CHAPTER 2

Tracking with Non-Linear Dynamic Models

In a linear dynamic model with linear measurements, there is always only one peak in the posterior; very small non-linearities in dynamic models can lead to a substantial number of peaks. As a result, it can be very difficult to represent the posterior: it may be necessary to represent all the peaks to be able to compute the mean or the covariance. We discuss these difficulties in section 2.1. There is no general solution to this problem, but there is one mechanism which has proven useful in some practical problems: we present this, rather technical, mechanism in section 2.2, and show some applications in section 2.3.

It is quite typical of vision applications that there is some doubt about what measurements to track — for example, a Kalman filter tracker following a series of corner points may need to decide which image measurement corresponds to which track. A poor solution to this problem may lead to apparently good tracks that bear no relationship to the underlying motions. In section **??**, we discuss how to attach measurements to tracks.

2.1 NON-LINEAR DYNAMIC MODELS

If we can assume that noise is normally distributed, linear dynamic models are reasonably easy to deal with, because a linear map takes a random variable with a normal distribution to another random variable with a (different, but easily determined) normal distribution. We used this fact extensively in describing the Kalman filter. Because we knew that everything was normal, we could do most calculations by determining the mean and covariance of the relevant normal distribution, a process that is often quite easy if one doesn't try to do the integrals directly. Furthermore, because a normal distribution is represented by its mean and covariance, we knew what representation of the relevant distributions to maintain.

Many natural dynamic models are non-linear. There are two sources of problems. Firstly, in models where the dynamics have the form

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i); \Sigma_{d_i})$$

(where $\boldsymbol{f}$ is a non-linear function), both $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ and $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ tend not to be normal. As section 2.1.1 will show, even quite innocuous looking nonlinearities can lead to very strange distributions indeed. Secondly, $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ may not be Gaussian either. This phenomenon is quite common in vision; it leads to difficulties that are still neither well understood nor easily dealt with (section 2.1.2).

Dealing with these phenomena is difficult. There is not, and will never be, a completely general solution. It is always a good idea to see if a linear model can be made to work. If one does not, there is the option of linearizing the model locally

and assuming that everything is normal. This approach, known as the **extended Kalman filter** tends to be unreliable in many applications. We describe it briefly in the appendix, because it is useful on occasion. Finally, there is a method that maintains a radically different representation of the relevant distributions from that used by the Kalman filter. This method is described in section 2.2. The rest of this section illustrates some of the difficulties presented by non-linear problems.

2.1.1 Unpleasant Properties of Non-Linear Dynamics

Non-linear models of state evolution can take unimodal distributions — like Gaussians — and create multiple, well-separated modes, phenomena that are very poorly modeled by a single Gaussian.

This effect is most easily understood by looking at an example. Let us have the (apparently simple) dynamical model $x_{i+1} = x_i + 0.1 * \sin x_i$. Notice that there is no random component to this dynamical model at all; now let us consider $P(X_1)$, assuming that $P(X_0)$ is a Gaussian with very large variance (and so basically flat over a large range). The easiest way to think about this problem is to consider what happens to various points; as figure 2.1 illustrates, points in the range $((2k)\pi, (2k+2)\pi)$ move towards $(2k+1)\pi$. This means that probability must collect at points $(2k+1)\pi$ (we ask you to provide some details in the exercises).

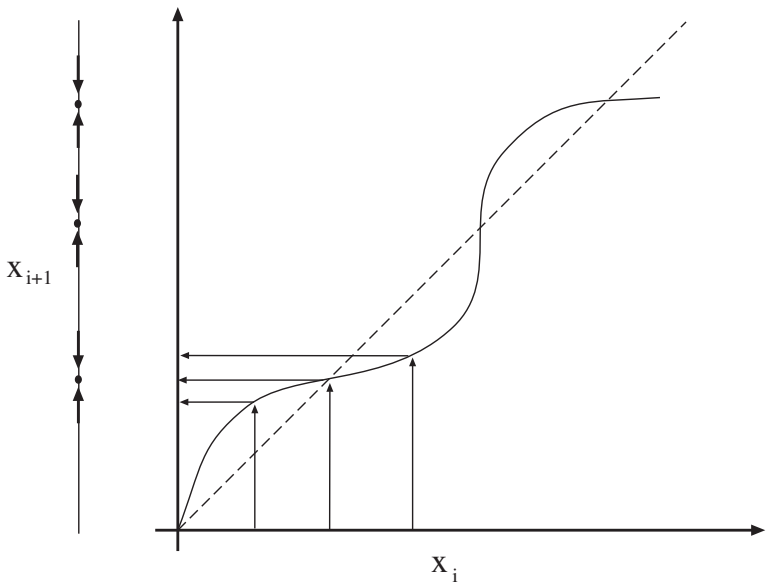

FIGURE 2.1: The non-linear dynamics $x_{i+1} = x_i + 0.1 \sin x_i$ cause points n the range $((2k)\pi, (2k+2)\pi)$ move towards $(2k+1)\pi$. As the figure on the **left** illustrates, this is because $x_i + 0.1 \sin x_i$ is slightly smaller than x_i for x_i in the range $((2k+1)\pi, (2k+2)\pi)$ and is slightly larger than x_i for x_i in the range $((2k)\pi, (2k+1)\pi)$. In fact, the nonlinearity of this function looks small — it is hardly visible in a scaled plot. However, as figure 2.2 shows, its effects are very significant.

This nonlinearity is apparently very small. Its effects are very substantial,

however. One way to see what happens is to follow a large number of different points through the dynamics for many steps. We choose a large collection of points according to $P(X_0)$, and then apply our dynamic model to them. A histogram of these points at each step provides a rough estimate of $P(X_i)$, and we can plot how they evolve, too; the result is illustrated in figure 2.2. As this figure shows, $P(X_i)$ very quickly looks like a set of narrow peaks, each with a different weight, at $(2k+1)\pi$. Representing this distribution by reporting only its mean and covariance involves a substantial degree of wishful thinking.

