Contents

CHAPTER 1

Tracking

Notes: *link the detect, mean shift stuff to the kalman filter via data association toward the end* **Notes:** *Example (s) for track by detect; less flow material, with less detail; wierd data association needs fixing; be more explicit about dynamical models and their usefulness*

Tracking is the problem of generating an inference about the motion of an object given a sequence of images. Generally, we will have some measurements that appear at each tick of a (notional) clock. These measurements could be the position of some image points, the position and moments of some image regions, or pretty much anything else. They are not guaranteed to be relevant, in the sense that some could come from the object of interest and some might come from other objects or from noise. We will have an encoding of the object's state, and some model of how this state changes from tick to tick. We would like to infer the state of the world from the measurements and the model of dynamics.

Tracking problems are of great practical importance. There are very good reasons to want to, say, track aircraft using radar returns (good summary histories include [4, 5, 12]; comprehensive reviews of technique in this context include [1, 3, 21]). Other important applications include:

- **Motion Capture:** If we can track the 3D configuration of a moving person accurately, then we can make an accurate record of their motions. Once we have this record, we can use it to drive a rendering process; for example, we might control a cartoon character, thousands of virtual extras in a crowd scene, or a virtual stunt avatar. Furthermore, we could modify the motion record to obtain slightly different motions. This means that a single performer can produce sequences they wouldn't want to do in person.

- **Recognition From Motion:** the motion of objects is quite characteristic. We may be able to determine the identity of the object from its motion; we should be able to tell what it's doing.

- **Surveillance:** Knowing what objects are doing can be very useful. For example, different kinds of trucks should move in different, fixed patterns in an airport; if they do not, then something is going wrong. Similarly, there are combinations of places and patterns of motions that should never occur (e.g., no truck should ever stop on an active runway). It could be helpful to have a computer system that can monitor activities and give a warning if it detects a problem case.

- **Targeting:** A significant fraction of the tracking literature is oriented toward (a) deciding what to shoot, and (b) hitting it. Typically, this literature describes tracking using radar or infrared signals (rather than vision), but the basic issues are the same — what do we infer about an object's future position from a sequence of measurements? Where should we aim?

Generally, we regard a moving object as having a **state**. This state — which may not be observed directly — encodes all the properties of the object we care to deal with, or need to encode its motion. For example, state might contain: position; position and velocity; position, velocity and acceleration; position and appearance; and so on. This state changes at each tick of time, and we then get new measurements which depend on the new state. These measurements are referred to as **observations** In many problems, the observations are measurements of state, perhaps incorporating some noise. For example, the state might be the position of the object, and we observe its position. In other problems, the observations are functions of state. For example, the state might be position and velocity, but we observe only position. In some tracking problems, we have a model of how the state changes with time. The information in this model is referred to as the object's **dynamics**. Tracking involves exploiting both observations and dynamics to infer state.

The most important property of visual tracking problems is that observations are usually hidden in a great deal of irrelevant information. For example, if we wish to track a face in a video frame, in most cases the face occupies fewer than a third of the pixels in the video frame. In almost every case, the pixels that do not lie on the face have nothing useful to offer about the state of the face. This means that we face significant problems identifying which observations are likely to be helpful. The main methods for doing so involve either building a detector (section **??**), or exploiting the tendency for objects to look the same over time, and to move coherently (section **??**). Dynamical information is more or less helpful depending on the details of the application, but a significant number of vision applications involve objects that are subject to large, poorly modelled accelerations. In this case, dynamical information tends not to be helpful. However, once we have identified useful observations, incorporating dynamical information is relatively straightforward algorithmically (sections **??** and **??**).

1.1 TRACKING BY DETECTION

Assume that we will only see one object in each frame of video, that the state we wish to track is position in the image, and that we can build a reliable detector for the object we wish to track. Then tracking is straightforward: we report the location of the detector response in each frame of the video. This observation is a good source of simple and effective tracking strategies, because we can build good detectors for some objects. For example, consider tracking a red ball on a green background, where the detector might just look for red pixels. In other cases, we might need to use a more sophisticated detector: for example, we might wish to track a frontal face looking at a camera (face detectors are discussed in detail in section **??**). There is a range of simple procedures that we can use to make this strategy apply to cases where there are many objects in frame, where detections are sometimes missed, and so on (section **??**). An extremely important case is a small image patch, and we can build strong tracking by detection strategies for such patches (section **??**).

1.1.1 Tracking Multiple Objects with Unreliable Detectors

In most cases, we can't assume only one object, or a reliable detector. If objects can enter or leave the frame (or if the detector occasionally fails to detect something), then it isn't enough to just report the location of an object at each frame. We must account for the fact that some frames have too many (or too few) objects in them. To do this, we will have an abstraction called a **track**, which represents a timeline for a single object. Assume that we have tracked for a while, and wish to deal with a new frame. We copy the tracks from the previous frame to this frame, and then allocate object detector responses to tracks. How we allocate depends on the application (we give some examples below). Each track will get at most one detector response and each detector response will get at most one track. However, some tracks may not receive a detector response and some detector responses may not be allocated a track. Finally, we deal with tracks that have no response and with responses that have no track. For every detector response that is not allocated to a track, we create a new track (because a new object may have appeared). For every track that has not received a response for several frames, we prune that track (because the object may have disappeared). Finally, we may postprocess the set of tracks to insert links where justified by the application. Algorithm 1 breaks out this approach.

The main issue in allocation is the cost model, which will vary from application to application. We need a charge for allocating detects to tracks. For slow moving objects, this charge could be the image distance between the detect in the current frame and the detect allocated to the track in the previous frame. For objects with slowly changing appearance, the cost could be an appearance distance (e.g. a χ-squared distance between color histograms). How we use the distance again depends on the application. In cases where the detector is very reliable, the objects are few, well spaced and slow-moving, then a greedy algorithm (allocate the closest detect to each track) is sufficient. This algorithm might attach one detector response to two tracks; whether this is a problem or not depends on the application. The more general algorithm solves a bipartite matching problem. The tracks form one side of a bipartite graph, and the detector responses are the other side. Each side is augmented by NULL nodes, so that a track (or response) can go unmatched. The edges are weighted by matching costs, and we must solve a maximum weighted bipartite matching problem (Figure **??**; see []), which can be solved by the Hungarian algorithm (see, for example, []). In some cases, we know where objects can appear and disappear, so that tracks can only be created for detects that occur in some region, and tracks can be reaped only if the last detect occurs in a disappear region.

Background subtraction is often a good enough detector in applications where the background is known and all trackable objects look different from the background. In such cases, it can be enough to apply background subtraction, and regard the big blobs as detector responses. This strategy is simple, but can be very effective. One useful case occurs for people seen on a fixed background, like a corridor or a parking lot. If the application doesn't require a detailed report of the body configuration, and if we expect people to be reasonably large in view, we can reason that large blobs produced by background subtraction are individual people.

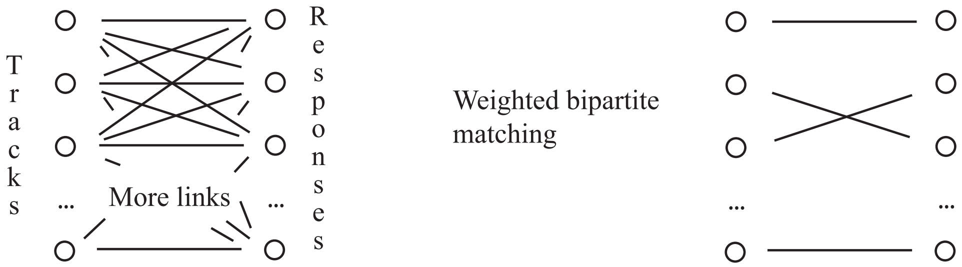

FIGURE 1.1: *In tracking by detection, we must associate detector responses in the current frame with tracks carried over from the previous frame. We compute a weight for associating a particular track with a particular detector response; this weight could involve information about how fast objects are expected to move, and about the appearance of the response. Large weights are good and small weights are bad. We must choose the matching that has the largest sum of weights. This is a maximum weighted bipartite matching problem, where the one side of the bipartite graph is the tracks and the other is the responses. A new track is created for any response not matched to a track. Tracks that are not matched to responses for several frames are dropped. In some cases, it is useful to attach null tracks and null responses with a fixed association weight, to encourage tracks to be unmatched.*

Although this method has weaknesses — for example, if people are still for a long time they might disappear; it would require more work to split up the large blob of foreground pixels that occurs when two people are close together; and so on — many applications require only approximate reports of the traffic density, or alarms if a person appears in a particular view. The method is well suited to these such cases.

This basic recipe for tracking by detection is worth remembering. In many situations, nothing more complex is required. The trick of creating tracks promiscuously and then pruning any track that has not received a measurement for some time is a quite general and extremely effective trick.

1.1.2 Matching Small Image Patches

Assume we have a small image patch in the start image of a sequence. It has enough structure that we could expect to find it in other images (we will be precise about what this means later). At the same time, it is small enough that we do not expect complex deformations — the most that can happen is an affine transformation. We can build a tracker by finding that patch in the second frame. We use the location in the second frame to find the patch in the third frame, and so on. In each case, we assume that there has been no significant deformation from frame to frame, that is, that the patch has not foreshortened or shrunk very significantly in an interframe interval, or, equivalently, the rotational and translational velocity of objects is limited. With this assumption, it is enough to use a translation model to find the patch in the next frame. To check the result, we will compare each match with the original using a more sophisticated deformation model (an affine

Algorithm 1.1: The simplest tracking by detection

Notation:
Write $\mathbf{x}_k(i)$ for the k'th response of the detector in the i'th frame
Write $t(k,i)$ for the k'th track in the i'th frame
Write $*t(k,i)$ for the detector response attached to the k'th track in the i'th frame
(Think C pointer notation)

Assumptions: We have a detector which is reasonably reliable.
We know some distance d such that $d(*t(k,i-1), *t(k,i))$ is always small.

First frame: Create a track for each detector response.

N'th frame:
Link tracks and detector responses by solving a bipartite matching problem.
Spawn a new track for each detector response not allocated to a track.
Reap any track that has not received a detector response for some number of frames.

Cleanup: We now have trajectories in space time. Link any where this is justified (perhaps by a more sophisticated dynamical or appearance model, derived from the candidates for linking).

transformation), and if the comparison is poor enough, we will end that track (the patch may have become occluded, and so on). This is a form of tracking by detection, using the patch as a model.

