

   recall that $\mathbf{I}_x, \mathbf{I}_y$ were the partial derivatives of $\mathbf{I}$

   Note that local images derivatives, $\mathbf{L}_x$ and $\mathbf{L}_y$, are computed by convolving the partial derivatives, $\mathbf{I}_x$, and $\mathbf{I}_y$, with a Gaussian kernel of variation σ_D (which is called the differentiation scale). Derivatives are then averaged in the neighborhood of each point using a

Gaussian window of variance (smoothing) σ_I (which we refer as integration scale). Factor σ_D^2 is for normalization.
Local extrema are those performing Harris evaluation function, $F(x, y)$ (recall Equation (2.2)) at a set of pre-selected levels $\sigma_I \in \{\sigma_n\}_n = \{\xi^n \sigma_0\}_n$ and $\sigma_D = s \cdot \sigma_I$. Typically, $\xi = \sqrt{2}$ and $s = 0.7$. That is, we find pixel position (x, y) such that $I(x, y)$ is an extremum once having fixed a level (σ_I, σ_D). Observe that values comparison is only made in space. Briefly, we keep the set:

$$\text{Extrema set, } S = \{(\hat{x}, \hat{y}, \sigma_I) : F(\hat{x}, \hat{y}, \sigma_I) = \max_{(x,y,\sigma_I) \in [\hat{x}-1,\hat{x}+1] \times [\hat{y}-1,\hat{y}+1] \times \sigma_I} F(x, y, \sigma_I)$$
$$\text{and } F(\hat{x}, \hat{y}, \sigma_I) \geq T,\ T \text{ threshold},\ \sigma_I \in \{\xi^n \sigma_0\}_n\}$$

2. *Refinement of the extrema set.*
The next step consists in finding a precise *characteristic* scale for the feature. *Characteristic* can be understood as the best length estimation of the feature structure. Laplacian of Gaussian is used.
For each point found in the multi-scale Harris step, an iterative algorithm is applied for simultaneously selecting point location and scale. Let $(\mathbf{x}^{(0)}, \sigma_I^{(0)})$ be a point of the extrema set S. We note $\mathbf{x} := (x, y)$. The algorithm below is applied at every iteration. That is, given $(\mathbf{x}^{(k)}, \sigma_I^{(k)})$, we do:

   (a) Find local extremum over scale with the LoG technique, as explained in Section 2.1.1, for the point $\mathbf{x}^{(k)}$, where (k) indicates the iteration number. Scale ranges are limited to ${\sigma_I}^{(k+1)} = t \cdot {\sigma_I}^{(k)}$, $t \in [0.7, \ldots, 1.4]$. Observe that, when taking this range, the gap between the consecutive, previous and posterior, scale-spaces in multi-Harris detection is fulfilled (recall in first step we had $\xi = \sqrt{2},\ \sigma_n = \xi^n \sigma_0$). Therefore, this step consists in introducing more precision for scale (initial σ_I turns into $\sigma_I^{(k+1)}$) while location remains the same. If not found any extremum, reject point; otherwise point $\left(\mathbf{x}^{(k)}, \sigma_I^{(k+1)}\right)$ is obtained.

   (b) Maximize Harris evaluation function, $F(x, y)$, in the region so point $(\mathbf{x}^{(k+1)}, \sigma_I^{(k+1)})$ is obtained. Observe that, here, only location is modified, not scale.

   (c) Repeat process until $\left(\mathbf{x}^{(k+1)}, \sigma_I^{(k+1)}\right) = \left(\mathbf{x}^{(k)}, \sigma_I^{(k)}\right)$ (stop condition).

Up to now, we have constructed a scale invariant detector which is called Harris-Laplace detector. We next estimate the affine shape of the region to obtain the affine invariant detector.

3. *Affinity invariance.*
To deal with affinity transformations we can not consider an uniform scale change in every direction. Scales are allowed to vary independently along orthogonal directions. With this purpose the Gaussian kernels of the LoG are replaced by multivariate Gaussian kernels:

$$g(x, y, \mathbf{\Sigma}) \sim \mathcal{N}((0, 0), \mathbf{\Sigma}), \quad g(x, y, \mathbf{\Sigma}) = \frac{1}{2\pi\sqrt{|\mathbf{\Sigma}|}} \exp\big(-(x, y)\mathbf{\Sigma}^{-1}(x, y)^T\big)$$

And the second moment matrix for the multi-scale Harris detector is:

$$\mu(x, y, \mathbf{\Sigma}_I, \mathbf{\Sigma}_D) = \det(\mathbf{\Sigma}_D) \cdot g(x, y, \mathbf{\Sigma}_I) * \begin{pmatrix} L_x^2(x, y, \mathbf{\Sigma}_D) & L_x L_y(x, y, \mathbf{\Sigma}_D) \\ L_y L_x(x, y, \mathbf{\Sigma}_D) & L_y^2(x, y, \mathbf{\Sigma}_D) \end{pmatrix},$$

where $\mathbf{\Sigma}_D$ has also been chosen as $\mathbf{\Sigma}_D = s\mathbf{\Sigma}_I$ (as case $\sigma_D = s\sigma_I$). Notice that using multivariate Gaussian distributions, ellipses will automatically become the geometrical object delimiting the region.
The success of this method also relies on the fact that relation between second moment matrix of points $\mathbf{s} = (s_x, s_y)$ and $\mathbf{t} = (t_x, t_y)$, where $\mathbf{t} = \Phi(\mathbf{s}) = \mathbf{A}\mathbf{s} + \mathbf{b}$, Φ affinity, is as simple as:

$$\mu(s_x, s_y, \mathbf{\Sigma}_I^{\mathbf{s}}, \mathbf{\Sigma}_D^{\mathbf{s}}) = \mathbf{A}^T \mu(t_x, t_y, \mathbf{\Sigma}_I^{\mathbf{t}}, \mathbf{\Sigma}_D^{\mathbf{t}}) \mathbf{A}$$

Also, a similar relation is found between covariance matrices: $\mathbf{\Sigma}_D^{\mathbf{t}} = A\mathbf{\Sigma}_D^{\mathbf{s}} A^T$.
It also can be shown that, assuming $\mathbf{\Sigma}_a^b = \sigma_a^b \cdot \boldsymbol{\mu}_b^{-1}$, where $a \in \{I, D\}$, $b \in \{\mathbf{s}, \mathbf{t}\}$ and $\boldsymbol{\mu}_b$ denotes $\mu(b_x, b_y, \sigma_I^b, \sigma_D^b)$, then $\mathbf{A}$ can be expressed as:

$$\mathbf{A} = \mu_{\mathbf{t}}^{-\frac{1}{2}} \mathbf{R}_\alpha \mu_{\mathbf{s}}^{-\frac{1}{2}} \quad \text{where } \mathbf{R}_\alpha \text{ is only a rotation matrix of a certain angle } \alpha.$$

Then, when normalizing $\mathbf{s'} = \boldsymbol{\mu}_{\mathbf{s}}^{-\frac{1}{2}}\mathbf{s}$ and $\mathbf{t'} = \boldsymbol{\mu}_{\mathbf{t}}^{-\frac{1}{2}}\mathbf{t}$, relation between transformed points is only $\mathbf{s'} = \mathbf{R}_\alpha \mathbf{t'}$. Rotation $\mathbf{R}_\alpha$ is easy to estimate by taking the mean of gradient orientations in the transformed image (the $\mathbf{s'}$ domain).
Summarizing, Harris-Affine detectors have some nice properties which lead to an easy comparison between similar images by only searching for a rotation once they have been normalized.
See Figure 2.6 for further comprehension.
Finally σ_D is also refined according to a convergence criterion based on ratio $\mathcal{Q} = \frac{\lambda_{\min}}{\lambda_{\max}}$, where $\lambda_{\min},\ \lambda_{\max}$ are the eigenvalues of matrix $\mu(x, y, \mathbf{\Sigma}_I, \mathbf{\Sigma}_D)$

As a final comment, it is worth to mention that the algorithm benefits from the fact that, after affine normalization, the region becomes to be circular, gaussian kernels are used in the transformed domain, instead of using multivariate gaussian kernels in the image domain. It is in the transformed domain where scales are found in order to maximize LoG (σ_I) or $\mathcal{Q}$ (σ_D). For the algorithm description, see Table 2.3.

- ***Edge-Based Region Detector, EBR [32].***

Edge-Based Region detectors (EBR) rely on the presence of geometric entities such as corners and edges.

More specifically, this detector delimits the affine region as a parallelogram based on the edges that define a corner, which acts as the keypoint region. It is the only method among those described in this subsection which does not define an elliptical region. Notice that, a parallelogram is also an affine invariant geometrical object.

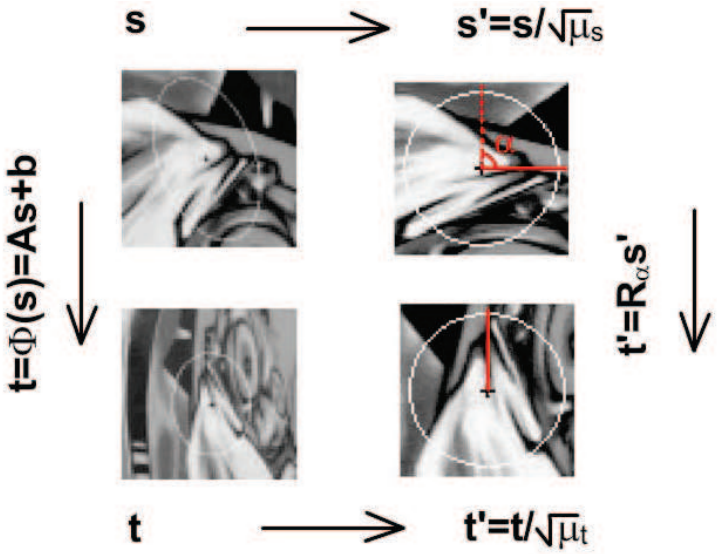

Figure 2.6: Illustration of the affine normalization. At left, two images, **s** and **t**, related by an affinity, Φ. In both, same interest point and region are shown. At right, we can see the respective transformed images, **s'** and **t'**. Observe the images in the transformed domain only differ on a rotation of angle α.

Let $\mathbf{p} = (x, y)$ be a Harris corner point on to an edge. Let $\mathbf{p}_1(s_1)$ and $\mathbf{p}_2(s_2)$ be the edge curves moving away from the corner in both directions (that is $\mathbf{p}_1(0) = \mathbf{p}_2(0) = \mathbf{p}$). Recall edges can be found by applying the Canny edge detector.

Then,

$$l_i \doteq \int |\det([\mathbf{p}_i'(s_i), \quad \mathbf{p} - \mathbf{p}_i(s_i)])| ds_i,$$

where $i \in \{1, 2\}$, and $\mathbf{p}_i'$ is curve derivative. Condition $l_1 = l_2 \doteq l$ is imposed for an invariance criterion. This defines a one parameter family of parallelograms, $\{\Omega(l)\}_l$, defined by sides $\overrightarrow{\mathbf{pp}}_1(l)$ and $\overrightarrow{\mathbf{pp}}_2(l)$.

The next step is to choose the best l, that is l such that $\Omega(l)$ gives the best performance, based on some photometric quantities evaluated over the parallelogram. These photometric functions consist on combinations of $M_{pq}^n = \int_{\Omega(l)} I^n(x, y) x^p y^q dx dy$, the nth order and $(p + q)$th degree moment. It can be proved that looking for local minima of these functions yields more balanced regions and also invariant regions with respect to photometric and geometric changes. Here, balance refers to having gravity center of the parallelogram (with intensity ponderation) close to one of the diagonals. See [32] for more specific information.

It is worth to mention that when edges are not curves but straight lines, $\mathbf{p}_i'(s_i)$ is parallel to $\mathbf{p} - \mathbf{p}_i(s_i) \quad \forall s_i$. Therefore $l \equiv 0$ and what has been discussed can not be applied. The solution proposed is to consider the "low valleys" of two of the photometric functions when considering the region as $\Omega(s_1, s_2)$ and take intersections of both valleys at the s_1, s_2-space. Point obtained $(\hat{s}_1, \hat{s}_2)$ will define the parallelogram region, which will also maintain the photometric and geometric invariance. The obtention of the invariant region Ω in both cases (whether edges

Given an interest point, obtained through multi-scale Harris detector, $\mathbf{x}_\omega^{(0)}$, do:
[1] Define $\mathbf{U}^{(0)} = \mathbf{Id}$ of size the patch centered at $\mathbf{x}_\omega^{(0)}$ ($\mathbf{U}$ is the transformation matrix, $\mathbf{x}_\omega$ refers to the original image while $\mathbf{x}$ refers to the transformed one).
[2] Normalize window $W(\mathbf{x}_\omega) = I(\mathbf{x})$ centered on $\mathbf{U}^{(k-1)}\mathbf{x}_\omega^{(k-1)} = \mathbf{x}^{(k-1)}$
[3] Select integration scale σ_I at point $\mathbf{x}_\omega^{(k-1)}$. As discussed, LoG is used.
[4] Select differentiation scale $\sigma_D = s\sigma_I$, with $s \in [0.5, 0.75]$, maximizing $\mathcal{Q}$.
Calculate $\mu = \mu(\mathbf{x}_\omega^{(k-1)}, \sigma_I, \sigma_D)$
[5] Maximize Harris evaluation function and find extremum $\mathbf{x}_\omega^{(k)}$ nearest to $\mathbf{x}_\omega^{(k-1)}$.
Compute $\mathbf{x}^{(k)}$ location, that is $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + U^{(k-1)} \cdot (\mathbf{x}_\omega^{(k)} - \mathbf{x}_\omega^{(k-1)})$
[6] Calculate $\mathbf{U}^{(k)}$ transformation, $\mathbf{U}^{(k)} = \mu^{\frac{-1}{2}}(\mathbf{x}_\omega^{(k)}, \sigma_I, \sigma_D) \cdot \mathbf{U}^{(k-1)}$.
Normalize $\mathbf{U}^{(k)}$ so that $\lambda_{\max}(\mathbf{U}^{(k)}) = 1$
[7] Go to step [2] if $\left(1 - \mathcal{Q}(\mathbf{U}^{(k)})\right) \geq \varepsilon$

Table 2.3: Harris-Affine algorithm

are curved or straight) is shown in Figure 2.7.

Finally, EBR detector is proposed in [32] and combined with a descriptor of moments invariants called *Generalized Color Moments*. For matching between images, Mahalanobis distance is used as measure. We will explain it in Section 2.3.

- ***Intensity-Based Region Detector, IBR [32].***

Although EBR and IBR are proposed in the same paper, their nature is completely different. For example, while EBR is based in corner detection, IBR is indeed a blob detector. The first step of the Intensity-Based method is to extract local extrema of intensity among image pixels as interest points.

Since images are usually pre-smoothed, location accuracy of extrema isn't very exact. This is not important as long as detected regions are chosen in a way that do not rely on accuracy of the interest point location. This is also the case of IBR.

IBR region of interest is grown starting from the interest point and spreading in a radial way, independently in every direction, until a certain function evaluated along the ray attains an extremum. So, region is pretty robust to location inaccuracy.

The function evaluated along each ray is:

$$f_I(t) = \frac{|I(t) - I(0)|}{\max\left(\frac{\int |I(t)-I(0)|dt}{t}, d\right)},$$

where t is the ray parametrization ($t = 0$ is the local intensity keypoint) and d is a threshold chosen to avoid indetermination. Extrema of the function in each ray are affine and photometric

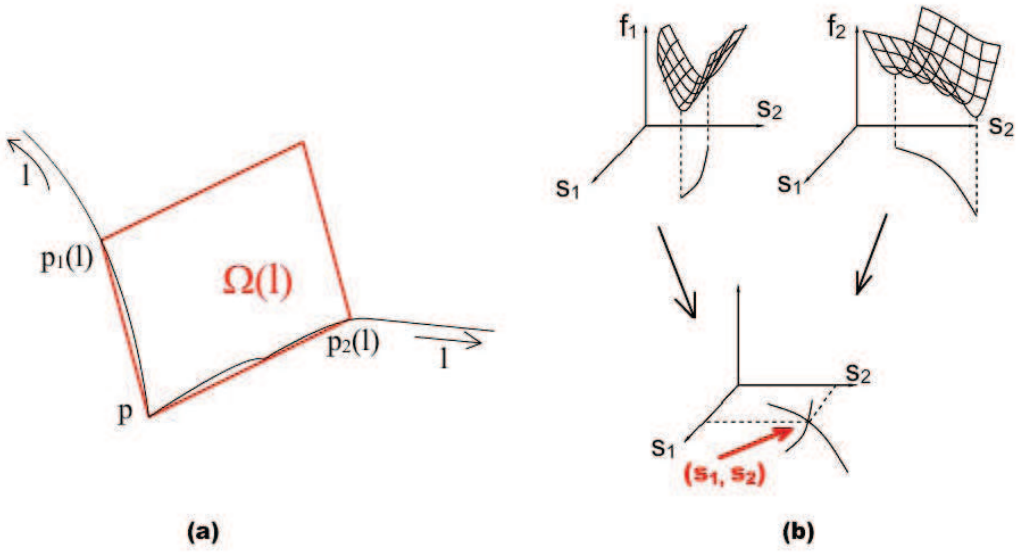

Figure 2.7: EBR detector. (a) region obtained in presence of curved edges; (b) the process for straight edges: graphs corresponding to the evaluation of the two defined functions, the characterization of the optimal values (point (s_1, s_2) in red) and the region obtained.

invariant.

Next, extrema found at each ray are linked and a region is defined. When more than a local extremum can be found along the same ray, the selection of the extremum is done imposing continuity constraints: the closest extremum to extrema in the neighboring rays is chosen.

The defined region is substituted by the best ellipse fitting in the region. Finally, ellipse size is doubled and that is the final invariant region taken. Doubling the size leads to a more distinctive region while the risk of breaking planar assumption increases. Region created is illustrated at Figure 2.8.

Descriptor and also the matching distance used by IBR in [32] are the same as EBR.

- ***Maximally Stable Extremal Region Detector, MSER [19].***

The conceptual idea of how different regions are chosen is the following: Consider the image embedded as a 2D surface in 3D space when considering intensity, I, the third coordinate. Then, consider also the set $I_t = \{(x, y, I)$ such that $I(x, y) \leq t\}$. Starting from $t = 0$, as t increases, some regions R_t begin to appear in I_t. R_t are connected regions considering components in (x, y) coordinates. While increasing t more and more, these regions begin to merge until, for I_1, we have all the image surface included in the set. The set of regions corresponding to local intensity minima is defined as all the connected components of I_t at each threshold t. Figure 2.9 depicts the method exposed.

Finally, the set is reduced by a region stability criterion which can be found in [33].

MSER is presented in [19] as the detector for a robust wide-baseline algorithm. Authors un-

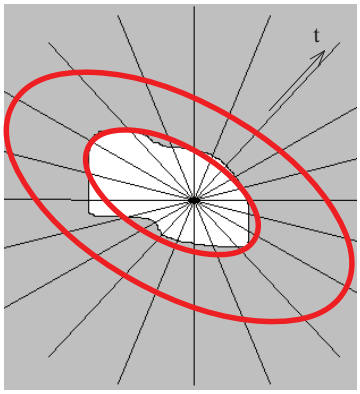

Figure 2.8: Region defined by the IBR detector. Observe first region defined, ellipse fitting and the final ellipse of size twice of the former one.

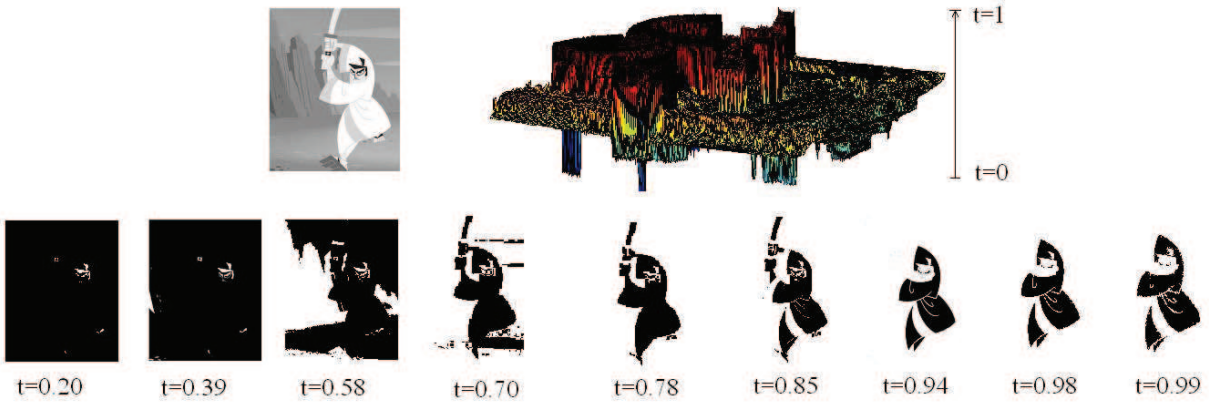

Figure 2.9: MSER are, before the maximal stability criterion, those obtained when intersecting the image embedded with the plane $I = t$ and considering regions whose intensities are below (above) the intersection for local minima (maxima) regions. At top left, original image, at right image as a 3D graph with colors proportional to intensity values, which move in interval $[0, 1]$). At bottom, existent regions (MSER) at different levels are seen in white while the non-existent in black.

derline its computational efficiency since the enumeration of the extremal set is $\mathcal{O}(n \cdot \log(\log(n)))$, where n is the number of image pixels.

- ***Salient Region Detector [11].***

Salient region detector approach is quite different from the methods presented above. Interest points are considered to be the ones that exhibit the most unpredictability not only along their local attributes but also along scale changes. The method consists on the following three steps:

1. Calculation of Shannon's entropy of local image attributes over a range of scales, s. A lot

of functions with which calculate the entropy function can be chosen. Here, pixel intensity is chosen. Therefore, in order to estimate the probability distribution function $p(I, \mathbf{x}, s)$, an histogram of intensity values is calculated within a circle Ω of radius s centered at point $\mathbf{x}$. Entropy function is then:

$$\mathcal{H}_\Omega(\mathbf{x}, s) = -\sum_\Omega p(I, \mathbf{x}, s) \log_2(p(I, \mathbf{x}, s))$$

This step cares about feature-space predictability.

2. Selection of scales $\hat{s}$ such that $\mathcal{H}_\Omega(\hat{s})$ is a maximum. $\hat{s}$ are points that belong to the set:

$$\left\{ s \;\middle|\; \frac{\partial}{\partial s}\mathcal{H}_\Omega(\mathbf{x}, s) = 0,\ \det\left(\frac{\partial^2}{\partial s^2}\mathcal{H}_\Omega(\mathbf{x}, s) < 0\right) \right\}$$

3. Calculation of the change experimented by the probability distribution function $p(I, \mathbf{x}, s)$ when varying s. The resulting measurement function is:

$$\mathcal{W}_\Omega(s) = s \int \left|\frac{\partial}{\partial s} p(I, \mathbf{x}, s)\right| dI$$

This step deals with inter-scale unpredictability. For instance, in a noisy image, pixel values are highly unpredictable at any scale (entropy is close to 1 for all s). However, over scale, statistics are stationary. This translates into a low value for $\mathcal{W}_\Omega$ measurement.

The final function measure for salient regions detection is $\mathcal{Y}_\Omega(\mathbf{x}, \hat{s}) \triangleq \mathcal{H}_\Omega(\mathbf{x}, s) \cdot \mathcal{W}_\Omega(s)$, which is a trade-off between space and scale intensity unpredictability in a concrete region. An example of the effects of both factors in the salient detector is shown in Figure 2.10.

Until now, the detected region has been found in a scale invariant way. To obtain an affine invariant detector, s is expanded in order to able to describe any ellipse centered at the pixel location. Then s turns into $\mathbf{s} = (s, \rho, \phi)$, where ρ is the axis ratio of the ellipse and ϕ is its orientation.

2.2 Local Descriptors

In the previous section, we have shown the most commonly used methods for detecting interest points and also delimiting their characteristic region. In this section, our aim is to show the methods used to represent these regions, which are called descriptors. Descriptors are vectors of a certain dimension where each component of the vector gives information about certain parameter measured in the detected region that descriptor is parameterizing.

There are many techniques for describing local image regions. The simplest descriptor is a vector of image pixels. Then, when comparing similarity between region descriptors, cross-correlation would be the best tool. Nevertheless, cross-correlation of image patches is very sensitive to changes (scale or affine) and also requires from a high computational cost. Since this first and natural way of feature matching, lots of descriptors and associated similarity measures have been proposed.

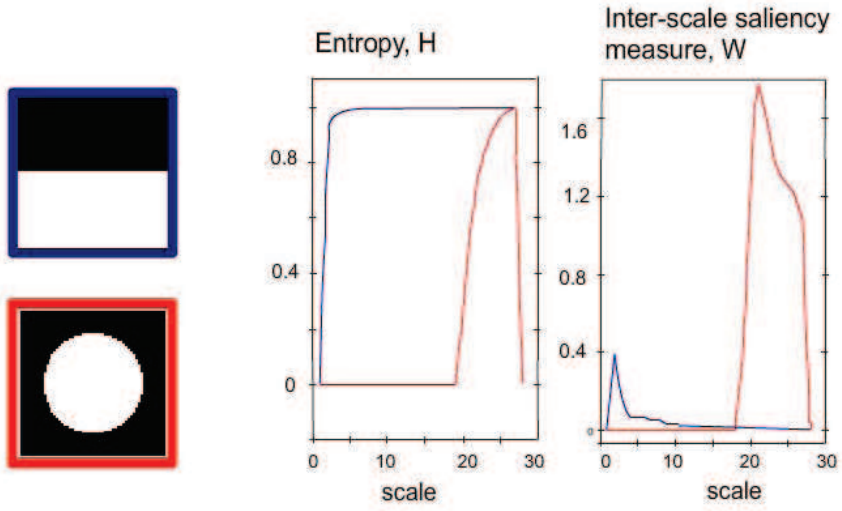

Figure 2.10: The behavior and differences between entropy and inter-scale saliency measure are clearly shown in the example above. While both regions achieve an entropy maximum (it is easy to appreciate it since the quantity of black pixels is the same as the one of white ones at a certain scale), by changing scale it is clear that changes on the intensity distribution of the red framed patch are greater than the ones of the blue framed. For a better illustration, s is here considered as an unique parameter indicating the radius of a circular window

First, the most relevant descriptors are schematically presented in Section 2.2.1.

Next, in Section 2.2.2 we explain one of the most used descriptors, SIFT, and also discuss its variants. SIFT shows to be a great descriptor as it has been proved in [21], where descriptors are evaluated under different situations for matching and recognition: affine transformations, scale changes, rotations, blur, JPEG compression and illumination changes. In [21], SIFT and SIFT-like methods give the best performance.

2.2.1 Descriptors' Overview

- Distribution Based Descriptors
  - $SIFT$
    - $GLOH$
    - $DAISY$
  - $SURF$[2]
  - $Histogram\ of\ Orientated\ Gradients$[6]
  - $Shape\ Context$
- Spatial-Frequency Techniques
  - $Gabor\ filters$
  - $Haar\ wavelets$
- Differential Descriptors,
  - $Steerable\ filters$
  - $Complex\ filters$
- Other Descriptors
  - $Cross-correlation$
  - $Generalized\ Moment\ Invariants$
  - $PCA-SIFT$

2.2.2 SIFT-like Descriptors

Scale Invariant Feature Transform, SIFT [18]

In the previous sections we have presented techniques to assign an image location, scale and orientation to each point. Remind that the detector used by SIFT is a slight modification of the LoG scale invariant detector. For further details on SIFT detector, see Section 3.3.1.

In order to compute the descriptor, image gradient magnitudes and orientations are sampled in a square grid around the point. Sampling is done at the scale at which the keypoint was detected. Patch and gradient orientations are rotated relative to the interest point orientation so that orientation invariance is obtained.

A Gaussian weighting function with variance one and a half of the descriptor size grid ($s \times s$, s the scale) is used so that gradient magnitudes are weighted according to their proximity to the keypoint. This provides certain robustness since small changes in window position do not affect much to descriptor.

Samples are grouped in subregions and each subregion is related to an orientation histogram. Histograms bins are a range of degrees. Samples within the subregion are counted and classified according to their gradient orientation. Gradient magnitudes, once adjusted by the values of the previous Gaussian window, act as weight of the sample importance to the histogram. To avoid boundary effects between subregions, samples effect in the histogram is also weighted by the inverse of the distance from the sample location to the subregion center (assuming subregion is of size $D \times D$ and has center (u, v), weight of samples located at pixel (u', v') in the subregion, has factor proportional to $(\frac{D-1}{2} - |u - u'|) \cdot (\frac{D-1}{2} - |v - v'|)$).

Subregions are created around the keypoint: first four subregions are located in a way that each ones has the interest point as vertex and occupies one of the quadrants. The next twelve

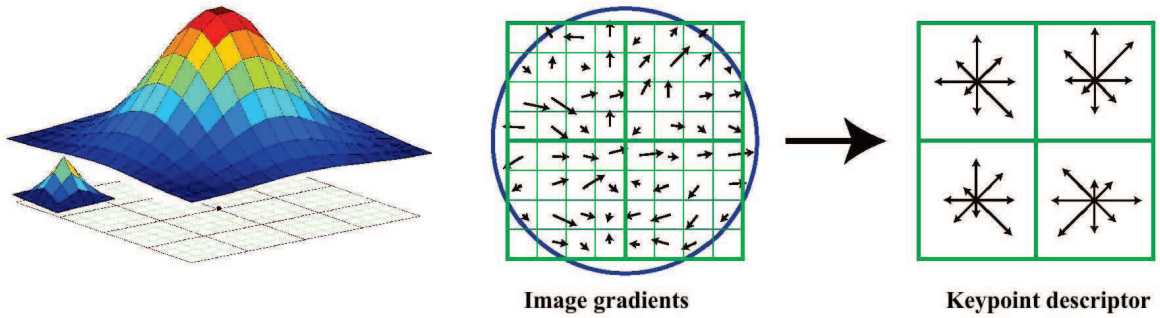

Figure 2.11: At left, SIFT grid for $n = 4$. See there is a total of 4×4 subregions surrounding the interest point. As explained, first, a circular window is applied to gradient magnitudes in the whole region, and then, histograms take place at each subregion where gradient magnitudes are also modified. This time, modification consists on convoluting by a triangular function in each direction so center pixel in the subregion contains the sum of the gradient orientations weighted by distance to the subregion center. Both the circular window and the triangular one are shown. Second and third figures show the same process in another perspective. Here, $n = 2$, there is a total of 4 regions where to construct the histogram. Gradients of the samples and the Gaussian window are shown at center. At right, histograms for the 4 subregions are shown. Observe histograms have eight bins ($r = 8$)

subregions are placed adjoiningly the first four and so on until the desired quantity of subregions has been placed. Figure 2.11 shows the distribution described.