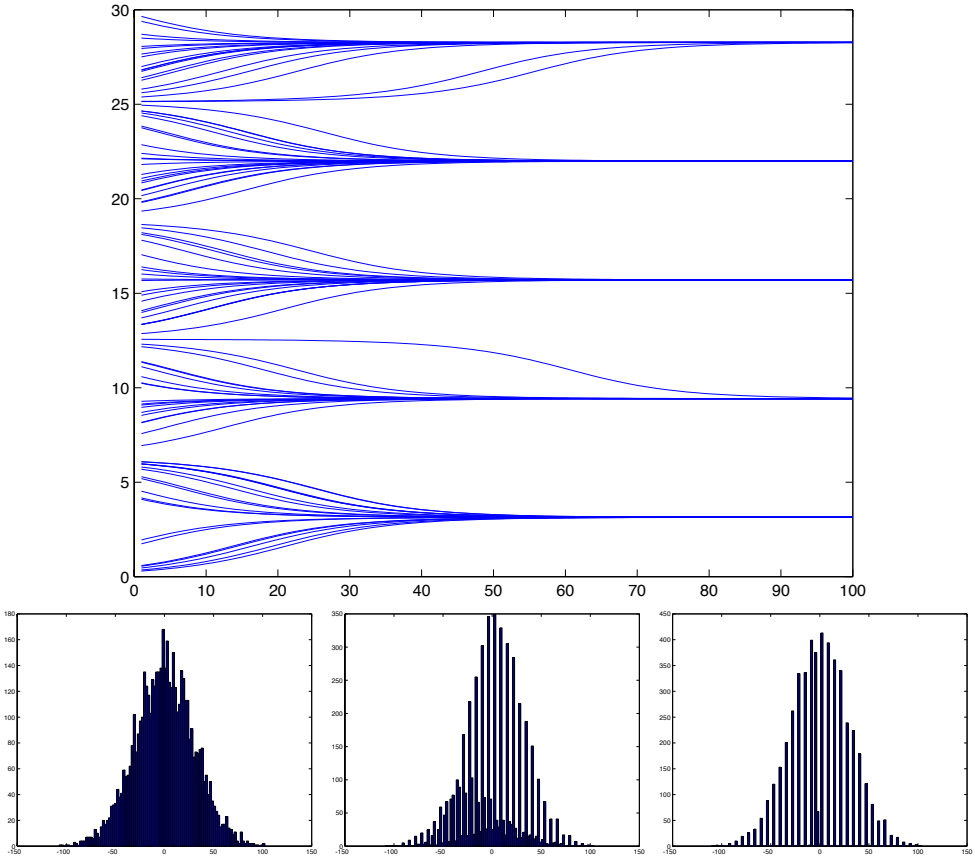

FIGURE 2.2: On the **top**, we have plotted the time evolution of the state of a set of 100 points, for 100 steps of the process $x_{i+1} = x_i + 0.1 * \sin x_i$. Notice that the points all contract rather quickly to $(2k+1)\pi$, and stay there. We have joined up the tracks of the points to make it clear how the state changes. On the **bottom left** we show a histogram of the start states of the points we used; this is an approximation to $P(x_0)$. The histogram on the **bottom center** shows a histogram of the point positions after 20 iterations; this is an approximation to $P(x_{20})$. The histogram on the **bottom right** shows a histogram of the point positions after 70 iterations; this is an approximation to $P(x_{70})$. Notice that there are many important peaks to this histogram — it might be very unwise to model $P(x_i)$ as a Gaussian.

2.1.2 Difficulties with Likelihoods

There is another reason to believe that $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ may be very complicated in form. Even if the dynamics do not display the effects of section 2.1.1, the likelihood function $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ can create serious problems. For many important cases we expect that the likelihood has multiple peaks. For example, consider tracking people in video sequences. The state will predict the configuration of an idealised human figure and $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ will be computed by comparing predictions about the image with the actual image, in some way. As the configuration of the idealised human figure changes, it will cover sections of image that aren't generated by a person but look as though they are. For example, pretty much any coherent long straight image region with parallel sides can look like a limb — this means that as $\boldsymbol{X}$ changes to move the arm of the idealised figure from where it should be to cover this region, the value of $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ will go down, and then up again. The likely result is a function $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ with many peaks in it.

We will almost certainly need to keep track of more than one of these peaks This is because the largest peak for any given frame may not always correspond to the *right* peak. This ambiguity should resolve itself once we have seen some more frames — we don't expect to see many image assemblies that look like people, move like people for many frames and yet aren't actually people. However, until it does, we may need to manage a representation of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ which contains several different peaks. This presents considerable algorithmic difficulties — we don't know how many peaks there are, or where they are, and finding them in a high dimensional space may be difficult. One partially successful approach is a form of random search, known as **particle filtering**.

2.2 PARTICLE FILTERING

The main difficulty in tracking in the presence of complicated likelihood functions or of non-linear dynamics is in maintaining a satisfactory representation of $P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$. This representation should be handle multiple peaks in the distribution, and should be able to handle a high-dimensional state vector without difficulty. There is no completely satisfactory general solution to this problem (and there will never be). In this section, we discuss an approach that has been useful in many applications.

2.2.1 Sampled Representations of Probability Distributions

A natural way to think about representations of probability distributions is to ask what a probability distribution is for. Computing a representation of a probability distributions is not our primary objective; we wish to represent a probability distribution so that we can compute one or another expectation. For example, we might wish to compute the expected state of an object given some information; we might wish to compute the variance in the state, or the expected utility of shooting at an object, etc. Probability distributions are devices for computing expectations — thus, our representation should be one that gives us a decent prospect of computing an expectation accurately. This means that there is a strong resonance between questions of representing probability distributions and questions of efficient numer-

ical integration.

Monte Carlo Integration using Importance Sampling. Assume that we have a collection of N points $\boldsymbol{u}^i$, and a collection of weights w^i. These points are independent samples drawn from a probability distribution $S(\boldsymbol{U})$ — we call this the **sampling distribution**; notice that we have broken with our usual convention of writing any probability distribution with a P. We assume that $S(\boldsymbol{U})$ has a probability density function $s(\boldsymbol{U})$.

The weights have the form $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$ for some function f. Now it is a fact that

$$\begin{aligned}\mathrm{E}\left[\frac{1}{N}\sum_i g(\boldsymbol{u}^i)w^i\right] &= \int g(\boldsymbol{U})\frac{f(\boldsymbol{U})}{s(\boldsymbol{U})}s(\boldsymbol{U})d\boldsymbol{U} \\ &= \int g(\boldsymbol{U})f(\boldsymbol{U})d\boldsymbol{U}\end{aligned}$$

where the expectation is taken over the distribution on the collection of N independent samples from $S(\boldsymbol{U})$ (you can prove this fact using the weak law of large numbers). The variance of this estimate goes down as $1/N$, and is independent of the dimension of $\boldsymbol{U}$.

Representing Distributions using Weighted Samples. If we think about a distribution as a device for computing expectations — which are integrals — we can obtain a representation of a distribution from the integration method described above. This representation will consist of a set of weighted points. Assume that f is non-negative, and $\int f(\boldsymbol{U})d\boldsymbol{U}$ exists and is finite. Then

$$\frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

is a probability density function representing the distribution of interest. We shall write this probability density function as $p_f(\boldsymbol{X})$.

Now we have a collection of N points $\boldsymbol{u}^i \sim S(\boldsymbol{U})$, and a collection of weights $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$. Using this notation, we have that

$$\begin{aligned}\mathrm{E}\left[\frac{1}{N}\sum_i w^i\right] &= \int 1\frac{f(\boldsymbol{U})}{s(\boldsymbol{U})}s(\boldsymbol{U})d\boldsymbol{U} \\ &= \int f(\boldsymbol{U})d\boldsymbol{U}\end{aligned}$$

Algorithm 2.1: Obtaining a sampled representation of a probability distribution

Represent a probability distribution

$$p_f(\boldsymbol{X}) = \frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

by a set of N weighted samples

$$\left\{(\boldsymbol{u}^i, w^i)\right\}$$

where $\boldsymbol{u}^i \sim s(\boldsymbol{u})$ and $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$.

