Gottfried Wilhelm Leibniz Universität Hannover
Institut für Informationsverarbeitung

Diplomarbeit

3D Object Recognition and Pose Estimation using Feature Descriptor Regression in a Bayes' Framework

Sergi Segura Morros

Betreuer:	Michele Fenzi, M. Sc.
Erstprüfer:	Prof. Dr.-Ing. Jörn Ostermann
Zweitprüfer:	Prof. Dr.-Ing. Bodo Rosenhahn

Hannover, August 2012

Contents

Notation and Constants

$p(A)$ Probability of event A.

$p(A \mid B)$ Probability of A conditioned to B (That is, knowing that B has occurred.)

σ_X^2 Covariance of X.

N Number of training images.

k Subset of training images used.

q Pose of the image.

$\bar{z}$ Feature Descriptor.

t Number of tracks used in the estimation of the pose.

d_i Euclidean distance between two features of the same track in different poses.

Z Matrix to store the track of features.

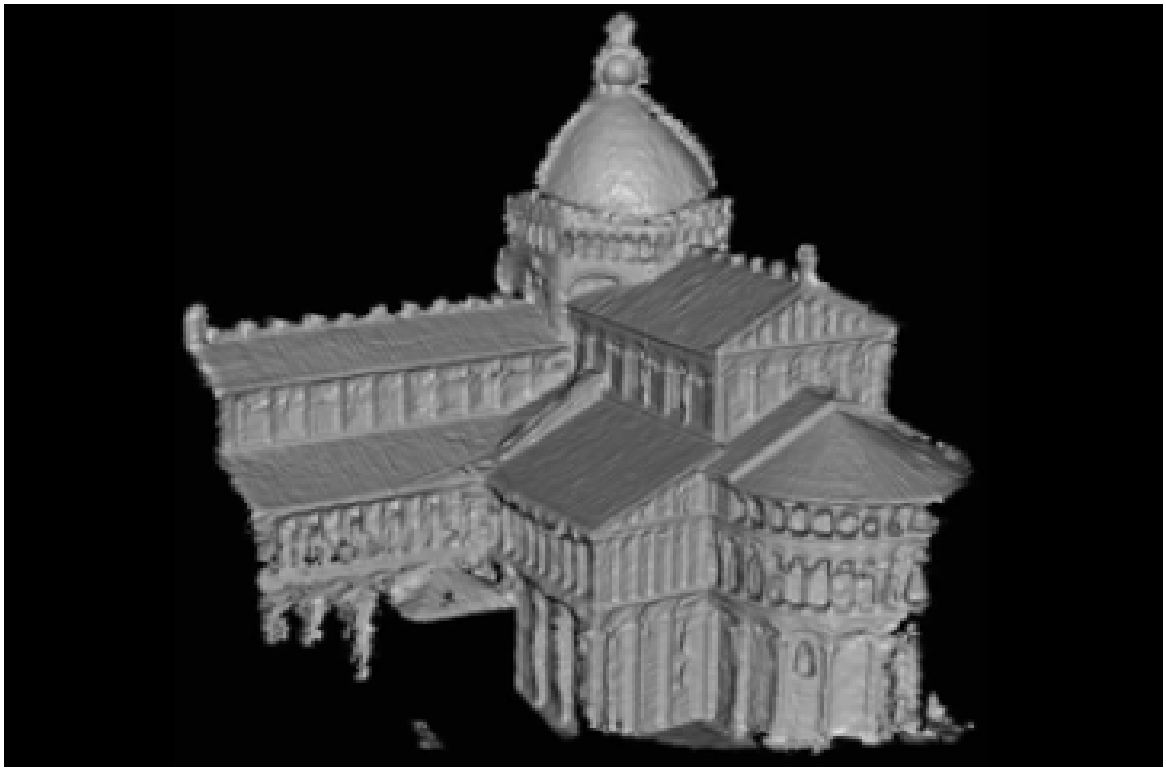
W Matrix to store the weight vectors.

1 Introduction

As humans, we perceive the three-dimensional structure of the world around us with apparent ease. One can distinguish the shape and texture of every form and effortlessly segment each object from the background of a scene. In solving these tasks, humans use the results of the visual system as input of a brain-based inference stage assisted by previously collected information. For example, human emotions are determined by combining the current facial appearance and past personal experience. Perceptual psychologists have spent decades trying to understand how the visual system works and interacts with the brain, but a complete solution to this problem remains elusive [1].

With the goal of reaching human performance, researchers in computer vision have been developing methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or visual results. As a stunning example, [2] presents reliable techniques for recovering the three-dimensional shape and appearance of extensive city areas by computing a 3D model from thousands of partially overlapping photographs collected randomly from the Internet (Figure 1.1 (a)). In a similar fashion, dense 3D models can be constructed using a large and detailed set of views of a particular object using stereo matching (Figure 1.1 (b)). Even though all this is already available, the performance that can be achieved at the moment is not nearly close to what humans can do. This is because computer vision can be considered as an inverse problem, in which, given insufficient information, some unknowns are tentatively recovered in order to fully specify the solution. Therefore, physics-based and probabilistic models are employed in order to disambiguate between potential solutions. Additionally, modeling the visual world in all of its rich complexity is far more difficult than, say, modeling the vocal tract that produces spoken sounds. To sum up, computer vision tries to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distribution in order to perform low- and high-level tasks.

(a) *3D model reconstructed using SfM*



(b) *3D model reconstructed using stereo matching*

Figure 1.1 – *Recovering of 3D shape and appearance*

1.1 Object recognition and pose estimation

Of all the visual tasks we might ask a computer to perform, analyzing a scene by recognizing all the constituent objects remains one of the most challenging problems. The recognition problem, which is the one that we will focus on, can be considered as a sub-problem of the former, as it aims at determining whether or not the image data contains some specific object.

Before going further, it is important to disambiguate between object recognition and object detection in order to avoid conceptual misunderstanding. The task of object recognition is the identification of specific stored or learned *objects*, usually together with their 2D position in the image or 3D pose in the scene. On the other hand, object detection envisages determining the presence of pre-specified or learned *classes* in an image. Examples include detection of possible abnormal cells or tissues in medical images or vehicle detection in an automatic road toll system. Detection and recognition are inherently connected, as the former can be used as a first step in a system based on the latter. With relatively simple and fast computations, smaller regions of interest can be detected in the image data, which can be further analyzed by more computationally demanding techniques in order to produce a correct object identification.

Even if object recognition can normally be solved robustly and without effort by a human, it is still not satisfactorily solved in computer vision for the general case: arbitrary objects in arbitrary situations. The existing methods for dealing with this problem have been developed in order to solve it only for few usual objects, such as simple geometric objects (*e.g.*, polyhedra), human faces, printed or hand-written characters, and vehicles, and only in specific situations, typically described in terms of well-defined illumination, easy background, and fixed pose of the object relative to the camera. In spite of this, object recognition is increasingly growing in popularity and is being applied in many computer vision fields. Here, we just mention a few in the following list in order to give a feeling for the breadth of its potential applicability.

- Augmented reality [3]
- Geo-localization [4]
- Robotic manipulation [5]
- Face detection [6]
- Optical Character Recognition [7]
- Content-Based Image Indexing [8]
- Automated vehicle parking systems [9]

By considering the approaches in the past and current literature dedicated to solve the object recognition problem, most of the interest is focused on methods based on local signatures that are designed to be invariant against predefined geometrical and illumination changes [10, 11, 12, 13]. Objects are described by means of many local

signatures, often called features, and these descriptors are stored in a database. Once a new test image is available, its description is compared with the database and the best matching object is returned after some geometrical verification. This approach allows to robustly detect the same object at different scales, lighting conditions, positions and orientations.

In our work, we will focus on object recognition and pose estimation for 3D objects. That is, we want to recognize a specific 3D object whose identity is already known and jointly estimate its pose relative to the camera. Object recognition does not imply the determination of the object pose *per se*, but many times the object pose is estimated as a by-product of recognition or even better, as a joint solution to the problem, as we also propose.

1.2 3D Object Recognition and Pose Estimation

When recognition and pose estimation are to be considered for 3D objects, the typical paradigm parallels the approach outlined above [14, 15]. This method starts by building a 3D model off-line from a set of training images of the object. The model is assembled by tracking a set of features over the training images and, by using Structure for Motion (SfM) techniques [16], a 3D point cloud is produced. We can see an example in Figure 1.2. Each point is characterized by its 3D position and by information regarding its appearance (like the training descriptors). Once a database of object models has been built, a new test image is input to the system. Features are extracted from it and matched against the model features in order to establish correspondences. If a reliable number of correspondences is found, it is possible to estimate a pose transformation that projects the 3D points onto the 2D points. To sum up, the appearance information is used for matching, while the geometric information is used for estimating the pose.

Drawbacks The paradigm outlined above has several drawbacks and it can lead to failure in various situations. A non-comprehensive list of reasons for failure is provided in the following.

- The reconstruction of the 3D model breaks down or provides inconsistent results in case the object is poorly textured. If the amount of object features is too small, the reconstruction cannot count on a sufficiently high number of stable feature tracks and thus, it fails.
- The application at hand is aimed towards classes and not individual objects, like class detection or class pose estimation. This is because it is impossible to collect the complete range of models of the class, and each individual model does not fit with the other instances of the same class due to the differences in appearance or geometry.

(a)

(b)

Figure 1.2 – *3D point cloud reconstruction using SfM techniques.*

- Objects possess repetitive feature patterns or the features themselves are located in dangerous configurations. For example, when features lie on one plane or even worse, on one line, reconstruction produces inconsistent results. This will cause that no 3D information can be extracted and, therefore, no 3D model can be reconstructed.
- If the method is used for large objects like buildings (for example, in a typical application of facade recognition), a huge number of training images is required in order to construct a reliable 3D model.
- A very accurate pose is not required in all applications. For example, the task of estimating the pose of a vehicle often requires to bin the pose only over large discrete intervals.

Evidently, the previous list of drawbacks points out that this paradigm should be chosen with care and circumvented as soon as the application at hand allows for it.

1.3 Thesis overview

Motivation As a motivation, we aim at creating a framework for object recognition and pose estimation that is not affected by the previous drawbacks and, at the same time, it is adequate for recognition and pose estimation applications with specific objects, like facades, faces and cars in which the objects to recognize present several constraints:

- The object is constrained to rotate in one dimension only.
- Suitable objects for feature extraction.
- Availability of a complete set of object views which is used to train the system.

For example, the objects mentioned above meet this constraints. As a matter of fact, only the 180° frontal range is of interest when it comes to recognize a face or a building, as it will be hard or even impossible to perform recognition at other poses. Additionally, in all these applications pictures are taken at eye level and centered at the object, so that the only available motion is the object rotation around its central axis. Furthermore, the aforementioned objects can provide a sufficient number of distinct features at different poses in order to allow for recognition, even though faces and facades are richer in features than cars. Regarding the third requirement, we will work on publicly available datasets that provide image sequences which frame the objects in their entirety.

Approach With regard to our approach, it consists of two different parts that somehow reminds of the previously outlined paradigm, as an off- and on-line parts are comprised. In the off-line part, the image data is collected and processed, while in the on-line part, a new image showing the object in a unknown pose is input to the system and its pose is estimated.

The main idea behind the presented approach draws its starting inspiration from a plausible way to cope with a common feature weakness. As a matter of fact, all features present in the literature are not invariant to certain changes in the pose of the object. Their repeatability drastically decreases when the object undergoes a three-dimensional change in its pose, for example, a rotation around its own axis. Given the experimentally proved assumption that feature descriptors have a well behavior as a function of the object pose, it is possible to learn a regressor for each feature that is able to provide an estimate of the feature descriptor for an unknown pose. By expressing the problem in a Bayesian fashion, a set of regressors learnt from the strongest features is used in order to obtain useful information about the object and its pose. As a result, an estimation of the pose of the current view can be obtained by minimizing an error function based on the distance in the feature space.

In a nutshell, our method tries to find a compromise between the brute-force approach of using all the ground truth data available and the complexity and preciseness of the regression function built out of as few appearances as possible in order to have reliably good results and estimate the pose of the object with a minimum error.

In the following chapter, a thorough presentation of the algorithms and theoretical tools used in this thesis, such as SIFT features, regression functions and function optimization is given. Chapter 3 contains a full description of the implementation of the method outlined above. Chapter 4 and 5 are dedicated to the experimental evaluation of the method and to conclusions and future research directions, respectively.

2 Theoretical Background

In this chapter, we provide a brief description for the basic theoretical tools that we have employed in this thesis in order to introduce the reader to the topics.

2.1 SIFT features

If the images in Figure 2.1 are to be matched, a common approach is to determine a set of good locations in both images, describe them in some robust way and match them [17, 18]. The first kind of feature that may be noticed are specific locations in the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. These kinds of localized features are often called keypoint features or interest points (or even corners) and are often described by the appearance of patches of pixels surrounding the point location. Another class of important features are edges (*e.g.*, the profile of mountains against the sky). These kinds of features can be matched based on their orientation and local appearance and can also be good indicators of object boundaries and occlusion events in image sequences.

SIFT [20], the acronym of Scale Invariant Feature Transform, is a method that detects interest points in an image and describes them through feature descriptor vectors that are invariant to image translation, scaling and rotation, and partially invariant to illumination changes and affine transformations. This means that these feature vectors are robust to changes and are good for matching and recognition. In Figure 2.2, a representation of the feature descriptors in an image is shown.

This keypoint detection and matching pipeline can be divided into three separate stages. During the feature detection (extraction) stage, each image is searched for locations that are likely to match well in other images. At the feature description stage, each region around detected keypoint locations is converted into a more compact and stable invariant descriptor that can be matched against other descriptors. The feature matching stage efficiently searches for likely matching candidates in other images.

SIFT features, unlike other description methods, are built in a scale-invariant way. This is accomplished examining an image at different scales. An image pyramid [21] is built to do it efficiently. This structure consists of a set of bandpass copies of an image, each representing the pattern information at a different scale. Since the object and its features in the image can appear at any size, their representation at different scales is

Figure 2.1 – *Two pairs of pictures to match*

necessary to determine their size and be able to localize them correctly. This is formed by convolution (filtering) of the original image with Gaussian functions of varying widths. The difference of Gaussian (DoG), $D(x, y, \sigma)$, is calculated as the difference between two filtered images, one being scaled k times:

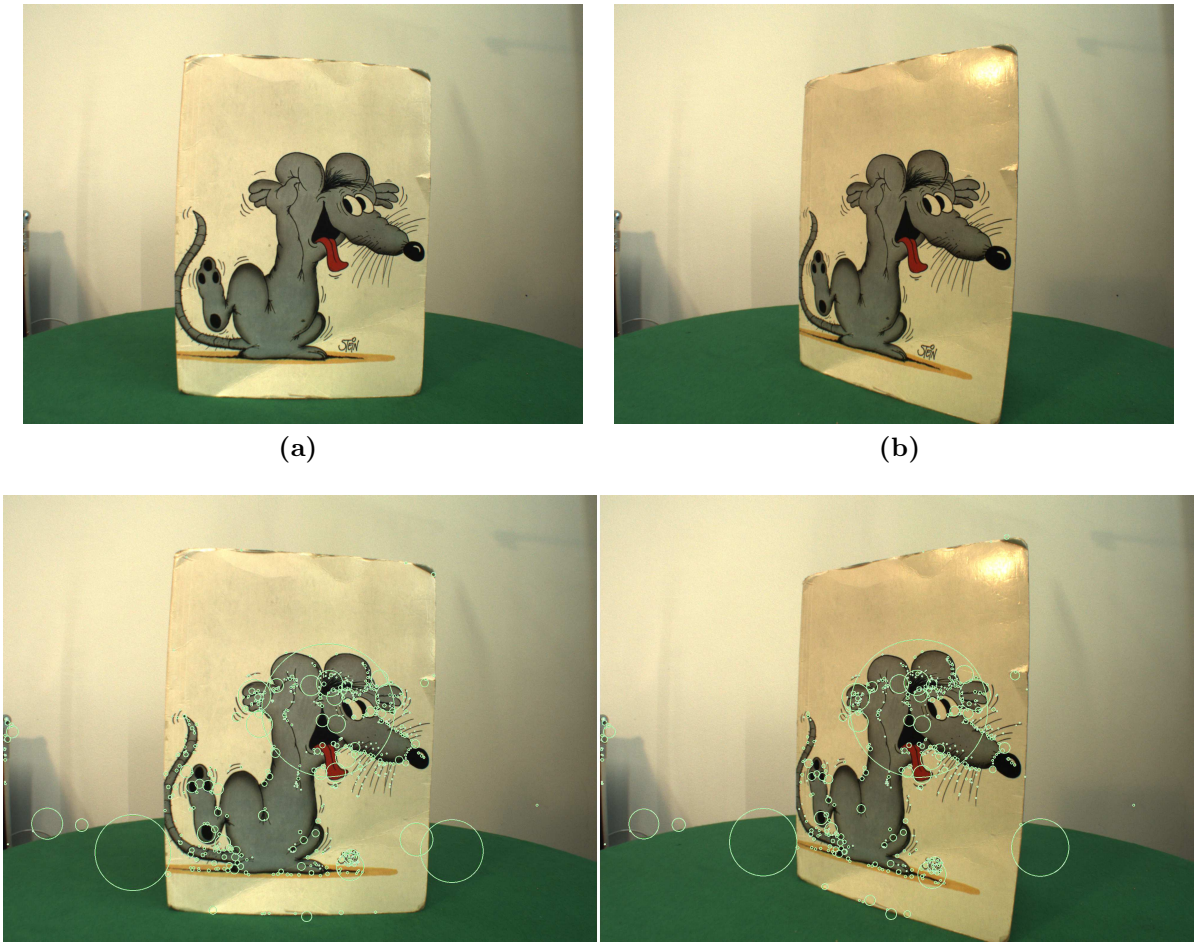
$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.1}$$

These images, $L(x, y, \sigma)$, are produced from the convolution of Gaussian functions, $G(x, y, k\sigma)$, with an input image, $E(x, y)$.

