Chapter 3

Tracking algorithms for active systems

3.1 Introduction

Vision to the humanoid is essential, it is the predominant sensor that will allow the robot to interact with the real world, without its sensor the robot would only be able to react to commands entered by the user. With the addition of a vision system the robot can be made to act autonomously, it can actively react to its surroundings making decisions based on the objects it encounters. Although a humanoid biped robot requires many sensors to allow accurate actuation of its limbs, without a vision sensor the system can never truly interact with its surrounds.

One of the main tasks of a robot for interact with their environment is the identify objects of interest and to follow their evolution along the time, that is known as visual tracking. Visual tracking can be described as the process of determining the location of a feature in an image sequence over time. Examples include tracking cars in an intersection via a traffic camera, or tracking the head of a computer user with a web-cam. Another possible application is tracking multiple small features of interest, such as corners of an object, in attempt to determine its 3-dimensional geometry.

The target to be tracked might be a complete object (e. g. a person or a ball) or small area on an object (e. g. a corner). In either case, the feature of interest is typically contained within a target region, where the position is described in X-Y coordinates in pixel units on the image.

Ideally, a tracking algorithm would be able to locate the object anywhere within the image at any point in time. However typically only a limited region of the image is searched (usually the same size as the target region). Reasons for this are the efficiency (especially necessary for real-time applications) and the fact that there might be many other similar-looking objects in the image.

The intuitive approach is to search within a region centred around the last

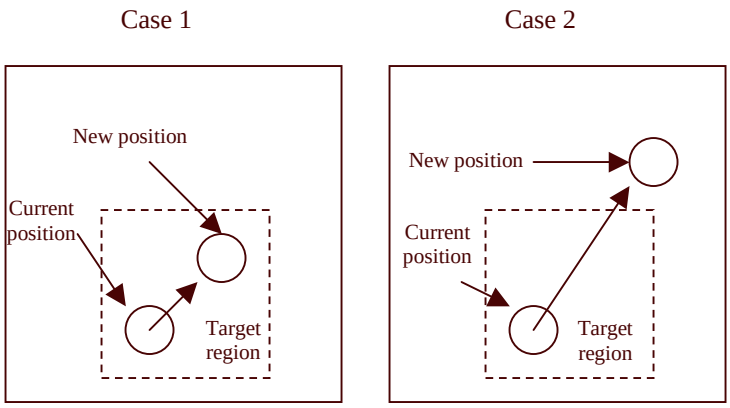


Figure 3.1: *Case1:* Tracking the object, without position prediction might be successful, *Case 2:* Tracking without prediction will fail.

position of the object. But as figure 3.1 illustrates, this approach will fail it the object moves outside the target range. There are many possible reasons why the object might not stay within this region:

- The object is moving to fast
- The frame rate is too low.
- The searched region is too small.

These problems are related to each other and could be avoided by ensuring a high enough frame rate for example. But given other constraints, these problems are often unavoidable.

In addition, even when the target can be located, it seldomly appears the same in all images. The appearance of the same target is continuously affected by changes in orientation, lighting, occlusions, and imperfections in the camera. So essentially, the true location of the target is very difficult to observe accurately under the usual circumstances.

In summary, two major problems have been identified:

1. The object can only be tracked if it does not move beyond the searched region.
2. Various factors such as lighting and occlusions can affect the appearance of the target, thus making accurate tracking difficult.

To solve the first problem, we can attempt, predicting the location of the target, and searching in the region centred around that location. But in making the

prediction, it is necessary to consider the second problem as well. The previously obtained location measurements are likely not accurate, so the prediction method needs to be robust enough to handle this source of error.

This thesis shows the design of a vision system for a humanoid robot so as to allow the robot to view a soccer ball and to follow it over time.

Computer vision systems have been traditionally designed disregarding the observer role (camera motion, stereo rig geometry, lens parameters, etc.) in the perceptual process. Actually, most systems were designed to process images which had been prerecorded in some sense or, at least, acquired independently of the perception process. However, most biological vision systems do not just see the surrounding space but they actively look at it and very often, their visual processing is related to a specific task or set of task [1].

3.2 Tracking in computer vision

Estimation and tracking of motion in image sequences is a well-established branch of computer vision. Real world scenes with large, rigid or deformable moving objects are usually considered. Two main classes of motion estimation methods are traditionally distinguished: the optical flow based and the local feature based techniques. Powerful algorithms are available in both classes. Applying them to particle flow image sequences may lead to essential improvement in visualisation and measurement results.

For this thesis was proven as tracking algorithms the based on optical flow as well as those based on features. The algorithms based on optical flow showed to be expensive to be implemented in real time tracking, for what was chosen an algorithm based in features. The algorithm must be able to track in real time yet not absorb a major share of computational resources: other tasks must be able to run while the visual tracking is being used. These algorithms are the CAMSHIFT, Kalman filter and Particle filter for this reason they will be explained in detail in this thesis.

3.2.1 Optical flow techniques

The optical flow based algorithms extract a dense velocity field from an image sequence assuming that image intensity is conserved during the displacement. This conservation law is expressed by a spatiotemporal differential equation which is solved under additional constraints of different form. A recent survey of the optical flow techniques and a comparative empirical evaluation study of their performance can be found in [130].

Recently, Quénot [132] presented an optical flow algorithm based on a dynamic programming technique. (A source code is available at the ftp site [131].) Dynamic programming was originally used for searching the optimal matching between two one-dimensional patterns. In [132], it is extended to two dimensions: the global matching is searched that minimizes the distance between two images. This is achieved by the Orthogonal Dynamic Programming (ODP)

Figure 3.2: Optical flow image

algorithm that slices the two images into properly selected sets of parallel overlapping strips. The corresponding strips are then matched as one-dimensional signals. Two passes are done for two orthogonal slicing directions. This process is iterated in a multiresolution way so as to refine the previously obtained matching. The number of iterations is a parameter of the algorithm and depends on the complexity of the velocity field. A few iterations are sufficient for relatively simple flows.

The algorithm [132] yields a dense velocity field between any two images of a sequence, provided that the displacements between the two frames are not too large. The method computes a velocity value in each pixel of the image, even though it may be less accurate near the image borders.

Both correlation methods and optical flow techniques imply matching performed for the whole image, which can be very time consuming for large images. When individual particles can be detected, and when there is no need to measure a velocity vector in each pixel of the image, feature based techniques are also applicable and may provide a faster solution.

3.2.2 Feature based techniques

The feature based techniques extract local regions of interest (features) from the images and identify the corresponding features in each image of the sequence [133,134]. Such algorithms have initially been developed for tracking a small number of salient features in long image sequences. The tracking process can be divided into two major subtasks: feature extraction and feature tracking. One can extract features only in the first image and search for the same features in the subsequent images. This dynamic feature extraction scheme is advocated by the KLT Tracker [134]. Alternatively, one can find features in each static image prior to tracking and then find the corresponding features along the sequence. Most of the feature tracking algorithms operate in this way such as Camshift, although the dynamic feature extraction seems to be more natural and reliable than the static one: the temporal information helps decide where and how to search for features in the next frame.

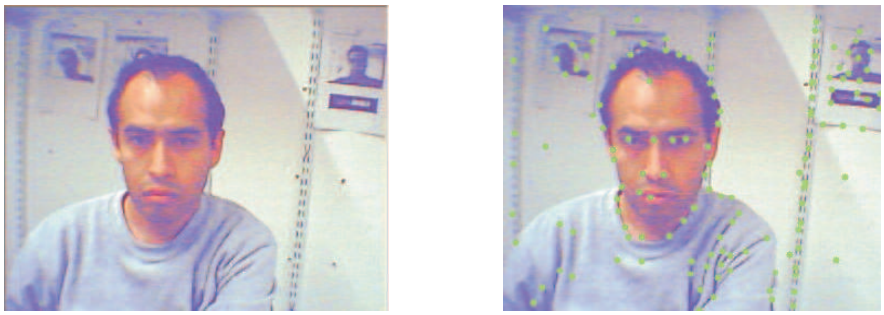


Figure 3.3: Illustrates feature selection by the KLT.

3.2.3 The KLT Tracker

In [4], Shi and Tomasi present the KLT (Kanade-Lucas-Tomasi) Tracker, an algorithm that selects features which are optimal for tracking, and keeps track of these features. The basic principle of the KLT is that a good feature is one that can be tracked well, so tracking should not be separated from feature extraction. If a feature is lost in a subsequent frame, the user can optionally ask the procedure to find another one to keep the number of features constant.

A good feature is a textured patch with high intensity variation in both x and y directions, such as a corner. Denote the intensity function by g(x,y) and consider the local intensity variation matrix

$$Z = \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix} \tag{3.1}$$

A patch defined by a *n* x *n* window is accepted as a candidate feature if in the center of the window both eigenvalues of *Z*, λ_1 and λ_2, exceed a predefined threshold λ: $\min(\lambda_1, \lambda_2) > \lambda$. A separate parameter sets the minimum distance between (centers of) features.

The (maximum) number of features to be tracked, N_f, is specified by the user. In the first frame of a sequence, the candidate features are ranked according to their strength defined by $\min(\lambda_1, \lambda_2) > \lambda$, then N_f strongest features are selected if available. (If not, all candidates are used.)

The KLT defines a measure of dissimilarity that quantifies the change of appearance of a feature between the first and the current image frame, allowing for affine image changes. At the same time, a pure translation model of motion is used to track the selected best features through the sequence. For reliable and fast processing, the maximum interframe displacement is limited, although it can be quite large compared to that of the conventional optical flow approaches.

3.3 Camshift

In order, therefore, to find a fast, simple algorithm for basic tracking, we have focused on a local feature based technique called color-based tracking [141]-[145], yet even these simpler algorithms are too computationally complex (and therefore slower at any given CPU speed) due to their use of color correlation, blob and region growing, Kalman filter smoothing and prediction, and contour considerations. The complexity of the these algorithms derives from their attempts to deal with irregular object motion due to perspective (near objects to the camera seem to move faster than distal objects); image noise; distractors, such as other objectes in the scene; object occlusion by other objects; and lighting variations. We want a fast, computationally efficient algorithm that handles these problems in the course of its operation, i.e., an algorithm that mitigates the above problems for free. The Continuously Adaptive Mean SHIFT (CAMSHIFT) algorithm [146], is based on the mean shift algorithm [140], a robust non-parametric iterative technique for finding the mode of probability distributions.

The mean shift algorithm operates on probability distributions. To track colored objects in video frame sequences, the color image data has to be represented as a probability distribution [140]; normally it is used color histograms to accomplish this. Color distributions derived from video image sequences change over time, so the mean shift algorithm has to be modified to adapt dynamically to the probability distribution it is tracking. The new algorithm that meets all these requirements is called CAMSHIFT. For object tracking, CAMSHIFT tracks the x, y, and Area of the flesh color probability distribution representing the object. Area is proportional to Z, the distance from the camera.

Figure 3.3 summarizes the algorithm description, taking as example the face tracking. For each video frame, the raw image is converted to a color probability distribution image via a histogram model of the color being tracked. The center and size of the color object are found via the CAMSHIFT algorithm operating on the color probability image (the gray box is the mean shift algorithm). The current size and location of the tracked object are reported and used to set the size and location of the search window in the next video image. The process is then repeated for continuous tracking.