Now this means that

$$\begin{aligned}
\mathrm{E}_{p_f}[g] &= \int g(\boldsymbol{U})p_f(\boldsymbol{U})d\boldsymbol{U} \\
&= \frac{\int g(\boldsymbol{U})f(\boldsymbol{U})d\boldsymbol{U}}{\int f(\boldsymbol{U})d\boldsymbol{U}} \\
&= \mathrm{E}\left[\frac{\sum_i g(\boldsymbol{u}_i)w_i}{\sum_i w_i}\right] \\
&\approx \frac{\sum_i g(\boldsymbol{u}_i)w_i}{\sum_i w_i}
\end{aligned}$$

(where we have cancelled some N's). This means that we can *in principle* represent a probability distribution by a set of weighted samples (algorithm 1). There are some significant practical issues here, however. Before we explore these, we will discuss how to perform various computations with sampled representations. We have already shown how to compute an expectation (above, and algorithm 2). There are two other important activities for tracking: marginalisation, and turning a representation of a prior into a representation of a posterior.

Marginalising a Sampled Representation.

An attraction of sampled representations is that some computations are particularly easy. Marginalisation is a good and useful example. Assume we have a sampled representation of $p_f(\boldsymbol{U}) = p_f((\boldsymbol{M}, \boldsymbol{M}))$. We write $\boldsymbol{U}$ as two components $(\boldsymbol{M}, \boldsymbol{N})$ so that we can marginalise with respect to one of them.

Now assume that the sampled representation consists of a set of samples which we can write as

$$\left\{((\boldsymbol{m}^i, \boldsymbol{n}^i), w^i)\right\}$$

In this representation, $(\boldsymbol{m}^i, \boldsymbol{n}^i) \sim s(\boldsymbol{M}, \boldsymbol{N})$ and $w^i = f((\boldsymbol{m}^i, \boldsymbol{n}^i))/s((\boldsymbol{m}^i, \boldsymbol{n}^i))$.

We want a representation of the marginal $p_f(\boldsymbol{M}) = \int p_f(\boldsymbol{M}, \boldsymbol{N})d\boldsymbol{N}$. We will use this marginal to estimate integrals, so we can derive the representation by

Algorithm 2.2: Computing an expectation using a set of samples

We have a representation of a probability distribution

$$p_f(\boldsymbol{X}) = \frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

by a set of weighted samples

$$\left\{(\boldsymbol{u}^i, w^i)\right\}$$

where $\boldsymbol{u}^i \sim s(\boldsymbol{u})$ and $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$. Then:

$$\int g(\boldsymbol{U})p_f(\boldsymbol{U})d\boldsymbol{U} \approx \frac{\sum_{i=1}^N g(\boldsymbol{u}^i)w^i}{\sum_{i=1}^N w^i}$$

thinking about integrals. In particular

$$\begin{aligned}\int g(\boldsymbol{M})p_f(\boldsymbol{M})d\boldsymbol{M} &= \int g(\boldsymbol{M}) \int p_f(\boldsymbol{M}, \boldsymbol{N})d\boldsymbol{N}d\boldsymbol{M} \\ &= \int \int g(\boldsymbol{M})p_f(\boldsymbol{M}, \boldsymbol{N})d\boldsymbol{N}d\boldsymbol{M} \\ &\approx \frac{\sum_{i=1}^N g(\boldsymbol{m}^i)w^i}{\sum_{i=1}^N w^i}\end{aligned}$$

meaning that we can represent the marginal by dropping the $\boldsymbol{n}^i$ components of the sample (or ignoring them, which may be more efficient!).

Transforming a Sampled Representation of a Prior into a Sampled Representation of a Posterior.

Appropriate manipulation of the weights of a sampled distribution yields representations of other distributions. A particularly interesting case is representing a posterior, given some measurement. Recall that

$$\begin{aligned}p(\boldsymbol{U}|\boldsymbol{V} = v_0) &= \frac{p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})}{\int p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})d\boldsymbol{U}} \\ &= \frac{1}{K}p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})\end{aligned}$$

where $\boldsymbol{v}_0$ is some measured value taken by the random variable $\boldsymbol{V}$.

Assume we have a sampled representation of $p(\boldsymbol{U})$, given by $\left\{(\boldsymbol{u}^i, w^i)\right\}$. We

Algorithm 2.3: Computing a representation of a marginal distribution

Assume we have a sampled representation of a distribution

$$p_f(\boldsymbol{M}, \boldsymbol{N})$$

given by

$$\left\{((\boldsymbol{m}^i, \boldsymbol{n}^i), w^i)\right\}$$

Then

$$\left\{(\boldsymbol{m}^i, w^i)\right\}$$

is a representation of the marginal,

$$\int p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N}$$

can evaluate K fairly easily:

$$\begin{aligned} K &= \int p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{U}) p(\boldsymbol{U}) d\boldsymbol{U} \\ &= \mathrm{E}\left[\frac{\sum_{i=1}^N p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i) w^i}{\sum_{i=1}^N w^i}\right] \\ &\approx \frac{\sum_{i=1}^N p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i) w^i}{\sum_{i=1}^N w^i} \end{aligned}$$

Now let us consider the posterior.

$$\begin{aligned} \int g(\boldsymbol{U}) p(\boldsymbol{U}|\boldsymbol{V} = v_0) d\boldsymbol{U} &= \frac{1}{K} \int g(\boldsymbol{U}) p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{U}) p(\boldsymbol{U}) d\boldsymbol{U} \\ &\approx \frac{1}{K} \frac{\sum_{i=1}^N g(\boldsymbol{u}^i) p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i) w^i}{\sum_{i=1}^N w^i} \\ &\approx \frac{\sum_{i=1}^N g(\boldsymbol{u}^i) p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i) w^i}{\sum_{i=1}^N p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i) w^i} \end{aligned}$$

(where we substituted the approximate expression for K in the last step). This means that, if we take $\left\{(\boldsymbol{u}^i, w^i)\right\}$ and replace the weights with

$$w'^i = p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i) w^i$$

the result $\left\{(\boldsymbol{u}^i, w'^i)\right\}$ is a representation of the posterior.

2.2.2 The Simplest Particle Filter

Assume that we have a sampled representation of $P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$, and we need to obtain a representation of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$. We will follow the usual two

Algorithm 2.4: Transforming a sampled representation of a prior into a sampled representation of a posterior.

Assume we have a representation of $p(\boldsymbol{U})$ as

$$\left\{(\boldsymbol{u}^i, w^i)\right\}$$

Assume we have an observation $\boldsymbol{V} = \boldsymbol{v}_0$, and a likelihood model $p(\boldsymbol{V}|\boldsymbol{U})$. The posterior, $p(\boldsymbol{U}|\boldsymbol{V} = \boldsymbol{v}_0)$ is represented by

$$\left\{(\boldsymbol{u}^i, w'^i)\right\}$$

where

$$w'^i = p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i)w^i$$

steps of prediction and correction.

We can regard each sample as a possible state for the process at step $\boldsymbol{X}_{i-1}$. We are going to obtain our representation by firstly representing

$$P(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

and then marginalising out $\boldsymbol{X}_{i-1}$ (which we know how to do). The result is the prior for the next state, and, since we know how to get posteriors from priors, we will obtain $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$.

Prediction. Now

$$p(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) = p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1})p(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

Write our representation of $p(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as

$$\left\{(\boldsymbol{u}_{i-1}^k, w_{i-1}^k)\right\}$$

(the superscripts index the samples for a given step i, and the subscript gives the step).

