# Towards efficient vision-based object tracking algorithms: rehabilitating visual information in particle filtering

**Péter Torma**

**Ph.D. Thesis**
**Supervisor: Csaba Szepesvári, Ph.D.**


Eötvös Loránd University
Faculty of Informatics

Informatics Ph.D. School
Foundations and Methods in Informatics PhD Program
János Demetrovics D.Sc.

Budapest, 2007

...for my kids Viola and Noémi
...to Junko, Mom and Dad

# Acknowledgement

Firstly I am grateful to my supervisor Csaba Szepesvári, not only for introducing me to many principles of machine learning, stochastic processes and AI, but also for teaching me how to research. I think my luck of having him as an advisor has the greatest impact on my motivations to analyze new ideas not only experimentally, but also in a mathematically rigorous way. This brings a better understanding of the structure of the problem, and usually helps further developing the theory while certainly useful in developing applications as well. I have learned that the symbiosis of engineering work, theoretical analysis, and experimenting forms an excellent base of research. Thank you, Csaba!

The first lecture I took on AI has been Visual Illusions by Péter Aszalós. My inspiration of creating algorithms that aim to solve problems the human visual system does, was born there. Thanks for this enthusiastic lectures! Besides Péter, I am also thankful to András Lőrincz for introducing me to the field of modeling the human visual cortex.

I would like to thank to companies I worked for. They involved me in inspiring engineering tasks, and allowed me to concentrate on their theoretical background as well. I am thankful to Zoltán Horváth, who was the leader of Mindmaker Ltd. and Morita Tsuneo, who is directing Tateyama Hungarian Laboratory Ltd.

My thanks goes to László Balázs leader of GusGus AB Hungarian Branch Office, for teaching me a lot on organizing scattered thoughts nicely. This is always a great help in systemizing my ideas, both in research and programming.

Lastly I can only admire my family. My mother, who always pushed and encouraged me in my Ph.D. work, my father who has been supportive in all aspects of my life, my lovely wife Junko, and kids Noémi and Viola for a lots of things that means family to me.

# Notation

### General

| | |
|---|---|
| $\mathbb{P}(U)$ | The probability of event $U$. |
| $\mathbb{E}(X)$ | The expectation of probability variable $X$. |
| $\mathrm{Var}(X)$ | The variance of probability variable $X$. |
| $x \sim p()$ | $x$ is drawn from density $p(\cdot)$. |
| $w \propto f(x)$ | equals to $cf(x)$, typically $c = (\int f(x)dx)^{-1}$. |
| $F(f)$ | The Fourier transform of $f$. |
| $(f * g)(x)$ | Convolution of $f$ and $g$. |

### Tracking Model

| | |
|---|---|
| $\mathcal{X}$ | State space of tracking. |
| $\mathcal{Y}$ | Observation space. |
| $X_t, Z_t$ | Probability variables (over $\mathcal{X}$); object's state at time $t$. |
| $Y_t$ | Probability variable (over $\mathcal{Y}$); observation signal at time $t$. |
| $r(Y_t = y \mid X_t = x)$ | Observation density, where $x \in \mathcal{X}$. |
| $K(X_t = z \mid X_{t-1} = x)$ | Dynamics ($x, z \in \mathcal{X}$); $\int K(z\|x)dz = 1$. |
| $M = (K, r)$ | A tracking model defined by a dynamical and an observation model. |
| $\pi(X_t = z \mid X_{t-1} = x, Y_{0:t})$ | The (global) importance function used in importance sampling. |
| $q_{X,Y}(Z_t = z)$ | The local importance function at $z$. The function also depends on the latest observation and particle position. |
| $p_M(X_t \mid Y_{0:t})$ | The posterior of $X_t$ given all observations. |

### Particle Filtering

| | |
|---|---|
| $X_t^{(i)}, Z_t^{(i)}$ | $i$-th particle at time $t$. |
| $w_t^{(i)}$ | The weight of the $i$-th particle at time $t$. |
| $S_t = \{(X_t^{(i)}, w_t^{(i)})\}_{i=1}^N$ | The particle representation at time $t$. |
| $N$ | Number of particles. |
| $N_{\text{eff}}$ | Effective sample size. |

**Abbreviations**

| | |
|---|---|
| **CONDENSATION** | Conditional Density Propagation |
| **iCONDENSATION** | Conditional Density Propagation with importance sampling |
| **SIR** | Sequential Importance Resampling |
| **N-IPS** | N-Interacting Particle System |
| **AVM** | Auxiliary Variable Method |
| **AVPF** | Auxiliary Variable Particle Filter |
| **JPDAF** | Joint Probabilistic Data Association Filter |
| **HS-SIR** | SIR with History Sampling |
| **RB-HS-SIR** | Rao-Blackwellised SIR with History Sampling |
| **SS-RB-HS-SIR** | Rao-Blackwellised Subspace SIR with History Sampling |
| **LS-N-IPS** | Local Search N-Interacting Particle System |
| **LLS** | Local Likelihood Sampling |
| **LIS** | Local Importance Sampling |

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

According to the legend "all this" started in the early 1960s when the Hungarian researcher, Rudolf Kalman, visited the NASA Ames Research Center. He realized the applicability of his ideas to the problem of trajectory estimation that came up in the Apollo program. His algorithm [17, 32, 16] soon became the main object tracking principle for many coming years. Applications were soon discovered by engineers in different areas, and the *Kalman filter* became the standard method of choice in designing aircraft autopilots, dynamic positioning system of ships, radar tracking and satellite navigation systems.

In the *trajectory estimation problem* there is an observer equipped with some sensors that gather information about the *state of the object* to be tracked. The sensor can really be anything that provides relevant information on the object. A GPS sensor in navigation and a height sensor in aircraft autopilots are natural choices. The aim that Kalman settled is to use these sensory readings (also called measurements or observations) to the greatest possible extent to give the best estimation of the *object's pose*. The sensors are assumed to supply noisy observations of the environment. A common suggestion is to treat the sensor noise independent in time. The object's state and the sensor readings are connected through the *observation model*. Without any further information, the best estimate of the object's state should be calculated using only the last sensor readings from the observation model. In particular the estimate would be independent of the previous observations. Naturally, this could lead to zigzagging object trajectories, which contradicts our expectations that objects move in a smooth manner. More reasonable trajectories can be obtained by making this expectation part of the model by assuming a *motion model*. As a result one gets smoother object trajectories. In summary, the object model has two parts: the *motion or dynamical model* provides predictions for the object's next state given the previous state, while the *sensor or observation model* describes the relation between the sensory readings and the object's state. The filtering task is to give the best estimate of the object's state assuming that the observations comply with the assumed model. When assuming probabilistic models and an initial distribution for the object's state for the time

step before the first observation arrived, the solution is given by the posterior distribution given the observations. The actual utility of the posterior is limited by the correctness of the models. As an example assume that one receives a sequence of GPS sensory readings with the task of estimating the trajectory of an object moving on the earth in a certain geographical region. Clearly, a simple auto-regressive motion model will lead to estimated trajectories which can be quite different from those obtained by assuming a model that uses road maps. Similarly, different sensor models will certainly lead to different estimates. In the case of GPS sensors, the model of the sensor noise becomes an issue as different noise models lead to different estimates. Hence, one should never forget that the estimates depend not only on the observations, but also the models assumed.

It is reasonable to assume that our brain uses both observation and motion models. As an example, imagine that you would like to catch a flying badminton shuttlecock. The weather is windy and the sun shines into your eyes. How does your brain knows how to move your hands? Due to air turbulence, it would certainly be hard to guess the exact landing position just by watching the moment of throw, and closing your eyes after that. It is harder to imagine, but sounds more obvious that only using your eyes and not your brain would be inefficient for catching the ball. Now, during the play the sun suddenly comes out from behind the clouds, shines into your eyes, reducing the quality of visual information drastically. Our brain helps us through these situations by not only estimating the position of the shuttlecock, but its speed and direction of motion as well, by effectively modeling the motion of the objects tracked.

As we discussed earlier, the dynamical model is meant to describe how the object generally moves from one time step to the next. This implies that the state should be a sufficient statistic for the object's future observations. In other words the state space and the dynamical model should be designed in such a way that the dynamical model is Markovian over the state space. One should prefer low dimensional state spaces, since estimation in high dimensional spaces is harder. Achieving sufficient precision then requires careful engineering. However often the preference to low dimensionality means that many physical effects stay un-modeled (e.g. the spinning of the shuttlecock, or how the wind blows in the above example). A particularly popular choice is to model the dynamics using a low-order auto-regressive process.

Our main interest is to construct algorithms that track the object's state well over time. Kalman filters are designed to work with the so-called linear Gaussian models. For such systems, the posterior is Gaussian whose parameters can be calculated in a recursive (and thus cheap) manner. It is, in fact, this algorithm that is called the Kalman filter. Unfortunately in several important trajectory estimation problems linear Gaussian models are inappropriate. The Extended Kalman Filter [36] and the more recent Unscented Kalman Filter [35, 52] represents one line of research that are designed to work for non-linear and/or non-Gaussian systems. Although these methods extend the scope of Kalman filters considerably, they are fundamentally limited to problems were the posterior is uni-modal. In several problems the available observations are often insufficient to rule out multiple significantly different hypotheses, hence one needs a tool capable of representing such multimodal posteriors. Particle filters are designed to handle such situations.

The development of particle filters goes back to the early 50s, but their intensive study started during the past 10 years. The original idea can be traced back to Metroplis and Ulam [27]. An early summary is given by Rubinstein [11]. The rebirth in the past decade was due to a review paper by Doucet [9] followed by the seminal papers of Kitagawa [18] or Liu [21]. In computer vision an algorithm called CONDENSATION means the starting point [14, 2].

The key idea in particle filtering is to use a discrete non-parametric representation for the posterior, $\sum_{i=1}^{N} w^{(i)}\delta(x - X^{(i)})$, where the weights $w_i$ are non-negative and sum to one and $\delta$ is the Dirac-delta distribution function. This representation is hierarchical in its nature, since the particle positions determine the importance weights as well.[1] It follows hence then that particle filtering algorithms should concentrate mainly on locating the particles to valuable positions. However, most of the particle filtering algorithms determine the particle positions using only the dynamical model and the prior (Bootstrap Filter), or only the most recent observation (likelihood sampling, SIR with observation based importance function). The questions investigated in this Thesis are:

a) What is the best way to let the dynamical model influence the particle locations in SIR when the proposal function depends only on the observation?

b) How to let the latest observation influence the dynamical model based state prediction in the Bootstrap Filter?

In this Thesis the sensor observing the environment will be a video camera. The observation signal is then the image frame. Problems with this setup are called *visual tracking problems*. There are several specialties of this type of tracking problems. An image as a measurement signal of the environment is a complicated one. Extracting position information of objects from images is usually a challenging image processing task. Furthermore since the complicated rules of camera optics and image projection, the object's motion model is usually kept very simple. We will assume that evaluation of the observation likelihood has the highest computational cost. This is why we will try to keep the number of observation density evaluations low.

An important reason why particle filtering is particularly appealing in visual tracking lies in the fact that the the images has to be processed only in the neighborhood of the predicted particle locations. This can be a great advantage compared to other image processing based tracking algorithms, when all the image frames has to be fully scanned for the search of objects. As a result, using particle filtering not only leads to better trajectory estimates, but also a faster algorithms.

At the same time observations coming from image-processing steps are usually more reliable than the state predictions arising from the dynamical model. As a result the observation likelihood function becomes 'peaky' (comparing to the prediction density) or

[1] More exactly, the particle positions determine the expected value of the weights, but in most of the algorithms the weights are fully determined by the particle positions and hence the weights are equal to the above mentioned expected value. An exception besides some of the algorithm introduced in this Thesis is the Auxiliary Variable Method [31], where only the expected value of the weights are determined.

concentrated around its modes (the modes correspond to the states that are locally most likely to 'cause' the past observations). If the position of a particle is not sufficiently close to one of these modes then the corresponding weight will bring in little information on the estimate of the posterior. As an analogue to Bellman's curse of dimensionality, we shall call this problem, particularly eminent in visual tracking, the *"curse of reliable observations"*.

## 1.2 The Filtering Problem

In this section we give a formal description of the filtering problem. From a mathematical viewpoint the problem is to estimate the state of a discrete time, partially observed stochastic system evolving in time. In this Thesis the dynamical model will be defined by a kernel function

$$K(x_t|x_{t-1})$$

which gives the probability density of the object's state $x_t \in \mathcal{X}$ at time $t$ given the object's state $x_{t-1} \in \mathcal{X}$ at time $t-1$. If one has a density of the object's state at time $t-1$, say $p_{t-1}(x_{t-1})$, then the state-density for the next time step can be given as

$$\hat{p}_t(x_t) = \int K(x_t|x_{t-1})p_{t-1}(x_{t-1})dx_{t-1}.$$

At each time step an observation signal is received from the sensors that depends on the object's current state only. Let us denote the observation signal at the $t$-th time step by $Y_t \in \mathcal{Y}$, where $\mathcal{Y}$ is the observation space. The choice of the observation space mainly depends on the sensors observing the system to be tracked. For example, in the case of a satellite navigation system with a GPS it includes the latitude, altitude and longitude. In the case of visual tracking it is the observed image frame. The observation model is required to give a distribution of the observation signal given the object's state. In this Thesis the observation density will be denoted by $r(y_t|x_t)$. It is by assumption that this density is only a function of $x_t$. Hence if one has a density of the object's state at time $x_t$ denoted by $\hat{p}_t(x_t)$ then the density for the observation at the corresponding time step is:

$$\hat{r}_t(y_t) = \int \hat{p}_t(x_t)r(y_t|x_t)d(x_t).$$

Another way of viewing this model is to say that at time $t$ the new state is assumed to be sampled from $K(\cdot|X_{t-1})$, while the new observation signal $Y_t$ is sampled from $r(.|X_t)$. Such models are called Hidden Markov Models [3].

In the literature it is a common practice to denote both the dynamical kernel and the observation density by $p$ and leave it only to the arguments to define the function under consideration. A common further overloading of the symbol $p$ is to use it to denote the prediction and/or posterior densities. The reason we make an effort to avoid this is to make it clear that we work with models of the motion and observation, which do need to match "reality". Whenever appropriate a sub-index $M = (K, r)$ referring to the model will be used to signify when a density depends on the assumed model.

The tracking problem is to compute the *posterior density*:

$$p_M(X_t|Y_{0:t}),$$

where we have used the convention $Y_{0:t} = Y_0, \ldots, Y_t$. By definition the posterior describes the density of the object's state given all available observations. When the model under consideration is unimportant we will omit the index $M$.

## 1.3 Example Tracking Problems

In this section we introduce some vision-based tracking problems to motivate the further developments. These problems will serve as testbeds in the rest of the Thesis.

### 1.3.1 Contour Tracking

The problem is to track an object contour on a video frame. For simplicity the dynamical model assumes that the shape of the object's contour changes only slightly, except for its size and rotation which are free parameters. Figure 1.1 shows some example frames of a video sequence, while an example contour with its B-spline control points is shown in Figure 1.2. We assume that the object does not disappear, and no similar objects enter the scene. The object might however make sudden movements and the scene lighting conditions might change.

#### State Space

The state space consists of the pose of the object to be tracked along with the previous pose. This is because the motion model uses the speed of the object as well. More exactly the motion model is a second order auto regressive model on the object's pose.

A pose defines a contour mapped onto the camera plane (i.e., onto the image) and observations will be defined in terms of contours and the observed images.

A contour is represented by a B-spline [43]. Let us consider the spline curve $s : [0, L] \to \mathbb{R}^2$ defined by its support points (which lie on the curve) $q^x = (q_1^x, \ldots q_n^x)^T, q^y = (q_1^y, \ldots q_n^y)^T$, i.e: $s(t) = ((A^{-1}q^x)^T\varphi(t), (A^{-1}q^y)^T\varphi(t))$, where $\varphi(t) = (\varphi_1(t), \varphi_2(t), \ldots, \varphi_n(t))^T$ are the usual B-spline basis functions (cf. [2]) and $s(i) = (q_i^x, q_i^y)^T$. $A$ is linear transformation mapping control points to support points and depends only on $\varphi$.

Let $q_0 = ((q_0^x)^T, (q_0^y)^T) \in \mathbb{R}^k$ be a vector of support points defining the template contour $s_0 = S(q_0) \subset \mathbb{R}^2$. If $G$ is a group of similarity transformations of the 2D plane ($\mathbb{R}^2$) then one can find a matrix $W = W_{G,q_0}$ such that $T \in G$ iff for some $z \in \mathbb{R}^d, d \geq 1$, the support vector $q = Wz + q_0$ yields the spline curve $Ts_0$, or

$$S(Wz + q_0) = Ts_0.$$

Here we use the convention that $z = 0$ corresponds to $T = \mathrm{Id}$. Thus, if $G$ represents the set of admissible transformations of the contour then the set of admissible object configurations will correspond to $\mathbb{R}^d$.

Figure 1.1: Frames of a contour tracking problem video together with the sample desired results (white contours).

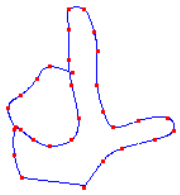

Figure 1.2: An example contour with its B-spline control points.

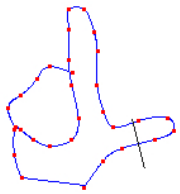

Figure 1.3: An example contour with its B-spline control points and a measurement line(black) used in the observation likelihood calculation.

We will use the group of Euclidean similarities of the plane. Thus in this Thesis $d = 4$ and

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & q_0^x & -q_0^y \\ \mathbf{0} & \mathbf{1} & q_0^y & q_0^x \end{pmatrix}, \tag{1.1}$$

where $\mathbf{0} = (0, 0, \ldots, 0)^T$, $\mathbf{1} = (1, 1, \ldots, 1)^T$ are of the appropriate dimensions.

#### Observation

Assume that a contour ($s$) corresponding to some pose ($z$) is given. The following observation model is due to Blake and Isard [14]. They define the likelihood of the observed image given a pose by assuming that the image is formed such that edge positions along measurement lines perpendicular to the contour follow Gaussian-distribution and edge positions at different measurement lines are independent of each other. Figure 1.3 shows a sample contour with a selected measurement line.

### 1.3.2 License Plate Tracking

In this problem tracking of Japanese license plates is considered. Japanese license plates have a fixed geometrical layout, which is very helpful in constructing a reliable observation model. Figure 1.4 shows some example frames of a video sequence. The cars appear on one side of the image and disappear at the other end. The cars entering the scene must be detected and tracked all the time when they are visible.

#### State Space

The license plate (LP) area can be represented by a parallelogram with two vertical lines and a fixed aspect ratio. Hence the configuration of a LP is defined through four parameters. In the followings we will denote these by $u$, $w$, and $\theta$, where $u$ and $w$ correspond to the horizontal and vertical position of the center of the LP on the image, while $\theta \in [0, \pi) \times [0, +\infty)$ determines the angle of the non-vertical side and the scale of the parallelogram. The state space is twice the size of the pose-space, again assuming a second order dynamics: $x = (u^{\text{now}}, w^{\text{now}}, \theta^{\text{now}}, u^{\text{prev}}, w^{\text{prev}}, \theta^{\text{prev}})$.

Figure 1.4: Frames from a Japanese license plate tracking video.

### Dynamical model

The dynamical model is a mixture of Gaussians. There is a Gaussian that is responsible for representing new cars entering the scene. This Gaussian has a weight of 0.1. Otherwise the state evolves according to a simple AR(2) model. The AR(2) is factored: the middle of the license plate, the size, the orientation and the rate of the parallelogram sides all develop independently of each other.

Strictly speaking, independence does not hold in practice, the choice of this model stems from our desire to minimize the number of free parameters and hence to increase the robustness of the model. The parameters of the AR models were selected by hand in a manner so that the variance of the process noise is overestimated. This was also meant to increase robustness against unexpected movements.

### Observation model

In the followings we will refer to a configuration (or pose) as $(u, w, \theta)$. Now, we define the observation model given a configuration. Japanese license plates enjoy a very restricted geometrical structure, as shown in Figure 1.5: The three main character areas are always positioned at the same place and areas between them remain free of edges. This gives the basic idea of our observation model. The likelihood itself is expressed as the product of three likelihood functions that correspond to three different qualities of the LP. The best way to describe the way these individual likelihoods are computed is to introduce a number of categories and pixel-sets.

Consider the categories and pixel sets listed in Table 1.1. For each specific area the likelihood is determined so that it results in a big value if the image looks 'right' over that given area. When checking if an image looks 'right' we look for the absence or presence of edges and lines over the appropriate areas. For detecting edges we use the Sobel operators.

| **Category** | **Pixel set** | **Sign in Figure.1.5** | |
|---|---|---|---|
| Area of horizontal edges | $P^{h,e}(u, w, \theta)$ | solid line | |
| Horizontal stripes free of edges and lines | $P^{h,fe}(u, w, \theta)$ | dashed line | |
| Area of vertical edges | $P^{v,e}(u, w, \theta)$ | solid line | |
| Vertical stripes free of edges and lines | $P^{v,fe}(u, w, \theta)$ | dashed line | |
| Area of small characters | $P^{sc}(u, w, \theta)$ | dotted | |
| Area of large characters | $P^{bc}(u, w, \theta)$ | checked | |

Table 1.1: *Categories of license plate parts and symbols denoting the corresponding pixel sets.*

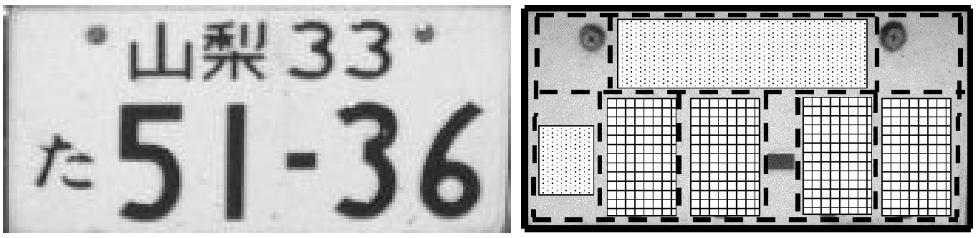


Figure 1.5: A Japanese license plate (left) and its template model (right): The area of large characters is checked, the area of small characters is dotted. The system of horizontal and vertical edge and line-free stripes is marked by dashed lines, whilst stripes of horizontal and vertical lines is marked by solid lines.

The corresponding horizontal (vertical) edge images shall be denoted by $\nabla_x I$ (resp. $\nabla_y I$). For evaluating line features the image was convolved with 2D difference of Gaussians (DOG) filters approximated by the difference of two 2D box-filters having different scales. Two such filters are needed, one with a larger scale (for detecting lines on thick characters) and one with a smaller one (to detect lines of thin characters). The corresponding processed images will be denoted by $G_s I$ and $G_S I$, corresponding to the smaller- and larger-scale lines, respectively.

The likelihood that a given image $y = I$ is generated by a license plate of configuration $(u, w, \theta)$ is thus computed as

$$r(y|u, w, \theta) = r^h(y|u, w, \theta)\, r^v(y|u, w, \theta)\, r^c(y|u, w, \theta),$$

where the likelihoods $r^h, r^v$ and $r^c$ are defined through their logarithms as follows:

$$\begin{aligned}
\log r^h(y|u,w,\theta) \;\propto\; & \frac{1}{\#P^{h,e}(u,w,\theta)} \sum_{p \in P^{h,e}(u,w,\theta)} |(\nabla_x I)(p)| - \\
& \frac{1}{\#P^{h,fe}(u,w,\theta)} \sum_{p \in P^{h,fe}(u,w,\theta)} |(G_s I)(p)| + |(G_S I)(p)| \\
\log r^v(y|u,w,\theta) \;\propto\; & \frac{1}{\#P^{v,e}(u,w,\theta)} \sum_{p \in P^{v,e}(u,w,\theta)} |(\nabla_y I)(p)| - \\
& \frac{1}{\#P^{v,fe}(u,w,\theta)} \sum_{p \in P^{v,fe}(u,w,\theta)} |(G_s I)(p)| + |(G_S I)(p)| \\
\log r^c(y|u,w,\theta) \;\propto\; & \frac{1}{\#P^{sc}(u,w,\theta)} \sum_{p \in P^{sc}(u,w,\theta)} |(G_s I)(p)| + \\
& \frac{1}{\#P^{bc}(u,w,\theta)} \sum_{p \in P^{bc}(u,w,\theta)} |(G_S I)(p)|.
\end{aligned}$$

The symbol $\#\mathcal{P}$ denotes the cardinality of the set $\mathcal{P}$. These various sets have been introduced in Table 1.1.

By inspecting a large number of images we have found that the proposed likelihood function is successful at discriminating plates and clutter: it has a single dominant peak at the location of the true plate or plates. Figure 1.6 shows a typical landscape of the likelihood as a function of $u$ and $w$. Certainly this observation model could be further refined by e.g. learning coefficients for each of the terms.

### 1.3.3 Bearings Only Ship Tracking

In contrary to the previous problems, in this section we will examine a simulated example. This implies that we know the object's exact dynamical and observation model. The problem is a standard one that has been considered previously by several authors [13, 31, 4, 1]. The aim is to track the (horizontal) motion of a ship, while observing only angles

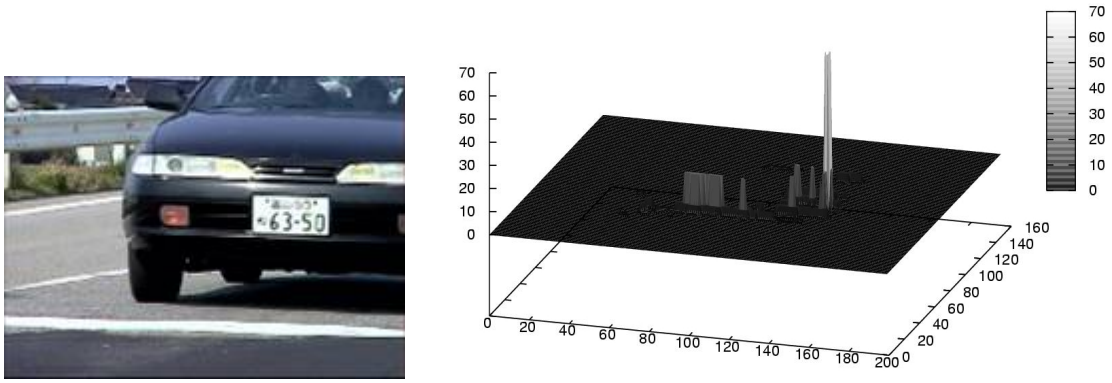


Figure 1.6: The observation likelihood (right) function for the frame shown (left). Note that the highest peak has a pretty small support, i.e. the observation likelihood function is sensitive to the precise positioning of the license plate. This is a desirable property if the goal is to determine the license plate's position with high accuracy. The scale and the orientation are selected to match their corresponding true values.

to it. We assume that the coordinate system is fixed to the observer. The ship's state is assumed to follow a second order AR process, with its acceleration driven by white noise:

$$X_{t+1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} X_t + \begin{pmatrix} \frac{1}{2} & 0 \\ 1 & 0 \\ 0 & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \xi_t,$$

where $X_t$, $\xi_t \in \mathbb{R}^2$, $\xi_{t1}, \xi_{t2} \sim \mathcal{N}(0,1)$, and $X_{t1}, X_{t3}$ represent the ship's respective vertical and horizontal positions, whilst $X_{t2}, X_{t4}$ represent the ship's vertical and horizontal velocities. The initial state is sampled from a 4-dimensional Gaussian with a diagonal covariance matrix.

What makes this problem particularly challenging is that the observations depend on the state only through the angle, $\theta_t = \tan^{-1}(X_{t3}/X_{t1})$, at which the ship is observed. The observation noise is defined by a "wrapped" Cauchy density:

$$r(y|\theta_t) = \frac{1}{2\pi} \frac{1-\rho^2}{(1+\rho^2-2\rho\cos(y-\theta_t))}.$$

This density (when $\rho$ is close to one) is thought to reflect well a sonar's behavior: angle measurements are typically very reliable, whilst possible outliers are well modeled by the heavy tails of the wrapped Cauchy distribution.

The parameters of the model used in the experiments are as follows: $\sigma_\eta = 0.001$, $\rho = 1 - 0.005^2$, and the initial state is sampled from a Gaussian with means $(-0.05, 0.001, 0.2, -0.055)$ and with a diagonal covariance matrix with diagonal entries given by

$$0.001 \times (0.5^2, 0.005^2, 0.3^2, 0.01^2).$$

Figure 1.7 gives an example of the ship's motion and the observed angles.