In order to use CAMSHIFT to track colored objects in a video scene, a probability distribution image of the desired color in the video scene must be created. In order to do this, we first create a model of the desired hue using a color histogram. Normally is used the Hue Saturation Value (HSV) color system [147][148] that corresponds to projecting standard Red, Green, Blue (RGB) color space along its principle diagonal from white to black (see arrow in Figure 3.4).

This results in the hexcone in Figure 3.5. Descending the V axis in Figure 3.5 gives us smaller hexcones corresponding to smaller (darker) RGB subcubes in Figure 3.4. HSV space separates out hue (color) from saturation (how concentrated the color is) and from brightness. Always it is created the color models

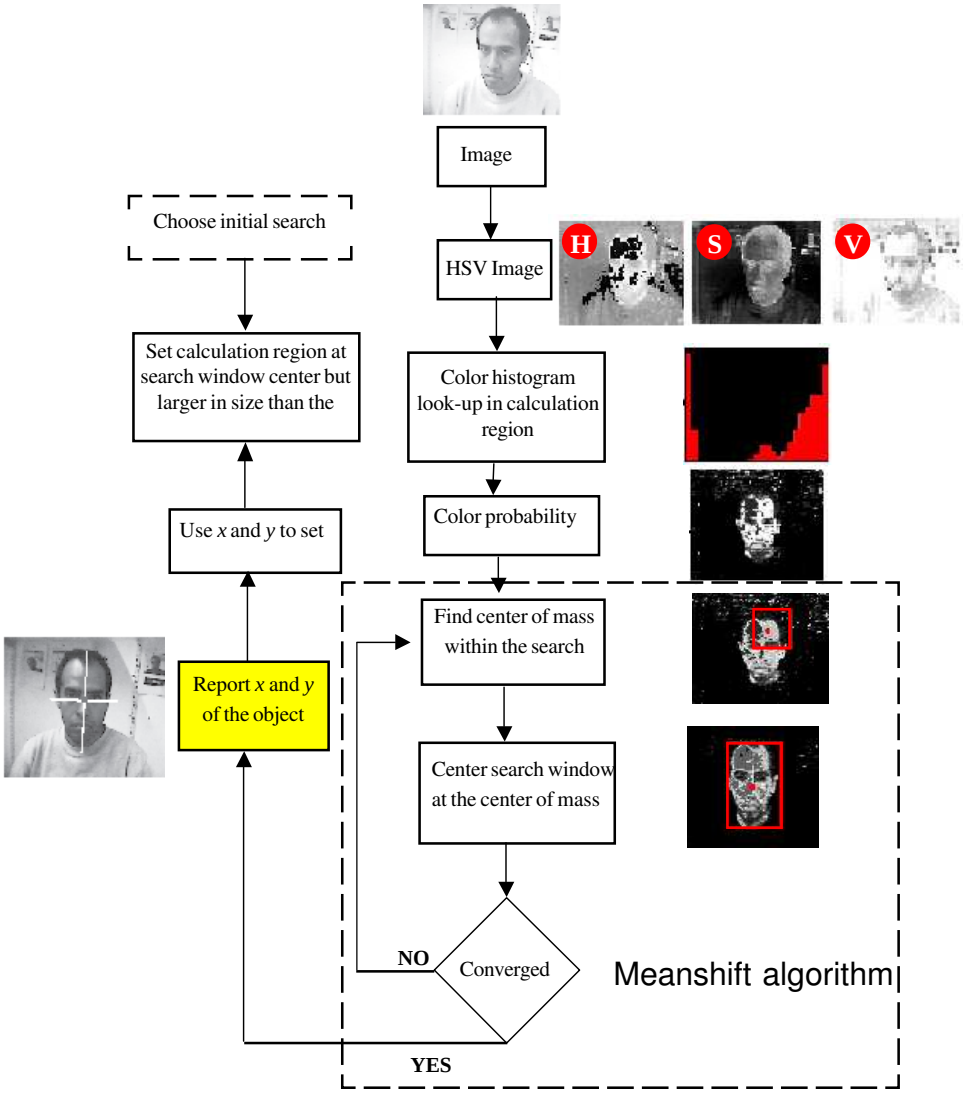


Figure 3.4: Block diagram of color object tracking

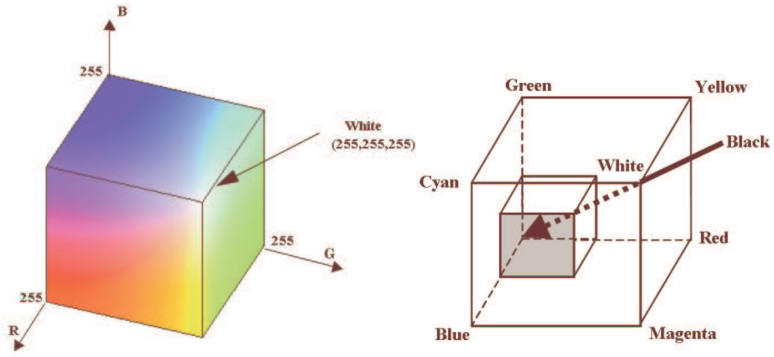


Figure 3.5: RGB color model.

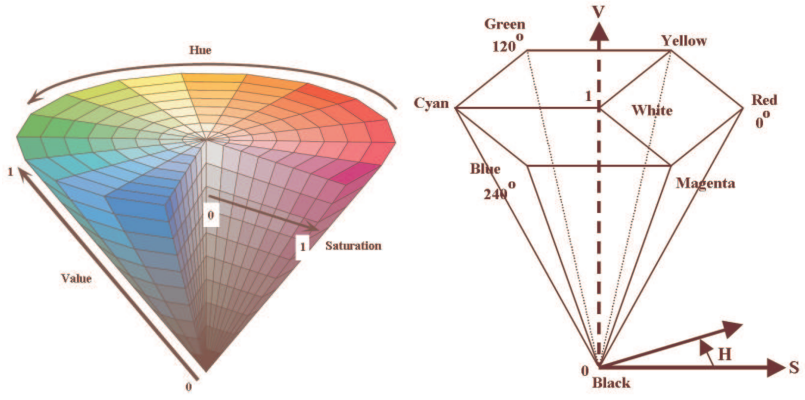


Figure 3.6: HSV color model.

by taking 1D histograms from the H (hue) channel in HSV space.

3.4 CAMSHIFT Derivation

The closest existing algorithm to CAMSHIFT is known as the mean shift algorithm [149][150]. The mean shift algorithm is a non-parametric technique that climbs the gradient of a probability distribution to find the nearest dominant mode (peak).

To calculate the mean shift algorithm, the following steps are carried out:

1. Choose a search window size.
2. Choose the initial location of the search window.
3. Compute the mean location in the search window.
4. Center the search window at the mean location computed in Step 3.
5. Repeat Steps 3 and 4 until convergence (or until the mean location moves less than a preset threshold).

3.4.1 Proof of Convergence

The Kernel estimation of density is the most popular method:

Given n points that are data $\mathbf{x}_i$, i=1,..., n in a d-dimensional space R^d, the estimator of density based on kernel with kernel $K(\mathbf{x})$ and matrix $\mathbf{H}$ that is defined positive of dimension d x d is given for:

$$\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}_i) \tag{3.2}$$

where

$$K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-\frac{1}{2}}\, K(\mathbf{H}^{-\frac{1}{2}}\mathbf{x}) \tag{3.3}$$

The d-variate kernel $K(\mathbf{x})$ is a bounded function with compact support satisfying:

$$\int_{\mathrm{R}^d} K(\mathbf{x})d\mathbf{x} = 1 \tag{3.4}$$

$$\int_{\mathrm{R}^d} \mathbf{x}K(\mathbf{x})d\mathbf{x} = 0 \tag{3.5}$$

$$\lim_{|\mathbf{x}|\to\infty} |\mathbf{x}|^d\, K(\mathbf{x}) = 0 \tag{3.6}$$

$$\int_{\mathrm{R}^d} \mathbf{x}\mathbf{x}^{\mathrm{T}} K(\mathbf{x}) d\mathbf{x} = c_K \mathbf{I} \tag{3.7}$$

where c_K is a constant and $\mathbf{I}$ the identity matrix. The previous conditions can be seen as: **(3.4)** the integral in all domain is one, **(3.5)** the mean or expected value is 0, **(3.6)** has compact support, and **(3.7)** it converges. The multivariate kernel can be generated in two different ways:

$$K^P(\mathbf{x}) = \prod_{i=1}^{d} K_1(x_i) \tag{3.8}$$

$$K^S(\mathbf{x}) = a_{k,d} K_1(|\mathbf{x}|) \tag{3.9}$$

where the constant $a_{k,d}$ is defined as:

$$a_{k,d}^{-1} = \int_{\mathrm{R}^d} K_1(|\mathbf{x}|) d\mathbf{x} \tag{3.10}$$

this constant simply assures that the integral of $K^S(\mathbf{x})$ be 1, although this condition can be ignored in this context.

In this case we are interested in the radially symmetric kernels as it is the case of $K^S(\mathbf{x})$ and in a special of kernels that satisfy the following condition:

$$K(\mathbf{x}) = c_{k,d} k(|\mathbf{x}|^2) \tag{3.11}$$

Only necessary to define *k*(x) (that is called the kernel profile) for positive values. The normalization constant $c_{k,d}$ that makes that *K*(**x**) integrates to one, is assumed strictly positive.

Using a matrix $\mathbf{H}$ fully parametrized increases the complexity of the estimate, in the practice is used $\mathbf{H}$ like a diagonal matrix with only one parameter *h*, that is to say $\mathbf{H} = h^2\mathbf{I}$, with this we reduce the complexity of the analysis.