Now for any given sample $\boldsymbol{u}_{i-1}^k$, we can obtain samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}_{i-1}^k)$ fairly easily. This is because our dynamic model is

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{x}_{i-1}) + \xi_i$$

where $\xi_i \sim N(0, \Sigma_{m_i})$. Thus, for any given sample $\boldsymbol{u}_{i-1}^k$, we can generate samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}_{i-1}^k)$ as

$$\left\{(f(\boldsymbol{u}_{i-1}^k) + \xi_i^l, 1)\right\}$$

where $\xi_i^l \sim N(0, \Sigma_{m_i})$. The index l indicates that we might generate several such samples for each $\boldsymbol{u}_{i-1}^k$.

We can now represent $p(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as

$$\left\{((f(\boldsymbol{u}_{i-1}^k) + \xi_i^l, \boldsymbol{u}_{i-1}^k), w_{i-1}^k)\right\}$$

(notice that there are *two* free indexes here, k and l; by this we mean that, for each sample indexed by k, there might be several different elements of the set, indexed by l).

Because we can marginalise by dropping elements, the representation of

$$P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

is given by

$$\left\{(f(\boldsymbol{u}_{i-1}^k) + \xi_i^l, w_{i-1}^k)\right\}$$

(we walk through a proof in the exercises). We will reindex this collection of samples — which may have more than N elements — and rewrite it as

$$\left\{(\boldsymbol{u}_i^{k,-}, w_i^{k,-})\right\}$$

assuming that there are M elements. Just as in our discussion of Kalman filters, the superscript '$-$' indicates that this our representation of the i'th state before a measurement has arrived. The superscript k gives the individual sample.

Correction. Correction is simple: we need to take the prediction, which acts as a prior, and turn it into a posterior. We do this by choosing an appropriate weight for each sample, following algorithm 4. The weight is

$$p(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-})w_i^{k,-}$$

(you should confirm this by comparing with algorithm 4). and our representation of the posterior is

$$\left\{(\boldsymbol{s}_i^{k,-}, p(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-})w_i^{k,-})\right\}$$

The Tracking Algorithm. In principle, we now have most of a tracking algorithm — the only missing step is to explain where the samples of $p(\boldsymbol{X}_0)$ came from. The easiest thing to do here is to start with a diffuse prior of a special form that is easily sampled — a Gaussian with large covariance might do it — and give each of these samples a weight of 1. It is a good idea to implement this tracking algorithm to see how it works (exercises!); you will notice that it works poorly even on the simplest problems (figure 2.3 compares estimates from this algorithm to exact expectations computed with a Kalman filter). The algorithm gives bad estimates because most samples represent no more than wasted computation. In jargon, the samples are called **particles**.

If you implement this algorithm, you will notice that weights get small very fast; this isn't obviously a problem, because the mean value of the weights is cancelled in the division, so we could at each step divide the weights by their mean

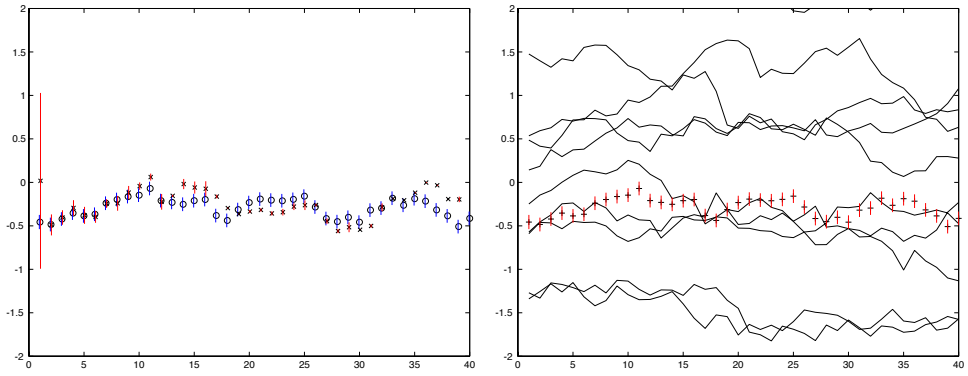

FIGURE 2.3: The simple particle filter behaves very poorly, as a result of a phenomenon called **sample impoverishment**, which is rather like quantisation error. In this example, we have a point on the line drifting on the line (i.e. $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise. In this case, we can get an exact representation of the posterior using a Kalman filter. In the figure on the **left**, we compare a representation obtained exactly using a Kalman filter with one computed from simple particle filtering. We show the mean of the posterior as a point with a *one* standard deviation bar (previously we used three standard deviations, but that would make these figures difficult to interpret). The mean obtained using a Kalman filter is given as an x; the mean obtained using a particle filter is given as an o; we have offset the standard deviation bars from one another so as to make the phenomenon clear. Notice that the mean is poor, but the standard deviation estimate is awful, and gets worse as the tracking proceeds. In particular, the standard deviation estimate woefully underestimates the standard deviation — this could mislead a user into thinking the tracker was working and producing good estimates, when in fact it is hopelessly confused. The figure on the **right** indicates what is going wrong; we plot the tracks of ten particles, randomly selected from the 100 used. Note that relatively few particles ever lie within one standard deviation of the mean of the posterior; in turn, this means that our representation of $P(x_{i+1}|y_0, \ldots, y_0)$ will tend to consist of many particles with very low weight, and only one with a high weight. This means that the density is represented very poorly, and the error propagates.

value. If you implement this step, you will notice that very quickly one weight becomes close to one and all others are extremely small. It is a fact that, in the simple particle filter, the variance of the weights cannot decrease with i (meaning that, in general, it will increase and we will end up with one weight very much larger than all others).

If the weights are small, our estimates of integrals are likely to be poor. In particular, a sample with a small weight is positioned at a point where $f(\boldsymbol{u})$ is much smaller than $p(\boldsymbol{u})$; in turn (unless we want to take an expectation of a function which is very large at this point) this sample is likely to contribute relatively little to the estimate of the integral.

Generally, the way to get accurate estimates of integrals is to have samples that lie where the integral is likely to be large — we certainly don't want to miss these points. We are unlikely to want to take expectations of functions that vary quickly, and so we would like our samples to lie where $f(\boldsymbol{u})$ is large. In turn, this

means that a sample whose weight w is small represents a waste of resources — we'd rather replace it with another sample with a large weight. This means that the effective number of samples is decreasing — some samples make no significant contribution to the expectations we might compute, and should ideally be replaced (figure 2.3 illustrates this important effect). In the following section, we describe ways of maintaining the set of particles that lead to effective and useful particle filters.

2.2.3 A Workable Particle Filter

Particles with very low weights are fairly easily dealt with — we will adjust the collection of particles to emphasize those that appear to be most helpful in representing the posterior. This will help us deal with another difficulty, too. In discussing the simple particle filter, we did not discuss how many samples there were at each stage — if, at the prediction stage, we drew several samples of $P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{s}_{i-1}^{k,+})$ for each $\boldsymbol{s}_{i-1}^{k,+}$, the total pool of samples would grow as i got bigger. Ideally, we would have a constant number of particles N. All this suggests that we need a method to discard samples, ideally concentrating on discarding unhelpful samples. There are a number of strategies that are popular.