There are two important steps. The first is to find how a patch has moved from frame t to frame $t+1$. In the most usual form of this algorithm, the patch is found by minimizing the **sum of squared differences** error, or **SSD**. Write $\mathcal{P}_t$ for the indices of the patch in the t'th frame and $I(\boldsymbol{x},t)$ for the t'th frame. Assume that the patch is at $\boldsymbol{x}_t$ in the t'th frame, and it translates to $\boldsymbol{x}_t+\boldsymbol{h}$ in the $t+1$'th frame. Then we can determine $\boldsymbol{h}$ by minimizing

$$E(\boldsymbol{h}) = \sum_{\boldsymbol{u}\in\mathcal{P}_t} \left[I(\boldsymbol{u},t) - I(\boldsymbol{u}+\boldsymbol{h},t+1)\right]^2$$

as a function of $\boldsymbol{h}$. The minimum of the error occurs when

$$\nabla_h E(\boldsymbol{h}) = 0.$$

Now if $\boldsymbol{h}$ is small, we can write $I(\boldsymbol{u}+\boldsymbol{h},t+1) \approx I(\boldsymbol{u},t) + \boldsymbol{h}^T\nabla I$ where ∇I is the image gradient. Substituting, and rearranging, we get

$$\left[\sum_{\boldsymbol{u}\in\mathcal{P}_t} (\nabla I)(\nabla I)^T\right]\boldsymbol{h} = \sum_{\boldsymbol{u}\in\mathcal{P}_t}\left[I(\boldsymbol{u},t) - I(\boldsymbol{u},t+1)\right]\nabla I$$

which is a linear system we could solve directly for $\boldsymbol{h}$. The solution of this system will be unreliable if the smaller eigenvalue of the symmetric positive semidefinite matrix $\left[\sum_{\boldsymbol{u}\in\mathcal{P}_t}(\nabla I)(\nabla I)^T\right]$ is too small. This occurs when the image gradients in $\mathcal{P}$ are all small — so the patch is featureless — or all point in one direction — so that we cannot localize the patch along that flow direction. If the estimate of $\boldsymbol{h}$ is unreliable, we must end the track.

The second important step is comparing the patch at the new location to the original patch in the start frame. For small time changes, it is reasonable to model an image patch as just translating. However, over a long time interval the surface on which the patch lies might rotate in 3D, and so the image patch will deform. This means that, if we simply translate the original patch to the current location, we may see a poor match even if the location is right. To get a better estimate of the match, we must assume a more complex deformation model. Because the image patch is small, an affine model is appropriate. The affine model means that the point $\boldsymbol{x}$ in frame one will become the point $\mathcal{M}\boldsymbol{x}+\boldsymbol{c}$ in frame t. To estimate $\mathcal{M}$ and $\boldsymbol{c}$, we will minimize

$$E(\mathcal{M},\boldsymbol{c}) = \sum_{\boldsymbol{u}\in\mathcal{P}_1}\left[I(\boldsymbol{u},1) - I(\mathcal{M}\boldsymbol{u}+\boldsymbol{c},t)\right]^2 .$$

Notice that, because we are comparing the original patch in frame 1 with that in the current frame, the sum is over $\boldsymbol{u}\in\mathcal{P}_1$. The expression for $\max M$ and $\boldsymbol{c}$, given below, is developed using the same analysis as for the translation. `Expression for M, c` Once we have $\mathcal{M}$ and $\boldsymbol{c}$, we can evaluate the SSD between the current patch and its original, and if this is below a threshold the match is acceptable.

`picture; algorithm`

These two steps lead to a quite flexible mechanism. We can start tracks using an interest point operator, perhaps a corner detector. To build a tracker that can create and reap tracks as necessary, we find all interest points in frame one. We then find the location of each of these in the next frame, and check whether the patch matches the original one. If so, it belongs to a track. If not, the track has ended. We now look for interest points or corners that don't belong to tracks and create new tracks there. Again, we advance tracks to the next frame, check each against their original patch, reap tracks whose patch doesn't match well enough and create tracks at new interest points. In section **??**, we show how to link this procedure with a dynamical model.

1.2 TRACKING USING MATCHING

The most complex practical model for the deformation of a small image patch is an affine transformation. While perspective effects occur, on small spatial scales they are hard to observe or to estimate. For a large patch, however, much more complicated motion models can occur. This means that it is harder to build detectors for large patches; for example, we have quite reliable frontal face detectors, but body segment detectors don't work terribly well. But if we know an appropriate motion model, we can exploit the tendency of objects to have coherent appearance over time. For example, if we are sure that body segments will only translate in the image, we can use the arguments that a segment is rectangular, and that the segment in one frame will look like the same segment in the previous frame to estimate that translation. As another example, imagine tracking a face in a webcam, which is useful for those building user interfaces where the camera on top of a display views the computer user. The face is not necessarily frontal, because computer users occasionally look away from their monitors, and so a detector will not work. But a face tends to be tends to be blobby, tends to have coherent appearance, and tends only to translate and rotate.

As with the strategy of section **??**, we have a domain of interest in the n'th image, $\mathcal{D}_n$, and we must search for a matching domain $\mathcal{D}_{n+1}$ in the $n+1$'st image. Now our motion model is more complex. There are two types of match we can work with. In **summary matching**, we match summary representations of the whole domain. We will represent a domain with a set of parameters; for example, we could work with circular domains of fixed radius, and represent the domain by the location of the center. We then compute a summary of the appearance within the circle $\mathcal{D}_n$ and find the best-matching circle $\mathcal{D}_{n+1}$ (section 1.2.1). In **flow-based matching**, we search for a transformation of the pixels in the old domain that produces set of pixels that match well, and so a good new domain. This allows us to exploit strong motion models (section 1.2.2).

1.2.1 Matching Summary Representations

Look at the football player's uniform in figure 1.2.1. From frame to frame, we see the player's back at different viewing angles. Individual pixels in one domain may have no corresponding pixels in the next. For example, the cloth may have folded slightly; as another example, there is motion blur in some frames. Nonetheless, the domain is largely white, with some yellow patches. This suggests that a summary representation of the domain may not change from frame to frame, even though the fine details do.

There is a quite general idea here. Write the domain of interest in frame n as $\mathcal{D}_n$. If we are tracking a deforming object, pixels in $\mathcal{D}_n$ may have no corresponding pixels in $\mathcal{D}_{n+1}$ or the motion of the pixels might be extremely complex, and so we should represent $\mathcal{D}_n$ with a well-behaved summary. If the patches deform, small scale structures should be preserved, but the spatial layout of these structures may not be. Example small scale structures include the colors of pixels, or the responses of oriented filters. A histogram representation of these structures is attractive, because two histograms will be similar only if the two patches have similar numbers of similar structures in them, but the similarity is not disrupted by deformation.

We assume that we have a parametric domain, with parameters $\boldsymbol{y}$, so that $\boldsymbol{y}_n$ represents $\mathcal{D}_n$. For our treatment we assume the domain is a circle of fixed radius whose center is at the pixel location $\boldsymbol{y}$, but the method can be reworked to apply to other kinds of domain. The **mean shift** procedure yields one way to find the $\mathcal{D}_{n+1}$ whose histogram is most like that of $\mathcal{D}_n$.

Measuring the distance between histograms.

We assume that the features we are working with can be quantized so that the histogram can be represented as a vector of bin counts, and we write this vector as $\boldsymbol{p}(\boldsymbol{y})$ and its u'th component representing the count in the u'th bin is $p_u(\boldsymbol{y})$. We wish to find the $\boldsymbol{y}$ whose histogram is closest to that at $\boldsymbol{y}_n$. We are comparing two probability distributions which we can do with the **Bhattacharyya coefficient** which is

$$\rho(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{p}(\boldsymbol{y}_n)) = \sum_u \sqrt{p_u(\boldsymbol{y}) p_u(\boldsymbol{y}_n)}$$

which will be one if the two are the same and near zero if they are very different. To obtain a distance function, we can work with

$$d(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{p}(\boldsymbol{y}_n)) = \sqrt{1 - \rho(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{p}(\boldsymbol{y}_n))}.$$

We will obtain $\boldsymbol{y}_{n+1}$ by minimizing this distance. We will start this search at $\boldsymbol{y}_{n+1}^{(0)}$. We assume that $\boldsymbol{y}_{n+1}$ is close to $\boldsymbol{y}_{n+1}^{(0)}$, and that as a result $\boldsymbol{p}(\boldsymbol{y}_{n+1})$ is similar to $\boldsymbol{p}(\boldsymbol{y}_{n+1}^{(0)})$. In this case, a Taylor expansion of $\rho(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{p}(\boldsymbol{y}_n))$ about $\boldsymbol{p}(\boldsymbol{y}_{n+1}^{(0)})$ gives

$$\begin{aligned}
\rho(\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{p}(\boldsymbol{y}_n)) &\approx \sum_u \sqrt{p_u(\boldsymbol{y}_{n+1}^{(0)}) p_u(\boldsymbol{y}_n)} + \sum_u (p_u(\boldsymbol{y}) - p_u(\boldsymbol{y}_{n+1}^{(0)})) \left(\frac{1}{2} \sqrt{\frac{p_u(\boldsymbol{y}_n)}{p_u(\boldsymbol{y}_{n+1}^{(0)})}} \right) \\
&= \frac{1}{2} \sum_u \sqrt{p_u(\boldsymbol{y}_{n+1}^{(0)}) p_u(\boldsymbol{y}_n)} + \frac{1}{2} \sum_u p_u(\boldsymbol{y}) \sqrt{\frac{p_u(\boldsymbol{y}_n)}{p_u(\boldsymbol{y}_{n+1}^{(0)})}}
\end{aligned}$$

and this means that, to minimize the distance, we must maximize

$$\frac{1}{2} \sum_u p_u(\boldsymbol{y}) \sqrt{\frac{p_u(\boldsymbol{y}_n)}{p_u(\boldsymbol{y}_{n+1}^{(0)})}}. \tag{1.1}$$

Building a Histogram. Now we need a method to construct a histogram vector for the circle with center $\boldsymbol{y}$. We expect we are tracking a deforming object, so that pixels far away from the center of two matching circles may be quite different. To deal with this, we should allow pixels far away from the center to have a much smaller effect on the histogram than those close to the center. We can do this with a **kernel smoother**. Write the feature vector (for example, the color) for the pixel at location $\boldsymbol{x}_i$ in the circle as $\boldsymbol{f}_i^{(n)}$. This feature vector is d-dimensional. Write the histogram bin corresponding to $\boldsymbol{f}_i^{(n)}$ as $b(\boldsymbol{f}_i^{(n)})$. Each pixel votes into its bin in the histogram with a weight that decreases with $\parallel \boldsymbol{x}_i - \boldsymbol{y} \parallel$ according to a **kernel profile** k. Two common choices are the **Epanechnikov profile**, which is

$$k_e(r) = \begin{cases} \frac{1}{2c_d}(d+2)(1-r) & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases}$$

(here c_d is the volume of the d-dimensional sphere) or the **normal profile**, given by

$$k_n(r) = 2\pi^{\frac{-d}{2}} \exp\left(\frac{-r}{2}\right).$$

The weight of the vote in bin u produced by $\boldsymbol{f}_i^{(n)}$ is given by $k(\parallel \boldsymbol{f}_i^{(n)} - u \parallel^2)$, where k is a kernel profile. Using this approach, the fraction of total votes in bin u produced by all features is

$$p_u(\boldsymbol{y}) = C_h \sum_{i \in \mathcal{D}_n} k(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \delta \left[b(\boldsymbol{f}_i - u) \right] \tag{1.2}$$

where h is a scale, chosen by experiment, and C_h is a normalizing constant to ensure that the sum of histogram components is one. This constant is given by

$$C_h = \sum_u \sum_{i \in \mathcal{D}_n} k(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \delta \left[b(\boldsymbol{f}_i - u) \right]$$

Substituting equation 1.2 into equation 1.1, we must maximize

$$f(\boldsymbol{y}) = \frac{C_h}{2} \sum_i w_i k(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \tag{1.3}$$

where

$$w_i = \sum_u \delta \left[b(\boldsymbol{f}_i - u) \right] \sqrt{\frac{p_u(\boldsymbol{y}_n)}{p_u(\boldsymbol{y}_{n+1}^{(0)})}}.$$

Locating a Domain with Mean Shift.