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

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2.3}$$

First, the initial image, E, is convolved with a Gaussian function, G_0, of width σ_0 to



(a) **(b)** **(c)**

Figure 2.2 – *The second image was generated by rotating the object around its axis.*

obtain L_0. L_0 is said to be a "reduced" version of E in that both resolution and sample density are decreased. Then, this blurred image, L_0 is used as the first image in the Gaussian pyramid and is incrementally convolved with a Gaussian filter, G_i , of width σ_i to create the i$^{\text{th}}$ image in the image pyramid, which is equivalent to the original image filtered with a Gaussian, G_k, of width $k\sigma_0$. The effect of convolving with two Gaussian functions of different widths is most easily found by looking at the Fourier domain, in which convolution becomes multiplication, *i.e.*,

$$G_{\sigma_i} * G_{\sigma_0} * f(x) \rightarrow \mathcal{G}_{\sigma_i}\mathcal{G}_{\sigma_0}\mathcal{F}(x) \tag{2.4}$$

The Fourier transform of a Gaussian function, e^{ax^2} is given by:

$$\mathcal{F}[e^{ax^2}](t) = \sqrt{\frac{\pi}{a}}e^{-\frac{\pi^2(t)^2}{a}} \tag{2.5}$$

By substituting this and equating it to a convolution with a single Gaussian of width $k\sigma_0$, it follows that:

$$e^{-t^2\sigma_i^2} e^{-t^2\sigma_0^2} = e^{-t^2k^2\sigma_0^2} \tag{2.6}$$

Performing the multiplication of the two exponentials on the left of this equation and comparing the coefficients of $-t^2$ gives:

$$\sigma_i^2 + \sigma_0^2 = k^2\sigma_0^2 \tag{2.7}$$

Figure 2.3 illustrates the effect of a Gaussian pyramid. The original image, on the far left, measures 257 by 257 pixels. This becomes level 0 on the pyramid. Each higher level array is roughly half as large in each dimension as its predecessor, due to reduced sample density.

Figure 2.3 – *First six levels of the Gaussian pyramid. The original image, level 1, measures 257 by 257 pixels and each higher level array has roughly half the dimensions of its predecessor. Thus, level 6 measures just 9 by 9 pixels.*

The images can be expanded to help visualizing the effects of the convolution at the different levels of the Gaussian pyramid, as it can be seen in Figure 2.4. The low-pass filter effect of the Gaussian pyramid is now clearly shown.

Figure 2.4 – *Expanded Gaussian pyramid.*

The next step is to subtract each level in the pyramid from the next lower level (being the lowest level the original unscaled image, E). Each level has different sample densities, so it is necessary to interpolate new sample values in order to perform subtraction. SIFT uses a bilinear interpolation with a pixel spacing of 1.5 in each direction. With this, what is called a Laplacian pyramid is constructed.

Pyramid construction acting as a bandpass filter tends to enhance image features (such as edges) at different scales, which are important for interpretation. To correctly choose these peaks, which will be the key locations, maxima and minima of a difference in the constructed images in the pyramid are looked for. Each pixel is compared with his 8 neighbors at the same level of the pyramid. If its a maximum or a minimum then the next lower level is considered and the same pixel (or the closest one taking into account the 1.5 interpolation) is compared with his other 8 neighbors. If the pixel value is still greater (or smaller) than this closest pixel and its 8 neighbors, then the test is repeated for the level above. An exemplification of the procedure can be seen in Figure 2.5

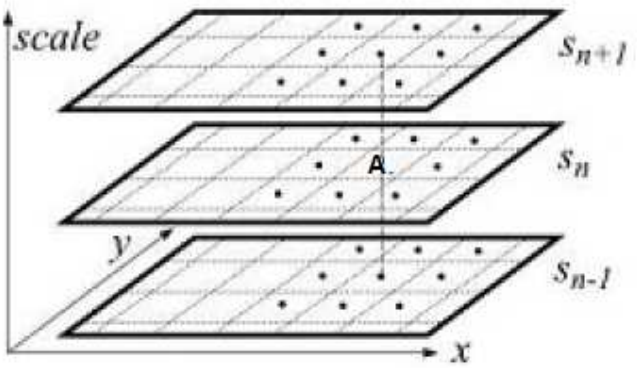
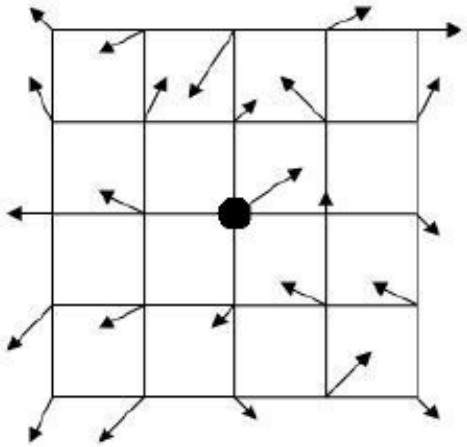


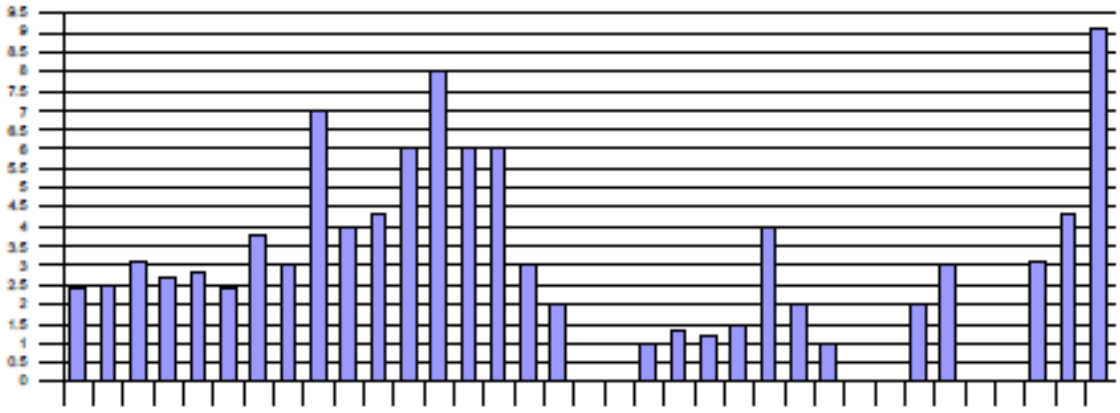
Figure 2.5 – *An extremum is defined as any value in the pyramid greater than all its neighbors in scale-space.*

In order to make the image descriptors invariant to rotation, a consistent orientation to the keypoints based on local image properties has to be assigned. An orientation

histogram is formed from the gradient orientations of the sample points within a region around the keypoint, as illustrated in Figure 2.6.



(a) *The point in the middle of the figure is the keypoint candidate. The orientation of the points in the square area around this point is precomputed using pixel differences.*



(b) *Each bin in the histogram represents 10 degrees, so it covers the whole 360-degree interval with 36 bins. The value of each bin holds the magnitude sum from all the points within that orientation range.*

Figure 2.6 – *Orientation Assignment*

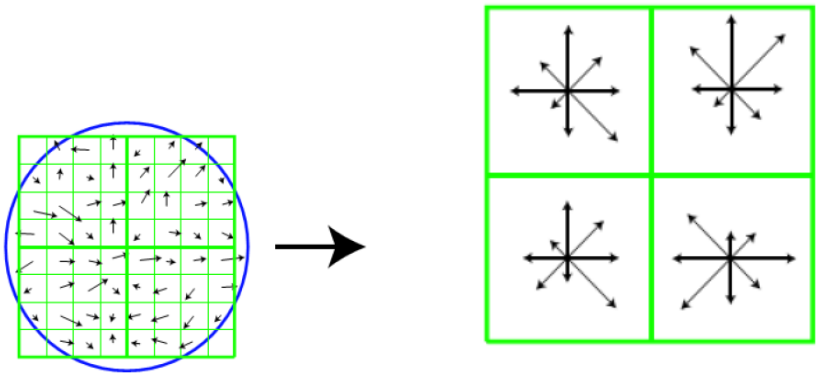
In the example, a 16×16 square is chosen. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each key location is characterized at each pixel A_{ij} by his image gradient magnitude M_{ij}, and his orientation R_{ij}, that are computed using pixel differences:

$$M_{ij} = \sqrt{(A_{ij} + A_{i+1,j})^2 + (A_{ij} + A_{i,j+1})^2} \tag{2.8}$$

$$R_{ij} = atan2(A_{ij} + A_{i+1,j}, A_{ij} + A_{i,j+1})^2 \tag{2.9}$$

The peak in the gradient orientation histogram is searched to find the canonical orientation for each key location, and corresponds to dominant directions of local gradients. The highest peak in the histogram is located and used along with any other local peak within 80% of the height of this peak to create a keypoint with that orientation. Some points will be assigned multiple orientations if there are multiple peaks of similar magnitude. A Gaussian distribution is fit to the 3 histogram values closest to each peak to interpolate the peaks position for better accuracy. This computes the location, orientation and scale of SIFT features that have been found in the image. These features respond strongly to corners and intensity gradients.

When we had selected all the SIFT key candidates for the sample image, it is necessary to compute a descriptor to characterize each keypoint. The image gradient magnitudes and orientations are sampled around the keypoint location. These values are illustrated with small arrows at each sample location in Figure 2.6(a). A Gaussian weighting function with σ related to the scale of the keypoint is used to assign a weight to the magnitude. A σ equal to one half the width of the descriptor window is used in this implementation. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. A 4×4 sample array is computed and a histogram with 8 bins is used. So a descriptor contains $4 \times 4 \times 8$ elements in total.



(a) *Image gradients* **(b)** *Keypoint descriptor*

Figure 2.7 – *(a) The gradient magnitude and orientation at a sample point in a square region around the keypoint location. These are weighted by a Gaussian window, indicated by the overlaid circle. (b) The image gradients are added into an orientation histogram. Each histogram includes 8 directions indicated by the arrows and is computed on* 4×4 *subregions. The length of each arrow corresponds to the sum of the gradient magnitudes near that direction within the region.*

When all the SIFT keys for the sample image are selected, they are stored and then used to identify matching keys in the image that we want to recognize.

To sum up, SIFT features possess the following properties:

- Scale-invariant.
- Rotation-invariant.
- Partially invariant to affine distortion (such as geometric contraction and expansion)
- Partially invariant to illumination changes.

As it is clear from this list, SIFT features are not invariant to out-of-plane changes in the pose of the object. Stated in a different way, the repeatability of these features drastically decreases when the three-dimensional pose of the object changes, for example, when the object rotates around its own axis. Therefore, SIFT features do not allow for a wide baseline matching and this is a strong weakness when it comes to applications that involve object recognition.

Nonetheless, we show in the following chapter that this weakness can be coped with by exploiting the well behavior of the feature descriptor as a function of the object pose. As a matter of fact, a regression function can be built for each feature which estimates the appearance of the descriptor vector given an unknown pose as input. This regression framework can be somehow "reverted" in a Bayesian sense in order to provide an estimation of the pose, as it is shown in the next chapter.

2.2 Regression

In order to identify the pose of an object in a new image when using a feature-based approach where features are not perspective invariant, two strategies are possible. The first is to have a database containing all the poses of the object so that test image can be compared against it and the most similar pose can be extracted. This method is evidently not efficient and hardly implementable, with an error exclusively dependent on the number of images at our disposal. The second method is to use a smaller set of object images at different poses and somehow estimate the descriptor appearance at poses that were not initially available.

In this thesis, we decided to use a regression function in order to first estimate the descriptors appearance at new poses and consequently solve the pose estimation problem as a distance minimization problem in the feature space. In the following, we give a brief introduction to regression fundamentals, while a thorough description of the regression approach used in this work is given in the following chapter.

A regression function f can be thought of as a function modelling the behaviour of an underlying unknown natural phenomenon. This modelling is usually expressed as

a weighted combination of the input variables that yields a *good* approximation of the output with respect to a certain optimality criterion, *i.e.*

$$Y \approx f(X, \beta),$$

where:

- β are the unknown weighting parameters.
- X are the independent variables.
- Y is the dependent variable.

The simplest regression function is a linear model that involves only one independent variable. This model states that the true mean of the dependent variable changes at a constant rate as the value of the independent variable increases or decreases. Thus, the functional relationship between the true mean of Y_i, denoted by $\xi(Y_i)$, and X_i is the equation of a straight line:

$$\xi(Y_i) = \beta_0 + \beta_1 X_i \tag{2.10}$$

β_0 is the intercept, *i.e.* the value of $\xi(Y_i)$ when $X = 0$, and β_1 is the slope of the line, *i.e.* the rate of change in $\xi(Y_i)$ per unit change in X. The observations on the dependent variable Y_i are assumed to be random observations from populations of random variables with the mean of each population given by $\xi(Y_i)$. The deviation of an observation Yi from its population mean $\xi(Y_i)$ is taken into account by adding a random error ϵ_i to give the statistical model

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i \tag{2.11}$$

The subscript i indicates the particular observational unit, $i = 1, 2, \ldots, n$. The X_i are the n observations on the independent variable and are assumed to be measured without error. That is, the observed values of X are assumed to be a set of known constants. The Y_i and X_i are paired observations; both are measured on every observational unit.

The random errors ϵ_i have zero mean and are assumed to have common variance σ^2 and to be pairwise independent. Since the only random element in the model is ϵ_i, these assumptions imply that the Y_i also have common variance σ^2 and are pairwise independent. The random errors are assumed to be normally distributed, which implies that the Y_i are also normally distributed.

Least Squares Estimation The simple linear model has two parameters β_0 and β_1, which are to be estimated from the data. If there were no random error in Y_i, any two data points could be used to solve explicitly for the values of the parameters. The random variation in Y, however, causes each pair of observed data points to give different results

(all estimates would be identical only if the observed data fell exactly on the straight line). A method is needed that will combine all the information to give one solution which is "best" by some criterion.

The least squares estimation procedure uses the criterion that the solution must give the smallest possible sum of squared deviations of the observed Y_i from the estimates of their true means provided by the solution. Let $\hat{\beta}_0$ and $\hat{\beta}_1$ be numerical estimates of the parameters β_0 and β_1, respectively, and let

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i \tag{2.12}$$

be the estimated mean of Y for each X_i, $i = 1, \ldots, n$. Note that $\hat{Y}_i$ is obtained by substituting the estimates for the parameters in the functional form of the model relating $\xi(Y_i)$ to X_i (Equation 2.10). The least squares principle chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the sum of squares of the residuals, $SS(Res)$:

$$SS(Res) = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum e_i^2 \tag{2.13}$$

where $e_i = Y_i - \hat{Y}_i$ is the observed residual for the *i-th* observation. The summation indicated by $\sum$ is over all observations in the data set as indicated by the index of summation, $i = 1$ to n (the index of summation is omitted when the limits of summation are clear from the context).

The estimators for β_0 and β_1 are obtained by using calculus to find the values that minimize $SS(Res)$. The derivatives of $SS(Res)$ with respect to $\hat{\beta}_0$ and $\hat{\beta}_1$ in turn are set equal to zero. This gives two equations in two unknowns called the normal equations:

$$n(\hat{\beta}_0) + \left(\sum X_i\right) \hat{\beta}_1 = \sum Y_i \tag{2.14}$$

$$\left(\sum X_i\right) \hat{\beta}_0 + \left(\sum X_i^2\right) \hat{\beta}_1 = \sum X_i Y_i \tag{2.15}$$

Solving the normal equations simultaneously for $\hat{\beta}_0$ and $\hat{\beta}_1$ gives the estimates of β_0 and β_1 as

$$\hat{\beta}_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2} = \frac{\sum x_i y_i}{\sum x_i^2} \tag{2.16}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X} \tag{2.17}$$

Note that $x_i = (X_i - \bar{X})$ and $y_i = (Y_i - \bar{Y})$ denote observations expressed as deviations from their sample means $\bar{X}$ and $\bar{Y}$, respectively. The more convenient forms for hand computation of sums of squares and sums of products are:

$$\sum x_i^2 = \sum X_i^2 - \frac{(\sum X_i)^2}{n} \tag{2.18}$$

$$\sum x_i y_i = \sum X_i Y_i - \frac{(\sum X_i)(\sum Y_i)}{n} \tag{2.19}$$

Thus, the computational formula for the slope is:

$$\hat{\beta}_1 = \frac{\sum X_i Y_i - \frac{(\sum X_i)(\sum Y_i)}{n}}{\sum X_i^2 - \frac{(\sum X_i)^2}{n}} \tag{2.20}$$

These estimates of the parameters give the regression equation:

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i \tag{2.21}$$

Extended Model Most models will use more than one independent variable to explain the behavior of the dependent variable. The linear additive model can be extended to include any number of independent variables:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \ldots + \beta_p X_{ip} + \epsilon_i \tag{2.22}$$

The subscript notation has been extended to include a number on each X and β to identify each independent variable and its regression coefficient. There are p independent variables and, including β_0, $p' = p + 1$ parameters to be estimated.

The usual least squares assumptions apply. The ϵ_i are assumed to be independent and to have common variance σ^2. For constructing tests of significance or confidence interval statements, the random errors are also assumed to be normally distributed. The independent variables are assumed to be measured without error.

The least squares method of estimation applied to this model requires that estimates of the $p + 1$ parameters be found such that:

$$SS(Res) = \sum (Y_i - \hat{Y}_i)^2 = \sum (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} - \hat{\beta}_2 X_{i2} - \ldots - \hat{\beta}_p X_{ip})^2 \tag{2.23}$$

is minimized. The $\hat{\beta}_j$, $j = 0, 1, \ldots, p$, are the estimates of the parameters. The values of $\hat{\beta}_j$ that minimize $SS(Res)$ are obtained by setting the derivative of $SS(Res)$ with

respect to each $\hat{\beta}_j$ in turn equal to zero. This gives $(p+1)$ normal equations that must be solved simultaneously to obtain the least squares estimates of the $(p+1)$ parameters.

In this thesis, the unknown parameters $\hat{\beta}_j$ are 128-dimensional vectors, p is the number of training samples used, the independent variable is a function of the actual pose, and the dependent value is the SIFT descriptor vector that is estimated given the actual pose. It is easily seen that other parameters apart form the pose of the object actually affects the value of the SIFT descriptor vector, such as lighting conditions and camera parameters, but these would be difficult and much more costly to recover [22].