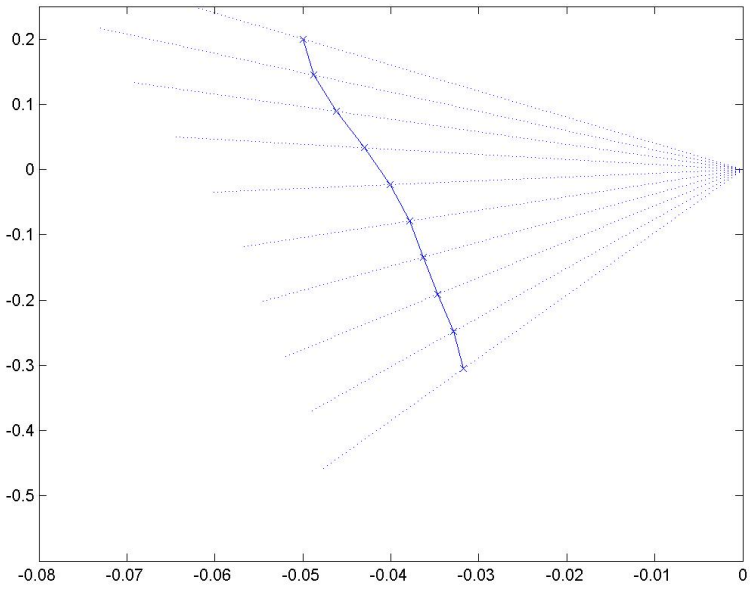


Figure 1.7: An example of the ship's motion in the bearings only tracking problem together with some rays representing the observations.

## 1.4 Thesis Overview

Let us now give an overview of the Thesis: In the next Chapter we first derive the exact solution to the filtering problem. Then we discuss the basic building blocks of particle filtering. Issues and important concepts are discussed, including resampling, importance sampling and the effective sample size.

The main motivation of this Thesis is to show how the observation model can be better used in the particle localization stage of particle filtering. The obvious way of incorporating information coming from the observation is importance sampling. However it is common that the last observation does not hold information for all the components of the state space. For instance observing only the pose of the object might bring no information in view about the speed or the direction of motion. The straightforward importance sampling scheme then might result in poor approximations, since the likelihood of most of the particles might get very small as the history part of the new state is not associated with the observation based innovation content. In Chapter 3 we propose a solution that overcomes the problem. The proposed family of algorithms samples the particle's history hence the algorithms are called history-sampling based particle filters.

Chapter 4 describes a more complicated but an often more effective method to locate the particles that uses both the observation and the dynamical model. They key idea lies in a modification of the Bootstrap Filter, whereas the particles are re-allocated after they

are predicted using the dynamical model. This step, that we call the position perturbation step, is done by locally analyzing the observation likelihood around each predicted location and let the particles be dragged by the local modes of the observation likelihood. One can think of this idea as a particle localization procedure that uses the global behavior of the dynamical model combined with the local characteristics of the observation model to locate the particles. This combination is particularly appealing since in visual tracking the dynamical model is less reliable, and hence it is able to give valuable particle location regions only. In this sense the dynamical model gives good particle locations in a "low frequency" manner. In contrary the much more reliable observation gives focus to spots of good particle locations, providing them in a "high-frequency" way. It is shown that these methods increase the efficiency of particle filtering in case of reliable observation models. The text focuses on the theory of the algorithms, but experiments on the above tracking problems are also given. Although the ideas are explained in a general framework, this chapter is highly motivated by visual object tracking situations.

In the Appendix several related advanced particle filtering algorithms are discussed using the notation of this Thesis. The reason that these algorithms are not detailed in the main body, is to keep the body focused on the new ideas.

# Chapter 2

# Particle Filtering Elements

## 2.1 The Bayesian Solution

The filtering problem formulated in Section 1.2 has an analytic solution that can be obtained by applying Bayes theorem, the law of total probability and exploiting the following independence assumptions:

(i) The observation signal given the last state is independent of the previous observation signals:

$$p_M(y_t|x_t, y_{0:t-1}) = p_M(y_t|x_t).$$

(ii) The state is independent of the previous observation signals, if the previous state is given:

$$p_M(x_t|x_{t-1}, y_{0:t-1}) = p_M(x_t|x_{t-1}).$$

Then the posterior can be calculated, up to a proportional factor, as follows:

$$\begin{aligned} p_M(x_t|Y_{0:t}) &= \frac{p_M(Y_t|x_t)p_M(x_t|Y_{0:t-1})}{p_M(y_t|Y_{0:t-1})} \\ &\propto p_M(Y_t|x_t)\int p_M(x_t|x_{t-1})p_M(x_{t-1}|Y_{0:t-1})dx_{t-1} \\ &= r(Y_t|x_t)\int K(x_t|x_{t-1})p_M(x_{t-1}|Y_{0:t-1})dx_{t-1}. \end{aligned} \tag{2.1}$$

Note that this is a recursive equation for $p_M(x_t|Y_{0:t})$. Thus, when the initial state density $p_0(x_0)$ is known, $p_M(x_t|Y_{0:t})$ can be computed via (2.1).

Unfortunately, the exact solution cannot be computed except in a few special cases such as the case of linear Gaussian models or finite Hidden Markov models (in the former case the solution is given by the Kalman filter, while in the latter case it is given by the Baum-Welsch equations). In practice the models are often more complicated. Therefore a large body of current work in the filtering literature is devoted to finding the posterior in an approximate manner.

## 2.2 Introducing Particle Representations

As it was noted above, the exact solution of the filter evolution equation (2.1) is not feasible in general. One approach to overcome this problem is to use sequential Monte-Carlo methods to sample from the posterior. The key issue here is to construct a weighted sample set that represents the posterior in a faithful manner. One approach is to target unbiased representations:

**Definition 2.2.1.** *A random variable $X$ is said to be* properly weighted *by the function $w$ with respect to the density $p$ and the integrable function $r$ if for any integrable function $h$,*

$$\mathbb{E}[h(X)w(X)] = I(hrp),$$

*where $I(f) \int f(x)\,dx$ is the usual integral operator. Alternatively, we say that $(X, w(X))$ forms a* properly weighted pair *with respect to $p, r$.*

*A series of random draws and weights $\{(X^{(k)}, w^{(k)})\}_{k=1,\ldots,N}$ is said to be* properly weighted *with respect to $p$ and $r$ if all members of the set are properly weighted with respect to $p$ and $r$.*

If $\{(X^{(k)}, w^{(k)})\}_{k=1,\ldots,N}$ are independent properly weighted samples with respect to $p$ and $r$ then by the law of large numbers the sample averages

$$J_N(h, w) = (1/N) \sum_{j=k}^{N} h(X^{(k)})w(X^{(k)})$$

will converge to $I(hrp)$ if it exists. Hence, in this sense, a properly weighted set with respect to $f$ and $r$ can be thought of as representing the product $r(\cdot)p(\cdot)$. In this context $h$ is an integrable test-function. Its form certainly affects the quality of the estimation, and hence the speed of convergence of $J_N(h, w)$ to $I(hrp)$.

Furthermore, if one is given a properly weighted sample-set $(X_t^{(i)}, w_t^{(i)})$, $i = 1, 2, \ldots, N$ from the posterior $p_M(X_{t-1}|Y_{0:t-1})$ then an unbiased sample from $p_M(X_t|Y_{0:t})$ can be generated by the following two-step method [9]:

1. **Prediction:** Draw $X_t^{(i)}$ from $K(X_t = \cdot|X_{t-1}^{(i)})$, $i = 1, 2, \ldots, N$,

2. **Evaluation:** $w_t^{(i)} = c_t w_{t-1}^{(i)} r(Y_t|X_t^{(i)})$, $i = 1, 2, \ldots, N$.

Here the unknown proportional factor $c_t$ is inversely proportional the the probability of the latest observation given the previous ones, i.e., $c_t = 1/p_M(Y_t|Y_{0:t-1})$. Note that $c_t$ depends only on the observations and does not depend on the particles' properties.

Elements of the sample $(X_t^{(i)})$ are traditionally called particles and the filtering method is called particle filtering [9].

## 2.3 Importance Sampling

Importance sampling is an important Monte-Carlo estimation procedure. It suggests to estimate $\mu = E_p(h(X))$ by

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} h(X_i)\frac{p(X_i)}{\pi(X_i)},$$

where $X_1, \ldots, X_N$ are independent samples from a distribution $\pi(\cdot)$ satisfying that whenever $p(x) > 0$ then also $\pi(x) > 0$.

The choice of $\pi$ certainly influences the efficiency of the sampling to a great extent. A good choice of $\pi$ is one that is close to $|h(x)p(x)|$. When $\pi \propto |hp|$ the variance becomes zero, and $\frac{h(X_i)p(X_i)}{\pi(X_i)} = \mu$. If sampling from $p(.)$ directly is difficult but generating from $\pi$ and computing the importance ratio $w(X) = p(X)/\pi(X)$ are easy, then using importance sampling might lead to an effective estimation procedure.

A special choice of $\pi$ is $p$ itself. In this case the importance sampling scheme is identical to the simple Monte-Carlo integral estimation.

Another useful variation of this idea is called weighted importance sampling. In this procedure we estimate $\mu$ by

$$\overline{\mu} = \frac{\sum_{i=1}^{N} h(X_i)w(X_i)}{\sum_{i=1}^{N} w(X_i)}.$$

This estimate is only asymptotically unbiased. This is easy to see by considering only one particle $N = 1$. The estimate becomes $h(X_i)$ independently of $p$, which is obviously biased in general. In contrary the simple importance sampling gives unbiased estimate even for a single particle. However in other performance measures such as $\mathbb{E}\left[(\overline{\mu} - \mu)^2\right)$ weighted importance sampling often outperforms importance sampling. Further note that this estimation works if one knows $p$ only proportionally.

## 2.4 Effective Sample Size

The asymptotic consistency of a Monte-Carlo procedure represents only a minimal requirement that does not tell anything about the *quality* of the estimates. One way to measure the quality of a randomized estimate is to compute its variance: in our specific case one goal can be to obtain a properly weighted set such that the variance of the sample average $J_N(h, w)$ is minimized (or is small).

Actually, we are more interested in studying the weight-normalized averages

$$I_N(h, w) = \frac{\sum_{j=1}^{N} h(X^{(j)})w(X^{(k)})}{\sum_{j=1}^{N} w(X^{(j)})} \tag{2.2}$$

which, again under proper conditions, converge to the normalized value $I(hrp)/I(rp)$ as $N \to \infty$ with probability one. As mentioned before, in practise $I_N(h, w)$ is more preferred then $J_N(h, w)$, since its variance is smaller. On the other hand it is true that $I_N(h, w)$ is

biased for any finite $N$. Particle filters make use of these weight-normalized averages to represent the empirical measure. In this way they do not have to estimate the normalization constant $c_t$, which is a significant advantage.

Obviously, the variance of the sample average depends on $h$. Certainly we are interested in the case when $h$ is not fixed. Liu [20] (based on [19]) shows that it is sensible to measure efficiency by a quantity inversely proportional to the variance of $w(X)$, provided that $w$ is such that $E[w(X)] = 1$. We will refer to this scheme as Liu's 'rule of thumb'. In turn this leads to a measure of efficiency

$$N_{\text{eff}} = \frac{N}{1 + Var(w(X))},$$

which is called the effective sample size. If $N_{\text{eff}} = N$, i.e, all weights are equal, then the optimal sampling strategy is implemented. In other words the particle locations are sampled exactly from the product $r(\cdot)p(\cdot)$.

Liu's rule of thumb will serve as our starting point. In particular, since for any properly weighted pair $(X, w)$, $E[w(X)] = I(rp)$ and $\text{Var}[w(X)] = E[w^2(X)] - E[w(X)]^2$ it follows that minimizing $\text{Var}[w(X)]$ is equivalent to minimizing $E[w^2(X)]$. Therefore, it follows that it is sufficient to compare unbiased algorithms on the basis of the second moment of the weights they use, since it is the same as comparing $N_{\text{eff}}$.

Note that if $X$ is drawn from $p$ and $w$ is set to be equal to $r$, as in the case of the sampling procedure underlying the two-step method of Section 2.2, then

$$E[w^2(X)] = I(r^2 p). \tag{2.3}$$

An empirical measure of the effective sample size follows from the fact that the estimate of the variance can be given as

$$Var(w(X)) = \frac{\sum_{i=1}^{N} (w^{(i)})^2}{N} - 1.$$

Hence:

$$\hat{N}_{\text{eff}} = \frac{N}{\mathbb{E}(w^2)} = \frac{1}{\sum_{i=1}^{N} (w^{(i)})^2}.$$

## 2.5 Resampling

The serious problem with the two-step method of Section 2.2 is that given any finite sample size the weights $w_t^{(i)}$ will degenerate: in general all of them converge to 0 except for one index, for which $w_t^{(i)}$ will converge to 1. This means that the observation model will have no effect on the sample trajectories, which hence becomes a simple random walk with the dynamical kernel $K$. In order to prevent this degeneration, a resampling step was introduced into the above procedure by Gordon et al. and Rubin [13, 34]. Given a particle set $\left\{(X^{(k)}, w^{(k)})\right\}_{i=1}^{N}$ the aim of resampling is to generate a new particle set

$\left\{(\hat{X}^{(k)}, \hat{w}^{(k)})\right\}_{i=1}^{N}$ with lower weight variance, most often equal weights. Since the only input of this method is the original particle set, the new particle positions are simply sampled from among the original ones. Certainly the new particle set should still be an unbiased estimator of the distribution in question. This can be ensured by considering random weights satisfying

$$E\left(\sum_{j:\hat{X}^{(j)}=X^{(j)}} \hat{w}^{(j)}\right) = Nw^{(j)}$$

We will mainly consider algorithms that generates equal weights, i.e: $\hat{w}^{(j)} = 1/N \quad j = 1, \ldots, N$.

The simplest resampling algorithm called Multinomial resampling is shown in Figure 2.1.

**For** $n = 1, 2, \ldots, N$

**Sample** $N_i$ from the multinomial distribution $Mult(N; w^{(1)}, w^{(2)}, \ldots, w^{(N)})$.

**Let** $\hat{X}^{(i)} = X^{(N_i)}$ and $\hat{w}^{(i)} = 1$.

**EndFor**

Figure 2.1: *The Multinomial Resampling.* $N$ is the number of particles and $i = 1, 2, \ldots, N$ is a particle index.

It is easy to imagine that the resampling step made regularly solves the above mentioned degeneracy problem of particle filters, by always keeping the majority of the samples in valuable location. It is important to notice that resampling gave an important boost to sequential Monte-Carlo methods, and made the two-step procedure of Section 2.2 a practical algorithm [13]. However nothing comes for free. As it is shown in the next section resampling increases sampling variance.

### 2.5.1 Degradation due to Resampling

The following statements and proofs are mainly the work of Móri Tamás (personal communication).

Imagine we have $N$ particles all of them with equal weight $(1/N)$. Assume that not all of the particles are unique, but we only have $m \leq N$ distinct ones. Let their respective duplication numbers be $N^1, \ldots N^m$. Let us define the duplication weights of the $m$ distinct particles as $W_1^0 = N^1/N, W_2^0 = N^2/N, \ldots, W_m^0 = N^m/N$ $(\sum_{i=1}^{m} W_i^0 = 1)$. Assume we recursively resample the particle set with multinomial resamplings for a couple of times.

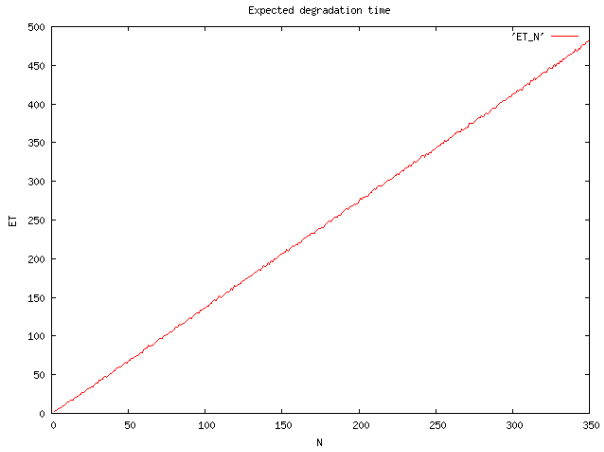


Figure 2.2: The expected degradation time of resampling as a function of particle number.

Let $W_i = (W_1^i, W_2^i, \ldots, W_m^i)^T$ be the duplication weights of the particles after the $i$-th resampling. It is clear that $NW_j^{i+1}$ is drawn from an $N$-th order binomial distribution with parameter $W_j^i$, i.e:

$$\mathbb{P}(NW_j^{i+1} = k|W_i) = \binom{N}{k}(W_j^i)^k(1 - W_j^i)^{N-k}$$

Let $T_n$ be the time of reaching a completely degenerated particle set (meaning that there is a single particle having all the weights), i.e.: $T_N = \inf\{k| \sum_{j=1}^N W_j^k(1 - W_j^i) = 0\}$. In Appendix A.1 it is shown that $\mathbb{E}T_N \leq 2N \log 2N$.

In fact Monte-Carlo experiments suggests that $\mathbb{E}T_N$ is even linear in $N$. Results of Monte Carlo experiments when $m = 2$ are shown in Figure 2.2.

Overall this suggests that degradation of a constant particle set with repeated multinomial resampling is very fast. Hence this procedure should be used with extra care. It should also be clear that it is a good idea to decrease the variance of the resampling scheme. Some popular low variance resampling methods are given in Appendix A. The paper [33] is an excellent summary of this issue.

## 2.6 The Bootstrap Filter

The Bootstrap Filter, also known as CONDENSATION [14] in the visual tracking literature, is the simple combination of the two-step method of Section 2.2 and resampling. The algorithm is shown in Figure 2.3. A visual representation is given in Figure 2.4.

**Initialize** $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**For** $k = 1, 2, \ldots, N$

**Resample** $\left[(Z_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ to obtain a new sample $\left[(Z_{t-1}^{(k)\prime}, 1/N)\right]_{j=1}^{N}$.

Sample $Z_t^{(k)} \sim K(.|Z_{t-1}^{(k)\prime})$.

Let $\hat{w}_t^{(k)} = r(y_t|Z_t^{(k)})$.

**EndFor**

**Normalize** weights using

$$w_t^{(k)} = \frac{\hat{w}_t^{(k)}}{\sum_{i=1}^{N} \hat{w}_t^{(i)}}$$

**EndFor**

Figure 2.3: *The Bootstrap Filter.* $N$ is the number of particles and $i = 1, 2, \ldots, N$ is a particle index.

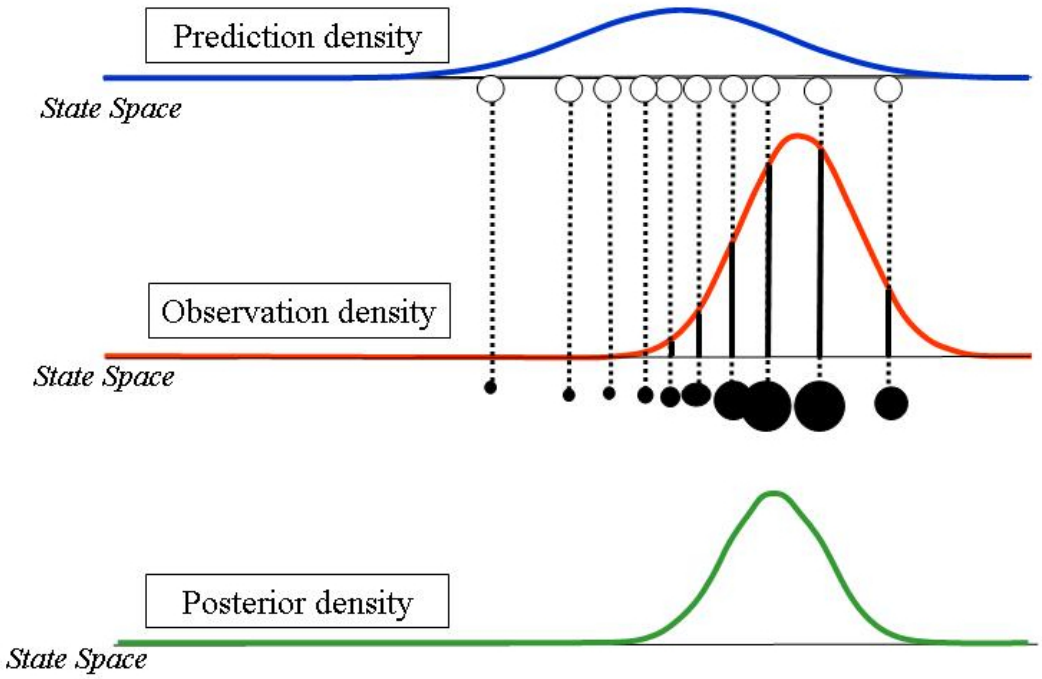


Figure 2.4: Sampling in the Bootstrap Filter. Positions of the particles (represented by white circles in the upper part) are drawn from the prediction density. The particles are propagated through the observation likelihood density to produce a weighted sample. The radii of the black discs are plotted in proportion to the size of the weight corresponding to the respective particles. The lower subplot shows the product of the prediction and observation. Ideally, the 'particle cloud' composed of the black discs should closely reflect the shape of the posterior.

# Chapter 3

# Importance Sampling in Particle Filtering

## 3.1 Sequential Importance Resampling (SIR)

Using importance sampling in particle filtering goes back to [34, 40]. For some reason one wants to sample the proposed the next states from a so-called *proposal density* $\pi(X_t = \cdot|X_{t-1}, Y_t)$ instead of $K(X_t = \cdot|X_{t-1})$. The only restriction on this density is that $\pi(x_t|x_{t-1}, Y_t) > 0$ whenever $K(x_t|x_{t-1}) > 0$.

The derivation of the algorithm is again simple:

$$p(X_t|y_{0:t}) \propto p(y_t|X_t) \int \pi(X_t|x_{t-1}, y_t) \frac{p(X_t|x_{t-1})}{\pi(X_t|x_{t-1}, y_t)} p(x_{t-1}|y_{0:t-1}) dx_{t-1}$$

The simplest importance sampling based particle filtering algorithm, the so called Sequential Importance Sampling/Resampling (SIR) algorithm, is shown in Figure 3.1.

Obviously, the choice of $\pi$ effects the efficiency of the algorithm in a fundamental way. As in the general case, the best is certainly if $\pi$ is close to the posterior itself. Note that the Bootstrap Filter itself is an importance sampling algorithm, with $\pi(x_t|x_{t-1}, Y_t) = K(x_t|x_{t-1})$, i.e: the proposal does not depend on the latest observation signal. On the other hand in visual object tracking it is common to choose proposals that depend only on the last observation, and not on the previous state, i.e, $\pi(x_t|x_{t-1}, Y_t) = \pi(x_t|Y_t)$. In these cases the proposal distribution is computed via some rough and fast image processing steps e.g: color blob detection or edge filtering, which proposes particle positions with possibly high observation likelihoods.

In some cases the proposal is not well defined on all dimensions of the state space. This happens, e.g., if the state space holds some historical information on the particles, or the image processing involved in the proposal function is not defined on all the dimensions of the object's pose, e.g. the color blob detector is usually not defined in the rotation component. In these cases the the proposed particle positions are not fully defined and hence the algorithm described in Figure.3.1 gets badly degraded. In this chapter we will discuss the particle filtering algorithms introduced in [48] that overcome these problems.

**Initialize** $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(X_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(\hat{X}_{t-1}^{(j)}, \hat{w}_{t-1}^{(j)})\right]_{j=1}^{N}$.

**For** $k = 1, 2, \ldots, N$

**Sample from the proposal**

$$X_t^{(k)} \sim \pi(.|\hat{X}_{t-1}^{(k)}, Y_t).$$

**Calculate the importance weight**

$$\overline{w}_t^{(k)} = \hat{w}_{t-1}^{(k)} r(Y_t|X_t^{(k)}) \frac{K(X_t^{(k)}|\hat{X}_{t-1}^{(k)})}{\pi(X_t^{(k)}|\hat{X}_{t-1}^{(k)}, Y_t)}.$$

**EndFor**

**Normalize weights** using

$$w_t^{(j)} = \frac{\overline{w}_t^{(j)}}{\sum_{i=1}^{N} \overline{w}_t^{(i)}}$$

**EndFor**

Figure 3.1: The Sequential Importance Sampling/Resampling (SIR) algorithm.

## 3.2 SIR with History Sampling

### 3.2.1 Problem Setting

In this section we will assume that the proposal density and the dynamics have a special form and that the cost of evaluating the likelihood function is high. Problems with this characteristics commonly arise in vision based tracking as we shall see it later. For now, let us consider our assumptions in more details.

**Assumption 3.2.1.** *"Factored dynamical model" We shall assume that the state space $\mathcal{X}$ is factored into two parts:*

$$\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \tag{3.1}$$

*where $\mathcal{X}_1$ is of dimension $n_1$ and $\mathcal{X}_2$ is of dimension $n_2$ ($n_1 + n_2 = n$, $n_1, n_2 \geq 1$). The dynamical model is given by a stochastic kernel*

$$K_1(X_{t,1} = \cdot|X_{t-1})$$

*defining the distribution over the first subspace and a deterministic function $f$ defining the evolution in the second subspace, i.e:*

$$X_{t,2} = f(X_{t-1}).$$

We shall call $X_{t,2}$ the *history* part of the state consisting of the previous configurations, whilst we call $X_{t,1}$ the *innovation* part of the state. As a particular example of a process of this kind let us consider auto-regressive (AR) processes. Remember that in the case of a $k$-dimensional order-$p$ AR process the dynamics is given as follows: The state $X_t$ is factored into $p$ parts: $X_t = ((Z_t)_0^T, \ldots, (Z_t)_{p-1}^T)^T$, where $X_t \in \mathbb{R}^{kp}$ and $(Z_t)_j \in \mathbb{R}^k$. Now, $X_{t+1}$ is given by

$$\begin{aligned} (Z_{t+1})_0 &= \sum_{j=0}^{p-1} A_j \cdot (Z_t)_j + e_{t+1}, \quad \text{and} \\ (Z_{t+1})_{j+1} &= (Z_t)_j, \quad j = 0, \ldots, p-2. \end{aligned} \tag{3.2}$$

Here $A_0, \ldots, A_{p-1} \in \mathbb{R}^{k \times k}$ are parameters of the process, and $e_0, e_1, \ldots$ is a series of independent, identically distributed zero-mean $k$-dimensional Gaussian random variables. The dynamics can be transformed into the form of Assumption 3.2.1 by defining $n_1 = k$, $n_2 = k(p-1)$, $X_{t,1} = (Z_t)_0$ and $X_{t,2} = ((Z_t)_1^T, \ldots, (Z_t)_{p-1}^T)^T$.

In visual tracking second order dynamical models are the most common choice. In these cases the pose of the object can be considered as the innovation component of the state space, while the remaining part describing the velocity is the history component.

**Assumption 3.2.2.** *"Restricted proposal" According to this assumption, the proposal $\pi$ depends only on the last observation signal and is defined only for the innovation component of the state. Therefore in what follows we shall write $\pi$ in the form $\pi(X_{t,1}|Y_t)$.*

In order to simplify the exposition we shall further assume the following:

**Assumption 3.2.3.** *"The observation density depends only on $X_{t,1}$, the innovation part of the state." According to this assumption one can write $r(Y_t|X_t) = r(Y_t|X_{t,1})$.*

Assumptions 3.2.1, 3.2.2, 3.2.3 are often satisfied when particle filters are used in visual tracking. First, the dynamics of the object to be tracked is often represented by some AR process (satisfying Assumption 3.2.1). One example when the other two assumptions are also valid is when a color blob detector is used as a proposal density for contour tracking(e.g. see [15]). Note that in the case of visual tracking, according to Assumption 3.2.2 the states proposed by $\pi$ will depend only on information derived using the images (a "bottom up" approach).

Under Assumptions 3.2.1, 3.2.2, 3.2.3 algorithm SIR takes the form presented in Figure 3.2. In what follows we shall call this algorithm "Basic-SIR". It is clear that the particle set $X_t^{(i)}$ updated using Basic-SIR gives an unbiased estimate of the posterior.[1]

[1]By unbiased estimate we mean asymptotically unbiased. For finite sample sizes the estimate given by particle filtering algorithms is biased.