The mean-shift procedure maximizes expressions of the form of equation 1.3. Write g for the derivative of the kernel profile k. We are seeking $\boldsymbol{y}$ such that

$$\begin{aligned}
\nabla f &= 0 \\
&= \frac{C_h}{2} \sum_i w_i \nabla k(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \\
&= \frac{C_h}{h} \sum_i w_i \left[\boldsymbol{x}_i - \boldsymbol{y} \right] \left[g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \right] \\
&= \frac{C_h}{h} \left[\frac{\sum_i w_i \boldsymbol{x}_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)}{\sum_i w_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)} - \boldsymbol{y} \right] \times \left[\sum_i w_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \right].
\end{aligned}$$

We expect that $\sum_i w_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)$ is non-zero, so that the maximum occurs when

$$\left[\frac{\sum_i w_i \boldsymbol{x}_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)}{\sum_i w_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)} - \boldsymbol{y} \right] = 0$$

or equivalently, when

$$\boldsymbol{y} = \frac{\sum_i w_i \boldsymbol{x}_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)}{\sum_i w_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2)}$$

The mean shift procedure involves producing a series of estimates $\boldsymbol{y}^{(j)}$ where

$$\boldsymbol{y}^{(j+1)} = \frac{\sum_i w_i \boldsymbol{x}_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}^{(j)}}{h} \|^2)}{\sum_i w_i g(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}^{(j)}}{h} \|^2)}.$$

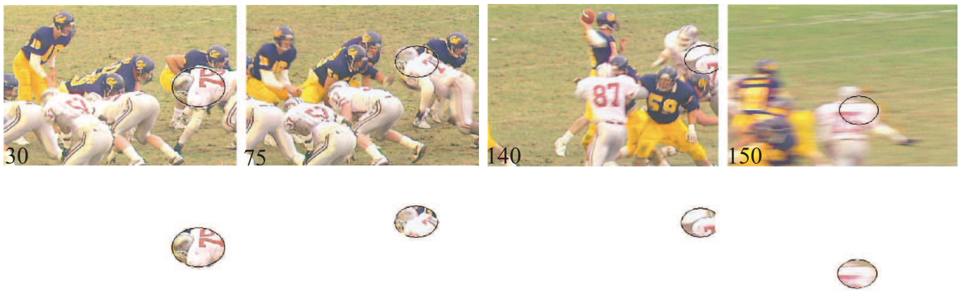

FIGURE 1.2: *Four frames from a sequence depicting football players, with superimposed domains. The object to be tracked is the blob on top of player 78 (at the center right in frame 30). We have masked off these blobs (*below*) to emphasize just how strongly the pixels move around in the domain. Notice the motion blur in the final frame. These blobs can be matched to one another, and this is done by comparing histograms (in this case, color histograms), which are less affected by deformation than individual pixel values. Figure from "Kernel-Based Object Tracking" Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer, IEEE TPAMI 2003, © IEEE 2003*Shown in draft in the fervent hope of receiving permission for final version

The procedure gets its name from the fact that we are shifting to a point which has the form of a weighted mean. The complete algorithm appears in algorithm **??**.

Notes: *link to sift feature ideas; and perhaps to naive bayes*

1.2.2 Matching and Flow

Assume we have a television view of a soccer field with players running around. Each player might occupy a box about 10-30 pixels high, so it would be hard to determine where arms and legs are (Figure **??**). The frame rate is 30Hz, and body parts don't move all that much (compared to the resolution) from frame to frame. In a case like this, we can assume that the domain translates. We can model a player's motion with two components. The first is the absolute motion of a box fixed around the player and the second is the player's movement relative to that box. To do so, we need to track the box, a process known as **image stabilization**. As another example of how useful image stabilization is, one might stabilize a box around an aerial view of a moving vehicle; now the box contains all visual information about the vehicle's identity.

In each example, the box translates. If we have a rectangle in frame n, we can search for the rectangle of the same size in frame $n+1$ that is most like the original. We can take the approach of section 1.1.2, and use the sum-of-squared differences of pixel values as a test for similarity. If we write $\mathcal{R}_{ij}^{(n)}$ for the i, j'th pixel in the rectangle in the n'th image, we choose $\mathcal{R}^{(n+1)}$ to minimize

$$\sum_{i,j}(\mathcal{R}_{ij}^{(n)} - \mathcal{R}_{ij}^{(n+1)})^2.$$

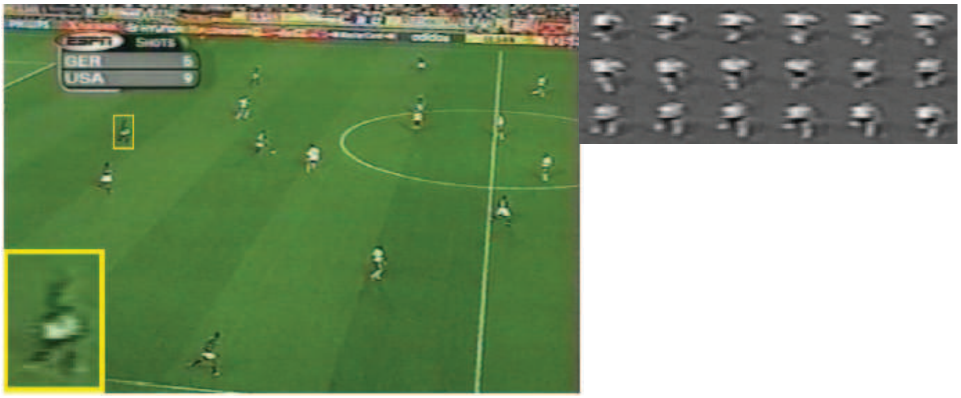

FIGURE 1.3: *A useful application of tracking by matching is to stabilize an image box around a more interesting structure, in this case a football player in a television-resolution video. A frame from the video is shown on the* **left***. Inset is a box around a player, zoomed to a higher resolution. Notice that the limbs of the player span a few pixels, are blurry and are hard to resolve. A natural feature to use to infer what the player is doing can be obtained by stabilizing the box around the player, then measuring the motion of the limbs with respect to the box. Players move relatively short distances between frames, and their body changes configuration a relatively small amount. This means the new box can be found by searching all nearby boxes of the same size to get the box whose pixels best match those of the original. On the* **right***, a set of stabilized boxes; the strategy is enough to center the player in a box.* Figure from "Recognizing Action at a Distance", Efros *et al.* , IEEE Int. Conf. Computer Vision 2003, © 2003 IEEE. Shown in draft in the fervent hope of receiving permission for final version

We could use the approximation of section 1.1.2 to minimize this expression. Alternatively, in some applications the distance the rectangle can move in an inter-frame interval is bounded because there are velocity constraints. If this distance is small enough, we could simply evaluate the sum of squared differences to every rectangle of the appropriate shape within that bound, or we might consider a search across scale for the matching rectangle (see section **??** for more information). **Notes:** *good example for scale pyramid search.*

In the image stabilization example, we compared two domains by placing pixels within the domain in correspondence then comparing the corresponding pixels. The correspondence is given by translating the pixels. More general correspondence models are possible, and the most natural form is a flow model. Here the correspondence is produced by a small motion, so that almost every pixel in frame n has a corresponding pixel in frame $n+1$ and vice versa. For a small movement, we will see relatively few new points, and lose relatively few points, so we can join each point in the first frame to its corresponding point on the second frame (which is overlaid) with an arrow. The head is at the point in the second frame, and, if the elapsed time is short, the field of arrows can be thought of as the instantaneous movement in the image. The arrows are known as the **optical flow**, a notion originally due

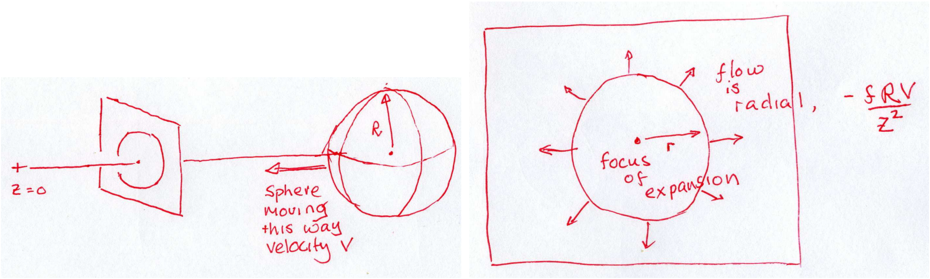

FIGURE 1.4: *A sphere of radius R approaches a camera along the Z-axis, at velocity V. The image is a circle, which grows as the sphere gets closer. The flow is radial, about a focus of expansion, and provides an estimate of the time to contact. This estimate works for other objects, too, and is used by gannets and flies (at least!).*

to Gibson [11].

Optical Flow and Motion .

Flow is particularly informative about relations between the viewer's motion, usually called **egomotion**, and the 3D scene. For example, when viewed from a moving car, distant objects have much slower apparent motion than close objects, so the rate of apparent motion can tell us something about distance. As another example, assume the egomotion is pure translation in some direction. Then the image point in that direction, which is known as the **focus of expansion** will not move, and all the optic flow will be away from that point (Figure **??**). This means that simply observing such a flow field tells us something about how we are moving. Further simple observations tell us how quickly we will hit something. Assume the camera points at the focus of expansion, and make the world move to the camera. A sphere of radius R whose center lies along the direction of motion and is at depth Z will produce a circular image region of radius $r = fR/Z$. If it moves down the Z axis with speed $V = dZ/dt$, the rate of growth of this region in the image will be $dr/dt = -fRV/Z^2$. This means that

$$\text{time to contact} = -\frac{Z}{V} = \frac{r}{\left(\frac{dr}{dt}\right)}.$$

The minus sign is because the sphere is moving down the Z-axis, so Z is getting smaller and V is negative.

The object doesn't need to be a sphere for this argument to work, and if the camera is spherical we don't need to be looking in the direction we are travelling either. This means that an animal that is translating fast can get an estimate of how long until it hits something very quickly and very easily.

Flow Models.

We can work with flow either by searching parametric flow fields for a good pixel correspondence, or by estimating an entire flow field. Generally, the latter is difficult to do accurately, because it involves estimating a large number of parameters (two per pixel). We will focus on parametric flow fields. The simplest method to build flow models is to take a pool of examples of the types of flow one would like to track, and try to find a set of basis flows that explains most of the variation (for examples, see [13]). In this case, writing θ_i for the i'th component of the parameter vector and $\boldsymbol{F}_i$ for the i'th flow basis vector field, one has

$$\boldsymbol{v}(\boldsymbol{x}) = \sum_i \theta_i \boldsymbol{F}_i$$

We can apply a singular value decomposition to reduce the number of parameters (section **??**). A second strategy is to assume that flows involve what are essentially 2D effects — this is particularly appropriate for lateral views of human limbs — so that a set of basis flows that encodes translation, rotation and some affine effects is probably sufficient. Write (x, y) for the components of $\boldsymbol{x}$. One can obtain such flows using the simple model

$$\boldsymbol{v}(\boldsymbol{x}) = \begin{pmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \\ \theta_8 \end{pmatrix} .$$

This model is linear in θ, and provides a reasonable encoding of flows resulting from 3D motions of a 2D rectangle (see figure 1.5).