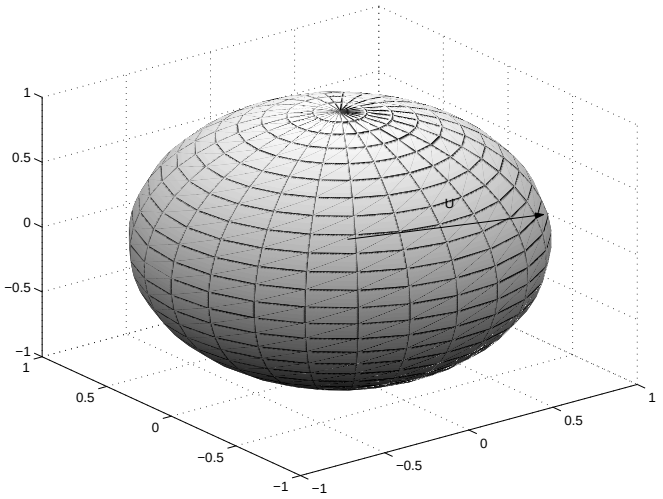
But to be able to use this approach it is necessary to assure that we have a Euclidian metric for the feature space. This means that the form in which the data are distributed is considered as threshold, the distance from a point to the data mean in the dimensional space. Of this way **x** belongs to the grouping m, if *dE*<=*U*. Considering that

$$dE(\mathbf{x}, \mu_m) = |\mathbf{x} - \mu_m|^2 = (\mathbf{x} - \mu_m)^{\mathrm{T}} (\mathbf{x} - \mu_m) \tag{3.12}$$

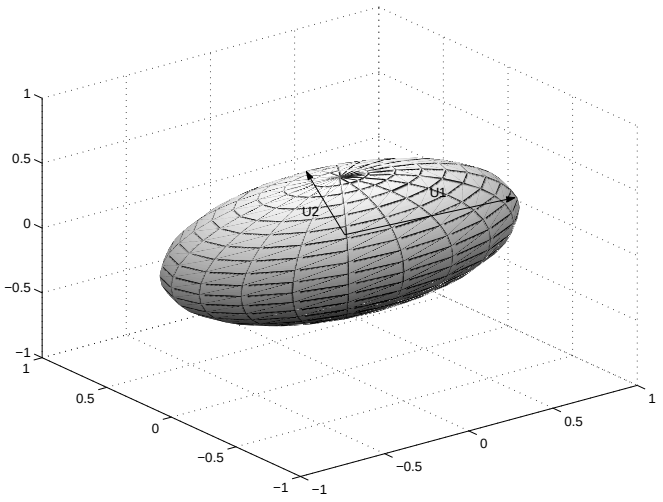
The figure 3.6 shows a Ecludian distribution (a) with a threshold U, while the other distribution (b) it is not Ecludian, where the thresholds U1 and U2 are different.

If we take only *h* as parameter, the kernel density estimator (KDE) has the form:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{\mathbf{x} - \mathbf{x}_i}{h}) \tag{3.13}$$



(a)



(b)

Figure 3.7: (a) Ecludian distribution with a threshold U, (b) not Ecludian distribution.

In the practice can be proven (by means of the calculation of AMISE - asymptotic mean integrated square error- between the density and their estimate) that this measure is minimized if we use the kernel of Epanechnikov

$$K_E(\mathbf{x}) = \begin{cases} \frac{1}{2}c_d^{-1}(d+2)(1-|\mathbf{x}|^2) & |\mathbf{x}| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

that has the profile:

$$k_E(x) = \begin{cases} 1-x & 0 \leq x \leq 1 \\ 0 & x > 1 \end{cases} \tag{3.15}$$

Where c_d is the volume of the *d*-dimensional sphere. Although we can also use the normal multivariate kernel

$$K_N(\mathbf{x}) = (2\pi)^{-\frac{d}{2}} \exp(-\frac{1}{2}|\mathbf{x}|^2) \tag{3.16}$$

that has the profile:

$$k_N(x) = \exp(-\frac{1}{2}x) \quad x \geq 0 \tag{3.17}$$

Using the profile notation, the density estimator can be written as

$$\hat{f}_{h,K}(\mathbf{x}) = \frac{c_{k,d}}{nh^d} \sum_{i=1}^{n} k\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right) \tag{3.18}$$

Now then, to make the analysis we need to locate the modes of this density, that is to say, we have to find the points where $\nabla f(\mathbf{x}) = 0$. Using the lineal property of (3.18) and the chain rule can be written:

$$\nabla \hat{f}_{h,K}(\mathbf{x}) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (\mathbf{x}-\mathbf{x}_i) k^{!}(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2) \tag{3.19}$$

Assuming that the derived of the profile k exists for all x we can define the profile:

$$g(x) = -k^{!}(x) \tag{3.20}$$

that originates the kernel

$$G(\mathbf{x}) = c_{g,d} g(|\mathbf{x}|^2) \tag{3.21}$$

Introducing *g*(*x*) in (3.19) we have that:

$$\nabla \hat{f}_{h,K}(\mathbf{x}) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (\mathbf{x}-\mathbf{x}_i) g(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2) \tag{3.22}$$

dividing and multiplying inside of (3.22) by

$$\sum_{i=1}^{n} g(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2) \tag{3.23}$$

we have:

$$\nabla \hat{f}_{h,K}(\mathbf{x}) = \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right)\right] \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right)}{\sum_{i=1}^{n} g\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right)} - \mathbf{x}\right] \tag{3.24}$$

It is assumed that (3.23) is a positive number, condition that satisfy the kernels usually used.

In the expression (3.24) we can see that the first term is proportional (only the constants change) to the density estimator in the point x using the kernel $G(\mathbf{x})$, which has the expression:

$$\hat{f}_{h,G}(\mathbf{x}) = \frac{c_{g,d}}{nh^d} \sum_{i=1}^{n} g\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right) \tag{3.25}$$

this term is the difference between the weighted mean (where the kernel $G(\mathbf{x})$ is using for the weights) and $\mathbf{x}$ that is the kernel center, from(3.24), (3.25) and (3.26) becomes:

$$\nabla \hat{f}_{h,K}(\mathbf{x}) = \hat{f}_{h,G}(\mathbf{x}) \frac{2c_{k,d}}{h^2 c_{g,d}} \mathbf{m}_{h,G}(\mathbf{x}) \tag{3.26}$$

yielding:

$$m_{h,G}(x) = \frac{1}{2} h^2 c \frac{\nabla \hat{f}_{h,K}(x)}{\hat{f}_{h,G}(x)} \tag{3.27}$$

As we can see, the vector Mean-Shift in the position $\mathbf{x}$ calculated with the kernel $G(\mathbf{x})$ is proportional to the gradient of the density estimator obtained with the kernel K. The normalization is carried out by the density estimator with the kernel $G(\mathbf{x})$ in the point $\mathbf{x}$ (the denominator of the expression). Then the direction of the vector Mean-Shift coincides with the maximum growth of the density. Therefore by means of the sequential calculation of this vector, we can find the stationary points of the estimated density, that is to say, the modes. Being then the procedure:

- Given an initial position, to calculate the vector Mean-Shift $m_{h,G}(\mathbf{x})$
- to displace the kernel $G(\mathbf{x})$ in the direction of $m_{h,G}(\mathbf{x})$.

The previously explained normalization is in fact which gives a great performance to the algorithm, because where there are few points, the vector has a big magnitude (because the normalization denominator is small), therefore the steps are large, which is correct because the areas of few points do not have interest for the feature space analysis. On the other hand when we are in an area of the space with high density the vector magnitude is small (because the normalization denominator is large) and the steps are small, to make an analysis but detailed.

3.5 Kalman filter for vision tracking.

The celebrated *Kalman filter*, rooted in the state-space formulation or linear dynamical systems, provides a recursive solution to the linear optimal filtering problem. It applies to stationary as well as nonstationary environments. The solution is recursive in that each updated estimate of the state is computed from the previous estimate and the new input data, so only the previous estimate requires storage. In addition to eliminating the need for storing the entire past observed data, the Kalman filter is computationally more efficient than computing the estimate directly from the entire past observed data at each step of the filtering process.

In this section, we present an introductory treatment of Kalman filters to pave the way for their application in vision tracking.

Consider a *linear, discrete-time dynamical system* described by the block diagram shown in Figure 3.8. The concept of *state* is fundamental to this description. The *state vector* or simply state, denoted by $\mathbf{x}_k$, is defined as the minimal set of data that is sufficient to uniquely describe the unforced dynamical behavior of the system; the subscript k denotes discrete time. In other words, the state is the least amount of data on the past behavior of the system that is needed to predict its future behavior. Typically, the state $\mathbf{x}_k$ is unknown. To estimate it, we use a set of observed data, denoted by the vector $\mathbf{y}_k$.

In mathematical terms, the block diagram of Figure 3.8 embodies the following pair of equations:

1. Process equation

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1,k}\mathbf{x}_k + \mathbf{w}_k \tag{3.28}$$

where $\mathbf{F}_{k+1,k}$ is the *transition matrix* taking the state $\mathbf{x}_k$ from time k to time $k + 1$. The process noise $\mathbf{w}_k$ is assumed to be additive, white, and Gaussian, with zero mean and with covariance matrix defined by

$$E\left[\mathbf{w}_n\mathbf{w}_k^T\right] = \begin{cases} \mathbf{Q}_k \text{ for } n{=}k \\ 0 \text{ for } n \neq k \end{cases} \tag{3.29}$$

where the superscript T denotes matrix transposition. The dimension of the state space is denoted by M.

2. Measurement equation

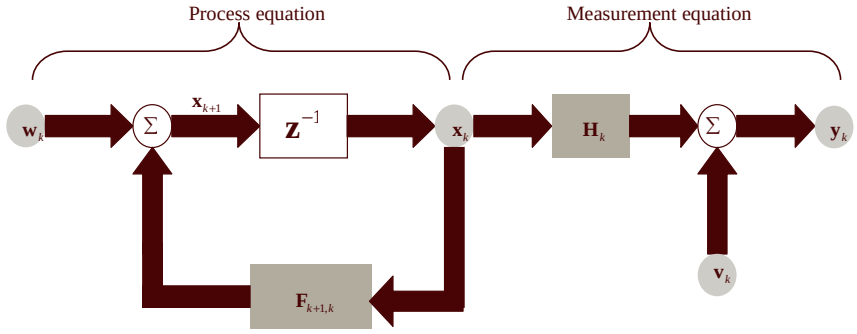


Figure 3.8: Signal-flow graph representation of a linear, distrete-time dynamical system.

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{3.30}$$

where $\mathbf{y}_k$ is the observable at time k and $\mathbf{H}_k$ is the *measurement matrix*. The measurement noise $\mathbf{v}_k$ is assumed to be addilive, white, and Gaussian, with zero mean and with covariance matrix defined by

$$E\left[\mathbf{v}_n \mathbf{v}_k^T\right] = \begin{cases} \mathbf{R}_k \text{ for } n{=}k \\ 0 \text{ for } n \neq k \end{cases} \tag{3.31}$$

Moreover, the measurement noise $\mathbf{v}_k$is uncorrelated with the process noise $\mathbf{w}_k$. The dimension of the measurement space is denoted by n.

The Kalman filtering problem, namely, the problem of jointly solving the process and measurement equations for the unknown state in an optimum manner may now be formally stated as follows:

Use the entire observed data, consisting of the vectors $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_k$, to find for each $k \geq 1$ the minimum mean-square error estimate of the state $\mathbf{x}_k$.

The problem is called filtering if $i = k$, prediction if $i > k$ and smoothing if $1 \leq i < k$.

3.6 Optimum estimates

Before proceeding to derive the Kalman filter, we find it useful to review some concepts basic to optimum estimation. To simplify matters, this review is presented in the context of scalar random variables; generalization of the theory to vector random variables is a straightforward matter. Suppose we are given the observable

$$y_k = x_k + v_k \tag{3.32}$$

where x_k is an unknown signal and v_k is an additive noise component. Let x_k denote the a posteriori estimate of the signal x_k, given the observations $y_1, y_2, ..., y_k$. In general, the estimate $\hat{x}_k$ is different from the unknown signal x_k. To derive this estimate in an optimum manner, we need a cost (loss) function for incorrect estimates. The cost function should satisfy two requirements:

The cost function is nonnegative.