Descriptor is obtained concatenating the values of each bin of each histogram. Therefore, its complexity depends on the number of subregions, $N = n^2$, where n is the number of subregions along an axis (X or Y), and also on the number of bins, r. Values n and r are set to 4 and 8 respectively since the best performance is achieved with a 4×4 array of histograms with eight orientation bins each one. Therefore, vector descriptor dimension is $n^2 \cdot r = 128$. Also, sampling set in each subregion is a 16×16 array. Finally, descriptor is normalized to unit length so that illumination changes affect less.

When changing the square grids where histograms are calculated for circular ones, the resulting method is referred to as circular SIFT or symmetric SIFT.

Note that, even though the SIFT detector is scale invariant but not affine invariant, SIFT descriptor performs quite well robustness to viewpoint changes (only a lost of 10% in repeatability when viewpoint changes from 0 to 50 degrees).

Gradient Location and Orientation Histogram, GLOH [21]

This new descriptor is an extension of SIFT descriptor, in which, what changes the most is the limits and shape of the different subregions where histograms are created. SIFT is computed for a log-polar grid with three subregions in radial direction (radii: 6, 11 and 15) and eight in angular direction (the log-polar configuration can be seen at Figure 2.12). The central bin is not divided in angular directions. In total, there are 17 subregions and, as consequence, 17 histograms. Each

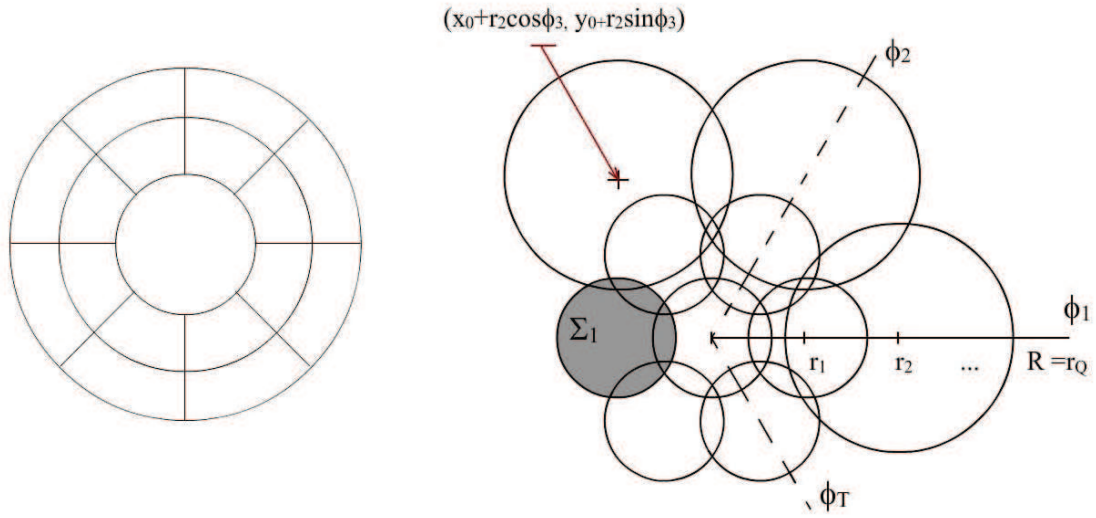

Figure 2.12: At left, the GLOH's log-polar configuration. At right, DAISY's configuration. Here notation introduced can be appreciated.

histogram consists of 16 bins. This yields a vector descriptor of $16 \cdot 17 = 272$ elements. This high dimensionality is reduced through PCA (see 3.4.2) so the descriptor dimension is reduced to, at most, 128.

DAISY [35] [31] [29] [30]

Although not explicitly called 'DAISY', this descriptor was first proposed in [35]. In this article, Brown and Winder carry out a general study on literature descriptors. They divide them in differentiated parts (such as the evaluation function characterizing descriptor values and the region configuration) so they can evaluate each part differently and mix parts from different descriptors. There, DAISY configuration can be found. Nevertheless, descriptor's name comes from [31], where authors underline the efficiency in programming the descriptor.

The method consists, first of all, on calculating image intensity derivatives along certain directions, $\mathbf{I}_\theta$, θ the direction:

$$\mathbf{I}_\theta = \left(\frac{\partial \mathbf{I}}{\partial \theta}\right)^+, \qquad \text{where } \mathbf{I} \text{ is the image, } \theta \in \{\theta_1, ..., \theta_H\} \text{ and } (.)^+ \doteq \max(0, .)$$

Second, each directional gradient is convolved with Gaussian kernels, $\mathbf{g}_{\Sigma_i}$ with $i = 1, \ldots, Q$,

$$\mathbf{G}_\theta^{\Sigma_i} = \mathbf{g}_{\Sigma_i} * \mathbf{I}_\theta$$

Notice that, fixed a point (x, y), $\mathbf{G}_\theta^{\Sigma_i}(x, y)$ is the weighted sum of the θ directional gradient around pixel (x, y). In fact, $\mathbf{G}_\theta^{\Sigma_i}(x, y)$ can be thought as doing the same as one of the bins of SIFT

histograms for a subregion centered in (x, y) since it measures θ orientation importance within the region. We then group $\mathbf{h}_{\Sigma_i}(x, y) = (\mathbf{G}_{\theta_1}^{\Sigma_i}(x, y), \ldots, \mathbf{G}_{\theta_H}^{\Sigma_i}(x, y))^T$ and, as usual, normalize $\mathbf{h}$ to unit length.

Next, descriptor $\mathcal{D}(x_0, y_0)$, where (x_0, y_0) is the interest point, is obtained by concatenating vectors resembling to $\mathbf{h}_{\Sigma_i}(x, y)$. Given a radius R, an angular quantization T and a radius quantization Q, the set $\mathcal{S}$ of $((x, y), \Sigma)$ points is:

$$\mathcal{S} = \mathcal{S}_1 \bigcup \{((x_0, y_0), \Sigma_1)\}, \quad \text{where}$$

$$\mathcal{S}_1 = \left\{ ((x, y), \Sigma) = ((r_i \cdot \cos\phi_k + x_0,\ r_i \cdot \sin\phi_k + y_0), \Sigma_i),\ r_i = \frac{(i+1)R}{Q},\ \phi_k = \frac{2\pi k}{T} \right.$$
$$\left. \text{and } \Sigma_i \text{ arbitrarily chosen such that } \Sigma_{i-1} < \Sigma_i < \Sigma_{i+1},\ \forall i = 1, \ldots, Q,\ k = 1, \ldots, T \right\}$$

This set of points characterize the DAISY configuration, as shown in Figure 2.12. Summarizing, the descriptor is:

$$\begin{aligned}\mathcal{D}(x_0, y_0) &= (\mathbf{h}_\Sigma(x, y)), ((x, y), \Sigma) \in \mathcal{S}) \\ &= \big(\mathbf{h}_{\Sigma_1}(x_0, y_0), \mathbf{h}_{\Sigma_1}(x_0 + r_1 \cos\phi_1, y_0 + r_1 \sin\phi_1), \ldots, \mathbf{h}_{\Sigma_1}(x_0 + r_1 \cos\phi_T, y_0 + r_1 \sin\phi_T), \\ &\quad \mathbf{h}_{\Sigma_2}(x_0 + r_2 \cos\phi_1, y_0 + r_2 \sin\phi_1), \ldots, \mathbf{h}_{\Sigma_Q}(x_0 + r_Q \cos\phi_T, y_0 + r_Q \sin\phi_T)\big)\end{aligned}$$

Notice that when the center of subregion associated to an histogram is far from the keypoint, gaussian is more smoothed and region considered is bigger. Notice also that subregions overlap, this favors robustness and soft changes between neighbor histograms.

The major differences with SIFT are that:

- The weighting of the samples in a subregion is done through a Gaussian kernel instead of weighting by convolving by a triangular window in both direction X and Y (this is what SIFT does when weighting is inverse proportional to distance to subregion center) and sampling set is a dense set instead of a discrete set within the region,
- The "daisy" configuration of subregions around interest point.
- And mostly, the efficiency.

PCA-SIFT [12]

The only similarities between PCA-SIFT and SIFT is that both are computed over the same image region (they use same DoG detector). PCA-SIFT descriptor works as follows.

On the one hand, first of all, a $m \times m$ patch centered over the keypoint and at the correct scale given by the detector[3] is extracted and rotated to align its dominant rotation to a prefixed

[3]Extracting a $m \times m$ patch at a given scale means forming the patch by sampling the $m \times m$ points according to scale so that the $m \times m$ patch shows the whole region delimited by scale. Therefore, patches extracted can be very different from the actual $m \times m$ pixels centered at the interest point

and common orientation. Value chosen for m is 41 but could be different. A vector is created concatenating the x and y directional derivatives for the patch. Therefore, vector dimensionality is $2 \cdot 39 \cdot 39 = 3042$. Vector is then normalized to minimize illumination effects.

On the other hand, there has been a previous evaluation over a lot of patches (21000 in [12]). Each patch, processed as previously described, creates a 3042-vector. The covariance matrix of the whole set is calculated and PCA (see 3.4.2) is applied. The top n eigenvectors (that is, the eigenvectors related to the n greater eigenvalues) are selected as a basis of the feature space. Finally, for every new vector, the PCA-SIFT descriptor is the projection into the n-dimensional feature space of the 3042-vector. Typically, $n = 20$. While one could thing higher values for feature space dimensionality to always perform better, in reality, it doesn't happen. That is because, when increasing n, feature space captures more and more distortions not really related to the feature itself (blurring, etc.).

In [12], it is shown that PCA-SIFT performs better than SIFT. It is claimed PCA-SIFT takes profit of SIFT detecting steps so remaining variation is basically due to keypoint identity and perspective distortions. Moreover, when discarding lower components, PCA improves accuracy since unmodeled distortions are rejected (they are written in terms of eigenvectors of lower eigenvalues). It is also remarkable the decrease in descriptor dimensionality: from 128-dimensional SIFT descriptor to 20-dimensional PCA-SIFT descriptor.

2.3 Matching

Here, we present some matching techniques closely related to object classification techniques.

2.3.1 Measures

The metrics used to compare different descriptors can be classified in several classes, such as heuristics distances, non-parametric test statistics, information-theory divergences and ground-distance measures. In the following, we will give some examples of each class.

Heuristic distances

L_p ***distance*** $d_{L_p}(\mathbf{x}, \mathbf{y}) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

Particularly, d_{L_2}, called the Euclidean distance, and d_{L_1}, called the absolute distance, are the most used. The most natural environment where using the L_p distances is when directly comparing two descriptor vectors. Descriptors such as SIFT, PCA-SIFT, GLOH, DAISY, shape context, SURF and spin images use the Euclidean distance.

Mahalanobis distance: Given a point $\mathbf{x}$ and a set of points Y with mean μ and covariance matrix $\mathbf{M}$, the Mahalanobis distance is defined as follows:

$$d_Y(\mathbf{x}) = \sqrt{(\mathbf{x} - \mu)^T \mathbf{M}^{-1} (\mathbf{x} - \mu)}$$

Mahalanobis distance differs from the Euclidean one because the former takes into account correlations of the data set and it is also scale invariant. In fact, when $\mathbf{M} = \mathbf{Id}$, $d_Y(\mathbf{x})$ degenerates

into $d_{L_2}(\mathbf{x}, \mu)$. It is commonly used by complex filters, differential invariants, moment invariant, steerable filters, etc. It is also commonly used in classification problems since distance is a measure point-to-sets.

Pyramid Matching Kernel [7][4][13]:

Before introducing this method, we briefly explain what kernels are [10]. Kernels are functions which measure similarity between their two inputs, $\mathbf{x}$, $\mathbf{y}$ vectors of a certain dimension. Their general expression is $\kappa(\mathbf{x}, \mathbf{y}) =< \phi(\mathbf{x}), \phi(\mathbf{y}) >$, that is an inner product after a transformation, ϕ, to a certain feature space. Some kernel examples are: $\kappa(\mathbf{x}, \mathbf{y}) =< \mathbf{x}, \mathbf{y} >$ (which is equivalent to the L_2 distance), $\kappa(\mathbf{x}, \mathbf{y}) = \exp(- < \mathbf{x}, \mathbf{y} > \cdot\sigma^{-2})$, etc. Methods as Support Vector Machines, kernel PCA or Gaussian Processes rely on these objects to perform the classification. A lot of efforts have been lately put in developing better kernels, specially when data (for example, descriptors) can not naturally fit into an Euclidean space. Here, the kernel presented below is able to even work with vectors of different dimensions ([7]) and also it is very efficient and robust to clutter.

Consider the input space as a set of sets of d-dimensional feature vectors (descriptors)

$$X = \left\{ \mathbf{x} \text{ such that } \mathbf{x} = \{\mathbf{f}^i = [f_1^i, \ldots, f_d^i] \in \mathbb{R}^d\}_{i=1 \div |\mathbf{x}|} \right\}^4$$

bounded by a sphere of dimension D at $\mathbb{R}^d$. Let the feature function be: $\phi(\mathbf{x}) = [H_{-1}(\mathbf{x}), H_0(\mathbf{x}), \ldots, H_L(\mathbf{x})]$, where $L = \lceil \log_2 D \rceil$ and $H_i(\mathbf{x})$ is an histogram formed over data in $\mathbf{x}$ (remember there is a total of $|\mathbf{x}|$ d-vectors in $\mathbf{x}$), containing d-dimensional bins of side length 2^i. $H_i(\mathbf{x})$ dimension is $\left(\frac{D}{2^i\sqrt{d}}\right)^d$. Simplifying, each histogram has bins that double in size, in each direction, the previous one. At the finest level, H_{-1}, bins are small enough to only contain, at most, a vector from $\mathbf{x}$.

Pyramid kernel, κ_Δ, measures similarity by a weighted sum of correspondences found at each level of the histograms pyramid, that is $\kappa_\Delta(\phi(\mathbf{x}), \phi(\mathbf{y})) = \sum_{i=0}^{L} \frac{1}{2^i} N_i$, where $\frac{1}{2^i}$ is the i-th weight and N_i is the number of newly matched pairs at level i, in detail:

$$N_i = \sum_{j=1}^{r_i} \min(H_i^j(\mathbf{x}), H_i^j(\mathbf{y})) - \sum_{j=1}^{r_{i-1}} \min(H_{i-1}^j(\mathbf{x}), H_{i-1}^j(\mathbf{y}))$$

where H_i^j is the j-th bin of the i-th histogram. Notice that weights are inversely proportional to the finesse of the matching, since features' similarity at a finer resolution is more distinctive. The main idea on the pyramid kernel is depicted in Figure 2.13, whereas special cases and detailed explanations can be seen at [7].

Non-parametric statistics

χ^2 ***distance:*** Let $\mathbf{P}, \mathbf{Q}$ be two distributions, P_i, Q_i the respective i-th bin. Then distance is:

$$d_{\chi^2}(\mathbf{P}, \mathbf{Q}) = \frac{\sum_i (P_i - M_i)^2}{M_i}, \quad \text{where } M_i = \frac{P_i + Q_i}{2}$$

[4]Observe that each $\mathbf{x} \in X$ is a set of cardinality $|\mathbf{x}|$ that can be different from the cardinality of another $\mathbf{x}' \in X$

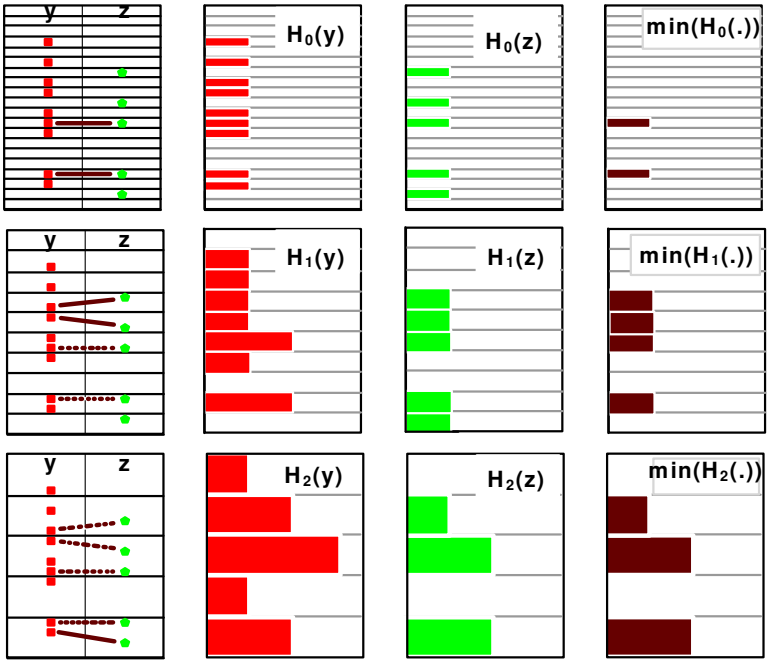

Figure 2.13: At left, point sets, at center, histogram pyramids for each set (first y and next z), at right, intersections histograms. Observe that, in the example, $N_0 = 2$, $N_1 = 2$ and $N_2 = 1$, therefore distances is $\frac{1}{2^0} \cdot N_0 + \frac{1}{2^1} \cdot N_1 + \frac{1}{2^2} \cdot N_2 = 2 + 1 + 0.25 = 3.25$.

$d_{\chi^2}(\mathbf{P}, \mathbf{Q})$ measure how unlikely is that distribution $\mathbf{P}$ was drawn from the population represented by $\mathbf{Q}$ or, in other words, how well samples from distribution $\mathbf{P}$, thought as the experimental distribution, would fit in distribution $\mathbf{Q}$, interpreted as the theoretical distribution being guessed.

Information-theory divergences

Kullback Leibler divergence It quantifies how different two given probability distributions are. Let $\mathbf{P}$ and $\mathbf{Q}$ be two different distributions, then the KL divergence is:

$$d_{KL}(\mathbf{P}||\mathbf{Q}) = \sum_i P_i \log\left(\frac{P_i}{Q_i}\right)$$

For $\mathbf{P}$ and $\mathbf{Q}$ continuous distributions, substitute the sum by an integral. The KL divergence is not a distance since it isn't even symmetric with respect to $\mathbf{P}$ and $\mathbf{Q}$. However, $d_{KL}(\mathbf{P}, \mathbf{Q}) = 0$ if and only if $\mathbf{P} = \mathbf{Q}$. Observe that $d_{KL} = E_{\mathbf{P}}\left[\log\left(\frac{\mathbf{P}}{\mathbf{Q}}\right)\right]$ and also $d_{KL} = \mathcal{H}(\mathbf{P}, \mathbf{Q}) - \mathcal{H}(\mathbf{P})$, where $\mathcal{H}(\mathbf{P}, \mathbf{Q}) = \sum_i (P_i \log Q_i)$ is the cross entropy. These properties make it attractive to use.

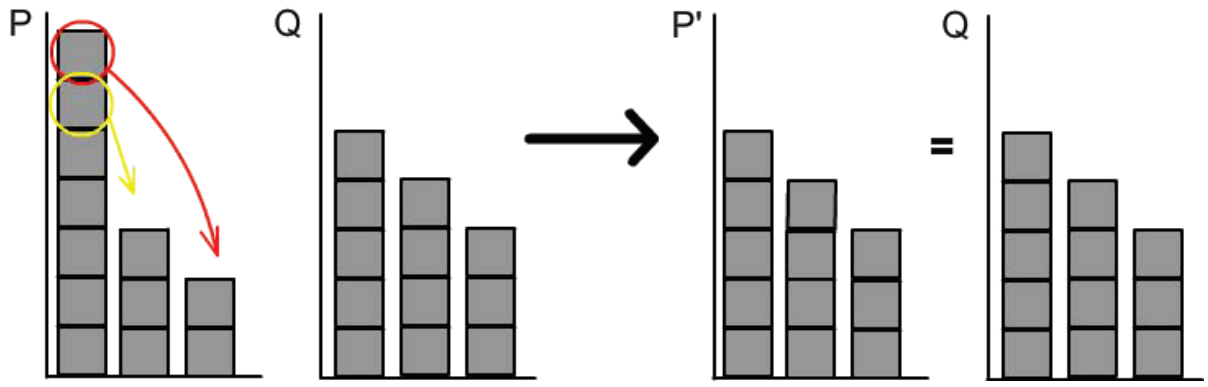

Figure 2.14: At left, **P** and **Q** distributions, at right, Earth Mover's distance concept: amount moved to put on the same level both distributions weighted by distance moved; that is, $d_{EMD} = 1 \cdot 1 + 1 \cdot 2 = 3$

Ground-distances measures

***Earth mover's distance*, [25]** $d_{EMD}(\mathbf{P}, \mathbf{Q}) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}}$

Distribution **P** has m bins, each one weighting P_i, whereas **Q** has n bins Q_j. Let d_{ij} be the distance between the i-th **P** bin and the j-th **Q** bin. f_{ij} represents the flow between the i-th **P** bin and the j-th **Q** bin when minimizing the cost to transform one distribution to another, which is expressed as $Cost(\mathbf{P}, \mathbf{Q}, \{f_{ij}\}_{i,j}) = \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij}$. $\{f_{ij}\}_{i,j}$ is obtained as a result of a linear optimization problem subject to restrictions:

$$(1) f_{ij} \geq 0,\ (2) \sum_{j=1}^{n} f_{ij} \leq P_i,\ (3) \sum_{i=1}^{m} f_{ij} \leq Q_j \text{ and } (4) \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} = \min\left(\sum_{j=1}^{n} Q_j, \sum_{i=1}^{m} P_i\right)$$

(1) restricts flows to movements from **P** to **Q**, (2) forces **P** to not give more than it can whereas (3) forces **Q** to receive what it can hold, and, finally, (4) counts the total amount of earth that can be moved. Process can be seen in Figure 2.14.

2.3.2 Elaborated Matching Methods

As previously said in the beginning of the section, the link between matching, recognition and classification is very close when dealing with local features since features can be thought as classes while a training is done to classify the query patch. Since our algorithm relies on a classification technique, random ferns, we would like to mention some of the existing techniques although we won't explain them in detail.

- ***Support Vector Machines***[10] Training is done maximizing an hyperplane equation which best separates the two considered classes. When a new sample arrives, classification is made according to the sign when evaluating the sample in the hyperplane equation. It

can also be formulated as a multi-way classifier. Lots of variants exist, for example a very efficient one is [36].

- ***Adaboost***. It constructs a strong classifier as a linear combination of weak ones. Algorithm serves to give weights to each weak classifier in the strong one. Some modifications have been added since first definition of the method, [34] is an efficient example.

- ***Random Forests and Ferns***. Since these ones are the techniques we use, we will explain them extensively in Chapter 3.

2.4 Dealing with Deformations

While lots of methods have been proposed to solve image recognition affected by common transformations as scale, viewpoint, blur and photometric change, it has been done very few to deal with transformations related to non-solid objects, that is general deformations. From what we know, in the state-of-the-art only two methods approaching to deformation transformations problem can be found. One is from 2005 and the other is from 2008. We will be explaining them in this section.

Deformation Invariant Image Matching, [17]

In [17], the deformations are treated as homeomorphisms between images. Homeomorphism (ϕ) between deformed images $\mathbf{I}_1$ and $\mathbf{I}_2$ is such that $I_2(u,v) = I_1(\phi(u,v)) = I_1(\phi_1(u,v), \phi_2(u,v)) = I_1(x,y)$, with ϕ continuous with inverse function continuous too. In short, that translates into considering intensity values changing position but not their value. Taking it into account, 2D images are embedded in a 3D space considering variables $(1-\alpha)x, (1-\alpha)y$ and αI, where α is a parameter. α is called the *aspect weight* and, as we will now show, it has a crucial role into the deformation invariance descriptor Ling and Jacobs present.

Consider two images $\mathbf{I}_1, \mathbf{I}_2$, related by an homeomorphism so that the latter is considered a deformation of the former. Let σ_i be the embedded surface in the 3D space from image $\mathbf{I}_i$ affected by the α weights explained above and $\gamma_i(t) = (x_i(t), y_i(t), z_i(t))$ a curve along the surface (let γ_2 be the deformed version curve of γ_1). Curves' lengths when moving from $t = a$ to $t = b$ are:

$$l_1 = \int_a^b \sqrt{(\dot{x}_1^2 + \dot{y}_1^2 + \dot{z}_1^2)}dt = \int_a^b \sqrt{\Big((1-\alpha)^2\dot{x}^2 + (1-\alpha)^2\dot{y}^2 + \alpha^2\dot{I}^2\Big)}dt$$

$$l_2 = \int_a^b \sqrt{(\dot{x}_2^2 + \dot{y}_2^2 + \dot{z}_2^2)}dt = \int_a^b \sqrt{\Big((1-\alpha)^2\dot{u}^2 + (1-\alpha)^2\dot{v}^2 + \alpha^2\dot{I}^2\Big)}dt,$$

where $\dot{(.)} \doteq \frac{\partial(.)}{\partial t}$. From both expressions, it can be easily seen that when $\alpha \to 1$, lengths converge to same value since differences, relying on pixel coordinates, get smaller with this condition. Figure 2.15 shows this behavior.

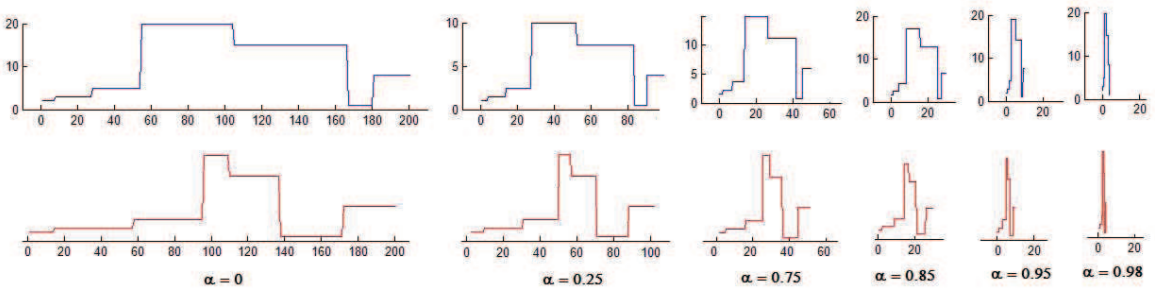

Figure 2.15: Example of the effects of the α aspect weight on a 1D function I(x) instead of the 2D function I(x,y). When α tends to 1, distance between points on both curves (one the deformation of the other) depends more on the intensity and less on the location coordinate. Meanwhile, curves tend to have same shape.

Once explained α's role, the descriptor is created as follows. Given an interest point $\mathbf{p} = (x, y)$, geodesic level curves[5] around the point are calculated, these are curves formed with points at same geodesic distance from $\mathbf{p}$. Notice that, as $\alpha \to 1$, geodesic distance depends more on intensities and less on pixel location and therefore it adapts to deformation. Figure 2.16 shows the geodesic level curves of a concrete point once taken $\alpha = 0.98$.

Then, descriptor consists on a 2D histogram where bins are formed considering a range of intensities and a geodesic distance interval. It is called the GIH (geodesic intensity histogram). Sampling set is obtained sampling geodesic level curves at uniform distance Δ along each curve. Concisely, $\mathbf{H_p}$, histogram characterizing interest point $\mathbf{p}$, is:

$$\mathbf{H_p}(k,m) = \#\left\{\mathbf{q} \in P_\mathbf{p} : I(\mathbf{q}) \in \left[\frac{k-1}{K}, \frac{k}{K}\right], d(\mathbf{q},\mathbf{p}) \in \left[\frac{m-1}{M} d_{max}, \frac{m}{M} d_{max}\right]\right\}$$

where $P_\mathbf{p}$ is the sampling set, $d(\mathbf{q},\mathbf{p})$ is the geodesic distance between $\mathbf{p}$ and $\mathbf{q}$, K is the number of intensity intervals, M the number of intensity intervals, $d(\mathbf{q},\mathbf{p})$ the geodesic distance between points $\mathbf{q}$ and $\mathbf{p}$ and d_{max} is the maximum distance used when sampling. Finally, $\mathbf{H_p}(k,m)$ is normalized; first with respect to k once fixed m and then normalized moving m. Similarity between histograms when comparing local features is given by the χ^2 distance.

Results seem to improve the other state of the art methods when using synthetic images. However, on a real data set, differences are not so clear.

A Deformable Local Image Descriptor, [5]

The method proposed in [5] relies on multiple scale support regions to describe the keypoint and on a similarity model that takes profit from this multiple representation. Choosing different

[5]The geodesic distance between two points is the shortest path between these points such that this path is embedded in the surface

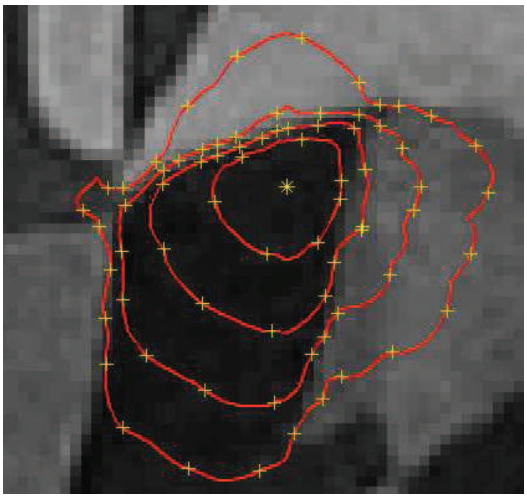

Figure 2.16: Taken from [17], this image shows the geodesic level curves around a concrete interest point when applying the method described. Also, the sampling set used for the histogram descriptor can be appreciated.

Figure 2.17: Two interest points with their different support regions.

support regions permits the problem of selecting an optimal scale (which is very difficult to general deformations) to be skipped and also permits to give more information about the local feature (an unique region may be not sufficient).

Multiple-size support regions around the interest point are obtained through a circular window varying its scale, σ_s. Region scales are $\sigma_s = s \cdot \sigma_0,\ s \in \mathcal{S} = \{0, 1, \ldots, 2N\}$. Figure 2.17 show the different support regions being used in the process.

Descriptor at a given scale is very similar to SIFT: first, gradients are calculated at each pixel and L_1 orientation histograms of L_2 bins each result when dividing the region into L_1 subregions. Before classifying pixels into histogram bins, the whole region is rotated $\theta_s = \arctan\left(\frac{v_{12}}{v_{11}}\right)$ degrees, where $\mathbf{v}_1$ is the eigenvector of the largest eigenvalue of the second moment matrix, $M(x, y)$, defined when describing the Harris detector, see Equation (2.1). In short, for each point $\mathbf{p}$, descriptor is $\mathcal{D}_{\mathbf{p}} = [\mathbf{H}^{\mathbf{p}}_0, \ldots, \mathbf{H}^{\mathbf{p}}_{2N}]$

From that point, what is left is to associate every interest point $\mathbf{q}$ from the query image to one of the candidate points (target image), denoted as $\mathbf{p}_1, \ldots, \mathbf{p}_M$. This is done using a Local-to-Global Similarity Model (LGS) which consists on an unsupervised learning method where support regions act as a cascade of "weak" classifiers.