1.2.3 Tracking using Flow

Tracking using flow is straightforward once we have a family of flow models. We write the image as a function of space and time as $\mathcal{I}(x, y, t)$, and scale and translate time so that each frame appears at an integer value of t. We have a domain in the n'th image, $\mathcal{D}_n$. We must find the domain in the $n+1$'th image that matches best under the flow model. We write $\rho(u, v)$ for a cost function that compares two pixel values u and v; this should be small when they match and large when they do not. We write $w(\boldsymbol{x})$ for a weighting of the cost function that depends on the location of the pixel. To find the new domain, we will find the best flow, and then allow our domain to follow that flow model. Finding the best flow involves minimizing

$$\sum_{\boldsymbol{x} \in \mathcal{D}_n} w(\boldsymbol{x}) \rho(\mathcal{I}(\boldsymbol{x}, n), \mathcal{I}(\boldsymbol{x} + \boldsymbol{v}(\boldsymbol{x}; \theta), n+1))$$

as a function of the flow parameters θ.

The cost function should not necessarily be the squared difference in pixel values. We might wish to compute a more complex description of each location (for example, a smoothed vector of filter outputs to encode local texture). Some pixels in the domain might be more reliable than others; for example, we might expect pixels near the boundary of the window to have more variation, and so we would

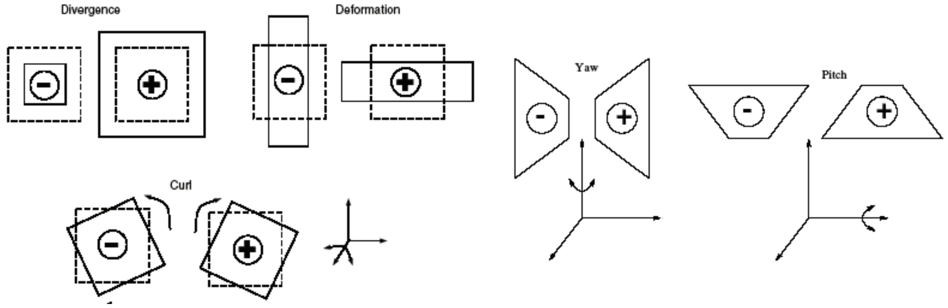

FIGURE 1.5: **Attribution:** Figure 2 from Ju, Black and Yacoob, "Cardboard People", no permission yet *Typical flows generated by the model* $(u(\boldsymbol{x}), v(\boldsymbol{x})^T = (\theta_1 + \theta_2 x + \theta_3 y + \theta_7 x^2 + \theta_8 xy, \theta_4 + \theta_5 x + \theta_6 y + \theta_y xy + \theta_8 y^2)$*. Different values of the* θ_i *give different flows, and the model can generate flows typical of a 2D figure moving in 3D.* **Divergence** *occurs when the image is scaled; for example,* $\theta = (0, 1, 0, 0, 0, 1, 0, 0)$*.* **Deformation** *occurs when one direction shrinks and another grows (for example, rotation about an axis parallel to the view plane in an orthographic camera); for example,* $\theta = (0, 1, 0, 0, 0, -1, 0, 0)$*.* **Curl** *can result from in plane rotation; for example,* $\theta = (0, 0, -1, 0, 1, 0, 0, 0)$*.* **Yaw** *models rotation about a vertical axis in a perspective camera; for example* $\theta = (0, 0, 0, 0, 0, 1, 0)$*. Finally,* **pitch** *models rotation about a horizontal axis in a perspective camera; for example* $\theta = (0, 0, 0, 0, 0, 0, 1)$*.*

weight them down compared to pixels near the center of the window. Robustness is another important issue. Outlier pixels, which are dramatically different from those predicted by the right transformation, could be caused by dead pixels in the camera, specularities, minor deformations on the object, and a variety of other effects. If we use a squared error metric, then such outlier pixels can have a disproportionate effect on the results. The usual solution is to adopt an M-estimator. A good choice of ρ is

$$\rho(u, v) = \frac{(u - v)^2}{(u - v)^2 + \sigma^2}$$

where σ is a parameter (there is greater detail on M-estimators in section **??**).

We now have the best value of θ, given by $\hat{\theta}$. The new domain is given by

$$\mathcal{D}_{n+1} = \left\{ \boldsymbol{u} \mid \boldsymbol{u} = \boldsymbol{x} + \boldsymbol{v}(\boldsymbol{x}; \hat{\theta}), \forall \boldsymbol{x} \in \mathcal{D}_n \right\}$$

We can build domain models that simplify estimating $\mathcal{D}_{n+1}$; for example, if the domain is always a circle, then the flow must represent a translation, rotation and scale, and we would allow the flow to act on the center, radius and orientation of the circle.

Tracking can be started in a variety of ways. For a while, it was popular to start such trackers by hand, but this is now rightly frowned on in most cases. In some cases, objects always appear in a known region of the image, and in that case one can use a detector to tell whether an object has appeared. Once it has appeared, the flow model takes over.

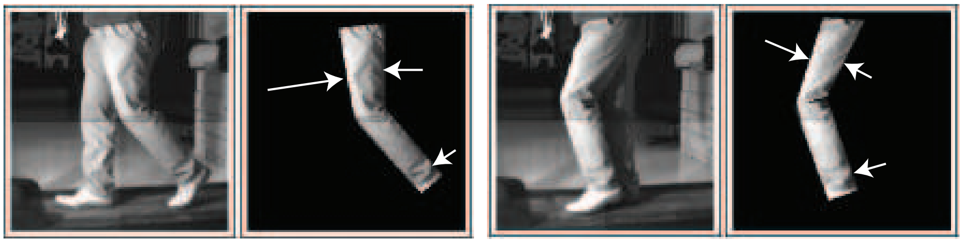

FIGURE 1.6: *An important pragmatic difficulty with flow-based tracking is that appearance is not always fixed. The folds in loose clothing depend on body configuration, as these images of trousers indicated. The trousers were tracked using a flow-based tracker, but enforcing equality between pixel values will be difficult, as the patches indicated by the arrows suggest. The folds are geometrically small, but, because they produce cast shadows, have a disproportionate effect on image brightness.*

The most important pragmatic difficulty with flow based trackers is their tendency to drift. A detection based tracker has a single appearance model for an object, encoded into the detector. This is applied to all frames. The danger is that this model might not properly account for changes in illumination, aspect and so on, and as a result will fail to detect the object in some frames. In contrast, a flow based tracker's model of the appearance of an object is based on what it looked like in the previous frame. This means that small errors in localization can accumulate. If the transformation estimated is slightly incorrect, then the new domain will be incorrect; but this means the new appearance model is incorrect, and might get worse. Correcting this drift requires a fixed, global model of appearance, like those of section **??**.

Another important pragmatic difficulty is that objects often don't have appearance as fixed as one would like. Loose clothing is a particularly important problem here, because it forms folds in different ways depending on the body configuration. These folds are very minor geometric phenomena, but can cause significant changes in image brightness, because they shadow patches of surface. This means that there can be a strong, time-varying texture signal that appears on the body segments (Figure **??**). While this signal almost certainly contains some cues to configuration, they appear to be very difficult to exploit.

1.3 TRACKING WITH PROBABILITY

Notes: *assume we are in a more difficult case and so on*

In section **??**, we described methods to match patches or object detector responses with tracks. This matching process is straightforward if we can be confident that the thing we are matching hasn't moved much — we search around the old location for the best match. To know where to search, we don't really need the object to be slow-moving. Instead, if it moves in a predictable way, the motion model can predict a search domain that might be far from the original location, but still reliable. Exploiting dynamical information effectively requires us to fuse information from observations with dynamical predictions. This is most easily done by building a probabilistic framework around the problem. The algorithmic goal is to maintain an accurate representation of the posterior on object state given observations and a dynamical model.

We model the object as having some internal state; the state of the object at the ith frame is typically written as $\boldsymbol{X}_i$. The capital letters indicate that this is a random variable — when we want to talk about a particular value that this variable takes, we use small letters. The measurements obtained in the ith frame are values of a random variable $\boldsymbol{Y}_i$; we write $\boldsymbol{y}_i$ for the value of a measurement, and, on occasion, we write $\boldsymbol{Y}_i = \boldsymbol{y}_i$ for emphasis. In **tracking**, (sometimes called filtering or state estimation) we wish to determine some representation of $P(X_k|Y_0, ..., Y_k)$. In **smoothing**, (sometimes called filtering) we wish to determine some representation of $P(X_k|Y_0, ..., Y_N)$ (i.e. we get to use "future" measurements to infer the state). These problems are massively simplified by two important assumptions.

- We assume measurements depend only the hidden state, that is, that $P(Y_k|X_0, ..., X_N, Y_0, ..., Y_N) = P(Y_k|X_k)$.
- We assume that the probability density for a new state is a function only of the previous state; that is, $P(X_k|X_0, ..., X_{k-1}) = P(X_k|X_{k-1})$, or, equivalently, that X_i form a **Markov chain**.

We will use these assumptions to build a recursive formulation for tracking around three steps:

- **Prediction:** We have seen $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{k-1}$ — what state does this set of measurements predict for the ith frame? To solve this problem, we need to obtain a representation of $P(\boldsymbol{X}_i|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_{k-1} = \boldsymbol{y}_{k-1})$. Straightforward manipulation of probability combined with the assumptions above yields that the **prior** or **predictive density** is

  $$P(X_k|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_{k-1} = \boldsymbol{y}_{k-1})) = \int P(X_k|X_{k-1})P(X_{k-1}|Y_0, ..., Y_{k-1})dX_{k-1}$$

- **Data association:** Some of the measurements obtained from the i-th frame may tell us about the object's state. Typically, we use $P(\boldsymbol{X}_i|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_{i-1} = \boldsymbol{y}_{i-1})$ to identify these measurements. For example, we might use this predictive density to build a search location for the methods of section **??**.
- **Correction:** Now that we have $\boldsymbol{y}_i$ — the relevant measurements — we need to compute a representation of $P(\boldsymbol{X}_i|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_i = \boldsymbol{y}_i)$. Straightforward manipulation of probability combined with the assumptions above yields that the **posterior** is

  $$P(X_k|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_k = \boldsymbol{y}_k) = \frac{P(Y_k = \boldsymbol{y}_k|X_k)P(X_k|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_k = \boldsymbol{y}_k)}{\int P(Y_k = \boldsymbol{y}_k|X_k)P(X_k|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_k = \boldsymbol{y}_k)dX_k}$$

Representing these probability distributions can be very difficult if the distributions have an arbitrary form. However, if the measurement models and the dynamical models are linear (in a sense to be described below), then all probability distributions will turn out to be Gaussian. In turn, this means that tracking and smoothing involve maintaining the values of the mean and covariance of the relevant densities (section 1.3.2)

1.3.1 Linear Measurements and Linear Dynamics

We will use the simplest possible measurement model, where the measurement is obtained from the state by multipying by some known matrix (which may depend on the frame), and then adding a normal random variable of zero mean and known covariance (which again may depend on the frame). We use the notation

$$\boldsymbol{x} \sim N(\boldsymbol{\mu}, \Sigma)$$

to mean that $\boldsymbol{x}$ is the value of a random variable with a normal probability distribution with mean $\boldsymbol{\mu}$ and covariance Σ. We write $\boldsymbol{x}_k$ for the state at step k. Our model is that $P(Y_k|X_k = \boldsymbol{x}_k)$ is a gaussian with mean $\mathcal{B}_k \boldsymbol{x}_k$ and covariance Σ. Using the notation above, we can write our model as

$$\boldsymbol{y}_k \sim N(\mathcal{B}_k \boldsymbol{x}_k, \Sigma_k).$$

This model may seem limited, but is very powerful (it is the cornerstone of a huge control industry). We do not need to observe the whole of the state vector at any given time to infer it. For example, if we have enough measurements of the position of a moving point we can deduce its velocity and its acceleration. This means that the matrix $\mathcal{B}_k$ does not need to have full rank (and in most practical cases it doesn't).