Evaluation Each quantity computed from the fitted regression line $\hat{Y}_i$ is used as:

- Estimation of the population mean of Y for that particular value of X.
- Prediction of the value of Y one might obtain on some future observation at that level of X.

Hence, the $\hat{Y}_i$ are referred to both as estimates and as predicted values.

If the observed values Y_i in the data set are compared with their corresponding values $\hat{Y}_i$ computed from the regression equation, a measure of the degree of agreement between the model and the data is obtained. As seen, the least squares principle makes this agreement as "good as possible" in the least squares sense. The residuals:

$$e_i = Y_i - \hat{Y}_i \tag{2.24}$$

measure the discrepancy between the data and the fitted model.

The least squares estimation procedure minimize the sum of squares of the e_i. That is, there is no other choice of values for the two parameters β_0 and β_1 that will provide a smaller $\sum e_i^2$.

2.3 Optimization Algorithm

In order to detect the pose of the object under study, the features extracted belonging to the training images or estimated from these training images by the regression function are compared with the features extracted from a new test image input to the system. Therefore, it is necessary to have an optimization algorithm in order to find the pose that provides the minimum difference in the feature space. As done previously, we give here a brief introduction to optimization and its difficulties, while our approach is fully described in the following chapter.

Optimization algorithms find the best possible elements x^* from a set X according to a set of criteria $F = \{f_1, f_2, \ldots, f_n\}$. These criteria are expressed as functions, the so-called objective functions ($f : X \to Y$ with $Y \subseteq \mathbb{R}$).

The codomain Y of an objective function as well as its range must be a subset of real numbers ($Y \subseteq \mathbb{R}$). The domain X of f is the problem space and can represent any type of elements like numbers, lists, construction plans, etc. It is chosen according to the problem to be solved with the optimization process. Objective functions are not necessarily mere mathematical expressions, but can be complex algorithms that, for example, involve multiple simulations.

Optimization algorithms can be divided in two basic classes:

- Deterministic.
- Probabilistic.

Deterministic algorithms are most often used if a clear relation between the characteristics of the possible solutions and their utility for a given problem exists. Then, the search space can efficiently be explored using, for example, a divide and conquer scheme [23]. If the relation between a solution candidate and its "fitness" is not so obvious, too complex, or the dimensionality of the search space is very high, it becomes harder to solve a problem deterministically. Trying it would possibly result in an exhaustive enumeration of the search space, which is not feasible even for relatively small problems. On the other hand, probabilistic algorithms trade in guaranteed correctness of the solution for a shorter runtime.

Heuristics used in global optimization are functions that help deciding which one of a set of possible solutions is to be examined next. On the one hand, deterministic algorithms usually employ heuristics in order to define the processing order of the solution candidates. Probabilistic methods, on the other hand, may only consider those elements of the search space in further computations that have been selected by heuristics.

Regarding the optimization algorithm, the goal is to achieve the best results given a reasonable time. There is a constraint between accuracy and speed. Since in our case the optimization is performed in the on-line part of the method, speed is a factor that has to be considered.

In the case of our paradigm, in which we want to optimize a single criterion f, an optimum is either its maximum or its minimum, depending on what we are looking for. It is a convention that optimization problems are most often defined as minimizations and if a criterion f is subject to maximization, we simply minimize its negation $(-f)$.

Figure 2.8 illustrates such a function f defined over a two-dimensional space $X = (X_1, X_2)$. As outlined in this plot, we distinguish between local and global optima. A global optimum is an optimum of the whole domain X while a local optimum is an optimum of only a subset of X.

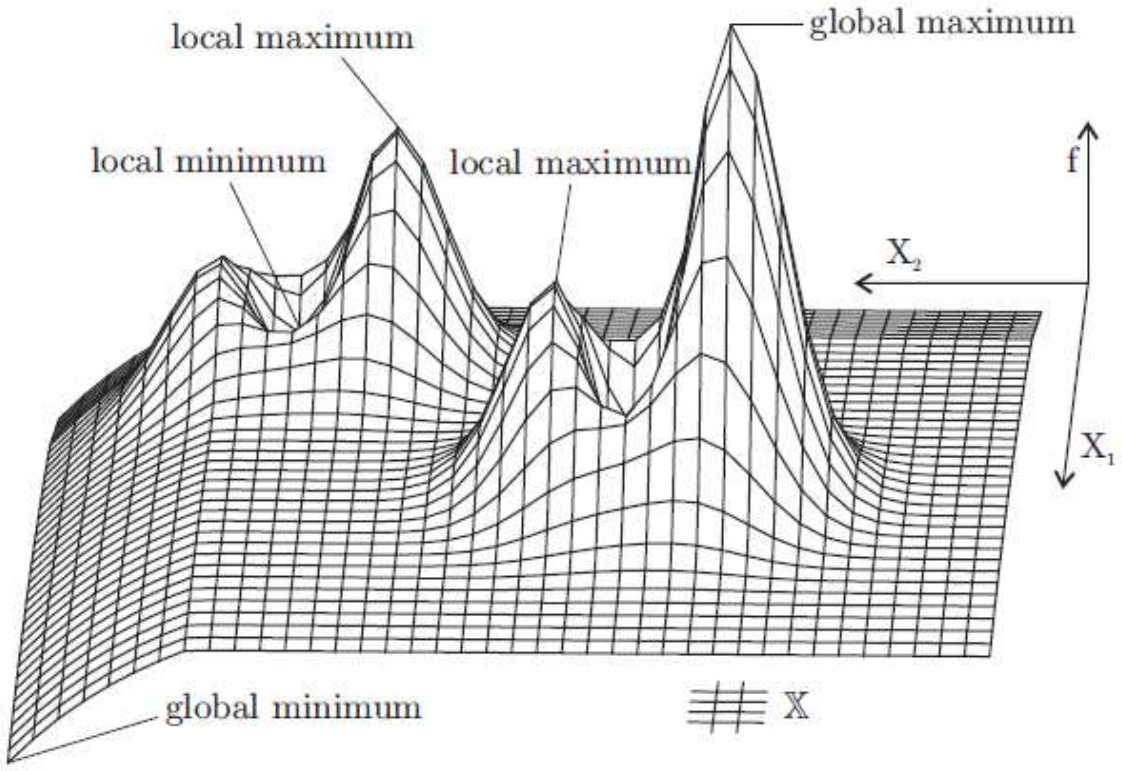
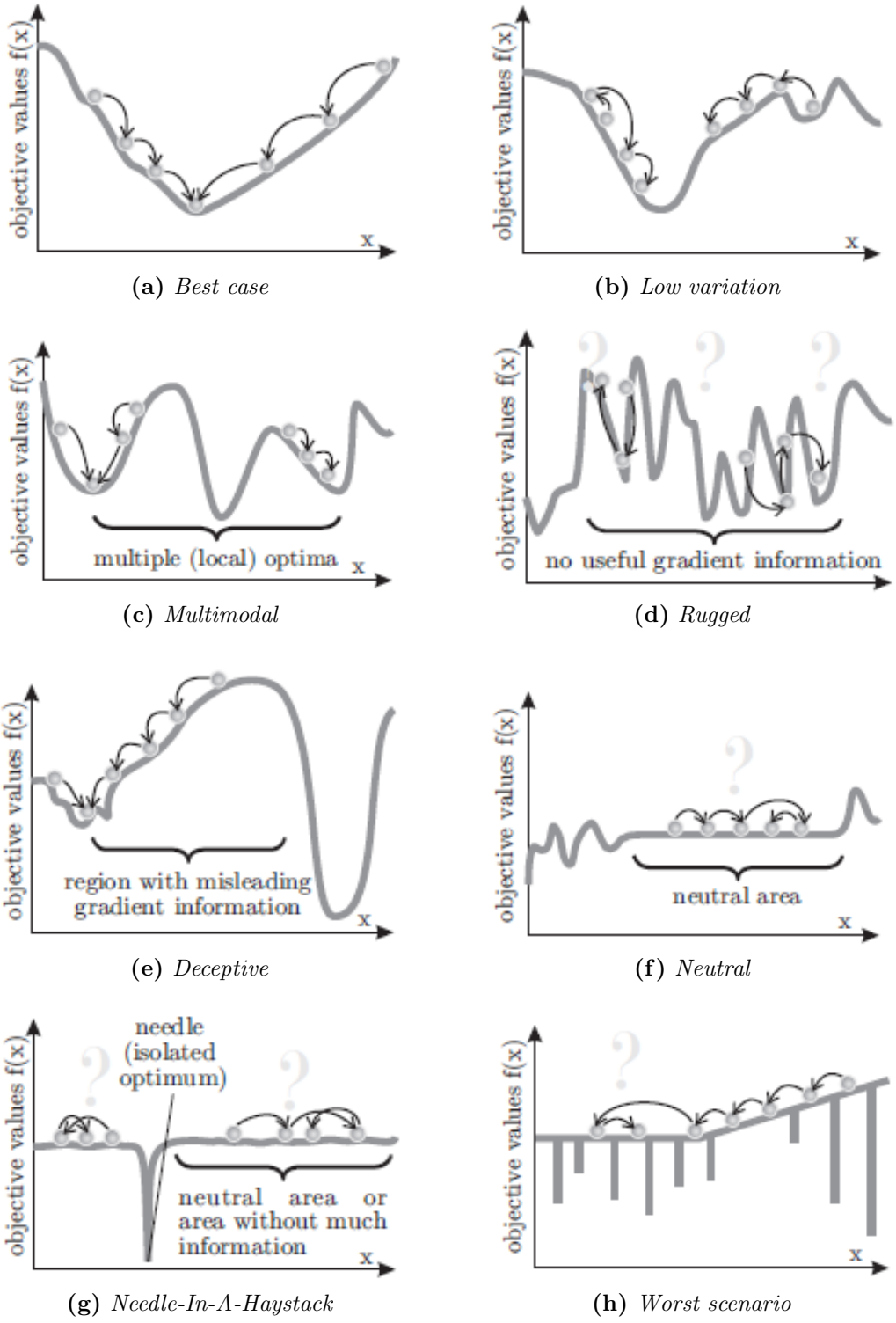


Figure 2.8 – *Global and local optima of a two-dimensional function.*

Even a one-dimensional function $f : X = R \rightarrow R$ may have more than one global maximum, multiple global minima, or even both in its domain X. In many real world applications of metaheuristic optimization, the characteristics of the objective functions are not known in advance. Optimization problems are often multi-modal; that is, they possess multiple good solutions. They could all be globally good (same cost function value) or there could be a mix of globally good and locally good solutions. We can see examples of different functions in Figure 2.9 and possible problems that may occur.



(a) *Best case* **(b)** *Low variation*

(c) *Multimodal* **(d)** *Rugged*

(e) *Deceptive* **(f)** *Neutral*

(g) *Needle-In-A-Haystack* **(h)** *Worst scenario*

Figure 2.9 – *The objective values in the figure are subject to minimization and the small bubbles represent solution candidates under investigation. An arrow from one bubble to another means that the second is found by applying one search operation to the first.*

For our paradigm, a special attention has to be given to the following issues:

Premature Convergence An optimization algorithm converges if it cannot reach new solution candidates or if it keeps on producing solution candidates from a "small" subset of the problem space. One of the problems in optimization is that it is often not possible to determine whether the current best solution is situated on a local or a global optimum and thus, if convergence is acceptable. In other words, it is usually not clear whether the optimization process can be stopped, whether it should concentrate on refining the current optimum, or whether it should examine other parts of the search space instead. This can, of course, only become a problem if there are multiple (local) optima, *i.e.*, the problem is multimodal as depicted in Figure 2.9 (c). A mathematical function is multimodal if it has multiple maxima or minima [24]. A set of objective functions (or a vector function) F is multimodal if it has multiple (local or global) optima (depending on the definition of "optimum" in the context of the corresponding optimization problem).

There is no general approach which can prevent premature convergence. The probability that an optimization process gets caught in a local optimum depends on the characteristics of the problem to be solved and the parameter settings and features of the optimization algorithms applied [25]. A sometimes effective measure is restarting the optimization process at randomly chosen points in time. One example for this method is GRASPs, Greedy Randomized Adaptive Search Procedures [26], which continuously restart the process of creating an initial solution and refining it with local search.

Deceptiveness If an optimization algorithm has discovered an area with a better average fitness compared to other regions, it will focus on exploring this region based on the assumption that highly fit areas are likely to contain the true optimum. Objective functions where this is not the case are called deceptive [27]. The gradient of deceptive objective functions leads the optimizer away from the optimum, as illustrated in Figure 2.9 (e).

Solving deceptive optimization tasks perfectly involves sampling many individuals with very bad features and low fitness. This contradicts the basic ideas of metaheuristics and thus, there are no efficient countermeasures against deceptiveness.

Evolutionary Algorithms Obtaining all (or at least some of) the multiple solutions is the goal of a multi-modal optimizer. Classical optimization techniques due to their iterative approach do not perform satisfactorily when they are used to obtain multiple solutions, since it is not guaranteed that different solutions will be obtained even with different starting points in multiple runs of the algorithm. Evolutionary Algorithms [28] are, however, a very popular approach to obtain multiple solutions in a multi-modal optimization task. There are many different variants of Evolutionary Algorithms. The common underlying idea behind all these techniques is the same: given a population of individuals the environmental pressure causes natural selection (survival of the fittest)

and this causes a rise in the fitness of the population. Given a quality function to be maximised we can randomly create a set of candidate solutions, i.e., elements of the function's domain, and apply the quality function as an abstract fitness measure (the higher the better). Based on this fitness, some of the better candidates are chosen to seed the next generation by applying recombination and/or mutation to them. Recombination is an operator applied to two or more selected candidates (the so-called parents) and results in one or more new candidates (the children). Mutation is applied to one candidate and results in one new candidate. Executing recombination and mutation leads to a set of new candidates (the offspring) that compete, based on their fitness (and possibly age), with the old ones for a place in the next generation. This process can be iterated until a candidate with sufficient quality (a solution) is found or a previously set computational limit is reached.

In this process, there are two fundamental forces that form the basis of evolutionary systems.

- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty.
- Selection acts as a force pushing quality.

The combined application of variation and selection generally leads to improving fitness values in consecutive populations. Such a process can be seen as if the evolution is optimizing, or at least "approximating", by approaching optimal values closer and closer over its course. Alternatively, evolution is often seen as a process of adaptation. From this perspective, the fitness is not seen as an objective function to be optimized, but as an expression of environmental requirements. Matching these requirements more closely implies an increased viability, reflected in a higher number of offspring. The evolutionary process makes the population adapt to the environment better and better. Many components of such an evolutionary process are stochastic. During selection fitter individuals have a higher chance to be selected than less fit ones, but typically even the weak individuals have a chance to become a parent or to survive. For recombination of individuals the choice of which pieces will be recombined is random. Similarly for mutation, the pieces that will be mutated within a candidate solution, and the new pieces replacing them, are chosen randomly. In Figure 2.10 a general scheme in a form of a block diagram can be seen.

In this thesis, we have employed a basic evolutionary algorithm to find the minimum of our error function. A more detailed explanation of our method is available in the following chapter.

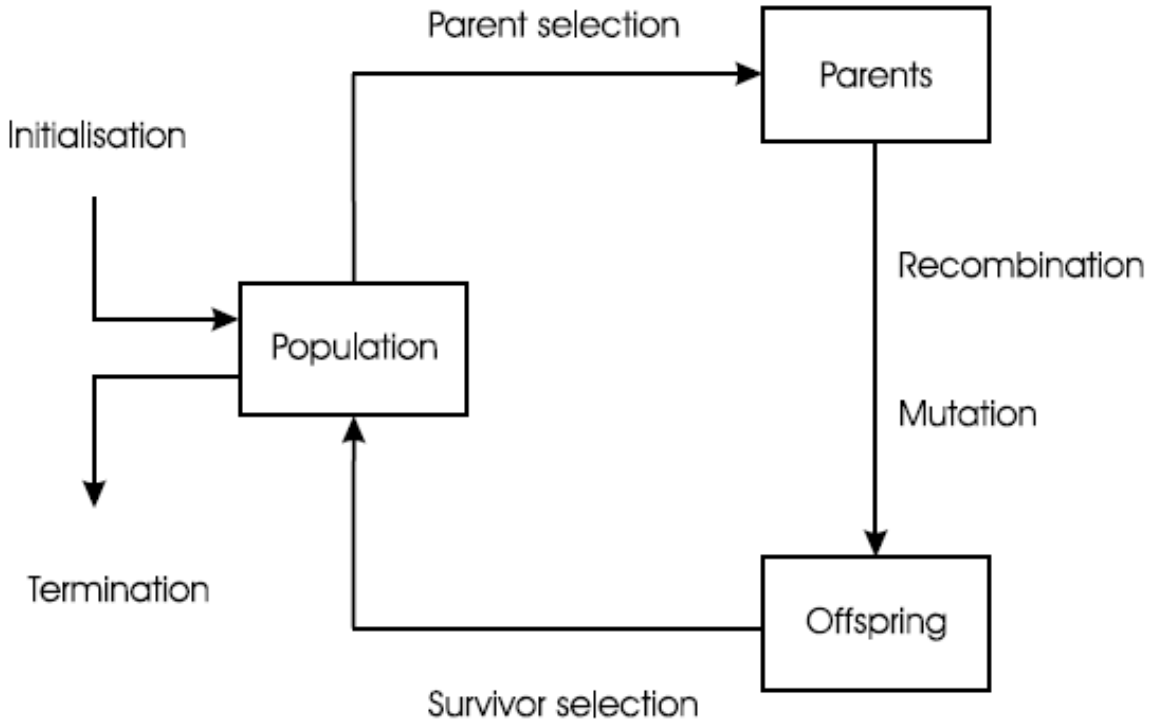


Figure 2.10 – *General scheme of evolutionary algorithms*

3 Implementation

3.1 Overview

The method proposed in this thesis leverages from previous methods and is addressed to specific object recognition and pose estimation applications. The method for recognizing an object and estimating its pose will be designed following these premises and requirements:

- Fast computation in the on-line stage.
- Object to detect constrained to one dimensional movement.
- Few sample features for each track in order to have a correct pose estimation.
- Cut for individual objects, but expandable to class recognition.