When comparing $\mathbf{q}$ to $\mathbf{p}_1, \ldots, \mathbf{p}_M$, it seems clear that histograms can not be only compared at the same scale: if image contains an scale transformation, most of histogram comparisons could be wrong even if interest points were the same in both images ($\mathbf{q}$ and $\mathbf{p}_j$ for some j). Because of this, the first thing to do is to find $k^* \in [-N, N]$ such that histograms between $\mathbf{q}$ and $\{\mathbf{p}_j\}_{j=1,\ldots,M}$ are the most similar possible (best scale comparison between them). k^* is assigned to k which minimizes similarity between descriptors $\mathcal{D}_{\mathbf{q}}$ and $\mathcal{D}_{\mathbf{p}_j}$, $\mathbf{p}_j$ being the candidate point given best performance once k is fixed. That is:

$$k^* = amin_{k\in[-N,N]}\{f(k)\}, \quad \text{where } f(k) = \min_{j\in\{1,\ldots,M\}} \left\{ \sum_{i=max(0,k)}^{max(0,k)+N} d_{\chi^2}(\mathbf{H}^{\mathbf{p}_j}_i, \mathbf{H}^{\mathbf{q}}_i) \right\}$$

For the next steps, only N histograms per point are considered, in particular: ($\mathbf{H}^{\mathbf{q}}_{k_0}$, $\ldots$, $\mathbf{H}^{\mathbf{q}}_{k_0+N-1}$), ($\mathbf{H}^{\mathbf{p}_j}_{k_0-k^*}, \ldots, \mathbf{H}^{\mathbf{p}_j}_{k_0-k^*+N-1}$) $\forall j \in \{1, \ldots, M\}$. These histograms serve as input to the unsupervised learning method LGS. As commonly desired, classification is done in two parts: a filtering module that pretends to reject as much unlikely candidates as possible and a refining one that ranks remaining candidate points.

At each classification module, a μ ratio of candidate points are rejected (the worst in the similarity evaluation). At the end of the filtering mode, only $k_{max} = (1 - \mu)^n \cdot M$, k_{max} a parameter and $n \leq N$ such that satisfies the condition.

Chapter 3

The Algorithm

3.1 Overview

Before explaining the main steps of the algorithm separately, in this section we want to give a general idea of the method we have implemented.

The method has an initially complex and large training process in which the image containing the object to be learnt to detect, even when it is deformed, has to be given.

Let's consider we have our image **I** without any deformation. The first thing to do is to **register the image**. This means to give the image a position in space, considering it in the XY plane; also we are giving it $N_x \times N_y$ vertices equally distributed on the image (at distances L_x and L_y one vertex to another along each direction). Registration gives us 3D information for the $N_x \times N_y$ vertices and also gives us the camera model that will be used during the training part.

The next step is to **select interest points on the image**. Selection will be done according the Difference-of-Gaussian method but also taking into account the resistance of these points to perspective changes. The local training will be done around each one of these chosen interest points.

Parallel to these steps, a large number of deformed meshes of $N_x \times N_y$ vertices are generated. Then, these meshes are split into $n_x \times n_y$ meshes, where $n_x,\ n_y << N_x,\ N_y$ respectively, and they are going to act as local deformed meshes. A selection process, involving PCA study, is applied in order to choose the most characterizing deformed small meshes from the whole set of meshes generated. This subset is the one that will be used to synthesize different deformed images around each interest point on the original image and it is called **the training meshes' set**.

Then, for all interest points, we **synthesize the local image** on the different meshes from the training mesh set. All synthesized meshes referring to the same interest point are called elements of the interest point class. In order to appropriately synthesize each mesh around each interest point, **a visibility model** has to be introduced and also **orientation** of each patch has to be determined. The former is applied so that deformations can not give any confusion on the pixel coloring for the training meshes. About orientation, each synthesized image has not

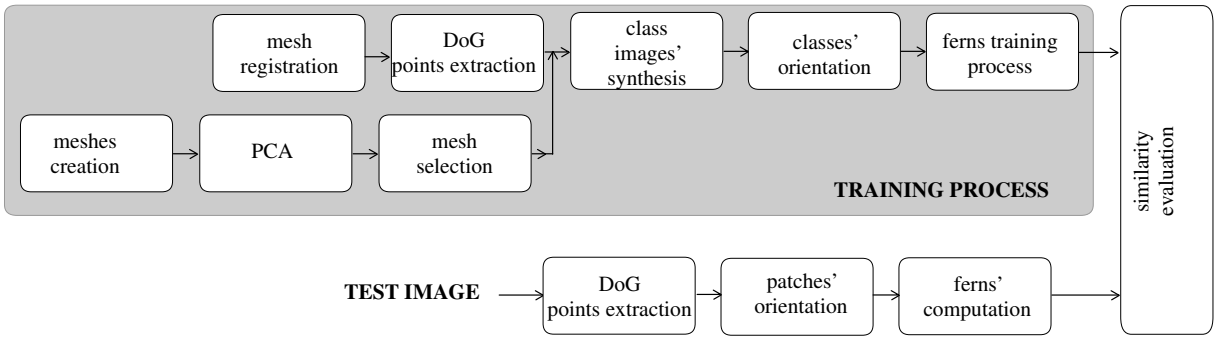

Figure 3.1: Block diagram for the algorithm implemented.

any principal direction a priori. For our process, we would like to have all the elements of the same class oriented the same way. To obtain it, orientation of the synthetic image is calculated with the parameters given for the concrete interest point (for example, scale at which it has been detected is required). And then, a rotation on the synthetic image is applied so that all orientations for each element in the class become the same. Finally classes' images are reduced to about hundred pixels along each direction and having the interest point at the center of this little patch.

The last step for the training process is to characterize each class in a way that it is distinctive of the other ones. This part of the process is done by using **random ferns** [24], which mainly consist on assigning a set of histograms for each class.

Although the learning process is very slow, the algorithm at running time is very fast. Given the test image **I'**, the same interest point detector (DoG) is applied. At each interest point, a region equivalent to the local image region on the training process is created. There, Ferns method for evaluation is used and the similarity to the histograms defining each class for which the image has been trained is computed.

Evaluation ends when giving the criterion for discarding points on the test image that don't correspond to any of the classes and also the criterion to affirm that some of the interest points on the test image correspond to points on the reference image.

An schematic representation of all the steps followed can be seen at Figure 3.1. The following sections explain the different steps taking part in the algorithm in a more detailed way.

3.2 Mesh Registration

3.2.1 Initial Concepts: Homogenous Coordinates and Camera Model [14]

In this section, we are going to study the relation between the 3D coordinates of a point **M** and its correspondent point **m** in the image captured by the camera. This will help to understand following sections where points are indifferently referred to according its 3D coordinates in the different coordinate systems we will explain and also referred to according its 2D representation.

Consider a point $\mathbf{M} = (X\ Y\ Z)^T$, defined in an arbitrary coordinate system, which we call

world coordinate system. Consider camera $\mathbf{C}$ as another point. Image is then formed in a plane at a certain distance from the camera point and it consists on a perspective projection taking rays departing from $\mathbf{C}$, passing through 3D points, such as $\mathbf{M}$, and intersecting the plane at 2D-coordinates described by $\mathbf{m}$. First transformation to obtain $\mathbf{m}$ from $\mathbf{M}$ is to reexpress point $\mathbf{M}$ in the camera coordinate system, that is the system where $\mathbf{C}$ is $\mathbf{C}_{cam} = (0\ 0\ 0)^T$. This transformation consists on a rotation and translation, $\mathbf{M}_{cam} = \mathbf{R}_{3\times3}\mathbf{M} + \mathbf{T}_{3\times1}$,

$$\mathbf{M}_{cam} = \begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{pmatrix} = \mathbf{R} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathbf{T} = (\mathbf{R}|\mathbf{T}) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = (\mathbf{R}|\mathbf{T})_{3\times4}\tilde{\mathbf{M}}$$

The 3×4 matrix $(\mathbf{R}|\mathbf{T})$ is called the *external calibration matrix*. Note that camera point in the world coordinate system is $\mathbf{C} = -\mathbf{R}^{-1}\mathbf{T}$.

Next step is to project $\mathbf{M}_{cam}$ into the image plane. The image plane is often located so that its normal vector in the camera coordinate system is $\vec{\mathbf{n}} = (0,\ 0,\ -1)^T$ and its distance from the camera is f. Using simple triangular relations we obtain:

$$\frac{m_X}{f} = \frac{X_{cam}}{Z_{cam}} \ \Rightarrow\ m_X = f\frac{X_{cam}}{Z_{cam}}$$

$$\frac{m_Y}{f} = \frac{Y_{cam}}{Z_{cam}} \ \Rightarrow\ m_Y = f\frac{Y_{cam}}{Z_{cam}}$$

The last step would be to reexpress $\mathbf{m} = (m_X, m_Y)^T$ into the image coordinate system, $\mathbf{m} = (u, v)^T$. Image coordinate system has its center at the top left corner of the image and its axis are $\{u, v\}$, parallel to axis $\{X, Y\}$ on the camera coordinate system respectively. Let (u_0, v_0) be the projection of the camera center $\mathbf{C}_{cam}$ on the image and k_u, k_v the inverse of the size of one pixel along axis u and v respectively. Then, $(u, v) = (u_0 + k_u m_X, v_0 + k_v m_Y)$. Putting together the last two steps (projection and transformation into pixel coordinates), we obtain:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{U}{W} \\ \frac{V}{W} \end{pmatrix}, \quad \text{where } \begin{pmatrix} U \\ V \\ W \end{pmatrix} = \mathbf{A}_{3\times3} \begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that $(U\ V\ W)^T$ are the homogenous coordinates defining $\mathbf{m}$. Matrix $\mathbf{A}$ is called the *internal calibration matrix*. It is usual to assume $k_u = k_v$ and (u_0, v_0) to be image center, so dimensionality is reduced.

Summarizing, relation between a point $\mathbf{M}$ in the world coordinate system and its image coordinate representation, $\mathbf{m} = (u, v)^T = \left(\frac{U}{W}, \frac{V}{W}\right)^T$, is:

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} = \mathbf{A}(\mathbf{R}|\mathbf{T}) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} & \mathbf{R}_{13} & \mathbf{T}_1 \\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} & \mathbf{T}_2 \\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} & \mathbf{T}_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Figure 3.2 shows graphically the different notations introduced in this section.

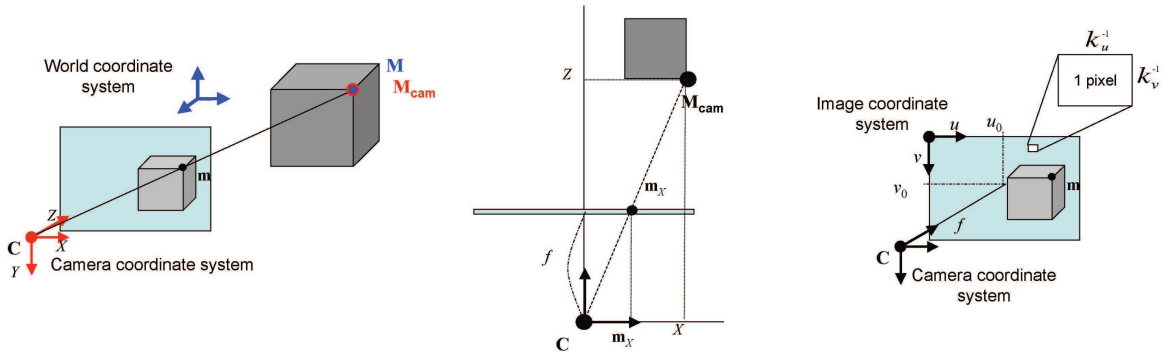

Figure 3.2: At left, correspondence between the world and the camera 3D coordinates systems. At center, correspondence between camera coordinates and projection in the image plane. At right, image coordinates system.

3.2.2 Mesh Registration in Our Algorithm

As previously mentioned in the previous section, our first aim is to give some spatial information that will be useful on next stages to the image. This means we want to know parameters: **R** and **T** explained at the last section and we also want the exact position of the $N_x \times N_y$ vertices on the mesh in 3D world coordinates. In particular, we set our attention in the synthetic experiment we have worked with.

We assume the camera to be calibrated and, hence, that we know the matrix **A**. In our synthetic experiments, we have worked with a 500×1000 Guernica image and **A** is set to:

$$\mathbf{A} = \begin{pmatrix} 1000 & 0 & 500 \\ 0 & 1000 & 250 \\ 0 & 0 & 1 \end{pmatrix}$$

Once the internal calibration matrix is known, we have a *perspective to point problem* to resolve: knowing some correspondences between image points and their 3D coordinates, we want to estimate the position and orientation, that is the external calibration matrix, $(\mathbf{R}|\mathbf{T})$. It can be shown that with only four 3D-2D correspondences we already have **R** and **T**. So we give an invented position for the extremes of the object in the image and we mark the related points on the image. With these inputs (**A** and XYZ-position for four concrete points in the image), we can determine the XYZ position of every pixel in the image[1].

Once we are provided the external calibration matrix, we can interpolate the $N_x \times N_y$ vertices in the image covering the object from one of the extremes marked to another. In our case we fitted a 41×21 mesh in the Guernica, which was considered a 30×60 *cm* rectangle.

Guernica registered image, with the 41×21 mesh fitted in the image, can be seen at Figure 3.3.

[1] **Remark:** That can be done because we are assuming a planar object in our image.

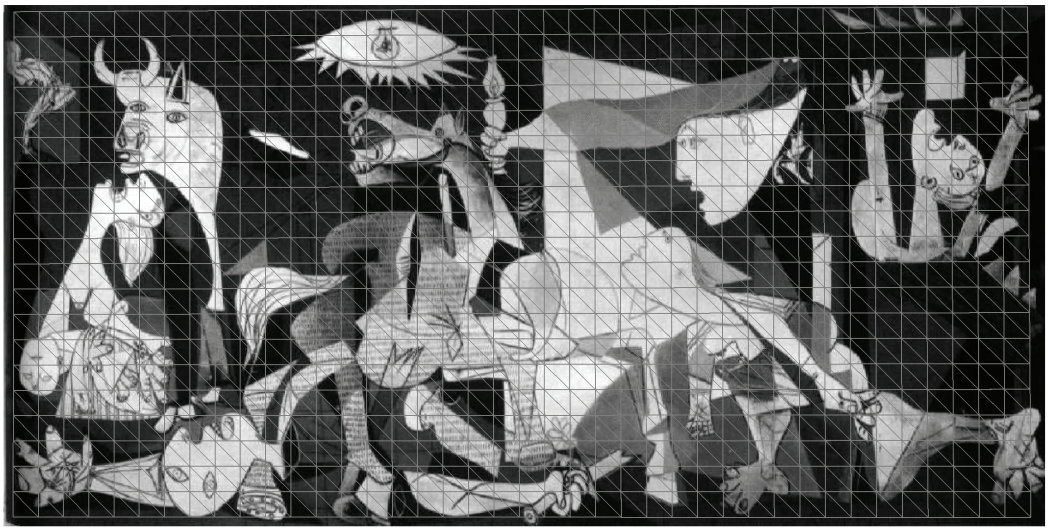

Figure 3.3: Once registration has been done we obtain coordinates XYZ of a 41×21 mesh covering the image.

3.3 Selecting Interest Points

The method for selecting the interest points is based on [16]. It consists on generating different views of the object. Then, at each view, we compute the interest points in the image using the SIFT detector's method, that is DoG. Since transformations applied to generate the different views can be inverted, all points can be reprojected at the reference image (the reference view). There, an easy counting process can determine the most repeatable interest points under perspectives changes and these are the keypoints we select.

In this section we first explain the SIFT detector's method and then we explain the weighting done by all the perspectives images generated.

3.3.1 SIFT Detector Based on DoG

As previously mentioned, the detector we have used in our method is the one used by Lowe in [18]. It is an scale invariant detector and, basically, an efficient modification of the Laplacian of Gaussian detector. We will now proceed to explain it.

Detection of Scale-space Extrema

The scale space of an image was defined (see Section 2.1.1) as $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$ where $G(x, y, \sigma) \sim \mathcal{N}(\mathbf{0}, \sigma)$, that is $G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$, and $I(x, y)$ is the input image. Instead of using directly this scale-space function, it is proposed to calculate the scale-space extrema in the difference-of-Gaussian function convolved with the image, $D(x, y, \sigma)$, computed

from the difference of two nearby scales separated by a multiplicative factor k:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

First, notice that $D(x, y, \sigma)$ is directly equal to $L(x, y, k\sigma) - L(x, y, \sigma)$. Consequently, in efficiency terms, it only differs on a substraction from the LoG method. Second, it can be shown that $\frac{\partial G}{\partial \sigma} = \sigma \bigtriangledown^2 G$ and, since $\frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$, we finally got

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \bigtriangledown^2 G. \tag{3.1}$$

This shows that Gaussian functions differing by a constant k, it already incorporate the σ^2 factor, which, remember, is a crucial normalization factor when defending scale-invariance for the Laplacian function. Since $(k-1)$ is a constant along the different scale, it doesn't affect in the extrema location. Approximation in Equation (3.1) is exact when taking $\lim_{k \to 1}$. Nevertheless, approximation seems to have a very little impact on extrema detection.

In practice the Difference-of-Gaussian process is done in the following way: The input image is successively convolved by Gaussians to produce images separated by k in the scale-space. From σ to 2σ (that would be an octave of the scale-space), different values depending on a parameter s are taken, where s is an integer number. Fixed an octave, we take $k = 2^{\frac{1}{s}}$ and the total amount of smoothed images is $s + 3$ (so that the octave is completely covered). Adjacent images from the same octave are subtracted so $s + 2$ Difference-of-Gaussian functions are obtained. Once an octave is completed, blurred images from the next octave are resampled taking half of the pixel rows and columns (down-sampling by a factor 2). This reduces image size from $n \times m$ to $\frac{n}{2} \times \frac{m}{2}$, which affects positively to computation. Meanwhile accuracy is not really affected: it remains the same that existed at the start of the previous octave.

Figure 3.4 shows the process explained and Figure 3.5 shows a real example.

As in LoG, the comparison for local extrema detection is done not only with the eight neighbors of the same scale Difference-of-Gaussian function, but also to the nine from the DoG function at the scale above and the nine at the scale below. When looking for extrema, a trade-off appears between efficiency and completeness. That is because, for example, there are features which suffer a continuous transition from a maximum to two maxima. And this has to be taken into account when deciding the value of the s parameter introduced before. In [18], the parameter s is chosen when evaluating different possibilities into a synthetic data set. The result is that best performance is obtained when $s = 3$ and that is also the scale value per octave that we use. When increasing more s, no improvement is shown since points lose part of their stability and therefore also their repeatability through different images. Another parameter that has to be previously fixed is the initial smooth, σ_0. This parameter also represents a trade-off between computational cost and repeatability. The value taken is $\sigma_0 = 1.6$.

Finally, another observation has to be made. Blurring the image before extrema detection, as we are doing, eliminates some of the extrema (the ones with the most sharpened changes, the ones from high spatial frequencies). In order to counteract somehow the effect, at the first octave, the image is doubled in size by using linear interpolation. In fact, when assuming a pre-existent blur of $\sigma = 0.5$ in the input image, which is perfectly consistent, doubling image

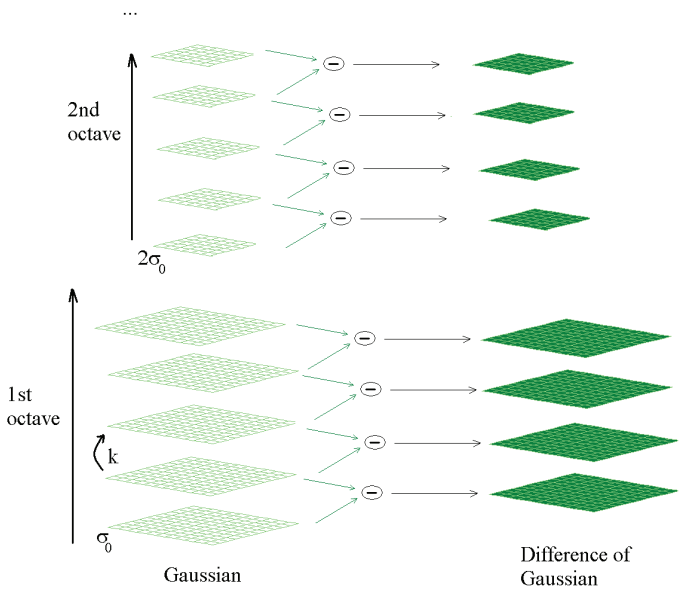

Figure 3.4: At left the first stage can be seen, where image suffers different convolution to Gaussian kernels. Octaves and down-sampling are shown. At right, the Difference-of-Gaussian functions obtained when subtract ing to adjacent smoothing levels at the same octave are shown.

size gives a blur of $\sigma = 1$ and then, choosing $\sigma_0 = 1.6$ incorporates very little smoothing in the first octave. Doubling the size has shown to increase the number of stable points by a factor of 4.

Accurate Keypoint Localization

Among the whole set of local extrema, those with a low contrast have to be discarded. In the proposed algorithm, an initial elimination is done for extrema which are weakly maxima (or minima) compared to their neighbors. This isn't enough and to obtain a final stable set of keypoints, accuracy on location and determination of principal curvatures of each interest point has to be done first.

For accuracy on location, let $\mathbf{x}_0$ be a keypoint found as a maximum of $D(\mathbf{x}, \sigma)$ for a certain, fixed, σ. Consider the Taylor expansion of scale-space function around $\mathbf{x}_0$:

$$D(\mathbf{x}, \sigma) = D_{\mathbf{x}_0} + \frac{\partial D^T}{\partial \mathbf{x}}\Big|_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\Big|_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) + \mathcal{O}((\mathbf{x} - \mathbf{x}_0)^3)$$

A first finer approximation for the extremum location is obtained when truncating Taylor expansion at the third term and imposing the derivative of the polynomial approximation to be zero. Result is $\hat{\mathbf{x}} = \mathbf{x} - \mathbf{x}_0 = \left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial D^T}{\partial \mathbf{x}}\Big|_{\mathbf{x}_0}$. This leads to a new location for the interest point, which is $\mathbf{x}'_0 = \mathbf{x}_0 + \hat{\mathbf{x}}$.

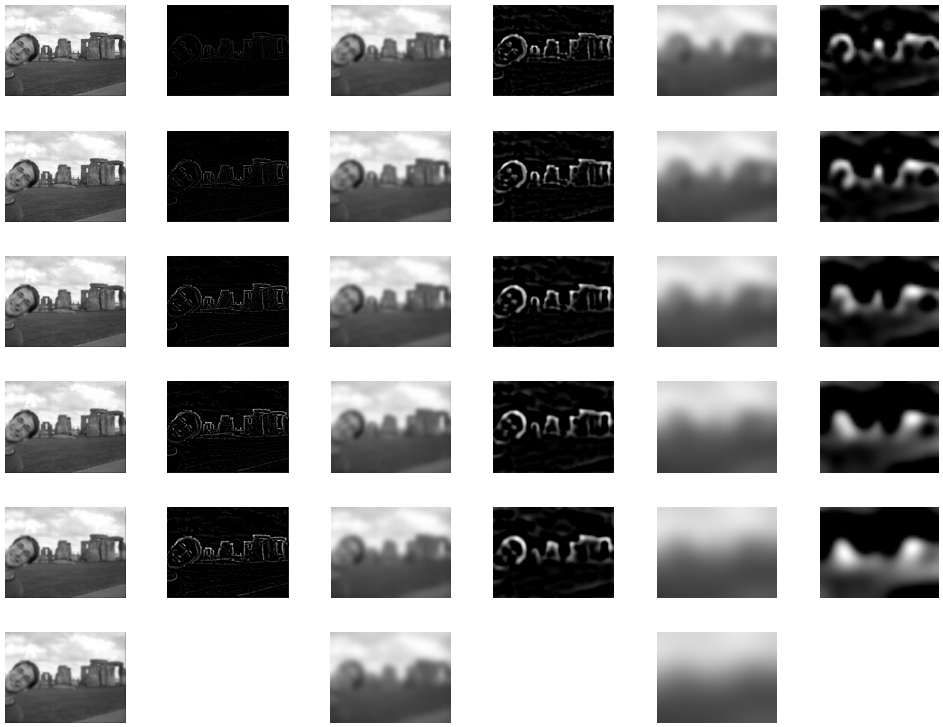

Figure 3.5: Octaves 1, 3 and 5 are shown by columns. First columns correspond to the Gaussian smoothing part while the second ones appear when computing the difference of Gaussian at adjacent scales. Contrast in the DoG images has been incremented so that the forms appearing can be appreciated.

For calculating both derivative and Hessian, an approximation by differences of neighboring sample points is done. Once the derivative and Hessian are calculated, $\hat{\mathbf{x}}$ is obtained as a result of a 3×3 linear system. When $\hat{\mathbf{x}}$ is found such that its absolute value is greater than 0.5 in a certain direction, this means that the extremum lies closer to a neighbor than the initial interest point, $\mathbf{x}_0$. Then, the sample point is changed and interpolation is calculated from the new sample point (it is an iterative method).

The value of the DoG function at the accurate extremum $\hat{\mathbf{x}} + \mathbf{x}_0$ and at the appropriate scale is then computed. Since it has been imposed that $\frac{\partial D^T(\hat{\mathbf{x}}+\mathbf{x}_0)}{\partial \mathbf{x}} = 0$, the value at extremum is

$$D(\hat{\mathbf{x}} + \mathbf{x}_0, \sigma) = D_0 + \frac{1}{2}\hat{\mathbf{x}}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\Big|_{\mathbf{x}_0} \hat{\mathbf{x}} = \left\{\hat{\mathbf{x}}^T = \frac{\partial D^T}{\partial \mathbf{x}}\Big|_{\mathbf{x}_0} \left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1}\right\} = D_0 + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\Big|_{\mathbf{x}_0} \hat{\mathbf{x}}$$

A second threshold is used: extrema with values below 0.03 (when assuming intensity range $[0, 1]$) are automatically discarded.

There is still a last step remaining in order to obtain the final keypoints' set. This consists on

calculating the principal curvatures of the Hessian matrix. Principal curvatures will be used to discard points belonging to edges. Edges are not convenient since they show very little stability for the DoG detection. As usual, Hessian eigenvalues are no directly needed and a cornerness criterion can be established from the trace and determinant of the Hessian matrix, $\mathbf{H}$. Criterion established is: $\frac{\text{trace}(\mathbf{H})^2}{\det(\mathbf{H})} = \frac{(\lambda_1+\lambda_2)^2}{\lambda_1 \cdot \lambda_2} = \frac{(r+1)^2}{r} < \frac{(10+1)^2}{10}$, where λ_1, λ_2 are the eigenvalues of $\mathbf{H}$ and r is such that $\lambda_1 > \lambda_2$, $\lambda_1 = r \cdot \lambda_2$. In Figure 3.6 points detected before taking on account the edges can be seen.

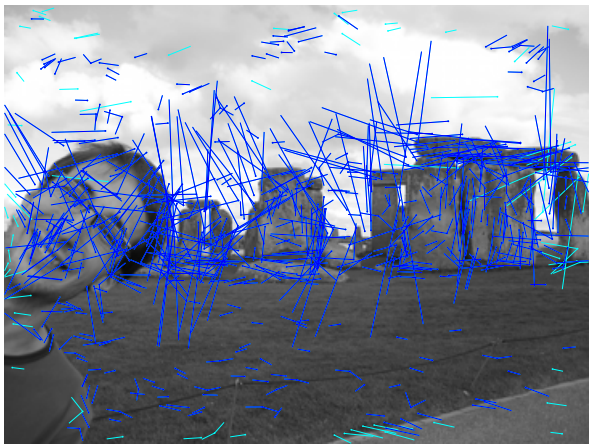

Figure 3.6: Set of detected points. Orientations are specified by vectors orientation which are born at the interest point location and which their magnitude is proportional to their gradient magnitude. In cyan, the points rejected after applying the edge criterion. In this image, initial set was of 738 points whereas, after eliminating boundary points, set had 669 elements.

Orientation Assignment

Orientation assignment is a previous step before computing the SIFT descriptor. It introduces rotation invariance to the local feature. The procedure for obtaining the characteristic orientation is similar to the ones described in Section 2.1.

Let $L(x, y, \sigma)$ the Gaussian smoothed image at the scale the keypoint has been recognized. For each image sample at this scale, the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, are computed using pixel differences, that is:

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

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

A Gaussian-weighted circular window with variance one and a half of the characteristic scale is applied to gradient magnitudes around the interest point. An orientation histogram is then constructed where samples are weighted by their gradient magnitude once they have been windowed. The orientation histogram has 36 bins covering the 360 degree range of orientations.

Peaks in the histogram represent the dominant directions of the local gradients. Not only the highest peak but also the ones within the 80% of the highest are considered. Each orientation serves to create an interest point at the same location. For a better accuracy of the orientation values, an interpolation is done using the adjacent bins to each peak bin. Figure 3.6 shows the detected points set and also the direction of each keypoint. Also, points with more than an orientation can be seen.

3.3.2 Perspective Weighting

We assume to have our planar object in the XY plane in world coordinates. With the registration process, we obtain matrix $(\mathbf{R}|\mathbf{T})$. Therefore, as seen in the previous section, we have the camera position in the world coordinates, $\mathbf{C} = -\mathbf{R}^{-1}\mathbf{T}$. With all these data, we can think about creating new perspectives by positioning camera at some other places in space. What we have done is to consider an sphere with equator at the XY-plane and to place camera at different discrete points on the sphere surface. Sphere radius is given by initial camera position distance to the XY-plane.

More specifically, considering spherical coordinates $\{r, \phi, \theta\}$, we have r fixed, $r = r_0 = d(\mathbf{C}, \{z = 0\})$. Then we place a camera for new views at positions:

$$(r, \phi, \theta) \in \{r_0\} \times \{0, \tfrac{1}{10}\pi, \ldots, \tfrac{19}{10}\pi\} \times \{\tfrac{1}{10}\tfrac{\pi}{2}, \ldots, \tfrac{10}{10}\tfrac{\pi}{2}\}$$

Once the camera centers have been placed at different points of the sphere, each one can be associated a new external calibration matrix $(\mathbf{R}'|\mathbf{T}')$ and the $N_x \times N_y$ mesh can be reexpressed in these new camera coordinates. Then, using barycentric coordinates and synthesizing the image for the new perspectives (as it is explained in Section 3.5), we obtain the different views of the same object which first was only seen in one perspective (the front one).

For each new viewpoint, SIFT detector is applied to compute the interest points. Since transformation is perfectly known, pixel location of the interest points can be transferred to the reference image via, once again, barycentric coordinates. Once this process is completed, all interest points in the different views are located in the reference image. There, every interest point in the reference image is weighted by the number of times that has appeared among all the different perspectives. That way, interest points are ordered and we only keep those more repeated.

