CHAPTER 11

Tracking

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 Brown (2000); Buderi (1998); and Jones (1998); comprehensive reviews of technique in this context include Bar-Shalom and Li (2001); Blackman and Popoli (1999); and Gelb and of the Analytical Sciences Corporation (1974)). 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 might 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 the 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 when 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 might 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 that 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 11.1.1) or exploiting the tendency for objects to look the same over time, and to move coherently (Section 11.1.2 and Section 11.2). It is straightforward to balance dynamical predictions against measurements using probabilistic methods if the dynamical model is relatively straightforward, because the probability models are easy to represent (Section 11.3). Furthermore, dynamical predictions can be used to identify useful measurements (Section 11.4). Non-linear dynamical models can produce probability models that need to be represented with approximate methods (Section 11.5).

11.1 SIMPLE TRACKING STRATEGIES

There are two simple ways to track objects. In the first, *tracking by detection*, we have a strong model of the object, strong enough to identify it in each frame. We find it, link up the instances, and we have a track. Some additional machinery can compensate for weaker models in many cases, too (Section 11.1.1). In the second, *tracking by matching*, we have a model of how the object moves. We have a domain in the nth frame in which the object sits, and then use this model to search for a domain in the $n+1$th frame that matches it (Section 11.1.2). Tracking by matching strategies become more elaborate as the motion model and the matching model becomes more elaborate; we deal with the more elaborate strategies in Section 11.2.

11.1.1 Tracking by Detection

Assume that we will see only 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. In this case, tracking is straightforward: we report the location of the detector response in each frame of the video. This observation

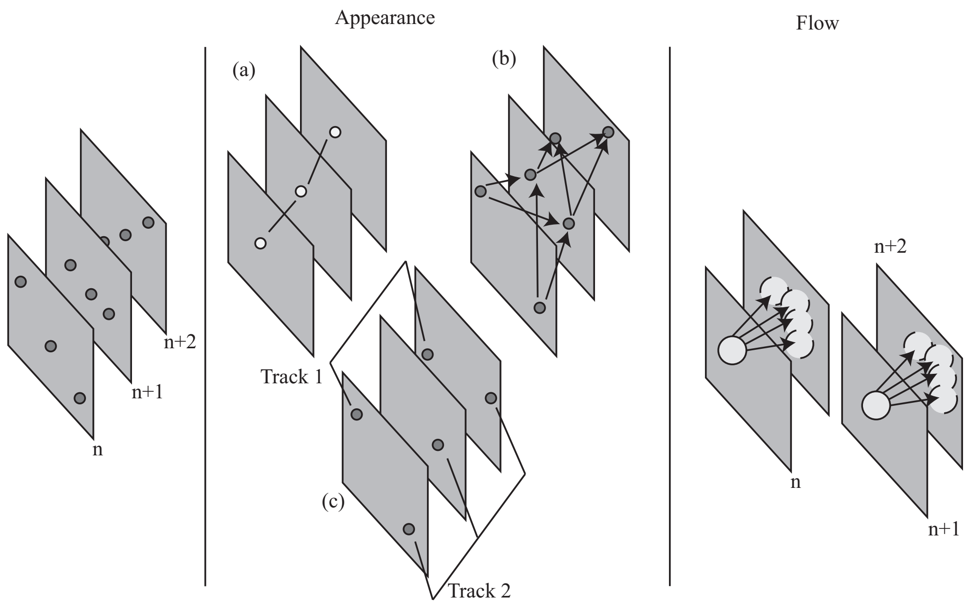

FIGURE 11.1: In tracking problems, we want to build space time paths followed by tokens—which might be objects, or regions, or interest points, or image windows—in an image sequence (**left**). There are two important sources of information; carefully used, they can resolve many tracking problems without further complexity. One is the appearance of the token being tracked. If there is only one token in each frame with a distinctive appearance, then we could detect it in each frame, then link the detector responses (**a**). Alternatively, if there is more than one instance per frame, a cost function together with weighted bipartite matching could be enough to build the track (**b**). If some instances drop out, we will need to link detector responses to abstract tracks (**c**); in the figure, track 1 has measurements for frames n and $n+2$, but does not have a measurement for frame $n+1$. Another important source of information is the motion of the token; if we have a manageable model of the flow, we could search for the flow that generates the best match in the next frame. We choose that match as the next location of the token, then iterate this procedure (**right**).