In the simplest possible dynamic model, the state is advanced by multiplying it by some known matrix (which may depend on the frame) and then adding a normal random variable of zero mean and known covariance. Similarly, the measurement is obtained by multiplying the state by some matrix (which may depend on the frame) and then adding a normal random variable of zero mean and known covariance. We can write our dynamic model as

$$x_i \sim N(\mathcal{D}_i \boldsymbol{x}_{i-1}; \Sigma_{d_i});$$

$$y_i \sim N(\mathcal{M}_i \boldsymbol{x}_i; \Sigma_{m_i}).$$

Notice that the covariances could be different from frame to frame as could the matrices. Although this model appears limited, it is in fact extremely powerful; we show how to model some common situations next.

Drifting Points. Let us assume that $\boldsymbol{x}$ encodes the position of a point. If $\boldsymbol{D}_i = Id$, then the point is moving under random walk — its new position is its old position plus some Gaussian noise term. This form of dynamics isn't obviously useful because it appears that we are tracking stationary objects. It is quite commonly used for objects for which no better dynamic model is known — we assume that the random component is quite large and hope we can get away with it.

This model also illustrates aspects of the **measurement matrix** $\mathcal{M}$. The most important thing to keep in mind is that we don't need to measure every aspect of the state of the point at every step. For example, assume that the point is in 3D: Now if $\mathcal{M}_{3k} = (0, 0, 1)$, $\mathcal{M}_{3k+1} = (0, 1, 0)$, and $\mathcal{M}_{3k+2} = (1, 0, 0)$, then at every third frame we measure, respectively, the z, y, or x position of the point. Notice that we can still expect to track the point, even though we measure only one component of its position at a given frame. If we have sufficient measurements, we can reconstruct the state — the state is **observable**. We explore observability in the exercises.

Constant Velocity. Assume that the vector $\boldsymbol{p}$ gives the position and $\boldsymbol{v}$ the velocity of a point moving with constant velocity. In this case, $\boldsymbol{p}_i = \boldsymbol{p}_{i-1} + (\Delta t)\boldsymbol{v}_{i-1}$ and $\boldsymbol{v}_i = \boldsymbol{v}_{i-1}$. This means that we can stack the position and velocity into a single state vector, and our model applies (Figure 1.7). In particular,

$$\boldsymbol{x} = \left\{ \begin{array}{c} \boldsymbol{p} \\ \boldsymbol{v} \end{array} \right\}$$

and

$$\mathcal{D}_i = \left\{ \begin{array}{cc} Id & (\Delta t)Id \\ 0 & Id \end{array} \right\}.$$

Notice that, again, we don't have to observe the whole state vector to make a useful measurement. For example, in many cases, we would expect that

$$\mathcal{M}_i = \left\{ \begin{array}{cc} Id & 0 \end{array} \right\}$$

(i.e., that we see only the position of the point). Because we know that it's moving with constant velocity — that's the model — we expect that we could use these measurements to estimate the whole state vector rather well.

Constant Acceleration. Assume that the vector $\boldsymbol{p}$ gives the position, vector $\boldsymbol{v}$ the velocity, and vector $\boldsymbol{a}$ the acceleration of a point moving with constant acceleration. In this case, $\boldsymbol{p}_i = \boldsymbol{p}_{i-1} + (\Delta t)\boldsymbol{v}_{i-1}$, $\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + (\Delta t)\boldsymbol{a}_{i-1}$, and $\boldsymbol{a}_i = \boldsymbol{a}_{i-1}$. Again, we can stack the position, velocity and acceleration into a single state vector, and our model applies (Figure 1.8). In particular,

$$\boldsymbol{x} = \left\{ \begin{array}{c} \boldsymbol{p} \\ \boldsymbol{v} \\ \boldsymbol{a} \end{array} \right\}$$

and

$$\mathcal{D}_i = \left\{ \begin{array}{ccc} Id & (\Delta t)Id & 0 \\ 0 & Id & (\Delta t)Id \\ 0 & 0 & Id \end{array} \right\}.$$

Notice that, again, we don't have to observe the whole state vector to make a useful measurement. For example, in many cases, we would expect that

$$\mathcal{M}_i = \left\{ \begin{array}{ccc} Id & 0 & 0 \end{array} \right\}$$

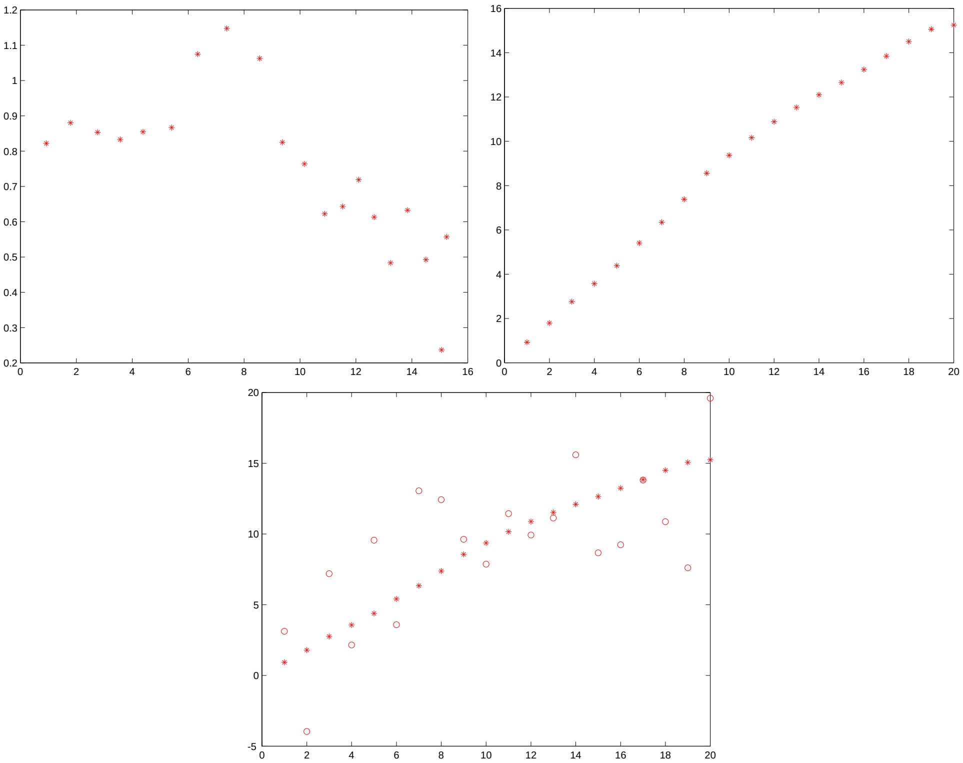

FIGURE 1.7: A constant velocity dynamic model for a point on the line. In this case, the state space is two dimensional — one coordinate for position, one for velocity. The figure on the **top left** shows a plot of the state; each asterisk is a different state. Notice that the vertical axis (velocity) shows some small change compared with the horizontal axis. This small change is generated only by the random component of the model, so the velocity is constant up to a random change. The figure on the **top right** shows the first component of state (which is position) plotted against the time axis. Notice we have something that is moving with roughly constant velocity. The figure on the **bottom** overlays the measurements (the circles) on this plot. We are assuming that the measurements are of position only, and are quite poor; as we see, this doesn't significantly affect our ability to track.

(i.e., that we see only the position of the point). Because we know that it's moving with constant acceleration — that's the model — we expect that we could use these measurements to estimate the whole state vector rather well.

Periodic Motion. Assume we have a point moving on a line with a periodic movement. Typically, its position p satisfies a differential equation like

$$\frac{d^2p}{dt^2} = -p.$$

This can be turned into a first order linear differential equation by writing the velocity as v and stacking position and velocity into a vector $\boldsymbol{u} = (p, v)$; we then

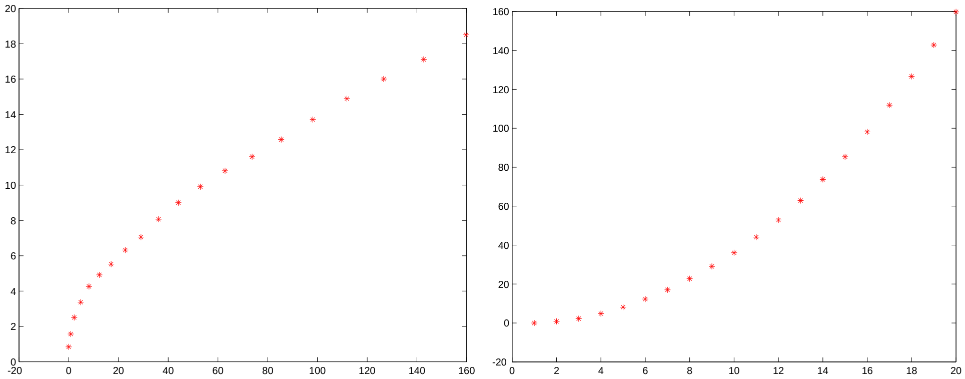

FIGURE 1.8: This figure illustrates a constant acceleration model for a point moving on the line. On the **left**, we show a plot of the first two components of state — the position on the x-axis and the velocity on the y-axis. In this case, we expect the plot to look like (t^2, t), which it does. On the **right**, we show a plot of the position against time — note that the point is moving away from its start position increasingly quickly.

have

$$\frac{d\boldsymbol{u}}{dt} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \boldsymbol{u} = \mathcal{S}\boldsymbol{u}.$$

Now assume we are integrating this equation with a forward Euler method, where the steplength is Δt; we have

$$\begin{aligned} \boldsymbol{u}_i &= \boldsymbol{u}_{i-1} + \Delta t \frac{d\boldsymbol{u}}{dt} \\ &= \boldsymbol{u}_{i-1} + \Delta t \mathcal{S} \boldsymbol{u}_{i-1} \\ &= \begin{pmatrix} 1 & \Delta t \\ -\Delta t & 1 \end{pmatrix} \boldsymbol{u}_{i-1}. \end{aligned}$$

We can either use this as a state equation, or we can use a different integrator. If we used a different integrator, we might have some expression in $\boldsymbol{u}_{i-1}, \ldots, \boldsymbol{u}_{i-n}$ — we would need to stack $\boldsymbol{u}_{i-1}, \ldots, \boldsymbol{u}_{i-n}$ into a state vector and arrange the matrix appropriately (see Exercises). This method works for points on the plane, in 3D, and so on, as well (again, see Exercises).

1.3.2 The Kalman Filter

An important feature of linear dynamic models is that all the conditional probability distributions we need to deal with are normal distributions. In particular, $P(\boldsymbol{X}_i|\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1})$ is normal, as is $P(\boldsymbol{X}_i|\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i)$. This means that they are relatively easy to represent — all we need to do is maintain representations of the mean and the covariance for the prediction and correction phase.

All this is much simplified in the case that the emission model is linear, the dynamic model is linear, and all noise is Gaussian. In this case, all densities are normal and the mean and covariance are sufficient to represent them. Both tracking

Algorithm 1.2: The Kalman filter updates estimates of the mean and covariance of the various distributions encountered while tracking a state variable of some fixed dimension using the given dynamic model.