Next figures show what we have explained for the Guernica image. At Figure 3.7, discrete positions in the imaginary sphere for the different perspectives are shown with the resulting images. For these examples, Figure 3.8 shows interest points found already projected at the reference image. Finally, Figure 3.9 shows the best 1200 points found in the image using perspective weighting. Depending on the experiment we have used a different quantity of interest points (from 400 to 1200), always using the most repeatable ones.

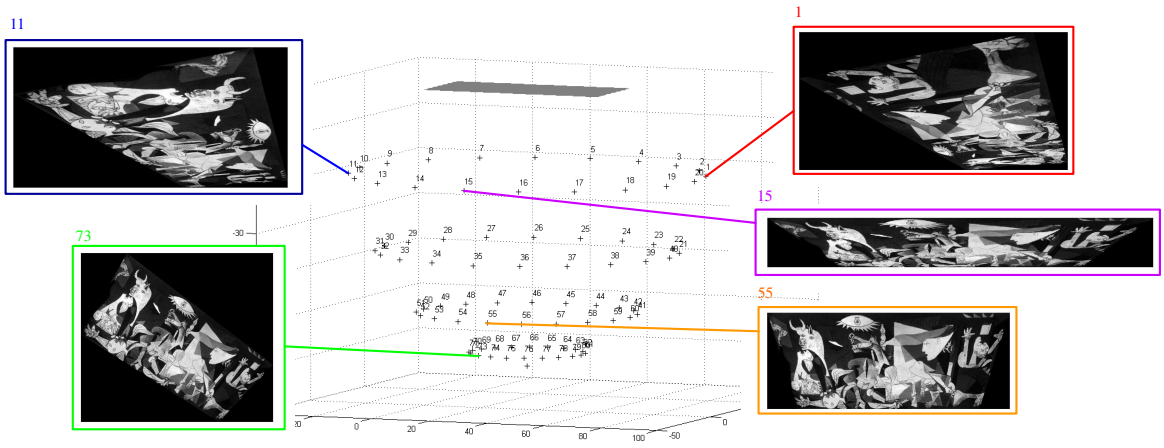

Figure 3.7: Different camera positions are shown so that different views of the Guernica can be obtained. The reference mesh is placed at the XY plane (at top of the image). Also five different views with the camera position indicated are shown.

3.4 Meshes

The aim of this section is to explain the way we construct a set of deformed meshes. Remember that, for every keypoint found, we will synthesize a local image on each deformed mesh. This gives us a description of each point in several ways from which ferns will be trained.

In order to construct the set of deformed meshes, we first create a bigger set of deformed meshes as it is explained in Section 3.4.1. Then, we study *the most representative* meshes and we elaborate the final set. To give sense to the concept of *most representative*, we have used PCA, explained in Section 3.4.2, and have also followed method explained in Section 3.4.3.

3.4.1 Meshes' Creation

As said before, we have focused in planar deformable objects such as tissues. We represent the surface by a triangulated mesh where deformations are modeled by angles between the planar triangles. Following [26] and [27], we also consider that the vertices in the mesh maintain their distances with their closest neighbors. We now explain the method followed for creating the meshes, the constraints and parameters taking part in the process.

Given the number of vertices in each direction, N_y for the number of rows and N_x for the number of columns, the vertex $(n, m) = (1, 1)$ is fixed in 3D-space. Its position will be taken randomly within a range of values in each direction ($[lim_{inf}, lim_{sup}]$). So position XYZ of the first vertex is: $XYZ(1, 1) = (x, y, z)$, where $x,\ y,\ z \sim U(lim_{inf}, lim_{sup})$, and U is an uniform distribution probability within the interval.

Also, neighbors' vertices of $(1, 1)$, that is $(1, 2)$ and $(2, 1)$, are fixed. In order to do it, the plane containing triangle $\triangle\{(1, 2)(1, 1)(2, 1)\}$ is determined from 3 random angles between a

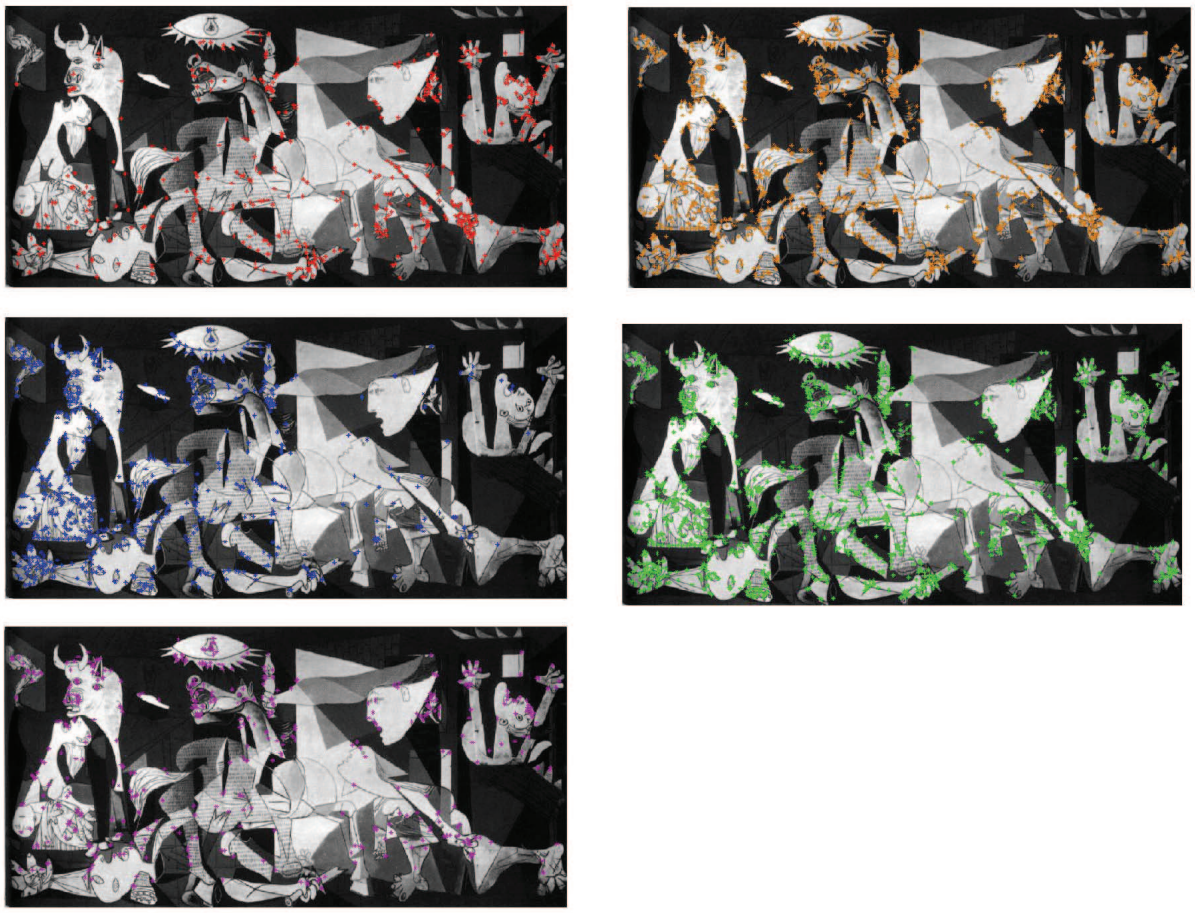

Figure 3.8: Here, interest points detected at each of the five views shown the previous image can be observed once reprojected at the reference image. Notice that distribution of the interest point at each image corresponds to the perspective used in each image (for example, image 11, in blue, has much more interest points to the left part).

specified interval, $\theta_x,\ \theta_y,\ \theta_z \sim U[\theta_{inf}, \theta_{sup}]$, so each angle represents a turn around one of the axis. Then, the plane is determined by the orthogonal vectors obtained as:

$$\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad \text{where } \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \text{ is}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{pmatrix} \begin{pmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{pmatrix} \begin{pmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Moreover, vertices $(1, 2)$ and $(2, 1)$ are placed in the plane at given distances L_x and L_y from

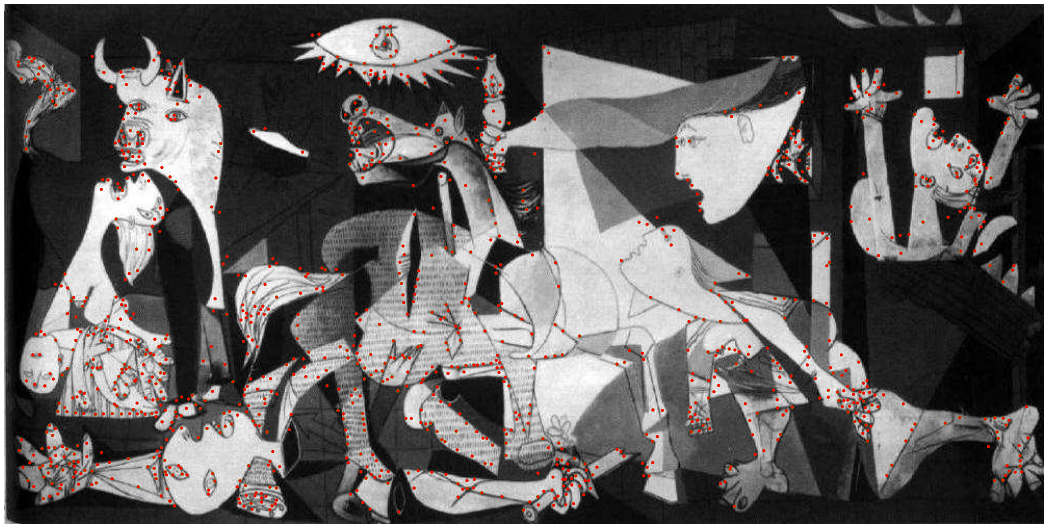

Figure 3.9: First 1200 points once detection by DoG has been made and also after having ordered the points according to their appearance in the different perspective images of the Guernica.

vertex $(1,1)$ in direction $\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \cdot \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$ and $\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \cdot \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^T$ respectively.

The next aim is to create the first row (*band*) of the mesh, that means creating triangles formed with vertices corresponding only to the first and second rows ($n \in \{1,2\}$). In order to do it, vertices are determined in the following order: $\{(2,2),(1,3),(2,3),(1,4),\ldots,(1,N_x),(2,N_x)\}$.

The process to obtain them is the following:

1. Given the triangle $\triangle_1 = \triangle\{(1,2)(1,1)(2,1)\}$[2], next vertex $(2,2)$ is obtained. Vector edge $\overrightarrow{L_2} = \overrightarrow{(2,1)(1,2)}$ is obtained and its length, L_2 calculated. We called edge $\vec{L}_2$ the common edge between last triangle made and the one being constructed. Other edges are called $\vec{L}_0$ and $\vec{L}_1$. In this particular case, since $\overrightarrow{L_0} \perp \overrightarrow{L_1}$, where we choose $\overrightarrow{L_0} \doteq \overrightarrow{(1,2)(1,1)}, \overrightarrow{L_1} \doteq \overrightarrow{(2,1)(1,1)}$ and, as known, $|\overrightarrow{L_0}| = L_x, |\overrightarrow{L_1}| = L_y$, we have got $|\overrightarrow{L_2}| = \sqrt{L_x^2 + L_y^2}$. Nevertheless, the new vertex construction method doesn't count on orthogonality between any of the triangle edges.

2. A point C is calculated on edge $\overrightarrow{L_2}$ at distance $a = \frac{L_1^2 - L_0^2 + L_2^2}{2L_2}$ from vertex $(1,2)$ (in this case, distance $a = \frac{L_y^2 - L_x^2 + L_x^2 + L_y^2}{2\sqrt{L_x^2+L_y^2}} = \frac{L_y^2}{\sqrt{L_x^2+L_y^2}}$). Vector $\overrightarrow{v} = \overrightarrow{C(1,2)}$ is taken. If a were very little, $\overrightarrow{v}$ would be taken as $\overrightarrow{v} = \overrightarrow{(2,1)C}$ just to avoid precision errors. Then a perpendicular

[2]Here, triangles will be always notated as $\triangle\{A,B,C\}$ where $\overrightarrow{AB} \perp \overrightarrow{BC}$, $|\overrightarrow{AB}| = L_x$ and $|\overrightarrow{BC}| = L_y$

vector $\overrightarrow{v}_\perp$ contained in the plane determined by $\triangle_1$ is considered ($\vec{v}, \vec{v}_\perp \in \triangle_1$, $\vec{v}_\perp \perp \vec{v}$). These two vectors allow to construct an orthonormal basis $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$ where $\vec{v}_2 = \frac{\vec{v}}{|\vec{v}|}$, $\vec{v}_3 = \frac{\vec{v}\times\vec{v}_\perp}{|\vec{v}\times\vec{v}_\perp|}$ and $\vec{v}_1 = \vec{v}_2 \times \vec{v}_3$. At the end, $\vec{v}_1 = \lambda\vec{v}_\perp$, with λ a positive or negative constant in order to create a direct base.

A new vector $\vec{r}$ with origin at C is obtained. It is contained in plane $< \vec{v}_1, \vec{v}_3 >$ forming an angle θ_1 given with the first vector $\vec{v}_1$. Its length is $r = \sqrt{L_1^2 - a^2}$. We then define vertex $(2,2)$ as point $C + \vec{r}$. It is easy to see that $\vec{r}$ length is such that $d((2,2),(1,2)) = L_1$ and $d((2,2),(2,1)) = L_0$, as we wanted:

$$d((2,2),(1,2)) = \left\{\overrightarrow{C(1,2)} = \lambda\vec{v}_2,\ \vec{r} \in < \vec{v}_1, \vec{v}_3 > \Rightarrow \vec{r} \perp \overrightarrow{C(1,2)}\right\} = r^2 + a^2 = L_1^2 - a^2 + a^2 = L_1^2$$

$$\begin{aligned} d((2,2),(2,1)) &= \overrightarrow{C(2,1)}^2 + r^2 = (L_2 - a)^2 + (L_1^2 - a^2) = (\sqrt{L_0^2 + L_1^2} - a)^2 + (L_1^2 - a^2) \\ &= L_0^2 + L1^2 - 2a\sqrt{L_1^2 + L_0^2} + a^2 + L_1^2 - a^2 = \left\{a = \frac{L_1^2}{2L_2}\right\} \\ &= 2L_1^2 + L_0^2 - 2\frac{L_1^2}{L_2}L_2 = L_0^2 \end{aligned}$$

Construction of triangle $\triangle_2$ from $\triangle_1$ can be seen at Figure 3.10.

3. Once triangle $\triangle_2 = \triangle\{(2,1)(2,2)(1,2)\}$ has been obtained, next vertex $(1,3)$ and triangle $\triangle_3 = \triangle\{(1,2)(1,3)(2,2)\}$ are calculated doing the same steps. In this case, $|\vec{L}_2| = L_y$, and we fix $\vec{L}_0$ such that $\vec{L}_0 = \sqrt{L_x^2 + L_y^2}$ and $\vec{L}_1 = L_x$. Then, $a = \frac{L_1^2 - L_0^2 + L_2^2}{2L_2} = L_x^2 - (L_x^2 + L_y^2) + L_y^2 2L_y = 0$, and C is consequently placed at vertex $(2,1)$, and basis obtained with the anterior method is $\{\vec{v}_1 \| \overrightarrow{(2,2)(2,1)},\ v_2 \| \overrightarrow{(2,2)(1,2)}, v_3 \perp \triangle_2\}$. Same way, $r = L_1^2 - a^2$ is then equal to L_1 and orientation is given by θ_2 in plane $< \vec{v}_1, \vec{v}_3 >$ given. Triangle obtained, $\triangle_3 = \triangle\{(1,2)(1,3)(2,2)\}$. Its construction can also be seen at Figure 3.10.

4. Finally, the process is repeated for all the vertices in the first two rows following the order announced before. That way, first $2(N_x - 1)$ triangles (the first band) is finished, see Figure 3.10.

   For the moment, we have required the following degrees of freedom: N_x, N_y, L_x, L_y, $XYZ(1,1)$, θ_x, θ_y, θ_z, θ_1, ..., $\theta_{2(N_x-1)-1}$.

Once created the first band, the next rows have less degrees of freedom since constraints are more stringent. As we will see, for each row, there are only two degrees of freedom. That will give a total of $2 \cdot (N_y - 2)$ additional values, $\theta_{2(N_x-1)}, \ldots, \theta_{2(N_x-1)-1+2(N_y-2)}$. From rows from one to $(n-1)$, vertices from row n, that is $\{(n,1), \ldots, (n,N_x)\}$ are obtained as follows:

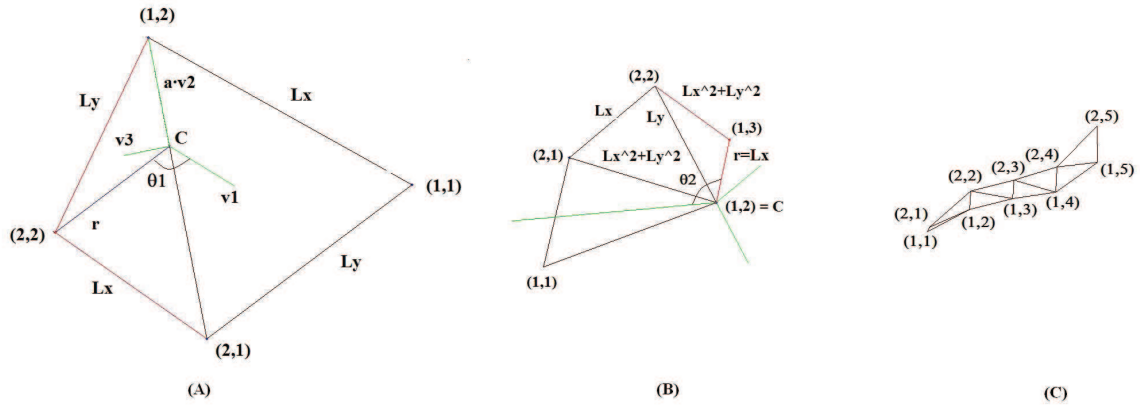

Figure 3.10: Creation process for the first band of the mesh: at (A), triangle $\triangle_2$ is created from the three first vertices. Parameters used can also be seen. At (B) triangle $\triangle_3$ is created from $\triangle_2$. At (C), the 1st band of a 7×5 mesh is shown.

1. Vertex (n,1) is obtained following the process described above applied to triangle $\triangle\{(n-1,1)(n-1,2)(n-2,2)\}$, with $\vec{L}_0 = \overrightarrow{(n-2,2)(n-1,2)}$, $\vec{L}_1 = \overrightarrow{(n-2,2)(n-1,1)}$ and $\vec{L}_2 = \overrightarrow{(n-1,1)(n-1,2)}$ and given an angle $\theta_{2(N_x-1)+2(n-3)}$ to determine $\vec{r}$.

2. Vertices $(n,2),\ldots,(n,M-1)$ are successively calculated. Each vertex (n,i) is perfectly determined since we have three constraints: $d((n,i),(n,i-1)) = L_x$, $d((n,i),(n-1,i)) = L_y$ and $d((n,i),(n-1,i+1)) = \sqrt{L_x^2 + L_y^2}$. This part is implemented by a non linear equation solver.

3. Finally, vertex (n,N_x) introduces a freedom degree since there is no constraint $d((n,N_x),(n-1,N_x+1)) = \sqrt{L_x^2 + L_y^2}$. That is why another angle, $\theta_{2(N_x-1)+2(n-3)+1}$, is used and (n,N_x) is determined through $\triangle\{(n-1,N_x)(n-1,N_x-1)(n,N_x-1)\}$ defining edges as follows: $\vec{L}_0 = \overrightarrow{(n-1,N_x-1)(n-1,N_x)}$, $\vec{L}_1 = \overrightarrow{(n-1,N_x-1)(n,N_x-1)}$ and $\vec{L}_2 = \overrightarrow{(n-1,N_x)(n,N_x-1)}$. These last steps can all be observed at Figure 3.11.

Generation of the Training Meshes in Our Algorithm

For synthesizing the images on deformable meshes, we first modeled the whole image as a mesh of $N_x \times N_y$ vertices, for example, 41×21 vertices in the Guernica image, with equal distances along X-axis and Y-axis, that is $L_x = L_y$. The next step consisted on generating lots of $N_x \times N_y$ meshes (about 800 meshes) given interval limits for the parameter angles between triangles in the mesh (all angles $\theta \sim U[\theta_{\text{lim inf}}, \theta_{\text{lim sup}}]$ where range values were specified at the algorithm and changed among the different experiments). $XYZ(1,1)$ and angles θ_x, θ_y and θ_z aren't important since all meshes are aligned: mean is subtracted and meshes' orientation is fixed to vector (0 0 1). The latter is done using PCA, explained in the next section.

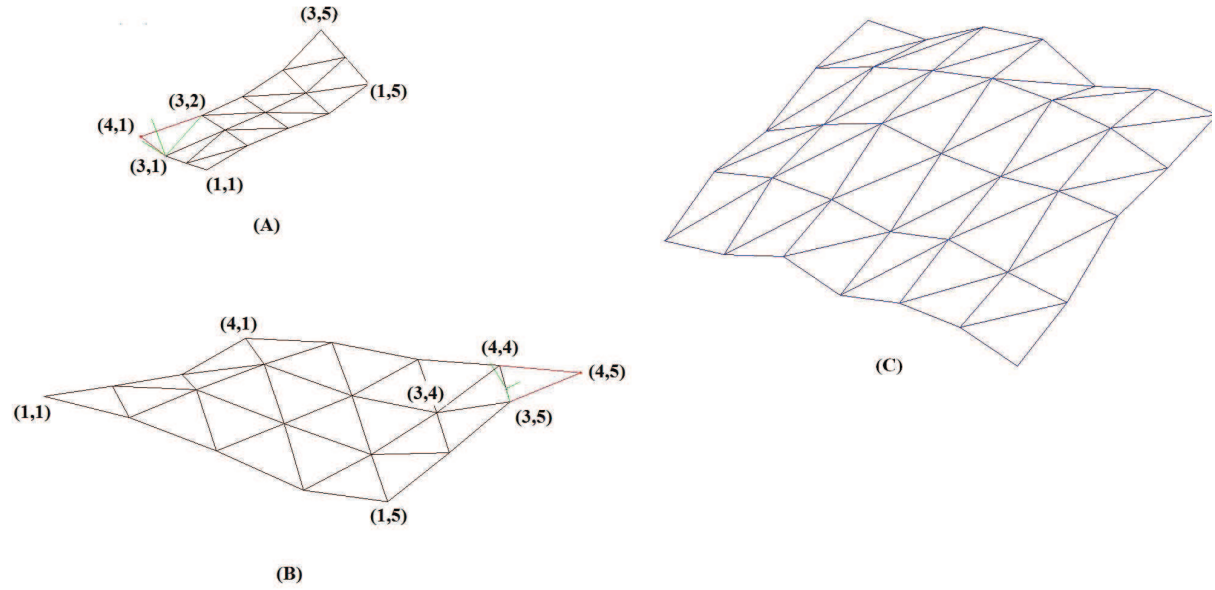

Figure 3.11: At (A), first vertex of an arbitrary row (row 3 in this case) is constructed as told. In (B) the other vertices of the row are constructed and last one introduces another freedom degree. At (C) the final 7×5 mesh is shown

In order to create small meshes where to synthesize the different interest points, we consider all the possible 9×9 meshes resulting from the bigger $N_x \times N_y$ ones and then the selection process starts by taking the more representative ones as it will be seen in Section 3.4.3. So, instead of directly creating small meshes, we take sub-meshes from a bigger one. This is done to avoid certain boundary artifacts produced by the algorithm. See Figure 3.12 for a better understanding.

3.4.2 Principal Component Analysis, PCA [28]

In this section, we explain the PCA concept in general. PCA has been helpful in various stages of our algorithm. As it will be explained, we have used this technique to orientate all meshes the same way and it has also been crucial in the configuration of the deformed meshes set for the training.

PCA is a standard technique for dimensionality reduction. It is also widely used when data analyzing since it is a simple non-parametric method which permits extracting relevant information from big and confusing data sets.

When taking measures, we often don't know what we are exactly looking for, what dynamics are appearing, which behavior data is following, if we are measuring more than it is needed, etc. What PCA does is to change the basis of the measurements so that dynamics and redundant data or noise are well distinct in the new basis.

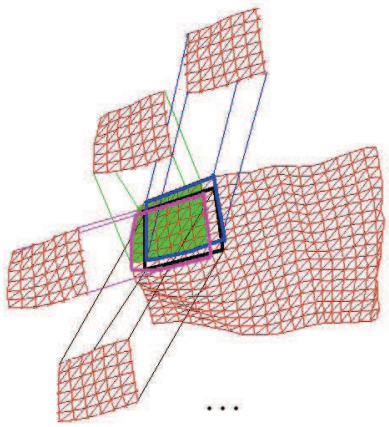

Figure 3.12: Here a 17×21 mesh is shown and some of the first 7×7 submeshes too. Notice that differences between some submeshes are only in two rows or two columns, etc. For example in a 17×21 mesh, 357 7×7 meshes can be found.

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ be the data obtained in n trials, where $\mathbf{x}_i$ is an m-dimensional vector consisting on m different measurements done at the same moment, a sample. PCA pretends to find the best basis that can be obtained by linear combination of the existent basis vectors, that best re-express the $\mathbf{X}$ data set, that is, find $\mathbf{P}$ such that $\mathbf{Y} = \mathbf{P} \cdot \mathbf{X}$ where the rows of $\mathbf{P}$, $\{\mathbf{p}^1, \ldots, \mathbf{p}^m\}$, are the vectors of the new basis.

But what is the best possible re-expression of the data set? Data set in a linear system can only be disturbed by noise, redundancy or underlying dynamics can be hidden by a rotation. A common assumption, and crucial when applying PCA, is that directions with the largest variances are the ones that contain the dynamics of the system or the data information, whereas the ones with little variances are the ones only affected by noise (noise variance depends on the data quality). Another fact that has to be taken on account is that some of the measures can be somehow related. When these ties are linear, they are easy to detect thanks to data correlations. These type of relations have to be eliminated when re-expressing the data set because we are looking also for a dimensionality reduction: if information can be expressed in less features that the ones being measures that signifies an improvement so we are removing the meaningless part of the data. All these observations are shown in a trivial example at Figure 3.13.

Once these considerations have been made, conditions to be imposed to $\mathbf{Y}$ are clear: we are looking for an expression of the data set in which each sample component contributes giving new information, as if each type of measure was independent to the others. That can easily be summarized by imposing the covariance matrix of the data set expressed in the new basis to be diagonal. If expressing the covariance matrix $\mathbf{C_Y}$ in function of the initial data set $\mathbf{X}$ we find the following relation:

$$\mathbf{C_Y} \doteq \frac{1}{n-1}\mathbf{Y} \cdot \mathbf{Y}^T = \frac{1}{n-1}(\mathbf{P} \cdot \mathbf{X}) \cdot (\mathbf{P} \cdot \mathbf{X})^T = \mathbf{P} \cdot \left(\frac{1}{n-1}\right) \mathbf{X} \cdot \mathbf{X}^T \cdot \mathbf{P}^T = \mathbf{P} \cdot \mathbf{C_X} \cdot \mathbf{P}^T$$

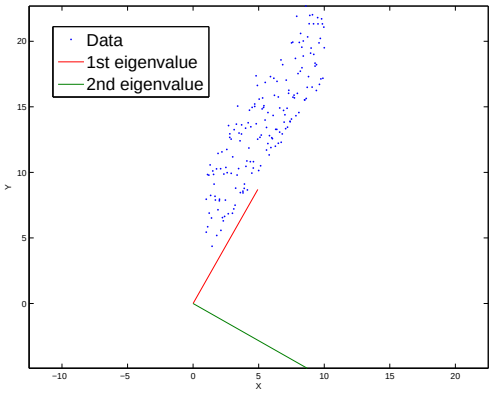

Figure 3.13: In this example, we have got data generated synthetically from a straight line affected with some random noise. As said, PCA re-expresses basis so that dynamics are best shown. In this case new vectors are shown in red and green (modulus has been modified so that they can be distinguished in the image). Decomposing data in the new axis, one can easily suggest that dynamics are only in one direction and that the variations on the second direction are only due to noise.

Moreover, without affecting to the calculations, rows of vectors in $\mathbf{P}$ are ordered so that variance in a dimension is greater than the variances of the following ones. That way, first vector directions have more importance than the last ones. Since now, assumptions made for PCA method have been:

(i) *Linearity:* otherwise we won't be looking only for an optimal change of basis, $\mathbf{P}$.
(ii) *Mean and variance as sufficient statistics:* conditions imposed rely only on the covariance matrix. But, unless Gaussian distributions, that isn't true.
(iii) *Dynamics relying on large variance components:* we assume the data information to be higher than noise level.

Essentially, what is left is to see how $\mathbf{P}$ is when $\mathbf{C_Y}$ is diagonal. Since a covariance matrix is symmetric, it always diagonalizes. Let $\mathbf{C_X}$ be decomposed as $\mathbf{C_X} = \mathbf{EDE}^T$, where D is diagonal and E is the matrix of eigenvectors of $\mathbf{C_X}$ (if rank of $\mathbf{C_X}$, r, is less than m, $\mathbf{E}$ can be completed by adding $(m - r)$ additional orthogonal vectors, corresponding to the eigenvalues equal to 0). Taking $\mathbf{P} = \mathbf{E}^T$ we obtain:

$$\begin{aligned} \mathbf{C_Y} &= \mathbf{P} \cdot \mathbf{C_X} \cdot \mathbf{P}^T = (\mathbf{E}^T) \cdot (\mathbf{EDE}^T) \cdot (\mathbf{E}^T)^T = \mathbf{E}^T\mathbf{EDE}^T\mathbf{E} \\ &= \{\mathbf{E}^{-1} = \mathbf{E}^T, \text{ since } \mathbf{E} \text{ is an orthogonal matrix}\} = \mathbf{Id} \cdot \mathbf{D} \cdot \mathbf{Id} = \mathbf{D} \end{aligned}$$

So choosing $\mathbf{P}$ that way, the covariance matrix $\mathbf{C_Y}$ diagonalizes.
In practice, before calculating $\mathbf{C_X}$ the mean of each measurement type is subtracted:

$$\mathbf{X} = \mathbf{X} - [\mathbf{x}_0, \mathbf{x}_0, \ldots^{(n)}, \mathbf{x}_0] \text{ where } \mathbf{x}_0 = \begin{pmatrix} E_[x_{i1}, i = 1 \div n] \\ \vdots \\ E[x_{im}, i = 1 \div n] \end{pmatrix}$$

Notice that principal components of data set $\mathbf{X}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^T$ or the rows of $\mathbf{P}$. Also, the i-th diagonal value of $\mathbf{C_Y}$ is the variance of $\mathbf{X}$ along the eigenvector $\mathbf{p}^i$ (ith row vector of $\mathbf{P}$).