The cost function is a nondecreasing function of the *estimation error* x_k defined by

$$\tilde{x}_k = x_k - \hat{x}_k \tag{3.33}$$

These two requirements are satisfied by the *mean-square error* defined by

$$J_k = E[(x_k - \hat{x}_k)^2]$$

$$J_k = E[(\tilde{x}_k)^2] \tag{3.34}$$

where *E* is the expectation operator. The dependence of the cost function J_k on time *k* emphasizes the nonstationary nature of the recursive estimation process.

To derive an optimal value for the estimate $\hat{x}_k$ we may invoke two theorems taken from stochastic process theory [153, 156]:

Theorem 3.1 Conditional mean estimator If the stochastic processes $\{x_k\}$ and $\{y_k\}$ are jointly Gaussian, then the optimum estimate $\hat{x}_k$ that minimizes the mean-square error $J_k$is the conditional mean estimator:

$$\hat{x}_k = E[x_k \,|y_1 \,, y_2, ..., y_k] \tag{3.35}$$

Theorem 3.2 Principle of orthogonality Let the stochastic processes $\{x_k\}$ and $\{y_k\}$ be of zero means; that is,

$$E[x_k] = E[y_k] = 0 \text{ for all } k \tag{3.36}$$

Then:

(i) the stochastic process $\{x_k\}$ and $\{y_k\}$ are jointly Gaussian; or

(ii) if the optimal estimate $\hat{x}_k$ is restricted to be a linear function of the observables and the cost function is the mean-square error,

(iii) then the optimum estimate $\hat{x}_k$ given the observables $y_1, y_2, ..., y_k$ is the orthogonal projection of x_k on the space spanned by these observables.

3.7 Kalman filter

Suppose that a measurement on a linear dynamical system, described by Eqs. (3.28) and (3.30), has been made at time *k*. The requirement is to use the information contained in the new measurement $\mathbf{y}_k$ to update the estimate of the unknown state $\mathbf{x}_k$. Let $\hat{\mathbf{x}}_k^-$ denote a priori estimate of the state, which is already available at time *k*. With a linear estimator as the objective, we may express the a posteriori estimate $\hat{\mathbf{x}}_k$ as a linear combination of the a priori estimate and the new measurement, as shown by

$$\hat{\mathbf{x}}_k = \mathbf{G}_k^{(1)}\hat{\mathbf{x}}_k^- + \mathbf{G}_k\mathbf{y}_k \tag{3.37}$$

where the multiplying matrix factors $\mathbf{G}_k^{(1)}$ and $\mathbf{G}_k$ are to be determined. The *state-error* vector is defined by

$$\tilde{\mathbf{x}}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k \tag{3.38}$$

Applying the principle of orthogonality to the situation at hand, we may thus write

$$E\left[\tilde{\mathbf{x}}_k\mathbf{y}_i^T\right] = 0 \text{ for } i = 1, 2, ..., k-1 \tag{3.39}$$

Using Eqs. (3.30), (3.37), and (3.38) in (1.39), we get

$$E[(\mathbf{x}_k - \mathbf{G}_k^{(1)}\hat{\mathbf{x}}_k^- - \mathbf{G}_k\mathbf{H}_k\mathbf{x}_k - \mathbf{G}_k\mathbf{v}_k)\mathbf{y}_i^T] = 0 \text{ for } i = 1, 2..., k. \tag{3.40}$$

Since the process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are uncorrelated, it follows that

$$E[\mathbf{v}_k\mathbf{y}_i^T] = 0 \tag{3.41}$$

Using this relation and adding the element $\mathbf{G}_k^{(1)}\mathbf{x}_k - \mathbf{G}_k^{(1)}\mathbf{x}_k$, we may rewrite Eq. (3.40) as

$$E[(\mathbf{I} - \mathbf{G}_k\mathbf{H}_k - \mathbf{G}_k^{(1)})\mathbf{x}_k\mathbf{y}_i^T + \mathbf{G}_k^{(1)}(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)\mathbf{y}_i^T] = 0 \tag{3.42}$$

where $\mathbf{I}$ is the identity matrix. From the principle of orthogonality, we now note that

$$E[(\mathbf{x}_k - \mathbf{x}_k^-)\mathbf{y}_i^T] = 0 \tag{3.43}$$

Accordingly, Eq. (3.42) simplifies to

$$(\mathbf{I} - \mathbf{G}_k\mathbf{H}_k - \mathbf{G}_k^{(1)})E[\mathbf{x}_k\mathbf{y}_i^T] = 0 \text{ for } i = 1, 2, ..., k-1 \tag{3.44}$$

For arbitrary values of the state $\mathbf{x}_k$ and observable $\mathbf{y}_i$, Eq. (3.44) can only be satisfied if the scaling factors $\mathbf{G}_k^{(1)}$ and $\mathbf{G}_k$ are related as follows:

$$\mathbf{I} - \mathbf{G}_k\mathbf{H}_k - \mathbf{G}_k^{(1)} = 0 \tag{3.45}$$

or, equivalently, $\mathbf{G}_k^{(1)}$ is defined in terms of $\mathbf{G}_k$ as

$$\mathbf{G}_k^{(1)} = \mathbf{I} - \mathbf{G}_k\mathbf{H}_k \tag{3.46}$$

Substituting Eq. (3.46) into (3.37), we may express the a posteriori estimate of the state at time *k* as

$$\mathbf{x}_k = \hat{\mathbf{x}}_k^- + \mathbf{G}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-) \tag{3.47}$$

in light of which, the matrix $\mathbf{G}_k$ is called the *Kalman gain*.

There now remains the problem of deriving an explicit formula for $\mathbf{G}_k$. Since, from the principle of orthogonality, we have

$$E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)\mathbf{y}_i^T] = 0 \tag{3.48}$$

it follows that

$$E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)\hat{\mathbf{y}}_i^T] = 0 \tag{3.49}$$

where $\hat{\mathbf{y}}_k^T$ is an estimate of $\mathbf{y}_k$ given the previous measurement $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_{k-1}$. Define the innovations process

$$\tilde{\mathbf{y}}_k = \mathbf{y}_k - \hat{\mathbf{y}}_k \tag{3.50}$$

The innovation process represents a measure of the "new" information contained in $\mathbf{y}_k$; it may also be expressed as

$$\tilde{\mathbf{y}}_k = \mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-$$

$$= \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-$$

$$= \mathbf{v}_k + \mathbf{H}_k\tilde{\mathbf{x}}_k^- \tag{3.51}$$

Hence, subtracting Eq. (3.49) from (3.48) and then using the definition of Eq. (3.50), we may write

$$E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)\tilde{\mathbf{y}}_k^T] = 0 \tag{3.52}$$

Using Eqs. (3.39) and (3.47), we may express the state-error vector $\mathbf{x}_k - \hat{\mathbf{x}}_x$ as

$$\mathbf{x}_k - \hat{\mathbf{x}}_k = \tilde{\mathbf{x}}_k^- - \mathbf{G}_k(\mathbf{H}_k\tilde{\mathbf{x}}_k^- + \mathbf{v}_k)$$

$$= (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\tilde{\mathbf{x}}_k^- - \mathbf{G}_k\mathbf{v}_k \tag{3.53}$$

Hence, substimting Eqs. (3.51) and (3.53) into (3.52), we get

$$E[\{(\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\tilde{\mathbf{x}}_k^- - \mathbf{G}_k\mathbf{v}_k\}(\mathbf{H}_k\tilde{\mathbf{x}}_k^- + \mathbf{v}_k)] = 0 \tag{3.54}$$

Since the measurement noise $\mathbf{v}_k$ is independent of the state $\mathbf{x}_k$ and therefore the error $\tilde{\mathbf{x}}_k^-$ the expectation of Eq. (3.54) reduces to

$$(\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)E[\tilde{\mathbf{x}}_k^-\tilde{\mathbf{x}}_k^{-T}]\mathbf{H}_k^T - \mathbf{G}_k E[\mathbf{v}_k\mathbf{v}_k^T] = 0 \tag{3.55}$$

Define the *a priori covariance matrix*

$$\mathbf{P}_k^- = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)^T]$$

$$= E[\tilde{\mathbf{x}}_k^-\tilde{\mathbf{x}}_k^{-T}] \tag{3.56}$$

Then, invoking the covariance definitions of Eqs. (3.31) and (3.56), we may rewrite Eq. (3.55) as

$$(\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\mathbf{P}_k^-\mathbf{H}_k^T - \mathbf{G}_k\mathbf{R}_k = 0 \tag{3.57}$$

Solving this equation for $\mathbf{G}_k$, we get the desired formula

$$\mathbf{G}_k = \mathbf{P}_k^-\mathbf{H}_k^T[\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k]^{-1} \tag{3.58}$$

where the symbol $[\bullet]^{-1}$ denotes the inverse of the matrix inside the square brackets. Equation (3.49) is the desired formula for computing the Kalman gain $\mathbf{G}_k$, which is defined in terms of the a priori covariance matrix $\mathbf{P}_k^-$. To complete the recursive estimation procedure, we consider the *error covariance propagation*, which describes the effects of time on the covariance matrices of estimation errors. This propagation involves two stages of computation:

1. The a priori covariance matrix $\mathbf{P}_k^-$ at time k is defined by Eq. (1.21). Given $\mathbf{P}_k^-$, compute the a posteriori covariance matrix $\mathbf{P}_k$, which, at time *k*, is defined by

$$\mathbf{P}_k = E[\tilde{\mathbf{x}}_k\tilde{\mathbf{x}}_k{}^T]$$

$$= E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T] \tag{3.59}$$

2. Given the "old" a posteriori covariance matrix, $\mathbf{P}_{k-1}$, compute the "updated" a priori covariance matrix $\mathbf{P}_k^-$.