Resampling the Prior. At each step i, we have a representation of

$$P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

via weighted samples. This representation consists of N (possibly distinct) samples, each with an associated weight. Now in a sampled representation, the frequency with which samples appear can be traded off against the weight with which they appear. For example, assume we have a sampled representation of $P(\boldsymbol{U})$ consisting of N pairs $(\boldsymbol{s}_k, w_k)$. Form a new set of samples consisting of a union of N_k copies of $(\boldsymbol{s}_k, 1)$, for each k. If

$$\frac{N_k}{\sum_k N_k} = w_k$$

this new set of samples is also a representation of $P(\boldsymbol{U})$ (you should check this).

Furthermore, if we take a sampled representation of $P(\boldsymbol{U})$ using N samples, and draw N' elements from this set with replacement, uniformly and at random, the result will be a representation of $P(\boldsymbol{U})$, too (you should check this, too). This suggests that we could (a) expand the sample set and then (b) subsample it to get a new representation of $P(\boldsymbol{U})$. This representation will tend to contain multiple copies of samples that appeared with high weights in the original representation.

This procedure is equivalent to the rather simpler process of making N draws with replacement from the original set of samples, using the weights w_i as the probability of drawing a sample. Each sample in the new set would have weight 1; the new set would predominantly contain samples that appeared in the old set with large weights. This process of resampling might occur at every frame, or only when the variance of the weights is too high.

Resampling Predictions.

Algorithm 2.5: A practical particle filter resamples the posterior.

Initialization: Represent $P(\boldsymbol{X}_0)$ by a set of N samples

$$\left\{(\boldsymbol{s}_0^{k,-}, w_0^{k,-})\right\}$$

where

$$\boldsymbol{s}_0^{k,-} \sim P_s(\boldsymbol{S}) \text{ and } w_0^{k,-} = P(\boldsymbol{s}_0^{k,-})/P_s(\boldsymbol{S} = \boldsymbol{s}_0^{k,-})$$

Ideally, $P(\boldsymbol{X}_0)$ has a simple form and $\boldsymbol{s}_0^{k,-} \sim P(\boldsymbol{X}_0)$ and $w_0^{k,-} = 1$.
Prediction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_{i-1})$ by

$$\left\{(\boldsymbol{s}_i^{k,-}, w_i^{k,-})\right\}$$

where

$$\boldsymbol{s}_i^{k,-} = f(\boldsymbol{s}_{i-1}^{k,+}) + \xi_i^k \text{ and } w_i^{k,-} = w_{i-1}^{k,+} \text{ and } \xi_i^k \sim N(0, \Sigma_{d_i})$$

Correction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_i)$ by

$$\left\{(\boldsymbol{s}_i^{k,+}, w_i^{k,+})\right\}$$

where

$$\boldsymbol{s}_i^{k,+} = \boldsymbol{s}_i^{k,-} \text{ and } w_i^{k,+} = P(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-}) w_i^{k,-}$$

Resampling: Normalise the weights so that $\sum_i w_i^{k,+} = 1$ and compute the variance of the normalised weights. If this variance exceeds some threshold, then construct a new set of samples by drawing, with replacement, N samples from the old set, using the weights as the probability that a sample will be drawn. The weight of each sample is now $1/N$.

A slightly different procedure is to generate several samples of $P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{s}_{i-1}^{k,+})$ for each $\boldsymbol{s}_{i-1}^{k,+}$, make N draws, with replacement, from this set using the weights w_i as the probability of drawing a sample, to get N particles. Again, this process will emphasize particles with larger weight over those with smaller weights.

The Consequences of Resampling.

Figure 2.4 illustrates the improvements that can be obtained by resampling. Resampling is not a uniformly benign activity, however: it is possible — but unlikely — to lose important particles as a result of resampling, and resampling can be expensive computationally if there are many particles.

Algorithm 2.6: An alternative practical particle filter.

Initialization: Represent $P(\boldsymbol{X}_0)$ by a set of N samples

$$\left\{(\boldsymbol{s}_0^{k,-}, w_0^{k,-})\right\}$$

where

$$\boldsymbol{s}_0^{k,-} \sim P_s(\boldsymbol{S}) \text{ and } w_0^{k,-} = P(\boldsymbol{s}_0^{k,-})/P_s(\boldsymbol{S} = \boldsymbol{s}_0^{k,-})$$

Ideally, $P(\boldsymbol{X}_0)$ has a simple form and $\boldsymbol{s}_0^{k,-} \sim P(\boldsymbol{X}_0)$ and $w_0^{k,-} = 1$.
Prediction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_{i-1})$ by

$$\left\{(\boldsymbol{s}_i^{k,-}, w_i^{k,-})\right\}$$

where

$$\boldsymbol{s}_i^{k,l,-} = f(\boldsymbol{s}_{i-1}^{k,+}) + \xi_i^l \text{ and } w_i^{k,l,-} = w_{i-1}^{k,+}$$

and

$$\xi_i^l \sim N(0, \Sigma_{d_i})$$

and the free index l indicates that each $\boldsymbol{s}_{i-1}^{k,+}$ generates M different values of $\boldsymbol{s}_i^{k,l,-}$. This means that there are now MN particles.
Correction: We reindex the set of MN samples by k. Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_i)$ by

$$\left\{(\boldsymbol{s}_i^{k,+}, w_i^{k,+})\right\}$$

where

$$\boldsymbol{s}_i^{k,+} = \boldsymbol{s}_i^{k,-} \text{ and } w_i^{k,+} = P(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-}) w_i^{k,-}$$

Resampling: As in algorithm 5.

2.2.4 If's, And's and But's — Practical Issues in Building Particle Filters

Particle filters have been extremely successful in many practical applications in vision, but can produce some nasty surprises. One important issue has to do with the number of particles; while the expected value of an integral estimated with a sampled representation is the true value of the integral, it may require a very large number of particles before the variance of the estimator is low enough to be acceptable. It is difficult to say how many particles will be required to produce usable estimates. In practice, this problem is usually solved by experiment.

Unfortunately, these experiments may be misleading. You can (and should!) think about a particle filter as a form of search — we have a series of estimates of state, which we update using the dynamic model, and then compare to the data; estimates which look as though they could have yielded the data are kept, and the others are discarded. The difficulty is that we may miss good hypotheses. This