Dynamic Model:

$$\begin{aligned} \boldsymbol{x}_i &\sim N(\mathcal{D}_i \boldsymbol{x}_{i-1}, \Sigma_{d_i}) \\ \boldsymbol{y}_i &\sim N(\mathcal{M}_i \boldsymbol{x}_i, \Sigma_{m_i}) \end{aligned}$$

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

$$\begin{aligned} \overline{\boldsymbol{x}}_i^- &= \mathcal{D}_i \overline{\boldsymbol{x}}_{i-1}^+ \\ \Sigma_i^- &= \Sigma_{d_i} + \mathcal{D}_i \sigma_{i-1}^+ \mathcal{D}_i \end{aligned}$$

Update Equations: Correction

$$\begin{aligned} \mathcal{K}_i &= \Sigma_i^- \mathcal{M}_i^T \left[\mathcal{M}_i \Sigma_i^- \mathcal{M}_i^T + \Sigma_{m_i} \right]^{-1} \\ \overline{\boldsymbol{x}}_i^+ &= \overline{\boldsymbol{x}}_i^- + \mathcal{K}_i \left[\boldsymbol{y}_i - \mathcal{M}_i \overline{\boldsymbol{x}}_i^- \right] \\ \Sigma_i^+ &= \left[Id - \mathcal{K}_i \mathcal{M}_i \right] \Sigma_i^- \end{aligned}$$

and filtering boil down to maintenance of these parameters. There is a simple set of update rules (given in algorithm 2; notation below), the **Kalman filter**.

Notation: We write $X \sim N(\mu; \Sigma)$ to mean that X is a normal random variable with mean μ and covariance Σ. Both dynamics and emission are linear, so we can write

$$X_k \sim N(\mathcal{A}_k X_{k-1}; \Sigma_k^{(d)})$$

and

$$Y_k \sim N(\mathcal{B}_k X_k; \Sigma_k^{(m)})$$

We will represent the mean of $P(X_i|y_0, \ldots, y_{i-1})$ as $\overline{X}_i^-$ and the mean of $P(X_i|y_0, \ldots, y_i)$ as $\overline{X}_i^+$ — the superscripts suggest that they represent our belief about X_i immediately before and immediately after the i'th measurement arrives. Similarly, we will represent the standard deviation of $P(X_i|y_0, \ldots, y_{i-1})$ as Σ_i^- and of $P(X_i|y_0, \ldots, y_i)$ as Σ_i^+. In each case, we will assume that we know $P(X_{i-1}|y_0, \ldots, y_{i-1})$, meaning that we know $\overline{X}_{i-1}^+$ and Σ_{i-1}^+.

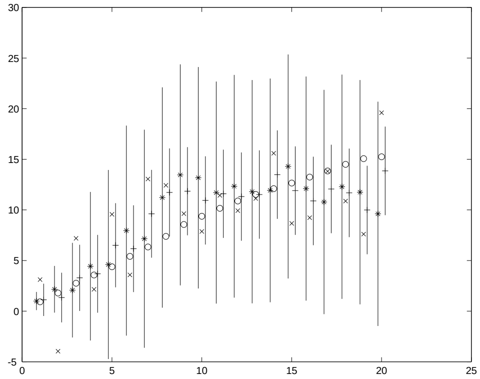

FIGURE 1.9: The Kalman filter for a point moving on the line under our model of constant velocity (compare with Figure 1.7). The state is plotted with open circles as a function of the step i. The *-s give $\overline{\boldsymbol{x}}_i^-$, which is plotted slightly to the left of the state to indicate that the estimate is made before the measurement. The x-s give the measurements, and the +-s give $\overline{\boldsymbol{x}}_i^+$, which is plotted slightly to the right of the state. The vertical bars around the *-s and the +-s are three standard deviation bars using the estimate of variance obtained before and after the measurement, respectively. When the measurement is noisy, the bars don't contract all that much when a measurement is obtained (compare with Figure 1.10).

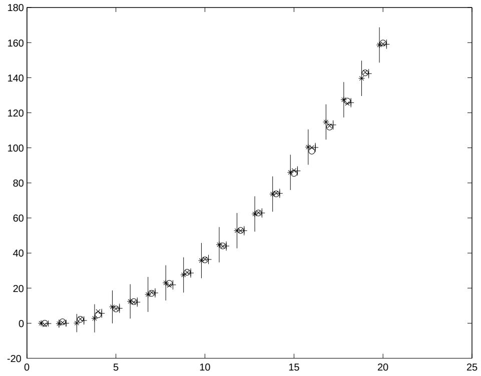

FIGURE 1.10: The Kalman filter for a point moving on the line under our model of constant acceleration (compare with Figure 1.8). The state is plotted with open circles as a function of the step i. The *-s give $\overline{\boldsymbol{x}}_i^-$, which is plotted slightly to the left of the state to indicate that the estimate is made before the measurement. The x-s give the measurements, and the +-s give $\overline{\boldsymbol{x}}_i^+$, which is plotted slightly to the right of the state. The vertical bars around the *-s and the +-s are three standard deviation bars using the estimate of variance obtained before and after the measurement, respectively. When the measurement is noisy, the bars don't contract all that much when a measurement is obtained.

1.3.3 Forward–Backward Smoothing

It is important to notice that $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ is not the best available representation of $\boldsymbol{X}_i$; this is because it doesn't take into account the future behavior of the point. In particular, all the measurements *after* $\boldsymbol{y}_i$ could affect our representation of $\boldsymbol{X}_i$. This is because these future measurements might contradict the estimates obtained to date — perhaps the future movements of the point are more in agreement with a slightly different estimate of the position of the point. However, $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ *is* the best estimate available at step i.

What we do with this observation depends on the circumstances. If our application requires an immediate estimate of position — perhaps we are tracking a car in the opposite lane — there isn't much we can do. If we are tracking off-line — perhaps for forensic purposes, we need the best estimate of what an object was doing given a videotape — then we can use all data points, and so we want to represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$. A common alternative is that we need a rough estimate immediately, and can use an improved estimate that has been time-delayed by a number of steps. This means we want to represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i+k})$ — we have to wait until time $i+k$ for this representation, but it should be an improvement on $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$.

We can incorporate future measurements with a clever trick. We must combine $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ — which we know how to obtain — with $P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_N)$. We actually know how to obtain a representation of $P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_N)$, too. We could simply run the Kalman filter *backward* in time, using *backward dynamics* and take the predicted representation of $\boldsymbol{X}_i$ (we leave the details of relabeling the sequence, etc., to the exercises).

Now we have two representations of $\boldsymbol{X}_i$: one obtained by running a forward filter and incorporating all measurements up to $\boldsymbol{y}_i$; and one obtained by running a backward filter and incorporating all measurements after $\boldsymbol{y}_i$. We need to combine these representations. We can get the answer by noting that *this is like having another measurement.* In particular, we have a new measurement generated by $\boldsymbol{X}_i$ — that is, the result of the backward filter — to combine with our estimate from the forward filter. We know how to combine estimates with measurements because that's what the Kalman filter equations are for.

All we need is a little notation. We attach the superscript f to the estimate from the forward filter and the superscript b to the estimate from the backward filter. We write the mean of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$ as $\overline{\boldsymbol{X}}_i^*$ and the covariance of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$ as Σ_i^*. We regard the representation of $\boldsymbol{X}_i^b$ as a measurement of $\boldsymbol{X}_i$ with mean $\overline{\boldsymbol{X}}_i^{b,-}$ and covariance $\Sigma_i^{b,-}$ — the minus sign is because the ith measurement cannot be used twice, meaning the backward filter predicts $\boldsymbol{X}_i$ using $\boldsymbol{y}_N \ldots \boldsymbol{y}_{i+1}$. This measurement needs to be combined with $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$, which has mean $\overline{\boldsymbol{X}}_i^{f,+}$ and covariance $\Sigma_i^{f,+}$ (when we substitute into the Kalman equations, these take the role of the representation *before* a measurement because the value of the measurement is now $\overline{\boldsymbol{X}}_i^{b,-}$).

Algorithm 1.3: The forward–backward algorithm combines forward and backward estimates of state to come up with an improved estimate.

Forward filter: Obtain the mean and variance of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \dots, \boldsymbol{y}_i)$ using the Kalman filter. These are $\overline{\boldsymbol{X}}_i^{f,+}$ and $\Sigma_i^{f,+}$.

Backward filter: Obtain the mean and variance of $P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1}, \dots, \boldsymbol{y}_N)$ using the Kalman filter running backward in time. These are $\overline{\boldsymbol{X}}_i^{b,-}$ and $\Sigma_i^{b,-}$.

Combining forward and backward estimates: Regard the backward estimate as a new measurement for $\boldsymbol{X}_i$, and insert into the Kalman filter equations to obtain

$$\Sigma_i^* = \left[(\Sigma_i^{f,+})^{-1} + (\Sigma_i^{b,-})^{-1}\right]^{-1};$$

$$\overline{\boldsymbol{X}}_i^* = \Sigma_i^* \left[(\Sigma_i^{f,+})^{-1}\overline{\boldsymbol{X}}_i^{f,+} + (\Sigma_i^{b,-})^{-1}\overline{\boldsymbol{X}}_i^{b,-}\right].$$

Substituting into the Kalman equations, we find that

$$\mathcal{K}_i^* = \Sigma_i^{f,+}\left[\Sigma_i^{f,+} + \Sigma_i^{b,-}\right]^{-1};$$

$$\Sigma_i^* = \left[I - \mathcal{K}_i\right]\Sigma_i^{+,f};$$

$$\overline{\boldsymbol{X}}_i^* = \overline{\boldsymbol{X}}_i^{f,+} + \mathcal{K}_i^*\left[\overline{\boldsymbol{X}}_i^{b,-} - \overline{\boldsymbol{X}}_i^{f,+}\right].$$

It turns out that a little manipulation (exercises!) yields a simpler form, which we give in Algorithm 3. Forward–backward estimates can make a substantial difference as Figure 1.11 illustrates.

Priors.