To proceed with stage 1, we substitute Eq. (3.52) into (3.59) and note that the noise process $\mathbf{v}_k$ is independent of the a priori estimation error $\tilde{\mathbf{x}}_k^-$. We thus obtain

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)E[\tilde{\mathbf{x}}_k^-\tilde{\mathbf{x}}_k^{-T}](\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)^T + \mathbf{G}_k E[\mathbf{v}_k\mathbf{v}_k^T]\mathbf{G}_k^T$$

$$= (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\mathbf{P}_k^-(\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)^T + \mathbf{G}_k\mathbf{R}_k\mathbf{G}_k^T \tag{3.60}$$

Expanding terms in Eq. (3.60) and then using Eq. (3.58), we may reformulate the dependence of the a posteriori covariance matrix $\mathbf{P}_k$ on the a priori covariance matrix $\mathbf{P}_k^-$ in the simplified form

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\mathbf{P}_k^- - (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\mathbf{P}_k^-\mathbf{H}_k^T\mathbf{G}_k^T + \mathbf{G}_k\mathbf{R}_k\mathbf{G}_k^T$$

$$= (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\mathbf{P}_k^- - \mathbf{G}_k\mathbf{R}_k\mathbf{G}_k^T + \mathbf{G}_k\mathbf{R}_k\mathbf{G}_k^T$$

$$= (\mathbf{I} - \mathbf{G}_k\mathbf{H}_k)\mathbf{P}_k^- \tag{3.61}$$

For the second stage of error covariance propagation, we first recognize that the a priori estimate of the state is defined in terms of the "old" a posteriori estimate as follows:

$$\tilde{\mathbf{x}}_k^- = \mathbf{F}_{k,k-1}\hat{\mathbf{x}}_{k-1} \tag{3.62}$$

We may therefore use Eqs. (3.28) and (3.62) to express the a priori estimation error in yet another form:

$$\tilde{\mathbf{x}}_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^-$$

$$= (\mathbf{F}_{k,k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1}) - (\mathbf{F}_{k,k-1}\hat{\mathbf{x}}_{k-1})$$

$$= \mathbf{F}_{k,k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}) + \mathbf{w}_{k-1}$$

$$= \mathbf{F}_{k,k-1}\tilde{\mathbf{x}}_{k-1} + \mathbf{w}_{k-1} \tag{3.63}$$

Accordingly, using Eq. (3.63) in (3.56) and noting that the process noise $\mathbf{w}_k$ is independent of $\hat{\mathbf{x}}_{k-1}$ we get

$$\mathbf{P}_k^- = \mathbf{F}_{k,k-1}E[\tilde{\mathbf{x}}_{k-1}\tilde{\mathbf{x}}_{k-1}^T]\mathbf{F}_{k,k-1}^T + E[\mathbf{w}_{k-1}\mathbf{w}_{k-1}^T]$$

$$= \mathbf{F}_{k,k-1}\mathbf{P}_{k-1}\mathbf{F}_{k,k-1}^T + \mathbf{Q}_{k-1} \tag{3.64}$$

which defines the dependence of the a priori covariance matrix $\mathbf{P}_k^-$ on the "old" a posteriori covariance matrix $\mathbf{P}_{k-1}$.

With Eqs. (3.62), (3.64), (3.58), (3.47), and (3.61) at hand, we may now summarize the recursive estimation of state as shown in figure 3.9. This figure also includes the initialization. In the absence of any observed data at time k = 0, we may choose the initial estimate of the state as

$$\mathbf{x}_0 = E[\mathbf{x}_0] \tag{3.65}$$

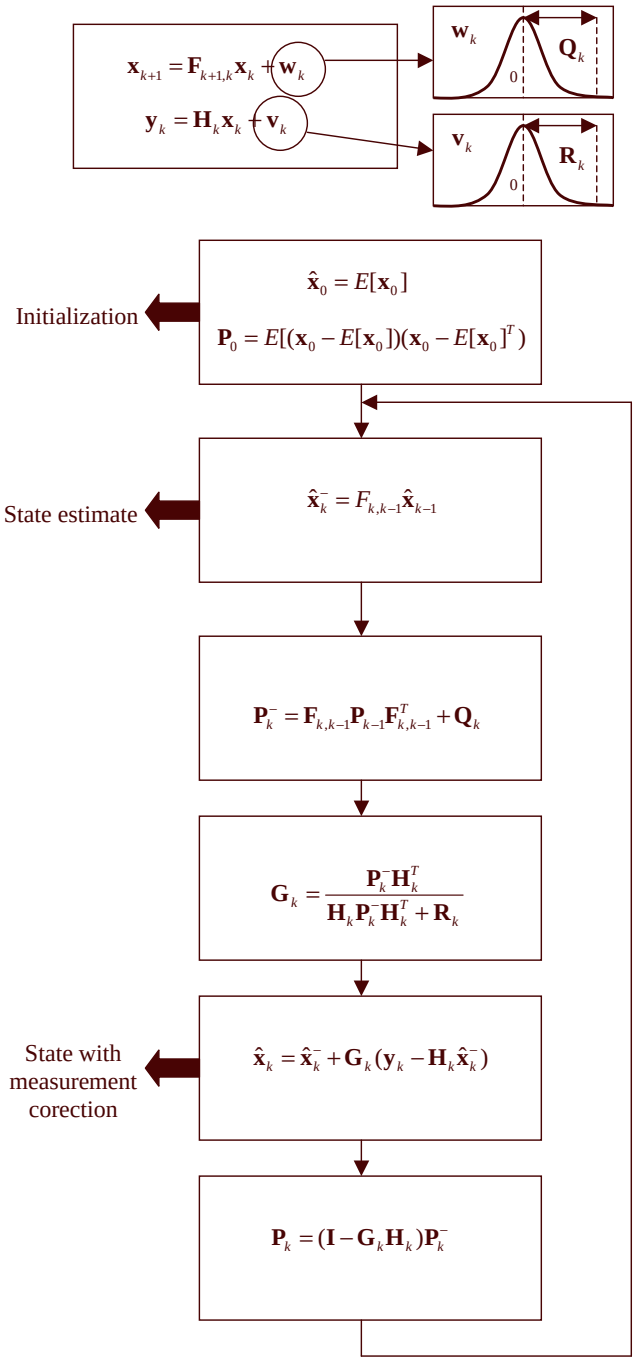


Figure 3.9: Sumary of the Kalman filter

and the initial value of the a posteriori covariance matrix as

$$\mathbf{P}_0 = E[(\mathbf{x}_0 - E[\mathbf{x}_0])((\mathbf{x}_0 - E[\mathbf{x}_0])^T] \tag{3.66}$$

This choice for the initial conditions not only is intuitively satisfying but also has the adventage of yielding an *unbiased* estimate of the state $\mathbf{x}_k$.

The Kalman filter uses Gaussian probability density in the propagation process, the diffusion is purely linear and the density function evolves as a gaussian pulse that translates, spreads and reinforced, remaining gaussian throughout.

The random component of the dynamical model $\mathbf{w}_k$ leads to spreading (increasing uncertainty) while the deterministic component $\mathbf{F}_{k+1,k}\mathbf{x}_k$ causes the density function to drift bodily. The effect of an external observation y is to superimpose a reactive effect on the diffusion in which the density tends to peak in the vicinity of observations.

3.8 Extended Kalman filter

The Kalman filtering problem considered up to this point in the discussion has addressed the estimation of a state vector in a linear model of a dynamical system. If, however, the model is *nonlinear*, we may extend the use of Kalman filtering through a linearization procedure. The resulting filter is referred to as *the extended Kalman filter* (**EKF**) [155,157]. Such an extension is feasible by virtue of the fact that the Kalman filter is described in terms of difference equations in the case of discrete-time systems.

To set the stage for a development of the extended Kalman filter, consider a nonlinear dynamical system described by the state-space model

$$\mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k) + \mathbf{w}_k \tag{3.67}$$

$$\mathbf{y}_k = \mathbf{h}(k, \mathbf{x}_k) + \mathbf{v}_k \tag{3.68}$$

where, as before, $\mathbf{w}_k$ and $\mathbf{v}_k$ are independent zero-mean white Gaussian noise processes with covariance matrices $\mathbf{R}_k$ and $\mathbf{Q}_k$ respectively. Here, however, the functional $\mathbf{f}(k, \mathbf{x}_k)$ denotes a nonlinear transition matrix function that is possibly time-variant. Likewise, the functional $\mathbf{h}(k, \mathbf{x}_k)$ denotes a nonlinear measurement matrix that may be time-variant, too.

The basic idea of the extended Kalman filter is to linearize the state-space model of Eqs. (3.69) and (3.70) at each time instant around the most recent state estimate, which is taken to be either $\hat{\mathbf{x}}_k$ or $\hat{\mathbf{x}}_k^-$ epending on which particular functional is being considered. Once a linear model is obtained, the standard Kalman filter equations are applied.

More explicitly, the approximation proceeds in two stages.

Stage 1. The following two matrices are constructed:

$$\mathbf{F}_{k+1,k} = \left.\frac{\partial \mathbf{f}(k, \mathbf{x}_k)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_k} \tag{3.69}$$

$$\mathbf{H}_k = \left.\frac{\partial \mathbf{h}(k, \mathbf{x}_k)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_k} \tag{3.70}$$

That is, the *ij*th entry of $\mathbf{F}_{k+1,k}$ is equal to the partial derivative of the *i*th component of $\mathbf{F}(k, \mathbf{x})$ with respect to the yth component of $\mathbf{x}$. Likewise, the *ij*th entry of $\mathbf{H}_k$ is equal to the partial derivative of the *i*th component of $\mathbf{H}(k, \mathbf{x})$ with respect to the *j*th component of $\mathbf{x}$. In the former case, the derivatives are evaluated at $\hat{\mathbf{x}}_k$ while in the latter case, the derivatives are evaluated at $\hat{\mathbf{x}}_k^-$. The entries of the matrices $\mathbf{F}_{k+1,k}$ and $\mathbf{H}_k$ are all known (i.e., computable), by having $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_k^-$ available at time *k*.

Stage 2. Once the matrices $\mathbf{F}_{k+1,k}$ and $\mathbf{H}_k$ are evaluated, they are then employed in a *first-order Taylor approximation* of the nonlinear functions $\mathbf{F}(k, \mathbf{x})$ and $\mathbf{H}(k, \mathbf{x})$ around $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_k^-$, respectively. Specifically, $\mathbf{F}(k, \mathbf{x})$ and $\mathbf{H}(k, \mathbf{x})$ are approximated as follows

$$\mathbf{F}(k, \mathbf{x}_k) \approx \mathbf{F}(\mathbf{x}, \hat{\mathbf{x}}_k) + \mathbf{F}_{k+1,k}(\mathbf{x}, \hat{\mathbf{x}}_k) \tag{3.71}$$

$$\mathbf{H}(k, \mathbf{x}_k) \approx \mathbf{H}(\mathbf{x}, \hat{\mathbf{x}}_k^-) + \mathbf{H}_{k+1,k}(\mathbf{x}, \hat{\mathbf{x}}_k^-) \tag{3.72}$$

With the above approximate expressions at hand, we may now proceed to approximate the nonlinear state equations (3.67) and (3.68) as shown by, respectively,

$$\mathbf{x}_{k+1} \approx \mathbf{F}_{k+1,k}\mathbf{x}_k + \mathbf{w}_k + \mathbf{d}_k \tag{3.73}$$

$$\bar{\mathbf{y}}_k \approx \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{3.74}$$

where we have introduced two new quantities:

$$\bar{\mathbf{y}}_k = \mathbf{y}_k - \{\mathbf{h}(\mathbf{x}, \hat{\mathbf{x}}_k^-) - \mathbf{H}_k\hat{\mathbf{x}}_k^-\} \tag{3.75}$$

$$\mathbf{d}_k = \mathbf{f}(\mathbf{x}, \hat{\mathbf{x}}_k) - \mathbf{F}_{k+1,k}\hat{\mathbf{x}}_k \tag{3.76}$$

The entries in the term $\bar{\mathbf{y}}_k$ are all known at time *k*, and, therefore, $\bar{\mathbf{y}}_k$ can be regarded as an observation vector at time *n*. Likewise, the entries in the term $\mathbf{d}_k$ are all known at time *k*.

Given the linearized slate-space model of Eqs. (3.75) and (3.76), we may then proceed and apply the Kalman filler theory of Section 3.7 to derive the extended Kalman filter. Figure 3.10 summarizes the recursions involved in computing the extended Kalman filter.