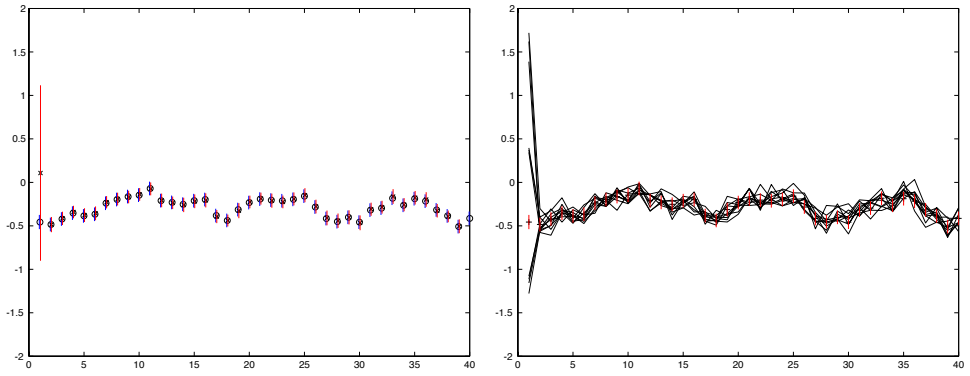

FIGURE 2.4: Resampling hugely improves the behaviour of a particle filter. We now show a resampled particle filter tracking a point drifting on the line (i.e. $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise, and are the same as for figure 2.3. In the figure on the **left**, we compare an exact representation obtained using a Kalman filter with one computed from simple particle filtering. We show the mean of the posterior as a point with a one standard deviation bar. The mean obtained using a Kalman filter is given as an 'x'; the mean obtained using a particle filter is given as an 'o'; we have offset the standard deviation bars from one another so as to make the phenomenon clear. Notice that estimates of both mean and standard deviation obtained from the particle filter compare well with the exact values obtained from the Kalman filter. The figure on the **right** indicates where this improvement came from; we plot the tracks of ten particles, randomly selected from the 100 used. Because we are now resampling the particles according to their weights, particles that tend to reflect the state rather well usually reappear in the resampled set. This means that many particles lie within one standard deviation of the mean of the posterior, and so the weights on the particles tend to have much smaller variance, meaning the representation is more efficient.

could occur if, for example, the likelihood function had many narrow peaks. We may end up with updated estimates of state that lie in some, but not all of these peaks; this would result in good state hypotheses being missed. While this problem can (just!) be caused to occur in one dimension, it is particularly serious in high dimensions. This is because real likelihood functions can have many peaks, and these peaks are easy to miss in high dimensional spaces. It is extremely difficult to get good results from particle filters in spaces of dimension much greater than about 10.

The problem can be significant in low dimensions, too — its significance depends, essentially, on how good a prediction of the likelihood we can make. This problem manifests itself in the best-known fashion when one uses a particle filter to track people. Because there tend to be many image regions that are long, roughly straight, and coherent, it is relatively easy to obtain many narrow peaks in the likelihood function — these correspond, essentially, to cases where the configuration for which the likelihood is being evaluated has a segment lying over one of these long, straight coherent image regions. While there are several tricks for addressing this problem — all involve refining some form of search over the likelihood — there is no standard solution yet.

2.3 TRACKING PEOPLE WITH PARTICLE FILTERS

Tracking people is difficult. The first difficulty is that there is a great deal of state to a human — there are many joint angles, etc. that may need to be represented. The second difficulty is that it is currently very hard to find people in an image — this means that it can be hard to initiate tracks. Most systems come with a rich collection of constraints that must be true before they can be used. This is because people have a large number of degrees of freedom: bits of the body move around, we can change clothing, etc., which means it is quite difficult to predict appearance.

People are typically modelled as a collection of body segments, connected with rigid transformations. These segments can be modelled as cylinders — in which case, we can ignore the top and bottom of the cylinder and any variations in view, and represent the cylinder as an image rectangle of fixed size — or as ellipsoids. The state of the tracker is then given by the rigid body transformations connecting these body segments (and perhaps, various velocities and accelerations associated with them).

Both particle filters and (variants of) Kalman filters have been used to track people. Each approach can be made to succeed, but neither is particularly robust. There are two components to building a particle filter tracker: firstly, we need a motion model and secondly, we need a likelihood model.

We can use either a strong motion model — which can be obtained by attaching markers to a model and using them to measure the way the model's joint angles change as a function of time — or a weak motion model — perhaps a drift model. Strong motion models have some disadvantages: perhaps the individual we are tracking moves in a funny way; and we will need different models for walking, walking carrying a weight, jogging and running (say). The difficulty with a weak motion model is that we are pretty much explicitly acknowledging that each frame is a poor guide to the next.

Likelihood models are another source of difficulties, because of the complexity of the relationship between the tracker's state and the image. The likelihood function (P(image features|person present at given configuration)) tends to have many local extrema. This is because the likelihood function is evaluated by, in essence, rendering a person using the state of the tracker and then comparing this rendering to the image. Assume that we know the configuration of the person in the previous image; to assess the likelihood of a particular configuration in the current image, we use the configuration to compute a correspondence between pixels in the current image and in the previous image. The simplest likelihood function can be obtained using the sum of squared differences between corresponding pixel values — this assumes that clothing is rigid with respect to the human body, that pixel values are independent given the configuration, and that there are no shading variations. These are all extremely dubious assumptions.

Of course, we choose which aspects of an image to render and to compare; we might use edge points instead of pixel values, to avoid problems with illumination. Multiple extrema in the likelihood can be caused by: the presence of many extended coherent regions, which look like body segments, in images; the presence of many edge points unrelated to the person being tracked (this is a problem if we use edge points in the comparison); changes in illumination; and changes in the appearance

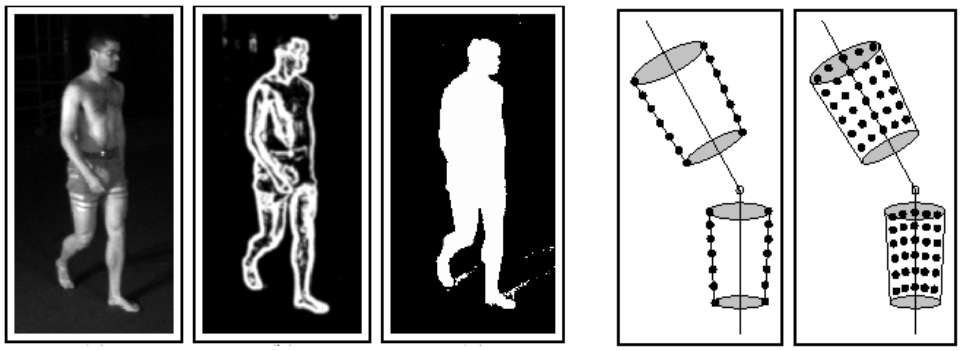

FIGURE 2.5: A typical likelihood computation identifies points in the image that provide evidence that a person is present. In one use of a particle filter for tracking people, Deutscher, Blake and Reid look for two types of evidence: the first is boundary information, and the second is "non-background" information. Boundary points are estimated using an edge detector, and "non-background" points are obtained using background subtraction; the figure on the **left** illustrates these types of point. On the **far left**, the image; **center left**, points near edges obtained using smoothed gradient estimates; **near left**, points where there is motion, obtained using background subtraction. Now each particle gives the state of the person, and so can be used to determine where each body segment lies in an image; this means we can predict the boundaries of the segments and their interiors, and compute a score based on the number of edge points near segment boundaries and the number of "non-background" points inside projected segments (**near right** shows sample points that look for edges and **far right** shows sample points that look for moving points). *Figure from "Articulated Body Motion Capture by Annealed Particle Filtering," J. Deutscher, A. Blake and I. Reid, Proc. Computer Vision and Pattern Recognition 2000*

of body segments caused by clothing swinging on the body. The result is a tendency for trackers to drift (see, for example, the conclusions in [Sidenbladh *et al.*, 2000b]; the comments in [Yacoob and Davis, 2000]).