PCA in Our Algorithm

We initially wanted to use PCA so that meshes' selection was done by fitting a Gaussian of variance σ_i, where $\mathbf{D} = diag(\sigma_1, \ldots, \sigma_m)$ along each direction i and then taking samples from these distributions to construct the N meshes used when training. But our meshes have distances constraints (distance between vertices must be maintained along the deformations) and PCA cannot model dimensionality reduction nor representation of data under restrictions.

Nevertheless, our approach uses PCA in two different ways.

First, all meshes are centered at same point (mean substraction) and then PCA is used to compute principal components of each mesh. The calculus of the principal components of a concrete mesh gives an insight of which is the best approximation for the mesh normal (like considering best plane that could be fitted in the mesh). The normal vector, $\vec{n}$, is the third principal vector when considering matrix data $\mathbf{X}$ as

$$\mathbf{X} = \begin{pmatrix} x_1 & x_2 & \ldots & x_{N_y \times N_x} \\ y_1 & y_2 & \ldots & y_{N_y \times N_x} \\ z_1 & z_2 & \ldots & z_{N_y \times N_x} \end{pmatrix},$$

where the mesh, $\mathbf{X}$, has the 3D information of the $N_y \times N_x$ vertices. Once found $\vec{n}$, $\mathbf{X}$ is modified into $\mathbf{X}'$ which is obtained by rotating the mesh so that $\vec{n} \| [\ 0\ 0\ 1\ ]$. Process is the following:

- PCA gives 3 eigenvectors, $\vec{u}_1$, $\vec{u}_2$ and $\vec{u}_3$. As said, normal vector is considered to be $\vec{u}_3$ normalized, $\vec{n} = \frac{\vec{u}_3}{|\vec{u}_3|}$.
- To verify that $\vec{n}$ is going out from the plate in the same direction (same side of the mesh) for all meshes we define: $\vec{a}_1 \doteq \overrightarrow{(1,1)(N,1)}$, $\vec{a}_2 \doteq \overrightarrow{(1,1)(\lceil N/2 \rceil, \lceil M/2 \rceil)}$ and $\vec{a}_3 \doteq \vec{a}_1 \times \vec{a}_2$, and check that $\vec{a}_3 \cdot \vec{n} > 0$. Otherwise we will use $-\vec{n}$ instead of $\vec{n}$ as $\vec{u}_3$ and change the sign of the other $\vec{u}_i$ so basis is still a direct one.
- Rotation to be applied to mesh $\mathbf{X}$ is such that transforms basis $\{\vec{u}_1, \vec{u}_2, \vec{u}_3\}$ into the standard $\{\vec{e}_1, \vec{e}_2, \vec{e}_3\}$. That implies

  $$\mathbf{R} = [\vec{e}_1, \vec{e}_2, \vec{e}_3][\vec{u}_1, \vec{u}_2, \vec{u}_3]^{-1} = [\vec{u}_1, \vec{u}_2, \vec{u}_3]^{-1}$$

  and therefore,

  $$\mathbf{X}' = \left[\mathbf{R} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}, \ldots, \mathbf{R} \cdot \begin{pmatrix} x_{N_y \times N_x} \\ y_{N_y \times N_x} \\ z_{N_y \times N_x} \end{pmatrix}\right]$$

We proceeded this way because we wanted to limit vision angles between camera and meshes' normal vectors. In a first approximation, camera was placed at the normal direction of the meshes with angle variation equal to zero (the z-axis). An example can be seen at Figure 3.14.

The second use of PCA in our algorithm appears when selecting the set of training meshes. This can be seen in the following section.

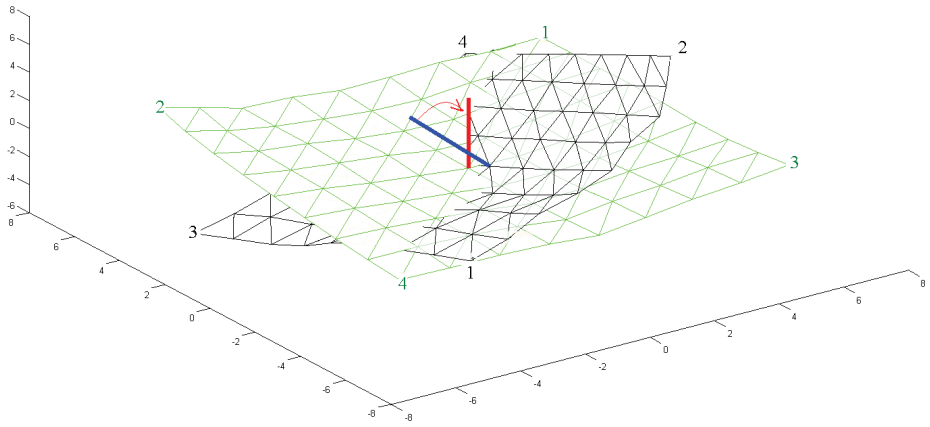

Figure 3.14: Mesh before and after reorientation. Mesh before the process is in black while its normal vector is in blue. Otherwise, mesh after reorientation is green and the normal vector, in red, is parallel to the Z-axis as we have imposed. Also, mean has been subtracted before and after reorientation

3.4.3 Meshes' Selection

A simple idea we used for selecting the set of training meshes was to consider a subset of a bigger set of deformed meshes. To construct the set so that chosen meshes were the most representative ones, we used a recursive method based on the PCA description of the meshes. Having taken the little meshes created from the big ones, and once centered by subtracting meshes' mean and all reoriented so that normal vector is $\vec{n} = (\ 0\ 0\ 1)$ direction, as explained before, first thing is to calculate PCA basis where to express the whole meshes' set.

Let $\mathbf{D}$ be the whole meshes' set where each column represents a mesh in xyz coordinates and $\mathbf{D}'$ the same set represented by new basis obtained from PCA:

$$\mathbf{D} = \begin{pmatrix} x^1_{(1,1)} & x^2_{(1,1)} & \cdots & x^{|D|}_{(1,1)} \\ y^1_{(1,1)} & y^2_{(1,1)} & \cdots & y^{|D|}_{(1,1)} \\ z^1_{(1,1)} & z^2_{(1,1)} & \cdots & z^{|D|}_{(1,1)} \\ \vdots & \vdots & \vdots & \vdots \\ x^1_{(N_y,N_x)} & x^2_{(N_y,N_x)} & \cdots & x^{|D|}_{(N_y,N_x)} \\ y^1_{(N_y,N_x)} & y^2_{((N_y,N_x)} & \cdots & y^{|D|}_{(N_y,N_x)} \\ z^1_{(N_y,N_x)} & z^2_{(N_y,N_x)} & \cdots & z^{|D|}_{(N_y,N_x)} \end{pmatrix}, \quad \mathbf{D}' = \begin{pmatrix} a^1_1 & a^2_1 & \cdots & a^{|D|}_1 \\ a^1_2 & a^2_2 & \cdots & a^{|D|}_2 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ a^1_{3\cdot N_y\cdot N_x-1} & a^2_{3\cdot N_y\cdot N_x-1} & \cdots & a^{|D|}_{3\cdot N_y\cdot N_x-1} \\ a^1_{3\cdot N_y\cdot N_x} & a^2_{3\cdot N_y\cdot N_x} & \cdots & a^{|D|}_{3\cdot N_y\cdot N_x} \end{pmatrix},$$

Then, we take the first two rows on $\mathbf{D}'$, which, remember, correspond to the projections of each of the $|D|$ meshes on the two eigenvectors with highest eigenvalues on the PCA representation. From these values, we create a 3-dimensional histogram of ten bins along each direction, which gives a total of a hundred bins. That is, we have the 3D-histogram:

$$\mathbf{H}^{1,2} = (h^{1,2}_{i,j})_{i,j\in\{1,\ldots,10\}}, \qquad \text{with } h^{1,2}_{i,j} = |\{(a^k_1, a^k_2) \in I^1_i \times I^2_j\}|,$$

$$\text{where, defining } m_t \doteq \min\{a^1_t, \ldots, a^{|D|}_t\},\ M_t \doteq \max\{a^1_t, \ldots, a^{|D|}_t\}, \text{ and } u \doteq \tfrac{M_t - m_t}{10},$$

$$I^t_s \text{ are the intervals: } I^t_s = \begin{cases} [m_t + (s-1)\cdot u, m_t + s\cdot u), & s = \{1,\ldots,9\} \\ [m_t + (s-1)\cdot u, m_t + s\cdot u], & s = 10 \end{cases}$$

Let N be the cardinal of the training set. We assign a number $n^{1,2}_{i,j}$ of meshes to be extracted from the meshes at bin $h^{1,2}_{i,j}$, restricted to constraint, $\sum_{i,j} n^{1,2}_{i,j} = N$. Numbers $\{n^{1,2}_{i,j}\}$ are assigned according to an algorithm that takes into account the weight of $h^{1,2}_{i,j}$ into $\mathbf{H}^{1,2}$. More precisely, let $w^{1,2}_{10\cdot(i-1)+j}$ be weights of bin $h^{1,2}_{i,j}$, $w^{1,2}_{10\cdot(i-1)+j} \doteq \frac{h^{1,2}_{i,j}}{|D|}$. Notice $\sum_{i,j} w^{1,2}_{10\cdot(i-1)+j} = 1$. And take N random values, $u_1, \ldots, u_N \sim U[0,1]$. Then, $n^{1,2}_{i,j}$ is defined as:

$$n^{1,2}_{i,j} = \left| \left\{ u_k, u_k \in \left[\sum_{m\le i, n<j} w^{1,2}_{10\cdot(m-1)+n} \quad , \quad \sum_{m\le i, n\le j} w^{1,2}_{10\cdot(m-1)+n} \right) \right\} \right|$$

That way, the quantity of meshes selected per bin is taken randomly but according to initial distribution of the created meshes D' on the PCA decomposition basis.

For $n^{1,2}_{i,j} = 1$ we take, as sample for the training set, mesh $\mathbf{D}^k = (a^k_1, \ldots, a^k_{3\cdot N\cdot M})^T$ (a column of set $\mathbf{D}'$) such that its a^k_1 and a^k_2 values are the closest to the mean values of intervals I^1_i and I^2_j respectively for all meshes belonging to $h^{1,2}_{i,j}$. For $n^{1,2}_{i,j} = h^{1,2}_{i,j}$, we take all meshes in $h^{1,2}_{i,j}$ as part of the training subset. For $n^{1,2}_{i,j} > h^{1,2}_{i,j}$, we take all meshes in $h^{1,2}_{i,j}$ as part of the training subset and also repeat picking meshes from $h^{1,2}_{i,j}$ until we have got $n^{1,2}_{i,j}$ meshes more in the training subset. For $n^{1,2}_{i,j} < h^{1,2}_{i,j}$, we iteratively repeat process explained above taking also third and fourth components, that is:

- Set to be considered now is $\mathbf{D'} \leftarrow \{\mathbf{D}^k \in \mathbf{D'} \cap h_{ij}^{1,2}\}$
- $N \leftarrow n_{i,j}^{1,2}$
- 3D-histogram is $\mathbf{H}^{3,4} = (h_{p,q}^{3,4})_{p,q\in\{1,\ldots,10\}}$ where $h_{p,q}^{3,4} = |\{(a_3^k, a_4^k) \in I_p^3 \times I_q^4\}|$. Intervals are defined as above but only considering meshes in the redefined set $\mathbf{D'}$.
- Following same algorithm as before, $n_{p,q}^{3,4}$ are computed, where $\sum_{1\leq p,q\leq 10} n_{p,q}^{3,4} = N$.
- Meshes $\mathbf{D}^k$ are selected from each bin having components $(a_1^k, a_2^k, a_3^k, a_4^k) \in I_i^1 \times I_j^2 \times I_p^3 \times I_q^4$

If $n_{p,q}^{3,4} < h_{p,q}^{3,4}$, process is repeated again by taking the fifth and sixth components and so on. If necessary, process stops when arriving at last components: $\{3N_yN_x - 1, 3N_yN_x\}$ if $3N_yN_x$ is even or $\{3N_yN_x - 2, 3N_yN_x - 1\}$ otherwise.

Figure 3.15 shows an example of the selection process.

3.5 Classes' Elements Image Synthesis

Once meshes have been created and interest points located, the next step is to synthesize the local images around the interest points to every deformed mesh of the training set. To do so and since meshes are no longer planar, we have had to consider a visibility model to distinguish among the visible and the occluded parts when there exists overlapping from the camera point of view. After the visibility model has been applied, synthesis is performed for every keypoint and mesh. This synthesis is done through the characterization of the pixel in terms of baryncentric coordinates, as it is explained below. Finally, all patches are oriented so that the different synthetic images of the same keypoint are aligned. All these steps are explained in more detail in the following sections.

3.5.1 Barycentric Coordinates

Any point $\mathbf{P} = (P_x, P_y, P_z)^T$ belonging to a triangle $\triangle$ with vertices $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ can be reexpressed as a linear combination of the triangle vertices coordinates. That is, $\mathbf{P} = \sum_{i=1}^3 \alpha_i \mathbf{v}_i$. Weights $\{\alpha_i\}_{i=1,2,3}$ are restricted to constraint $\sum_{i=1}^3 \alpha_i = 1$ and we take $\mathbf{P}_{bar} = (\alpha_1\ \alpha_2\ \alpha_3)^T = \alpha$. This coordinate system is called the *barycentric coordinates system* and α are the *barycentric coordinates* of point $\mathbf{P}$.

In order to know α, we build the equations corresponding to $\mathbf{P}$'s coordinates and the constraint:

$$\begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix} = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_{1x} & v_{2x} & v_{3x} \\ v_{1y} & v_{2y} & v_{3y} \\ v_{1z} & v_{2z} & v_{3z} \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \mathbf{V}\alpha$$

And we operate as follows:

$$\mathbf{P} = \mathbf{V}\alpha \Rightarrow \mathbf{V}^T\mathbf{P} = \mathbf{V}^T\mathbf{V}\alpha,\ (\mathbf{V}^T\mathbf{V})_{3\times 3} \Rightarrow \alpha = (\mathbf{V}^T\mathbf{V})^{-1}\mathbf{V}^T\mathbf{P} = \mathbf{V}^\dagger\mathbf{P},$$

where $\mathbf{V}^\dagger \doteq \mathbf{V}^{-1}\mathbf{V}^{-T}\mathbf{V}^T$ is the *pseudoinverse* of $\mathbf{V}$.

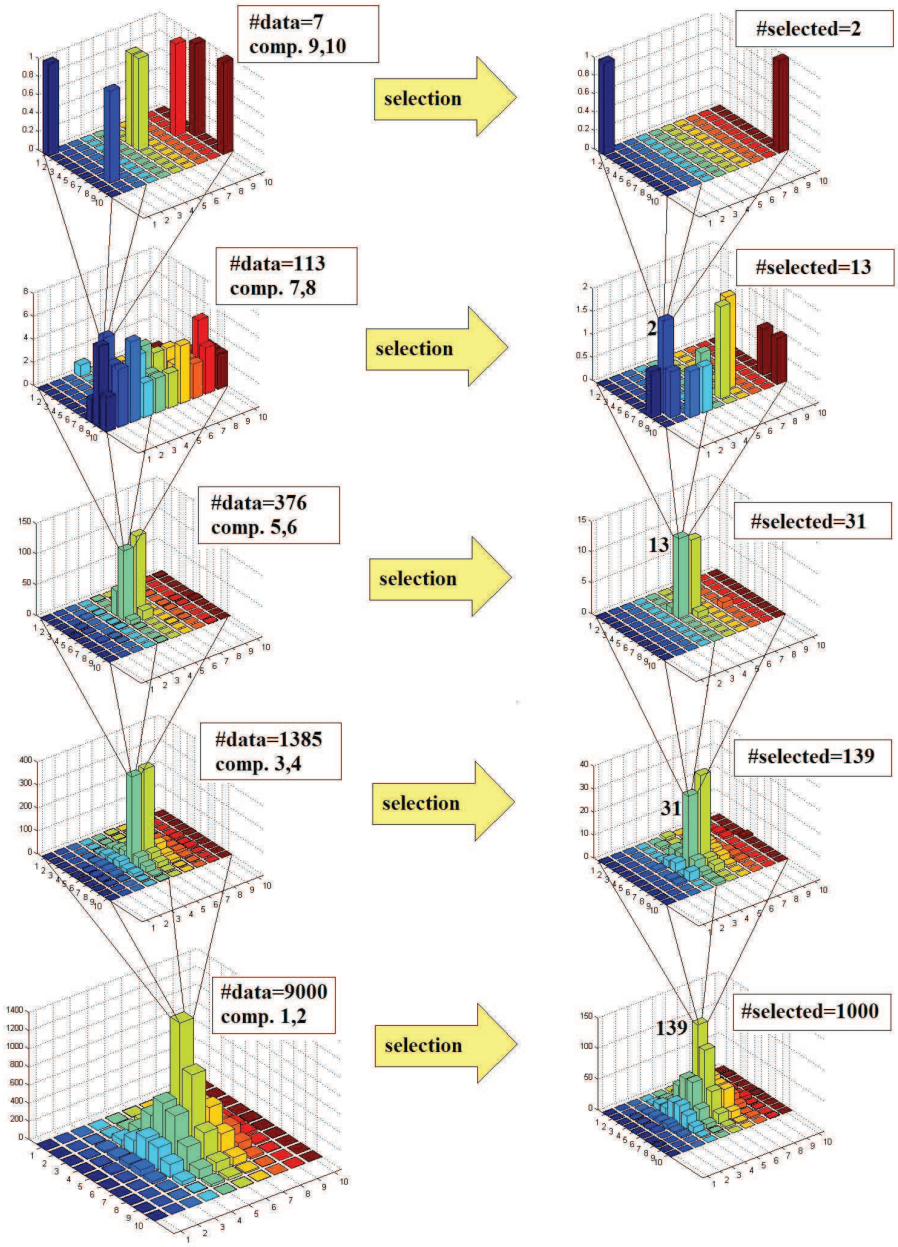

Figure 3.15: Example of the process selection. A total of 9000 samples have been used. Each sample is 75-dimensional. Thousand samples from the set have to be selected. At bottom and at left, distribution into 10 classes per dimension and for 1st and 2nd components of all data set is shown. At bottom right, distribution of the number of samples per class in the training set is shown. Notice distribution on the number of training samples is more or less the same of the total (remember selection method). The total amount of samples selected is, as said, 1000. At each bin, subset is reconsidered by taking 3rd and 4th components and classifying into 100 classes more. Once again, selection process is applied with the new weights according to the subset distribution in the 3rd and 4th components (this is the second row beginning at bottom from the image). Process is repeated at each level until data set in the each bin is equal or greater than the number of samples to be extracted or only one sample is to be selected. In that case, we choose the sample from the data set the closest to the center of the bin. In this figure, levels are shown for a concrete example.

In fact, any point $\mathbf{P}$ in the 3D space can be reexpressed in barycentric coordinates even if it is not contained in the triangle defining the barycentric coordinates. When $\mathbf{P}$ is within triangle boundaries, then $0 < \alpha_i < 1\ \forall i = 1,2,3$; when $\mathbf{P}$ lies in one of the edges, then $\alpha_i = 0$ for $i \in \{1,2,3\}$ and the other two $\alpha_j, \alpha_k \in (0,1)$; finally, when $\mathbf{P}$ is a vertex of the triangle, say vertex $\mathbf{v}_i$, then $\alpha_i = 1, \alpha_j = \alpha_k = 0$, $i \neq j \neq k$, and $i,j,k \in \{1,2,3\}$. Finally, if $\mathbf{P}$ is outside the triangle, then $\exists i \in \{1,2,3\}$ such that $\alpha_i < 0$.

Barycentric Coordinates and Image Synthesis in Our Algorithm

In order to synthesize images of the deformed meshes from a reference planar mesh, we use the barycentric coordinates as follows:

- Every pixel in the reference mesh is labeled according to the triangle to which it belongs. The mesh is then represented as

$$M \doteq \{((u_i,v_i), I_i, \triangle_i)\ \forall i \text{ pixel in the mesh}, \quad (u_i,v_i)\ uv\text{-coordinates},\ I_i \text{ pixel intensity}\ \triangle_i \text{ face to which pixel belongs}\}$$

- The reference mesh is assumed to be at a certain known distance f from camera and barycentric coordinates are then calculated for every pixel.

$$\begin{aligned} M' &\doteq \{((fu_i, fv_i, f), I_i, \triangle_i)\ \forall i \text{ pixel in the reference mesh}\} \\ M'_{bar} &\doteq \{(\alpha, I_i, \triangle_i)\ \forall i \text{ pixel in the reference mesh}, \alpha = \mathbf{V}^{\dagger}_{\triangle_i} \cdot (fu_i, fv_i, f)^T, \\ &\quad \mathbf{V}_{\triangle_i} \doteq \begin{pmatrix} \mathbf{v}_1^i & \mathbf{v}_2^i & \mathbf{v}_3^i \\ 1 & 1 & 1 \end{pmatrix}, \triangle_i = \triangle(\mathbf{v}_1^i, \mathbf{v}_2^i, \mathbf{v}_3^i))\} \end{aligned}$$

- New pixel positions are obtained when using the barycentric coordinates of each pixel with the new vertices coming from the synthetic mesh. Remember that synthetic meshes were defined by the 3D position of its vertices.Also, the pixels in the new positions have same gray intensity as in the reference mesh.

$$\begin{aligned} M_{syn,0} &= \{\ (\tilde{U}_i, \tilde{V}_i, \tilde{W}_i), I_i, \triangle_i),\ \forall i \text{ pixel in the reference mesh},\ (\tilde{U}_i\ \tilde{V}_i\ \tilde{W}_i\ 1)^T = \tilde{\mathbf{V}}_{\triangle_i}\alpha \\ &\quad \tilde{\mathbf{V}}_{\triangle_i} = \begin{pmatrix} \tilde{\mathbf{v}}_1^i & \tilde{\mathbf{v}}_2^i & \tilde{\mathbf{v}}_3^i \\ 1 & 1 & 1 \end{pmatrix}, \triangle_i = \triangle(\tilde{\mathbf{v}}_1^i, \tilde{\mathbf{v}}_2^i, \tilde{\mathbf{v}}_3^i)\ \} \end{aligned}$$

- 3D interpolation is used to homogenize gray colors in the mesh (see Figure 3.16).

$$\begin{aligned} M_{syn,1} &= \{(\tilde{U}_j, \tilde{V}_j, \tilde{W}_j), I_j),\ \forall j \text{ pixel in the synthetic mesh,} \\ &\quad \text{obtained interpolating subset } M_{syn,0}\} \end{aligned}$$

- Finally, the 3D pixel coordinates are projected onto the image by means of the calibration matrices $\mathbf{A}$ and $(\mathbf{R}|\mathbf{T})$. Also, pixels are rounded to be integers.

$$M_{syn,2} = \left\{(\tilde{u}_j, \tilde{v}_j), I_j),\ (\tilde{u}_j, \tilde{v}_j) = \left(round\left(\frac{\tilde{U}_j}{\tilde{W}_j}\right), round\left(\frac{\tilde{V}_j}{\tilde{W}_j}\right)\right)\right\}$$

So, as it can be seen, the barycentric coordinates are perfect to transport in a very efficient way, gray intensities from a reference mesh to another one created synthetically.

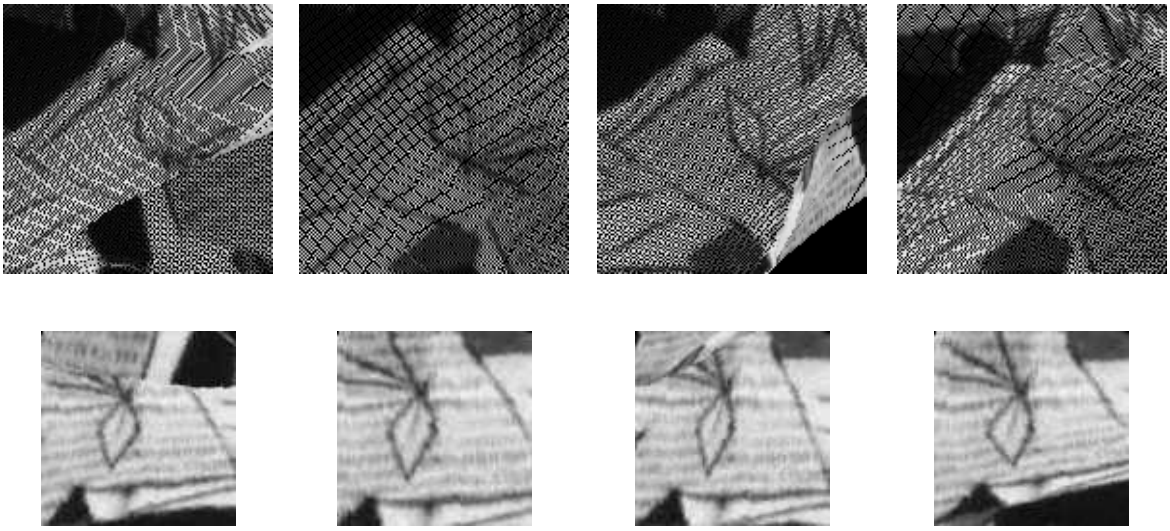

Figure 3.16: Patch is initialized at 0 for every pixel (that is black colored). When all pixels in reference mesh are projected into the deformed mesh via the barycentric coordinates, still a lot of pixel remain black (first row). Intensity interpolation (second row) gives the solution. Here, interpolated patches have gone through the visibility model and have also been reoriented and cut at the precise specified size.

3.5.2 Visibility Model

Since meshes are deformed and some parts of the mesh may be occluded by others from the camera viewpoint, it is very important to be able to determine the visible parts of the synthesized mesh. In a more technical way, occlusion is defined by different points in the 3D mesh which have same coordinates (u, v) in the 2D image.

Our model to resolve such ambiguity departs from the external and internal calibration matrices, the 2D coordinates of the dense synthetic mesh from the training set and also its 3D vertices coordinates and faces information. Let's explain it.

Let $\mathbf{A}$ be the internal calibration matrix and $(\mathbf{R}|\mathbf{T})$ the external one. Let $\mathbf{P}_{(1,1)}, \ldots, \mathbf{P}_{(N_y,N_x)}$ be the world coordinates of the $N_y \times N_x$ vertices of the synthetic mesh being examined. Moreover, let $\mathcal{D} = \{\triangle_1, \ldots, \triangle_{2(N_y-1)(N_x-1)}\}$ be the set of faces where $\triangle_i$ is a 3×1 vector containing the three vertices defining the i-th triangle, for example $\triangle_1 = \begin{pmatrix} \mathbf{P}_{(1,2)} & \mathbf{P}_{(1,1)} & \mathbf{P}_{(2,1)} \end{pmatrix}$, vertices ordered the same way we did in Section 3.4.1. Let $\mathbf{U}_{Q\times 2}$ be a list of Q coordinates (u, v) belonging to the mesh. More concretely, they correspond to the projection of the 3D coordinates in the synthetic mesh obtained from the dense reference mesh through the barycentric coordinates. And finally, let $\mathbf{I}_{Q\times 2}$ be the list of the gray intensities corresponding to each uv-coordinate in $\mathbf{U}_{Q\times 2}$.

The main idea is to examine each point (u, v) in the $\mathbf{U}$ list by discovering if there is more than one triangle intersecting with a ray born at $\mathbf{CAM}$, camera point in the world coordinates system, and intersecting with the image at same coordinates (u, v). If that is so, from all the

points found, algorithm only returns 3D coordinates of the closest points and its associated gray intensity.

In a technical way, the algorithm is described in Table 3.1.

$\mathbf{U}' = \emptyset$ and $\mathbf{I}' = \emptyset$. Given $((u,v), I)$ in $\mathbf{U} \times \mathbf{I}$ do:
[1] Consider, arbitrarily, $Z_{cam} = K$, $K = 8$ for example.
[2] Let $\mathbf{P}$ be a 3D point with coordinates $\mathbf{P}_{cam} = \begin{pmatrix} X_{cam} & Y_{cam} & Z_{cam} \end{pmatrix}^T$ in the camera coordinate system and coordinates $\mathbf{P} = \begin{pmatrix} X & Y & Z \end{pmatrix}^T$. Calculate:

$$\begin{pmatrix} K \cdot u \\ K \cdot v \\ K \end{pmatrix} = \mathbf{A} \begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{pmatrix} \Rightarrow \mathbf{P}_{cam} = \mathbf{A}^{-1} \begin{pmatrix} K \cdot u \\ K \cdot v \\ K \end{pmatrix}$$

And, $\mathbf{P}_{cam} = \mathbf{RP} + \mathbf{T} \Rightarrow \mathbf{P} = \mathbf{R}^{-1}(\mathbf{P}_{cam} - \mathbf{T})$
[3] Calculate camera viewpoint, which is, as said before, $\mathbf{CAM} = -\mathbf{R}^{-1}\mathbf{T}$ in the world coordinates system
[4] Define the normalized projection ray from camera to point $\mathbf{P}$, that is: $\mathbf{ray} = \left(\frac{\mathbf{P-CAM}}{||\mathbf{P-CAM}||}\right)$
[5] $\mathcal{DE}_{((u,v),I)} = \emptyset$. For all triangle $\triangle = \begin{pmatrix} \mathbf{B} & \mathbf{A} & \mathbf{C} \end{pmatrix}$ in $\mathcal{D}$ do:
[5.1] Calculate edges $\overrightarrow{\mathbf{BA}}$ and $\overrightarrow{\mathbf{CA}}$.
[5.2] Calculate intersection point, $\mathbf{J}_{\triangle} = \begin{pmatrix} J_t & J_u & J_v \end{pmatrix}^T$ between the line defined by ray and the plane defined by $\triangle$. It can be demonstrate that intersection is the solution of system:

$$\begin{pmatrix} \mathbf{CAM}_x - \mathbf{A}_x \\ \mathbf{CAM}_y - \mathbf{A}_y \\ \mathbf{CAM}_z - \mathbf{A}_z \end{pmatrix} = \begin{pmatrix} \mathbf{ray}_x & \overrightarrow{\mathbf{BA}}_x & \overrightarrow{\mathbf{CA}}_x \\ \mathbf{ray}_y & \overrightarrow{\mathbf{BA}}_y & \overrightarrow{\mathbf{CA}}_y \\ \mathbf{ray}_z & \overrightarrow{\mathbf{BA}}_z & \overrightarrow{\mathbf{CA}}_z \end{pmatrix} \begin{pmatrix} J_t \\ J_u \\ J_v \end{pmatrix}$$

[5.3] If point $\mathbf{J}_{\triangle}$ has $J_u, J_v \in [0,1]$ and also $J_u + J_v \leq 1$, do $\mathcal{DE}_{((u,v),I)} \leftarrow [\triangle, \mathbf{I}_{\triangle}] \cup \mathcal{DE}_{((u,v),I)}$. These are the triangles that have intersection point between plane they define and projection ray within their boundaries.
[6] Among points in $\mathcal{DE}_{((u,v),I)}$. Find $[\triangle^0, \mathbf{J}_{\triangle^0}] = [(\mathbf{B}^0\ \mathbf{A}^0\mathbf{C}^0)\ ,\ (J_t^0\ J_u^0,\ J_v^0)]$, such that
$J_t^0 = \min\{J_t, [(\mathbf{B}\ \mathbf{A}\ \mathbf{C})\ ,\ (J_t\ J_u,\ J_v)] \in \mathcal{D}_{((u,v),I)}\}$.
[7] If $\mathbf{P} \in \triangle^0$, do $\mathbf{U}' \leftarrow (u,v) \cup \mathbf{U}'$ and $\mathbf{I}' \leftarrow I \cup \mathbf{I}'$.