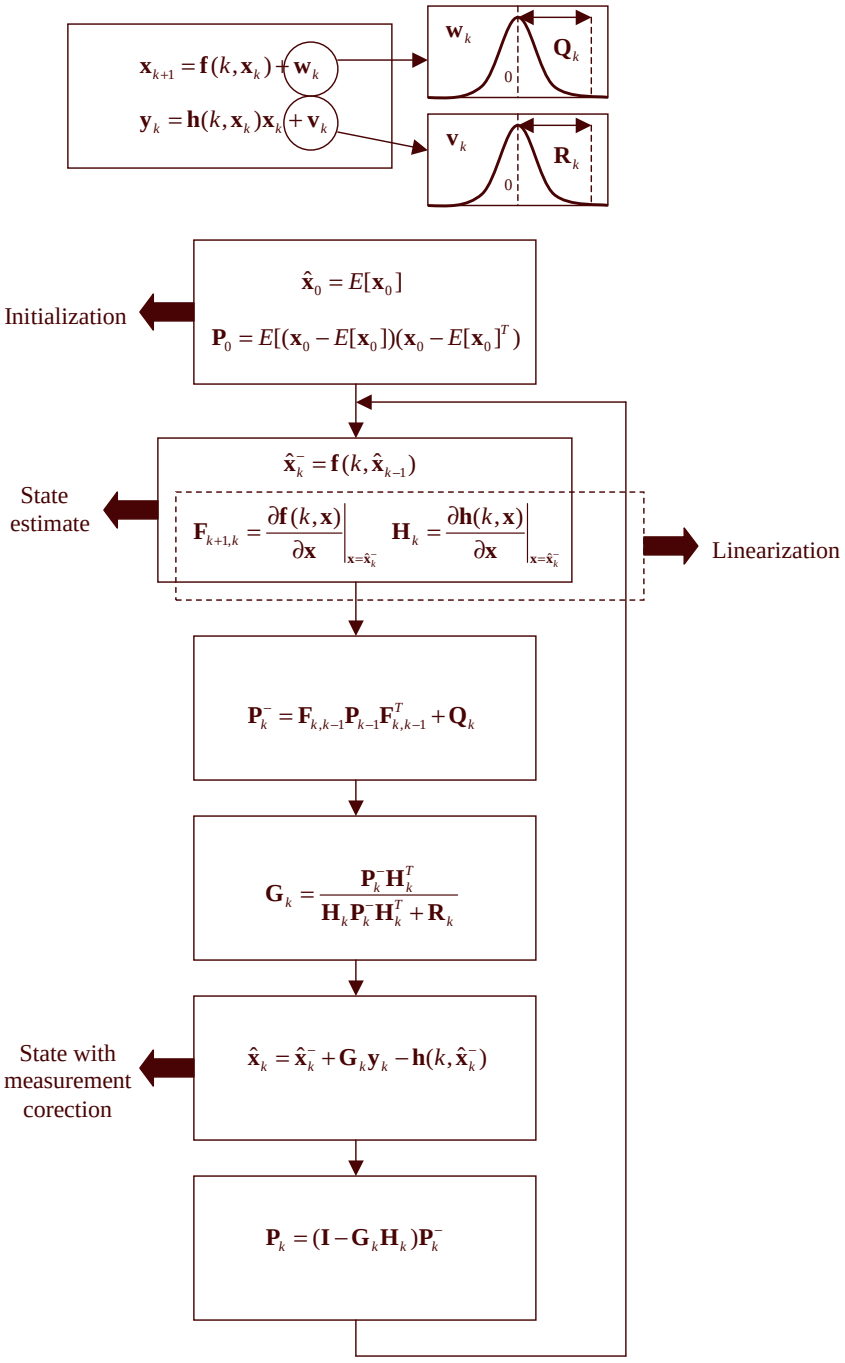


Figure 3.10: Summary of the Extended Kalman Filter

3.9 Vision Tracking with the Kalman filter

The main application of the Kalman filter in robot vision is the following object, also called *tracking* [158]. To carry out this, it is necessary to calculate the object position and speed in each instant. As input is considered a sequence of images captured by a camera containing the object. Then using a image processing method the object is segmented and later calculated their position in the image. Therefore we will take as system state $\mathbf{x}_k$the position x and y of the object in the instant k. Considering the above-mentioned we can use the Kalman filter to make more efficient the localization method of the object, that is to say instead of looking for to the object in the whole image plane we define a search window centered in the predicted value $\hat{\mathbf{x}}_k^-$ of the filter.

The steps to use the Kalman filter for vision tracking are:

1. Initialization (k=0). In this step it is looked for the object in the whole image due we do not know previously the object position. We obtain this way $\mathbf{x}_0$. Also we can considerer initially a big error tolerance ($\mathbf{P}_0 = 1$).

2. Prediction (k>0). In this stage using the Kalman filter we predict the relative position of the object, such position $\hat{\mathbf{x}}_k^-$ is considered as search center to find the object.

3. Correction (k>0). In this part we locate the object (which is in the neighborhood point predicted in the previous stage $\hat{\mathbf{x}}_k^-$) and we use its real position (measurement) to carry out the state correction using the Kalman filter finding this way $\hat{\mathbf{x}}_k$.

The steps 2 and 3 are carried out while the object tracking runs. To exemplify the results of the use of the Kalman filter in vision tracking, we choose the tracking of a soccer ball and consider the following cases:

a) In this test we carry out the ball tracking considering a lineal uniform movement, which could be described by the following system equations

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1,k}\mathbf{x}_k + \mathbf{w}_k$$

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \Delta x_{k+1} \\ \Delta y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \Delta x_k \\ \Delta y_k \end{bmatrix} + \mathbf{w}_k$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

$$\begin{bmatrix} xm_k \\ ym_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \Delta x_k \\ \Delta y_k \end{bmatrix} + \mathbf{v}_k$$

In the figure 3.11 the prediction of the object position is shown for each instant as well as the real trajectory.

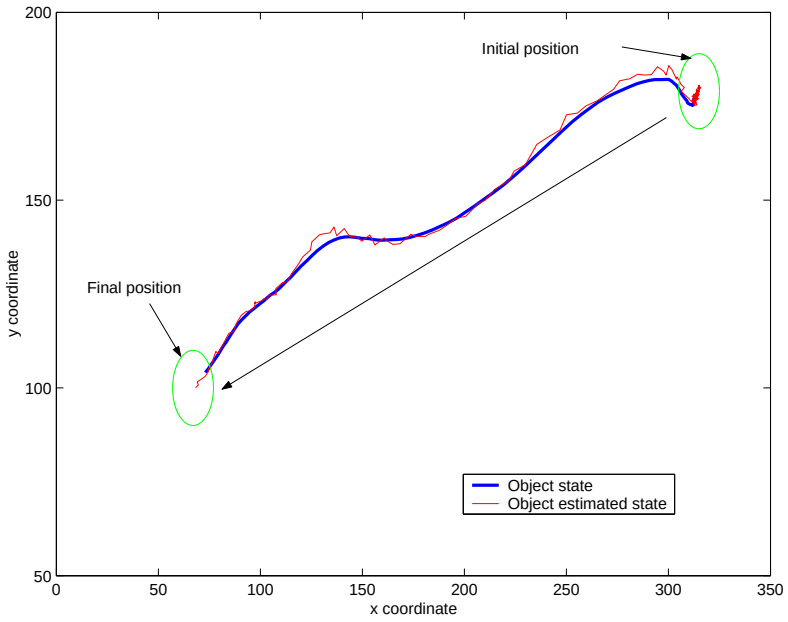


Figure 3.11: Position prediction with the Kalman filter

b) One advantage of the Kalman filter for the vision tracking is that can be used to tolerate small occlusions. The form to carrying out it, is to consider the two work phases of the filter, prediction and correction. That is to say, if the object localization is not in the neighborhood of the predicted state by the filter (in the instant *k*), we can consider that the object is hidden by some other object, consequently we will not use the measurement correction and will only take as object position the filter prediction. The figure 3.12 shows the filter performance during the object occlusion. The system was modeled with the same equations used in the previous case.

c) Most of the complex dynamic trajectories (changes of acceleration) cannot be modeled by lineal systems, which results in that we have to use for the modeling nonlinear equations, therefore in these cases we will use the extended Kalman filter. The figure 3.13 shows the acting of the extended Kalman filter for the vision tracking of a complex trajectory versus the poor performance of the normal Kalman filter. For the extended Kalman filter the dynamic system was modeled using the unconstrained Brownian motion equations

$$\mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k) + \mathbf{w}_k$$

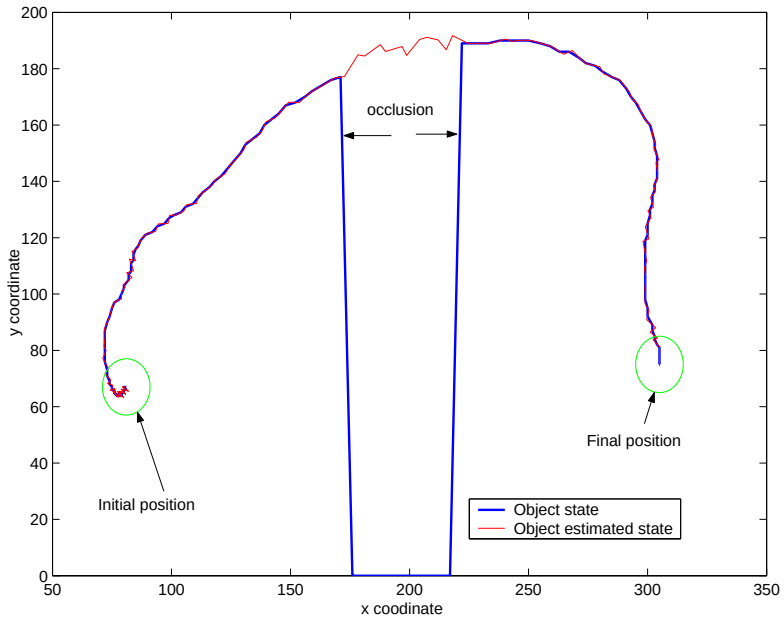


Figure 3.12: Kalman filter during the occlusion

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \Delta x_{k+1} \\ \Delta y_{k+1} \end{bmatrix} = \begin{bmatrix} \exp\left(-\frac{1}{4}(x_k + 1.5\Delta x_k)\right) \\ \exp\left(-\frac{1}{4}(y_k + 1.5\Delta y_k)\right) \\ \exp\left(-\frac{1}{4}\Delta x_k\right) \\ \exp\left(-\frac{1}{4}\Delta y_k\right) \end{bmatrix} + \mathbf{w}_k$$

$$\mathbf{y}_k = \mathbf{h}(k, \mathbf{x}_k) + \mathbf{v}_k$$

$$\begin{bmatrix} xm_k \\ ym_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \Delta x_k \\ \Delta y_k \end{bmatrix} + \mathbf{v}_k$$

while for the normal Kalman filter, the system was modeled using the equations of the case a).