In typical vision applications, we are tracking forward in time. This leads to an inconvenient asymmetry: We may have a good idea of where the object started, but only a poor one of where it stopped (i.e., we are likely to have a fair prior for $P(\boldsymbol{x}_0)$, but may have difficulty supplying a prior for $P(\boldsymbol{x}_N)$ for the forward–backward filter). One option is to use $P(\boldsymbol{x}_N|\boldsymbol{y}_0, \dots, \boldsymbol{y}_N)$ as a prior. This is a dubious act as this probability distribution does not in fact reflect our prior belief about $P(\boldsymbol{x}_N)$ — we've used all the measurements to obtain it. The consequences can be that this distribution understates our uncertainty in $\boldsymbol{x}_N$ and so leads to a forward–backward estimate that significantly underestimates the covariance for

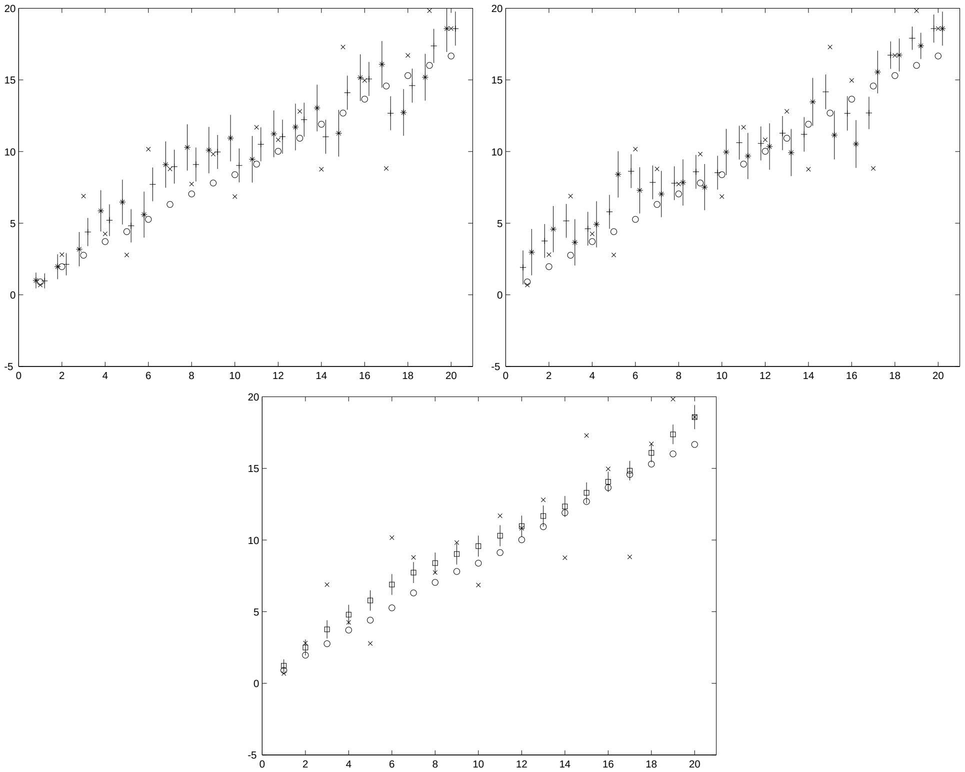

FIGURE 1.11: Forward–backward estimation for a dynamic model of a point moving on the line with constant velocity. We are plotting the position component of state against time. On the **top left**, we show the forward estimates, again using the convention that the state is shown with circles, the data is shown with an x, the prediction is shown with a *, and the corrected estimate is shown with a +; the bars give one standard deviation in the estimate. The predicted estimate is shown slightly behind the state and the corrected estimate is shown slightly ahead of the state. You should notice that the measurements are noisy. On the **top right** we show the backward estimates. Now time is running backward (although we have plotted both curves on the same axis) so that the prediction is slightly ahead of the measurement and the corrected estimate is slightly behind. We have used the final corrected estimate of the forward filter as a prior. Again, the bars give one standard deviation in each variable. On the **bottom**, we show the combined forward–backward estimate. The squares give the estimates of state. Notice the significant improvement in the estimate.

the later states. An alternative is to use the mean supplied by the forward filter, but enlarge the covariance substantially; the consequences are a forward–backward estimate that overestimates the covariance for the later states.

Not all applications have this asymmetry. For example, if we are engaged in a forensic study of a videotape, we might be able to start both the forward tracker and the backward tracker by hand and provide a good estimate of the prior in each case. If this is possible, then we have a good deal more information which may be able to help choose correspondences, and so on — the forward tracker should finish rather close to where the backward tracker starts.

Smoothing Over an Interval. Although our formulation of forward–backward smoothing assumed that the backward filter started at the last data point, it is easy to start this filter a fixed number of steps ahead of the forward filter. If we do this, we obtain an estimate of state in real time (essentially immediately after the measurement) and an improved estimate some fixed numbers of measurements later. This is sometimes useful. Furthermore, it is an efficient way to obtain most of the improvement available from a backward filter if we can assume that the effect of the distant future on our estimate is relatively small compared with the effect of the immediate future. Notice that we need to be careful about priors for the backward filter here; we might take the forward estimate and enlarge its covariance somewhat.

1.4 DATA ASSOCIATION

Not all observations are informative. For example, if one wishes to track an aircraft — where state might involve pose, velocity and acceleration variables, and measurements might be radar returns giving distance and angle to the aircraft from several radar aerials — some of the radar returns measured might not come from the aircraft. Instead, they might be the result of noise, of other aircraft, of strips of foil dropped to confuse radar apparatus (**chaff** or **window**; see [12]), or of other sources. The problem of determining which observations are informative and which are not is known as **data association**. Data association is the dominant difficulty in tracking objects in video. This is because so few of the very many pixels in each frame lie on objects of interest. It can be spectacularly difficult to tell which pixels in an image come from an object of interest and which do not. The tracking methods of the first two sections focus on solving data association problems using familiar ideas of detection (section **??**) and of coherent appearance (section **??**). These methods can be linked to probabilistic representations to exploit any dynamical information that happens to be available.

1.4.1 Linking Kalman Filters with Detection Methods

Notes: *(1)Examples: - Kalman filter and Lucas Kanade; - Kalman filter and mean shift; (2) More general points about data association (3)?*

In section 1.1.2, we took a patch in image n and looked for one in image $n+1$ that was "similar". To find a match, we assumed that the patch translated a relatively small distance from a known start point, and searched around that start point using an approximate method to find the best SSD match. In that section, we assumed that the start point was the patch's location in image n. It could, instead, be predicted by a Kalman filter. Similarly, in section 1.2.1, we had a domain that was a circle of a fixed size in image n, and we searched around a start point for a matching domain in image $n+1$ using an approximate method to find the best matching histogram. Again, we assumed the start point was the circle's location in image n, but it could be predicted by a Kalman filter. We will do the case of the circle in somewhat greater detail to illustrate how this works.

The circular domains was represented with a vector $\boldsymbol{y}$ giving the center of the circle. Our search needed a start point, which we wrote as $\boldsymbol{y}_0$. This start point could come from a dynamical model. In fact, assuming as we did in section 1.2.1 that it was the configuration of domain in image n is a simple dynamical model where the object doesn't move. An alternative model might be better. For example, we might be observing an object in free fall in a plane parallel to the camera plane. In this case, the object might move a considerable distance between frames, but its location in image $n+1$ is quite easily predicted from earlier locations. To predict $\boldsymbol{y}_0$ with a Kalman filter, we choose a state vector representation so that the configuration $\boldsymbol{y}$ is a linear function of the state vector. We apply the prediction step of the Kalman filter to obtain the predicted state vector, and compute $\boldsymbol{y}_o$ from that. We then search for $\boldsymbol{y}^{(n+1)}$ as in section **??**. The computed value is the observation for the Kalman filter, and we use this to obtain the revised estimate of the state vector. This gives the procedure of algorithm **??**.

As an example, the state vector is $\boldsymbol{x} = (\boldsymbol{y}, \dot{\boldsymbol{y}})^T$. The velocity drifts slightly, so the dynamical model is given by

$$\boldsymbol{x}_{n+1} = \mathcal{A}\boldsymbol{x}_n + \xi = \begin{pmatrix} \mathcal{I} & \Delta t\mathcal{I} \\ 0 & \mathcal{I} \end{pmatrix} \boldsymbol{x}_n + \xi_{n+1}$$

where $\xi_{n+1} \sim N(\mathbf{0}, \Sigma_d)$. The measurement model is given by

$$\boldsymbol{y}_n = \mathcal{B}\boldsymbol{x}_n = \begin{pmatrix} \mathcal{I} & 0 \end{pmatrix} \boldsymbol{x}_n + \eta_n$$

where $\eta_n \sim N(\mathbf{0}, \Sigma_m)$. Now assume we have $\boldsymbol{x}_n^{(+)}$. We predict $\boldsymbol{x}_{n+1}^{(-)} = \mathcal{A}\boldsymbol{x}_n^{(+)}$, which gives a prediction $\boldsymbol{y}_0 = \mathcal{B}\boldsymbol{x}_{n+1}^{(-)}$. We start the search at this point, and obtain $\boldsymbol{y}^{(n+1)}$. This goes into the Kalman gain formula (algorithm **??**) to give $\boldsymbol{x}_{n+1}^{(+)}$. The advantage of doing this is that, if the object has a significant but fixed velocity, our search starts at the location predicted by this velocity model, and so is much more likely to find the match we want.

1.4.2 Key Methods of Data Association

The case of the Kalman filter applied to tracking by detection is an instance of a general strategy. In particular, we have an estimate of $P(X_n|Y_0, ..., Y_{n-1})$ and we know $P(Y_n|X_n)$. From this we can obtain an estimate of $P(Y_n|Y_0, ...Y_{n-1})$, which gives us hints as to where the measurement might be. These hints can be applied in a variety of ways.

One can use a **gate** — we look only at measurements that lie in a domain where $P(Y_n|Y_0, ..., Y_{n-1})$ is big enough. This is a method with roots in radar tracking of missiles and aeroplanes, where one must deal with only a small number (compared with the number of pixels in an image!) of returns, but the idea has been useful in visual tracking applications. For example, if we are tracking using an object detector, we would apply it only within the gate (or ignore detector responses outside the gate). This approach is quite commonly adopted within vision, and is useful.

One can use **nearest neighbours**. In the classical version, we have a small set of possible measurements, and we choose the measurement with the largest value of $P(Y_n|Y_0, ..., Y_{n-1})$. This has all the dangers of wishful thinking — we are deciding that a measurement is valid because it is consistent with our track — but is often useful in practice. Our example of using a Kalman filter to identify a start point for a search is a nearest neighbours strategy. Again, this approach is commonly adopted and is useful.

One can use **probabilistic data association**, where we construct a virtual observation from a weighted combination of measurements within a gate, weighted using (a) the predicted measurement and (b) the probability a detector has failed to detect. For example, if we are using tracking by detection, we could form a virtual observation as a weighted sum of detects that appear within a gate. This approach tends not to be used, however, perhaps because it is uncommon in practice to have a detector that is reliable enough to use like this but not good enough to support nearest neighbours.

1.5 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.

1.5.1 Multiple Modes

The Kalman filter is the workhorse of estimation, and can give useful results under many conditions. One doesn't need a guarantee of linearity to use a Kalman filter — if the logic of the application indicates that a linear model is reasonable, there is a good chance a Kalman filter will work. Rohr used a Kalman filter to track a walking person successfully, evaluating the measurement by matches to line segments on the outline [17, 16].

More recently, the method tends not to be used because of concerns about multiple modes. The representation adopted by a Kalman filter (the mean and co-variance, sufficient statistics for a Gaussian distribution) tends to represent multi-modal distributions poorly. There are several reasons one might encounter multiple modes.

First, nonlinear dynamics — or nonlinear measurement processes, or both — can create serious problems. The basic difficulty is that even quite innocuous looking setups can produce densities that are not normal, and are very difficult to represent and model. For example, let us look at only the hidden state. Assume that this is one dimensional. Now assume that state updates are *deterministic*, with $X_{i+1} = X_i + \epsilon \sin(X_i)$. If ϵ is sufficiently small, we have that for $0 < X_i < \pi$, $X_i < X_{i+1} < \pi$; for $-\pi < X_i < 0$, $-\pi < X_{i+1} < X_i$; and so on. Now assume that $P(X_0)$ is normal. For sufficiently large k, $P(X_k)$ will not be; there will be "clumps" of probability centered around the points $(2j + 1)\pi$ for j an integer. These clumps will be very difficult to represent, particularly if $P(X_0)$ has very large variance so that many clumps are important. Notice that what is creating a problem here is that quite small non-linearities in dynamics can cause probability to be concentrated in ways that are very difficult to represent. In particular, nonlinear dynamics are likely to produce densities with complicated sufficient statistics. There are cases where nonlinear dynamics does lead to densities that can be guaranteed to have finite-dimensional sufficient statistics (see [2, 7, 6]); to our knowledge, these have not been applied to human tracking.