In all of the examples we show, the tracker must be started by hand. An alternative to using a strong motion model is to use a weak motion model and rely on the search component of particle filtering. The best example of this approach is a device, due to Deutscher *et al.*, known as an **annealed particle filter** (figure 2.6), which essentially searches for the global extremum through a sequence of smoothed versions of the likelihood [Deutscher *et al.*, 2000]. However, clutter creates peaks that can require intractable numbers of particles. Furthermore, this strategy requires a detailed search of a high dimensional domain (the number of people being tracked times the number of parameters in the person model plus some camera parameters).

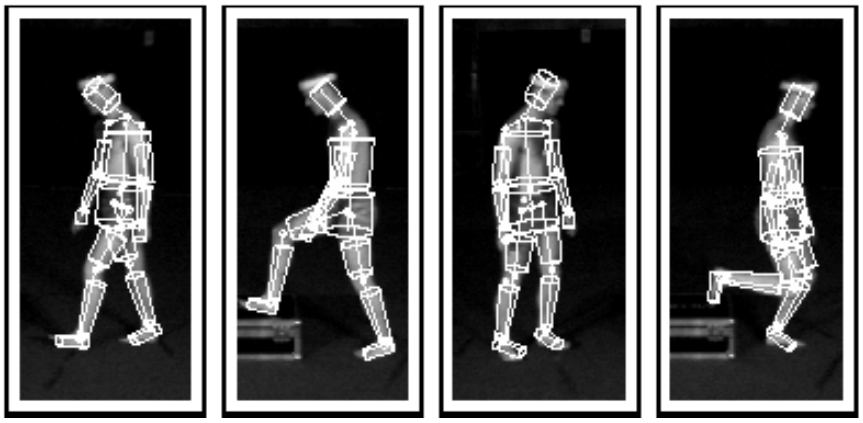

FIGURE 2.6: If a weak motion model is used to track a person with a particle filter, the likelihood function can create serious problems. This is because the state is high-dimensional, and there are many local peaks in the likelihood — even for a person on a black background, as in these images. It is quite possible that none of our particles are near the peaks, meaning that the filter's representation becomes unreliable. One way to deal with this is to search the likelihood by *annealing* it. This process creates a series of increasingly smooth approximations to the likelihood, whose peaks lie close to or on the peaks of the likelihood. We weight particles with a smoothed approximation, then resample the particles according to their weights, allow them to drift, then weight them with a less smooth approximation, etc. The result is a random search through the likelihood that should turn up the main local minima. This yields a tracker that can track people on simple backgrounds, but requires only very general motion models — the tracker illustrated above models human motion as drift. *Figure from "Articulated Body Motion Capture by Annealed Particle Filtering," J. Deutscher, A. Blake and I. Reid, Proc. Computer Vision and Pattern Recognition 2000* © *2000, IEEE*

2.4 NOTES

Space has forced us to omit some topics, important in radar tracking and likely to become important in vision. Firstly, the radar community is accustomed to generating tracks automatically; this practice is not unknown in vision, but most trackers are initialized by hand. Secondly, the radar community is accustomed to tracking multiple targets, with multiple returns; this complicates data association significantly, which is now seen as a weighted matching problem and dealt with using one of the many exact algorithms for that problem. Thirdly, the radar community is accustomed to dealing with tracking for objects that can switch dynamic model — this leads to so-called IMM filters, where a series of filters with distinct dynamic models propose different updates, and these proposals are weighted by evidence and updated (for a good general summary of tracking practice in radar, see [Blackman and Popoli, 1999]). There is a little work on this topic in the vision community, but it tends to be somewhat difficult to do with a particle filter (actually, this means it needs an inconvenient number of particles; you can do pretty much anything with a particle filter, if you have enough particles).

Topic	What you must know
Sampled representations	A probability distribution is a device for computing expectations, which are integrals. A set of samples from one probability distribution can be weighted such that a weighted sum of function values taken at the samples is a good estimate of the expectation of that function with respect to some (possibly different) probability distribution. Such a set of samples is called a sampled representation.
Particle filtering	A collection of sample states is maintained so that the samples represent the posterior distribution. Typically, these states are propagated forward in time according to the dynamics, compared with observations, weighted according to the observation likelihood, and then resampled to remove samples with low weights. The algorithm should be seen as a randomised search of the likelihood, using the prior as a proposal process. This approach useful and powerful if (a) the state space is of low dimension and (b) the likelihood function is not too complicated.
Applications of particle filters	Particle filters have been widely applied to tracking kinematic representations of human figures. In this application, they have had considerable success, but currently require manual initialisation; this is a significant impediment to their adoption in practical applications.

Chapter summary for chapter 2: *When object dynamics is not linear, the posterior distributions encountered in tracking problems tend to have forms that are difficult to represent. Particle filtering is an inference algorithm that is well suited to tracking non-linear dynamics.*

The Particle Filter

We have been able to provide only a brief overview of a subject that is currently extremely active. We have deliberately phrased our discussion rather abstractly, so as to bring out the issues that are most problematic, and to motivate a view of particle filters as convenient approximations. Particle filters have surfaced in a variety of forms in a variety of literatures. The statistics community, where they originated, knows them as particle filters (e.g. [Kitagawa, 1987]; see also the collection [Doucet *et al.*, 2001]). In the AI community, the method is sometimes called survival of the fittest [Kanazawa *et al.*, 1995]. In the vision community, the method is sometimes known as condensation [Isard and Blake, 1996; Blake and Isard, 1996; Blake and Isard, 1998].

Particle filters have been the subject of a great deal of work in vision. Much of the work attempts to sidestep the difficulties with likelihood functions that we sketched in the particle filtering section (see, in particular, the annealing method of [Deutscher *et al.*, 2000] and the likelihood corrections of [Sullivan *et al.*, 1999]). Unfortunately, all uses of the particle filter have been relentlessly top-down — in the sense that one updates an estimate of state and then computes some comparison between an image and a rendering, which is asserted to be a likelihood. While this strategy represents an effective end-run around data association, it means that we

are committed to searching rather nasty likelihoods.

There is a strong analogy between particle filters and search. This can be used to give some insight into what they do and where they work well. For example, a high dimensional likelihood function with many peaks presents serious problems to a particle filter. This is because there is no reason to believe that any of the particles each step advances will find a peak. This is certainly not an intrinsic property of the technique — which is just an algorithm — and is almost certainly a major strategic error. The consequence of this error is that one can track almost anything with a particle filter (as long as the dimension of the state space is small enough) but it has to be initialized by hand. This particular ghost needs to be exorcised from the party as soon as possible.

The exorcism will probably involve thinking about how to come up with clean probabilistic models that (a) allow fast bottom-up inference and (b) don't involve tangling with likelihoods with as complex a form as those commonly used. We expect an exciting struggle with this problem over the next few years.

Particle filters are an entirely general inference mechanism (meaning that they can be used to attack complex inference problems uniting high level and low level vision [Isard and Blake, 1998b; Isard and Blake, 1998a]). This should be regarded as a sign that it can be very difficult to get them to work, because there are inference problems that are, essentially, intractable. One source of difficulties is the dimension of the state space — it is silly to believe that one can represent the covariance of a high-dimensional distribution with a small number of particles, unless the covariance is very strongly constrained. A particular problem is that it can be quite hard to tell when a particle filter is working — obviously, if the tracker has lost track, there is a problem, but the fact that the tracker seems to be keeping track is not necessarily a guarantee that all is well. For example, the covariance estimates may be poor; we need to ask for how long the tracker will keep track; etc.