Table 3.1: Visibility model algorithm.

The explanation has been adapted to our case but the generalization is easy. The list of vertices coordinates for the triangular surfaces and the list of triangles can be from more than one object and not necessarily from a unique mesh. Procedure is exactly the same. In Figure 3.17 we show an example of the visibility model which isn't referring to a mesh. Only some of the 2D points of $\mathbf{U}$, and consequently projection rays, have been used to see the results in a better way.

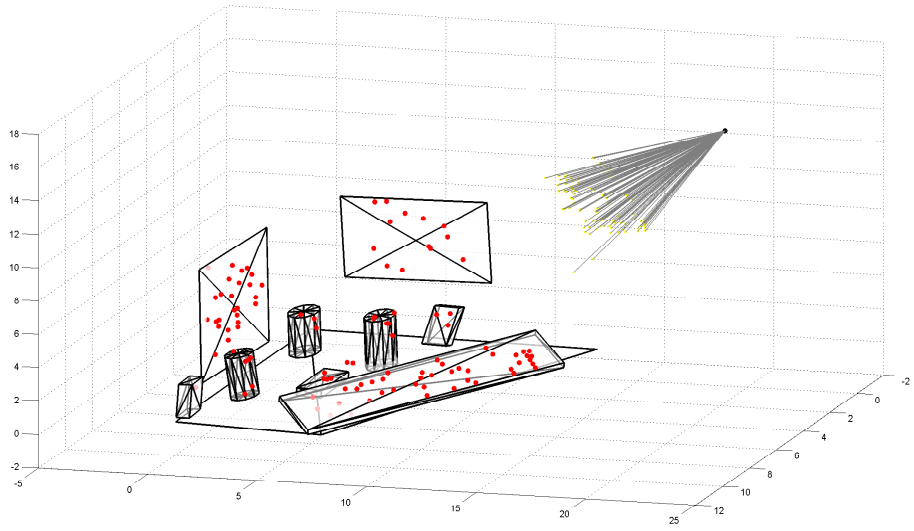

Figure 3.17: Visibility model general example. As it can be seen, faces do not correspond to a unique mesh but to all kind of objects.

Visibility Model in Our Algorithm

We first wanted to give some randomness to the position of the camera within the interest point being analyzed at each moment. That way, keypoints would be characterized by images obtained not only varying deformations but also the viewpoint. But this step of our approach model is certainly the slowest part of the algorithm because changing the camera viewpoint forces us to repeat the verification of visible parts not only for each mesh of the training set but also for each interest point being analyzed. That's why the camera was fixed (at a certain distance, perpendicular to the mesh at its center point) and a common analysis of the visibility was done for each training mesh independently of the interest point being trained.

The visibility model is applied before the interpolation step when synthesizing the image onto the training mesh. Only visible points lying in the set of uv-coordinates obtained from the reference mesh to the synthetic one through barycentric coordinates are used on the following set.

3.5.3 Images' Orientation

The synthetic meshes generated for training are created without taking into account any type of orientation further than aligning its normal vector. However, when training, as we will see in next section, it is vital that all image patches are oriented the same way. Remember, from the overview of the state of the art, that lots of methods use orientation gradient histograms over the interest region to obtain orientation invariance. That is what we are looking for in this

section.

The method we follow is very similar to the one explained in Section 3.3.1 when discussing orientation assignment on the SIFT detector used in the algorithm. We first obtain an initial orientation of the synthetic mesh and then rotate the whole mesh such that main orientation of the image patch is 0 degrees.

The initial data are pixels of the image patch. That set is $M = \{((u, v), I)\}$. We first we extract the scale at which interest point were detected from the SIFT detection process. Scale, s, gives us the significative region to consider around the interest point. We define $\sigma = 1.5 \cdot s$ and $W = 3 \cdot \sigma$. These values are the ones used at [18].

Then, we consider the subset N of uv-coordinates and pixel intensities from points within the region given by W in the following way: $N = \{((u, v), I) \in M$, such that $|u_c - u| \leq W$, and $|v_c - v| < W\}$, being (u_c, v_c) the mesh center point or, equivalently, the interest point. The set N changes through interpolation to the set $N' = \{((u, v), I), \forall (u, v) \in [u_c - W, u_c + W] \times [v_c - W, v_c + W] \cap \mathbb{Z}$, and I the corresponding one from interpolation$\}$. We have also image $\mathbf{I}_{(2W+1)\times(2W+1)}$ formed from data in N' sorted according to uv-coordinates.

We take image $\mathbf{I}_{(2W+1)\times(2W+1)}$ and add a Gaussian blur with $\sigma = 1.5$. Consider $\mathbf{I}_{blur}$ the blurred image. We then compute the gradient modulus for each pixel in the image, $\mathbf{G}_m = (m_{uv})_{u,v}$ and also gradient orientation $\mathbf{G}_\theta = (\theta_{uv})_{u,v}$. We also calculate the values of a gaussian window of size $(2W + 1) \times (2W + 1)$ and σ, which we note as matrix $\mathbf{GW} = (w_{uv})_{u,v}$. Next, we construct a 10-bin histogram considering all $\theta_{uv} \in \mathbf{G}_\theta$ weighted by product $w_{uv} \cdot m_{uv}$. The initial orientation assigned to the patch is the center angle ϕ from the highest bin of the orientation histogram.

Finally, we rotate the whole set of uv-coordinates. Rotation is done with an angle ϕ and center at point (u_c, v_c). That way, the output is set:

$$M' = \left\{ ((u', v'), I) \mid ((u, v), I) \in M, \begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} \begin{pmatrix} u - u_c \\ v - v_c \end{pmatrix} + \begin{pmatrix} u_c \\ v_c \end{pmatrix} \right\}$$

Images' orientation is also a step previous to final images' synthesis and it is posterior to the application of the visibility model. Figure 3.18 shows some examples of the orientation process.

3.6 Ferns' Training

As said at the beginning of this chapter, the way we solve the recognition of a deformable object is focusing on local features. These local features are not characterized by specific descriptors on both the reference and the test images as it is often. We turn the matching problem into a classification one.

Instead of matching local descriptors between the reference image and the input one, lots of different local images defining same feature are created and then, a training process learns how to discern between features. Next, the test image is compared to all trained features and it is classified as belonging to one of these features. The method for handling this classification problem is based on random ferns.

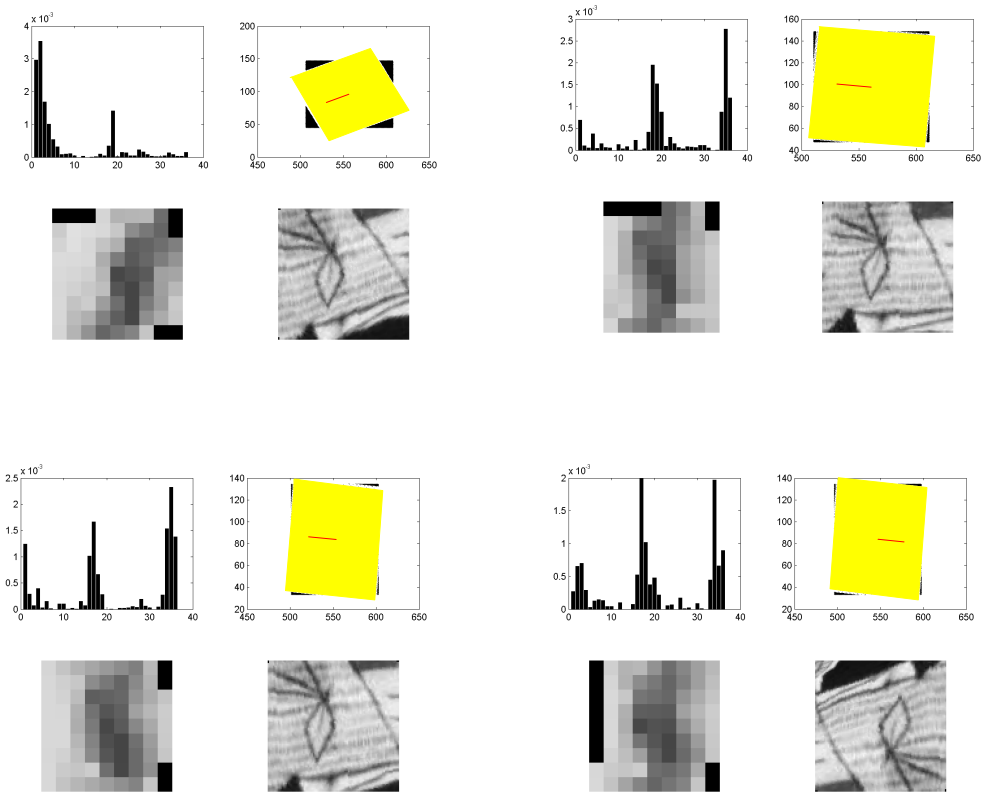

Figure 3.18: Four elements of the same class as example for the orientation process. At each image, the ten bin histogram is at left at top. At top right, mesh before and after the turn determined by the maximum bin in the histogram. At bottom left, window at center of the patch at which histogram is computed. Since it depends on the class, all the examples have the same width of window. Finally, at right bottom, image synthesized once it has already been oriented. Majority of the patches are oriented the same way. But we also have noticed that most of the errors come from the ambiguity of the 180 turn, as it happens at one of the examples shown here. This fact can also be appreciated at the different histograms, where, in general, peaks are in a region of bins and also 18 bins further (modulus 36).

Until now, all we have explained concerning to our algorithm are the previous steps to do the ferns' training. We have just constructed the classes where to apply the ferns classification method. That is, local features have been treated as classes and all the synthetic images created for each local feature serve as elements which represent the same class. The result of applying the theory of the former sections for the Guernica can be seen at Figure 3.19.

In this section, we first explain a more general method, the random forests in which ferns are based. After that, the method we have used, random ferns, is theoretically explained and also some information about the parameters we used is given.

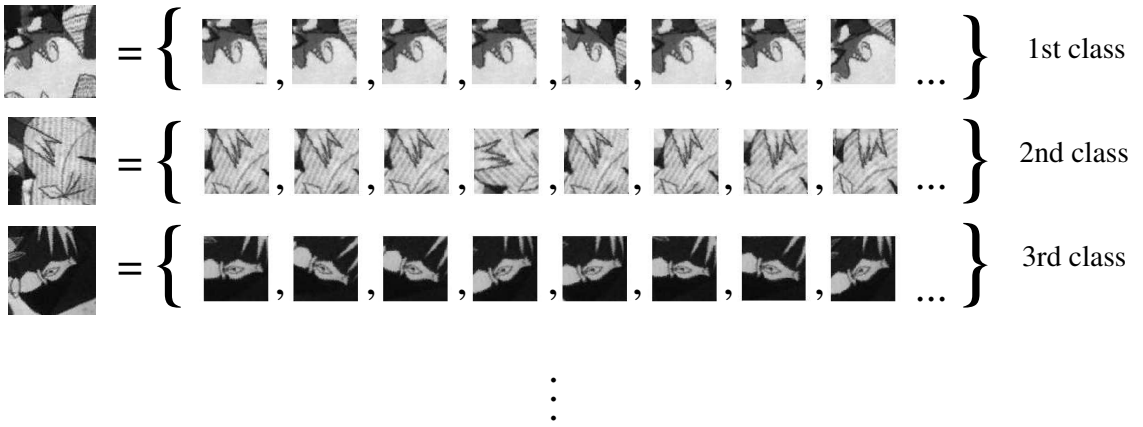

Figure 3.19: Each interest point is converted into a class when considering a local region around it. Synthesis of the deformed meshes around each interest point with the visibility model and orientation determination compose the elements of each class.

3.6.1 Formulation of the Classification Problem, [16],[15]

Let $\mathbf{K}$ be the set of classes used in the training process, $\mathbf{K} = \{\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_{N_{class}}\}$, with N_{class} the total number of classes (which is equivalent to say that N_{class} is the total number of interest points considered in the training stage). Each class, $\mathbf{k}_i$, is a set of $N_{training}$ synthesized deformed meshes, $\mathbf{k}_i = \{k_i^1, k_i^2, \ldots, k_i^{N_{training}}\}$, k_i^j a synthetic 2D patch.

Given a test image, we call k^{input} a test patch consisting in a local region around an interest point of the test image. The aim is to decide if k^{input} belongs to one of the classes of $\mathbf{K}$. In other words, we want to find the best class where k^{input} fits in, that is $\mathbf{k}_{\tilde{\imath}}$, with:

$$\tilde{\imath} = \underset{i=1,\ldots,N_{class}}{\operatorname{argmax}} \; P(\{k^{input} \in \mathbf{k}_i\})$$

In case, $P(\{k^{input} \in \mathbf{k}_{\tilde{\imath}}\})$ is low, that is the probability falls below a certain threshold, we can also consider that k^{input} does not belong to any of the classes in set $\mathbf{K}$.

3.6.2 Random Forests

Random forests classifier consists on a number of trees. A tree is built as follows: at root node a simple test on the training images (k_i^j, $i = 1, \ldots, N_{class}$, $j = 1, \ldots, N_{training}$) is run. That way, the total set is split into different subsets according to the response to the root's test. For all the other nodes, a simple test is run but only to the images arriving at that node. Every patch generated in the training, k_i^j, is classified by dropping it down the tree, responding to the node tests and arriving to one of the tree leaves. A simple example of a tree with node tests given randomly can be found at Figure 3.20.

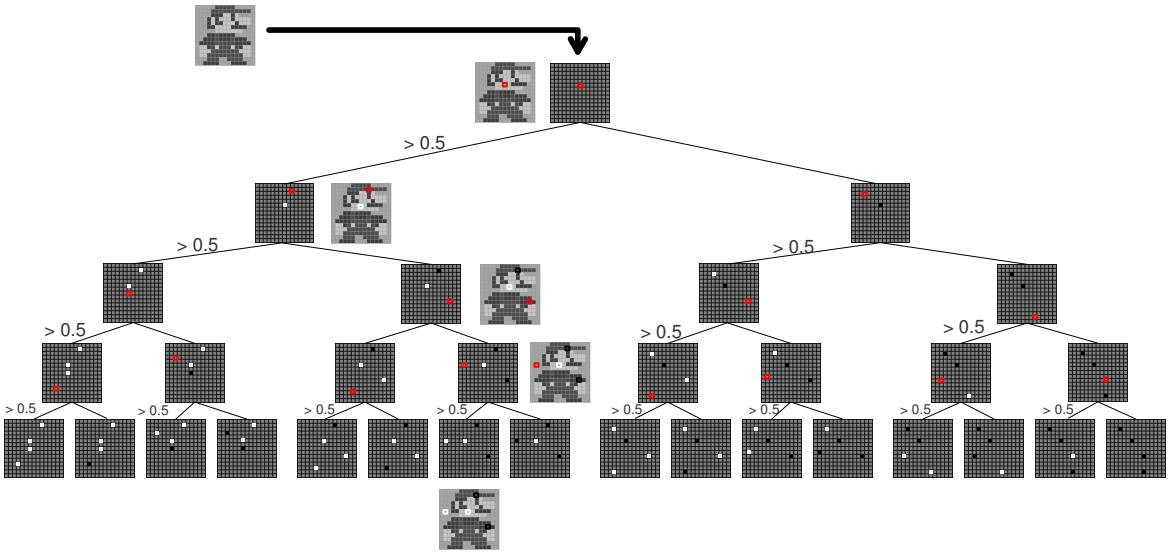

Figure 3.20: Tree example. Node tests of the tree consist on classifying a random pixel according to if its intensity value is greater than 0.5 or not. Also an input image is dropped down the tree until achieves one of the leaves.

At each leaf, the distribution of the training meshes that have arrived with respect to the class they belong to is kept. That way, for each leaf η, the posterior probabilities can be computed:

$$P(k^{input} \in \mathbf{k}_i \mid \text{ reached leaf } = \eta) = \frac{|\{k_i^j, j = 1, \ldots, N_{training}, \quad \text{leaf } \eta \text{ reached}\}|}{|\{k_h^j, j = 1, \ldots, N_{training}, \ h = 1, \ldots, N_{class}, \quad \text{leaf } \eta \text{ reached}\}|}$$

Then, taking into account each tree T_l, $l = 1, \ldots, L$, for a given k^{input} which has reached leaf η_l at tree T_l, the class which k^{input} is more likely to belong to is $\mathbf{k}_{\tilde{i}}$ with:

$$\tilde{\imath} = \underset{i=1,\ldots,N_{class}}{\operatorname{argmax}} \sum_{l=1}^{L} P(k^{input} \in \mathbf{k}_i \mid \text{ reached leaf } = \eta_l)$$

Thus, the estimation of the best classification is done by considering the average of the results achieved at each tree (which is not the only option).

Concerning to the way in which trees are grown, there exist two possibilities. The simplest one consists on randomly choosing the test among a previous tests' set (as Figure 3.20). Second one uses a greedy algorithm[3] which picks the test that best separates the given training subset arrived at that node. *Best*, here, consists on giving the maximum entropy to the partition originated by the responses of the data set to the test (we want that branches generated by the

[3]Greedy algorithm refers to the fact that at each node decision is just focused in the local best results in that node not globally.

answers to the test support more or less same quantity of data). When a node receives very few data from the training set, it becomes a leaf. Also, there is a fixed maximum depth for each tree which is given as a parameter.

Random forests classifier has been used in lots of applications, such as [1], [3], [16], etc. Nevertheless, most of the techniques relying in random forests have moved to random ferns usage since it has been proved that ferns outperform trees in several ways (classification rate, computational cost, handling more classes, etc) in [24].

3.6.3 Random Ferns

Random ferns are very similar to random forests in spirit but they have some important differences which make them a better option. The main one is that ferns aren't a hierarchical structure and this fact implies that posterior probabilities can be computed multiplicatively. This also has a positive effect in the algorithm speed.

Let's first remember our classification problem: for a given input patch, k^{input}, we want to find best i such that $k^{input} \in \mathbf{k}_i$. Like in random forests, we will decide the best option by running some tests, $f_t, t = 1, \ldots, T$. We are looking for:

$$\mathbf{k}_{\hat{\mathbf{i}}} = \underset{\mathbf{k}_i,\ i=1,\ldots,N_{class}}{\operatorname{argmax}} P(k^{input} \in \mathbf{k}_i \mid f_1, \ldots, f_T), \tag{3.2}$$

which is the probability that patch k^{input} belongs to class $\mathbf{k}_i$ once the result of the tests $f_1, \ldots, f_T$ is known.

Using Bayes' theorem, we obtain:

$$P(k^{input} \in \mathbf{k}_i \mid f_1, \ldots, f_T) = \frac{P(f_1, \ldots, f_T \mid k^{input} \in \mathbf{k}_i) P(k^{input} \in \mathbf{k}_i)}{P(f_1, \ldots, f_T)}$$

Assuming that the prior distribution for k^{input} belonging to a class is a uniform one (that is $P(k^{input} \in \mathbf{k}_i) = P(k^{input} \in \mathbf{k}_j) \quad i, j = 1, \ldots, N_{class}$) and since denominator is independent from the class, Equation (3.2) becomes:

$$\mathbf{k}_{\hat{\mathbf{i}}} = \underset{\mathbf{k}_i,\ i=1,\ldots,N_{class}}{\operatorname{argmax}} P(f_1, \ldots, f_T \mid k^{input} \in \mathbf{k}_i)$$

Up to this point, the formulation is pretty much the same that is used for random forests. What really changes in the random ferns is the definition of these tests. First of all, with random forests, tests are run in a top-down manner: a training patch k_i^j only receives the tests appearing in its way down to a leaf, while random ferns apply the same $f_1, \ldots, f_T$ test to all the training patches. Therefore, contrary to random forests, ferns have got a non-hierarchical structure. Another different fact is the tests' set. For random ferns, the tests are only comparisons between two pixels within the patch. Let's see it.

Assume $d_1^t = (u_1^t, v_1^t)$ and $d_2^t = (u_2^t, v_2^t)$ be two random positions within a patch and let I_1^t and I_2^t be their respective intensities. Then, the test f_t is defined as follows:

$$f_t = \begin{cases} 1 & \text{if } I_1^t > I_2^t \\ 0 & \text{otherwise} \end{cases}$$

The chosen tests are very simple, only based on binary comparisons, and any good characterization of the different classes will demand a high number of comparisons T large enough (for example, $T \approx 300$). Every patch could then be characterized with a value between 0 and $2^T - 1$ and this is very difficult to handle (because of its size). A first way to compress the different representation, the most naive one, is to assume independence between the different tests, then

$$P(f_1, \ldots, f_T \mid k^{input} \in \mathbf{k}_i) = \prod_{t=1}^{T} P(f_t \mid k^{input} \in \mathbf{k}_i)$$

But this point of view completely ignores the correlations between the different tests. A trade-off between making the problem tractable and taking into account the different correlations is to partition the test features into M groups of equal size S such that $T = M \cdot S$. Then, the conditional probability is computed as:

$$P(f_1, \ldots, f_T \mid k^{input} \in \mathbf{k}_i) = \prod_{m=1}^{M} P(F_m \mid k^{input} \in \mathbf{k}_i)$$

where F_m is called a *fern* and it is a group of S binary tests; $F_m = \{f_1^m, \ldots, f_S^m\}$, $f_s^m \in \{f_1, \ldots, f_T\}$ and $f_s^m = f_{s'}^{m'}$ iff $s = s', m = m'$.

This compression of data is based in what is called a semi-naive Bayesian approach which consists on modeling only some of the whole set of dependencies between features. Instead of characterizing classes with a distribution of 2^T possibilities, with ferns we obtain M distributions of 2^S possible cases each, which is much more tractable than the initial option. Usual values for S are 10, 12 or 14, and M is such that makes $T \approx 300$. Maintaining the parallelism with randomized forests, M is equivalent to the number L of trees and S would be comparable to the tree's depth.

The computation of $P(F_m \mid k \in \mathbf{k}_i)$ is not as trivial as one could think (see [24] for a theoretical development). Nevertheless, when considering the training set truly representative of the actual variations within a class (which is an assumption), then the probability can be computed as:

$$P(F_m = (f_1^m, \ldots, f_S^m),\ f_s^m \in \{0,1\},\ s = 1, \ldots, S \mid k \in \mathbf{k}_i) = \frac{1 + n^i_{(f_1^m, \ldots, f_S^m)}}{\sum_{F_m=(0,\ldots,0)}^{(1,\ldots,1)} \left(1 + n^i_{F_m}\right)}$$

This equation can be read as the probability that the fern F_m gets a value $(f_1^m, \ldots, f_S^m)$ for a patch k belonging to class $\mathbf{k}_i$ is the ratio between the number of patches k_i^j of class $\mathbf{k}_i$ getting fern value $F_m = (f_1^m, \ldots, f_S^m)$ (this is $n^i_{(f_1^m, \ldots, f_S^m)}$) plus one between the total number of patches k_i^j of class $\mathbf{k}_i$ getting any of the possible values for F_m (this is $F_m = (0, \ldots, 0), \ldots, (1, \ldots, 1)$) plus 2^S (equivalently, $\sum_{F_m=(0,\ldots,0)}^{(1,\ldots,1)} \left(1 + n^i_{F_m}\right)$).

Graphically, the computation of this probability is very easy and is shown at Figure 3.21. Given the fern F_m, an histogram with 2^S bins is created and each bin is initialized to 1. Given a class $\mathbf{k}_i$, for all patch k_i^j $(j = 1, \ldots, N_{training})$, the fern test F_m is run and a binary number

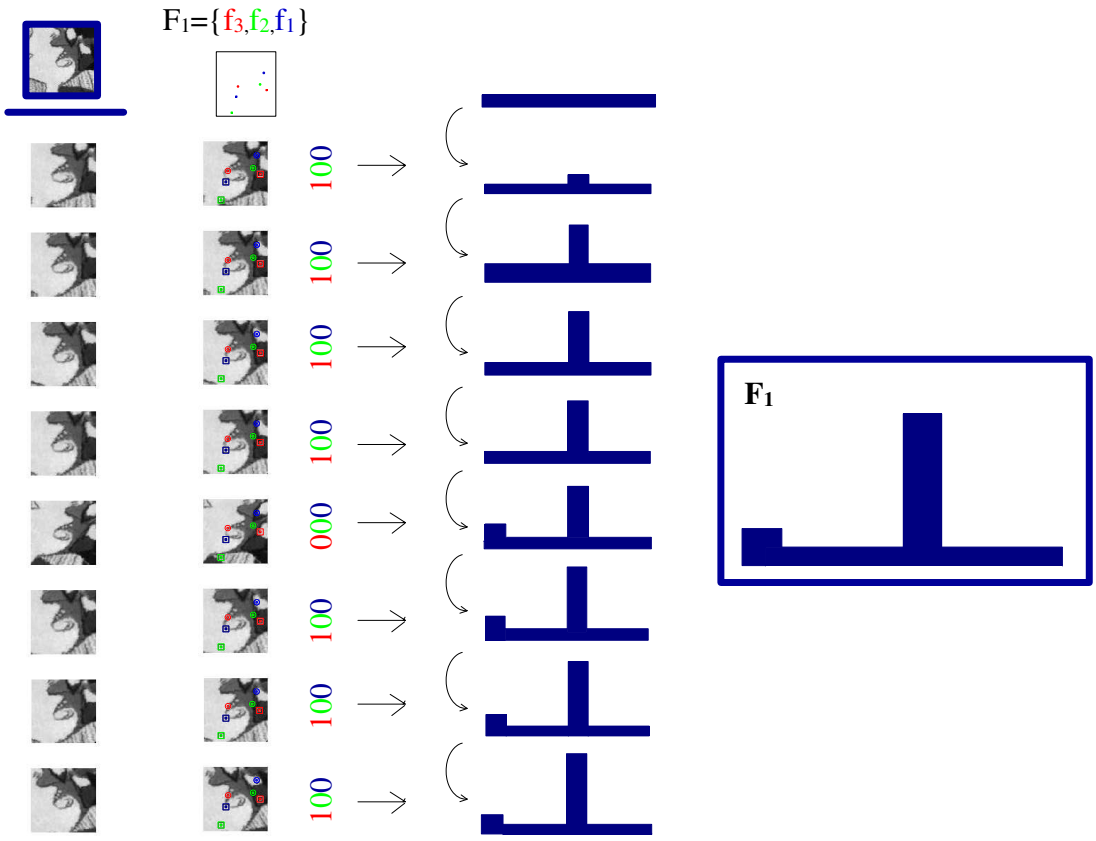

Figure 3.21: Fixed a class (first class on Figure 3.19) and a fern (F_1), probability that this fern takes some concrete values conditioned to a patch k which belongs to this class is computed. In this example, fern has size $S = 3$ so fern F_1 can get $2^3 = 8$ different values, $F_1 \in \{(0,0,0),(0,0,1),(0,1,0),\ldots,(1,1,1)\}$. Also, there are $N_{training} = 8$ patches in this class. As said before, histogram bins are initialized to one and each training patch modifies distribution by adding 1 to the bin corresponding to the value obtained when running the three binary test f_1, f_2, f_3 forming fern F_1.

$(f_1^m,\ldots,f_s^m)$ is obtained for that patch. The bin corresponding to this value is incremented in a unit. Once the fern F_m has been run for all the patches of the concrete class $\mathbf{k}_i$ the histogram is complete. Once it has also been normalized to one, each bin contains the information of $P(F_m = (f_1^m,\ldots,f_S^m) \mid k \in \mathbf{k}_i)$ for a concrete configuration $(f_1^m,\ldots,f_S^m) \in \{0,1\}^S$. Also, in Figure 3.22, different distribution results of applying same fern to different classes are shown.

At the end of the training process, probabilities $P(F_m|k = \mathbf{k}_i)$ have been computed for all m from 1 to M and for every class belonging to $\mathbf{K} = \{\mathbf{k}_1,\ldots,\mathbf{k}_{N_{class}}\}$. An example of what is obtained at the end of the training process can be seen at Figure 3.23.

As told before, once training has been made, given a patch k^{input} from the test image, decision about which class $\mathbf{k}_i$ is the one where k^{input} fits the best is obtained by running ferns

Figure 3.22: Probability distributions of fern F_1 conditioned to patches belonging to classes 1, 2 or 3 respectively ($P(F_1|k = \mathbf{k}_1)$, $P(F_1|k = \mathbf{k}_2)$, $P(F_1|k = \mathbf{k}_3)$).

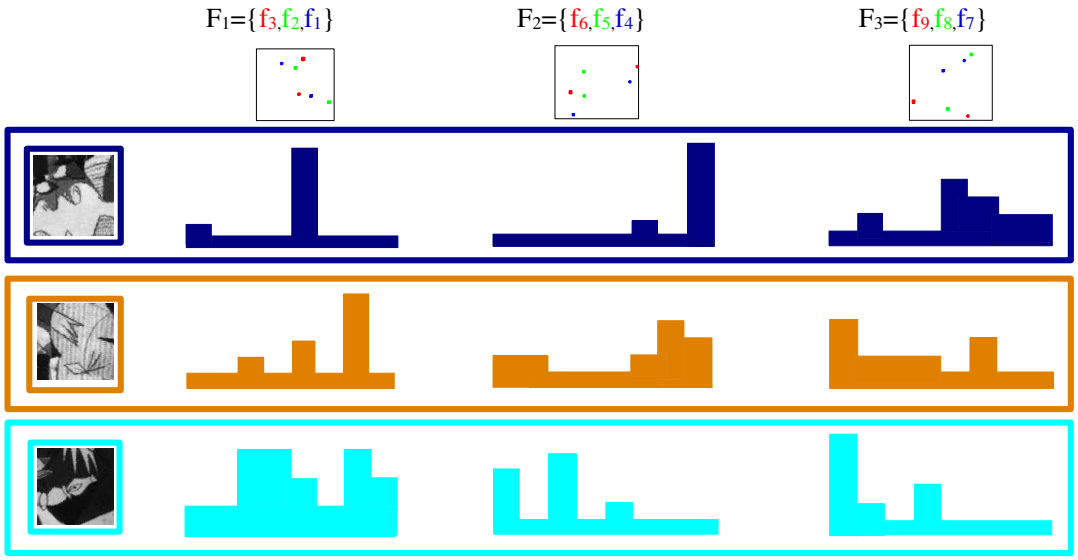

Figure 3.23: For every class, all ferns have been tested and final distributions kept. Figure shows distributions for ferns F_1, F_2, F_3 and classes $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$.