This method has two different stages, the off-line stage (Figure 3.1) and the on-line stage (Figure 3.2).

In the off-line stage we:

- Take several images of the object to recognize.
- Extract and match all the feature descriptors.
- Create a track for each feature.
- Estimate a regression function for each track. The function is based on a weight matrix built out of a selection of sample features for each track.
- Use the regression function to estimate feature descriptors in unknown poses.

As a by-product of the building of this regression function, it is also possible, in case any outlier is detected, to substitute it with the appropriate estimated value, so that a more stable track is obtained and tracking failures are reduced.

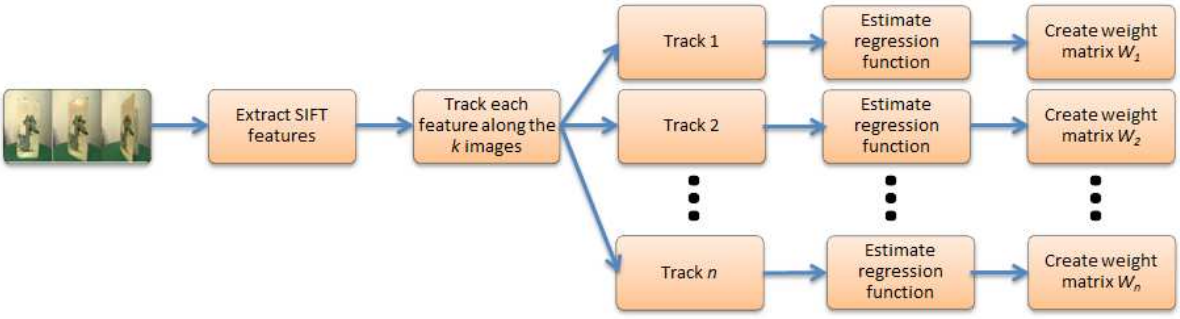


Figure 3.1 – *Off-line stage.*

In the on-line part, a new image is input to the system and its pose is estimated. The weight matrix that was created in the off-line part is used to estimate the pose of the object by regressing on the descriptors associated to the matching database features. We would like to remind the reader that the regression function is built out of as few training sample as possible so that the size of the stored data is kept to a minimum. So, in the on-line part, we:

- Extract the feature descriptors in the new image.
- Compare all the features with the tracks in the training images in order to match a track to a feature.
- Estimate the pose as an inverse regression problem embedded in a Bayesian approach.
- Practically estimate the pose by using an optimization algorithm that finds the minimum Euclidean distance in the feature space.

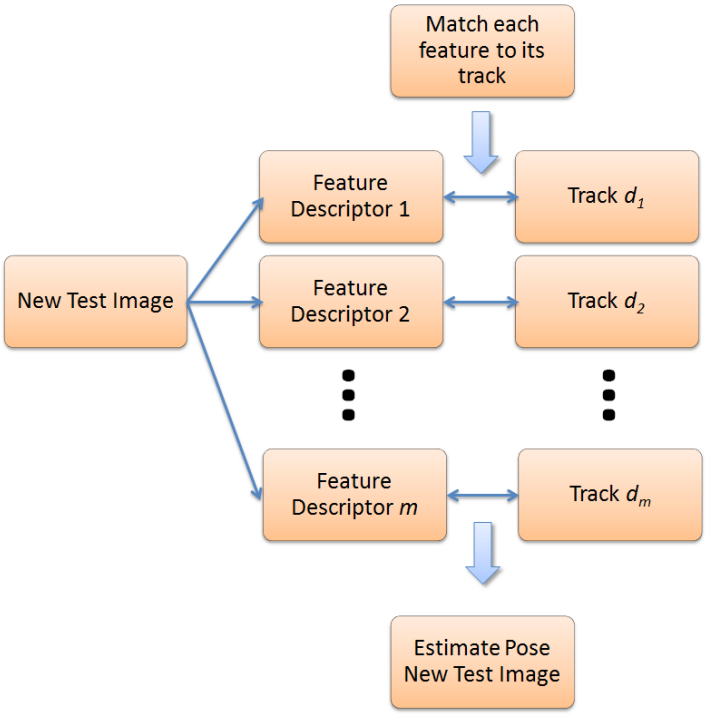


Figure 3.2 – *On-line stage.*

3.2 Implementation: Off-line stage

3.2.1 Creation of the tracks

The first part of the implementation consists on detecting and following the evolution of the feature descriptors in a few selected orientations of one training object as shown in Figure 3.3. Every feature descriptor consists of a vector of 128 values, plus its orientation, scale and relative position (x, y) in the image. By using a previously created program named *kpmatcher*, we can, having two images as input, extract all the matching SIFT features between them.

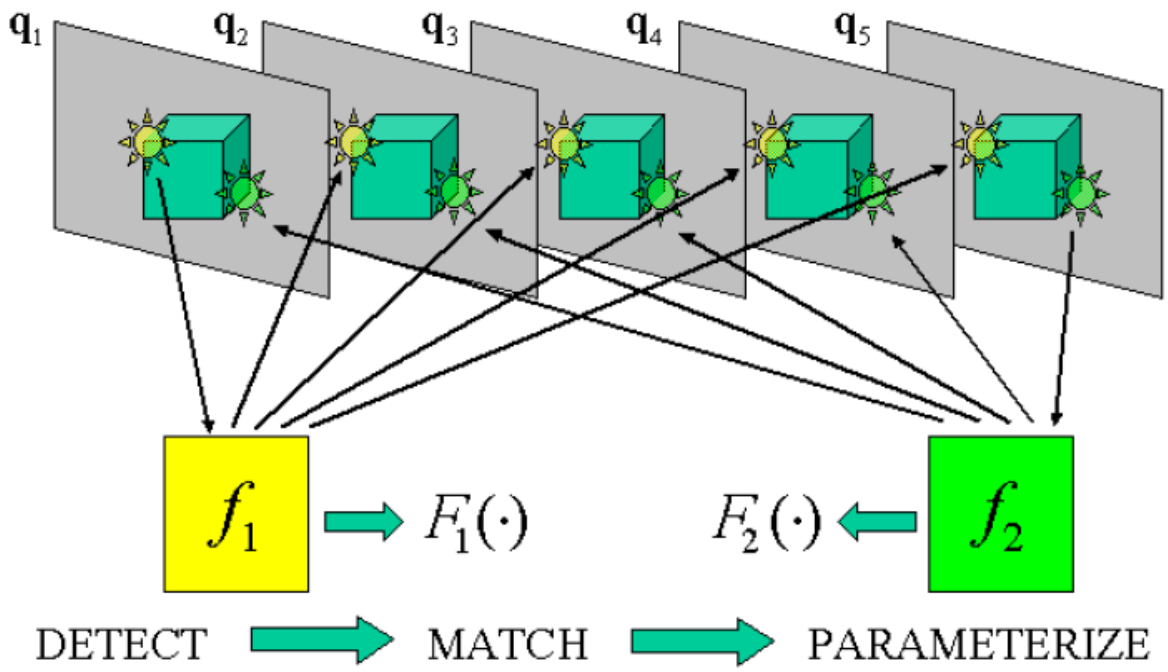


Figure 3.3 – *Tracking the features from a set of images (top rectangles). Each feature is extracted and matched, and modeled using a generative model.*

One problem that can occur is that gaps may exists in the detection of a feature descriptor through several images. That is, if we have n_m matches between images 1 and 2, it is possible that one feature that appears in image 2 and has a matching feature in image 1, does not have a match in image 3. This feature could reappear in the matching between 3 and 4. So, in order to know that this feature relates to those previously found and to be able to create a long track, we cannot compare the images only in a sequential way (i.e. 1-2, 2-3, 3-4, ...), but also to the following images (i.e. 1-2, 1-3, 1-4, ...,2-3,2-4, ...).

As output of the *kpmatcher* program, two files are obtained. One contains all the matching feature descriptors that belong to the first image (here named *bok12*). The other file contains the matching feature descriptors for the other image (here named *bok21*).

In order to follow a track, we compare each 128-dimensional vector yielded by a matching between two images (*e.g.*, 3-4) with the following one (*e.g.*, 4-5). In order to do this, we have to go through the file *bok21* of the first matching (3-4), compare every line (each line contains a feature descriptor vector) with all the lines of the file *bok12* of the following matching (4-5). Instead of using the 128 components for the comparison, we can use only the position (x, y) and the orientation of the feature descriptor, since it gives enough accuracy to distinguish among all the features descriptors in the image. We do not use only the position of the feature descriptor because it may be that two

features share the same position but have different orientation values.

If we detect a gap in the track, that is, we are unable to find the next correspondence for one feature descriptor, we look in the previous images in order to "jump" this gap (Figure 3.4). For example, if we are following a track between images 1-2-3-4 but we cannot find the continuation of this track in image 5, we refer to the next image (6) and, by using the file *bok12* between the last image on which we found the feature (4 in this case) and image 6, we try to find if this feature appears again in image 6. If it does not appear, we try to find the same feature but now in the *bok12* file between images 3 and 6 and so on until we can re-establish the track. If we cannot overcome this gap we move to the following image (7) and we use the *bok12* file using the same method as before. The difference is that now we use image 7 instead of 6.

We have empirically determined that, for the database used, the average angular length of a feature descriptor track is approximately 33 degrees, as the object rotates in one direction. So, in order to save time for comparisons, and as the orientation of our training object is exactly known, gap solving can be stopped when a difference of more than 33 degrees of orientation between the images is reached. This saves us time.

A *.txt* file containing the evolution of the feature descriptor along the training images is created. Every line of this file contains the 128 components of the descriptor along with the corresponding image or orientation of the object to which it belongs. For each track, its corresponding *.txt* file is stored.

In order to have a complete set of tracks for all the orientations of the training object, in every new training image, it is necessary to store the new features descriptors that appear and still do not belong to any track (Figure 3.5).

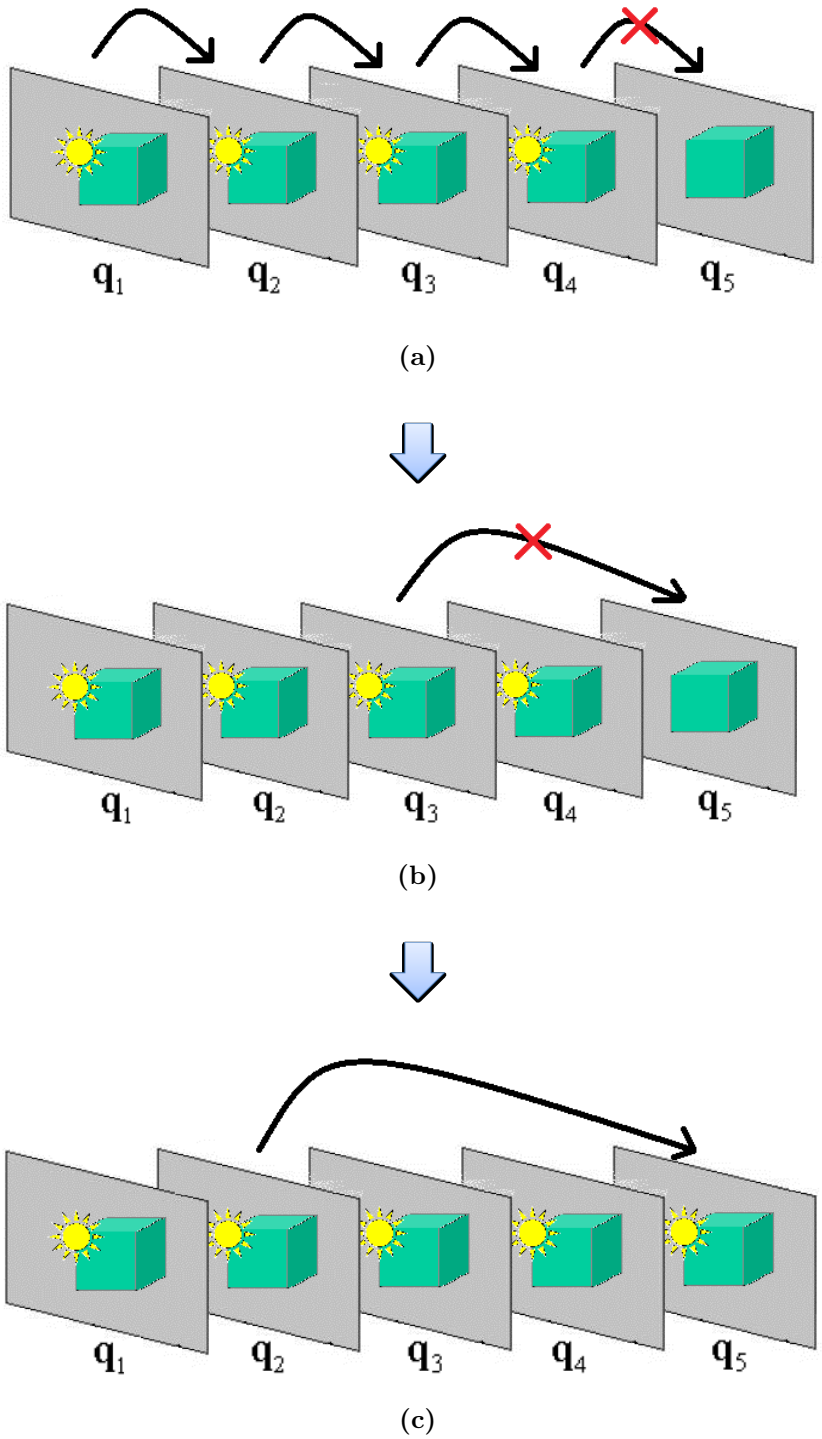


Figure 3.4 – *Example of gap problem solving. In (a) the track of the feature is lost in pose q_5. We will go to the previous more similar pose (b), and try to recover. We will go further away (c) until a match arise*

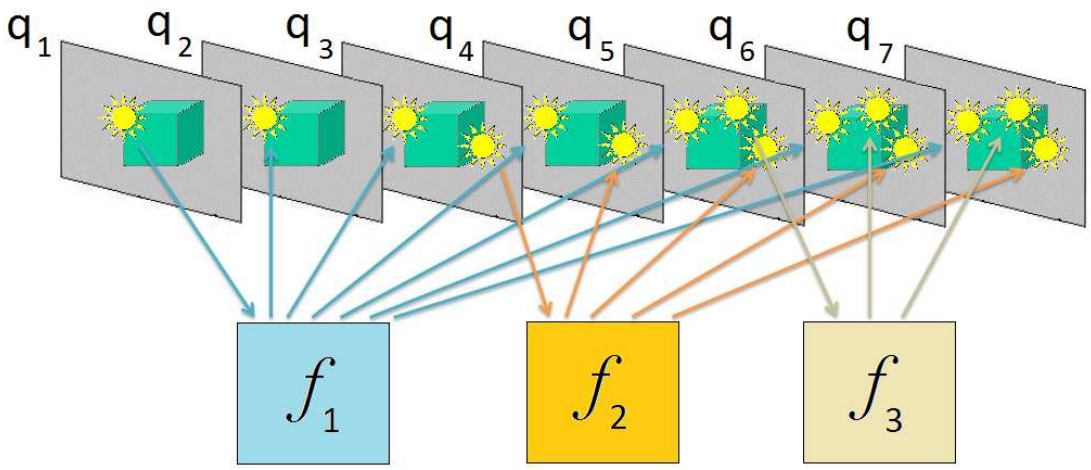


Figure 3.5 – *In poses q_4 and q_6 new features appear. It is necessary to create new tracks for them.*

Once we have established a track for each feature in the training images, it is possible now to study them and determine a regression function that fits each individual behavior.

3.2.2 Computation of the regression function

We now turn our attention to the problem of inferring a generative feature model. The goal is to learn a pose-dependent model of a scene feature, given a set of observations of the feature from known camera positions. The model has to be capable of producing maximum-likelihood virtual observations (predictions) of the feature from previously unvisited poses. It will also be used for estimating the likelihood of a new observation $p(z_i \mid q)$, given the pose q from which it might have been observed.

Any observation $\bar{z}$ of a feature f is represented only by its 128-valued descriptor, neglecting any information regarding its position, scale or orientation:

$$\bar{z} = [v_1 \quad v_2 \quad \ldots \quad v_{128}] \tag{3.1}$$

The observation $\bar{z}$ can be considered as the output of a vector-valued function $F(\cdot)$ of the camera pose q. The goal is to learn an approximation $\hat{F}(\cdot)$ of this function. As this method is intended for being used as a fast method for pose recognition in the framework that was outlined in the introduction, only a one-dimensional parameter, *i.e.*, the rotation of the object around its central axis, will be considered for the pose.

The approach for learning $\hat{F}(\cdot)$ is to model each element of the feature vector $\bar{z}$ as a linear combination of radial basis functions (RBFs), each of which is centered at a particular pose of the object determined by the set of training poses.

Given a set of training images (observations), a set of weight vectors w_i can be computed such that a linear combination of RBF's interpolates the observations, approximating the function that generated the observations. Formally, given a set of observations from known poses (z_i, q_i), a predicted observation $\bar{z}$ from pose q is expressed as:

$$\bar{z} = \hat{F}(\cdot) = \sum_{i}^{k} w_i G(q, q_i) \tag{3.2}$$

where k is the number of training poses, and an exponentially decaying RBF $G(\cdot, \cdot)$ is used:

$$G(q, q_i) = \exp\left(-\frac{\|q - q_i\|^2}{2\sigma^2}\right) \tag{3.3}$$

where q_i represents the center of the RBF (in the observation i), and the response of the RBF is measured as a function of q. The width, or influence, of the RBF is defined by σ.