Unfortunately, Basic-SIR can be very inefficient. It may require a large number of particles to achieve even a modest precision. This is because many weights can get pretty small at the same time, since the innovation $X_{t,1}^{(i)}$ that will be associated with particle $i$ at time $t$ is sampled *independently* of the state $(X_{t-1}^{(i)})$ associated with that particle. Therefore, with high probability, the value of $K_1(\hat{X}_t^{(i)}|X_{t-1}^{(i)})$ will be small when e.g. the density $K_1(X_t^{(i)}|X_{t-1}^{(i)})$ is concentrated to a small portion of the state space. This happens e.g. when the variance of the system noise $K_1$ (cf. Assumption 3.2.1) is small. This problem is illustrated in Figure 3.3. In this example the resulting particle set failed to represent both peaks of the posterior density due to the random association of innovations and histories. A common method to overcome this shortcoming is to increase the variance of the observation model. Hence we will seek more principled solutions that have potential of achieving better quality at the price of some additional work. Note that the computational example used by Isard and Blake to illustrate their ICondensation algorithm [15] satisfies all of our assumptions and is hence subject to problems described above.

The idea of the algorithms we consider in the followings is to ensure that for each particle the history component of the particle will match the innovation component sampled from the proposal. We achieve this by drawing an appropriate history for each innovation component.

### 3.2.2 SIR with History Sampling

The main loop of our first algorithm, called HS-SIR (SIR with History Sampling), is shown in Figure 3.4.

In order to understand this algorithm, let us introduce the auxiliary variables $(X_t^{(i,j)}, w_t^{(i,j)})$ such that $(X_{t-1}^{(i,j)}, w_{t-1}^{(i,j)}) = (X_{t-1}^{(i)}, w_{t-1}^{(i)})$ and let a particle set at time $t$ be defined by the equations

$$X_t^{(i,j)} = \begin{pmatrix} X_{t,1}^{(j)} \\ f_2(X_{t-1}^{(i)}) \end{pmatrix}, \tag{3.3}$$

$$w_t^{(i,j)} = w_{t-1}^{(i,j)} \frac{r(Y_t|X_{t,1}^{(i,j)}) K_1(X_{t,1}^{(i,j)}|X_{t-1}^{(i,j)})}{\pi(X_{t,1}^{(j)}|Y_t)} = w_{t-1}^{(i)} \frac{r(Y_t|X_{t,1}^{(j)}) K_1(X_{t,1}^{(j)}|X_{t-1}^{(i)})}{\pi(X_{t,1}^{(j)}|Y_t)}. \tag{3.4}$$

The particle $\left(X_t^{(i,j)}, w_t^{(i,j)}\right)$ associates the $i$-th history with the $j$-th innovation. Here the last equation follows by our assumptions on the observation and proposal densities. Now assume that at time $t-1$ the particle set $(X_{t-1}^{(i)}, w_{t-1}^{(i)})_{i=1}^N$ represents an unbiased estimate of the posterior $p(X_{t-1}|Y_{0:t-1})$. Clearly, by the unbiasedness of the basic importance sampling scheme, the particle set $(X_t^{(i,j)}, w_t^{(i,j)})_{i,j=1}^N$ will represent an unbiased estimate of the

**Initialize** $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(X_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(\hat{X}_{t-1}^{(j)}, \hat{w}_{t-1}^{(j)}\right]_{j=1}^{N}$.

**For** $k = 1, 2, \ldots, N$

**Sample innovation** from the proposal $X_{t,1}^{(k)} \sim \pi(\cdot|Y_t)$ and let

$$X_t^{(k)} = \begin{pmatrix} X_{t,1}^{(k)} \\ f_2(X_{t-1}^{(k)}) \end{pmatrix}$$

.

**Calculate the importance weight**

$$\overline{w}_t^{(k)} = \hat{w}_{t-1}^{(j)} \frac{r(Y_t|X_{t,1}^{(k)}) K_1(X_{t,1}^{(k)}|\hat{X}_{t-1}^{(k)})}{\pi(X_{t,1}^{(k)}|Y_t)}$$

**EndFor**

**Normalize weights** using

$$w_t^{(j)} = \frac{\overline{w}_t^{(j)}}{\sum_{i=1}^{N} \overline{w}_t^{(i)}}$$

**EndFor**

Figure 3.2: The Basic-SIR algorithm with our assumptions 3.2.1, 3.2.2, and 3.2.3.

posterior $p(X_t|Y_{0:t})$. Now, if $I_t^{(i)}$ and $J_t^{(i)}$ are the random indexes drawn in HS-SIR, then

$$p(I_t^{(i)} = l, J_t^{(i)} = k) = p(I_t^{(i)} = l|J_t^{(i)} = k) \cdot p(J_t^{(i)} = k) =$$

$$= \frac{w_{t-1}^{(l)} K_1(X_{t,1}^{(k)}|X_{t-1}^{(l)})}{\sum_{j=1}^{N} w_{t-1}^{(j)} K_1(X_{t,1}^{(k)}|X_{t-1}^{(j)})} \frac{\frac{r(Y_t|X_{t,1}^{(k)})}{\pi(X_{t,1}^{(k)}|Y_t)} \sum_{j=1}^{N} w_{t-1}^{(j)} K_1(X_{t,1}^{(k)}|X_{t-1}^{(j)})}{\sum_{n=1}^{N} \sum_{j=1}^{N} \frac{r(Y_t|X_{t,1}^{(n)})}{\pi(X_{t,1}^{(n)}|Y_t)} w_{t-1}^{(j)} K_1(X_{t,1}^{(n)}|X_{t-1}^{(j)})} \propto w_t^{(l,k)}.$$

Hence the probability that $I_t^{(i)} = k$ and $J_t^{(i)} = l$ both hold is proportional to the weight of particle $X_t^{(k,l)}$. Therefore sampling $J_t^{(i)}$ and $I_{t+1}^{(i)}$ in HS-SIR takes the form of a standard resampling step for the particle set $(X_t^{(i,j)}, w_t^{(i,j)})$, and therefore the resulting particle set of the HS-SIR algorithm $(X_t^{(i)}, 1/N)_{i=1}^{N}$ will represent an unbiased estimate of the posterior. Actually, these steps of the above algorithm can be considered as sampling from

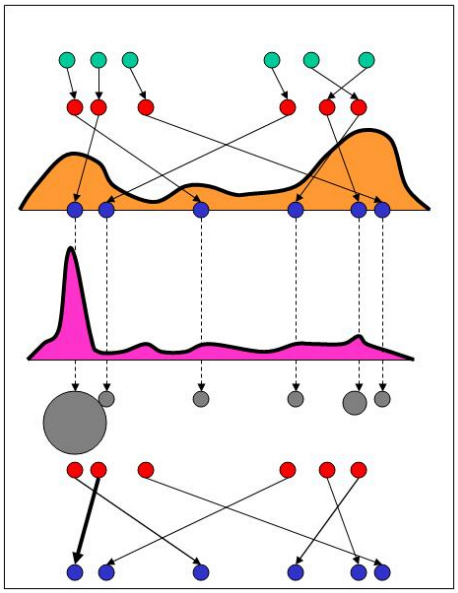

Figure 3.3: *Illustration of the behavior of Basic-SIR.* Consider a system where the state $X_t = (X_{t,1}, X_{t,2})$ evolves according to a one-dimensional first-order AR-model. The first two rows of the figure represent the particle set at time $t-1$, where the individual particles are identified by the arrows connecting $X_{t-1,2}$ (green) and $X_{t-1,1}$ (red). The next row shows the proposal density (orange) and the innovations ($X_{t,1}^{(i)}$) (blue) drawn from it. The arrows from $X_{t-1,1}$ to $X_{t,1}$ show the association of the randomly sampled innovations and the particles. In the lower part of the figure the new particle set is depicted after proper weighting using the observation likelihood (pink) and the proposal (orange). Weights of the individual particles are represented by the strength of the respective arrows.

$(\ldots, w_t^{(\cdot,\cdot)}, \ldots)$ by means of factored sampling (see Appendix C): Drawing $J_{t+1}^{(i)}$ samples the innovation components, whilst drawing $I_{t+1}^{(i)}$ samples the appropriate histories to be associated with these components.

The advantage of HS-SIR over Basic-SIR should be clear by now: HS-SIR selects (by random sampling) pairs of innovations and histories that have high probability of co-occurring and thereby it will in general reduce the variance of the estimate of the posterior. Note that this algorithm does resampling in every step, in contrary to other algorithms where resampling is done only if the effective sample size drops too low. Although this is disadvantageous the effect that history and the innovation part of the state space does not decouple might be more important. Also note that the low variance resampling methods discussed in Appendix A) can be adjusted easily to sample the indices $J_t^{(i)}$, and $I_t^{(i)}$.

**Initialize** $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Sample all innovations** from the proposal $X_{t,1}^{(j)} \sim \pi(\cdot|Y_t)$ , $j = 1, 2, \ldots, N$.

**For** $i = 1, 2, \ldots, N$

**Sample innovation index** $J_t^{(i)}$ with law

$$p(J_t^{(i)} = k) \propto \frac{r(Y_t|X_{t,1}^{(k)})}{\pi(X_{t,1}^{(k)}|Y_t)} \sum_{j=1}^{N} w_{t-1}^{(j)} K_1(X_{t,1}^{(k)}|X_{t-1}^{(j)}).$$

**Sample history index** $I_t^{(i)}$ with law

$$p(I_t^{(i)} = l|J_t^{(i)}) \propto w_{t-1}^{(l)} K_1(X_{t,1}^{(J_t^{(i)})}|X_{t-1}^{(l)})$$

**Let the new particle**

$$X_t^{(i)} = \begin{pmatrix} X_{t,1}^{(J_t^{(i)})} \\ f_2\left(X_{t-1}^{(I_t^{(i)})}\right) \end{pmatrix}$$

.

**EndFor**
**EndFor**

Figure 3.4: SIR with history sampling (HS-SIR).

### 3.2.3 Rao-Blackwellised SIR with History Sampling

Our next algorithm can be considered as a Rao-Blackwellised[2] version of the previous one, whereas sampling of the innovation component indexes ($J_t^{(i)}$) is avoided - causing not only a speed-up, but also a reduction in the variance of the estimate of the posterior. The algorithm, that we call RB-HS-SIR (Rao-Blackwellised SIR with history sampling) is shown in Figure 3.5, whilst Figure 3.6 illustrates the algorithm's working principles.

Again, one expects that during the course of the algorithm the effective sample size will stay high as the algorithm will prefer (on the average) highly probable history-innovation associations.

[2]Assume that $Y$ has some relevant information regarding the value of $X$. Rao-Blackwellisation suggests to "integrate out" relevant information to decrease variance, i.e., $\mathrm{Var}(E(X|Y)) \leq \mathrm{Var}(X)$.

In order to show the unbiasedness of the proposed method we evaluate

$$R = E[\sum_{i=1}^{N} \overline{w}_t^{(i)} h(X_t^{(I_t^{(i)},i)}) | Y_{0:t}] = \tag{3.5}$$

$$= \sum_{i=1,l=1}^{N} E\Bigg[ P(I_t^{(i)} = l \big| Y_{0:t}, w_{t-1}^{(\cdot)}, X_{t-1}^{(\cdot)}, X_{t,1}^{(i)}) \tag{3.6}$$

$$E\big[\overline{w}_t^{(i)} h(X_t^{(I_t^{(i)},i)}) \,|\, Y_{0:t}, I_t^{(i)} = l, w_{t-1}^{(\cdot)}, X_{t-1}^{(\cdot)}, X_{t,1}^{(\cdot)}\big] \,|\, Y_{0:t} \Bigg]. \tag{3.7}$$

Note that evaluating $E[\overline{w}_t^{(1)} h(X_t^{(I_t^{(1)},1)}) | Y_{0:t}]$ only would be inefficient, since the history sampling step depends on all particles. This is why we included the sum over all particles in the expectation.

Now, since

$$P(I_t^{(i)} = l \big| Y_{0:t}, w_{t-1}^{(\cdot)}, X_{t-1}^{(\cdot)}, u_t^{(\cdot)}) = \frac{w_{t-1}^{(l)} K_1(X_{t,1}^{(i)} | X_{t-1}^{(l)})}{\sum_{j=1}^{N} w_{t-1}^{(j)} K_1(X_{t,1}^{(i)} | X_{t-1}^{(j)})},$$

by the definition of $\overline{w}_t^{(i)}$ one gets that

$$\begin{aligned} R &= \sum_{l=1,i=1}^{N} E\Bigg[ \frac{w_{t-1}^{(l)} K_1(X_{t,1}^{(i)} | X_{t-1}^{(l)})}{\sum_{r=1}^{N} w_{t-1}^{(r)} K_1(X_{t,1}^{(i)} | X_{t-1}^{(r)})} \\ &\qquad \frac{r(Y_t | X_{t,1}^{(i)}) \sum_{k=1}^{N} w_{t-1}^{(k)} K_1(X_{t,1}^{(i)} | X_{t-1}^{(k)})}{\pi(X_{t,1}^{(i)} | Y_t)} h(X_t^{(l,i)}) \,\Big|\, Y_{0:t} \Bigg] = \\ &= \sum_{l=1,i=1}^{N} E\Bigg[ w_{t-1}^{(l)} K_1(X_{t,1}^{(i)} | X_{t-1}^{(l)}) \cdot \frac{r(Y_t | X_{t,1}^{(i)})}{\pi(X_{t,1}^{(i)} | Y_t)} \cdot h(X_t^{(l,i)}) \,\Big|\, Y_{0:t} \Bigg] = \\ &= \sum_{l=1,i=1}^{N} E\Big[ w_t^{(l,i)} h(X_t^{(l,i)}) \,\Big|\, Y_{0:t} \Big], \end{aligned}$$

which finishes the proof of unbiasedness, since we have already seen in Section 3.2.2 that the particle set $(X_t^{(p,q)}, w_t^{(p,q)})$ gives an unbiased representation of the posterior.

Note that RB-HS-SIR can be viewed as an auxiliary variable method (see Appendix B), with the importance function:

$$\pi(X_t, k^{(j)} | Y_{0:t}) = p(k^{(j)} | X_{1,t}, Y_{0:t}) \pi(X_{1,t} | Y_{0:t}),$$

since $k^{(j)}$ defines $X_{2,t}$ deterministically, where $k^{(j)}$ is the history index.

Other variants of the algorithm can also be given. As an example let us mention the variant when the particles' innovation components are resampled, whilst their history components are retained. This variant can be advantageous if the particle set bears more information about the posterior than the proposal function.

### 3.2.4 Rao-Blackwellised Subspace SIR with History Sampling

Finally, let us mention the practical variant when the proposal function is further restricted to a few selected components of the innovation. For example in visual tracking, often the configuration is composed of translational and other components (e.g. rotation, scale) and the proposal depends only on the translational component (this is the case e.g. when color blob detection is used to define the proposal [15]). Formally, let us assume that the state space is factored as follows:

$$\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 = \mathcal{X}_{1a} \times \mathcal{X}_{1b} \times \mathcal{X}_2 \tag{3.8}$$

where we assume that the proposal is only defined on $\mathcal{X}_{1a}$. We also assume that the dynamics can be factored as

$$K_1(X_{t,1}|X_{t-1}) = K_1(X_{t,1a}, X_{t,1b}|X_{t-1}) = K_{1b}(X_{t,1b}|X_{t,1a}, X_{t-1})K_{1a}(X_{t,1a}|X_{t-1}),$$

where we can sample from $K_{1b}(X_{t,1b} = \cdot|X_{t,1a}, X_{t-1})$ and we can evaluate $K_{1a}(X_{t,1a}|X_{t-1})$. For these assumptions we propose a variant of RB-HS-SIR that we shall call RB-SS-HS-SIR (Rao-Blackwellised subspace SIR with history sampling). The algorithm is given in Figure 3.7.

The unbiasedness of the algorithm is straightforward. Note however, that sampling from $K_{1b}(X_{t,1b}|X_{t,1a}, X_{t-1})$ is not necessarily straightforward. Two exceptions are when the system noise is Gaussian (in this case $K_{1b}(X_{t,1b}|X_{t,1a}, X_{t-1})$ will still be Gaussian) and when $X_{t,1b}$ and $X_{t,1a}$ are independent given $X_{t-1}$ and $K_{1b}(X_{t,1b}|X_{t,1a}, X_{t-1})$ assumes a form that is easy to sample from. In this latter case $K_{1b}(X_{t,1b}|X_{t,1a}, X_{t-1}) = K_{1b}(X_{t,1b}|X_{t-1})$ and hence sampling from $K_{1b}(X_{t,1b}|X_{t,1a}, X_{t-1})$ reduces to sampling from $K_{1b}(X_{t,1b}|X_{t-1})$.

### 3.2.5 Experiments with Contour Tracking

In order to study the efficiency of the above algorithms, contour tracking experiments were run. The task was to track the contour of an artificial object moving in front of a camera in a normal office room environment (see Figure 3.8). The resolution of the images was set to $240 \times 180$. Note that another object with color and shape identical to that of the object to be tracked was lying on the table. As a consequence, the proposal keeps to draw "fake" positions that need to be "filtered out", making the job of blob-detection based trackers non-trivial.

#### The Proposal

The output of a Gaussian color blob detector working on the original frames was used as the basis of the proposal, just like in [15]. First, the output of the blob detector was down-sampled to a resolution of $24 \times 18$ pixels. Then spatial coordinates were drawn from the appropriately re-scaled output of the blob detector. These coordinates were then mapped back to the original coordinate system of the images. The final coordinates were obtained by applying a random perturbation to the coordinates calculated so far, by

adding a random "fine-scale" random displacement vector drawn uniformly from the set $\{-5,-4,\ldots,5\}\times\{-5,-4,\ldots,5\}$.

**Results**

For the sake of comparisons Basic-SIR and RB-SS-HS-SIR were implemented and tried on a number of image sequences. A typical tracking scenario using RB-SS-HS-SIR is presented in Figure 3.8. In a sequence of 5 seconds of video sampled at 30Hz the configurations of the object to be tracked were determined manually (this is the sequence shown on Figure 3.8). Both algorithms were then tested on this sequence with 100 different random seeds. Tracking error and the probability of losing the object to be tracked were measured as a function of frame number. Equivalent running time experiments were considered on an Intel Pentium IV 1.4GHz computer with 128MB RAM, i.e., the particle sizes were set so that the running time of the algorithms were the same, and both resulted in acceptable average tracking performance. No attempt was made to do any serious optimizations of the algorithms. Respective particle set sizes are given in Table 3.1.

| **Algorithm** | **# Particles** |
|---|---|
| Basic-SIR | 3000 |
| RB-SS-HS-SIR | 400 |

Table 3.1: *Particle sizes used in the experiments.*

Tracking errors of the algorithms averaged over the 100 runs are shown in Figure 3.9. The error is computed as the distance (in pixels) in between the estimated position and the true position of the object to be tracked. It should be clear from the figure that in terms of the errors: RB-SS-HS-SIR is much better than Basic-SIR. The estimated probability of losing the object as a function of frame indexes is shown in Figure 3.10. These estimates are computed by counting the fraction of cases (of the 100 runs) when the output of the tracker is outside of a certain large neighborhood of the object to be tracked (50 pixels). In order to separate the effect of losing the object from problems with accuracy when the object is tracked, when the object is lost at a certain point in time, the corresponding distances are not included in the computation of the average error. Again, RB-SS-HS-SIR performs better than Basic-SIR.

### 3.2.6 Discussion

These experiments indicate that under a wide range of conditions the proposed algorithms do indeed overcome the inefficiency of Basic-SIR. Although the results are encouraging, one should bear in mind that the new algorithms are computationally more expensive than the original Basic-SIR algorithm: now one iteration requires $O(CN)$ evaluations of the density $K_1(X_t|X_{t-1})$, where $C$ is the number of distinct particles after resampling. If resampling is not made in a time step, then the number of evaluations will be $O(N^2)$. Note that this

is also the case for ICondensation [15]. Fortunately, however, the number of times the observation density needs to be evaluated still scales linearly with the number of particles. Therefore the history sampling algorithms can be cheaper than the Basic-SIR when the cost of evaluating the observation density for a larger number of particles is higher than the cost of evaluating the prediction density $K(X_t|X_{t-1})$ $O(N)$ times. This is in fact quite often the case when dealing with visual object tracking, as the image processing steps are typically very expensive.

What remains is the discussion of the relation of the algorithms to ICondensation, the algorithm introduced by Isard and Blake in [15]. At a first glance ICondensation looks very similar to Basic-SIR. However, let us take a closer look at this algorithm. In Figure 1 of [15] in Step 2(a) the next state is sampled from the proposal as usual. However, importance weights are calculated with the formula used in RB-HS-SIR[3] - possibly causing small performance deterioration.[4] Note that if all the particles are concentrated into a relatively small portion of the state space then the importance weights calculated as in RB-HS-SIR will be close to the "correct" ones. The same applies when the dynamics is close to the uniform distribution. Also, note that ICondensation as described in [15] mixes several algorithms: re-initialization, CONDENSATION and Basic-SIR with the modification described above, therefore it is hard to analyze theoretically.

[3]The same problem appears when they describe the details of the algorithm in Section 4.2.

[4]Note that "incorrect" weights do not necessarily cause a problem, see e.g. Theorem 3.1 of [45].

**Initialize** $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(X_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(\hat{X}_{t-1}^{(j)}, \hat{w}_{t-1}^{(j)})\right]_{j=1}^{N}$.

**For** $i = 1, 2, \ldots, N$

**Sample innovation** from the proposal $X_{t,1}^{(i)} \sim \pi(\cdot|Y_t)$.

**Sample history index** $I_t^{(i)}$ with law

$$p(I_t^{(i)} = l) \propto \hat{w}_{t-1}^{(l)} K_1(X_{t,1}^{(i)}|\hat{X}_{t-1}^{(l)})$$

**Let the new particle**

$$X_t^{(i)} = \begin{pmatrix} X_{t,1}^{(i)} \\ f_2\left(\hat{X}_{t-1}^{(I_t^{(i)})}\right) \end{pmatrix}.$$

**Calculate the importance weight**

$$\overline{w}_t^{(i)} = \frac{r(Y_t|X_{t,1}^{(i)}) \sum_{l=1}^{N} \hat{w}_{t-1}^{(l)} K_1(X_{t,1}^{(i)}|\hat{X}_{t-1}^{(l)})}{\pi(X_{t,1}^{(i)}|Y_t)}.$$

**EndFor**

**Normalize weights** using

$$w_t^{(j)} = \frac{\overline{w}_t^{(j)}}{\sum_{i=1}^{N} \overline{w}_t^{(i)}}$$

**EndFor**

Figure 3.5: SIR with Rao-Blackwellised history sampling (RB-HS-SIR).

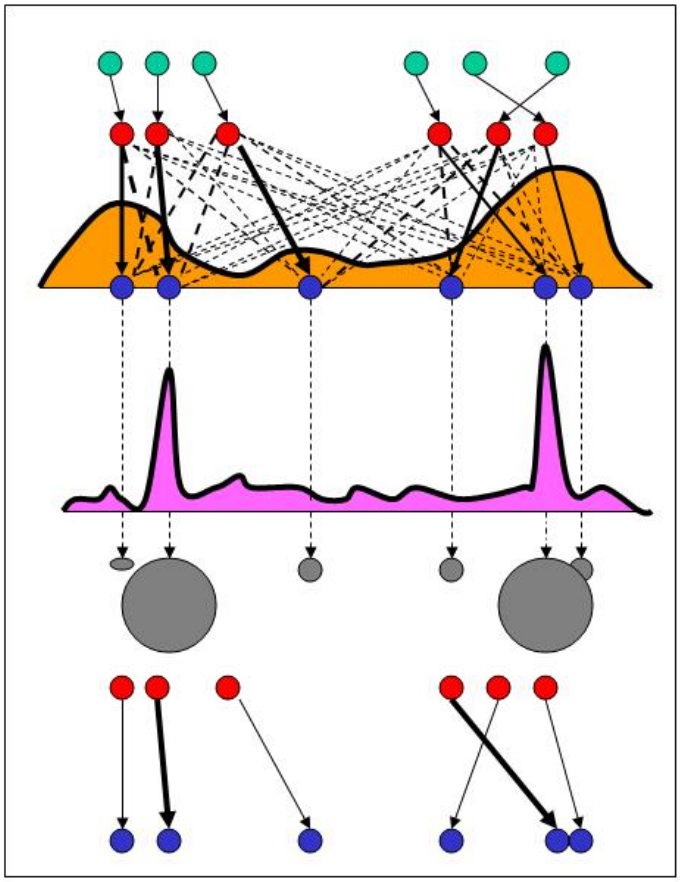

Figure 3.6: *Illustration of the behavior of the RB-HS-SIR.* The figure is almost identical to Figure 3.3, except that now the arrows from $X_{t-1,1}$ to $X_{t,1}$ show all the possible associations of innovations and particles, and the strength of these arrows are proportional to the weights that are used in associating particles histories and innovation components. It should be clear that the posterior is grabbed better by this algorithm.

**Initialize** $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(X_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(\hat{X}_{t-1}^{(j)}, \hat{w}_{t-1}^{(j)})\right]_{j=1}^{N}$.

**For** $k = 1, 2, \ldots, N$

**Sample innovation** from the proposal $X_{t,1a}^{(k)} \sim \pi(\cdot|Y_t)$.

**Sample history index** $I_t^{(k)}$ with law

$$p(I_t^{(k)} = l) \propto \hat{w}_{t-1}^{(l)} K_{1a}(X_{t,1a}^{(k)}|\hat{X}_{t-1}^{(l)})$$

**Sample** remaining part of the innovation $X_{t,1b}^{(k)}$ from

$$K_{1b}(X_{t,1b} = \cdot|X_{t,1a}^{(k)}, X_{t-1}^{(I_t^{(k)})})$$

.

**Let the new particle**

$$X_t^{(k)} = \begin{pmatrix} X_{t,1a}^{(k)} \\ X_{t,1b}^{(k)} \\ f_2\left(X_{t-1}^{(I_t^{(k)})}\right) \end{pmatrix}.$$

**Calculate the importance weight**

$$\overline{w}_t^{(k)} = \frac{r(Y_t|X_{t,1}^{(k)}) \sum_{l=1}^{N} w_{t-1}^{(I_t^{(k)})} K_{1a}(X_{t,1}^{(k)}|\hat{X}_{t-1}^{(I_t^{(k)})})}{\pi(X_{t,1}^{(k)}|Y_t)}.$$

**EndFor**

**Normalize weights** using

$$w_t^{(j)} = \frac{\overline{w}_t^{(j)}}{\sum_{i=1}^{N} \overline{w}_t^{(i)}}$$

**EndFor**

Figure 3.7: *Rao-Blackwellised subspace SIR with history sampling (RB-SS-HS-SIR).*

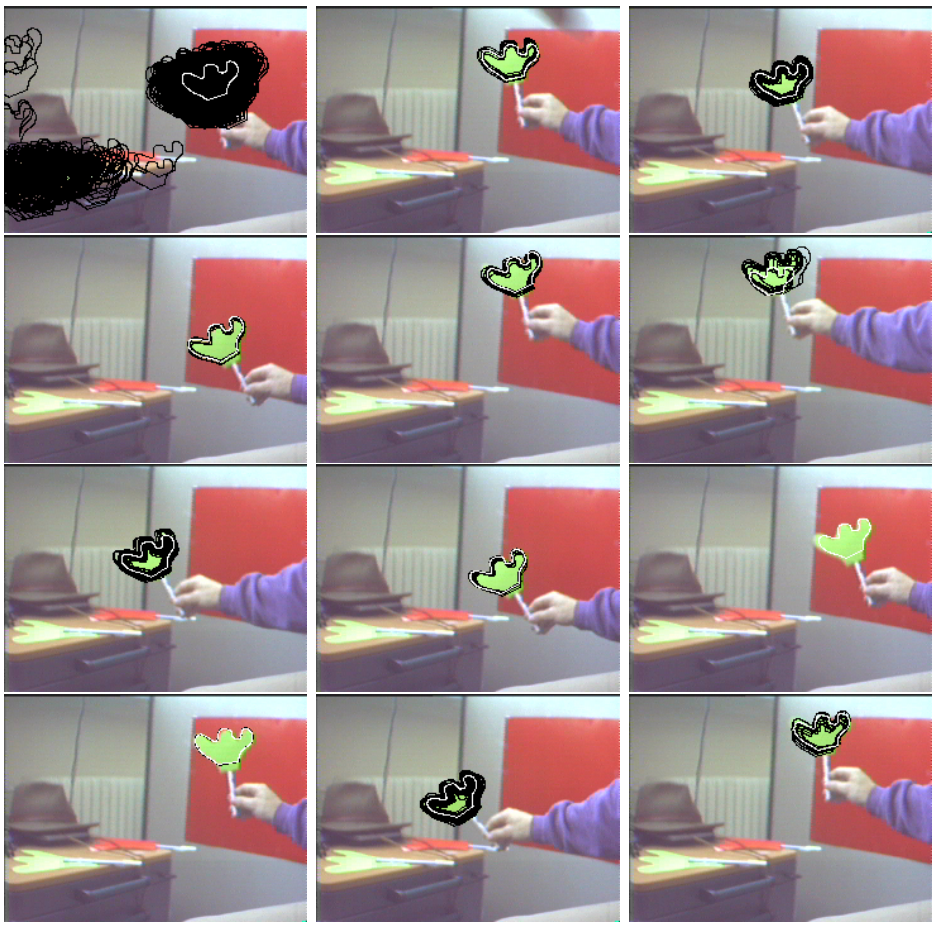

Figure 3.8: *A typical tracking sequence with RB-SS-HS-SIR and* 600*-particles.* Black contours show configurations with high probabilities, while the white contour represents the average configuration. Note that there is an object lying on the table that has the same characteristics (e.g. is made of the same material of the same color) as the object to be tracked, making the tracking task more difficult.