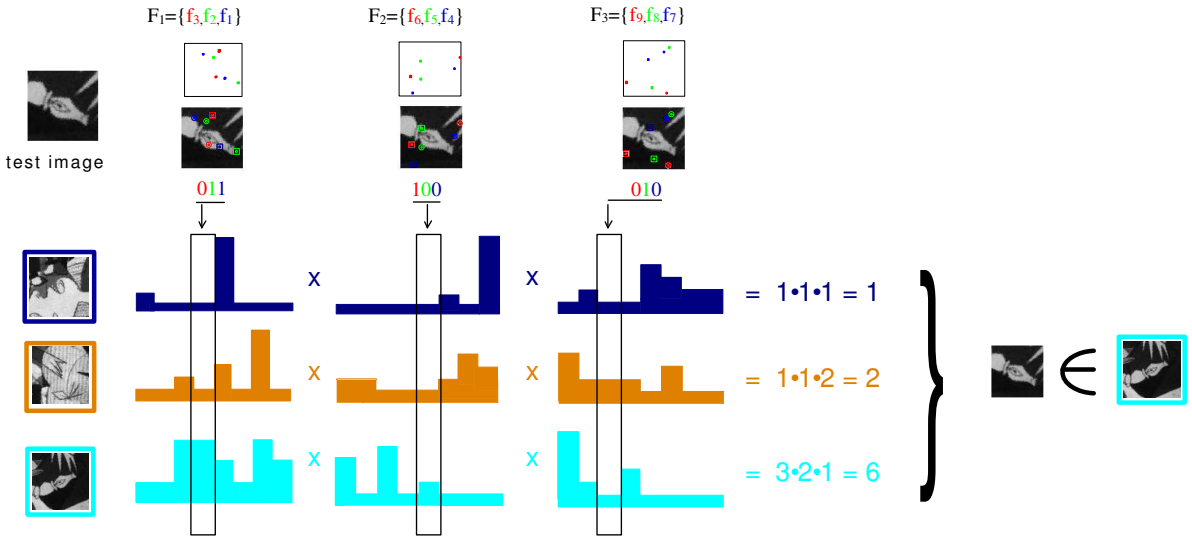

Figure 3.24: Evaluation of a patch k^{input} from a test image. Ferns F_1, F_2, F_3 are run on the test image. As said, for each class, $\mathbf{k}_i$, the probability all binary test taking this concrete configuration conditioned to the fact that k^{input} belongs to class $\mathbf{k}_i$ is computed in a multiplicative way where each factor is the probability that a certain fern gets the same configuration obtained when running the fern to k^{input} conditioned to the fact that k^{input} belongs to class $\mathbf{k}_i$. The class chosen for k^{input} is the one achieving the maximum for function $\max_{i=1,\ldots,N_{class}}\{\prod_{m=1}^{M} P(F_m|k^{input} \in \mathbf{k}_i)\}$

on k^{input} and considering $k^{input} \in \mathbf{k}_{\tilde{\imath}}$ such that

$$\mathbf{k}_{\tilde{\imath}} = \underset{\mathbf{k}_i=\mathbf{k}_1,\ldots,\mathbf{k}_{N_{class}}}{\operatorname{argmax}} \prod_{m=1}^{M} P(F_m|k^{input} \in \mathbf{k}_i)$$

This process can be observed at Figure 3.24.

A last remark that is used to speed up the test evaluation is the following one (very common in this type of applications):

$$\begin{aligned}
\mathbf{k}_{\tilde{\imath}} &= \underset{\mathbf{k}_i=\mathbf{k}_1,\ldots,\mathbf{k}_N}{\operatorname{argmax}} \prod_{m=1}^{M} P(F_m|k^{input} \in \mathbf{k}_i) = \{\log \text{ is a monotonically increasing function}\} \\
&= \underset{\mathbf{k}_i=\mathbf{k}_1,\ldots,\mathbf{k}_N}{\operatorname{argmax}} \sum_{m=1}^{M} \log\left(P(F_m|k^{input} \in \mathbf{k}_i)\right) \\
&= \underset{\mathbf{k}_i=\mathbf{k}_1,\ldots,\mathbf{k}_N}{\operatorname{argmax}} \sum_{m=1}^{M} \log\left(\frac{1+n^i_{(f_1^m,\ldots,f_S^m)}}{\sum_{F_m=(0,\ldots,0)}^{(1,\ldots,1)}\left(1+n^i_{F_m}\right)}\right)
\end{aligned}$$

Random Ferns in Our Algorithm

Based on parameter values given at [24], the values we use in our training processes were the following:

- Total number of binary tests, T, varies between 160 and 750.
- Pixel random positions (d_1^t and d_2^t) for a binary test f_t where to compare pixel intensities are taken within a window $PS \times PS$ centered at the keypoint pixel. PS is the patches' size along each direction and we have taken $PS \in [41, 101]$,
- Parameters M and S controlling number and size of ferns have been set to:
  $M \in \{20, 30, 40, 50\},\ S \in \{8, 10, 12, 14\}$.
- Set of classes $\mathbf{K}$ has varied its cardinality, N_{class} between 400 and 1200.

3.7 Comments on the Computational Cost

When discussing the computational cost of our method, it is very important to stress that training and evaluation are two processes completely different. While training requires high computational time, evaluation can almost be done at run-time.

The training process involves some stages with a large computational cost, such as mesh creation and the patches synthesis. Parts like the selection of interest points or the ferns' training have very little effect compared to the synthesis of the patches. The latter consumes the most part of the time of the training process.

Nevertheless, the synthesis process is easily parallelized, which we have done. Also, some other techniques could be applied to improve the computational cost. For example, we have mainly worked in MATLAB but we have implemented the visibility model in C++ to speed up. This type of optimization could be done to the other parts of the code. Moreover, it would also be helpful to work with the GPU (Graphical Process Unit).

Chapter 4

Results

In this chapter we present the results obtained by the method we have implemented. First, we present the way the results have been obtained and also indicate how we have compared them to the ones achieved by SIFT. Second, we effectively compare both methods. Finally, we investigate the influences of the different parameters on our method.

4.1 Experiment Setup

Test Set

We have used our method to train the Guernica image. Therefore, the test consists on measuring the response of our method to deformed Guernica images.

These deformed images have been synthetically generated. As explained in the previous chapter (Sections 3.4.1 and 3.5), the Guernica reference image is considered to be a 60×30 cm rectangular shape, modeled by a 41×21 mesh. The camera it placed approximately at 40 cm from the mesh and we use a focal length of 1000 pixels , on images of 1000×500 pixels Then deformed meshes of same size are created, taking into account maximum deformation angles. Finally, while taking into account a visibility model, Guernica image is synthesized on the new deformed mesh through the use of the barycentric coordinates. Figure 4.1 shows an example of a synthetic deformed image used to test the method.

We have run the different methods in images coming from meshes generated with different deformation angles. More precisely, each experiment has been iterated for sets of images of a concrete deformation type. Results are presented as the mean obtained for a set of four images of same deformation type.

For the deformations, we have here abbreviated types by using the codes 56, 67, 78 and 910. Each one of these codes consist on two numbers, $n_1 n_2$. First one, n_1 refers to the interval $\left[-\frac{\pi}{n_1}, \frac{\pi}{n_1}\right]$ where the angles to fix first triangle of the mesh are randomly taken. Remember that first of the triangles was constructed fixing a position for vertex $(1, 1)$ and rotating the initial triangle in plane XY with respect to each axis with angles θ_x, θ_y and θ_z. Then, what we say is that θ_x, θ_y, θ_z follow a uniform distribution in interval $\left[-\frac{\pi}{n_1}, \frac{\pi}{n_1}\right]$. Referring to n_2, all the angles

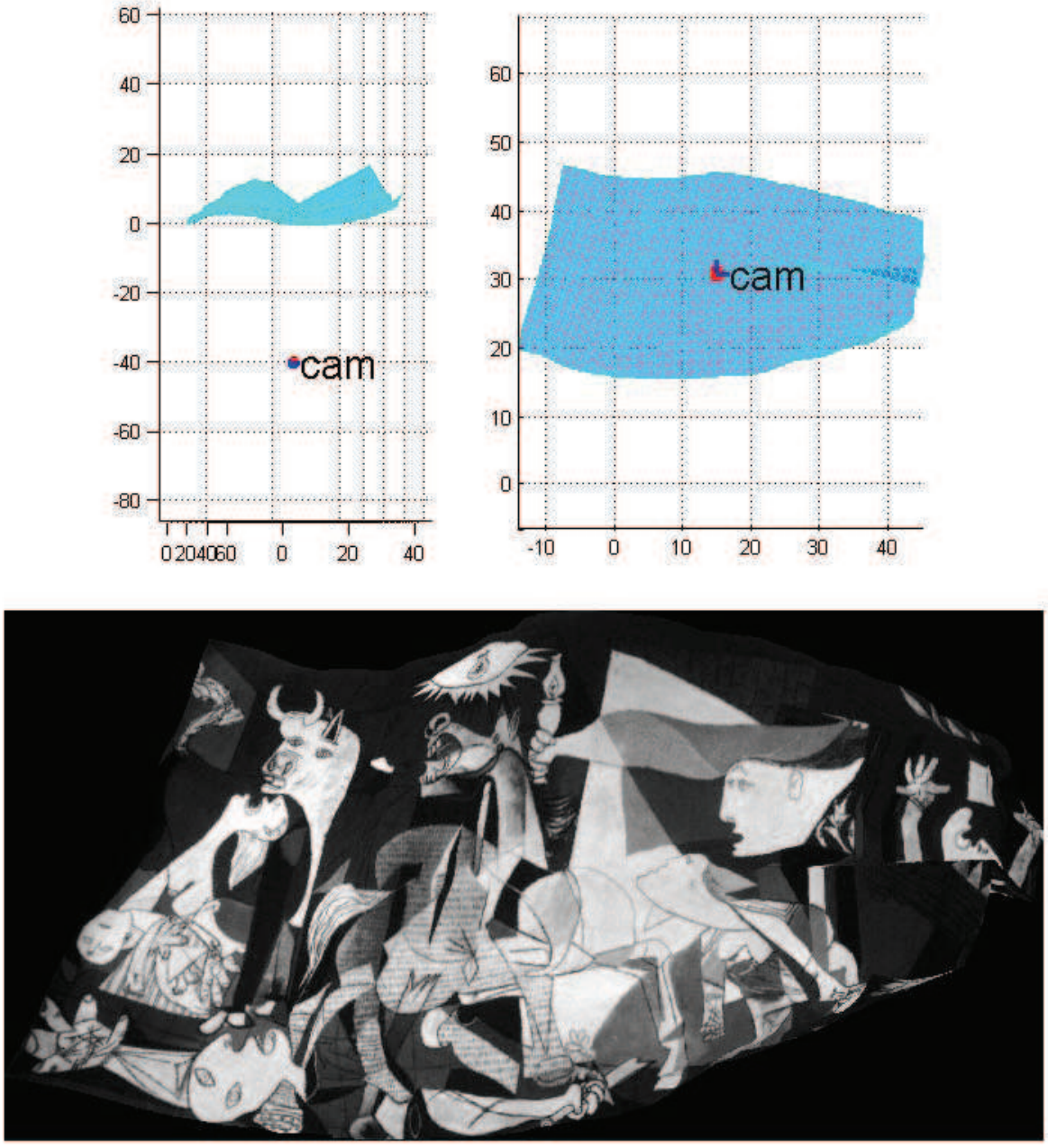

Figure 4.1: On top, two different views of the mesh used to generate the first synthetically deformed image of the Guernica. Camera position is also shown. At bottom, the test image once synthesized.

between triangles of the mesh are taken randomly in the interval $\left[-\frac{\pi}{n_2}, \frac{\pi}{n_2}\right]$. What is important is to realize that the meshes the most deformed are the ones with the smallest code (56) and the least deformed meshes are the ones with the biggest code (910).

Interest Points

As explained, both our method and also SIFT rely on local regions. In order to extract local patches on the test images, we have run the DoG detector (Difference-of-Gaussian). The 3000 most relevant points have been kept to be entries for the fern evaluation and also for the SIFT evaluation.

Since test images have been obtained synthetically we know the exact position of every interest point of the deformed image on the reference image. That way, the prediction of their position on the reference image can be known and, therefore, the methods can be evaluated. Figure 4.2 shows the 3000 most relevant interest points of a test image and their corresponding points on the reference image.

SIFT's Results

For every interest point extracted from the test image, we compute its corresponding SIFT descriptor, which is a 128-dimensional unitary vector.

We extract N_{class} interest points on the reference image. The N_{class} interest points for the training were kept according to a perspective weighting criterion (as seen in Section 3.3.2). The parameter N_{class} responds to the number of classes taken into account in the training process. Originally, it has been fixed to $N_{class} = 1200$ but as it will be seen, we have also obtained results with N_{class} fixed to other values. To obtain the SIFT's results, we have also computed the SIFT descriptor corresponding to each of the N_{class} points used for the training.

Then, for each SIFT descriptor of a point on the test image, we measure the Euclidean distance between it and every SIFT descriptor of the N_{class} points for the training. That way, for each point on the test image, we can sort the N_{class} points of the reference image from the closest one to the farthest one according to the SIFT descriptors' Euclidean distances. The closest point will represent the first candidate and the farthest one will be the last one for the deformed point being studied.

Our Method's Results

In a similar way as we do for the SIFT, to evaluate our method, we compute a sorted list of possible candidates for each interest point. Figure 4.3 shows the first interest point extracted on the test image and the first candidates obtained by our method.

We also want to recall the different parameters that influence on the results obtained by our method. These parameters are the ones discussed on the next sections:

- **Radius:** As explained in Section 3.6, ferns consist on various comparisons between intensity values of pairs of pixels. These pixels are chosen within a radius R from the interest point being described by ferns, where R has been varied between 20 and 50.

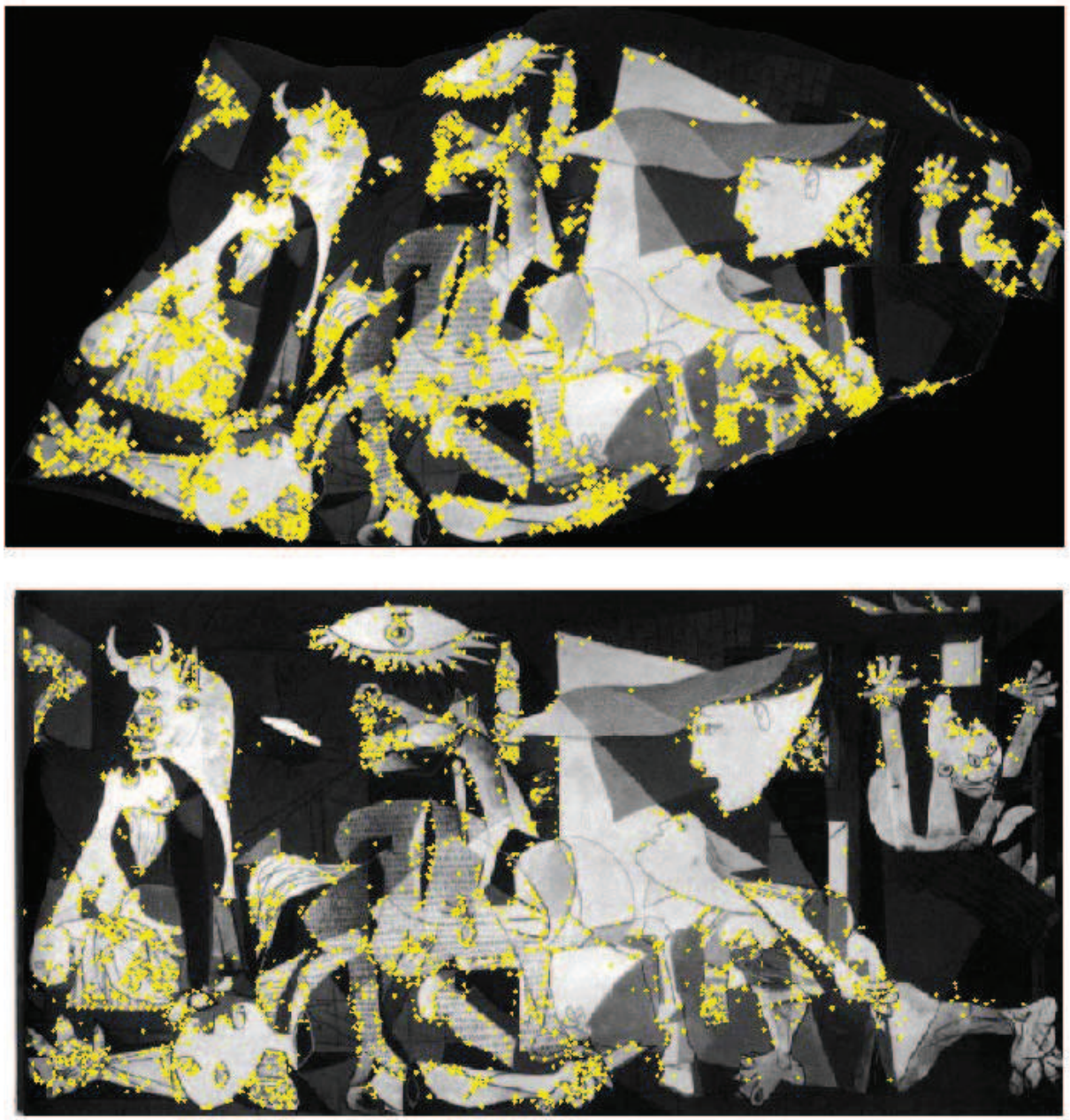

Figure 4.2: On top, once the test image is obtained, we first compute the most significative points through the Difference-of-Gaussian method. At bottom, thanks to the knowledge of the synthetic mesh, we associate each interest point on the test image to its real position on the reference image.

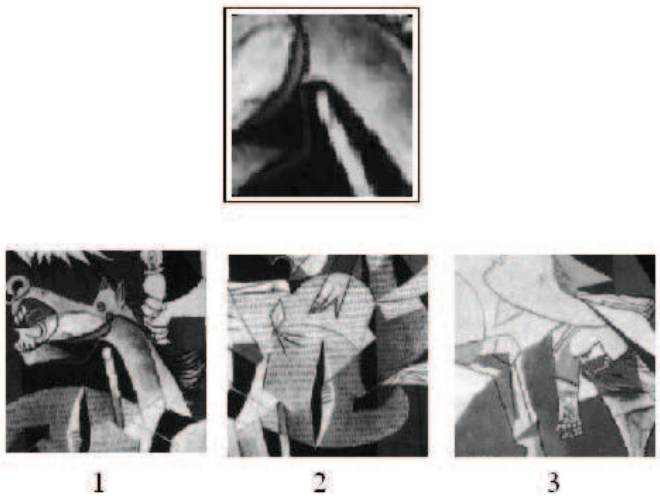

Figure 4.3: On top, first keypoint analyzed of the synthetic test image. At bottom, first three candidates found via our method. Observe first one is the correct one.

- **Number of ferns:** The quantity of ferns used in the classification problem, M, varies between 20 and 50.
- **Size of ferns:** The size of ferns, S, is the number of binary comparisons done to obtain a fern. The fern will be defined as an histogram with 2^S bins. S takes values from 8 to 14.
- **Deformation type for the training images:** We have trained Guernica with different ranges of deformation. According to the the abbreviations given before, we have trained images with meshes of type 56, 78 and 910.
- **Number of classes:** Finally, we have also considered the possibility of training for a different number of classes, N_{class}, and we have considered values $N_{class} \in \{400, 600, 800, 1000, 1200\}$.

Evaluation criterion

To evaluate each method, we have proceeded as follows:

- As previously said, for each deformed point, we have a list of all the training points on the reference image sorted according to the method criterion.
- Since we know the exact position of every deformed point in the reference mesh, we can also associate an error to every candidate according to the Euclidean distance between position of the deformed point on the reference mesh and position of the candidate on the

same mesh. If this distance is larger than 25 pixels, we say that the candidate given by the method is wrong. Otherwise, we will say that the candidate is a correct one.

This quantity of 25 pixels has been chosen according to the dimensions of the image being treated which in our experiments is of 500×1000. Figure 4.4 shows the precision obtained when working with this tolerance of 25 pixels.

- The results presented consist on the averaging of the response of the method to the 3000 points extracted from a specific test image and a set of four different test images of the same deformation type.

  For the results of the following sections, we plot the detection rate of the method: the abscissa informs about the quantity of candidates being taken into account (N) and the ordinates is the percentage between the number of points that method has associated at least one correct candidate to them among the $N - th$ first candidates and the total of points examined (in fact, in our case, denominator is $4 \cdot 3000$).

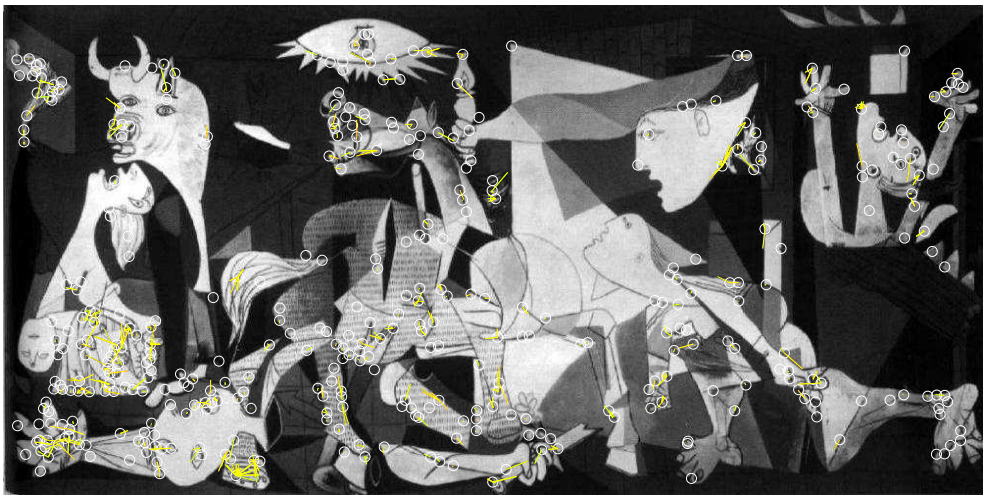

Figure 4.4: For the particular synthetic image shown in the previous figures, we here show the associations between the deformed points and their corresponding first candidate whose distances are less or equal than 25 pixels. Points of the training process which have been matched with at least one of the deformed points of the test image are circled in white. Distance between pairs are shown with yellow lines.

4.2 Overview of the Method's Improvement

While in next section we analyze in more detail the influence of every parameter in the detection rate given by our method, we first want to give an idea of the method's improvement with respect to SIFT.

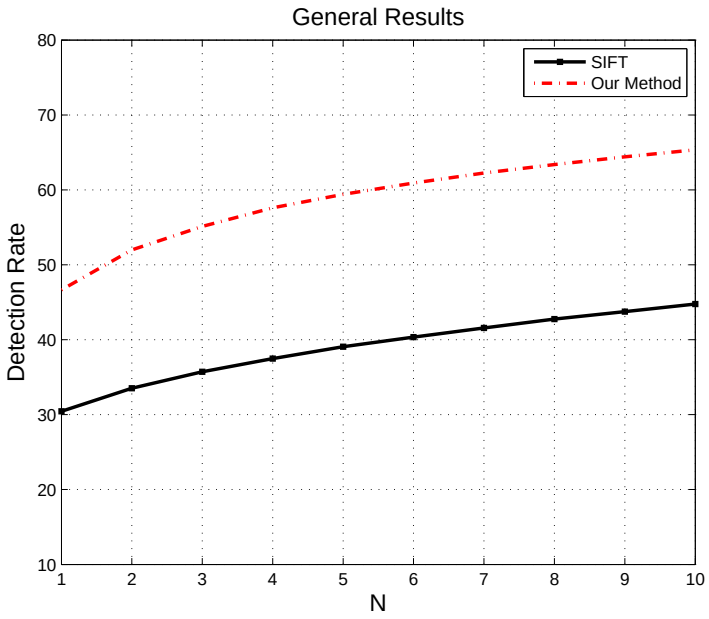

Figure 4.5: Detection rates obtained by the two methods. Taking only into account the first candidate, SIFT gives a positive response in the 30.44% of the cases whether our method starts with a 46.54%. This difference increases and when considering the first ten candidates, SIFT attains a 46.74% of correct pairs and our method adds about 20 percentage points more (65.35%). Here, data have been obtained making an average of the results presented for each type of deformation set of four synthetic images. Moreover, for our method, parameters selected have been: $R = 50$, $M = 50$, $S = 12$, $TrainDef = 56$ and $N_{class} = 1200$.

As seen in Figure 4.5, when choosing an optimal combination of parameters, our method outperforms in about 17 percentage points the detection rate given by SIFT. While taking into account only the first of the candidates we are able to increase a 15% the detection rate, when the number of candidates increases the improvement of our method also increases, achieving a difference of more than a 19% when considering the first 10th candidates. Although we have chosen one of the best parameter values' combination, we will show that all combinations outperform SIFT results except for radius fixed to $R = 20$ in the next section.

It is remarkable that, while our method achieves a detection rate of a 50% using only the first candidate, SIFT needs more than the first tenth candidates to achieve a similar rate.

4.3 Discussion with Parameter Variation

4.3.1 Radius' Selection

When changing the different parameters of our algorithm we found that the radius was the one introducing more variation on the final results. Figure 4.6 shows exactly this behavior.

As done in all the parameters' discussion, Figure 4.6 is split into four subfigures, each one corresponding to a different test set, composed of test images of a specific deformation type. While responses are quite similar among the different test sets, there's a subtle improvement of the different methods when applying them to test sets less deformed.

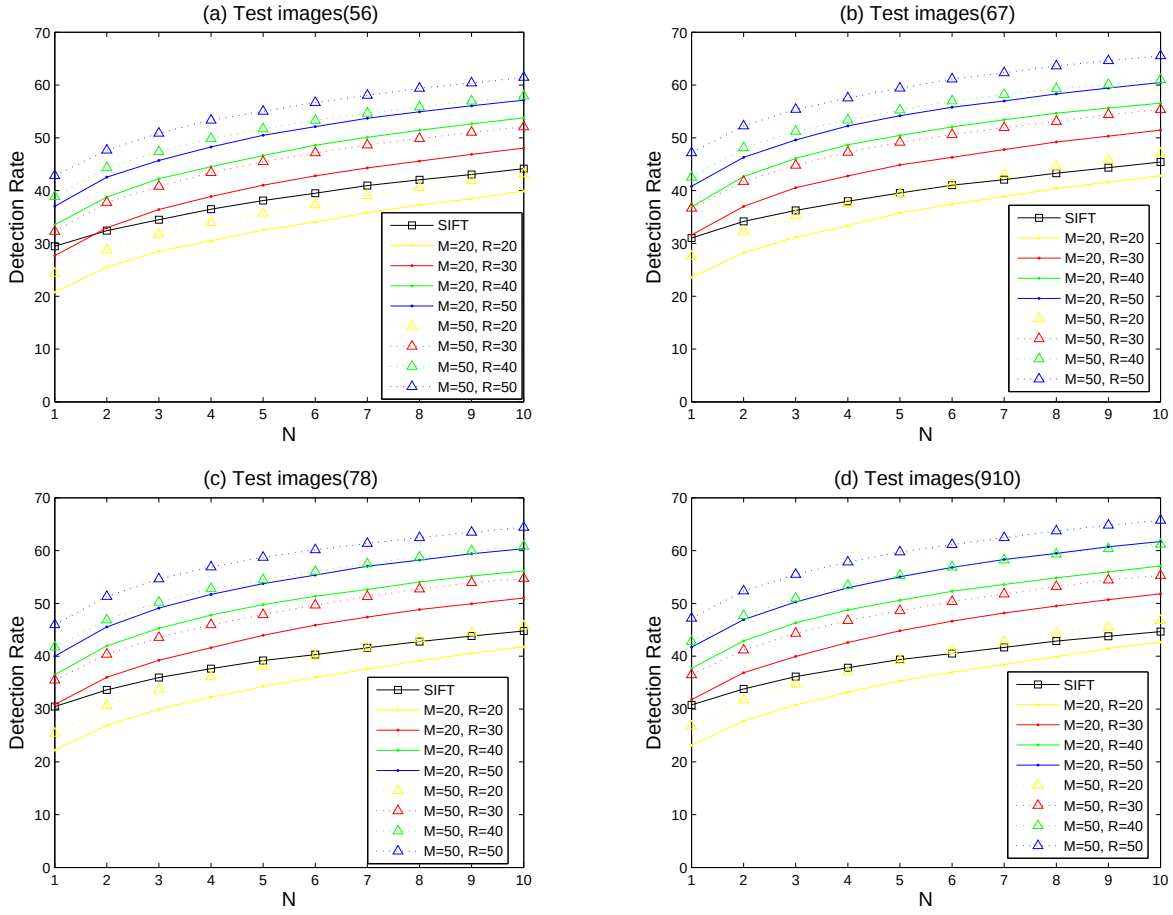

Figure 4.6: The first parameter discussed is the radius' election. Subfigures are taken fixing different test sets: Graphs in (a) are obtained with test set of type deformation 56, (b) of type 67, (c) of type 78, and (d) is of type 910. SIFT results are shown squared in black. At every plot parameter S is set to 12, $TrainDef$ to 56 and N_{class} to 1200. Color refers to the variation of the radius: $R = 20$ (in yellow), $R = 30$ (in red), $R = 40$ (in green) and $R = 50$ (in blue) while different markers indicate two different values of M ($M = 20$ in triangles and $M = 50$ in points).

In all test sets, it can be observed that a bigger radius yields a better response. More precisely, difference between the smallest radius, $R = 20$[1], and the biggest one, $R = 50$, is about 20 percentage points once all other parameters have been fixed to same value. Nevertheless, the

[1]In all cases, the radius is expressed in pixels

decay between the results presented for $R = 50$ and $R = 20$ is basically due to the decay between the detection rate achieved for $R = 30$ and $R = 20$, which is about 10 percentage points. With respect to $R = 50$, results for $R = 40$ decay about 4 percentage points and for $R = 30$ the decay is about 10 percentage points.

Another relevant issue is that, since the radius variation is shown for two different values of the number M of ferns, we are able to see that the radius has a greater influence than the amount of ferns. As it can be observed, the plots are distributed in bands according to the radius' selection (different colors correspond to different values of R). In other sections we will see that, after N_{class}, the radius is the most influential parameter.

Finally, we stress that the results obtained with our method always overcome the results obtained by SIFT, except for the case $R = 20$, where results are more or less the same as the those obtained by SIFT. Nevertheless, it can be observed that our method presents a bigger growth of the detection rate when taking into account more candidates. That way, when achieving $N = 10$, in the majority of test sets, the only parameter combination which doesn't get through SIFT results is $R = 20$ and $M = 20$.