For the computation of the weight vectors w_i, interpolation theory and works such as [29] are resorted to. In brief, the optimal weights $W = [w_{ij}]$ are the solutions to the linear least squares problem

$$(G + \lambda I)W = Z \tag{3.4}$$

where the elements G_{ij} of the design matrix G correspond to the previous equation (3.3), evaluated at observation pose i and RBF center j, the matrix W corresponds to the matrix of the unknown training weights, and the rows of the matrix Z correspond to the training observations. When λ is 0 and G^{-1} exists, the computed weights result in a network whereby Equation 3.2 exactly interpolates the observations. However, the presence of noise and outliers and the complexity of the underlying function being modeled, can result in an interpolation which is highly unstable. The solution can be stabilized by adding a diagonal matrix of regularization parameters λI to the design matrix G. These regularization parameters and the RBF width σ are set following the experiments presented in Chapter 4. While ridge regression can be employed to compute the optimal regularization parameters, empirical experience indicates that this approach is not necessary for the distributions of measurements that are being interpolated.

For computational savings as well as data storage, but at the cost of reduced accuracy, the number of RBF centers can be limited to a subset of the observation poses. An evaluation of the performance of the system given different choices is given in the experimental part of this thesis in Chapter 4. We need to find a compromise between

accuracy, amount of data and speed.

By using Equation 3.4, it is possible to calculate the weight matrix W as follows:

$$W = Z(G + \lambda I)^{-1} \tag{3.5}$$

In the end, for each feature track t_x, it is only necessary to store its weight matrix W_x, a set of features in determined poses stored in Z_x and an index vector that connects each row of Z_x to its ground-truth pose.

$$Z = \begin{bmatrix} \bar{z}_1 \\ \bar{z}_2 \\ \bar{z}_3 \\ \vdots \\ \bar{z}_k \end{bmatrix} = \begin{bmatrix} v_{1_1} & v_{1_2} & \cdots & v_{1_{128}} \\ v_{2_1} & v_{2_2} & \cdots & v_{2_{128}} \\ v_{3_1} & v_{3_2} & \cdots & v_{3_{128}} \\ \vdots & \vdots & & \vdots \\ v_{k_1} & v_{k_2} & \cdots & v_{k_{128}} \end{bmatrix}$$

$$W = \begin{bmatrix} w_{1_1} & w_{1_2} & \cdots & w_{1_{128}} \\ w_{2_1} & w_{2_2} & \cdots & w_{2_{128}} \\ w_{3_1} & w_{3_2} & \cdots & w_{3_{128}} \\ \vdots & \vdots & & \vdots \\ w_{k_1} & w_{k_2} & \cdots & w_{k_{128}} \end{bmatrix}$$

Each of these matrices is stored into separate *.txt* files indicating the ID of the track to which they belong.

As a by-product, it is possible to evaluate the quality of each track Z_x. Each feature model is evaluated using a leave-one-out cross-validation approach, which operates by constructing the model with one data point z excluded, predicting that data point z^* using the regression function and measuring the difference $e = \|z - z^*\|$ between the actual point and the prediction. By iterating over several (ideally all) training samples, and computing the covariance σ_e^2 of the resulting error measures, we can build up a measure of how well the model fits the data and, more importantly, how well we might expect it to predict new observations. The model covariance σ_e^2 is defined as:

$$\sigma_e^2 = \frac{1}{k} \sum_{i=1}^{k} e_i e_i^T \tag{3.6}$$

where k is the number of observations of the feature and e_i is measured for each removed observation i.

When the construction of the regression function is finished, we are capable to estimate the pose of a new image.

3.3 Implementation: On-line stage

3.3.1 Identify each track

When a new test image is input to the system, the first step consists in extracting its $\{f_n\}$ SIFT features, where n is the number of SIFT features found in the image. To establish matches with the database, it is necessary to compare this new test image with one of our k training images in order to determine which feature of the test image corresponds to which track z_x of the training images. The program *kpmatcher* is executed again between the two images (the new one and the one of the training images) and the *bok12* and *bok21* files are considered. We use the *bok* file corresponding to the stored values of the training features in order to search in each matrix Z_x at the appropriate pose (the one of the training image) and find the matching track (it will share the same 128 values). Once the corresponding track is found, it is possible to perform pose estimation by comparing it against the corresponding feature in the test image, which is stored at the same line of the other *bok* file. An example can be seen in Figure 3.6.

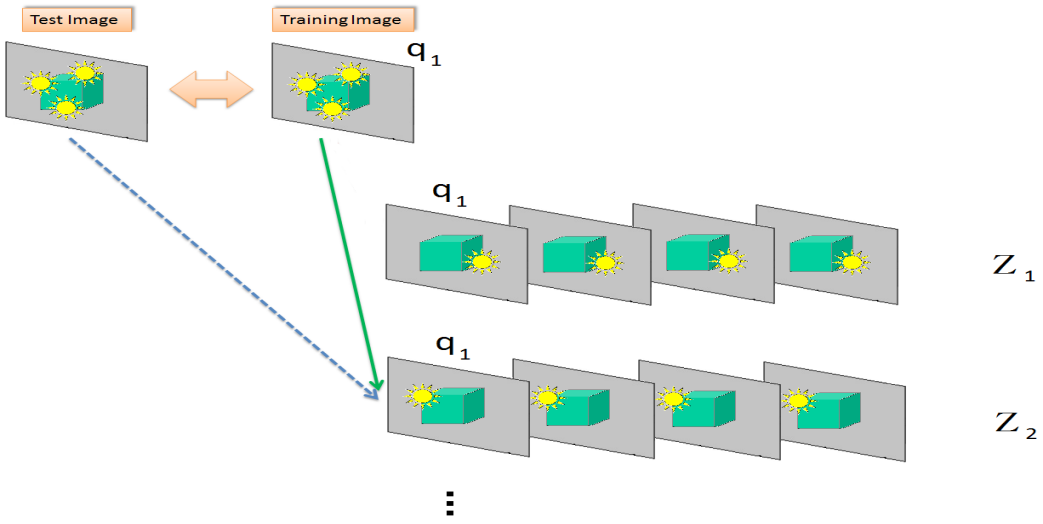


Figure 3.6 – *A number of features arises in the matching between the new test image and the training image (in this case in pose q_1 of the object). The training image is used to match each track with the corresponding test feature (marked as a green arrow). Then, it is possible to establish a correspondence between the feature in the new image and the appropriate track (blue dotted arrow)*

3.3.2 Estimation of the pose

The second goal of the feature learning framework after object recognition is to achieve an accurate pose estimation. Given an observation z, the probability distribution over object poses can be constructed from Bayes' Rule, as

$$p(q_x \mid z) = \frac{p(z \mid q_x)p(q_x)}{p(z)} \tag{3.7}$$

where $p(q_x)$ is the *a priori* distribution over object orientations, and $p(z)$ is independent of q_x, and hence treated as a normalizing constant. The pose q_x can be estimated by maximizing the probability:

$$q^* = \arg\max_q p(q_x \mid z) \tag{3.8}$$

Since it is not possible to have any prior information about the realization of the feature descriptors of the test image, it is possible to simplify Equation 3.7 to

$$p(q_x \mid z) \propto p(z \mid q_x) \tag{3.9}$$

Pose inference, on the basis of the observation of a set of image features, can be accomplished by assuming that the observation model $p(z \mid q_x)$ is approximated by the joint likelihood of the set of feature observations conditioned on the pose q_x:

$$p(q_x \mid z) \propto p(z \mid q_x) \simeq p(z_1, z_2, ..., z_t \mid q_x) \tag{3.10}$$

The previous formula is assumed to be an approximation because we ignore any information that might be present in parts of the image other than those occupied by the detected features. Additionally, we can assume conditional independence between the individual feature observations, even though there can be some joint dependence in the way feature descriptors change. As a matter of fact, similar patterns on the same surface of the image may change their appearance in a consistent way as the object changes its pose. All these topics are definitely worth to be addressed in future research.

The probability of an observed image is thus defined to be the joint likelihood of the individual observations:

$$p(q_x \mid z) \propto p(z \mid q_x) = \prod_i^t p(z_i \mid q_x) \tag{3.11}$$

In the absence of informative prior, the pose q_x that maximizes the joint likelihood of the observations is considered to be the maximum likelihood pose of the object. It is not

clear, however, if the conditional independence assumption holds for features derived from a single image and, furthermore, if outliers can lead to catastrophic cancellation of the joint distribution. Therefore, we employ a mixture probabilistic model defined by

$$p(q_x \mid z) \propto p(z \mid q_x) = \frac{1}{t} \sum_{i}^{t} p(z_i \mid q_x) \tag{3.12}$$

where all features are given the same weight.

Since each feature f in the image helps determining the correct pose of the object, the quantity of matched features is a critical point in the method as few matching features are not likely to provide reliable results.

By taking into account the way SIFT feature matching is done between two images, the Euclidean distance is measured between the feature in the test image and its corresponding feature in the track as

$$d_i = \|\bar{z}_i - z_i\|^2 \tag{3.13}$$

where $\bar{z}_i$ is the descriptor of the feature i in the training image and z_i is the descriptor of the matching feature in the test image.

As outlined above, the method can be embedded into a Bayesian framework, so it is possible to produce a measure that describes the likelihood of any object pose as follows:

$$p(z_i \mid q_x) = \sum_{i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{d_i^2}{2\sigma^2}\right) \tag{3.14}$$

The objective is to maximize Equation 3.14, as we had seen in Equation 3.8

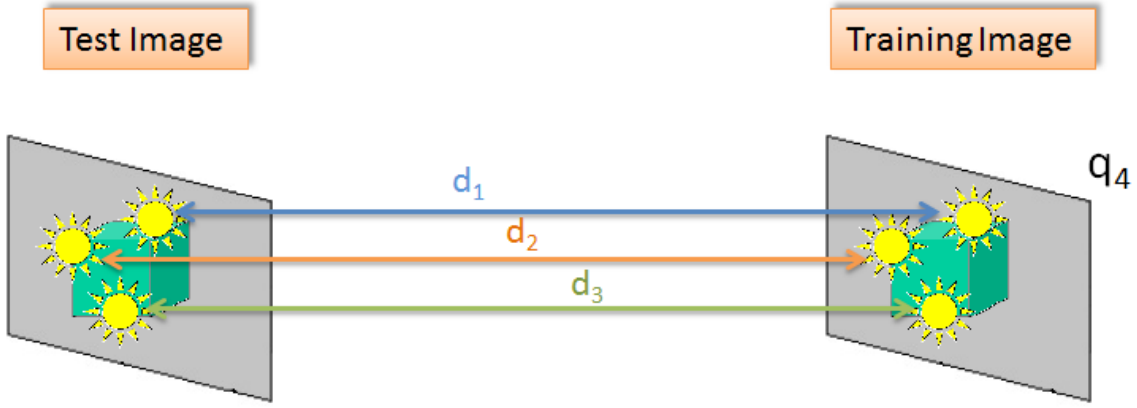
3.3.3 Maximization algorithm

In order to search for the optimum pose, an optimization algorithm is to be used. As we had seen, the pose similarity can be probabilistically measured by comparing the Euclidean distances of the features z_i under study in the test image and its corresponding feature in the training image $\bar{z}_i$. The smaller the error, the higher the probability that z_i corresponds to the pose to which $\bar{z}_i$ belongs. This probability is more accurate as more features are used at that pose. So, our goal is to maximize the following probability:

$$q^* = \arg\max_{q} \frac{1}{t} \sum_{i}^{t} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{d_i^2}{2\sigma^2}\right) \tag{3.15}$$

where t is the number of tracks used in the pose. Equation 3.15 can be simply seen as a

minimization of the average Euclidean distance $d_i = ||\bar{z}_i - \hat{z}_i||^2$. In Figure 3.7, a simple example is given where three features are used.



$$p(z \mid q_4) = \frac{1}{3} \sum_{i}^{3} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{d_i^2}{2\sigma^2}\right)$$

Figure 3.7 – *Features* 1, 2 *and* 3 *are used to compute the probability that the test image (whose pose is unknown) relates to pose* 4 *of the object*

Usually the function to be minimized is not perfectly convex and therefore several local minima exist. To overcome this problem, an approach leveraging from evolutionary algorithms is used.

In order to estimate the best pose, the track Z_x for each feature descriptor is divided in a set of sub-tracks Z_{x_1}, Z_{x_2}, ... Z_{x_w} (Figure 3.8). The optimization algorithm is used on all sub-tracks to determine the local minimum. Comparing all local minima and choosing the fittest will give us the best result.

The length of the window used for each sub-track is chosen considering the average length of the tracks. This windowed approach, apart from solving the local minimum problem, provides more tracks to perform the comparison (a critical point in the algorithm), as not only the most robust tracks will be used (*i.e.* the ones that comprise all the set of training poses), but also the shorter ones that only comprise a subset of the orientations. For example, in Figure 3.8, not only the tracks that contains poses from 1 to 8 will be used, but also the less robust tracks covering poses from 1 to 4 or from 4 to 8.

The program *kpmatcher* is executed for each sub-track using the median image of the interval to match each feature f of the test image to its corresponding track parallel to what was explained in Section 3.3.1.

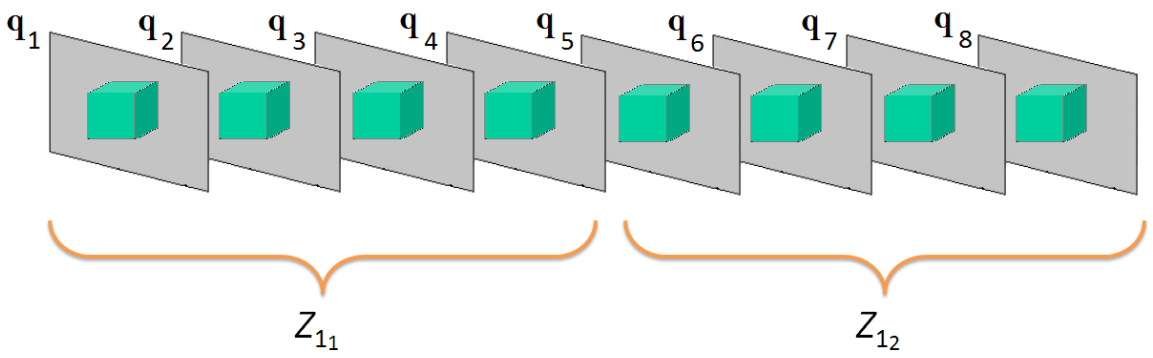


Figure 3.8 – *The track* Z_1*, containing poses from* 1 *to* 8*, will be divided in the tracks* Z_{1_1} *and* Z_{1_2} *using a window of length* 4*.*

Once all matches are identified, first the values z are compared to the training orientations that are available in the sub-track. Then, with a first-order optimization algorithm based on gradient descent, the maximum probability of the pose is determined. To do so, the regression function which estimates the descriptors at the unknown poses and an eventual comparison in the feature space is employed. The optimization algorithm is executed in every sub-track to identify the local minima. The method is based on the gradient descent algorithm, but with some changes. Instead of starting the minimization in a random point of the function, we start at the median image of the window. We then compute the average error using all available tracks (Figure 3.9), and move forward (or backward) until the value is higher than the current one. When this happens, we change the direction and reduce the size of the step. In Figure 3.10, a representation of the implementation can be seen. The search ends when the step change achieves a minimum value depending on the set precision. When all the windows have reached a solution, the minimum value is chosen.

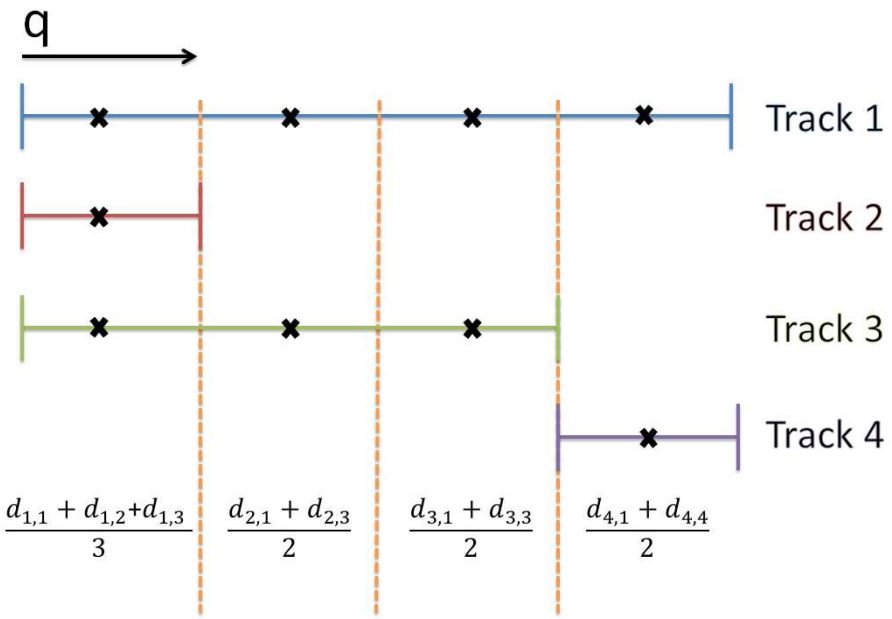


Figure 3.9 – *The dotted orange lines indicate the limits of the window. The minimization of the error starts at the median image of each the window. An average of the error using all the tracks available is done in each window to know the next step. Then the error is computed in the same way in the next iteration of the optimization algorithm.*

The error function is minimized over all the features used in that window. The higher the number of features used to compute the error, the better the result will be.