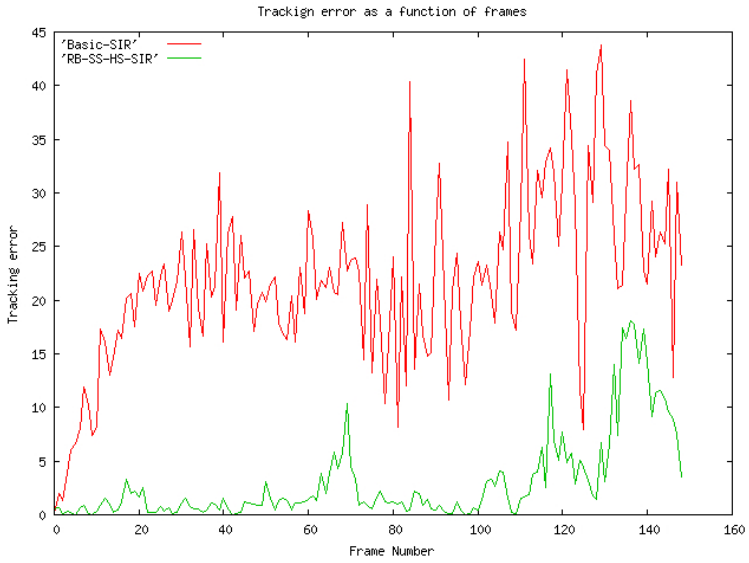


Figure 3.9: Tracking error as a function of frame number.

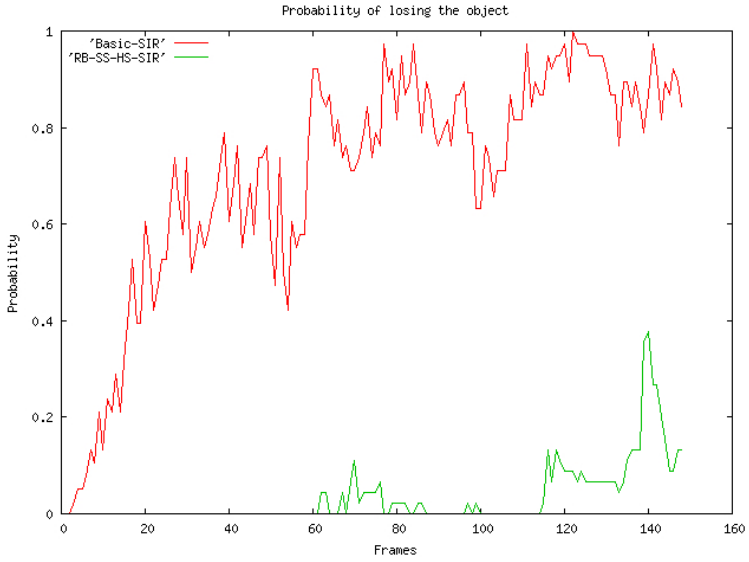


Figure 3.10: Probability of losing the object.

# Chapter 4

# Local Perturbations in Particle Filtering

## 4.1 Motivation

In visual tracking the vision engineer has a strong influence on how the observations are derived from the image. By employing a rich set of features it is possible to construct reliable observations, e.g. by combining the output of features working with different modalities such as shape, color, texture, contours and intensity [30, 29, 8, 37, 14]. Unfortunately, the performance of particle filters degrades seriously when the level of observation noise is low and the number of particles is not sufficiently high (see Figure 4.1). In this chapter we will consider methods to improve the performance of particle filters when the level of the observation noise is low.

In the followings we will assume to work with reliable observations. In this case the observation likelihood function becomes 'peaky' or concentrated around its modes (the modes correspond to the states that are locally most likely to 'cause' the past observations). If the position of a particle is not sufficiently close to one of these modes then the corresponding weight will bring in little information into the estimate of the posterior. If this happens for most of the particles then the quality of the approximation to the posterior may become seriously degraded. We call this problem the *"curse of reliable observations"*.

The curse of reliable observations is a well-known peculiarity of particle filters and many proposals have been suggested to overcome it. The most generic of these is importance sampling discussed earlier, where the algorithm designer can choose a proposal density that may depend on both the most recent observation *and* the state to sample the particles' new positions from. However usually the proposal just depends on the last observation, which was also our assumption in Chapter 3. Choosing a proposal is often considered an art: no generic designs can be found in the literature. Another particularly straightforward way to overcome the curse is to increase the number of particles until it is ensured that a sufficiently high number of particles will be close to the peaks of the observation likelihood function. In high dimensional state spaces, this approach may require an enormous number

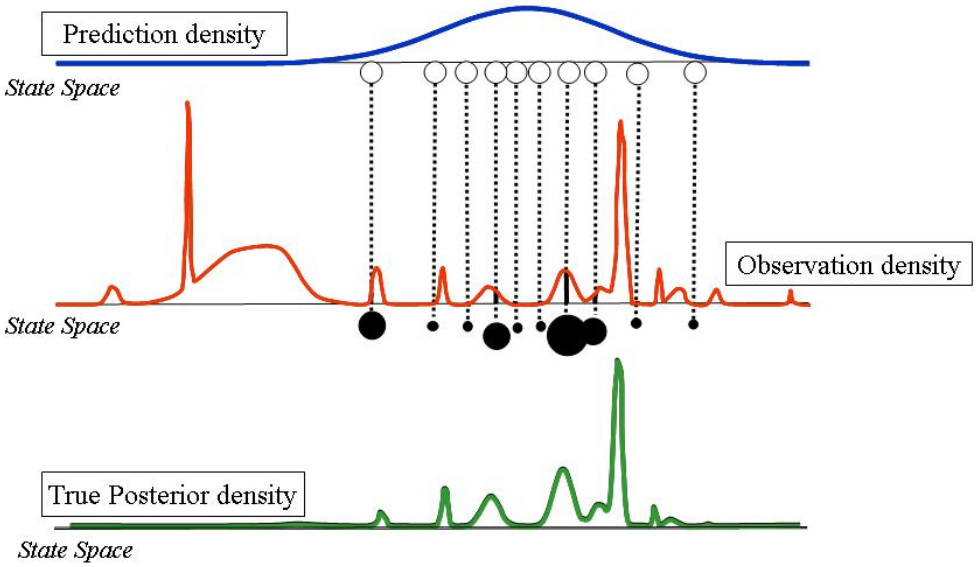


Figure 4.1: Sampling with Bootstrap Filter: the case of peaky observation densities. Since the particles' positions do not match the dominant modes of the observation density the representation becomes degraded.

of particles. In addition, in visual tracking it is the evaluation of the observation likelihood that is the computationally most expensive step, as evaluating this likelihood requires computations that involve image processing steps. Thus, a large number of particles is highly undesirable as it can slow down the filtering process.

Since the inefficiency stems from the particles' positions not being close enough to the modes of the observation likelihood function, it is a natural idea to design algorithms where the observation likelihood influences the particles' positions. Indeed this is the idea that we follow in this chapter. In all methods introduced here the particles' positions are generated in a two-stage sampling process, where one of the sampling steps uses information of the observation likelihood density. In the algorithm studied first (called LS-N-IPS), a local search is done on all predicted particle positions, thus introducing some bias in the estimation procedure. Still the algorithm will be shown to outperforms its competitors by a large margin, thanks to the reduced variance of the estimates it provides.

In the case of the *Local Likelihood Sampling* based particle filter (LLS), a localized version of the observation likelihood function is used to adjust the particles' positions that are initially sampled from the prediction density $p_M(X_t = x|Y_{0:t-1})$, just like in case of the Bootstrap Filter. Weights are calculated so that the process remains unbiased. In the case of the *Local Importance Sampling* (LIS), the first sampling step remains the same, whilst in the second step the observation likelihood is replaced by a user-chosen proposal density function called *local importance function* that should be close to the localized observation

likelihood. The weight update equations are modified so that unbiasedness is retained. The motivation for this algorithm is that in some cases sampling from the exact likelihood function's localized version cannot be implemented efficiently. In such a case it might be worthwhile to use a density whose shape resembles the localized likelihood, but which is easier to sample from.

## 4.2 Literature Overview

Alternative proposals aiming to overcome the curse of reliable in observations include the Auxiliary Variable Method (AVM) introduced by Pitt and Shephard [31]. AVM uses a proposal density of the form $\pi(X_t|X_{t-1}^{(1)},\ldots,X_{t-1}^{(N)},Y_t) = \sum_{k=1}^{N} r(Y_t|\overline{X}_t^{(k)})K(X_t|X_{t-1}^{(k)})$, where $\overline{X}_t^{(k)}$ is e.g. the expected next state for particle $k$. Sampling is implemented by first doing a weighted resampling step (for details see Appendix C) using the weights $r(Y_t|\overline{X}_t^{(k)})$ and then drawing the next states using the transition density kernel $K$. For exact details of the AVM procedure see Appendix B. AVM can be more efficient than the Bootstrap Filter when the process noise variance is low and the observation likelihood is not too peaky. When the observation likelihood is peaky and the number of particles is not high enough then resampling the particle set using $\{r(Y_t|\overline{X}_t^{(k)})\}$ might not be successful at picking the particles that were likely to produce the observation. A similar problem occurs when the prediction density is multi-modal and hence $\overline{X}_t^{(k)}$ is not a good candidate to predict the likelihood $p(Y_t|\overline{X}_t^{(k)})$. Also, when the prediction density has a large variance as compared to the observation noise then even if the first stage is successful, sampling the particle's next state from $K$ might yield to a set of particles that are spread out too much in the state space.

Annealed sampling [8] and the work of Cham and Rehg [5] is closely related to LS-N-IPS, however in [8] the search made after the prediction stage is clearly not local, which can cause serious bias in the posterior, as the effect of dynamical model decreases. In [5] an interesting approach is described. The density representation is piecewise Gaussian, and as a result after the local search operator one estimates the covariance matrix, using perturbation analysis (a time consuming step). Another disadvantage with this representation is that the Bayes' rule has to be applied to all pairs of Gaussians coming from the prediction, observation representation. This also implies that if the feature space and the state space does not coincide, then the algorithm cannot be used. In [5] a constant velocity model is used (a first order model) to overcome this problem.

Another recent account is likelihood sampling considered e.g. in details in [6]. In this approach it is the likelihood function $p(Y_t|\cdot)$ that is used as the proposal, whilst the prediction density is used to calculate the weights. Thus the success of this method depends on whether the likelihood is a good predictor of the true state. For multi-modal likelihoods (when aliasing effects are severe) a large number of particles can be generated away from the likely next positions of the true state. These particles will get low weights in the weighting process and thus will have no significant effect on the estimated posterior.

Hence the effective sample size can be small in this case. Our methods overcome this problem by first sampling from the prediction density and hence concentrating the sample into the vicinity of the 'correct' peaks of the likelihood function.

Boosted particle filters (BOOPF) [28] are probably the closest to LLS/LIS filters in their spirit. BOOPF is an instance of SIR. Its proposal can be written in the form:

$$\begin{aligned}\pi(X_t = \cdot|X_{t-1}, Y_t) &= \alpha(\overline{X}_t, Y_t)\, r(Y_t|X_t = \cdot) g(\overline{X}_t - X_t = \cdot) \\ &\quad + \big(1 - \alpha(\overline{X}_t, Y_t)\big)\ K(X_t = \cdot|X_{t-1}),\end{aligned}$$

where $\overline{X}_t$ is the expected next state given $X_{t-1}$, $g$ is a rectangular window function and

$$\alpha(\overline{x}, y) = \begin{cases} A, & \text{if } r_0 < \max_{x':d(x',\overline{x})\leq\lambda} r(y|x'); \\ B, & \text{otherwise.} \end{cases}$$

Here $0 < B \ll A < 1$, and $r_0, \lambda$ are parameters to be chosen by the user. In effect, BOOPF will sample the next state from the localized version of the observation likelihood when the observation likelihood is sufficiently large in a neighborhood of the expected next state, whilst in the other case the prediction density is used to sample the next state. Note that the version of this algorithm presented in [28] uses heuristically derived approximations to the observation likelihood and it is slightly more complicated than the one presented here. In any case, this algorithm is a nice exception when the proposal depends on both the dynamical prediction and on the most recent observation and that it is possible to implement sampling in an efficient manner. However when the prediction density is multi-modal or when it has a large variance then the expected value of the next state can be a bad predictor of where the object might be in the next step. In such cases this algorithm would practically perform identically to the naive sampling scheme of the Bootstrap Filter and thus it would suffer from the curse of reliable observations. Our algorithms are capable of coping with such difficult cases thanks to the two-stage sampling scheme.

## 4.3 The LS-N-IPS Algorithm

The "Local Search"-modified N-IPS[1] algorithm (LS-N-IPS) was introduced in [44, 45, 47, 46]. The algorithm is shown in Figure 4.2.

The difference between LS-N-IPS and N-IPS (or Bootstrap Filter) is in the update of the proposed states. LS-N-IPS uses a non-trivial local search operator, $LS_\lambda$, to "refine" the predictions drawn from the dynamical model, as shown visually in Figure 4.3. Note that since we do not compensate for this local perturbation, we introduce some bias as compared to the posterior predicted using the basic Bootstrap Filter.

The idea here is that a good local search operator, $LS_\lambda$ should satisfy $r(y|LS_\lambda(x, y)) \geq r(y|x)$. The parameter $\lambda > 0$ defines the "search length": $LS_\lambda$ is usually implemented as a (local) search trying to maximize $r(y|\cdot)$ around $x$, in a neighborhood with size $\lambda$, e.g.

$$LS_\lambda(x, y) = \text{argmax}\{r(y|\tilde{x}) \,|\, ||\tilde{x} - x|| \leq \lambda\} \tag{4.1}$$

[1] The name N-IPS comes from N-Interacting Particle System and was introduced in [7].

**Initialize** $\left[(X_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**For** $k = 1, 2, \ldots, N$

**Sample** $\hat{X}_t^{(k)} \sim K(X_t = \cdot | X_{t-1}^{(k)})$.

**Search locally** around $\hat{X}_t^{(i)}$ to get a new state : $X_t^{(k)} = LS_\lambda(\hat{X}_t^{(k)}, Y_t)$.

**Calculate the importance weight** $\hat{w}_t^{(k)} = r(Y_t | X_t^{(k)})$.

**EndFor**

**Normalize** weights using

$$w_t^{(k)} = \frac{\hat{w}_t^{(k)}}{\sum_{i=1}^{N} \hat{w}_t^{(i)}}$$

**Make resampling** if needed.

**EndFor**

Figure 4.2: The LS-N-IPS algorithm.

Here $||\,.\,||$ can be the scaled maximum norm, or some other appropriate norm.

If $\lambda = 0$, then no local search modification is involved, and we get the usual N-IPS algorithm (Bootstrap Filter). On the other hand, $\lambda \to \infty$ yields to a complete search in $X$, since the algorithm only samples the likelihood without considering the dynamical model. This is clearly not a desired behavior. These arguments show that $\lambda$ is an important design parameter, the choice of which not just improves the particle representation, but also controls the bias on the original tracking model.

For moderate $\lambda$ one might explain the local search as a slight modification to the original dynamical model, which is in practical cases an approximate one by itself. However one can prove that the introduced bias stays bounded [45]. In [43] a detailed theoretical analysis of this algorithm can be found. Here we only include some experimental results on the contour tracking problem.

### 4.3.1 Implementing Local Search for Contour Tracking

In contour tracking the task of the local search procedure is to find the best matching admissible contour in a small vicinity of the one predicted by the dynamical model [44]. The problem is illustrated in Figure 4.4. In the followings we will describe two methods, a least mean square (LMS) based local search and a neural network (NN) based one [47].

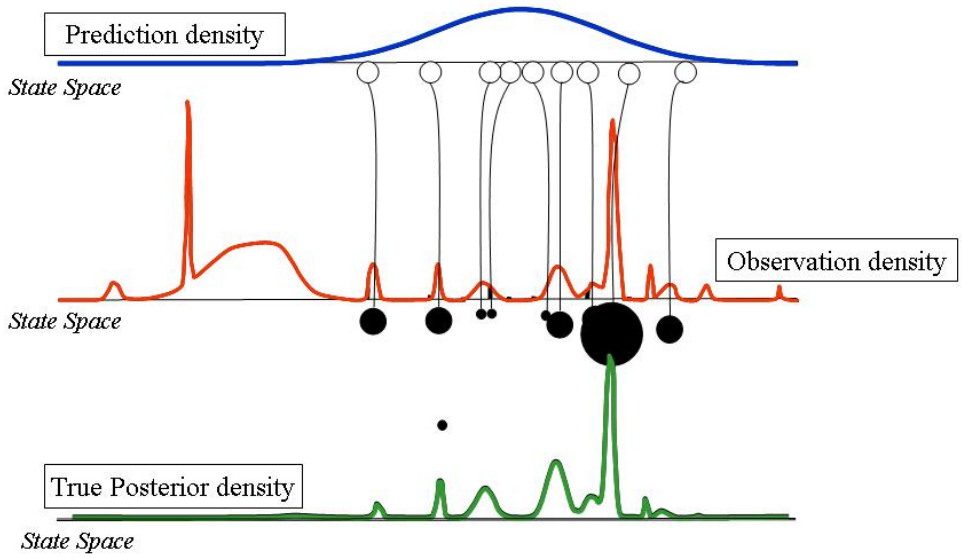


Figure 4.3: Visualization of the LS-N-IPS algorithm. The curves instead of the straight lines are meant to represent the local search process. Note that the posterior is represented nicely.

In both cases the search is implemented by first finding the most likely edge locations and directions along some normal lines of the predicted curve, and then finding the configuration that matches best to these measurements. The main difference between the two approaches lies in the way of finding the configuration given the edge measurements, or in other words mapping the new contour control point locations to the configuration space.

### Least Mean Square based Local Search

Assume that a contour ($S$) corresponding to some pose $Z$ (usually a subspace of the state space $X$) is given. As we have discussed in Section1.3.1 the likelihood of the contour given the image (the observation) is the product of the individual "likelihoods" of edges being located at some contour normal measurement points along the spline curve.

Motivated by this definition, the LMS-based local search algorithm [43] searches for maximal edge 'likelihood' values along the normals in the vicinity of the measurement points, the neighborhood itself defined by the search length, $l > 0$. The measurement points are chosen to be the support points $(q_1^x, q_1^y)^T, \ldots, (q_n^x, q_n^y)^T$.

Assume that the search yields to the points $(\hat{q}_1^x, \hat{q}_1^y)^T, \ldots (\hat{q}_n^x, \hat{q}_n^y)^T$. Let

$$\hat{q} = (\hat{q}_1^x, \ldots \hat{q}_n^x, \hat{q}_1^y, \ldots \hat{q}_n^y)^T$$

(yellow points in Figure 4.4). Let $\hat{s}$ be the spline curve corresponding to $\hat{q}$.

Figure 4.4: *Local search problem in contour tracking.* Left image shows the predicted contour (blue) with its control points (red) and a single edge likelihood measurement line (black), and the result of a local search (green contour with orange control points. The middle image shows the same on a real image. The searched edge locations lying on the measurement lines are shown in yellow. The right image is just a zoom of the middle image in the relevant region.

The next step is to find a configuration whose corresponding spline curve matches $\hat{s}$ the best:

$$\hat{Z} = \text{argmin}_Z \|S(Z) - \hat{S}\|_2^2.$$

Here $S(Z)$ denotes the spline curve corresponding to the configuration $Z$, i.e., the spline curve corresponding to the support vector $q_Z = WZ + q_0$.

It is well known that if $S$ is the spline contour corresponding to $q$ then $||S||_2^2$ can be expressed as a function of $q$ alone. In particular,

$$||S||_2^2 = \frac{1}{L}\int_0^L S^2(t)dt = q^T \begin{pmatrix} B & 0 \\ 0 & B \end{pmatrix} q = q^T U q,$$

where

$$B = A^{-T}\left(\frac{1}{L}\int_0^L \varphi(u)\varphi(u)^T du\right) A^{-1}.$$

Therefore, if we let $\|q\|_S^2 = q^T U q$ denote the weighted $\ell^2$ norm of $q$ then

$$\hat{Z} = \text{argmin}_{Z\in\mathbb{R}^d} \|WZ + q_0 - \hat{q}\|_S^2.$$

Now, by standard LMS calculations $\hat{Z} = W^+(\hat{q} - q_0)$, where $W^+$ is the pseudo inverse with the above norm, i.e. $W^+ = (W^T U W)^{-1} W^T U$. The corresponding projected support vector shall be denoted by $q^\perp = W\hat{Z} + q_0$.

Note that it is not obvious to see why the dynamics perturbed by this local search operator $LS_\lambda$ stays close to the original dynamics. One can however check weather the adjusted pose is close enough to the predicted one, and reject the result of the local search if it is too far in the configuration space.

**Neural Network based Local Search**

In the general formulation of the LS-N-IPS algorithm, the local search procedure was given as a mapping of a predicted configuration $Z$ and the actual observation $Y$ to an adjusted configuration $\hat{Z}$. One approach to "design" this operator is to learn it by some machine learning algorithm [47].

This approach is particularly fruitful if the observation function ($r$) can be simulated, since in this case training examples can be generated at no cost by drawing pairs of configurations $(Z, Z')$ "close to each other" and drawing measurement signal $Y$ with considering $Z$ as the predicted particle and $Z'$ as the true state. A training data point of the machine learning algorithm can be generated by simulating the local search process at $Z$ and simulating the observations by $r(Y|Z')$. The desired output is set to $Z'$. The learning criterion might be to choose $LS_\lambda$ such that for $\|Z - Z'\| \leq \lambda$ the cost

$$\mathrm{E}[\|LS_\lambda(Z, r(Y|Z')) - Z'\|^2]$$

is minimized. A practical approximation of the above criterion based on sample averages is

$$E = \frac{1}{n}\sum_{i=1}^{n} \|LS_\lambda(Z_i, r(Y|Z_i')) - Z_i'\|^2,$$

where $\|Z_i - Z_i'\| \leq \lambda$. One hopes that training e.g. a neural network to minimize the above criterion will then yield a "good" local search operator.

Note that if the above described mapping $(Z, Y|Z') \to Z'$ is invariant to some similarity group $G$, i.e.: $H \in G$ implies $(H(Z), Y|H(Z')) \to H(X')$, then the above learning problem can be further simplified by choosing $Z$ as a normal pose $Z_0$.

In the followings neural networks will be used to represent $LS_\lambda$ in the contour tracking problem. Note that normal pose trick can be used here as well.

The training data was generated as follows: Random poses are generated in the vicinity of a "normal" pose $Z_0$. Then a contour search procedure is executed, starting from the contour corresponding to the normal pose. This procedure finds intersection points of the contour corresponding to the randomly generated pose $Z'$ and the normals of the contour $Z_0$ at its support points, just like as it was described in Section 1.3.1. (Figure 4.4 gives an example.) If, for a given normal, more than one intersection point is found one of them is selected at random. The input to the neural network is then composed of the coordinates of the found intersection points. The desired output is set to $Z'$.

When the neural net is trained, it is used as shown in Figure 4.5.

### 4.3.2 Experimental Results on the Contour Tracking Problem

Several networks were tried with various contours. All networks were trained with $100,000$ randomly generated data points, perturbed by Gaussian-like noise (applied for regularization purposes).

In all the experiments the net outputs were scaled to ensure that the configuration coordinates receive equal importance during training. We used $\lambda = (7, 7, 0.15, 0.1)^T$, where

**For** $i = 1, \ldots, n$

**Calculate edge likelihoods along the normal segment** of $S(WZ + q_0)$ at the $i$-the control point $(q_i^x, q_i^y)$, with (search) length $l = l_0 s$, where $s$ is the scale encoded in $Z$.

**Select the maximum value** on the normal segment: $(\hat{q}_i^x, \hat{q}_i^y)$.

**Normalize** $\{(\hat{q}_i^x, \hat{q}_i^y)\}_{i=1}^n$ with the transformation $H$ for which $HZ = Z_0$, where $Z_0$ is the reference pose that was used for training the neural network.

**EndFor**
**Excite the neural network with** $\left\{H(\hat{q}_i^x, \hat{q}_i^y)^T\right\}_{i=1}^n$. Let the resulting configuration be $\hat{Z}_0$.
**Compensate for the normalization step** by transforming $\hat{Z}_0$ back by $\hat{Z} = H^{-1} Z_0$.
**Return** $\hat{Z}$.

Figure 4.5: Neural Net Based Local Search for the contour tracking problem.

the first two coordinates correspond to the translation parameters (the contour to be tracked was about of size $60 \times 60$ on the same scale), the third coordinate is rotation in radians, whilst the last is the scale parameter. We have chosen $l_0 = 16$. The neural network was trained using RProp [26].

The above algorithm was tested on several real-world tracking problems. In the most difficult scenario, a fast moving hand was tracked in a highly cluttered environment. Using the LMS-based algorithm with 50 particles we could achieve a tracking speed of 30 frame/second on our Intel Pentium 4 machine, without scarifying accuracy and robustness. Using the NN-based algorithm similar tracking accuracy is achieved with as few as 25 particles [47], half of the amount needed in case of LMS [44]. In this example the number of support points was 30. This leads to a tracking speed of 73 frames/seconds which is a significant improvement (see table).

Figure 4.6 shows every 15th frame of a typical tracking session. This image sequence was recorded at 30 frames/second. The duration of the whole sequence is 6 seconds and the hand moves 9 times from one corner of the image to the other. The number of particles was chosen to be 30. The image resolution was $240 \times 180$.

## 4.4 Local Likelihood Sampling

In this section after introducing some basic notations we will introduce the Local Likelihood Sampling scheme [49] that can be used to generate a particle representation of the

| **Experiment** | **N-IPS** (Bootstrap Filter) | **LS-N-IPS with LMS** | **LS-N-IPS with NN** |
|---|---|---|---|
| Number of Particles need | 2000 | 100 | 30 |
| Pred. time per particle(ms) | 0.123 | 0.31 | 0.452 |
| Tracking Frame rate(frame/sec) | 4 | 32 | 73 |

Table 4.1: *Comparison of tracking result with different algorithms. For an explanation see the text.*

Figure 4.6: *Tracking hand in clutter with* 30 *particles. Black contours show particles having high observation likelihoods, whilst the white contour shows the predicted contour of the hand. When no black contour is given, then typically there is one particle which has significantly higher likelihood than the others.*

product of a density and a likelihood function. In fact the main interest is the density coming from the product after normalization. After analyzing this general sampling scheme theoretically, we show how to use it in particle filtering.

**Some Notation**

Remember that for an integrable function $f$, $I(f)$ denoted the integral of $f$ with respect to the Lebesgue measure. $L^p$ $(0 < p \leq +\infty)$ denotes the set of functions with finite $p$-norm. The $p$-norm of a function is denoted by $\|f\|_p$. For a function $f \in L^s(\mathbb{R}^d)$, $s \in \{1, 2\}$, $F(f)$ denotes its Fourier transform:

$$F(f)(\omega) = \int e^{-i\omega^T x} f(x) dx \ .$$

The inner product defined over $L^2(\mathbb{R}^d)$ is defined by

$$\langle f, g \rangle = \int f(x) \overline{g}(x) dx \ ,$$

where $\overline{a}$ denotes the complex conjugate. Convolution is denoted by $*$:

$$(f * g)(x) = \int f(y) g(x - y) dy \ .$$

**The Local Likelihood Sampling (LLS) scheme**

Assume we want to sample from a product of two densities $f$ and $p$. The basic idea of the proposed sampling scheme is to first draw samples from $p$, then allow the density $f$ to 'perturb' the position of the particles. A window-function ($g$) is used to localize the effect of $f$ on the sample, hence the name of the procedure. The sampling scheme is shown in Figure 4.7, and is illustrated on Figure 4.8. The output of the sampling scheme is a set of particles $\left[(Z^{(i)}, w^{(i)})\right]_{i=1}^N$, that can be used to approximate $I(hfp)$ by $J_N = \sum_{j=1}^N w^{(j)} h(Z^{(j)})$, where $h \in L^1$ is any function of interest. Actually, as usual, we are more interested in studying

$$I_N(h, w) = \frac{\sum_{j=1}^N h(X^{(j)}) w^{(j)}}{\sum_{j=1}^N w^{(j)}} \tag{4.2}$$

which also converges to $I(hfp)/I(fp)$ as $N \to \infty$ with probability one when e.g. $(X^{(j)}, w^{(j)})$ are independent of each other.