There are practical phenomena in human tracking that tend to suggest that non-normal distributions are a significant part of the problem. Assume we wish to track a 3D model of an arm in a single image. The elbow is bent; as it straightens, it will eventually run into an **end-stop** — the forearm can't rotate further without damage. At the end-stop, the posterior on state can't be a normal distribution, because a normal distribution would have some support on the wrong side of the end-stop, and this has a significant effect on the shape of the posterior (see figure 1.12). Another case that is likely, but not guaranteed, to cause trouble is a **kinematic singularity**. For example, if the elbow is bent, we will be able to observe rotation about the humerus, but current observation models will make this unobservable if the elbow is straight (because the outline of the arm will not change; no current method can use the changes in appearance of the hand that will result). The dimension of the state space has collapsed. The posterior might be a normal distribution in this reduced dimension space, but that would require explicitly representing the collapse. The alternative, a covariance matrix of reduced rank, creates unattractive problems of representation. Deutscher *et al.* produce evidence that, in both cases, posteriors are not, in fact, normal distributions, and show that an extended Kalman filter can lose track in these cases [8].

Kinematic ambiguity in the relations between 3D and 2D are a major source of multiple modes. Assume we are tracking a human figure using a 3D representation of the body in a single view. If, for example, many 3D configurations correspond exactly to a single 2D configuration, then we expect the posterior to have multiple modes. Chapter **??** discusses this issue in extensive detail.

The richest source of multiple modes is data association problems. An easy example illustrates how nasty this problem can be. Assume we have a problem with linear dynamics and a linear measurement model. However, at each tick of the clock we receive more than one measurement, exactly one of which comes from the process being studied. We will continue to write the states as $\boldsymbol{X}_i$, the measurements as $\boldsymbol{Y}_i$; but we now have δ_i, an indicator variable that tells which measurement comes from the process (and is unknown). $P(\boldsymbol{X}_N|\boldsymbol{Y}_{1..N}, \delta_{1..N})$ is clearly Gaussian. We want $P(\boldsymbol{X}_N|\boldsymbol{Y}_{1..N}) = \sum_{histories} P(\boldsymbol{X}_N|\boldsymbol{Y}_{1..N}, \delta_{1..N})P(\delta_{1..N}|\boldsymbol{Y}_{1..N})$, which is clearly a mixture of Gaussians. The number of components is exponential in the number of frames — there is one component per history — meaning that $P(\boldsymbol{X}_N|\boldsymbol{Y}_{1..N})$ could have a very large number of modes.

1.5.2 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

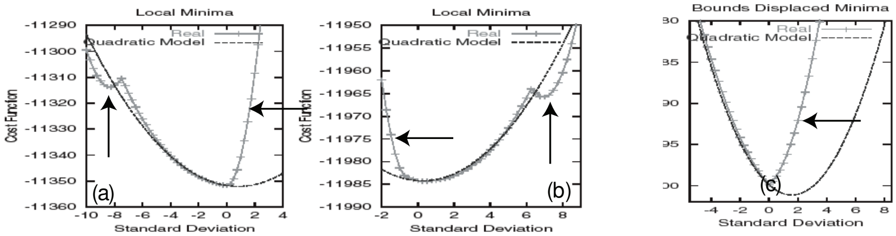

FIGURE 1.12: **Attribution:** *Figure 4, 2a of Sminchisescu and Triggs, Covariance scaled..., particlefilter/009090509 Several nasty phenomena result from kinematic ambiguities and from kinematic limits, as Sminchisescu and Triggs [19, 20]. Ambiguities in reconstruction — which are caused because body segments can be oriented in 3D either toward or away from the camera, as described in the section* **??** *— result in multiple modes in the posterior. The two graphs on the* **left** *(a and b) show the fitting cost (which can be thought of as a log-likelihood) as a function of the value of two state variables (scaled by their standard deviation). The state variables refer to the kinematic configuration of the 3D model. Note the significant "bumps" from the second mode (the* **vertical arrows***). For reference, there is a quadratic approximation shown as well. Note also the the significant deformations of modes resulting from a kinematic limit (the* **horizontal arrows***). This is caused by the fact that no probability can lie on the other side of the limit, so the mode must be "squashed".* Figure from "Covariance Scaled Sampling for Monocular 3D Body Tracking", Sminchisescu and Triggs, Proc. Computer Vision and Pattern Recognition, 2001 © 2001 IEEE.

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 numerical 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

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

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 4). 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 5). There are two other important activities for tracking: marginalisation, and turning a representation of a prior into a representation of a posterior.

Marginalizing 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.

Algorithm 1.4: 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 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 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 $\boldsymbol{V}$.

Algorithm 1.5: 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:

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

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

Algorithm 1.6: 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,

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

1.5.3 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 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

$$\left\{(\boldsymbol{u}^k_{i-1}, w^k_{i-1})\right\}$$

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

Now for any given sample $\boldsymbol{u}^k_{i-1}$, we can obtain samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}^k_{i-1})$ 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 $\boldsymbol{u}^k_{i-1}$, we can generate samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}^k_{i-1})$ as

$$\left\{(f(\boldsymbol{u}^k_{i-1}) + \xi^l_i, 1)\right\}$$

Algorithm 1.7: 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

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

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 7. 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 7). 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 1.13 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 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 1.13 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.

1.5.4 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

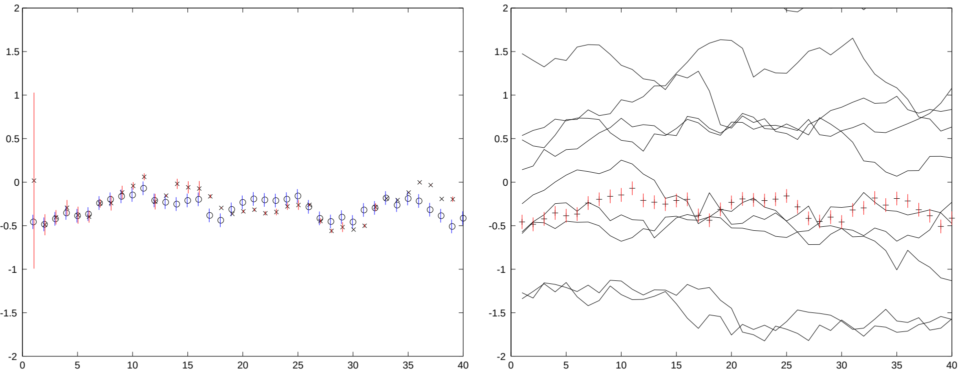

FIGURE 1.13: The simple particle filter behaves very poorly, as a result of a phenomenon called **sample impoverishment**, which is rather like quantisation error. In this example, we have a point on the line drifting on the line (i.e., $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted 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.

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.

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 1.14 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.

1.5.5 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 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

Algorithm 1.8: 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 $1/N$.

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

Algorithm 1.9: 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 8.

this problem — all involve refining some form of search over the likelihood — there is no standard solution yet.

1.6 NOTES

Main points: look at various books. Much of the flow/detection stuff can be complicated by probability, but there isn't much point. Data association is the hard bit.

In this chapter, there is a brief discussion of the particle filter, a current favorite method for dealing with multi-modal densities. There are other methods: Beneš describes a class of nonlinear dynamical model for which the posterior can be represented with a sufficient statistic of constant finite dimension [2]. Daum extends the class of models for which this is the case ([6, 7]; see also [18] for an application and [10] for a comparison with the particle filter). Extensive accounts

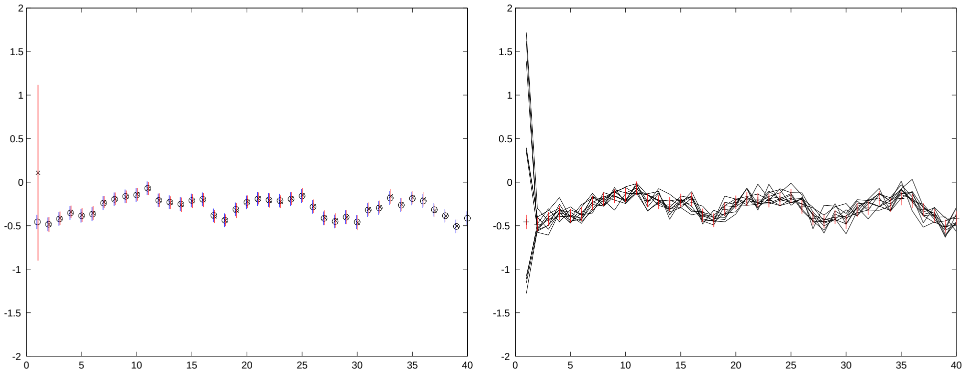

FIGURE 1.14: Resampling hugely improves the behavior of a particle filter. We now show a resampled particle filter tracking a point drifting on the line (i.e., $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise, and are the same as for Figure 1.13. In the figure 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.

of particle filters appear in [9, 14, 15].

Index

Bibliography

[1] Yaakov Bar-Shalom and Xiao-Rong Li. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[2] V.E. Beneš. Exact finite-dimensional filters with certain diffusion non linear drift. *Stochastics*, 5:65–92, 1981.

[3] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[4] Louis Brown. *A Radar History of World War II: Technical and Military Imperatives*. Institute of Physics Press, 2000.

[5] Robert Buderi. *The Invention that Changed the World*. Touchstone Press, 1998. reprint.

[6] F.E. Daum. Beyond kalman filters: practical design of nonlinear filters. In *Proc. SPIE*, volume 2561, pages 252–262, 1995.

[7] F.E. Daum. Exact finite dimensional nonlinear filters. *IEEE. Trans. Automatic Control*, 31:616–622, 1995.

[8] J. Deutscher, B. North, B. Bascle, and A. Blake. Tracking through singularities and discontinuities by random sampling. In *Int. Conf. on Computer Vision*, pages 1144–1149, 1999.

[9] A. Doucet, N. De Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[10] A. Farina, D. Benvenuti, and B. Ristic. A comparative study of the benes filtering problem. *Signal Processing*, 82:133–147, 2002.

[11] J. Gibson. *The perception of the visual world*. Houghton-Mifflin, 1950.

[12] R. V. Jones. *Most Secret War*. Wordsworth Military Library, 1998. reprint.

[13] Shanon X. Ju, Michael J. Black, and Yaser Yacoob. Cardboard people: A parameterized model of articulated image motion. In *Proc. Int. Conference on Face and Gesture*, pages 561–567, 1996.

[14] J.S. Liu and R. Chen. Sequential monte-carlo methods for dynamic systems. Technical report, Stanford University, 1999. preprint.

[15] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2004.

[16] K. Rohr. Incremental recognition of pedestrians from image sequences. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 9–13, 1993.

[17] K. Rohr. Towards model-based recognition of human movements in image sequences. *CVGIP: Image Understanding*, 59(1):94–115, 1994.

[18] G.C. Schmidt. Designing nonlinear filters based on Daum's theory. *Journal of Guidance, Control and Dynamics*, 16:371–376, 1993.

[19] C. Sminchisescu and B. Triggs. Covariance scaled sampling for monocular 3d body tracking. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages I:447–454, 2001.

[20] C. Sminchisescu and B. Triggs. Estimating articulated human motion with covariance scaled sampling. *The International Journal of Robotics Research*, 22(6):371–391, 2003.

[21] A. Gelb with Staff of the Analytical Sciences Corporation. *Applied Optimal Estimation*. MIT Press, 1974.