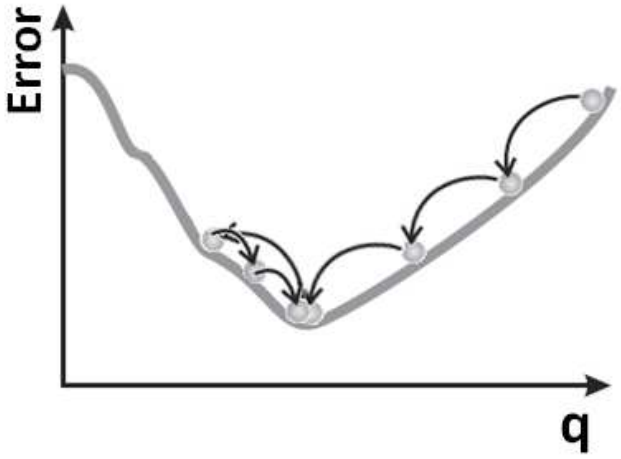


Figure 3.10 – *In each window, the minimization algorithm is implemented. The minimization is performed using all features available in that window.*

4 Experiments and results

4.1 Parameter estimation

In our framework for object detection and pose estimation, there are some parameters of the regression function that have to be set in each case. These parameters are:

- λ, needed for calculating the weight matrix W (equation 3.4)
- σ, needed in the scalar exponential function G (equation 3.3)

It is possible to find the best value of these parameters in the off-line part of the implementation using a leave-one-out cross-validation technique. This operates by constructing the model with one data point excluded and using it as validation data and the remaining observations as training data. This data point will be predicted by our regression function using the training data and different values of σ and λ. The value of σ results different depending on the training object. The best value of λ was found to be 0.012

4.2 Results

For our experiments we have two datasets available:

- Three different objects (Figure 4.1) used to determine the best performance of our method. Each image has been taken every $D = 5$ degrees (Figure 4.3), including only the frontal 180° range of the object, with a total number of 36 images.
- A complete dataset of car sequences, representing the typical problem of pose estimation for vehicles. The dataset can be downloaded at the following web page *http://cvlab.epfl.ch/data/pose/*. Two different instances of a car are used (seq_1 in Figure 4.2 (a) and seq_19 in Figure 4.2 (b)), having a difference of $D = 3.16$ (seq_1) and $D = 3.7$ degrees (seq_19) degrees of rotation between consecutive images.

(a) *Book* **(b)** *Mouse Folder* **(c)** *Plane Box*

Figure 4.1 – *One training image for each different object.*

(a) *Sequence 1* **(b)** *Sequence 19*

Figure 4.2 – *Two training images of the car dataset.*

For the first dataset containing the three objects, different tests have been devised:

- Different number of images taken.
- Different lengths of the window.
- Change in illumination.
- Partial occlusion.
- Cluttering.

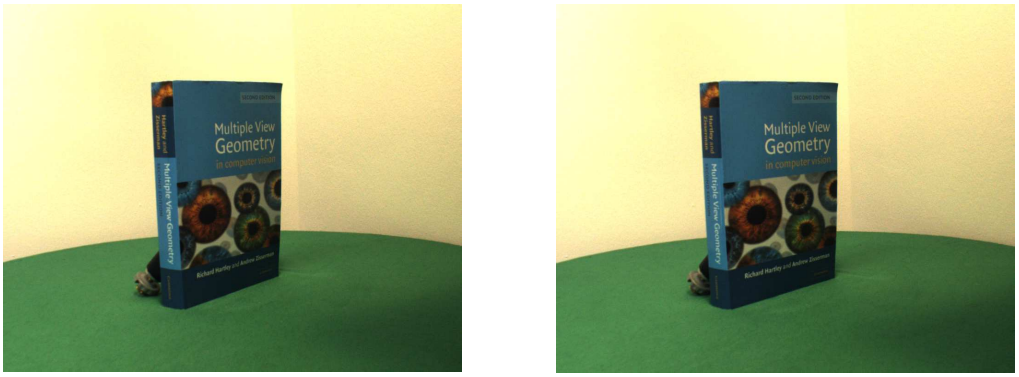


Figure 4.3 – *Every image on the object is taken with a* 5 *degree separation from the previous one.*

For the second dataset, as the original model is not available, it is only possible to work with the given images, without any possibility of taking new ones in different scenarios. For this dataset, we will show results for the following different tests:

- Different number of images taken.
- Different window lengths.
- Different car instances.

We will consider an average error in the estimation of the pose higher than 5º as a failure in the estimation. The following tables are a summary of the test objects and parameters

Car sequence 1		
Test name	**Difference between images (D) (º)**	**Window length (L) (º)**
Seq1_1_1	6.32	12.64
Seq1_1_2	6.32	18.96
Seq1_1_3	6.32	25.28
Seq1_1_4	6.32	31.6
Seq1_1_5	6.32	37.92
Seq1_2_1	9.48	18.96
Seq1_2_2	9.48	28.44
Seq1_2_3	9.48	37.92
Seq1_2_4	9.48	47.4
Seq1_3_1	12.64	25.28
Seq1_3_2	12.64	37.92
Seq1_3_3	12.64	50.26
Seq1_4_1	15.8	31.6
Seq1_4_2	15.8	47.4
Seq1_5_1	18.96	37.92
Seq1_5_2	18.96	56.88
Seq1_6_1	22.12	44.24
Seq1_6_2	22.12	66.36
Seq1_7_1	25.28	50.56
Seq1_7_2	25.28	75.84
Seq1_8_1	28.44	56.88

Car sequence 19		
Test name	**Difference between images (D) (º)**	**Window length (L) (º)**
Seq19_1_1	7.4	14.8
Seq19_1_2	7.4	22.2
Seq19_1_3	7.4	29.6
Seq19_1_4	7.4	37
Seq19_1_5	7.4	44.4
Seq19_1_6	7.4	51.8
Seq19_1_7	7.4	59.2
Seq19_2_1	11.1	22.2
Seq19_2_2	11.1	33.3
Seq19_2_3	11.1	44.4
Seq19_2_4	11.1	55.5
Seq19_2_5	11.1	66.6
Seq19_3_1	14.8	28.6
Seq19_3_2	14.8	44.4
Seq19_3_3	14.8	59.2
Seq19_4_1	18.5	37
Seq19_4_2	18.5	55.5

Objects		
Test name	**Difference between images (D) (º)**	**Window length (L) (º)**
Objects_1_1	10	20
Objects_1_2	10	30
Objects_1_3	10	40
Objects_1_4	10	50
Objects_1_5	10	60
Objects_2_1	15	30
Objects_2_2	15	45
Objects_2_3	15	60
Objects_2_4	15	75
Objects_3_1	20	40
Objects_3_2	20	60
Objects_3_3	20	80
Objects_4_1	25	50
Objects_4_2	25	75
Objects_4_3	25	100
Objects_5_1	30	60
Objects_5_2	30	90
Objects_5_3	30	120
Objects_6_1	35	70

4.2.1 Car Dataset: Sequence 1

The first step is to determine the best value of the parameter σ. Using the leave-one-out technique, we measured the average error using different σ, and we chose the one that gave us the smallest error. For this dataset a value of 120 is used.

We keep this value of σ fixed, and using the leave-one-out technique, we compare the results using different window lengths (L) and different rotation differences in degrees per consecutive image (D). As the database used is closed, it is possible only to use multiples of 3.16°. The first step will be to compare different window lengths using the same difference in rotation. A plot of the results showing the average error in degrees for the estimated pose can be seen in Figure 4.4. A detailed table with the average of the error and the standard deviation in each experiment can be seen next.

Car sequence 1		
Test name	**Average error (°)**	**Standard deviation (°)**
Seq1_1_1	2.15	4.18
Seq1_1_2	0.85	0.92
Seq1_1_3	1.00	0.97
Seq1_1_4	44.14	77.23
Seq1_1_5	52.63	72.27
Seq1_2_1	1.15	1.10
Seq1_2_2	1.93	2.34
Seq1_2_3	21.85	47.64
Seq1_2_4	139.20	61.38
Seq1_3_1	2.27	1.87
Seq1_3_2	21.60	49.15
Seq1_3_3	82.90	66.44
Seq1_4_1	15.8	31.6
Seq1_4_2	15.8	47.4
Seq1_5_1	3.49	4.52
Seq1_5_2	89.31	70.31
Seq1_6_1	5.62	16.15
Seq1_6_2	84.54	74.60
Seq1_7_1	9.66	37.45
Seq1_7_2	96.68	69.53
Seq1_8_1	67.86	68.06

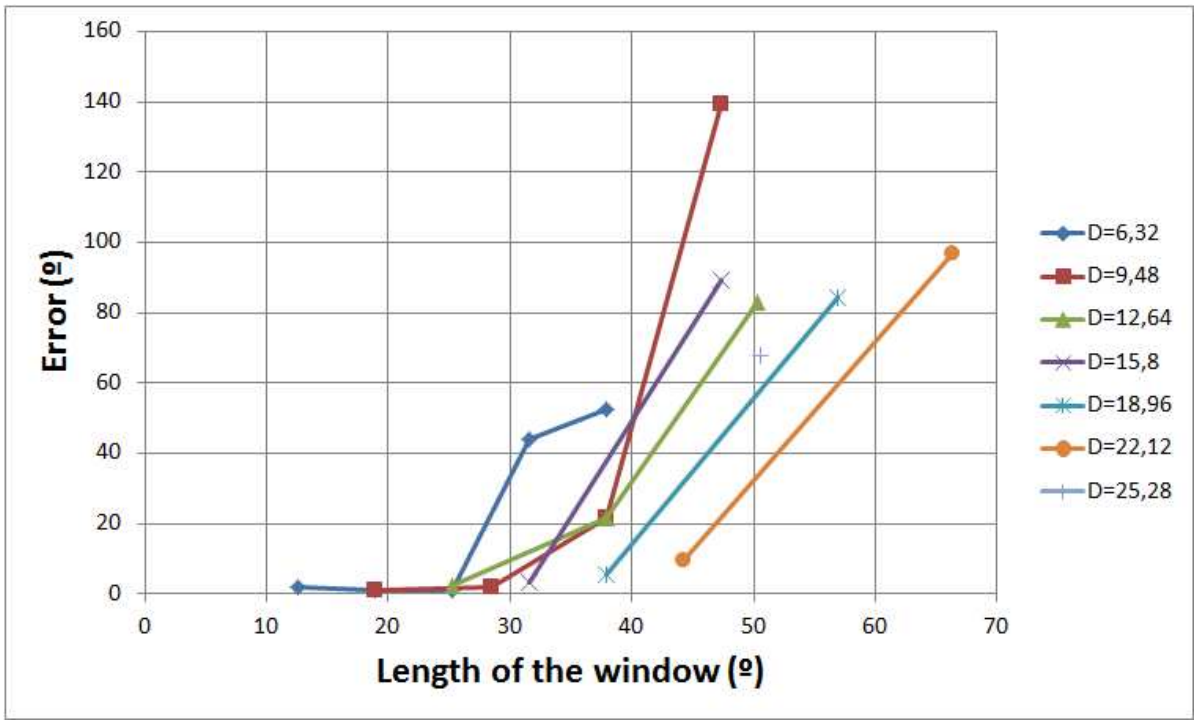


Figure 4.4 – *Graphic car dataset seq_1*

It can be seen that although the results choosing low values of L and D are close to the real, when the window increases the values worsen. It is possible to give an explanation for this if a deeper look at the results is taken. For example, in the test Seq1_2_3:

Estimated image	**Error in degrees**
23	2.9015
24	2.0191
26	1.3574
27	0.94126
29	1.0178
30	1.3574
32	102.12
33	105.28
35	111.6
36	1.2132
38	0.44519

Estimated image	**Error in degrees**
39	0.75652
41	0.31056
42	1.3574
44	140.04
45	0.92659
47	149.52
48	152.68
50	1.3574
51	0.18071
53	0.63745
54	0.74135
56	6.4403
57	3.16
59	1.0165
60	0.25854
62	0.45613
63	0.94127
65	1.0177
66	1.3574
68	0.3119
69	3.16
71	2.1484
72	2.9793
74	2.2106
75	0.94127
77	3.16

The image number corresponds to the sample that it is left out in the estimation. The indices that do not appear refer to the tracks used in the estimation and thus the error is 0, so their value is not taken into account. The absolute orientation of the object in each image is given by

$$\text{Object Orientation} = \text{Image Number} \times \text{Rotation between consecutive images} \quad (4.1)$$

So, for example, image number 60 corresponds to a rotation of the object of 189.6 degrees. The orientation can be selected freely as far as the same rule is followed for the whole dataset. As we can see, there are few big outliers in the estimation and therefore, the mean error increases a lot. By considering only the best 80% results (80% percentile), the average error would decrease to $1,52$. The reason for the error is due to the geometry of the object. For objects like cars, problems occur because of their inherent symmetry. As it can be seen in Figure 4.5, a lot of matches arises in parts like car tires.

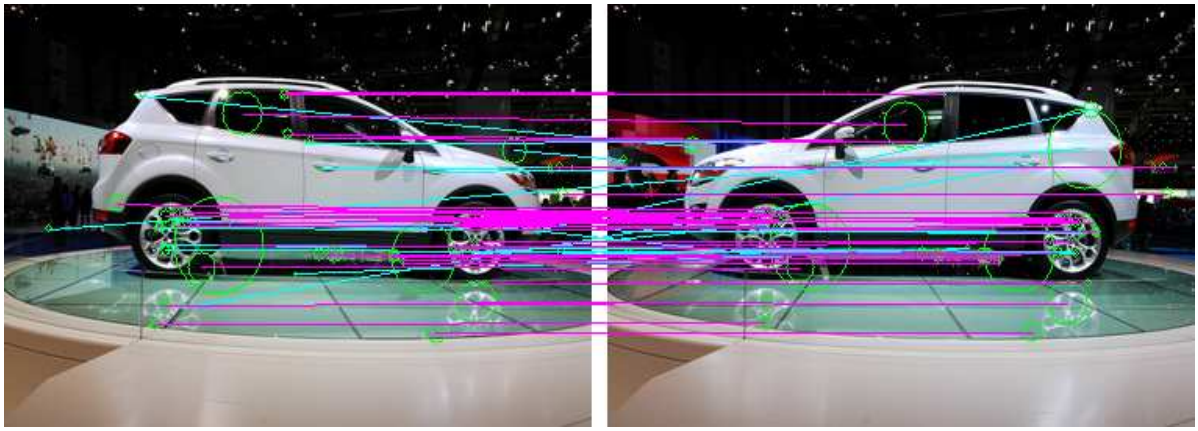


Figure 4.5 – *Problem of symmetry in the car object*

Another cause of failure is the number of tracks used. Let's consider the tracks used in each window for the test Seq1_2_3.

Estimated image	**W1 (22-34)**	**W2 (34-46)**	**W3 (46-58)**	**W4 (58-70)**	**W5 (70-78)**
23	8	0	0	1	4
24	9	0	0	0	5
26	7	0	0	0	2
27	9	0	0	0	5
29	9	3	1	1	2
30	9	4	2	2	5
32	4	2	3	1	3
33	5	3	2	2	3
35	2	3	2	2	0
36	2	6	2	2	2
38	4	7	2	2	3

Estimated image	**W1 (22-34)**	**W2 (34-46)**	**W3 (46-58)**	**W4 (58-70)**	**W5 (70-78)**
39	1	7	2	2	2
41	2	7	2	2	2
42	0	7	3	2	2
44	3	5	3	2	2
45	1	6	1	2	2
47	1	4	4	2	2
48	1	3	5	2	2
50	1	2	6	2	1
51	0	2	7	2	0
53	2	3	7	4	2
54	0	2	6	3	2
56	3	2	3	6	3
57	3	2	5	8	2
59	1	2	3	7	0
60	2	2	4	10	2
62	3	2	5	9	2
63	3	3	5	10	3
65	2	2	4	10	5
66	3	2	4	10	7
68	5	2	4	8	8
69	2	2	4	9	10
71	2	2	5	8	10
72	5	2	4	6	13
74	1	2	2	4	10
75	4	2	0	0	13
77	5	2	2	2	12

The first thing to notice is the number of tracks that can be used depending on the position of the image to estimate. The closer to the ground truth image the higher the number of tracks resulting from the matching stage. Again, the problem due the object symmetry can be seen in the first images taken.

Experiments show that this method fails in 50% of the cases when the number of features representing one pose is less than 5. So, in order to achieve better results, a constraint in every window is imposed. If the window do not use more than 5 tracks, the result is discarded. For more precise results, this number can be increased. The failure of this method principally falls upon the lack of tracks. When the separation of the training images or the window length gets higher, the number of used tracks gets lower, and therefore the estimation of the pose fails.

In order to have a good estimation of the feature evolution, it is necessary to have a feature track comprising at least 3 different poses. The estimation fails at estimating the values of the feature out of the track endpoints. For example, let's consider the track of one feature in poses q_1, q_5 and q_9. The estimation of the value of the feature between the poses q_1 and q_9 will give us good results, but trying to estimate poses outside these boundaries, like for example in pose q_{11}, will result in a bad approximation of the real value. It is important to chose window length according to this.