Here and in what follows we assume that $g \in L^1$ and is a non-negative function. It could be e.g. a Gaussian, or the characteristic function of some convex set.

### 4.4.1 Theoretical Analysis

The following proposition shows that weights are calculated so that $(Z^{(j)}, w^{(j)})$ is a properly weighted pair for $f, p$:

**For** $i = 1, \ldots, N$

**Sample** $X^{(i)}$ from $p(\cdot)$.

**Draw** $Z^{(i)}$ from

$$Z^{(i)} \sim \tfrac{f(\cdot)g(X^{(i)}-\cdot)}{(f*g)(X^{(i)})}.$$

**Calculate importance weight**

$$w(Z^{(i)}) = w^{(i)} = (f*g)(X^{(i)})\frac{p(Z^{(i)})}{p(X^{(i)})}.$$

**EndFor**

Figure 4.7: The Local Likelihood Sampling (LLS) Algorithm to generate a particle representation from the product of $f$ and $p$.

**Proposition 4.4.1.** *Assume that $g \in L^1$ is a window function satisfying $g \geq 0$ and $I(g) = 1$. Let $f$ be a bounded, integrable function and let $p > 0$ be a density. Then, the above sampling procedure yields properly weighted pairs $(Z^{(j)}, w^{(j)})$ with respect to $f, p$. (see Definition 2.2.1).*

*Proof.* Let $h$ be an arbitrary integrable function and let $I = E[w^{(j)}h(Z^{(j)})]$. By the law of total probability, $I = E[E[w^{(j)}h(Z^{(j)}) \,|\, X^{(j)}]]$. By the definition of $Z^{(j)}$ and $w^{(j)}$,

$$\begin{aligned} E[w_j h(Z_j) \,|\, X_j] &= \int h(z)p(Z^{(j)}|X^{(j)})w(Z^{(j)}), dz \\ &= \int h(z)\frac{(f*g)(X^{(j)})p(z)}{p(X^{(j)})}\,\frac{f(z)g(X^{(j)}-z)}{(f*g)(X^{(j)})}\,dz \\ &= \int h(z)\frac{p(z)}{p(X^{(j)})}\,f(z)g(X^{(j)}-z)\,dz\ , \end{aligned}$$

and hence

$$\begin{aligned} I &= \int\int h(z)\frac{p(z)}{p(x)}\,f(z)g(x-z)\,dz\,p(x)\,dx \\ &= \int\int h(z)p(z)f(z)g(x-z)\,dz\,dx\ . \end{aligned}$$

The order of integration can be exchanged, because Fubini's theorem's conditions are satisfied. Using $I(g) = 1$ we get the desired equality:

$$I = \int h(z)p(z)f(z)dz\ .$$

□

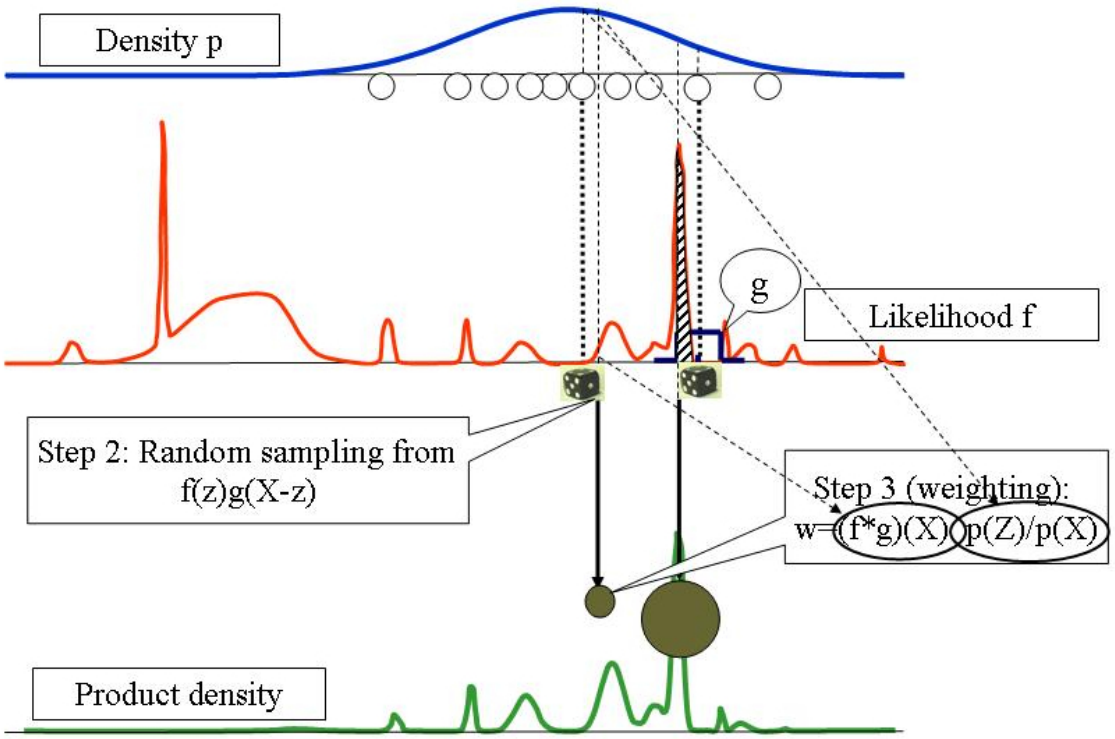


Figure 4.8: The visualization of the Local Likelihood Sampling (LLS) algorithm. The figure shows the sampling steps of the LLS algorithm. First, particles sampled from density $p$. The positions of the particles undergo a random perturbation that depends on the shape of likelihood $f$ in the neighborhood of the particle's position. In the figure dotted line positions changes to solid line positions. Weights are modified such that the sampling scheme remains unbiased.

The efficiency of LLS will depend on the correlation of $f$ and $p$: if the main modes of $p$ were far away from the main modes of $f$ then the scheme will be inefficient. Another source of inefficiency is when the support of $g$ is too small to move the samples to the vicinity of the essential modes of $f$ or when it is too large. In the limit when the support of $g$ grows to $\mathcal{X}$ the scheme gets worst than the likelihood sampling because of the variance is added in the calculation of importance weights. In the other limiting case when the support of $g$ shrinks to a single point then the procedure becomes identical to the Bootstrap Filter's naive sampling strategy.

In order to compare the efficiency of LLS with that of the Bootstrap Filter's naive sampling scheme we compare the variances of the weights of these two algorithms, as suggested by the Liu's rule of thumb discussed previously (see Section. 2.4). Note that, in general $E[w(X)] \neq 1$. However, defining $\tilde{w} = w/I(fp)$, we have $I_N(h, w) = J_N(h, \tilde{w})$. Hence, the efficiency of the scheme defined by $(X, w)$ is the same as the efficiency of the scheme defined by $(X, \tilde{w})$. Now, since $E[\tilde{w}(X)] = 1$, Liu's "rule of thumb" applies and we get that the relative efficiency of using $(X, \tilde{w})$ is

$$\frac{1}{1 + \mathrm{Var}[\tilde{w}(X)]} = \frac{1}{1 + \frac{\mathrm{Var}[w(X)]}{I(fp)^2}} ,$$

since $\mathrm{Var}[\tilde{w}(X)] = \frac{\mathrm{Var}[w(X)]}{I(fp)^2}$. Noting that in our applications $f$ and $p$ are fixed (do not change when comparing different schemes) we find that, according to the "rule of thumb" in order to maximize the efficiency of a scheme one should still try to minimize $\mathrm{Var}[w(X)]$. Since for any properly weighted pair $(X, w)$, $E[w(X)] = I(fp)$ and $\mathrm{Var}[w(X)] = E[w^2(X)] - E[w(X)]^2$ we find that minimizing $\mathrm{Var}[w(X)]$ is equivalent to minimizing $E[w^2(X)]$.

For the naive sampling scheme of the Bootstrap Filter we have seen (Equation 2.3) that $E[w_{\mathrm{N}}^2(X)] = \langle f,\, fp \rangle = I(f^2 p)$. It is easy to see that for the proposed sampling scheme:

$$\begin{aligned} E[w_{\mathrm{LLS}}^2] &= E[E[(w^{(j)})^2 \mid X^{(j)}]] \\ &= \int_{p>0} \int (f * g)(x) \frac{f(z)p^2(z)}{p^2(x)} g(x - z)\, dz\, p(x)\, dx \\ &= \int_{p>0} (f * g)(x) \frac{1}{p(x)} \int f(z) p^2(z) g(x - z)\, dz\, dx \\ &= \langle f * g,\, \frac{(fp^2) * g}{p} \rangle. \end{aligned}$$

At a first sight this formula might seem a little complicated, but the following proposition provides a nice reformulation and allows us to make comparisons between the performance of the naive method and that of LLS.

**Proposition 4.4.2.** *Assume that $g \in L^1$ is an even, window function satisfying $g \geq 0$ and $I(g) = 1$ and let $f$ be a bounded, integrable, nonnegative function and let $p > 0$ be a*

*density. Define the operator* $A : L^1 \to L^\infty$ *by*

$$(Ah)(u) = \int h(t)p(t)g(t-u)\left(\frac{p(t)}{p(u)} - 1\right) dt$$

*Assume that for some* $s \in [1, \infty]$,

$$\epsilon = \sup_{h \in L^1, h \geq 0} \sup_u \frac{(Ah)(u)}{\|h\|_s} < +\infty \,. \tag{4.3}$$

*Let* $(X, Z, w)$ *be a random sample as defined in the above algorithm. Then*

$$E[w_{LLS}^2] \leq \langle f * g,\, fp * g \rangle + \epsilon I(f)\|f\|_s. \tag{4.4}$$

*Proof.* Pick an arbitrary $x$. Using $(Af)(x) \leq \epsilon\|f\|_s$ we get

$$\int f(z)p(z)g(x-z)\frac{p(z)}{p(x)}\,dz \leq \int f(z)p(z)g(x-z)\,dz + \epsilon\|f\|_s = \big((fp) * g\big)(x) + \epsilon\|f\|_s \,.$$

Hence,

$$E[w_{LLS}^2] \leq \int (f * g)(x)\Big(\big((fp) * g\big)(x) + \epsilon\|f\|_s\Big)\,dx \,.$$

Now, since $I(g) = 1$ and $f, g$ are non-negative, an application of Fubini's theorem yields that $I(f * g) = I(f)$. Therefore, exploiting again that $f, p, g \geq 0$, we get the desired inequality:

$$E[w^2] \leq \langle f * g,\, (fp) * g \rangle + \epsilon I(f)\|f\|_s \,,$$

thus finishing the proof. □

It follows immediately that LLS is more efficient than the naive sampling scheme whenever

$$\epsilon I(f)\|f\|_s \leq \langle f,\, fp\rangle - \langle f * g,\, (fp) * g\rangle.$$

Clearly, this formula agrees well with the intuition that LLS works well if $f$ is peaky and $p$ is smooth: the right hand side is maximized, when the cross-correlation of $f$ and $fp$ is high and the cross-correlation of $f * g$ and $(fp) * g$ is small. Note that convolution with $g$ can be thought of as low-pass filtering (e.g. think about when $g$ is the characteristic function of the unit interval). Using Parseval's equality, and the Convolution Theorem, we get:

$$\langle f * g,\, (fp) * g\rangle = \langle F(f * g),\, F((fp) * g)\rangle = \int F(f)\overline{F(fp)}|F(g)|^2.$$

Hence $g$ cuts some of the high frequency of $f$ and $fp$. As a result, the cross-correlation of $f * g$ and $(fp) * g$ can be expected to be smaller than the cross-correlation of $f$ and $fp$:

$$\langle f,\, fp\rangle = \langle F(f),\, F(fp)\rangle = \int F(f)\overline{F(fp)}.$$

Now let us turn our attention to the left hand side of inequality (4.4). First, note that

$$\epsilon \leq \sup_{t,u} \frac{p(t)}{p(u)} g(t-u)(p(t)-p(u)) = \left(\sup_u p(u)\right) \sup_d \sup_t \frac{p(t)}{p(t+d)} g(d) \left(\frac{p(t)}{p(t+d)} - 1\right),$$

assuming that $g$ is any function satisfying the conditions of the previous proposition. Let $\alpha(d) = \sup_t \frac{p(t)}{p(t+d)}$. Clearly $\alpha(d)$ characterizes the smoothness of $p$ well. The above expression can be upper bounded if e.g.: $\alpha(d)$ stays close to 1 when $d$ is small, i.e: $p$ is locally smooth, and $\alpha(d)g(d)$ stays close to 0 if d is bigger, i.e.: $g$'s tail decays more than $\alpha(d)$ increases.

A more detailed analysis can be given is $g$ is compactly supported. Let $d > 0$ be such that $g(x) = 0$ whenever $\|x\| \geq d$, choose $s = 1$ and define

$$p_d(x) = \inf_{\|y\|<d} p(x+y).$$

Assume that $\sup_x p(x)/p_d(x) \leq +\infty$ and $g \leq \gamma$. Then $g(t-u)/p(u) \leq \gamma/p_d(t)$ and thus

$$p(t)g(t-u)\frac{p(t)-p(u)}{p(u)} \leq \gamma p^2(t)/p_d(t) - \gamma p_d(t) \leq \gamma \|p\|_\infty \|p/p_d\|_\infty,$$

hence $\epsilon \leq \|p\|_\infty \|p/p_d\|_\infty$. Hence $\epsilon$ is smaller when $p$ and $p_d$ stay close to each other, ie. if $p$ is sufficiently smooth.

As a summary we expect LLS to be more efficient than naive sampling whenever the observation density $f$ is peaky (has lots of high frequency energy) and the prediction density $p$ is smooth on the scale of the window function. In a typical visual tracking where the object is expected to make sudden movements the dynamics is typically chosen to have a large variance. Further, as it was already discussed above, a good observation likelihood function would separate the object from the background and the surrounding clutter sharply, hence one expects the observation likelihood function to be considerably high peaked. Thus, one expects LLS to give a considerable advantage over the naive sampling scheme in visual tracking tasks.

### 4.4.2 Inside LLS

Figure 4.9 illustrates the difference between the performance of LLS and the naive scheme in the simple scenario that we used previously to illustrate the possible inefficiency of the Bootstrap Filter. In this experiment $p$ was a single Gaussian and $f$ was a mixture of Gaussians; the graphs of $p$ and $f$ are shown in Figure 4.1. The product is shown on Figure 4.9. 50 samples were drawn using both methods and histograms were calculated. Since the number of particles is very small, both estimates are rather crude. However, it should be clear from the figure that LLS has a clear advantage over the naive sampling scheme in representing the highest peak of $pf$: the naive sampling scheme fails to allocate any particles in the vicinity of this peak hence it fails to capture its mass (note that for a sample size of 50 this happens with very high probability). Note that the simulation is set

up so that in the two cases the samples $X^{(1)}, \ldots, X^{(50)}$ were exactly the same: the advantage of LLS comes from its ability to modify these initial 'guesses' to obtain a 'refined' sample $Z^{(1)}, \ldots, Z^{(50)}$. Here the window function $g$ is a characteristic function with a support of size 51.

In order to obtain qualitative results we ran Monte-Carlo experiments to estimate the variance of the weights. In particular, our goal was to estimate the dependence of the representation's quality on the size of the window's support. Hence we changed the window size from zero to half of the scale of the support of $fp$ and for each window size tested we drew $500,000$ samples and computed the sample variance of the weights. The resulting graph is shown in Figure 4.10. Note that the variance of the weights of the naive sampling scheme correspond to sampling using LLS with a window-size of zero. The measured variance looks quasi-convex[2] and is quite flat around the optimal window size - such a behavior can be very advantageous in applications as it shows that the algorithm is quite robust to the window size parameter. Furthermore it might be possible to find a good window size by estimating the variance at a few window sizes and fitting e.g. a quadratic function. Given the graph we conclude that for a wide range of window sizes LLS is substantially more efficient than the naive sampling scheme, at least in this particular example.

### 4.4.3 LLS based Particle Filtering

The particle filtering algorithm, where the Local Likelihood Sampling scheme is used to sample from the product of the prediction density and the observation likelihood, is given in Figure 4.11.

Note that the algorithm can be generalized easily to use window functions that change their shape depending on the proposed samples $X_t^{(k)}$: One just needs to replace $g(X_t^{(k)} - x)$ in the sample-perturbation step by $g(X_t^{(k)} - x; X_t^{(k)})$ and redefine $\alpha_t^{(k)}$ accordingly: $\alpha_t^{(k)} = \int p(Y_t|x) g(X_t^{(k)} - x; X_t^{(k)})\, dx$. The simplest application of this is to match the size of the support of $g$ to the 'scale' information in $X_t^{(k)}$. Similarly $g$ can be made dependent on the observation $Y_t$. Also, when the state space is multi-dimensional, $g$ can be used to perturb the samples along certain selected dimensions only. As an example to illustrate for the usefulness of this, consider contour tracking. During the course of calculating the likelihood values, the derivative of the image is calculated in a small neighborhood of the support points of the splines. Hence, in this case perturbing the samples along the translation components can be implemented with almost no increase in the computation cost. On the other hand perturbing the samples along the scale and rotation components would require a substantial increase in the computational cost.

[2] A function is quasi-convex, if all its level sets are convex.

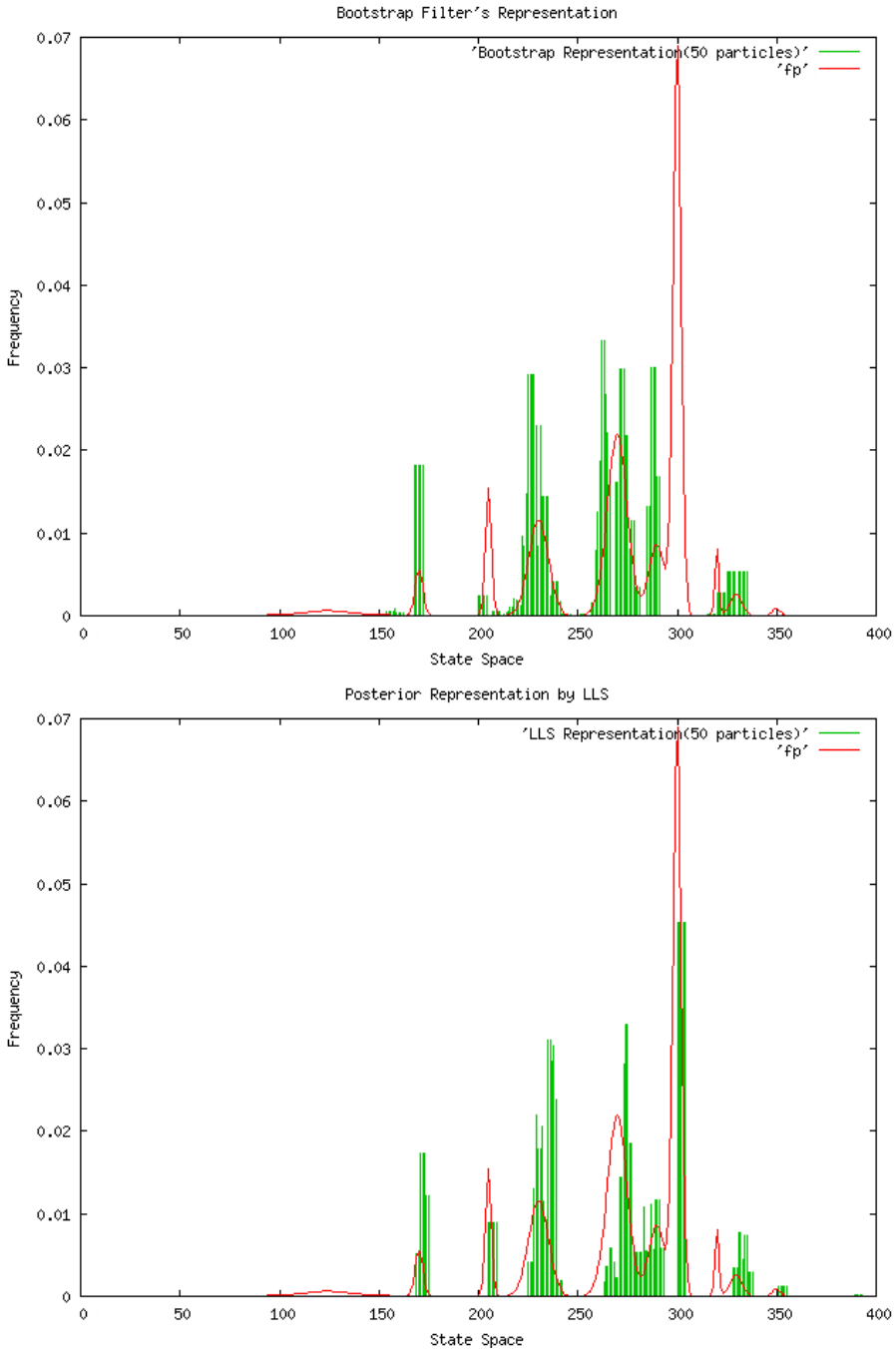


Figure 4.9: Representations of product densities given by the naive sampling scheme of the Bootstrap Filter and LLS. Shown is the product function $f\,p$ and the empirical measures corresponding to the two algorithms (top figure: naive sampling, bottom figure: LLS). The basic sample set is the same in both cases and has 50 elements. Note that the representation provided by the naive method fails to capture the highest peak of the product, whilst LLS succeeds at allocating a few particles to the peak.

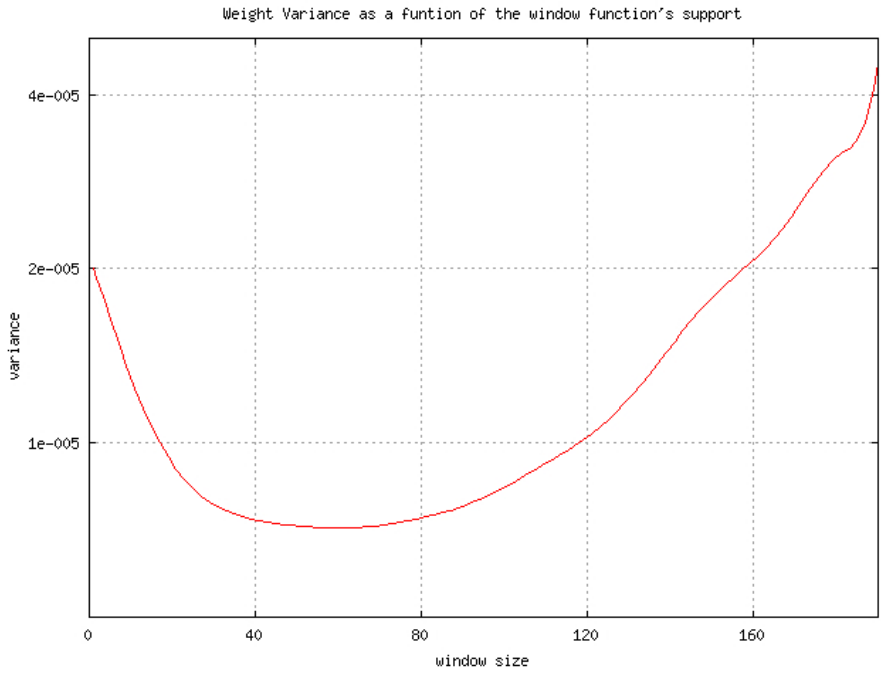


Figure 4.10: Variance of the weights as a function of the size of the window used in LLS. Smaller variance means better performance. For comparison, the variance of the naive sampling scheme corresponds to the variance shown at 0, i.e: $2 \cdot 10^{-5}$, the variance of the likelihood sampling is $1.25 \cdot 10^{-5}$, the LLS variance with the optimal window function is $7.15 \cdot 10^{-6}$. Note that in the worst case LLS is worse than likelihood sampling, because of the additional random sampling steps. In the experiments the window function was a characteristic function.

**Initialize** a sample set $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ according to the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(Z_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(Z_{t-1}^{(j)\prime}, 1/N)\right]_{j=1}^{N}$.

**For** $k = 1, 2, \ldots, N$

**Predict** $X_t^{(k)}$ by drawing a sample from $K(\cdot|Z_{t-1}^{k\prime})$.

**Perturb** $X_t^{(k)}$ by drawing $Z_t^{(k)}$ from $\frac{1}{\alpha_t^{(k)}} r(Y_t|\cdot) g(X_t^{(k)} - \cdot)$, where

$$\alpha_t^{(k)} = \int r(Y_t|x) g(X_t^{(k)} - x)\, dx \ .$$

**Update** the weight $w_t^{(k)}$ using

$$w_t^{(k)} = \alpha_t^{(k)} \frac{K(Z_t^{(k)}|Z_{t-1}^{(k)\prime})}{K(X_t^{(k)}|Z_{t-1}^{(k)\prime})} \ .$$

**End**

**Normalize** the weights using $w_t^{(k)} := w_t^{(k)} / \sum_{j=1}^{N} w_t^{(j)}$.

**End**

Figure 4.11: The LLS-based Particle Filter.

## 4.5 Local Importance Sampling

Despite the wide range of possibilities to choose $g$ it can happen that sampling from $r(Y_t|\cdot)g(X_t^{(k)} - \cdot)/\alpha_t^{(k)}$ is prohibitive. In this case it is worthwhile to introduce a proposal function $q_{X_t^{(k)}}(\cdot)$ that makes the implementation of sampling possible, efficient, and approximates $r$ locally well. The corresponding algorithm, called Local Importance Sampling [50, 51], is shown in Figure 4.12. Notice that with the choice $q_{X_j} = f$ the algorithm becomes identical to LLS.

**For** $i = 1, \ldots, N$

**Sample** $X^{(i)}$ from $p(\cdot)$.

**Draw** $Z^{(i)}$ from

$$Z^{(i)} \sim \frac{q_{X^{(i)}}(\cdot)g(X^{(i)} - \cdot)}{(q_{X^{(i)}} * g)(X^{(i)})}.$$

**Calculate importance weight**

$$w(Z^{(i)}) = w^{(i)} = (q_{X^{(i)}} * g)(X^{(i)}) \frac{f(Z^{(i)})}{q_{X^{(i)}}(Z^{(i)})} \frac{p(Z^{(i)})}{p(X^{(i)})}.$$

**EndFor**

Figure 4.12: The Local Likelihood Sampling (LLS) Algorithm to generate a particle representation from the product of $f$ and $p$. For any function $h \in L^1$ of interest, approximate $I(hfp)$ by $J_N = \sum_{j=1}^{N} w^{(j)} h(Z^{(j)})$.

### 4.5.1 Theoretical Analysis

The particle filtering algorithm that employs LIS in its main loop is shown in Figure 4.13. Building on the previous argument that showed that LLS is more efficient than the naive algorithm, one expects that LIS will also be more efficient under similar conditions provided that the proposal function $q_{X_i}$ fits $f(\cdot)$ locally at $X_j$. This is demonstrated in the next section in some experiments with tracking license plates of cars on video sequences.

Regarding the bias of the algorithms, the following proposition holds:

**Proposition 4.5.1.** *Let $(X_t, Y_t)$ evolve according to a tracking model described in Section 1.2 and consider the LIS filter. Assume that $g$ is a non-negative, integrable function satisfying $I(g) = 1$ and let $q_{x,y}(\cdot) > 0$ be a bounded, integrable function for all $x, y$. Let $\{(\hat{w}_t^{(k)}, Z_t^{(k)})\}_{k=1}^N$ be the particle set obtained at time step $t$. Then*

$$\mathbb{E}[\hat{w}_t^{(k)} h(Z_t^{(k)})|Y_{1:t}] = \mathbb{E}[h(X_t)|Y_{1:t}]p(Y_t|Y_{1:t-1}).$$

**Initialize** a sample set $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ according to the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(Z_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(Z_{t-1}^{(j)\prime}, 1/N)\right]_{j=1}^{N}$.