3.10 Particle filter for vision tracking.

In this chapter, the extended Kalman filter (EKF) has been used as the standard technique for performing recursive nonlinear estimation in vision tracking. The EKF algorithm, however, provides only an approximation to optimal

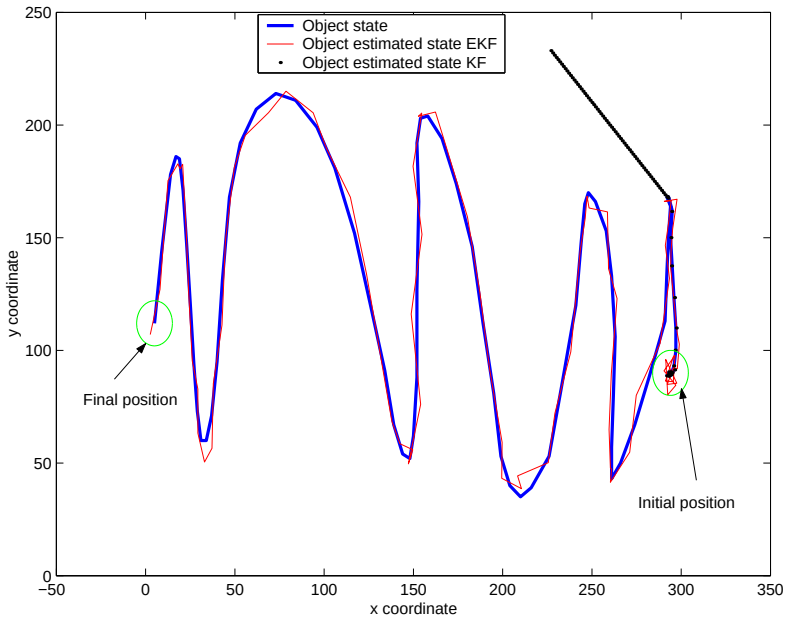


Figure 3.13: Tracking of a complex movement using an EKF.

nonlinear estimation. In this section, we present an alternative filter with performance superior to that of the EKF. This algorithm, referred to as the *Particle filter* (Condensation filter, as it is known in the vision community). The basic difference between the EKF and Particle filter stems from the manner in which random variables are represented for propagating through system dynamics. In the EKF, the state distribution is approximated by a gaussian random variable which is then propagated analytically through the first-order linearization of the nonlinear system. This can introduce large errors in the true posterior mean and covariance of the transformed gaussian random variables which may lead to suboptimal performance and sometimes divergence of the filter. The Particle filter address this problem by using a deterministic sampling approach. The state distribution is approximated by a random variable (not necessarily gaussian), but is now represented using a minimal set of carefully chosen sample points. These sample points completely capture the true mean and covariance of the random variable and, when propagated through the *true* nonlinear system, captures the posterior mean and covariance accurately to second order (Taylor series expansion) for any nonlinearity. The EKF, in contrast, only achieves first-order accuracy. No explicit Jacobian or Hessian calculations are necessary for the Particle filter. Remarkably, the computational complexity of the particle filter is the same order as that of the EKF.

3.11 Optimal recursive estimation

Given observations $\mathbf{y}_k$, the goal is to estimate the state $\mathbf{x}_k$. We make no assumptions about the nature of the system dynamics at this point. The optimal estimate in the minimum mean-squared error (MMSE) sense is given by the conditional mean:

$$\hat{\mathbf{x}}_k = E[\mathbf{x}_k | \mathbf{Y}_0^k] \tag{3.77}$$

where $\mathbf{Y}_0^k$ is the sequence of observations up to time *k*. Evaluation of this expectation requires knowledge of the *a posteriori* density $p(\mathbf{x}_k | \mathbf{Y}_0^k)$. Given this density, we can determine not only the MMSE estimator, but any "best" estimator under a specified performance criterion. The problem of determining the a posteriori density is in general referred to as the Bayesian approach, and can be evaluated recursively according to the following relations:

$$p(\mathbf{x}_k | \mathbf{Y}_0^k) = \frac{p(\mathbf{x}_k | \mathbf{Y}_0^{k-1}) p(\mathbf{y}_k | \mathbf{x}_k)}{p(\mathbf{y}_k | \mathbf{Y}_0^{k-1})} \tag{3.78}$$

where

$$p(\mathbf{x}_k | \mathbf{Y}_0^{k-1}) = \int p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{y}_k | \mathbf{x}_k) d\mathbf{x}_{k-1} \tag{3.79}$$

and the normalizing constant $p(\mathbf{x}_k | \mathbf{Y}_0^k)$ is given by

$$p(\mathbf{y}_k | \mathbf{Y}_0^{k-1}) = \int p(\mathbf{x}_k | \mathbf{Y}_0^{k-1}) p(\mathbf{y}_k | \mathbf{x}_k) d\mathbf{x}_k \tag{3.80}$$

This recursion specifies the current state density as a function of the previous density and the most recent measurement data. The state-space model comes into play by specifying the state transition probability $p(\mathbf{x}_k | \mathbf{x}_{k-1})$ and measurement probability or likelihood, $p(\mathbf{y}_k | \mathbf{x}_k)$. Specifically, $p(\mathbf{x}_k | \mathbf{x}_{k-1})$ is determined by the process noise density $p(\mathbf{w}_k)$ with the state-update equation

$$\mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k) + \mathbf{w}_k \tag{3.81}$$

For example, given an additive noise model with Gaussian density, $p(\mathbf{w}_k) = \mathrm{N}(0, \mathbf{R}^{\mathbf{v}})$, then $p(\mathbf{x}_k | \mathbf{x}_{k-1}) = \mathrm{N}(\mathbf{F}(\mathbf{x}_{k-1}), \mathbf{R}^{\mathbf{v}})$. Similarly, $p(\mathbf{y}_k | \mathbf{x}_k)$ is determined by the observation noise density $p(\mathbf{v}_k)$ and the measurement equation

$$\mathbf{y}_k = \mathbf{h}(k, \mathbf{x}_k) + \mathbf{v}_k \tag{3.82}$$

In principle, knowledge of these densities and the initial condition $p(\mathbf{x}_0 | \mathbf{y}_0) = \frac{p(\mathbf{y}_0 | \mathbf{x}_0) p(\mathbf{x}_0)}{p(\mathbf{y}_0)}$ determines $p(\mathbf{x}_k | \mathbf{Y}_0^k)$ for all *k*. Unfortunately, the multidimensional integration indicated by Eqs. 3.78-3.80 makes a closed-form solution intractable for most systems. The only general approach is to apply Monte Carlo sampling techniques that essentially convert integrals to finite sums, which converge to the true solution in the limit.

Particle filtering [159, 161] was originally developed to track objects in clutter or a variable of interest as it evolves over time, typically with a non-Gaussian and potentially multi-modal probability density function (pdf). The basis of the method is to construct a sample-based representation of the entire pdf (equation 3.78). A series of actions are taken, each one modifying the state of the variable of interest according to some model (equation 3.81). Moreover at certain times an observation arrives that constrains the state of the variable of interest at that time.

Multiple hypothetical state (particles) of the variable of interest $\mathbf{x}_k$ are used, each one associated with a weight that signifies the quality of that specific particle. An estimate of the variable of interest is obtained by the weighted sum of all the particles. The particle filter algorithm is recursive in nature $\mathbf{f}(k, \mathbf{x}_k)$ (*prediction* stage), including the addition of random noise $\mathbf{w}_k$ in order to simulate the effect of noise on the variable of interest. Then, each particle's weight is re-evaluated based on the latest measurements available (*update* stage). At times the particles with small weights are eliminated, with a process called *resampling*. More formally, the variable of interest (in this case the object position $\mathbf{x}_k = [\ x_k \quad y_k\ ]$) at time *k* is represented as a set of *M* samples (the "particles") $S_k^i = [\ \mathbf{x}_k^i \quad b_k^i\ ] : i = 1, 2, ..., M$, where the index *i* denotes the particle number, each particle consisting of a hypothetical value of the variable of interest $\mathbf{x}_k$ and a weight *b* that defines the contribution of this particle to the overall estimate of the variable, where $\sum_{i=1}^{M} b_k^i = 1$. The figure 3.14 shown the process carried out by the particle filter.

If at time *k* we know the pdf of the system at the previous instant *k* -1 then we model the movement effect with $\mathbf{f}(k, \mathbf{x}_k)$ to obtain a prior of the pdf at time k (prediction). In other words, the *prediction* phase uses a model in order to simulate the effect that a movement has on the set of particles with the appropriate noise added $\mathbf{w}_k$. The *update* phase uses the information obtained from the measurements to update the particle weights in order to accurately describe the moving object's pdf. Algorithm 1 presents a formal description of the particle filter algorithm.

3.12 Particle filter in Vision Tracking

Robust real-time tracking of non-rigid objects in computer vision is a challenging task. Particle filtering has proven very successful for non-linear and non-Gaussian estimation problems. Particle filtering was originally developed to track objects in clutter, that is to say, one of its main characteristics represents the possibility to track objects although exists the presence of other objects that have similar characteristic.

We want to apply a particle filter in a color-based context. Color distributions are used as target models as they achieve robustness against non-rigidity, rotation and partial occlusion. Suppose that the distributions are discretized into *m*-bins. The histograms are produced with the function $h(\mathbf{x}^i)$, that assigns the color at location to the corresponding bin (considering $\mathbf{x}^i$ the pixel coordi-

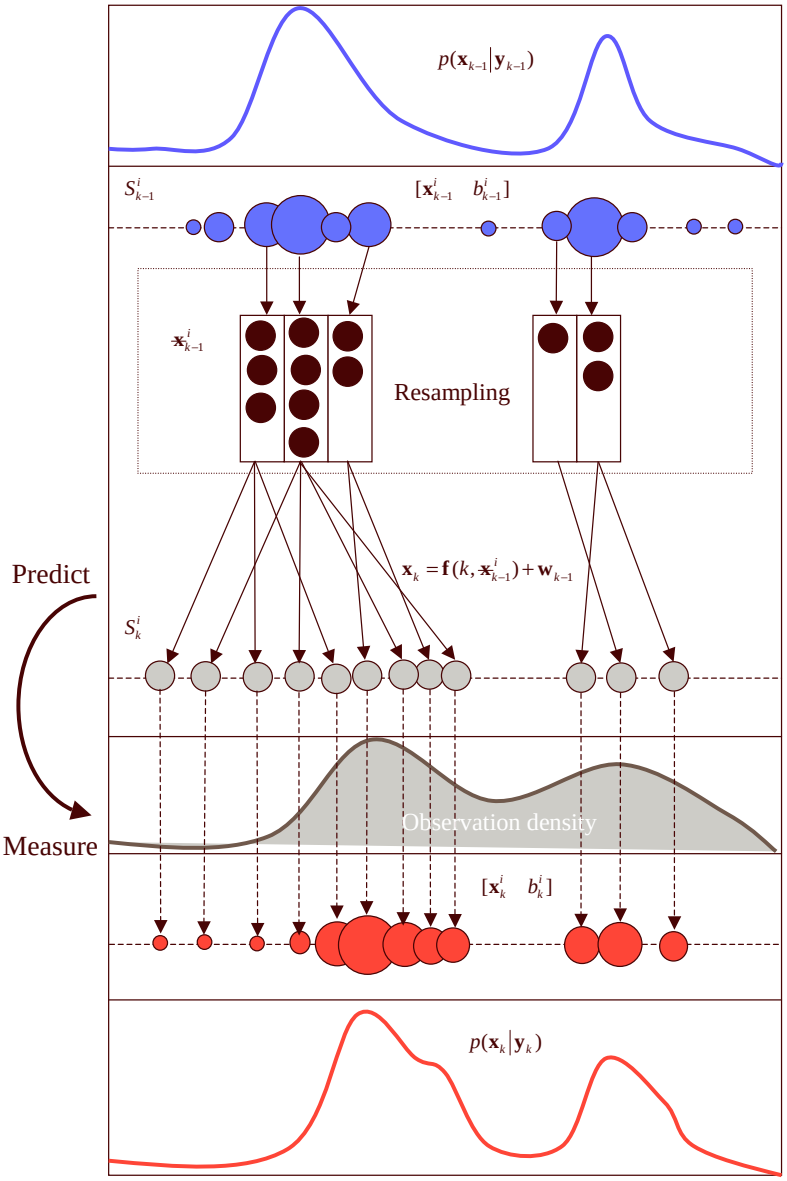


Figure 3.14: Process carried out by the particle filter.