One way to simplify this problem is to use tightly parametrised motion models. This reduces the dimension of the state space in which we wish to track, but at the cost of not being able to track some objects *or* of being compelled to choose which model to use. This approach has been extremely successful in applications like gesture recognition [Black and Jepson, 1998]; tracking moving people [Sidenbladh *et al.*, 2000a]; and classifying body movements [Rittscher and Blake, 1999]. A tracker could track the state of its own platform, instead of tracking a moving object [Dellaert *et al.*, 1999].

There are other methods for maintaining approximations of densities. One might, for example, use a mixture of Gaussians with a constant number of components. It is rather natural to do data association by averaging, which will result in the number of elements in the mixture going up at each step; one is then supposed to cluster the elements and cull some components. We haven't seen this method used in vision circles yet.

Starting a People Tracker

Desiderata for a tracking application are:

- that tracks are initiated automatically;

- that tracks can be discarded automatically, as necessary (this means that the occasional erroneous track won't affect the count of total objects);
- that the tracker can be shown to work robustly over long sequences of data.

We discussed relatively few of the many kinematic human trackers, because none can meet these tests. It would be nice if this remark were obsolete by the time this book reaches its readers, but we don't think this will be the case (which is why we made it!). Tracking people on a general background remains extremely challenging; the difficulty is knowing how to initiate the track, which is hard because the variations in the appearance of clothing mean that it is generally difficult to know which pixels come from a person. Furthermore, the inference problem is very difficult, because the conditional independence assumptions that simplify finding people no longer apply — the position of the upper arm in frame n, say, depends on *both* the position of the torso in frame n *and* the position of the upper arm in frame $n-1$. It is possible to evade this difficulty in the first instance by assembling multi-frame motion vectors [Song *et al.*, 1999; Song *et al.*, 2000], but these too have unpleasant dependencies over time (the motion of the upper arm in frame n, etc.), and the consequences of ignoring these dependencies are unknown.

Typically, current person trackers either initialize the tracker by hand, use aggressively simplified backgrounds which have high contrast with the moving person, or use background subtraction. These tricks are justified, because they make it possible to study this (extremely important) problem, but they yield rather unconvincing applications.

There is currently (mid 2001) no person tracker that represents the configuration of the body and can start automatically; all such trackers use manual starting methods. One way to start such a tracker would be to find all possible people, and then track them. But finding people is difficult, too. No published method can find clothed people in arbitrary configurations in complex images. There are three standard approaches to finding people described in the literature. Firstly, the problem can be attacked by template matching (examples include [Oren *et al.*, 1997], where upright pedestrians with arms hanging at their side are detected by a template matcher; [Niyogi and Adelson, 1995; Liu and Picard, 1996; Cutler and Davis, 2000], where walking is detected by the simple periodic structure that it generates in a motion sequence; [Wren *et al.*, 1995; Haritaoglu *et al.*, 2000], which rely on background subtraction — that is, a template that describes "non-people"). Matching templates to people (rather than to the background) is inappropriate if people are going to appear in multiple configurations, because the number of templates required is too high. This motivates the second approach, which is to find people by finding faces (sections **??** and **??**, and [Poggio and Sung, 1995; Rowley *et al.*, 1996a; Rowley *et al.*, 1996b; Rowley *et al.*, 1998a; Rowley *et al.*, 1998b; Sung and Poggio, 1998]). The approach is most successful when frontal faces are visible.

The third approach is to use the classical technique of search over correspondence (search over correspondences between point features is an important early formulation of object recognition; the techniques we describe have roots in [Faugeras and Hebert, 1986; Grimson and Lozano-Pérez, 1987; Thompson and Mundy, 1987; Huttenlocher and Ullman, 1987]). In this approach, we search over correspondence

between image configurations and object features. There are a variety of examples in the literature (for a variety of types of object; see, for example, [Huang *et al.*, 1997; Ullman, 1996]). Perona and collaborators find faces by searching for correspondences between eyes, nose and mouth and image data, using a search controlled by probabilistic considerations [Leung *et al.*, 1995; Burl *et al.*, 1995]. Unclad people are found by [Fleck *et al.*, 1996; Forsyth and Fleck, 1999], using a correspondence search between image segments and body segments, tested against human kinematic constraints. A much improved version of this technique, which learns the model from data, appears in [Ioffe and Forsyth, 1998].

II APPENDIX: THE EXTENDED KALMAN FILTER, OR EKF

We consider non-linear dynamic models of the form

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i); \Sigma_{d_i})$$

Again, we will need to represent with

$$P(\boldsymbol{x}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

(for prediction) and

$$P(\boldsymbol{x}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$$

(for correction). We take the position that these distributions can be represented by supplying a mean and a covariance. Typically, the representation works only for distributions that look rather like normal distributions — a big peak at one spot, and then a fast falloff. To obtain an extended Kalman filter, we linearize the dynamics about the current operating point, and linearize the measurement model. We do not derive the filter equations (it's a dull exercise in Laplace's approximation to integrals), but simply present them in algorithm 7. We write the Jacobian of a function $\boldsymbol{g}$ — this is the matrix whose l, m'th entry is

$$\frac{\partial f_l}{\partial x_m}$$

— as $\mathcal{J}(\boldsymbol{g})$, and when we want to show that it has been evaluated at some point $\boldsymbol{x}_j$, we write $\mathcal{J}(\boldsymbol{g}; \boldsymbol{x}_j)$.

Algorithm 2.7: The extended Kalman filter maintains estimates of the mean and covariance of the various distributions encountered while tracking a state variable of some fixed dimension using the given non-linear dynamic model.

Dynamic Model:

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i), \Sigma_{d_i})$$

$$\boldsymbol{y}_i \sim N(\boldsymbol{h}(\boldsymbol{x}_i, i), \Sigma_{m_i})$$

Start Assumptions: $\overline{\boldsymbol{x}}_0^-$ and Σ_0^- are known
Update Equations: Prediction

$$\overline{\boldsymbol{x}}_i^- = \boldsymbol{f}(\overline{\boldsymbol{x}}_{i-1}^+)$$

$$\Sigma_i^- = \Sigma_{d_i} + \mathcal{J}(\boldsymbol{f}; \overline{\boldsymbol{x}}_{i-1}^+)^{-T} \Sigma_{i-1}^+ \mathcal{J}(\boldsymbol{f}; \overline{\boldsymbol{x}}_{i-1}^+)^{-1}$$

Update Equations: Correction

$$\mathcal{K}_i = \Sigma_i^- \mathcal{J}_{\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-}^T \left[\mathcal{J}(\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-) \Sigma_i^- \mathcal{J}(\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-)^T + \Sigma_{m_i} \right]^{-1}$$

$$\overline{\boldsymbol{x}}_i^+ = \overline{\boldsymbol{x}}_i^- + \mathcal{K}_i \left[\boldsymbol{y}_i - \boldsymbol{h}(\overline{\boldsymbol{x}}_i^-, i) \right]$$

$$\Sigma_i^+ = \left[Id - \mathcal{K}_i \mathcal{J}(\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-) \right] \Sigma_i^-$$