**For** $k = 1, 2, \ldots, N$

**Predict** $X_t^{(k)}$ by drawing a sample from $K(\cdot|Z_{t-1}^{k\prime})$.

**Perturb** $X_t^{(k)}$ by drawing $Z_t^{(k)}$ from $\frac{1}{\alpha_t^{(k)}} q_{X_t^{(k)},Y_t}(\cdot) g(X_t^{(k)} - \cdot)$, where

$$\alpha_t^{(k)} = \int q_{X_t^{(k)},Y_t}(x) g(X_t^{(k)} - x)\, dx \; .$$

**Update** the weight $w_t^{(k)}$ using

$$w_t^{(k)} = \alpha_t^{(k)} \frac{r(Y_t|Z_t^{(k)})}{q_{X_t^{(k)},Y_t}(Z_t^{(k)})} \frac{K(Z_t^{(k)}|Z_{t-1}^{(k)\prime})}{K(X_t^{(k)}|Z_{t-1}^{(k)\prime})} \; .$$

**EndFor**

**Normalize** the weights using

$$w_t^{(k)} := \frac{w_t^{(k)}}{\sum_{j=1}^{N} w_t^{(j)}}$$

.

**EndFor**

Figure 4.13: LIS-based Particle Filter

*Proof.* First note that $\alpha_t^{(k)} = (q_{X_t^{(k)},Y_t} * g)(X_t^{(k)})$. Let $h$ be an arbitrary integrable function and let $I = \mathbb{E}[\hat{w}_t^{(k)} h(Z_t^{(k)}) | Y_{1:t}]$. By the law of total probability, $I = \mathbb{E}[\mathbb{E}[\hat{w}_t^{(k)} h(Z_t^{(k)}) \mid X_t^{(k)}, Y_{1:t}]]$. By the definition of $Z_t$ and $\hat{w}_t$,

$$\begin{aligned}
&\mathbb{E}[\hat{w}_t^{(k)} h(Z_t^{(k)}) \mid X_t^{(k)}, Y_{1:t}] \\
= &\int h(z)(q_{X_t^{(k)},Y_t} * g)(X_t^{(k)}) \frac{r(Y_t|z)}{q_{X_t^{(k)},Y_t}(z)} \frac{K(z|Z_{t-1}^{(k)\prime})}{K(X_t^{(k)}|Z_{t-1}^{(k)\prime})} \frac{q_{X_t^{(k)},Y_t}(z) g(X_t^{(k)} - z)}{(q_{X_t^{(k)},Y_t} * g)(X_t^{(k)})} \, dz = \\
= &\int h(z) \frac{K(z|Z_{t-1}^{(k)\prime})}{K(X_t^{(k)}|Z_{t-1}^{(k)\prime})} \, r(Y_t|z) \, g(X_t^{(k)} - z) \, dz \, ,
\end{aligned}$$

and hence by Fubini's theorem

$$\begin{aligned}
I &= \int\int h(z) \frac{K(z|Z_{t-1}^{(k)})}{K(x|Z_{t-1}^{(k)})} \, r(Y_t|z) \, g(x - z) \, dz K(x|Z_{t-1}^{(k)}) \, dx \\
&= \int h(z) K(z|Z_{t-1}^{(k)\prime}) r(Y_t|z) \, dz \, ,
\end{aligned}$$

which equals the posterior multiplied by $p(Y_t|Y_{1:t-1})$, finishing the proof. □

Similarly to Proposition 4.4.1, this statement shows that the two-step sampling step of the LIS filter is unbiased. Hence, we can expect that LIS filters will enjoy similar theoretical properties as the Bootstrap Filter, or the more general SIR filters [13]. Building on our previous argument, it is not hard to show that LIS is more efficient than the Bootstrap Filter under conditions when LLS is more efficient than the Bootstrap Filter and when the proposal function $q_{x,y}$ fits $r(y|\cdot)$ around $x$ for any $x, y$. Instead of developing such a theoretical result[3], the efficiency of the new algorithm will be demonstrated in the following sections on a number of problems.

### 4.5.2 Implementing LIS for Japanese License Plate Tracking

Since we already defined the dynamics and a likelihood model in Section 1.3.2, we only need to specify the window function $g$ and the importance function in order to define a LIS-based particle filter for this problem.

We have chosen the perturbation to act only on the translation components (i.e. the perturbation does not change the scale and the orientation, nor does it change the 'history' components of the state). This can be expressed by making $g$ to have the form

$$g(u, w, \theta, u^{\text{prev}}, w^{\text{prev}}, \theta^{\text{prev}}) = g_0^\theta(u) g_1^\theta(w) \delta(\theta) \delta(u^{\text{prev}}) \delta(w^{\text{prev}}) \delta(\theta^{\text{prev}}),$$

where $\delta(\cdot)$ is the Dirac-delta function. The function $g_0^\theta$ ($g_1^\theta$) is a characteristic function of an interval centered around 0 with a length twice of the horizontal (vertical) size of the LP.

[3]It can be shown that the variance of weights in LIS is: $E_{LIS}[w^2] = \langle \frac{q*g}{p}, \frac{f^2 p^2}{q} * g \rangle$.

This means that in order to implement LLS the likelihood function $r(y|u, w, \theta)$ must be calculated for a set of values of $u, v$ centered around the predicted position of the plate's center. Unfortunately, the evaluation calculation of the likelihood function at a single point is already quite expensive.

The main idea is to write the proposal $q$ using Bayes-theorem in the form $q(u, w) = q(w)q(u|w)$ and to sample $(U', W')$ from $q$ by first drawing $W'$ from the marginal $q(w)$ and then drawing $U'$ from the conditional $q(u|W')$. The procedure we use is as follows: Assume that we are given a predicted pose $(U, W, \theta)$. Then let the perturbed translation components $(U', W')$ be sampled as follows:

1. Sample the new vertical position $W'$ from

$$q_X^h(Y_t|U, \cdot, \theta) = \frac{r^h(Y_t|U, \cdot, \theta)g_1(W - \cdot)}{\int r^h(Y_t|U, w', \theta)g_1(W - w')dw'}.$$

.

2. Sample the new horizontal position $U'$ from

$$q_X^v(Y_t|\cdot, W', \theta) = \frac{r^v(Y_t|\cdot, W', \theta)g_0(U - \cdot)}{\int r^v(Y_t|u', W', \theta)g_0(U - u')du'}.$$

Note that if the LP's width and height are $a$ and $b$, respectively, then this procedure evaluates $r^h$ $2b$ times, whilst $r^v$ is evaluated $2a$ times. Note that for a naive LLS implementation we would need to evaluate both of these likelihoods $4ab$ times, hence the proposed method scales linearly whilst LIS scales quadratically.

Will the above procedure be efficient in practice? The key observation here is that the likelihood $r^h(y|u, \cdot, \theta)$ is not sensitive to $u$, i.e. $r^h(y|u, \cdot, \theta) \approx r^h(y|u', \cdot, \theta)$ when $u$ and $u'$ are close to each other. This follows because LPs are horizontally elongated. Hence, if $q_{X,Y}(u, w|y, \theta)$ denotes the sampling distribution of $(U', W')$ then $q_{X,Y}$ will be highly correlated with $r(Y_t|\cdot, \cdot, \theta)$ in the neighborhood of $(U, W)$ (note that $U$ and $W$ are components of $X$).

It follows by inspection that the proposal function $q$ corresponding to the above procedure has the form:

$$q_{X,Y}(U', W') = q_X^h(Y_t|U, W', \theta)q_X^v(Y_t|U', W', \theta).$$

Straightforward calculations show that $\alpha_t = (q_{X,Y_t} * g)(X) = 1$. Hence, it follows that the weight $w$ associated with a perturbed particle $Z = (U', W', \ldots)$ is

$$w = \frac{r(Y|U', W', \theta)}{q_X^h(Y_t|U, W', \theta)q_X^v(Y_t|U', W', \theta)} \frac{K(U', W', \theta|U^{\text{prev}}, W^{prev}, \theta^{prev})}{K(U, W, \theta|U^{\text{prev}}, W^{\text{prev}}, \theta^{\text{prev}})}.$$

### 4.5.3 License Plate Tracking Results

The performance of the LIS-based filter with $N = 100$ particles was compared to the performance of the Bootstrap Filter that used $N = 750$ particles. The number of particles was determined in preliminary experiments with the goal to match the running times of the two filters. Actually it turned out that we have slightly overestimated the running time of the LIS-based particle filter. In particular, on our 1.7GHz Intel test machine we have measured a processing speed of approximately 48 frames per second for the Bootstrap Filter (with $N = 750$), whilst for the LIS-based particle filter we have measured a processing speed of approximately 65 frames per second (using $N = 100$).

Performance evaluation was done as follows: We have selected a video sequence at random from our collection of samples. The test sequence is composed of 298 frames. 'Ground truth' was obtained by running the Bootstrap Filter for the test video sequence with a larger number of particles and then correcting the results manually. In each time step particle locations were averaged to get a point estimate of the LP's position. This position was compared to the ground truth. Some frames of this sequence are shown in Figure 4.14 together with the plate positions estimated by the LIS based particle filter and projected back onto the image.

When evaluating the precision of tracking we had to calculate the distance of the point estimates of the LP-configurations to the true configurations. This we implemented by computing the sum of distances of their corresponding vertex points. Furthermore, when we evaluated the reliability of tracking we declared the LP as 'lost' if the distance of the estimated LP configuration to the true one was bigger than one third of the true LP's height. The probability of this event was estimated for each frame by means of running a large number of Monte-Carlo experiments. Average tracking error was measured for the rest of the cases.

Figure 4.15 shows the probability that the LP is lost on the test sequence as a function of the frame number. It should be clear from this figure, that the LIS-based particle filter tracks the license plates significantly more robustly than the Bootstrap Filter. The error of tracking is shown in Figure 4.16. Note that not all frames contain a LP (e.g. the frames around 50 and 150 do not contain any LPs). For these frames tracking error was artificially reset to zero. Results depicted on Figure 4.16 indicate that despite the facts that the LIS based particle filter uses a smaller number of particles, runs faster than the Bootstrap Filter, and track LPs more reliably than the Bootstrap Filter, it can also yield the same or better tracking performance than the Bootstrap Filter. Hence, we conclude that the proposed algorithm does improve upon the Bootstrap Filter, at least in the particular example studied here.

### 4.5.4 Using Gauss-Mixture Proposals in LIS Filters

In this section we will show that if in LIS the window function is chosen to be a Gaussian and the proposal function is chosen to be a mixture of Gaussians at the same time, we get a particularly attractive algorithm [50]. Since continuous densities can be well-approximated

Figure 4.14: Sample images of the test video sequence. The video sequence is recorded by a commercial NTSC camera. The frame indexes of the images are 9, 29, 82, 105, 117 and 125. The red rectangles on the image correspond to the estimated LP position using the proposed LIS-based particle filter.

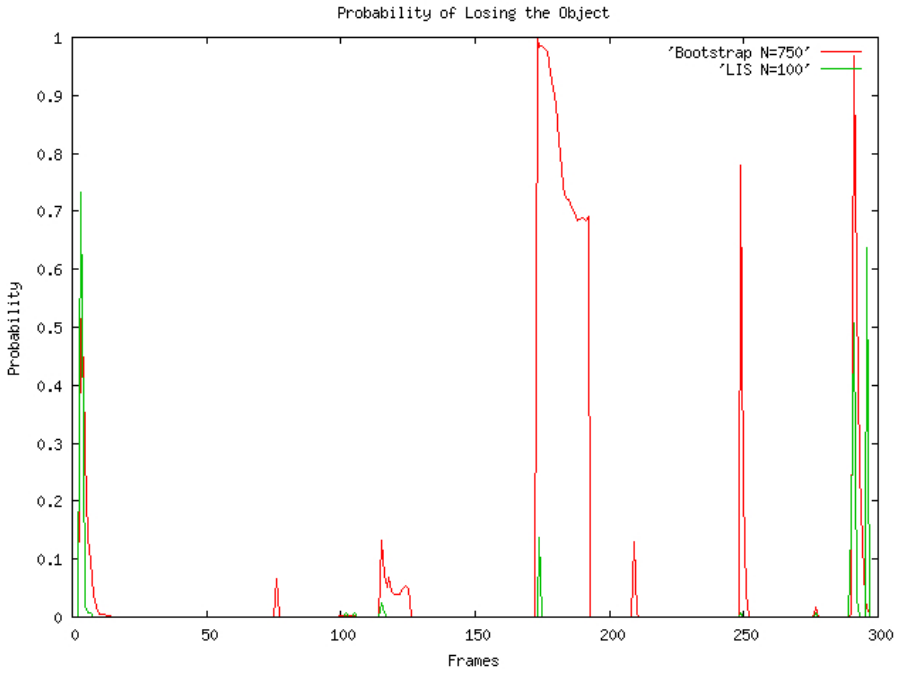


Figure 4.15: Probability of not tracking the license plate as a function of the frame index. Note that the Bootstrap Filter tends to lose the LP at difficult frames. LIS filter's tracking reliability is much better than the Bootstrap Filter's, despite that it uses less particles and runs faster than the Bootstrap Filter.

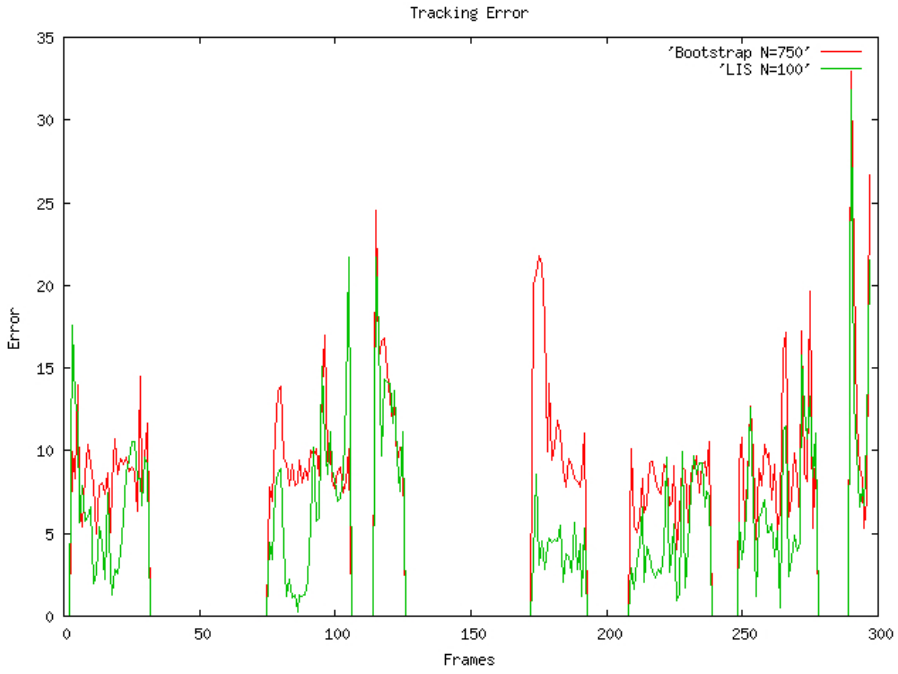


Figure 4.16: Average tracking error as a function of the frame index for the Bootstrap Filter and the LIS filter. Tracking error is estimated for those cases only when the estimated LP is closer to the true one than a certain threshold.

to any error by mixtures of Gaussians [42, 38] the resulting algorithm retains it generality. Further, as we show it in the followings, the algorithm can also be implemented efficiently.

Let $u = (x, y)$ and choose $q_{x,y} = q_u$ to be a mixture of Gaussians with $n$ components, having priors $p_{u,1}, \ldots, p_{u,n}$, means $\mu_{u,1}, \ldots, \mu_{u,n}$ and covariance matrices $\Sigma_{u,1}, \ldots, \Sigma_{u,n}$. Then

$$q_u(z) = \sum_{i=1}^{n} p_{u,i} \frac{e^{-1/2(z-\mu_{u,i})^T \Sigma_{u,i}^{-1}(z-\mu_{u,i})}}{((2\pi)^N |\Sigma_{u,i}|)^{1/2}}. \tag{4.5}$$

Let the window function be a zero-mean Gaussian with variance $\Sigma_g$:

$$g(z) = ((2\pi)^N |\Sigma_g|)^{-1/2} e^{-1/2 z^T \Sigma_g^{-1} z}. \tag{4.6}$$

In order to implement the LIS filter one needs to be able to draw samples from $q_u(\cdot)g(x-\cdot)$ and to evaluate $(q_u * g)(x)$. For the above specific choices, it turns out (see appendix D for the multiplication of Gaussian densities) that $q_u(\cdot)g(x-\cdot)$ is a mixture of Gaussians, too with covariances and means defined by the following equations:

$$C_{u,i} = (\Sigma_{u,i}^{-1} + \Sigma_g^{-1})^{-1}, \tag{4.7}$$

$$\nu_{u,i} = C_{u,i}\Sigma_g^{-1} x + C_{u,i}\Sigma_{u,i}^{-1}\mu_{u,i}, \tag{4.8}$$

and un-normalized weights:

$$L_{u,i} = p_{u,i} \frac{e^{-1/2(\mu_{u,i}-x)^T \Sigma_{u,i}^{-1} C_{u,i} \Sigma_g^{-1}(\mu_{u,i}-x)}}{((2\pi)^N |\Sigma_{u,i}||\Sigma_g|/|C_{u,i}|)^{1/2}}. \tag{4.9}$$

Let $L_u = \sum_{i=1}^{n} L_{u,i}$. Notice that $(q_u * g)(x) = L_u$. Further, sampling from $q_u(\cdot)g(x-\cdot)$ can be implemented by first drawing an index from the normalized weights $(L_{u,1}/L_u, \ldots, L_{u,n}/L_u)$ and then drawing a sample from the appropriate Gaussian. The corresponding algorithm is shown in Figure 4.17.

### 4.5.5 Experiments on the Bearings Only Problem

The purpose of this section is to present the results of a series of experiments where the LIS filter is compared with the baseline Bootstrap Filter and the Auxiliary Variable particle filter(AVPF, see appendix B), the purpose being to systematically compare LIS with these other algorithms in a controlled environment. Actually, we used two versions of the bearings-only tracking problem: The standard single object version where the problem is to track a single ship by using angular measurements only and a version with three ships, where they move independently of each other and the observations carry information about the identity of the ships that they originate from. The purpose of considering this second problem was to study the scaling properties of the algorithms studied as a function of the dimensionality of the state space.

In the experiments, where three ships are to be tracked, the initial state is sampled from three Gaussians. For all of these Gaussians we use the covariance matrix given in

Section 1.3.3. The means of the Gaussian belonging to first ship are identical to the means used in the single-ship experiments, whilst the means of the of the other Gaussian were $(0.02, -0.01, 0.6, -0.055)$ and $(0.05, -0.01, -0.2, -0.02)$.

In the multi-ship experiments the three ships move independently of each other. Further, unlike in multi-target tracking, we assume that the observations are not unordered, i.e., we do not consider here the ambiguity of the assignment of observations to the objects. Formally, if $y_1, y_2, y_3$ are the observed angles, $\theta_1$,$\theta_2$ and $\theta_3$ are the angles corresponding to the positions of ships one, two and three then

$$r_3(y_1, y_2, y_3|\theta_1, \theta_2, \theta_3) = r(y_1|\theta_1)r(y_2|\theta_2)r(y_3|\theta_3).$$

It should be clear that this model is limited in the sense that in many cases one would never know the correspondence between the observations and objects. In fact, the major source of difficulty for real-world multi-target tracking lies in resolving this ambiguity. However, our focus here are the scaling properties of the algorithms as a function of the dimensionality of the system and for this purpose the above problem is a just good enough. We leave it for the next chapter to extend it to more realistic multi-object tracking problems.

**The Choice of the Proposal for LIS**

We first discuss the choice of the proposal $q_{x,y}$ for the single ship tracking task. The idea underlying the design is that the observation is simple: the predicted particle positions should be adjusted to fit closely the observed angles as the angular measurements are reliable. The actual choice of the importance function $q_{x,y}$ is a Gaussian with one of its axes parallel to the observation angle. The mean of $q_{x,y}$ was set to the particle's position projected to the observation angle line. To be more exact $q_{x,y}$ is a Gaussian with mean $(x_1 \cos^2(y) + x_3 \sin^2(y))$, where $y$ is the observed angle, whilst the covariance is set to $U\Lambda U^T$ with

$$U = \begin{pmatrix} \cos(y) & -\sin(y) \\ \sin(y) & \cos(y) \end{pmatrix}$$

and

$$\Lambda = \begin{pmatrix} \kappa\sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}.$$

Here $\kappa > 0$ is a design parameter defining the ratio of the variance along the axis parallel to the observed angle to the variance along the orthogonal axis. Its value is chosen arbitrarily to be 100. A reasonable value for $\sigma^2$, the variance along the orthogonal axis, is the value that makes the Gaussian fit the Wrapped Cauchy observation density the best. We used this value in our experiments.

As $U$ is independent of the particles' positions, $U$ can be pre-computed at the beginning of the sampling steps, leaving only the calculation of the Gaussians' means in the body of the loop. The window function $g$ is defined as a zero-mean Gaussian with covariance matrix

$$C = \begin{pmatrix} \delta_g & 0 \\ 0 & \delta_g \end{pmatrix}.$$

It is easy to see that these choices enable us to use the Gaussian-mixture LIS (cf. Figure 4.17).

We must remark that the proposal function and the window function as defined above depend only on the position of the ship. Hence, when a particle's state is adjusted in the second sampling step, its velocity component must be appropriately adjusted (in a deterministic manner).

**Results for Single Ship Tracking**

In the rest of this section we compare these filters in a qualitative manner. The tracking error defined as the Euclidean distance between the mean predicted and the actual ship positions was used as the basis of the measurements. Measurements were made along trajectories of length 10, following the literature. The tracking errors were measured with 10 independent state-observation sequences, whilst keeping the initial position fixed. Unless otherwise noted, each result is the average of 100 runs for these 10 sequences.

The simplest way to compare particle filters is by their tracking error whilst the number of particles is kept the same. Figure 4.18 shows such results as a function of time steps. Here the number of particles is kept at 100 for all algorithms. As expected, AVPF performs better than the baseline Bootstrap Filter. LIS improves upon the performance of both the Bootstrap Filter and AVPF quite significantly.

In order to develop an understanding of what these performance differences mean, we present 'particle clouds' generated by the three algorithms for an arbitrary selected time-step in Figure 4.19. It should be clear from the figure that the particle set generated by LIS is much better concentrated along the lines pointing towards the true state than the sets generated by both AVPF and the Bootstrap Filter. Literally, LIS makes a better use of the available information. Moreover, the particle sets generated by AVPF are more concentrated around the true state than those generated by the Bootstrap Filter. We note that a straightforwardly implemented likelihood sampling algorithm would perform much weaker than any of these algorithms as it would have no clue about the distance of the ship, and thus it would need to distribute samples evenly along the measurement lines.

The running time of all the considered algorithms scales linearly with the number of particles. Hence in real-time applications where the per iteration time is limited, the running time will limit the number of particles that can be used. Since the per particle cost of the algorithms is different, the slower algorithms' performance will suffer more from real-time constraints. Comparing the computational cost of algorithms is not easy. Here we provide both a detailed analysis of the per particle computations for each of the algorithms, as well as the results of some empirical results. Table 4.2 shows a detailed account for the various elementary computational steps for the algorithms considered. This table can serve as the basis of predicting the running times of the various algorithms. For example, in computer vision applications the evaluation of the observation density is the far most expensive step as it involves calling the actual image processing routines. In other applications (like the one considered here) the various steps have roughly the same cost. In such a case, LIS with $N$ particles can be expected to be cheaper to execute than the

Bootstrap Filter with $M$ particles or AVPF with $K$ particles if $3.5N < M$, and respectively, $1.75N < K$. The next table (Table 4.3) shows the actual measured running times of the

| | **BPF** | **AVPF** | **LIS** | **IS** |
|---|---|---|---|---|
| Sampling from the dynamics | 1 | 2 | 1 | 0 |
| Evaluating the obs. density | 1 | 2 | 1 | 1 |
| Evaluating the dyn. density | 0 | 0 | 2 | 1 |
| Sampling from the proposal | 0 | 0 | 1 | 1 |
| Proposal Density Evaluation | 0 | 0 | 1 | 1 |
| Preprocessing | 0 | 0 | 1 | 0 |

Table 4.2: Per-particle computation steps of the Bootstrap Filter (BPF), AVPF, LIS and Importance Sampling(IS). Preprocessing for LIS is the calculation of the means of two Gaussians. This boils down to computing two matrix-vector products, implementable with 8 multiplications.

algorithms.[4] The codes of the three algorithms were written in C++ and all of them have reasonable implementations. No special effort was made, except those already mentioned, to optimize the codes of the algorithms. The table shows both the total CPU time when the number of particles is the same for all the three algorithms and the total CPU time when the particle numbers are set so that the errors of the algorithms are roughly equal (see Figure 4.20). Given this table, we may conclude that LIS is roughly 12.5 times slower

| | **BPF** | **AVPF** | **LIS** |
|---|---|---|---|
| CPU time with 10000 particles | 122ms | 590ms | 1523ms |
| CPU time with equal error | 36.6ms N=3000 | 29.5ms N=500 | 15.2ms N=100 |

Table 4.3: Measured running times of the Bootstrap Filter (BPF), AVPF and LIS with equal particle numbers and equal errors ($N$ is the number of particles)

than the Bootstrap Filter and is roughly 2.6 times slower than AVPF, thus the theoretical predictions are off by a factor of 3.6 and 1.5, respectively. Despite this when the tracking

[4]The machine used was a Mobile Intel Celeron 2.5GHz with 256 MB RAM

errors are kept equal we get that LIS is the winner (in terms of execution time), followed by AVPF and the Bootstrap Filter.

Figure 4.21 shows the tracking error and the deviations of the errors of the three algorithms in the 10th time step for a number of sample sizes. As expected, the error decays (although not very rapidly) as the number of particles is increased for all the three algorithms. Interestingly, LIS keeps a considerable margin over the other algorithms over the range investigated, though its gain, by the nature of the problem studied, decreases when the number of particles is increased.

In the above experiments the standard deviation of the window function $\sigma_g$ was set to 0.0005 and the ratio parameter $\kappa$ of the proposal was set to 100. In order to test the sensitivity of LIS to these parameters we experimented with a number of values for these parameters. First, the window parameter was changed. This resulted in no significant changes in the performance as long as the parameter was kept in a reasonable range. It should be clear, however, that if the window size is too small then LIS degrades to the Bootstrap Filter. As remarked earlier, smaller values of $\kappa$, the parameter that governs how much we trust in the predicted distance of the ship, were found to yield to enhance performance. As $\kappa$ and the window size both grow to infinity the algorithm degrades to likelihood sampling.

### Results for Tracking Multiple Ships

Several real-world tasks require tracking of objects in high-dimensional spaces. It is well-known that the Bootstrap Filter suffers from an exponential breakdown as the dimensionality of the state space is increased. This holds even when the state variables evolve independently of each other. In this section we study the performance of the algorithms for the simplified multi-object tracking problem that was described earlier. Formally, for simplicity we assume that the observation likelihood for the observations $Y_1, \ldots, Y_M$ (assuming $M$ ships) is of the product form:[5]

$$r(Y_1, \ldots, Y_M | \theta_1, \ldots, \theta_M) = \prod_{k=1}^{M} r(Y_k | \theta_k).$$

Clearly, when $M$ ships are tracked, the dimension of the state becomes $4M$. In reporting the errors the Euclidean distance of the mean predicted and actual positions is divided by the number of ships tracked, so as to allow a meaningful comparison of results obtained with differing ship-numbers. Exploiting the simple structure of the problem, the proposal of LIS is chosen to take a product form, just like the window functions.

In the first set of experiments 3 ships were tracked, resulting in a state-space of dimension 12. Figure 4.22 shows the tracking errors of the algorithms as a function of time. These results were obtained with 10 tracking sequences and 100 runs for each sequences. Compared with Figure 4.18, we observe that the advantage of LIS against the other algorithms increases considerably.

[5]Of course, this assumes that correspondence between the observations and the ships has been resolved.

In order to gain further insight into the relative efficiency of these algorithms we present results for the equal CPU time case, as well. Table 4.4 serves as the basis for computing the respective particle numbers. We note that as compared with the results for tracking a single ship, the execution times for the Bootstrap Filter and AVPF are doubled only, whilst that for LIS are tripled. The better than expected execution times for the Bootstrap Filter and AVPF are a bit of surprising. We conjecture that some low-level mechanisms (caching, loop unrolling) might have caused this differences. Based on these results, the

| | **BPF** | **AVPF** | **LIS** |
|---|---|---|---|
| CPU time with 10000 particles | 245ms | 796ms | 4597ms |
| CPU time with equal error | 245ms N=10000 | 238ms N=3000 | 4.6ms N=10 |

Table 4.4: Measured running times of the Bootstrap Filter (BPF), AVPF and LIS with equal particle numbers and equal errors for tracking 3 ships ($N$ is the number of particles)

number of particles in the subsequent experiments was set to 10, 50 and 100, respectively. Figure 4.23 shows the resulting tracking errors as a function of time steps. LIS again clearly performs better than the other algorithms, despite that it uses 10 particles only. Figures 4.24–4.26 plot the particle clouds for the three algorithms. Note that on these figures a single particle is represented by 3 points. We also remark that the horizontal and vertical scales are different in these figures. This creates the (wrong) impression that the estimated posterior's variance in the horizontal direction for both the Bootstrap Filter and AVPF were larger than the variance in the vertical direction.