Considering all the points explained, in the Figure 4.6 we can see the new results, taking the best 80%. Also a detailed table with the average of the error and the standard deviation in each experiment can be seen next:

Car sequence 1 (80%)		
Test name	**Average error (º)**	**Standard deviation (º)**
Seq1_1_1	0.79	0.63
Seq1_1_2	0.49	0.36
Seq1_1_3	0.64	0.49
Seq1_1_4	12.99	40.25
Seq1_1_5	27.45	51.75
Seq1_2_1	0.70	0.63
Seq1_2_2	1.08	0.65
Seq1_2_3	1.35	0.93
Seq1_2_4	121.31	53.84
Seq1_3_1	1.51	0.93
Seq1_3_2	1.87	1.27
Seq1_3_3	62.86	57.29
Seq1_4_1	1.82	1.42
Seq1_4_2	68.91	60.84
Seq1_5_1	2.38	1.50
Seq1_5_2	60.84	62.99
Seq1_6_1	1.79	1.15
Seq1_6_2	75.90	59.22
Seq1_7_1	46.48	55.94

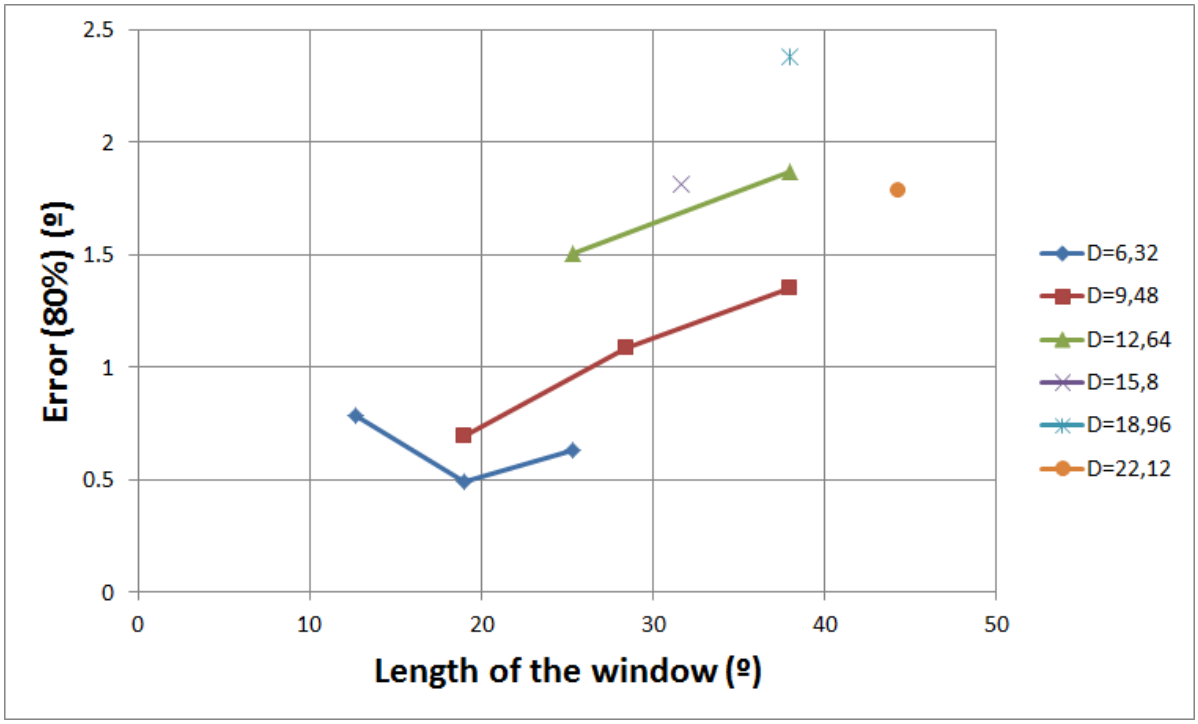


Figure 4.6 – *Graphic car dataset seq_1*

4.2.2 Car Dataset: Sequence 19

For this dataset, a value of $\sigma = 640$ is used. We can see the results in the Figure 4.7. A detailed table with the average of the error and the standard deviation in each experiment can be seen next:

Car sequence 19		
Test name	**Average error (º)**	**Standard deviation (º)**
Seq19_1_1	15.78	24.81
Seq19_1_2	7.74	19.23
Seq19_1_3	8.02	28.77
Seq19_1_4	17.04	42.28
Seq19_1_5	13.59	33.13
Seq19_1_6	3.65	5.40
Seq19_1_7	14.08	32.87
Seq19_2_1	12.61	23.49
Seq19_2_2	10.83	24.30
Seq19_2_3	14.91	33.88
Seq19_2_4	15.44	33.34
Seq19_2_5	15.10	33.66
Seq19_3_1	22.51	41.42
Seq19_3_2	21.68	46.75
Seq19_3_3	26.35	49.01
Seq19_4_1	17.38	25.72
Seq19_4_2	16.83	24.45

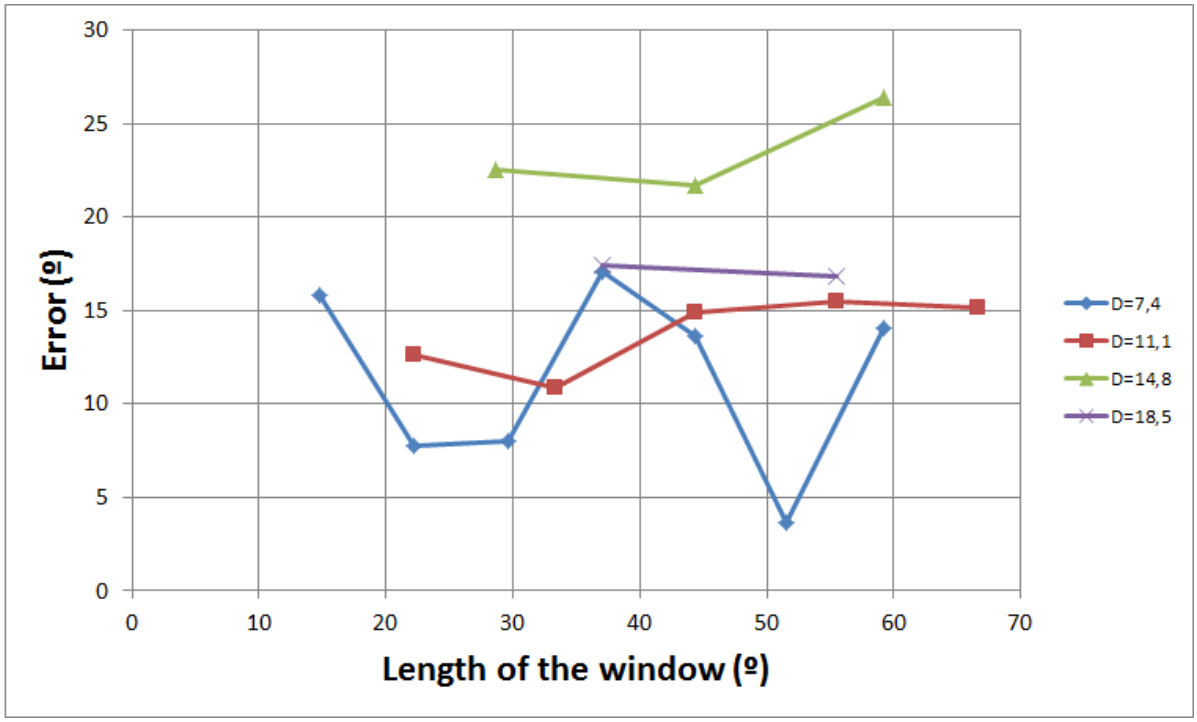


Figure 4.7 – *Graphic car dataset seq_19*

We will take now the best 80% of the results and see the improvement in Figure 4.8 and in the next table:

Car sequence 19 (80%)		
Test name	**Average error (º)**	**Standard deviation (º)**
Seq19_1_1	6.14	12.16
Seq19_1_2	0.69	0.45
Seq19_1_3	1.05	1.15
Seq19_1_4	1.24	1.21
Seq19_1_5	2.37	4.81
Seq19_1_6	1.63	1.73
Seq19_1_7	3.17	5.42
Seq19_2_1	2.60	5.36
Seq19_2_2	1.57	1.14
Seq19_2_3	1.64	1.26
Seq19_2_4	2.29	2.16
Seq19_2_5	2.36	2.76
Seq19_3_1	5.20	7.39
Seq19_3_2	3.09	4.15
Seq19_3_3	5.13	8.15
Seq19_4_1	6.38	7.22
Seq19_4_2	6.87	7.47

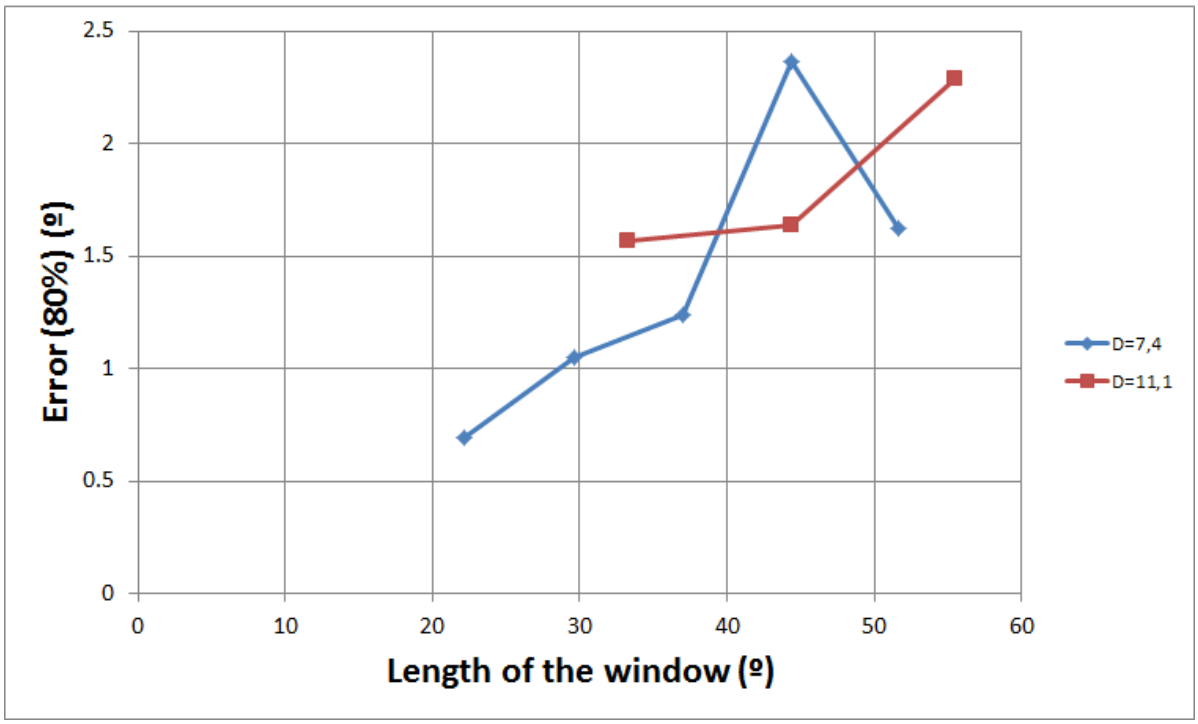


Figure 4.8 – *Graphic car dataset seq_19*

4.2.3 Objects

For the test objects, as we have more control over the dataset, with more uniform training images (no changing of illumination or background), we can see even better results. For all objects we collected images every 5 degrees for the frontal part as training images. The test images are taken every 3 degrees. This results in a total of 36 training images and 60 test images. The experiments include:

- Determination of best window length and distance between training images.
- Performance with change in illumination.
- Performance with occlusion and background noise
- Performance with change in scale

Using the leave-one-out technique, we measured the average error using different σ, and we chose the one that gave us the smallest error. We have a value of 640, 240 and 300 for the objects book, mouse folder and plain box respectively.

By keeping these values of σ fixed, and using again the leave-one-out technique, we compare the results using different window lengths (L) and different difference of rotation in degrees per consecutive image (D). The first step will be to compare different window lengths using the same difference in rotation. A plot of the results showing the average

error in degrees of the estimated pose can be seen in Figures 4.9, 4.10 and 4.11 for the objects book, mouse folder and plain box respectively. A detailed table with the average of the error and the standard deviation in each experiment can be seen next.

Book		
Test name	**Average error (°)**	**Standard deviation (°)**
Objects_1_1	1.25	1.23
Objects_1_2	1.30	1.35
Objects_1_3	2.09	3.58
Objects_1_4	16.04	34.26
Objects_2_1	1.17	1.29
Objects_2_2	1.78	2.23
Objects_2_3	21.49	39.97
Objects_3_1	1.77	1.92
Objects_3_2	10.00	24.43
Objects_3_3	58.46	47.32
Objects_4_1	2.55	2.42
Objects_4_2	19.77	41.21
Objects_4_3	37.90	37.01
Objects_5_1	4.24	3.61
Objects_5_2	18.15	34.79
Objects_5_3	50.09	35.02
Objects_6_1	28.69	53.13

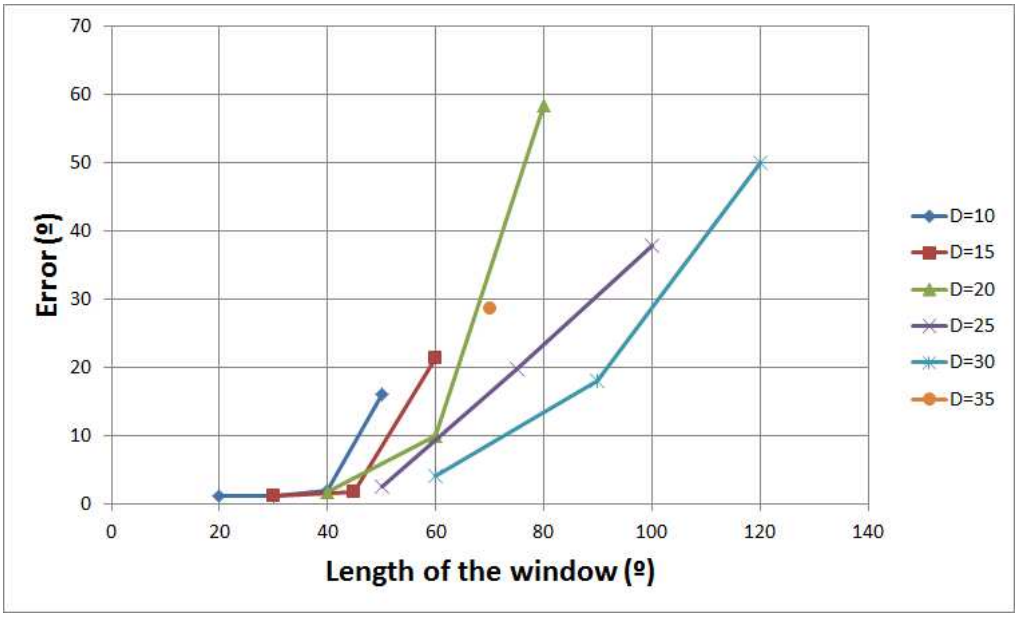


Figure 4.9 – *Graphic object book.*

Mouse Folder		
Test name	**Average error (º)**	**Standard deviation (º)**
Objects_1_1	1.78	2.68
Objects_1_2	2.56	4.08
Objects_1_3	5.55	21.96
Objects_1_4	15.70	37.25
Objects_1_5	14.74	30.87
Objects_2_1	1.23	1.01
Objects_2_2	9.29	22.30
Objects_2_3	15.26	36.14
Objects_2_4	19.79	39.05
Objects_3_1	2.35	2.37
Objects_3_2	4.82	6.32
Objects_3_3	21.60	40.79
Objects_4_1	3.50	2.57
Objects_4_2	11.14	27.45
Objects_4_3	18.60	39.09
Objects_5_1	7.14	17.74
Objects_5_2	12.48	27.98
Objects_5_3	55.33	37.46
Objects_6_1	20.13	32.87

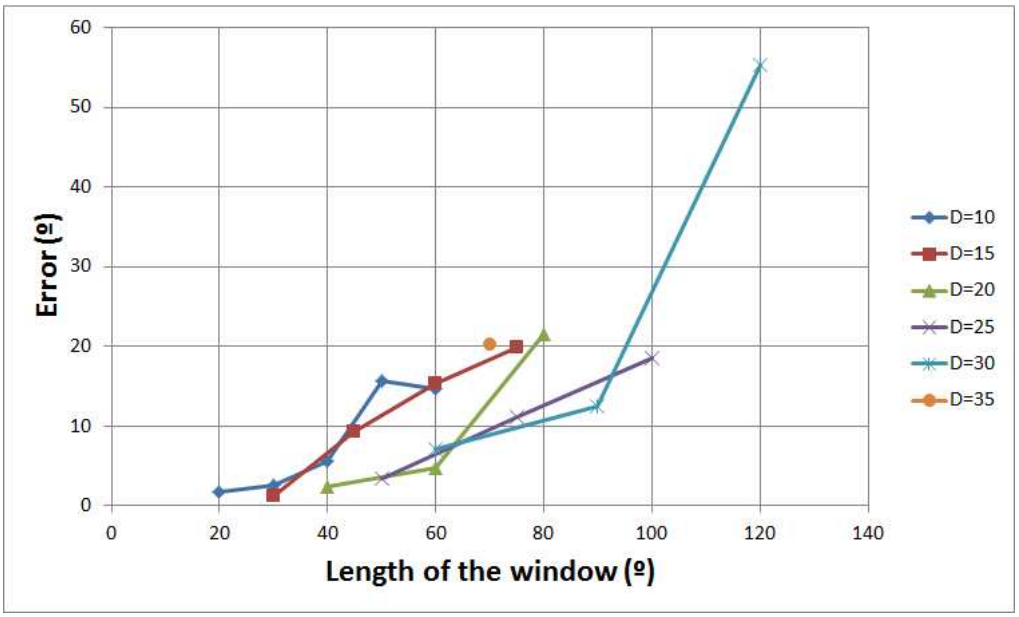


Figure 4.10 – *Graphic object mouse folder.*

Plane Box		
Test name	**Average error (º)**	**Standard deviation (º)**
Objects_1_1	1.03	1.20
Objects_1_2	0.98	1.15
Objects_1_3	1.11	1.45
Objects_1_4	23.15	33.48
Objects_2_1	1.22	1.09
Objects_2_2	9.46	29.35
Objects_2_3	12.36	36.21
Objects_2_4	44.67	56.84
Objects_3_1	1.99	1.63
Objects_3_2	13.10	36.10
Objects_3_3	17.07	42.56
Objects_4_1	22.76	30.29

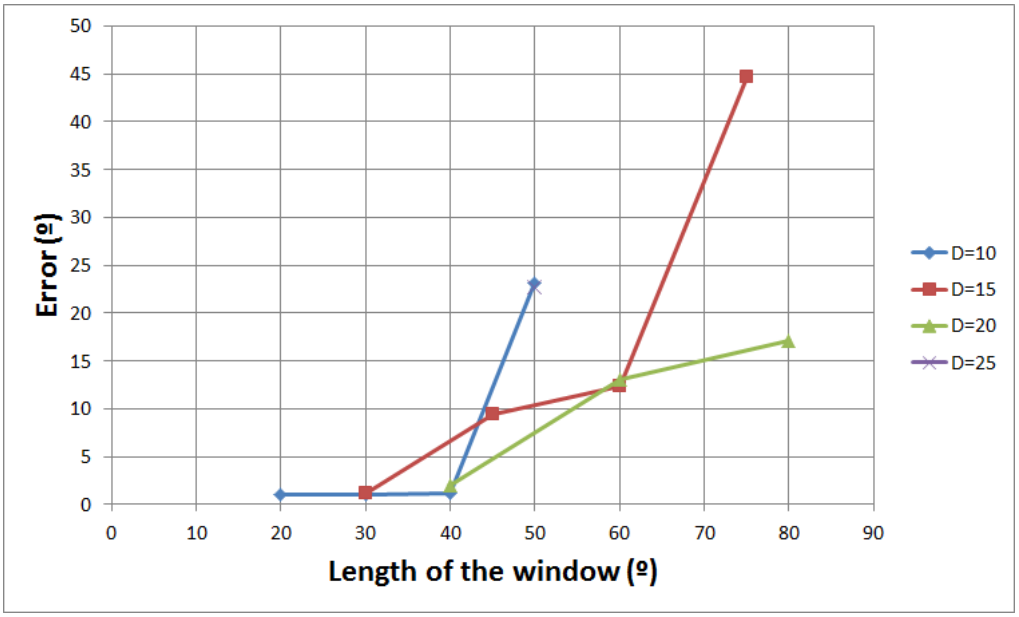


Figure 4.11 – *Graphic object plane box.*

We will take again the 80% percentile of the values (leaving most of the outliers). The results can be seen in the Figures 4.12, 4.13 and 4.14 for the objects book, mouse folder and plain box respectively. A detailed table with the average of the error and the standard deviation in each experiment can be seen next:

Book (80%)		
Test name	**Average error (º)**	**Standard deviation (º)**
Objects_1_1	0.79	0.53
Objects_1_2	0.78	0.67
Objects_1_3	0.83	0.63
Objects_1_4	4.84	6.31
Objects_2_1	0.71	0.47
Objects_2_2	1.03	0.70
Objects_2_3	3.26	5.42
Objects_3_1	1.10	0.72
Objects_3_2	1.58	1.66
Objects_3_3	43.15	36.00
Objects_4_1	1.67	1.15
Objects_4_2	2.85	2.87
Objects_4_3	24.04	23.82
Objects_5_1	2.88	1.69
Objects_5_2	3.49	3.48
Objects_5_3	38.67	27.52
Objects_6_1	5.51	6.29

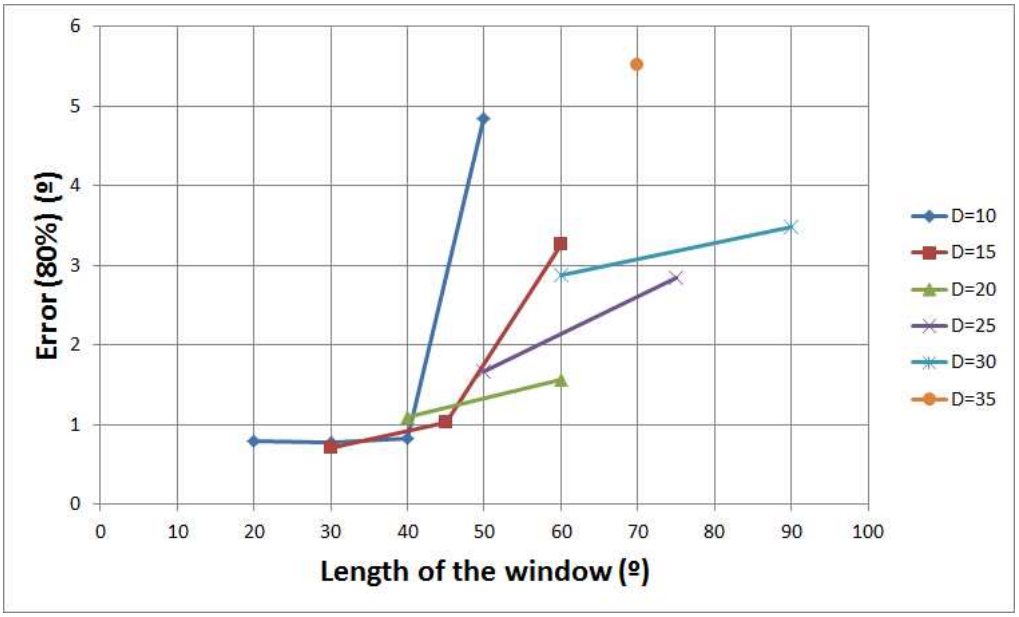


Figure 4.12 – *Graphic object book.*

Mouse Folder (80%)		
Test name	**Average error (º)**	**Standard deviation (º)**
Objects_1_1	0.71	0.68
Objects_1_2	0.90	0.89
Objects_1_3	1.11	1.11
Objects_1_4	2.93	3.67
Objects_1_5	4.81	6.66
Objects_2_1	0.85	0.44
Objects_2_2	2.41	2.47
Objects_2_3	2.45	3.06
Objects_2_4	5.21	7.02
Objects_3_1	1.46	0.90
Objects_3_2	2.12	1.67
Objects_3_3	5.26	6.61
Objects_4_1	2.58	1.55
Objects_4_2	2.92	2.25
Objects_4_3	3.20	2.20
Objects_5_1	3.03	2.01
Objects_5_2	3.13	2.26
Objects_5_3	43.62	30.20
Objects_6_1	7.41	6.66

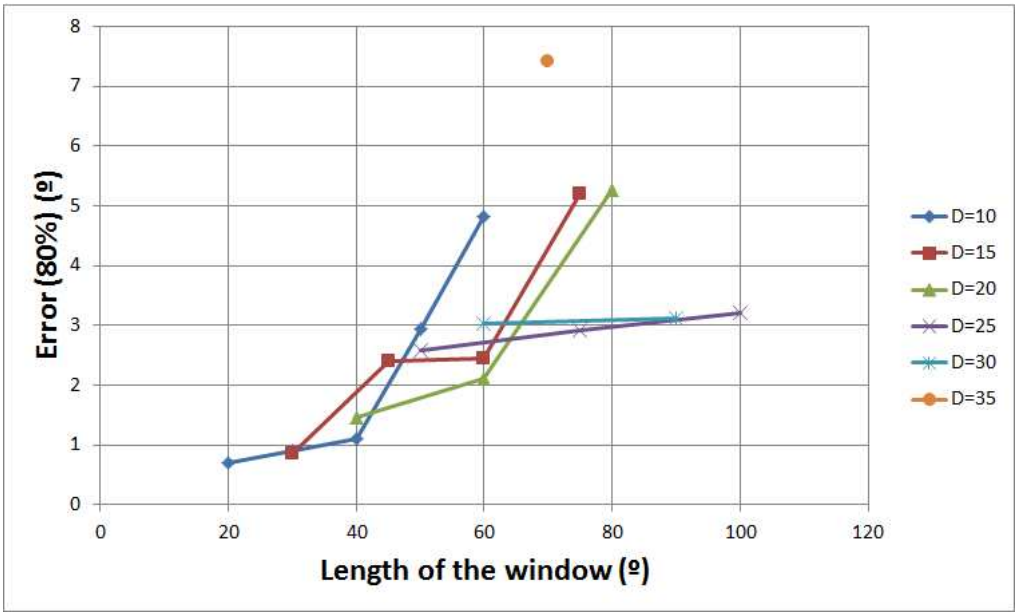


Figure 4.13 – *Graphic object mouse folder.*

Plane Box (80%)		
Test name	**Average error (º)**	**Standard deviation (º)**
Objects_1_1	0.57	0.42
Objects_1_2	0.55	0.40
Objects_1_3	0.65	0.52
Objects_1_4	10.17	19.51
Objects_2_1	0.83	0.56
Objects_2_2	1.01	0.91
Objects_2_3	1.11	0.94
Objects_2_4	21.47	29.61
Objects_3_1	1.42	0.94
Objects_3_2	2.05	1.32
Objects_3_3	2.95	3.32
Objects_4_1	10.88	17.30

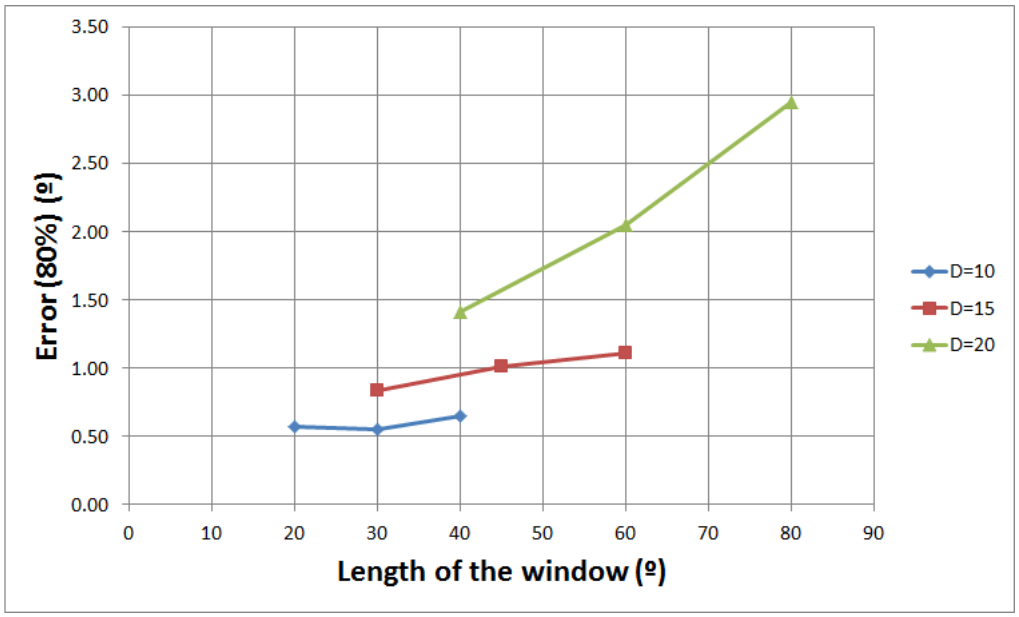


Figure 4.14 – *Graphic object plane box.*

The next table shows the results of tests changing illumination, scale and occlusion and background noise with the higher D that give us good results (20º) taking the best 80%:

Test name	**Average error (°)**	**Standard deviation (°)**
Change of illumination	2.54	2.45
Change in scale	7.03	10.25
Occlusion and background noise	7.75	26.21
Change of illumination (80%)	1.65	1.22
Change in scale (80%)	4.05	2.47
Occlusion and background noise (80%)	1.52	1.05

The only problem with occlusion comes when it is too severe in the test images and we have not available sufficient tracks to establish a relation with the training images. This also can happen with changing of scale. Background noise and changes of illumination has no severe effect in the estimation of the pose.

5 Conclusions and Future Research

For every kind of application some methods can be more suitable than others. In this thesis, we have tried to find a suitable method to solve typical applications of pose recognition for cars, face, or facades. We have looked for efficient algorithms that allow us to solve the problem without the need of reconstructing a 3D model of the object and therefore, with a much lower computational load. The method mimics the first steps of a 3D reconstruction, where we need to take pictures of the object at different orientations, but, instead of building a computationally complex 3D model of the object, we use the information extracted in the feature descriptors of each image to estimate the feature appearance at unknown poses. We can take advantage of the fact that descriptors change their values when a change in the orientation of the object occurs, and predict the values at orientations for which the ground truth information is not available.

The method is separated in two parts, the Off-line and the On-line Stage. In the Off-line Stage, we take pictures in a few known poses of the object to recognize, and we establish a track for each feature along the available images. For each feature track, we build a regression function that will estimate the value of the feature at unavailable poses. In the On-line Stage, a test image is input to the system. We extract its features and compare them with the features of the available training poses to establish correspondences. Once this matching is done, and following the principles on which SIFT features are matched, we compute the Euclidean distance between each feature in the track and the test image to find the most similar one. In order to achieve a more accurate result, we estimate the value of the feature at the poses that are not available by applying the regression function at those orientations. The pose estimation is conceived as an optimization problem as we have to minimize the error function given by the distance between the estimated descriptor and the current one. As the error function presents various local minima (the error function is not perfectly concave), we divide it into windows and then choose the global minimum among them, retrieving in this way the correct pose of the test image. The other main reason to divide the domain in sub-intervals is to maximize the number of tracks used. By embedding the minimization inside a Bayesian framework, we can estimate the probability of the actual pose given the feature descriptors of the test image.

As we had seen in the results section, it is possible to choose among different numbers of separation between the training images and window lengths to have the best results. With the experiments done, using a separation between images lower than 15 degrees, the method will success to estimate the pose in the 80% of the cases with an average

error of 1.3 degrees. Depending on our goal, it is possible to adjust the separation of the training images and the length of the window to have improved results in each case. This will affect the quantity of data used, and the speed of the computation. Another important parameter that affects the speed is the number of tracks available. For objects with a big quantity of features, it is possible to limit the tracks used, for example using only the best ones identified using the method explained with Formula 3.6. This will improve the speed of the paradigm, as the process to identify each track will be faster and there will be fewer comparisons to do.

A number of improvements can still be applied to the method. It is possible to look for a better optimization algorithm to achieve better results and avoid local minima in a more efficient way. A lot of research is done in this area and it exists a very large number of methods. Another improvement can be applied to the regression function. We have tried to look for speed and accuracy using the linear estimation, achieving good results, but if the number of training images available is not enough, or the speed in the on-line part is not a critical point, it is possible to use a more complex estimation to calculate the unavailable values, such as spline estimation. This could also improve the estimation outside the boundaries of the window.

Future research include a deep study in the evolution of the 128 values of one SIFT feature descriptor. We have noticed that some components are invariant to pose changes (usually the lowest or highest ones), and therefore this data does not have an important weigh in tge pose estimation and thus, it can be excluded. The rotation of the object is another area that can be expanded. For now, only a rotation in one dimension is considered (as we wanted to implement the method in some specific applications), but a study about the changes in the SIFT features when a rotation in multiple dimensions occurs can lead to some interesting results.

Bibliography

[1] Livingstone, M.: Vision and Art: The Biology of Seeing.
Abrams, New York, (2008).

[2] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz and Richard Szeliski: Building Rome in a day.
ICCV, (2009).

[3] Youngmin Park and Vincent Lepetit and Woontack Woo: Multiple 3D Object tracking for augmented reality.
ISMAR, (2008).

[4] Irschara, Arnold and Zach, Christopher and Frahm, Jan-Michael and Bischof, Horst: From Structure-from-Motion Point Clouds to Fast Location Recognition.
CVPR, (2009).

[5] Edward Hsiao and Alvaro Collet Romea and Martial Hebert: Making Specific Features Less Discriminative to Improve Point-based 3D Object Recognition.
CVPR, (2010).

[6] Thomas Serre, Maximillian Riesenhuber, Jennifer Louie, Tomaso Poggio: On the Role of Object-Specific features for Real World Object Recognition in Biological Vision.
Artificial Intelligence Lab, and Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Center for Biological and Computational Learning, Mc Govern Institute for Brain Research, Cambridge, MA, USA, (2002).

[7] Andrew Kae, Gary Huang, Carl Doersch, and Erik Learned-Miller: Improving State-of-the-Art OCR through High-Precision Document-Specific Modeling.
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2010).

[8] Nister, David and Stewenius, Henrik: Scalable Recognition with a Vocabulary Tree.
Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2, (2006).

[9] Ho Gi Jung, Dong Suk Kim, Pal Joo Yoon, Jaihie Kim: Structure Analysis Based Parking Slot Marking Recognition for Semi-automatic Parking System.
Structural, Syntactic, and Statistical Pattern Recognition, Springer Berlin / Heidelberg, (2006).

[10] David G. Lowe: Object Recognition from Local Scale-Invariant Features.
Vancouver, B.C., Canada: Department of Computer Science, University of British Columbia, (1999).

[11] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool: SURF: Speeded Up Robust Features.
Computer Vision and Image Understanding (CVIU), (2008).

[12] Krystian Mikolajczyk and Cordelia Schmid: A performance evaluation of local descriptors.
IEEE Transactions on Pattern Analysis and Machine Intelligence, (2005).

[13] Navneet Dalal and Bill Triggs: Histograms of Oriented Gradients for Human Detection.
CVPR, (2005).

[14] Iryna Gordon and David Lowe: What and Where: 3D Object Recognition with Accurate Pose.
Toward Category-Level Object Recognition, (2006).

[15] Fred Rothganger and Svetlana Lazebnik and Cordelia Schmid and Jean Ponce: 3D Object Modeling and Recognition Using Local Affine-invariant Image Descriptors and Multi-view Spatial constraints.
IJCV, (2006).

[16] S. Ullman: The interpretation of structure from motion.
A.I. Memo 476, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, (1976).

[17] Shi, J. and Tomasi, C.: Good features to track.
In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94), pp.593-600, Seattle, (1994).

[18] Triggs, B.: Detecting keypoints with stable position, orientation, and scale under illumination changes.

In Eighth European Conference on Computer Vision (ECCV 2004), pp. 100-113, Prague, (2004).

[19] Brown, M. and Lowe, D.: Automatic panoramic image stitching using invariant features.
International Journal of Computer Vision, 74(1):59-73, (2007).

[20] Jeffrey S. Beis, David G. Lowe: Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces.
Vancouver, B.C., Canada: Department of Computer Science, University of British Columbia, (1997).

[21] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden: Pyramid methods in image processing.
RCA Engineer, 29-6, (1984).

[22] G. P. Stein and A. Shashua: Model-based brightness constraints: On direct estimation of structure and motion.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(9):992-1015, (2000).

[23] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani: Algorithms.
Berkeley and U.C. San Diego, (2006).

[24] J. Shekel: Test functions for multimodal search techniques.
In Proceedings of the Fifth Annual Princeton Conference on Information Science and Systems, pages 354-359. Princeton University Press, Princeton, NJ, USA., (1971).

[25] Ioan Cristian Trelea: The particle swarm optimization algorithm: convergence analysis and parameter selection.
Information Processing Letters, (2003).

[26] Paola Festa and Mauricio G.C. Resende: An annotated bibliography of grasp.
AT&T Labs Research Technical Report TD-5WYSEW, AT&T Labs, (2004).

[27] Pete Bettinger and Jianping Zhu: A new heuristic method for solving spatially constrained forest planning problems based on mitigation of infeasibilities radiating outward from a forced choice.
Silva Fennica, 40(2):315-333. ISSN: 0037-5330, (2006).

[28] Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.
Oxford Univ. Press, (1996).

[29] S. Haykin: Neural Networks.
New York, NY: MacMillan College, (1994).

[30] Fei-Fei, L., Fergus, R., and Torralba, A.: Short course on recognizing and learning object categories.
In Twelfth International Conference on Computer Vision (ICCV 2009), Kyoto, Japan, (2009).