4.3.2 Ferns' Size and Number of Ferns

After having studied the influence of radius' selection on the results, we will now study some of the parameters involving the amount of information computed.

Figure 4.7 shows the effects of choosing different values for the number of ferns M. Here, only three subfigures are shown. This is because we have used same deformation type for training and testing and we haven't trained for type 67. Then, the results are presented for a unique combination of parameters and only varying the amount of ferns. As it would be expected, the more ferns taken into account, the better the detection rate is. In particular, a variation on M from 50 to 20 implies a drop of about 4 percent. It can also be observed that just as in the radius case, the biggest drop is produced when varying M from 30 to 20 and also that, for more candidates, the differences on detection rates for $M = 50$ and $M = 40$ are reduced to less than a point in the worst case.

Finally, note that, once we fix R to 50, the variation on M gives an improvement of, at least, about a 15% (when M is equal to 20) and a 20% at most ($M = 50$).

The number of ferns is not the only way of processing more information on the algorithm. Also the number of classes used and the size of the ferns modify the amount of information. We now try to study this last parameter, the ferns' size, S. The variation on S can be observed in Figure 4.8.

In contrast to most of the parameters, the influence of the ferns' size on the results isn't monotone. As it can be seen, the best results are given by $S = 10$ and $S = 12$ depending on the other parameters' values.

While at first sight these results may be surprising, they are not. Remember that ferns' training is done for different classes which consist on interest points deformed and synthesized in different ways. The number of representatives or patches of the same class is a parameter which we fixed to 3000 according to implementation demands. For each class, a fern consists on a histogram resulting of the evaluation of S binary tests on each patch. Moreover, there is an

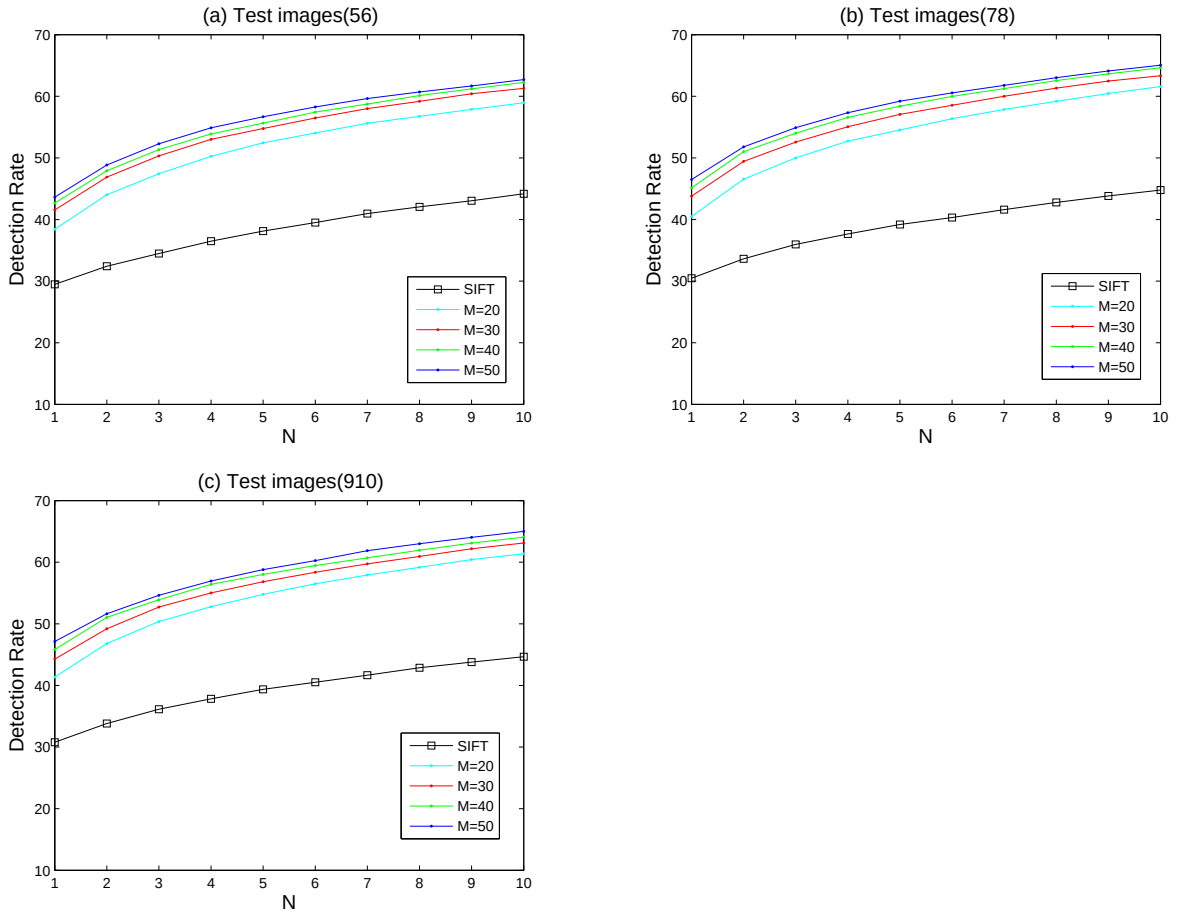

Figure 4.7: In these figures, we show the effect on the number of ferns used. Different subfigures are creating using a different test set: set of deformation type 56 in **(a)**, 78 in **(b)** and 910 in **(c)**. Analogously, $TrainDef$ has been set to 56 in (a), 78 in (b), and 910 in (c). SIFT results are presented in black squares. Color differentiates the choices of M: 20 in cyan, 30 in red, 40 in green and 50 in blue. The other parameters have been fixed to $R = 50$, $S = 12$ and $N_{class} = 1200$. Note that introducing more ferns gives more profitable information and results are improved.

initialization of one per bin before all tests are run in all the class representatives. Then, the binary number obtained when running the S tests on a patch counts per one in the corresponding bin. That way, if S is too big, the 3000 binary number obtained modify too little the initialization on the histogram. That is what we have found when using $S = 14$. Then, we have 2^{14} bins (16384) and only 3000 representatives to modify the initialization of weight one in each bin. For $S = 10$ and $S = 12$ the number of bins is 1024 and 4096 respectively and give the best results. For S set to 8, we have 256 bins and the results are worse than the ones obtained with the other values of S. Possibly, if we had given freedom to the number of representatives used per class, results for a bigger S would have been better than other values of S when the number of

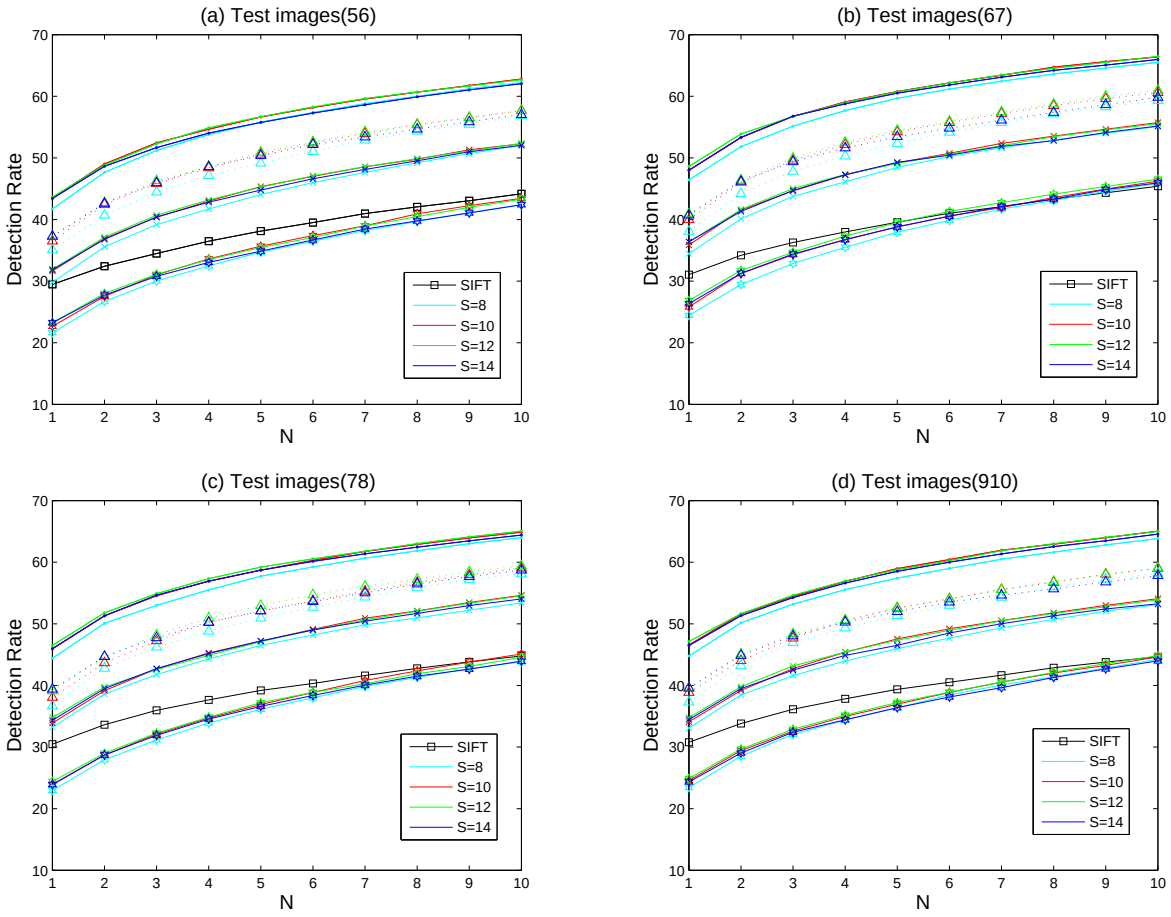

Figure 4.8: Influence of the ferns' size. We show a plot for every test set according to the deformation type (see each image's title). As usual, SIFT results are marked in black with squares. Colors distinct the different values taken by S. More precisely, we find $S = 8$ in cyan, $S = 10$ in red, $S = 12$ in green and $S = 14$ in blue. We have used $TrainDef = 56$ and $N_{class} = 1200$. Markers differentiate different set of values given to other parameters: $R = 50, M = 50$ in points, $R = 40, M = 30$ in triangles, $R = 30, M = 40$ in crosses and $R = 20, M = 30$ in hexagrams.

representatives per class was big enough. Nevertheless, the differences on detection rates do not exceed the 2% when varying the parameter S, so we can conclude that this parameter has not an important effect on the results.

4.3.3 Number of Classes

Another parameter that we have studied is the number of interest points taken into account for the training process. The results obtained when varying the number of classes, N_{class}, are shown in Figure 4.9.

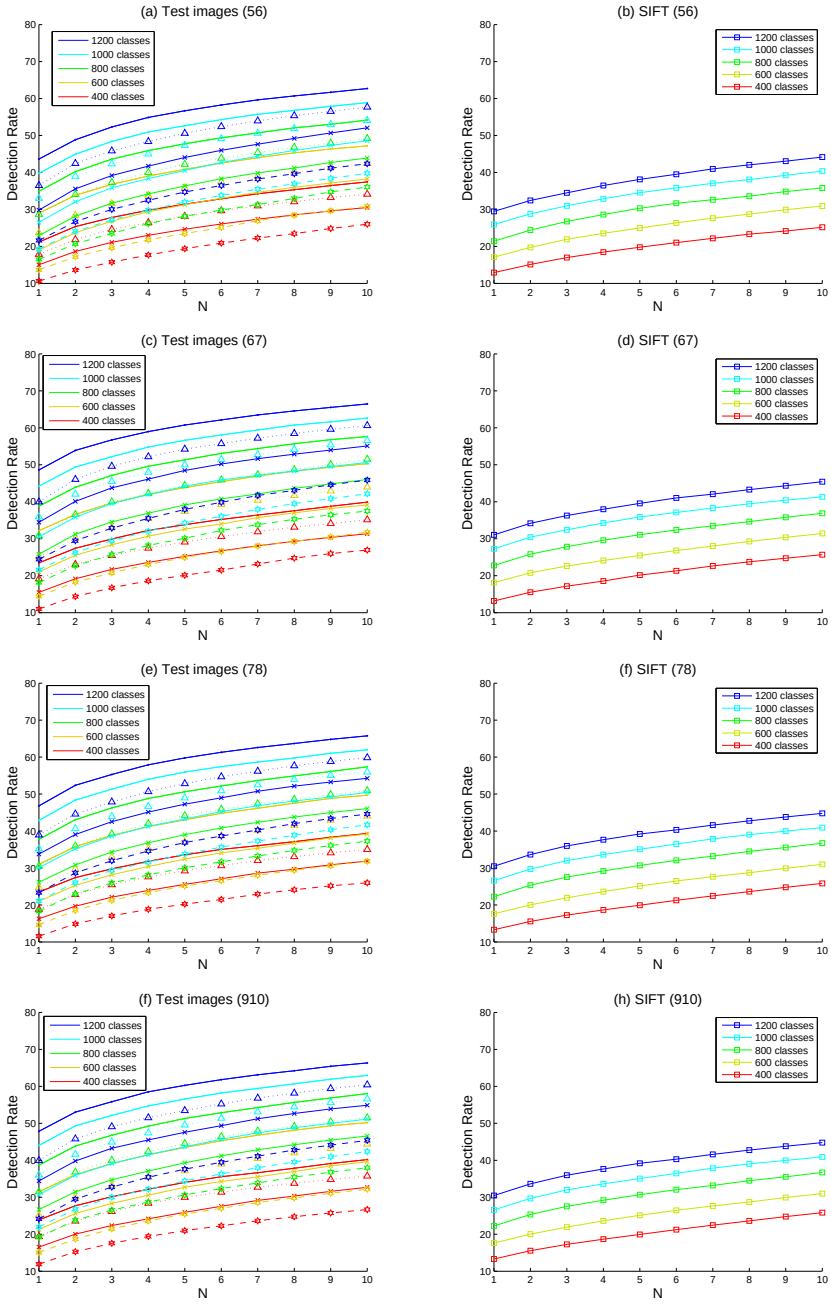

Figure 4.9: The influence of the number of classes considered on the training process is studied. On all plots, color is the distinction of the different number of classes taken (N_{class}=1200 (blue), 1000 (cyan), 800 (green), 600 (yellow) and 400 (red)). First column gives information of our method implemented with different parameters and second column deals with SIFT's evaluations. Each row refers to different types of test images (type 56, 67, 78 and 910). The different graphs shown for our method are: $R = 50, M = 50, S = 12$ (marked with points), $R = 40, M = 30, S = 10$ (triangles), $R = 30, M = 40, S = 8$ (crosses) and $R = 20, M = 30, S = 8$ (hexagrams). $TrainDef$ has been fixed to 56.

As it can be observed, choosing a value of N_{class} has a great influence on the detection rate given by our method. Even if other parameters have also an influence, figures show that the value of N_{class} is the most determining.

Results obtained either by our method or SIFT are better as the number of classes increases. For our method, the difference among results using same parameters except for N_{class} is of a 24%. Differences are bigger when the other parameters are set to their optimal values. When using non optimal values for the other parameters, the differences are shortened among the results with different number of classes to a 17%.

Moreover, once N_{class} is fixed, differences on the detection rates grow different when varying the other parameters. For example, once we fix N_{class} to 1200, there is about a 25% between the detection rate for R and M fixed to 50 and the case with $R = 20$ and $M = 30$. In contrast, fixing N_{class} to 400, detection rates related to this parameters' selection only differ in a 14%.

4.3.4 Deformation on the Training Process

Finally, the last parameter we analyze is the deformation type on the training process. We have trained ferns with three different sets of local meshes (which have served to create the set of 3000 representatives of each class once the meshes have been synthesized). These three sets differ on the deformation type of the meshes being used: 56, 78 or 910. Remember that this code corresponds to the ranges where angles used to localize first triangle in the space and also angles between triangles of the meshes are randomly taken. For example, a mesh of type 56 has the three angles that determine the orientation of the first triangle in the space in the interval $\left[-\frac{\pi}{5}, \frac{\pi}{5}\right]$ and angles between the triangles of the mesh in the interval $\left[-\frac{\pi}{6}, \frac{\pi}{6}\right]$.

Figure 4.10 show the response obtained when using the different training sets to each test set. Detection rates obtained with the different training sets differ less than two percentage points in the majority of cases. Therefore, we can confirm that deformation in the training process is not the most relevant part of the parameters' election.

Nevertheless, it can be observed that, for all test sets, the best response is the one using the most deformation in the training (type 56). This can be explained because type 56 also gathers information of meshes with less deformation since 56 refers to the maximum variation that can be obtained by angles in the mesh creation process.

4.3.5 Deformation on the Test Images

Although the deformation on the test sets is not a parameter to be discussed, we want to briefly mention some observations about it.

As one could have already recalled, figures presented up to now present a non monotone behavior considering the deformation type on the test sets. While it would be expected that for the test sets with more deformation, results were a little worst, it always happens that for the test set with deformation type 67, results are somehow better. This fact can be observed more precisely in Figure 4.11. We associate this phenomenon to the fact that the four meshes of the test set have had very little deformations. Remember that in the meshes generation, each angle

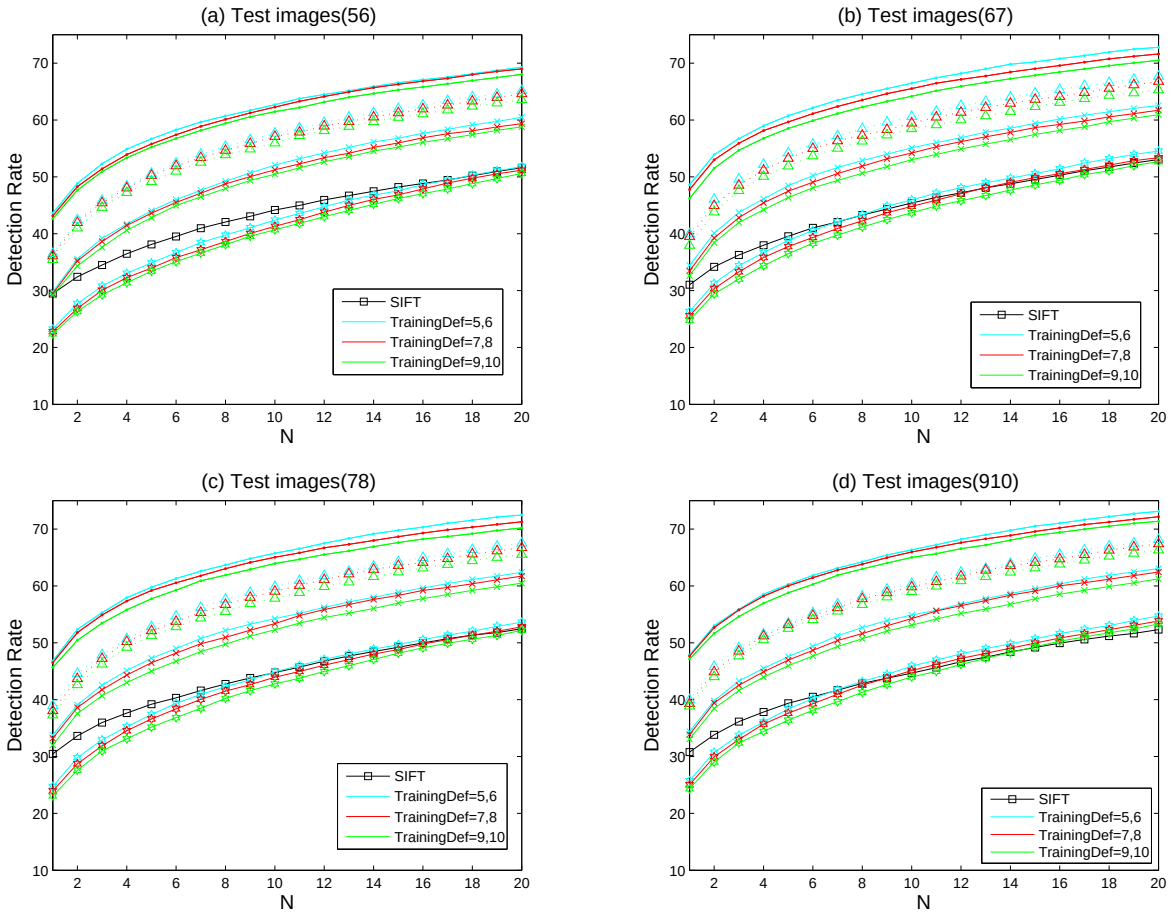

Figure 4.10: Once fixed the test set according to the deformation type (see the subfigures' titles). We have run the methods for different types of deformation in the training process. Therefore, we have used cyan for $TrainDef = 56$, red for $TrainDef = 78$ and green for $TrainDef = 910$. In all graphs, $N_{class} = 1200$. Different markers serve to distinguish different parameter values: $R = 50, M = 50, S = 12$, (points), $R = 40, M = 30, S = 10$ (triangles), $R = 30, M = 40, S = 8$ (crosses) and $R = 20, M = 30, S = 14$ (hexagrams). As usual, SIFT appears in black squares on all plots. Rather different than it would be expected, in all cases response is better when using the meshes the most deformed in the training process.

is chosen uniformly without a range determined by the values of the deformation type. The rest of test set behave as it would be expected: the less the deformation, the more the detection rate.

It is also remarkable that differences among results for different test sets are bigger in case we use our method than in SIFT's case. When applying our method with the same set of parameters, detection rate of type 910 is nearly a 5% greater than the one for the test set of type 56. On the other hand, for SIFT, the difference is shortened to a 2%. Therefore, we can conclude that SIFT is already very sensitive to small deformations and so, our method is more suitable in any type of deformations.

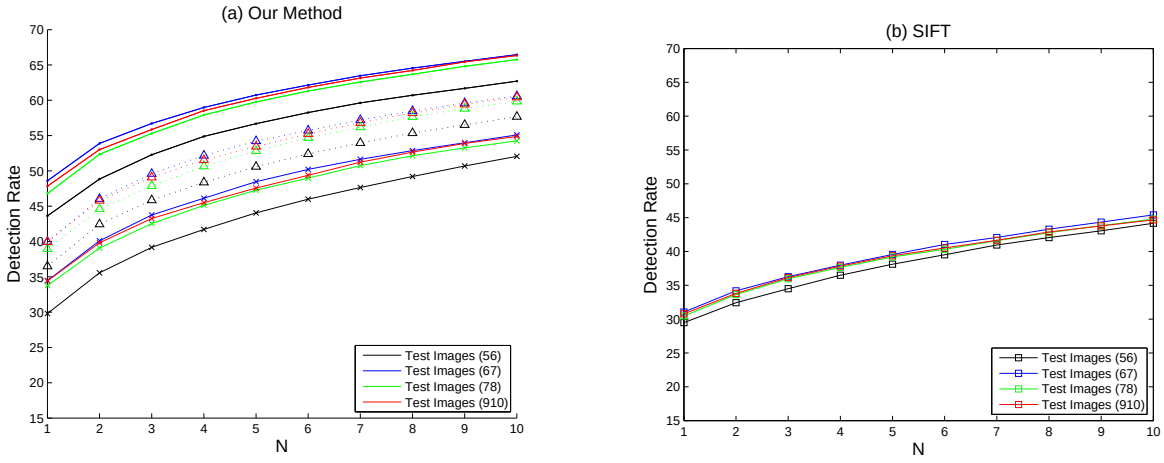

Figure 4.11: The responses of the same parameter evaluation are shown for different images set. At left, our method has been implemented with 4 different parameter combination: $R = 50, M = 50, S = 12$ (points), $R = 40, M = 30, S = 10$ (triangles), $R = 30, M = 40, S = 8$ (crosses) and $R = 20, M = 30, S = 14$ (hexagrams). N_{class} and $TrainDef$ have always been fixed to 1200 and 56 respectively. Different colors indicate the responses to the different test sets (see legend). At right, SIFT results are presented all at once for the different test sets. Surprisingly, a better response is given for test set of type 67. We attributed to the fact that the 4 meshes used for the deformed meshes have less deformation than others of the other type, since in their generation angles are chosen uniformly between a range and it can happen that deformation for these concrete images is more little than the other cases.

4.3.6 Conclusions on the Parameters' Selection

In this section, we have discussed the differences when choosing one value or another for each parameter on the parameter sets. We have seeen that the best results are achieved when using $R = 50$, $M = 50$, $TrainDef = 56$ and $N_{class} = 1200$. For the ferns' size $S = 10$ or $S = 12$ give more or less similar results. We have never abandoned the comparison of our results when varying the parameters and SIFT results. We have shown that with a good choice of the parameters, our method always gives about 18 percentage points more on the detection rate than SIFT.

Chapter 5

Conclusions

We have presented and implemented a method to solve the correspondence problem in images of non-rigid objects. The method has shown to perform well and significantly better than one of the most well-known methods in keypoint detection.

The method implemented turns the correspondence problem into a classification one. This classification problem is solved thanks to ferns and requires a training process.

For the training process, we have first chosen the keypoints taking part in the training process. This selection has combined the Difference-of-Gaussian detector and different perspectives of the planar reference mesh.

Then, the meshes used to perform the ferns' training have been selected and a large number of patches synthesized. To create deformed meshes, we have used a model based on planar triangles combined with other triangles through random angles which cause the deformation. Referring to the final selection of the meshes for the training, we have created a recursive method where PCA have a principal role. We consider from the first to the last components of the PCA basis. At each step, we distribute the meshes from the set according to the value of this components. This distribution is then used to weight the meshes to be taken of each group.

Next, for each keypoint, all the meshes' training set are synthesized. The synthesis is done taking into account a visibility model such that occluded parts of the meshes are not shown. Also, the orientation of the patches is considered so, when training, all patches of the same class or keypoint are aligned. This fact is very important since ferns rely on pixel intensity comparisons and otherwise they wouldn't be effective.

Finally, the comparisons between pixels of every generated patch are done. These binary tests let us to construct the histograms which describe the response of the different classes to same tests. Then, for a new input patch, we only have to look for the histograms which more likely pick up the responses given by the input patch.

For the evaluation, when a given image needs to be evaluated, we proceed as follows. First, we compute the interest points with the Difference-of-Gaussian detector. We consider a patch for every interest point and we align it following same method used during the training. Then, each patch is evaluated through the ferns and we should be able to decide if image corresponds to same object or not.

In our study, we have not gone on the prediction of whether the image corresponds to same object or not. What we have done is to compute the effective detection rates given by our method for the patches on a set of test images. We have then compared these rates to those obtained by SIFT. We have found that the response of our method is much better than the one given by SIFT. More specifically, detection rates are almost 20% above SIFT results. Moreover, we have studied the influence of the different implementation parameters on our method and found optimal values although almost for any combination of parameters our method was able to overcome SIFT.

Bibliography

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–588, 1998.

[2] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: speeded up robust features. *ECCV*, 2006.

[3] A. Bosch, X. Muñoz, and A. Zisserman. Image classification using random forests and ferns. *ICCV*, pages 1–8, 2007.

[4] A. Bosch, A. Zimmerman, and X. Muñoz. Representing shape with a spatial pyramid kernel. *CIVR*, 2007.

[5] H. Cheng, Z. Liu, N. Zheng, and J. Yang. A deformable local image descriptor. *CVPR*, 35(3):1–8, 2008.

[6] N. Dalal and B. Triggs. Histogram of orientated gradients for human detection. *CVPR*, 2005.

[7] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. *ICCV*, 2005.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Fourth Alvey Vision Conference*, pages 147–151, Manchester, UK, 1988.

[9] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[10] T. Joachims. Support vector and kernel methods. SIGIR 2003 Tutorial. `http://www.cs.cornell.edu/courses/cs578/2007fa/slides_sigir03_tutorial.pdf`, 2003.

[11] T. Kadir, A. Zisserman, and M. Brady. An affine invariant salient region detector. *IEEE Transactions on Pattern Analysis and Machine Learning*, 27(10):1615–1630, 2004.

[12] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:506–513, 2004.

[13] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scenes categories. *CVPR*, 2006.

[14] V. Lepetit. Selected topics in computer vision. Lecture 1. `http://cvlab.epfl.ch/teaching/topics/lectures_pdfs/stcv_lecture1.pdf`, September 2009.

[15] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE, PAMI*, 28(9):1465 – 1479, 2006.

[16] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. *CVPR*, 2:775 – 781, 2005.

[17] H. Ling and D. W. Jacobs. Deformation invariant image matching. *ICCV*, pages 1466–1473, 2005.

[18] D. G. Lowe. Distinctive images features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[19] J. Matas, O. Chum, M. Urban, and T. Padjla. Robust wide baseline stereo from maximally stable extremal regions. *BMVC*, pages 384–393, 2002.

[20] K. Mikolajczyk et al. A comparison of affine region detectors. *IJCV*, 65:43–72, 2005.

[21] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Learning*, 27(10):1615–1630, 2004.

[22] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectorss. *IJCV*, 60(1):63–86, 2004.

[23] H. Moravec. Obtacle avoidance and navigation in the real world by a seeing robot rover. *Tech Report CMU-RI-TR-3, Carnegie-Mellon University, Robotics Institute*, 1980.

[24] M. Ozuysal, V. Lepetit, and P. Fua. Fast keypoint recognition in ten lines of code. *CVPR*, 2007.

[25] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. *IEEE International Conference on Computer Vision*, pages 59–66, 1998.

[26] M. Salzmann, J. Pilet, S. Ilic, and P. Fua. Surface deformation models for nonrigid 3d shape recovery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1481–1487, 2007.

[27] M. Salzmann, R. Urtasun, and P. Fua. Local deformation models for monocular 3d shape recovery. *Proc. CVPR*, 2008.

[28] J. Shlens. A tutorial on principal component analysis. `http://www.snl.salk.edu/~shlens/pub/notes/pca.pdf`, December 2005.

[29] E. Tola, , V. Lepetit, and P. Fua. A fast local descriptor for dense matching. *Proc. CVPR*, 2008.

[30] E. Tola, V. Lepetit, and P. Fua. A fast local descriptor for dense matching. `cvlab.epfl.ch/~tola/research/08/daisy/cvpr08-daisy.pptx`, June 2008.

[31] E. Tola, V. Lepetit, and P. Fua. DAISY: an efficient dense descriptor applied to wide baseline stereo. *PAMI*, 99(1):1–31, 2009.

[32] T. Tuytelaars and L. Van Gool. Matching widely separated views based on affine invariant regions. *ICJV*, 59(1):61–85, 2004.

[33] A. Vedaldi. An implementation of multi-dimensional maximally stable extremal regions. `http://www.vlfeat.org/~vedaldi/assets/mser/mser.pdf`, February 2007.

[34] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 2001.

[35] S. A. J. Winder and M. Brown. Learning local image descriptors. *Proc. CVPR*, pages 1–8, 2007.

[36] H. Zhang, M. Berg, A.C. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. *CVPR*, 2006.