Figures 4.27–4.29 show the sample paths of the ships together with the mean predicted positions for a given sequence. In these plots the number of particles are 1000 for the Bootstrap Filter, 200 for AVPF and 10 for LIS. Visual inspection reveals that LIS indeed makes a better use of the available information in the observations. In fact, the figures show that the error of tracking of the second ship becomes overly large for the Bootstrap Filter, whilst for AVPF the error becomes somewhat large both for the second and the first ship. Although the tracking error towards the last steps increases for ship 2 for LIS, LIS's errors are still much smaller than those obtained for the other algorithms for all ships and almost all time steps.

For the sake of completeness, the average tracking error with the corresponding standard deviations are plotted in Figure 4.30 as a function of the number of particles. The figure confirms that tracking errors decrease with increasing the number of particles. Again, LIS is able to keep its margin over the range investigated.

Figure 4.31 compares the tracking error and its standard deviation for LIS and the Bootstrap Filter when the number of objects to be tracked is increased from 2 to 20. As

noted before, the Euclidean distance is normalized by the number of ships so as to allow a meaningful comparison between results of tracking when the object numbers are different.[6] In these experiments the ships' initial positions were set systematically with equal spacings along a circle with a fixed radius and setting the initial velocity direction to the tangent of the circle at the initial point. We remark that when the number of ships is 20, the state-space is 80 dimensional.

In these experiments only the Bootstrap Filter and LIS were compared. As it can be observed from the figure, the performance of the Bootstrap Filter degrades as the dimensionality is increased, whilst the performance of LIS stays steady. The degradation of the performance of the Bootstrap Filter is not as severe as one would expect due to the smoothness of the dynamics of the system (for less smooth systems the error might blow up exponentially).[7] We think that it is quite encouraging that the tracking error of LIS is not effected by the increased dimensionality, raising the hope that LIS could be used as a basis of efficient particle filters that are able to track very high-dimensional systems with a fairly high precision.

[6] This normalization causes the decrease of the standard deviations.

[7] In fact, we suspect that the errors at $N = 20$ are very close to the error level that can be obtained by pure prediction given the initial state and not taking into account the observations.

**Initialize** a sample set $\left[(Z_0^{(k)}, 1/N)\right]_{k=1}^{N}$ according to the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Resample** from $\left[(Z_{t-1}^{(j)}, w_{t-1}^{(j)})\right]_{j=1}^{N}$ if needed, to obtain $\left[(Z_{t-1}^{(j)\prime}, 1/N)\right]_{j=1}^{N}$.

**For** $k = 1, 2, \ldots, N$

**Predict** $X_t^{(k)}$ by drawing a sample from $K(\cdot|Z_{t-1}^{k\prime})$. Let $u = (X_t^{(k)}, Y)$.
**Calculate the Gauss-mixture parameters** of $q_u(\cdot)g(X^{(k)} = \cdot)$

$$C_{u,i} = (\Sigma_{u,i}^{-1} + \Sigma_g^{-1})^{-1}, \quad \nu_{u,i} = C_{u,i}\Sigma_g^{-1}x + C_{u,i}\Sigma_{u,i}^{-1}\mu_{u,i}$$
$$L_{u,i} = p_{u,i}\frac{e^{-1/2(\mu_{u,i}-x)^T\Sigma_{u,i}^{-1}C_{u,i}\Sigma_g^{-1}(\mu_{u,i}-x)}}{((2\pi)^N|\Sigma_{u,i}||\Sigma_g|/|C_{u,i}|)^{1/2}}$$

**Draw an index** $k$ from $\{L_{u,i}/L_u\}_{i=1}^{n}$.
**Draw** $Z_j$ from a Gaussian with mean $\nu_{u,k}$ and covariance $C_{u,k}$.
**Update** the weight $w_t^{(k)}$ using

$$\hat{w}_t^{(k)} = \left(\sum_{i=1}^{n} L_{u,i}\right)\frac{f(Z_j)}{q_u(Z_j)}\frac{K(Z_j|X)}{K(X_j|X)}.$$

**EndFor**

**Normalize** the weights using

$$w_t^{(k)} := \frac{\hat{w}_t^{(k)}}{\sum_{j=1}^{N}\hat{w}_t^{(j)}}$$

.

**EndFor**

Figure 4.17: Local Importance Sampling with mixture of Gaussians proposals and Gaussian window function ($q_{x,y}(z)$ is defined by (4.5), and $g(z)$ is defined by (4.6))

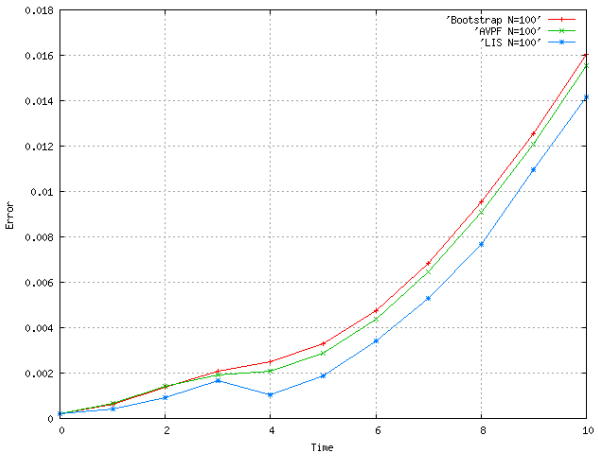


Figure 4.18: Tracking errors of the Bootstrap Filter, AVPF and LIS as a function of time. In these experiments the number of particles was kept fixed at the same value (here 100).

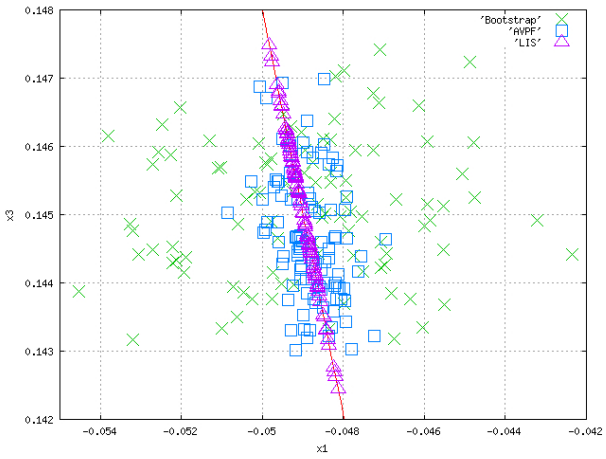


Figure 4.19: Particle clouds generated by AVPF ('blue □'), the Bootstrap Filter ('green ×') and LIS ('purple △') for an arbitrary selected time step.

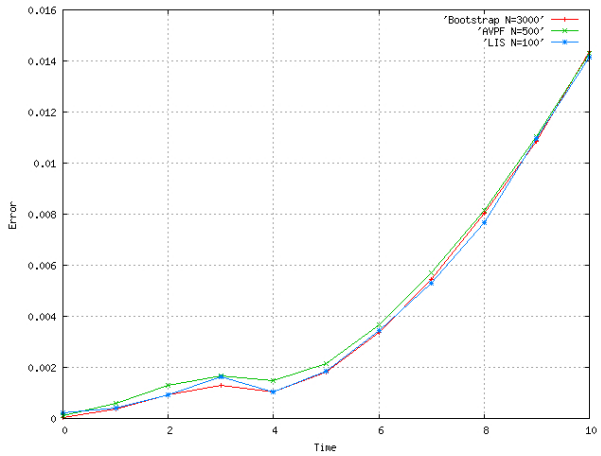


Figure 4.20: Tracking errors of the Bootstrap Filter, AVPF and LIS as a function of time. The particle sizes are set such that the errors are roughly equal. It can be seen from the figure that the performance of all three algorithms is the same uniformly in time.

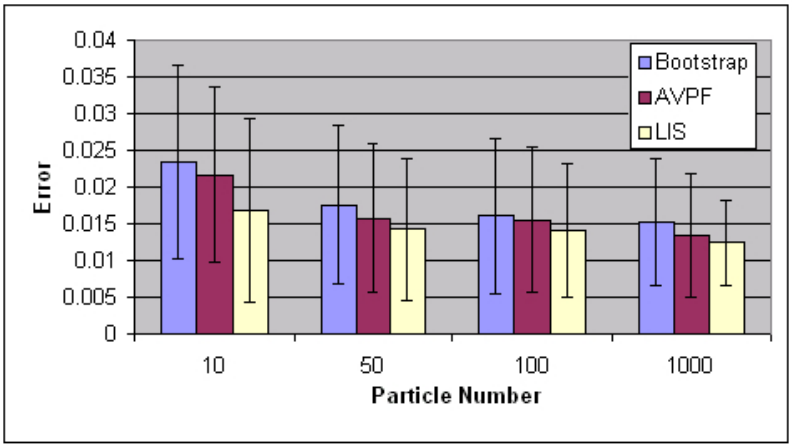


Figure 4.21: Tracking errors of the Bootstrap Filter, AVPF and LIS and the deviation of the error as a function of the number of particles

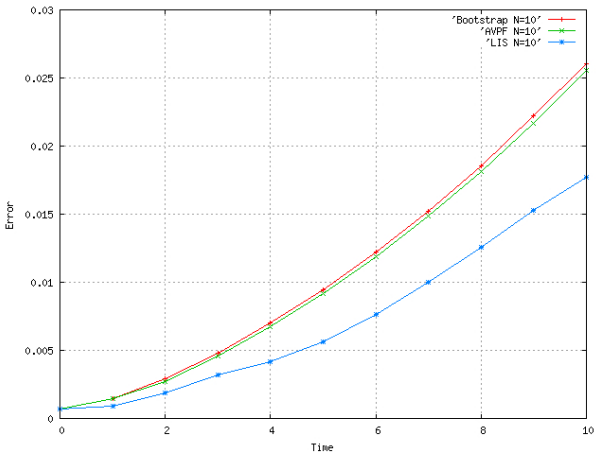


Figure 4.22: The average tracking error of the Bootstrap Filter, AVPF and LIS on the multi-object tracking problem with 10 particles

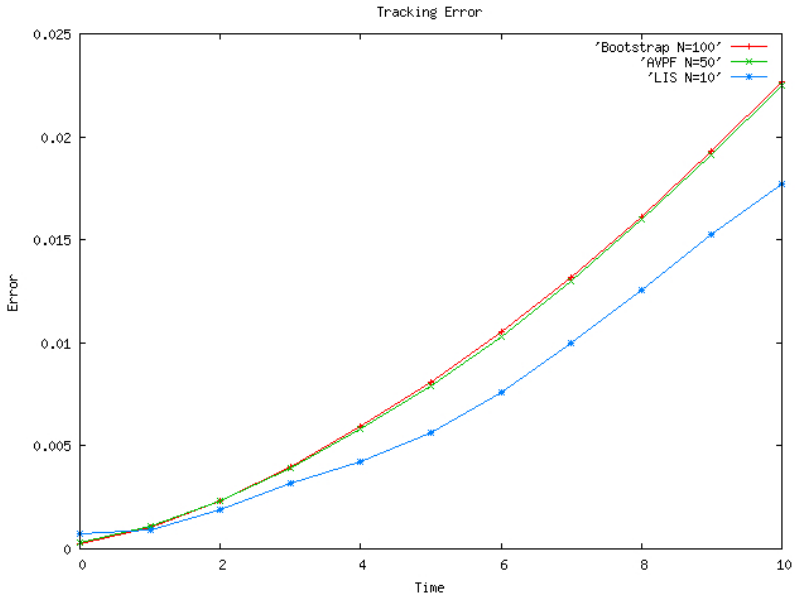


Figure 4.23: Average tracking error for the Bootstrap Filter, AVPF and LIS when tracking 3 ships. The number of particles are set to 100, 50 and 10, respectively so as to ensure that the algorithms' running times are the same.

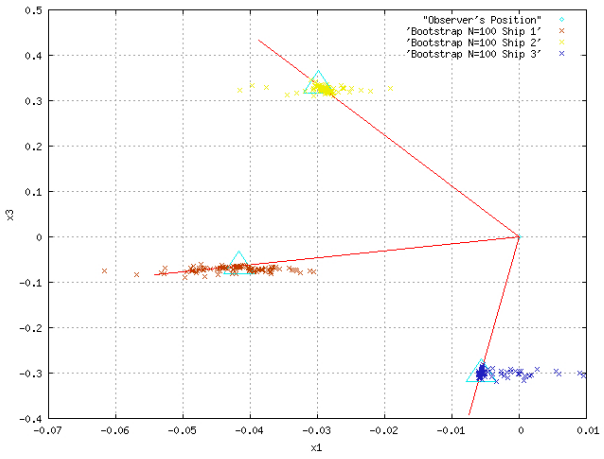


Figure 4.24: Illustration of the posterior representation of the Bootstrap Filter when 3 ships are tracked simultaneously. The number of particles is 100. The ships' positions are represented by larger circles. The figure also shows the straight lines connecting the observer's position with positions of the ships.

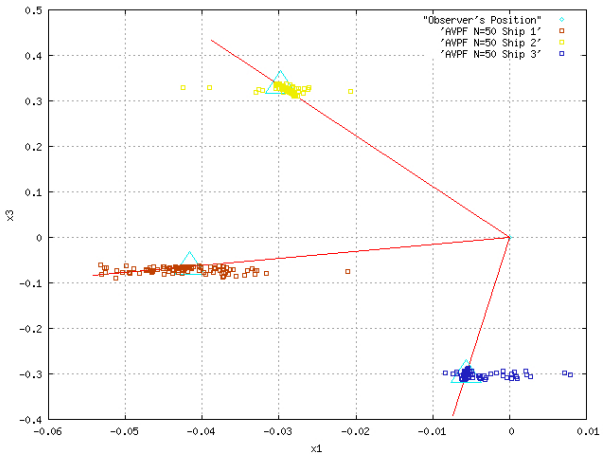


Figure 4.25: Illustration of the posterior representation of AVPF when 3 ships are tracked simultaneously. The number of particles is 50.

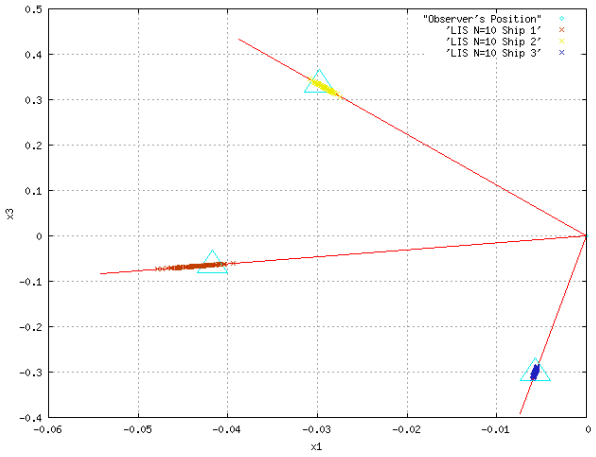


Figure 4.26: Illustration of the posterior representation of LIS when 3 ships are tracked simultaneously. The number of particles is 10.

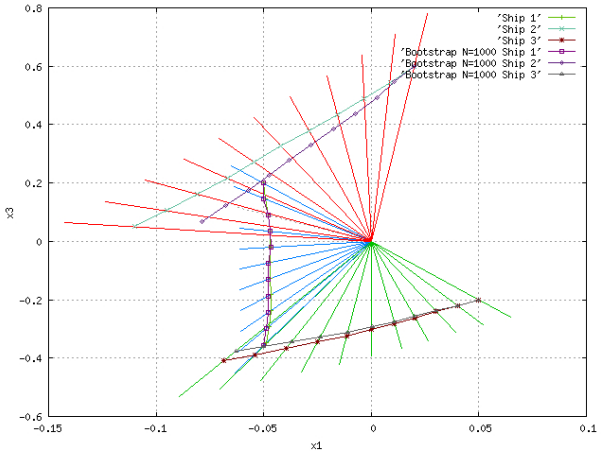


Figure 4.27: The ships' trajectories and the corresponding mean predicted positions of the 3 ships for the Bootstrap Filter with 1000 particles

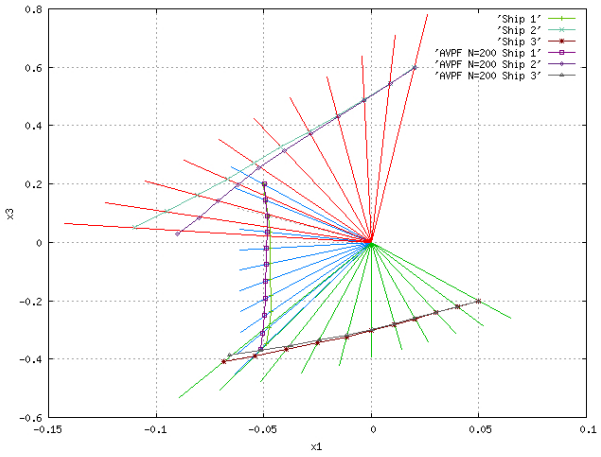


Figure 4.28: The ships' trajectories and the corresponding mean predicted positions of the 3 ships for AVPF with 200 particles

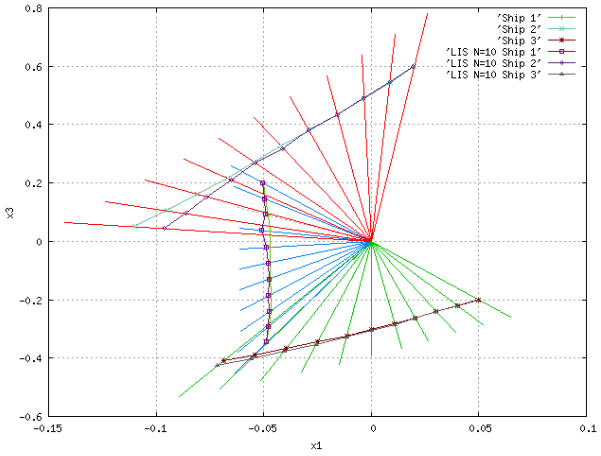


Figure 4.29: The ships' trajectories and the corresponding mean predicted positions of the 3 ships for LIS with 10 particles

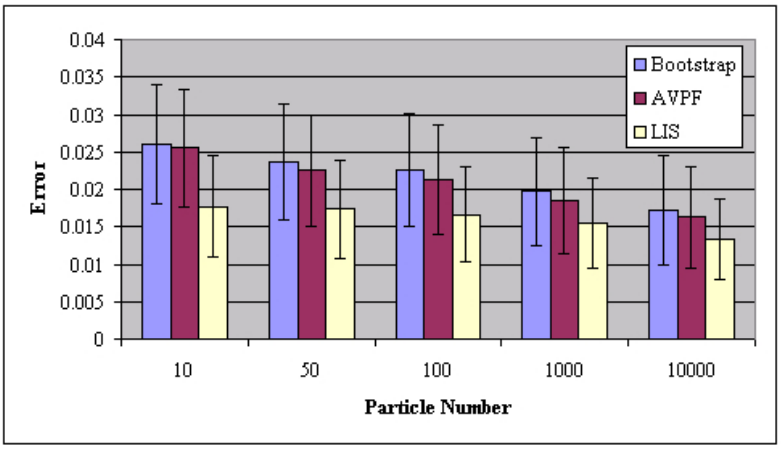


Figure 4.30: Tracking errors of the Bootstrap Filter, AVPF and LIS and the deviation of the error with different particle sizes when tracking 3 ships

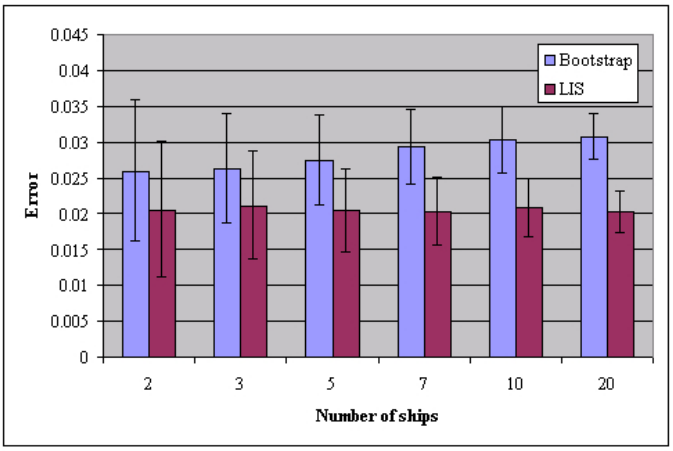


Figure 4.31: Tracking error for time step 10, as a function of the number of ships to be tracked for the Bootstrap Filter and LIS. Both algorithms use the same number of particles. The statistics is obtained by running 100 experiments for 10 independent realizations.

# Chapter 5

# Summary

In this Thesis particle filtering algorithms were studied from the point of view of generating good particle representations of the posterior. *The key observation is that the positions of the particles play a crucial role in the quality of the estimates.* This implies that the particles should be positioned using as much of the available information as possible. In particular, both the dynamical prediction density and the observation likelihood function should be taken into account when positioning the particles. In fact there are very few algorithms in the literature that make a good use of both models at this stage. In this Thesis three families of algorithms are suggested and analyzed that subscribe to this idea. These algorithms are the main contributions in this Thesis. They are designed to work in different situations. These situations together with the proposed algorithms can be summarized as follows:

**Thesis 1: History Sampling for SIR** (Section 3.2). Importance sampling is an efficient design tool in constructing Monte Carlo methods. In theory, an importance function could be designed using both the dynamical and observation model. However, it is commonly designed using the last observation only. As a result regular importance sampling algorithms will fail to generate good particle representations, since the dynamics has no effect on the particle locations. I proposed a family of algorithms that take into account the dynamics by sampling those "history" components of the state that are not defined by the importance function. I argued for the correctness of this new algorithms and have shown its efficiency in a series of experiments.

**Thesis 2: Local Likelihood Sampling** (Section 4.4). The Bootstrap Filter is very inefficient in case of reliable observations, i.e., when the observation likelihood function is peaky around its modes. Then, unless the number of particles is enormous, the Bootstrap Filter fails to develop a good representation of the posterior. This is because its prediction step will fail to allocate particles in the close vicinity of the modes. I proposed the Local Likelihood Sampling scheme that suggests a local perturbation step based on the observation likelihood in the vicinity of the position predicted by the dynamical model to overcome this problem. I have showed that the

sampling scheme is unbiased. A theoretical analysis was developed to show LLS's efficiency as compared to that of the Bootstrap Filter.

**Thesis 3: Local Importance Sampling** (Section 4.5). I proposed an algorithm that represents a practical variant of the Local Likelihood Sampling particle filter. The suggested method does not require one to perform the local perturbation exactly according to the likelihood function. Instead any approximation of the likelihood can be used. I have showed that the algorithm is unbiased. Furthermore, I proposed an efficient version of this method that can be used when the local proposal that approximates the likelihood is Gaussian. Experiments with a real-world visual tracking problem and on a difficult simulated tracking problem show the improved tracking efficiency of the new algorithm.

# Appendix A

# Low Variance Resampling Methods

As it was discussed in Section 2.5, basic resampling (also known as multinomial resampling) causes a fast degeneration of the particle representation. Degeneration follows from the variance increase of the posterior representation. Here the corresponding theorem will be proved, followed by the discussion of some alternative resampling techniques that have lower variance. A nice summary on the topic is [33].

## A.1 Degradation of Multinomial Resampling

**Proposition A.1.1.** *Let $N$ be the number of particles with equal weights ($1/N$). Assume that not all the particles are unique, but we only have $m$ distinct ones. Denote the weights of the $m$ distinct particles by $W_1^0, W_2^0, \ldots, W_m^0$ ($\sum_{i=1}^m W_i^0 = 1, W_i^0 \geq 0$). Assume that we resample the particle set with multinomial resamplings for a couple of times in a recursive fashion. Let $W_i = (W_1^i, W_2^i, \ldots, W_m^i)^T$ be the weights of the particles after the $i$-th resampling step. Let $T_n$ be the time of reaching a completely degenerated particle set, i.e.: $T_N = \inf\{k | \sum_{j=1}^N W_j^k (1 - W_j^i) = 0\}$. Then:*

$$\mathbb{E}\left[T_N\right] \leq 2N \log 2N$$

*Proof.* Let us first calculate $R_i = \mathbb{E}\left[\sum_{j=1}^N W_j^{i+1}(1 - W_j^{i+1})|W_i\right]$

$$\begin{aligned} R_i &= \sum_{j=1}^N \mathbb{E}\left[W_j^{i+1}|W^i\right] - \mathbb{E}\left[(W_j^{i+1})^2|W^i\right] = \\ &= \sum_{j=1}^N \mathbb{E}\left[W_j^{i+1}|W^i\right] - \left(\mathrm{Var}(W_j^{i+1}|W^i) - \mathbb{E}(W_j^{i+1}|W^i)^2\right) \end{aligned}$$

Using the well known formula for the expected value and the variance of the binomial

distribution, we get

$$R_i = \sum_{j=1}^{N} W_j^i - \frac{1}{N^2} N W_j^i (1 - W_j^i) - (W_j^i)^2 = \frac{N-1}{N} \sum_{j=1}^{N} W_j^i (1 - W_j^i).$$

Hence $\mathbb{E}[R_i | W_j^{i-1}] = \frac{N-1}{N} \mathbb{E}[R_{i-1} | W_j^{i-1}]$ and thus

$$\mathbb{E}\left[ \sum_{j=1}^{N} W_j^i (1 - W_j^i) \right] = \left( \frac{N-1}{N} \right)^i \sum_{j=1}^{N} W_j^0 (1 - W_j^0).$$

Observe that $T_N > k$ if and only if $\sum_{j=1}^{N} W_j^k (1 - W_j^k) \geq \frac{2}{N} \left(1 - \frac{1}{N}\right)$, as this is the smallest possible non-zero value of the expression. Using Markov's inequality:

$$\begin{aligned} \mathbb{P}(T_N > k) &= \mathbb{P}\left( \sum_{j=1}^{N} W_j^k (1 - W_j^k) \geq 2 \frac{N-1}{N^2} \right) \leq \frac{N^2}{N-1} \mathbb{E}\left( \sum_{j=1}^{N} W_j^k (1 - W_j^k) \right) = \\ &= 2 \left( \frac{N-1}{N} \right)^k \frac{N^2}{N-1} \sum_{j=1}^{N} W_j^0 (1 - W_j^0). \end{aligned} \tag{A.1}$$

Let $q = \frac{N-1}{N}$ and $c = 2 \frac{N^2}{N-1} \sum_{j=1}^{N} W_j^0 (1 - W_j^0)$. Hence $\mathbb{P}(T_n > k) \leq c q^k$. Note that the estimate of (A.1) is very loose as long as $cq^k > 1$, i.e. when $k < k_0 = \log_{\frac{1}{q}} c$.

We are ready to estimate the expected degradation time:

$$\mathbb{E} T_N = \sum_{k=0}^{\infty} P(T_N > k) \leq k_0 + \frac{1}{1-q}.$$

Now note that $\log \frac{1}{q} = -\log \frac{N-1}{N} \geq \frac{1}{N}$ to get

$$\mathbb{E} T_N \leq N + 2N \log \left( N \sum_{j=1}^{N} W_j^0 (1 - W_j^0) \right) \leq 2N \log 2N,$$

which finishes the proof. □

## A.2 Residual Resampling

The process of resampling can be viewed as generating *duplication numbers* (the number of times a particle is repeated in the new resampled set) of the original particles in exchange of omitting the weights (see Section 2.5.1). In our first alternative resampling scheme, called residual resampling, the duplication numbers are set as a sum of a deterministic and a stochastic component:

$$N^i = \lfloor N w^{(i)} \rfloor + \hat{W}^i.$$

Here $\hat{W}^i$ is drawn from the multinomial density defined as $Mult(N-R; \hat{w}^{(1)}, \hat{w}^{(2)}, \ldots, \hat{w}^{(N)})$, with $R = \sum_{i=1}^{N} \lfloor N w^{(i)} \rfloor$ and $\hat{w}^{(i)} = \frac{N w^{(i)} - \lfloor N w^{(i)} \rfloor}{N-R}$. Here $w^{(1)}, \ldots, w^{(N)}$ are the original weights ($w^{(i)} \geq 0, \sum_{i=1}^{N} w^{(i)} = 1$). This algorithm sets a large amount of the new particles deterministically, which is decreasing the variance of the resampling.