Algorithm 1 Particle Filter Algorithm

From the particles at time-step *k*-1, $S^i_{k-1} = \{\mathbf{x}^i_{k-1}, b^i_{k-1} \big| \, i = 1, 2, ..., M\}$.
1. For each particle we calculate the cumulative probability as

$$\begin{array}{c} c^0_{k-1} = 0 \\ c^i_{k-1} = c^{i-1}_{k-1} + b^i_{k-1} \end{array} \Bigg| \, i = 1, 2, ..., M$$

we have in this way $S^i_{k-1} = \{\mathbf{x}^i_{k-1}, b^i_{k-1}, c^i_{k-1} \big| \, i = 1, 2, ..., M\}$.
2. We select M states (they can repeat) starting from S^i_{k-1} (resampling), carrying out the following procedure

- We generate a random number $r \in [0, 1]$, uniformly distributed.
- We find , the smallest *j* for which $c^j_{k-1} \geqslant r$.
- The elected state is $\mathbf{x}^i_{k-1} = \mathbf{x}^j_{k-1}$

3. We spread the states $\{\mathbf{x}^i_{k-1} \big| \, i = 1, 2, 3..., M\}$ using the model $\mathbf{x}^i_k = \mathbf{f}(k, \mathbf{x}^i_{k-1}) + \mathbf{w}_k$.
4. For each new state $\mathbf{x}^i_k$ we find their corresponding *b* starting from the measurement $p(\mathbf{y} | \, \mathbf{x})$obtained for each hypothesis.
5. We carry out the normalization $\sum_{i=1}^{M} b^i_k = 1$ and build the particles $S^i_k = \{\mathbf{x}^i_k, b^i_k \big| \, i = 1, 2, ..., M\}$.
6. Once the *M* samples have been constructed: estimate, if desired, moments of the tracked position at time *k* as

$$E[S^i_k] = \sum\nolimits_{i=1}^{M} b^i_k \mathbf{x}^i_k$$

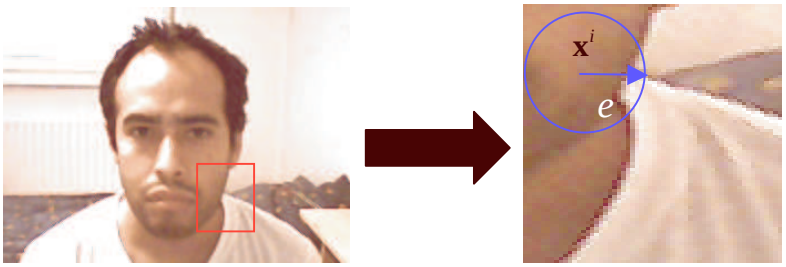


Figure 3.15: Configuration of the density of the particles, centered in $\mathbf{x}^i$dependent of the distance e.

nates (*x*,*y*)). In our experiments, the histograms are typically calculated in the RGB space using 8x8x8 bins. To make the algorithm less sensitive to lighting conditions, the HSV color space could be used instead with less sensitivity to V (e.g. 8x8x4 bins).

We determine the color distribution inside an upright circular region centered in with radius *r*. To increase the reliability of the color distribution when boundary pixels belong to the background or get occluded, smaller weights are assigned to the pixels that are further away from the region center by employing a weighting function

$$k(e) = \begin{cases} 1 - e^2 & e < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.83}$$

where e is the distance from the region center. Thus, we increase the reliability of the color distribution when these boundary pixels belong to the background or get occluded. The figure 3.15 shows the advantage of using the distance e to improve the reliability of the measurement.

The color distribution $p_y = \{p_y^{(u)}\}_{u=1,2,3,...m.}$ at location y is calculated as

$$p_{\mathbf{y}}^{(u)} = f \sum_{i=1}^{I} k\left(\frac{|\mathbf{y} - \mathbf{x}^i|}{r}\right) \delta[h(\mathbf{x}^i) - u] \tag{3.84}$$

where I is the number of pixels in the circular region, δ is the Kronecker delta function and the normalization factor

$$f = \frac{1}{\sum_{i=1}^{I} k\left(\frac{|\mathbf{y}-\mathbf{x}^i|}{r}\right)} \tag{3.85}$$

ensures that $\sum_{u=1}^{m} p_{\mathbf{y}}^{(u)} = 1$.

In a tracking approach, the estimated state is updated at each time step by incorporating the new observations. Therefore, we need a similarity measure which is based on color distributions. A popular measure between two distributions $p(u)$ and $q(u)$ is the Bhattacharyya coefficient [162, 163].

$$\rho[p,q] = \int \sqrt{p(u)q(u)}du \tag{3.86}$$

Considering discrete densities such as our color histograms $p = \{p^{(u)}\}_{u=1,2,3,...m.}$ and $q = \{q^{(u)}\}_{u=1,2,3,...m.}$ the coefficient is defined as

$$\rho[p,q] = \sum_{u=1}^{m} \sqrt{p^{(u)}q^{(u)}} \tag{3.87}$$

The larger ρ is, the more similar the distributions are. For two identical normalized histograms we obtain $\rho = 1$, indicating a perfect match. As distance between two distributions we define the measure

$$d = \sqrt{1 - \rho[p,q]} \tag{3.88}$$

which is called the Bhattacharyya distance.

The proposed tracker employs the Bhattacharyya distance to update the a priori distribution calculated by the particle filter. Each sample of the distribution represents a circle with radius *r* and is given as

$$\mathbf{x}_k^i = [\ x_k^i \quad y_k^i \quad \Delta x_k^i \quad \Delta y_k^i\ ] \tag{3.89}$$

where *x*, *y* specify the location of the circle, Δx and Δy the motion. As we consider a whole sample set the tracker handles multiple hypotheses simultaneously.

The sample set is propagated through the application of a dynamic model

$$\mathbf{x}_{k+1}^i = \mathbf{f}(k, \mathbf{x}_k^i) + \mathbf{w}_k^i \tag{3.90}$$

where $\mathbf{f}(k, \mathbf{x}_k^i)$ defines the deterministic component of the model and $\mathbf{w}_k^i$ is a multivariate Gaussian random variable. In this thesis we currently use an unconstrained Brownian model for describing the region movement with velocity Δx, Δy and radius *r*.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \Delta x_{k+1} \\ \Delta y_{k+1} \end{bmatrix} = \begin{bmatrix} \exp\left(-\frac{1}{4}(x_k + 1.5\Delta x_k)\right) \\ \exp\left(-\frac{1}{4}(y_k + 1.5\Delta y_k)\right) \\ \exp\left(-\frac{1}{4}\Delta x_k\right) \\ \exp\left(-\frac{1}{4}\Delta y_k\right) \end{bmatrix} + \mathbf{w}_k \tag{3.91}$$

To weight the sample set, the Bhattacharyya coefficient has to be computed between the target histogram and the histogram of the hypotheses. Each hypothetical region is specified by its state vector S_k^i. Both the target histogram q and the candidate histogram $p_{S_k^i}$ are calculated from Eq. 3.84 where the target is centered at the origin of the circular region.

As we want to favor samples whose color distributions are similar to the target model, small Bhattacharyya distances correspond to large weights

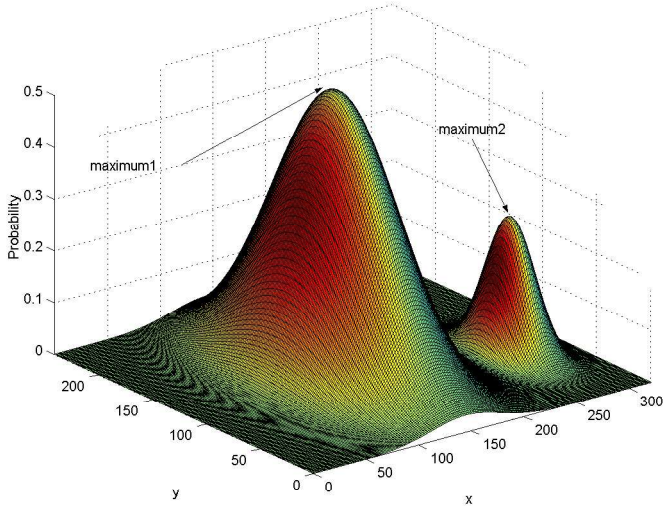
$$b^i = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(1-\rho[p_{S_k^i}, q])}{2\sigma}\right) \tag{3.92}$$

that are specified by a Gaussian with variance σ. During filtering, samples with a high weight may be chosen several times, leading to identical copies, while others with relatively low weights may not be chosen at all. The programming details for one iteration step are given in the Algorithm 1.

To illustrate the distribution of the sample set, Figure 3.15 shows the samples distribution considering the flesh color as target histogram q. The samples are located around the maximum of the Bhattacharyya coefficient which represents the best match to the target model.

Given a particle distribution S_k^i, we need to find the state which defines with accuracy the object position. Three different methods of evaluation have been used in order to obtain an estimate of the position. First, the weighted mean ($\hat{\mathbf{x}}_k \approx \sum_{i=1}^{M} b_k^i \mathbf{x}_k^i$) be used; second, the best particle (the $\mathbf{x}_k^j$ such that $b_k^j = \max(b_k^i) : i = 1, 2, ...M$) and, third, the weighted mean in a small window around the best particle (also called robust mean) can be used. Each method has its advantages and disadvantages: the weighted mean fails when faced with multi-modal distributions, while the best particle introduces a discretization error. The best method is the robust mean but it is also the most computationally expensive. In cases where the object to track is surrounded of objects whose characteristics are similar the best method is to use as state that defines the object position the best particle.

(a)



(b)

Figure 3.16: a) Distribution of the sample set and b) generated multi-modal probability density function.