## A.3 Stratified Resampling

Let us define the cumulative distribution of particle weights. Let $w^{(1)}, w^{(2)}, \ldots, w^{(N)}$ be a normalized set of particle weights. Define

$$C_k = \sum_{j=1}^{k} w^{(j)}.$$

Note that $C_N = 1$. It is easy the see that a set of numbers $\{U_i\}_{i=1}^{N}$ in the $(0, 1]$ interval define a resampling method when the indexes of the $i$-th particle index in the resampled set will be: $\text{argmin}_k U_i \leq C_k$.

With this definition Multinomial-resampling can be described as sampling all $U_i$ from the uniform distribution in the $(0, 1]$ interval, i.e., $U_i \sim U(0, 1]$. This is in fact the most common description of Multinomial resampling in the Monte-Carlo books (see, e.g., [39]).

In case of stratified resampling $U_i$ each is drawn from uniform distribution on the $(\frac{i-1}{N}, \frac{i}{N}]$ interval, i.e., $U_i \sim U(\frac{i-1}{N}, \frac{i}{N}]$. This method, again, reduces the variance of resampling by introducing some restrictions on the generated indexes.

## A.4 Systematic Resampling

Systematic resampling is similar to stratified resampling with the only difference that $u_i$ is defined using only one random sample $U$ drawn from $U(0, \frac{1}{N}]$. Then

$$U_i = \frac{i-1}{N} + U.$$

This algorithm is the most preferred one in the literature, because of its computational simplicity and good empirical performance. Unfortunately the resulting particle positions are no longer independent, which make the theoretical analysis of these algorithms more challenging. This method is an instance of the generic Monte-Carlo technique called the method of "Common Random Numbers" [12].

# Appendix B

# Auxiliary Variable Method

The Auxiliary Variable Method (AVM) was introduced by Pitt and Shephard in [31]. The motivation is to decrease the degeneration occurring in particle filtering caused by the curse of reliable observations. In the followings we will follow the description of [10].

Observe that given particle representation $S_{t-1} = \{(X_{t-1}^{(i)}, w_{t-1}^{(i)})\}_{i=1}^{N}$ of $p(X_{t-1}|Y_{0:t-1})$, the best estimate of the posterior at time $t$ is:

$$p(X_t = \cdot|Y_{0:t}) \approx p(X_t = \cdot|Y_t, S_{t-1}) \propto r(Y_t|X_t = \cdot) \sum_{i=1}^{N} w_{t-1}^{(i)} K(X_t = \cdot|X_{t-1} = X_{t-1}^{(i)}). \quad \text{(B.1)}$$

Hence the task of particle filtering is to sample from this density. The trick in AVM is to perform particle filtering first in a higher dimension, to represent the joint density

$$p(X_t = \cdot, k|Y_t, S_{t-1}) \propto r(Y_t|X_t = \cdot) w_{t-1}^{(k)} K(X_t = \cdot|X_{t-1} = X_{t-1}^{(k)}) \quad k = 1, \ldots, N,$$

and then marginalize this by discarding the index $k$ to get a sample from the empirical filtering density (B.1). The index $k$ is called an auxiliary variable, for obvious reasons.

Note that this very general idea can be combined with several other particle filters. For example combining it with importance sampling would be to sample $(X_t^{(j)}, k^{(j)})$ pairs from $\pi(X_t, k|Y_{0:t})$ and calculate weights $w_j = r(Y_t|X_t^{(j)}) w_{t-1}^{(k^{(j)})} K(X_t^{(j)}|X_{t-1}^{(k^{(j)})}) / \pi(X_t, k^{(j)}|Y_{0:t})$.

The Auxiliary Variable Method [31] is generally used with the following generic proposal:

$$\pi(X_t, k|Y_{0:t}) \propto r(Y_t|\mu_t^{(k)}) w_{t-1}^{(k)} K(X_t|X_{t-1}^{(k)}),$$

which is thought to be a good approximation of $p(X_t, k|Y_t, S_{t-1})$. Here $\mu_t^{(k)}$ is the mean, the mode, or a sample from $K(X_t = \cdot|X_{t-1}^{(k)})$. Note that given this density $\pi(k|Y_{0:t}) = \int \pi(X_t, k|Y_{0:t}) dX_t \propto r(Y_t|\mu_t^{(k)}) w_{t-1}^{(i)}$ holds.

This means that sampling from $\pi(X_t, k|Y_{0:t})$ is possible in a three step process, when after generating the value $\mu_t^{(k)}$ from $K(X_t|X_{t-1}^{(k)})$ for each $k$, a random index $K^{(j)}$ is drawn from $\lambda_k \propto r(Y_t|\mu_t^{(k)}) w_{t-1}^{(k)}$. Finally, $X_t^{(j)}$ is sampled from $K(X_t = \cdot|X_{t-1}^{(K^{(j)})})$. The $\lambda_k$-s are

called the first stage weights. The particle weights are set to

$$w_j = \frac{r(Y_t|X_t^{(j)})}{r(Y_t|\mu_t^{(k^{(j)})})}.$$

The idea is that in the first step (when sampling $K^{(j)}$) the algorithm effectively filters out those particles which do not have a good chance to produce an "offspring" that survive (because $r(Y_t|\mu_t^{(k)})$ is low). This can be a big win since if the particles in the new generation derive from only a few ancestors then the quality of the approximation of the posterior will be doomed to degrade.

Although this algorithm can be expected to perform much better than the Bootstrap Filter when the likelihood function is peaky, the algorithm still suffers from the problem that the observation influences the particles only in an indirect manner. Thus when the prediction densities $K(X_t = \cdot|X_{t-1}^{(k)})$ are "broad" as compared with the observation likelihood, the algorithm will still require a large number of particles.

Another disadvantage of the algorithm is that it requires two observation likelihood evaluations for each particle. Since these steps are the computationally most expensive ones in visual tracking (since they include expensive image processing calculations) algorithms that perform less observation evaluations are preferred.

The detailed algorithm is shown in Figure B.1.

**Initialize** $\left[(X_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Generate the likely values** $\mu_t^{(i)}$ from $K(X_t|X_{t-1}^{(i)})$. Let $\lambda_i \propto w_{t-1}^{(i)} r(Y_t|\mu_t^{(i)})$, for $i = 1, \ldots, M >> N$.

**For** $i = 1, 2, \ldots, N$

**Sample auxiliary variable index** $K_i$ with $p(K_i = j) = \lambda_j$.

**Sample next state** $X_t^{(i)}$ from $K(X_t|X_{t-1}^{(K^{(i)})})$.

**Calculate weights**

$$\hat{w}_t^{(i)} = \frac{r(Y_t|X_t^{(i)})}{r(Y_t|\mu_t^{(K^{(i)})})}.$$

**EndFor**

**Normalize weights** using

$$w_t^{(j)} = \frac{\hat{w}_t^{(j)}}{\sum_{i=1}^{N} \hat{w}_t^{(i)}}$$

**EndFor**

Figure B.1: The Auxiliary Variable Particle Filter algorithm.

# Appendix C

# Partitioned Sampling

The strength of particle filters come from the weighted resampling step that helps it to concentrate the particles to the most relevant part of the state space. When the state of the object has many components the same idea can be applied to drawing components in a sequential manner. The resulting algorithm is called partitioned sampling and was introduced in [23, 24, 22]. The origin of the name is that the components of the state are "partitioned" with components in separate partitioned sampled at once.

In the followings we will give a slightly more general description of the concept than e.g. [22]. First we introduce a sequence of auxiliary probability variables $Z_{t,i}$, $i = 1..\tau$ with $Z_{t,\tau} = X_t$ together with some kernel functions $K_i(z_{t,i}|z_{t,i-1}, X_{t-1})$ that defines the transition probabilities between the variables. Furthermore we introduce a series of importance functions $\pi_i(z_{t,i}|Y_t)$ whose purpose is to feed information from the observation to the sample. The key idea of partitioned sampling is to generate the next state $X_t$ through the auxiliary variables, with resampling after each step, i.e.,

1. **For** $j = 1, \ldots, \tau$
   - Generate the next particle set $\left[Z_{t,j}^{(i)}\right]_{i=1}^{N}$ by
   $$Z_{t,j}^{(i)} \sim K_j(z_{t,j} = \cdot|Z_{t,j-1}^{(i)}, X_{t-1}).$$
   - Weight samples by $w_t^{(i)} = \pi_i(Z_{t,j}^{(i)}|Y_t)$.
   - Do weighted resampling on $\left[Z_{t,j}^{(i)}\right]_{i=1}^{N}$ using $w_t^{(i)}$.
2. **EndFor**

The simple observation that ensures that the above algorithm produces unbiased weighted sample set from $K(X_t|X_{t-1})$ is as follows:

$$\begin{aligned} K(X_t|X_{t-1}) &= \int K_\tau(X_t|z_{t,\tau-1}, X_{t-1}) \ldots \int K_3(z_{t,3}|z_{t,2}, X_{t-1}) \\ &\quad \int K_2(z_{t,2}|z_{t,1}, X_{t-1}) K_1(z_{t,1}|X_{t-1}) dz_{t,1} \ldots dz_{t,\tau-1}. \end{aligned} \tag{C.1}$$

Until now, we gave the general description of the partitioned sampling algorithm. In the followings we will discuss the scenarios, in which it is most typically used.

The key idea is to divide the state space into $K$ partitions[1] as $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_\tau$ and let $Z_{t,i} = (X_{t,1:i}, X_{t-1,i+1:\tau})$, where $X_{t,i:j}$ means $X_{t,i}, \ldots X_{t,j}$. With this choice the algorithm can be viewed as one that builds up the next state $X_t$ gradually in several steps, and resampling is used to populate successful state parts. Note that if the importance functions $(\pi_i(z_{t,i}|Y_t))$ are related to the observation likelihood, then the partitioned sampling algorithm have the appealing characteristic that particle positions are determined using both the dynamical and the observation model.

Before giving the specific algorithm for this case, we argue that a significant advantage in memory usage can be achieved if instead of C.1, which holds unconditionally, we use the following assumption, that may not hold always:

$$K(X_t|X_{t-1}) = \int K_\tau(X_t|z_{t,\tau-1}) \ldots \int K_3(z_{t,3}|z_{t,2}) \int K_2(z_{t,2}|z_{t,1}) K_1(z_{t,1}|X_{t-1}) dz_{t,1} \ldots dz_{t,\tau-1}. \tag{C.2}$$

The advantage lies in the fact that one does not have to store the value of $X_{t-1}$, but instead its components can gradually be exchanged by the new components of $X_t$. The algorithm that uses this trick is given in Figure C.1.

One example when this latter equation holds comes from multi-target tracking with objects moving independently. In this case a good state factorization is if each object's state is a factor over the joint state space. Target occlusions can also be modeled if a visibility ordering of the targets can be estimated. In this case the most frontal object should have the lowest factorization index.

Another example is tracking articulated objects. For example in [25] a hand is tracked with a four-partition system. In the first partition are the scale, orientation and translation of the fist, the second partition of variables is the joint angle of the thumb, the third is the joint angle of the tip of the thumb, and the fourth is the joint angle of the index finger. Note that the ordering of the partitions plays an important role here.

One problem with Partitioned Sampling is that the sequential weighted resampling steps causes an impoverishment effect of the particle partitions that are generated earlier similarly to what happens with particle filters when resampling is used for two many steps (see AppendixA). For example in hand tracking of [25] the variance in the first subspace (translation, rotation and scale) might get very low after the weighted resamplings of the other subspaces. In case of tracking multiple objects, this effect increases with the number of objects, creating an erratic and undesirable behavior [41].

One solution to overcome particle partition impoverishment is to use Branched Partitioned Sampling [22]. The idea behind this algorithm can be summarized as follows: if

[1]The word partition, that gives the name of the algorithm, is commonly used in the literature, however this is strictly speaking a misnomer. A more clear description would be to say that the state space is factored of $K$ sub-spaces.

**Initialize** $\left[(X_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$
  **For** $j = 1, \ldots, \tau$
    **For** $k = 1, 2, \ldots, N$
      **Sample** the $j$-th component of the $k$-th particle from

$$\hat{X}_{t,j}^{(k)} \sim K_j(X_{t,j} = \cdot | X_{t,1:j-1}^{(k)}, X_{t-1,j:\tau}^{(k)})$$

      **Calculate component likelihood**

$$l_{t,j}^{(k)} = \pi_j(X_{t,j}^{(k)} | X_{t,1:j-1}^{(k)}, Y_t).$$

    **EndFor**
    **Do weighted resampling** on $\left\{\left(X_{t,1:j-1}^{(k)}, \hat{X}_{t,j}^{(k)}, X_{t-1,j+1:\tau}^{(k)}\right), w_{t,j-1}^{(k)}\right\}_{k=1}^{N}$ with $p(X_{t,j} = \hat{X}_{t,j}^{(n)}) \propto l_{t,j}^{(n)}$ to get $\left\{\left(X_{t,1:j-1}^{(k)}, X_{t,j}^{(k)}, X_{t-1,j+1:\tau}^{(k)}\right), w_{t,j}^{(k)}\right\}_{k=1}^{N}$.
    We used the convention that $w_{t,0}^{(k)} = w_{t-1}^{(k)}$.
  **EndFor**
  **Calculate likelihoods using** $\hat{w}_t^{(k)} = w_{\tau,t}^{(k)} r(Y_t | X_t^{(k)}), \quad k = 1, \ldots, N.$
  **Normalize weights** using

$$w_t^{(j)} = \frac{\hat{w}_t^{(j)}}{\sum_{i=1}^{N} \hat{w}_t^{(i)}}$$

**EndFor**

Figure C.1: The Partitioned Sampling algorithm. Note that if $r(X_t = |Y_t) = \pi_1(X_{t,1}|Y_t)\pi_2(X_{t,2}|X_{t,1}, Y_t) \ldots \pi_\tau(X_{t,\tau}|X_{t,1:\tau-1}, Y_t)$ holds, then $l_{j,t}^{(k)}$ might be reused to calculate the importance weights $\hat{w}_t^{(k)}$.

there are several partition orderings that satisfy Equation C.2, then using these might decrease the above mentioned particle impoverishment effect. The rule of thumb here is that components generated earlier suffer from particle impoverishment. Perturbing the order of components helps since for each partition there are particles in which they are generated at the end of the partition process line. The name "Branched" comes from the visual clue that different particles follow different branches of component sampling. They are joined together at the weight normalization step.

Note that in case of tracking an articulated body, or multiple targets with occlusion the partition orderings have to satisfy some constraints, since the algorithm can be used just on the orderings that satisfy our main assumption. This makes it difficult to avoid the particle impoverishment completely. As a result the branching idea might be used in a limited way.

The algorithm is shown in Figure C.2.

We finally note that the method can be combined with other particle filtering methods such as LLS or LIS.

**Initialize** $\left[(X_0^{(k)}, 1/N)\right]_{k=1}^{N}$ from the prior $p_0(\cdot)$.
**For** $t = 1, 2, \ldots$

**Acquire partition orderings** $\sigma^{(i)}$ for all particles $X_{t-1}^{(i)}, \quad i = 1, \ldots, N$.

**Collect particles with the same orderings** $H_\sigma = \{i | \sigma^{(i)} = \sigma\}$.

**For** all $\sigma$ permutations of $\{1, \ldots, \tau\}$ when $H_\sigma$ is non-empty

**For** $j = 1, \ldots, \tau$

**For all indexes** $k$ in $H_\sigma$

**Sample** the $\sigma(j)$-th partition of the $k$-particle from

$$\hat{X}_{t,\sigma(j)}^{(k)} \sim K_\sigma(j)(X_{t,\sigma(j)} = \cdot | X_{t,\sigma(1):\sigma(j-1)}^{(k)}, X_{t-1,\sigma(j):\sigma(\tau)}^{(k)})$$

**Calculate partition likelihood**

$$l_{t,\sigma(j)}^{(k)} = \pi_\sigma(j)(X_{t,\sigma(j)}^{(k)} | X_{t,\sigma(1):\sigma(j-1)}^{(k)}, Y_t).$$

**EndFor**

**Make weighted resampling** on $\left\{\left(X_{t,\sigma(1):\sigma(j-1)}^{(k)}, \hat{X}_{t,\sigma(j)}^{(k)}, X_{t-1,\sigma(j+1):\sigma(\tau)}^{(k)}\right), w_{t,\sigma(j-1)}^{(k)}\right\}_{k=1}^{N}$ with $p(X_{t,\sigma(j)} = \hat{X}_{t,\sigma(j)}^{(n)}) \propto l_{t,\sigma(j)}^{(n)}$ to get $\left\{\left(X_{t,\sigma(1):\sigma(j-1)}^{(k)}, X_{t,\sigma(j)}^{(k)}, X_{t-1,\sigma(j+1):\sigma(\tau)}^{(k)}\right), w_{t,\sigma(j)}^{(k)}\right\}_{k=1}^{N}$. We used the convention that $w_{t,\sigma(0)}^{(k)} = w_{t-1}^{(k)}$.

**EndFor**

**Calculate likelihoods using** $\hat{w}_t^{(k)} = w_{\sigma(\tau),t}^{(k)} r(Y_t | X_t^{(k)}), \quad k = 1, \ldots, N$.

**Normalize weights** using

$$w_t^{(j)} = \frac{\hat{w}_t^{(j)}}{\sum_{i=1}^{N} \hat{w}_t^{(i)}}$$

**EndFor**

Figure C.2: The Branched Partitioned Sampling algorithm. $X_{t,\sigma(i):\sigma(j)}^{(k)}$ means $X_{t,\sigma(i)}^{(k)}, \ldots, X_{t,\sigma(j)}^{(k)}$.

# Appendix D

# Products of Gaussian Densities

Let $f_1(x)$ and $f_2(x)$ be the density function of two multi-dimensional Gaussians:

$$f_1(x) = ((2\pi)^d|\Sigma_1|)^{-1/2}e^{-1/2(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)},$$

and

$$f_2(x) = ((2\pi)^d|\Sigma_2|)^{-1/2}e^{-1/2(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)},$$

It is well known and immediate from the definition of Gaussians that $f_1(x)f_2(x)$ is also a density function of a Gaussian if it is appropriately normalized. The question investigated here is what the mean, covariance and the normalizing factor of this product Gaussian is.

First we examine the exponent:

$$\begin{aligned} A &= (x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + (x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) = \\ &= x^T(\Sigma_1^{-1}+\Sigma_2^{-1})x - (\mu_1^T\Sigma_1^{-1}+\mu_2^T\Sigma_2^{-1})x - x^T(\Sigma_1^{-1}\mu_1+\Sigma_2^{-1}\mu_2) \\ &\quad +\mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2. \end{aligned}$$

Using that $(\Sigma^{-1})^T = \Sigma^{-1}$ as covariance matrixes are symmetric and positive definite,

$$A = x^T(\Sigma_1^{-1}+\Sigma_2^{-1})x - 2(\mu_1^T\Sigma_1^{-1}+\mu_2^T\Sigma_2^{-1})x + \mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2.$$

Denote the mean and the covariance of the product Gaussian as $\mu$ and $\Sigma$. From the second order term of $A$:

$$\Sigma^{-1} = \Sigma_1^{-1} + \Sigma_2^{-1}.$$

From the first order term of $A$:

$$\mu^T\Sigma^{-1} = \mu_1^T\Sigma_1^{-1} + \mu_2^T\Sigma_2^{-1}.$$

Hence:

$$\mu = \Sigma\Sigma_1^{-1}\mu_1 + \Sigma\Sigma_2^{-1}\mu_2.$$

The difference of $A$ and the corresponding part of the product Gaussian is

$$\begin{aligned}
K &= \mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2 - \mu^T\Sigma^{-1}\mu = \\
&= \mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2 - (\mu_1^T\Sigma_1^{-1}\Sigma + \mu_2^T\Sigma_2^{-1}\Sigma)\Sigma^{-1}(\Sigma\Sigma_1^{-1}\mu_1 + \Sigma\Sigma_2^{-1}\mu_2) = \\
&= \mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2 - (\mu_1^T\Sigma_1^{-1} + \mu_2^T\Sigma_2^{-1})\Sigma(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2) = \\
&= \mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2 - (\mu_1^T\Sigma_1^{-1}\Sigma\Sigma_1^{-1}\mu_1 + 2\mu_2^T\Sigma_2^{-1}\Sigma\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\Sigma\Sigma_2^{-1}\mu_2) = \\
&= \mu_1^T\Sigma_1^{-1}\Sigma\Sigma_2^{-1}\mu_1 - 2\mu_2^T\Sigma_2^{-1}\Sigma\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\Sigma\Sigma_1^{-1}\mu_2) = \\
&= (\mu_1 - \mu_2)^T\Sigma_1^{-1}\Sigma\Sigma_2^{-1}(\mu_1 - \mu_2)
\end{aligned}$$

In summary:

$$f_1(x)f_2(x) = ((2\pi)^d|\Sigma_1|)^{-1/2}((2\pi)^d|\Sigma_2|)^{-1/2}e^{-1/2K}e^{-1/2(x-\mu)^T\Sigma^{-1}(x-\mu)}.$$

Denote the density function of the product Gaussian by $f(x)$, then:

$$\begin{aligned}
f_1(x)f_2(x) &= ((2\pi)^d|\Sigma_1|)^{-1/2}((2\pi)^d|\Sigma_2|)^{-1/2}((2\pi)^d|\Sigma|)^{1/2}e^{-1/2K}f(x) = \\
&= ((2\pi)^d|\Sigma_1||\Sigma_2|/|\Sigma|)^{-1/2}e^{-1/2K}f(x)
\end{aligned}$$

Hence the normalizing factor of $f_1f_2$ that makes it a density is

$$((2\pi)^d|\Sigma_1||\Sigma_2|/|\Sigma|)^{1/2}e^{1/2K}.$$

# Bibliography


[1] N. Bergman. *Recursive Bayesian Estimation: Navigation and Tracking Applications*. PhD thesis, Department of Electrical Engineering, Linkoping, 2000.

[2] A. Blake and M. Isard. *Active Contours*. Springer-Verlag, 1998.

[3] O. Cappe and E. Moulines. *Inference in Hidden Markov Models*. Springer, 2005.

[4] J. Carpenter, P. Clifford, and P. Fearnhead. An improved particle filter for nonlinear problems. *IEEE Proceedings-Radar Sonar and Navigation*, 146(1):2–7, 1999.

[5] T.-J. Cham and J. M. Rehg. A multiple hypothesis approach to figure tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Fort Collins*, pages 239–245, 1999.

[6] W. Burgard D. Fox, S. Thrun and F. Dellaert. Particle filters for mobile robot localization. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, New York, 2001. Springer.

[7] P. del Moral. A uniform convergence theorem for the numerical solving of the nonlinear filtering problem. *Journal of Applied Probability*, 35:873–884, 1998.

[8] J. Deutscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. In *Conference on Computer Vision and Pattern Recognition*, 2000.

[9] A. Doucet. On sequential simulation based methods for Bayesian filtering. Technical report, CUED/F-INFENG/TR310, Cambridge Univeristy, Department of Engineering, 1998.

[10] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001.

[11] Rubinstein F.Y. *Simulation and the Monte Carlo Method*. John Wiley and Sons, 1981.

[12] P. Glasserman and D.D. Yao. Some guidelines and guarantees for common random numbers. *Management Science*, 38:884–908, 1992.

[13] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Proc. Inst. Elect. Eng. F*, 140(2):107–113, 1993.

[14] M. Isard and A. Blake. CONDENSATION – conditional density propagation for visual tracking. *International Journal Computer Vision*, 29:5–28, 1998.

[15] M. Isard and A. Blake. ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework. *Proc 5th European Conf. Computer Vision*, 1998.

[16] A. M. Jazwinsky. *Stochastic Processes and Filtering Theory*. Academic, New York, 1970.

[17] R.E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engineering*, pages 35–45, 1960.

[18] G. Kitagawa. Monte-Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, pages 1–25, 1996.

[19] A. Kong. A note on importance sampling using renormalized weights. Technical report, Department of Statistics, University of Chicago, 1992.

[20] J.S. Liu. Metropolized independent sampling. *Statistics and Computing*, 6:113–119, 1996.

[21] J.S. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

[22] J. MacCormick. *Probabilistic Modelling and Stochastic Algorithms for Visual Localisation and Tracking*. PhD thesis, University of Oxford, 2000.

[23] J. MacCormick and A. Blake. A probabilistic contour discriminant for object localisation. In *Proceedings of the 6th International Conference on Computer Vision*, pages 390–395, 1998.

[24] J. MacCormick and A. Blake. A probabilistic exclusion principle for tracking multiple objects. In *Proceedings of the 7th International Conference on Computer Vision*, pages 572–578, 1999.

[25] J. MacCormick and M. Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *Proceedings of the European Conference on Computer Vision*, volume 2, pages 3–19, 2000.

[26] R. Martin and B. Heinrich. RPROP - a fast adaptive learning algorithm. *Proc. of ISCIS VII.*, 1992.

[27] N. Metropolis and S.Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, pages 335–341, 1949.

[28] K. Okuma, A. Taleghani, N. de Freitas, J.J. Little, and D.G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *European Conference on Computer Vision*, volume 1, pages 28–39, 2004.

[29] T. Drummond P. Smith and R. Cipolla. Edge tracking for motion segmentation and depth ordering. In *10th British Machine Vision Conference Nottingham*, pages 369–378, 1999.

[30] P. Perez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *European Conference on Computer Vision*, pages 661–675, 2002.

[31] M.K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[32] Maybeck P.S. *The Kalman Filter: An introduction to concepts*. Springer, 1990.

[33] E. Moulines R. Douc, O. Cappe. Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis*, pages 52–57, 2005.

[34] D.B. Rubin. Using the SIR algorithm to simulate posterior distributions. In D.V. Lindley et A.F.M. Smith) J.M. Bernardo, M.H. DeGroot, editor, *Bayesian Statistics 3*, pages 395–402. Oxford University Press, 1988.

[35] Julier S. and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *In International Symposium on Aerospace/Defence Sensing, Simulate and Control*, 1997.

[36] W. Burgard. S. Thrun and D. Fox. *Probabilsitc Robotics*. MIT Press, 2005.

[37] K. Schwerdt and J. Crowley. Robust face tracking using color. In *4th International Conference on Automatic Face and Gesture Recognition*, pages 90–95, 2000.

[38] D.W. Scott. *Multivariate Density Estimation. Theory, Practice, and Visualization*. wiley, New York, London, Sydney, 1992.

[39] George S.Fishman. *Monte Carlo Concepts, Algorithms and Applications*. Springer, 1999.

[40] A.F.M Smith and A.E. Gelfand. Bayesian statistics without tears: a sampling-resampling perspective. *America Statistican*, pages 84–88, 1992.

[41] K. Smith and D. Gatica-Perez. Order matters: A distributed sampling method for multiple-object tracking. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 4–25, 2004.

[42] D.M. Titterington, A.F.M. Smith, and U.E. Makov. *Statistical Analysis of Finite Mixture Distributions*. wiley, New York, London, Sydney, 1985.

[43] P. Torma. Improving Monte-Carlo Bayesian filtering by means of local search: Theory and applications in visual tracking. Master's thesis, Eötvös Lóránd University, Budapest, 2001.

[44] P. Torma and Cs. Szepesvári. Efficient visual object tracking by means of ls-n-ips. *Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis*, pages 277–282, 2001.

[45] P. Torma and Cs. Szepesvári. LS-N-IPS: an improvement of particle filters by means of local search. *Proc. Non-Linear Control Systems(NOLCOS'01) St. Petersburg, Russia*, pages 715–719, 2001.

[46] P. Torma and Cs. Szepesvári. Towards facial pose tracking. In *Proceedings of the 1st Hungarian Conference on Computer Graphics and Geometry*, pages 10–16, 2002.

[47] P. Torma and Cs. Szepesvári. Combining local search, neural networks and particle filters to achieve fast and reliable contour tracking. In *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, 2003.

[48] P. Torma and Cs. Szepesvári. Sequential importance sampling for visual tracking reconsidered. In *In C.M.Bishop and B.J.Frey (eds.), Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, pages 271–279, 2003.

[49] P. Torma and Cs. Szepesvári. Enhancing particle filters using local likelihood sampling. *ECCV2004, Prague. Lecture Notes in Computer Science*, pages 16–28, 2004.

[50] P. Torma and Cs. Szepesvári. On using likelihood-adjusted proposals in particle filtering: Local importance sampling. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis*, pages 58–63, 2005.

[51] P. Torma and Cs. Szepesvári. Local importance sampling: A novel technique to enhance particle filtering. *Journal of Multimedia*, pages 32–43, 2006.

[52] van der Merwe. *Sigma-Point Kalman Filters for Probabilistc Inference in Dynamic State Space Models*. PhD thesis, OGI School of Engineering and Science, 2004.