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 (detectors are discussed in detail in Chapter 17).

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 might have appeared). For every track that has not received a response for several frames, we prune that track (because the object might have disappeared). Finally, we may postprocess the set of tracks to insert links where justified by the application. Algorithm 11.1 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 and 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 11.1). We could solve this exactly with the Hungarian algorithm (see, for example, Cormen *et al.* (2009); Schrijver (2003); or Jungnickel (1999)); very often, however, the quite good approximation that a greedy algorithm will supply is sufficient. In some cases, we know where objects can appear and disappear, so that tracks can be created only 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, such as 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. 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 when a person appears in a particular view. The method is well suited to such cases.

This basic recipe for tracking by detection is worth remembering. In many

Notation:
Write $\mathbf{x}_k(i)$ for the k'th response of the detector in the ith frame
Write $t(k, i)$ for the k'th track in the ith frame
Write $*t(k, i)$ for the detector response attached to the k'th track in the ith 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 anywhere this is justified (perhaps by a more sophisticated dynamical or appearance model, derived from the candidates for linking).

Algorithm 11.1: Tracking by Detection.

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 quite general and extremely effective.

11.1.2 Tracking Translations by Matching

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 11.2). 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 are looking for a box that looks a lot like the current box, so we can use the *sum-of-squared differences* (or *SSD*) of pixel values as a test for similarity. If we write $\mathcal{R}^{(n)}$ for the rectangle in the nth frame, $\mathcal{R}^{(n)}_{ij}$ for the i, jth pixel in the

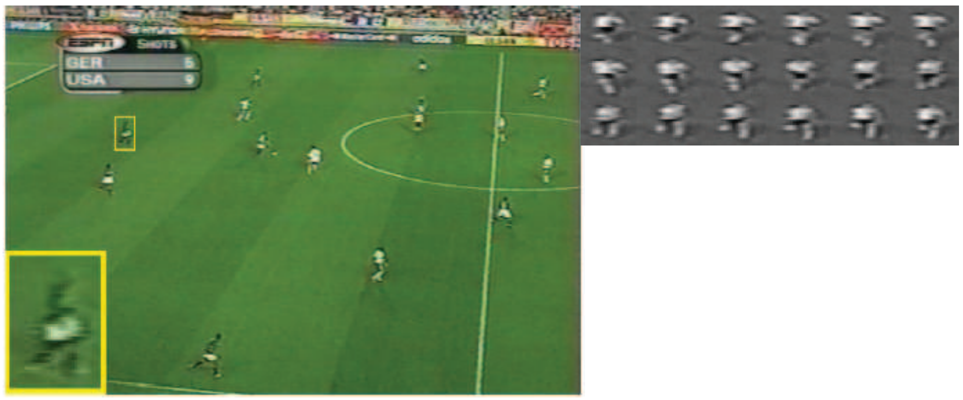

FIGURE 11.2: A useful application of tracking is to stabilize an image box around a more interesting structure, in this case a football player in a television-resolution video. A frame from the video is shown on the **left**. Inset is a box around a player, zoomed to a higher resolution. Notice that the limbs of the player span a few pixels, are blurry, and are hard to resolve. A natural feature for inferring 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 configuration changes 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. *This figure was originally published as Figure 7 of "Recognizing Action at a Distance," A. Efros, A.C. Berg, G. Mori, and J. Malik, Proc. IEEE ICCV, 2003, © IEEE, 2003.*

rectangle in the nth image, then we choose $\mathcal{R}^{(n+1)}$ to minimize

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

In many 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 4.7 for more information).

Now write $\mathcal{P}_t$ for the indices of the patch in the tth frame and $I(\boldsymbol{x}, t)$ for the tth frame. Assume that the patch is at $\boldsymbol{x}_t$ in the tth 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

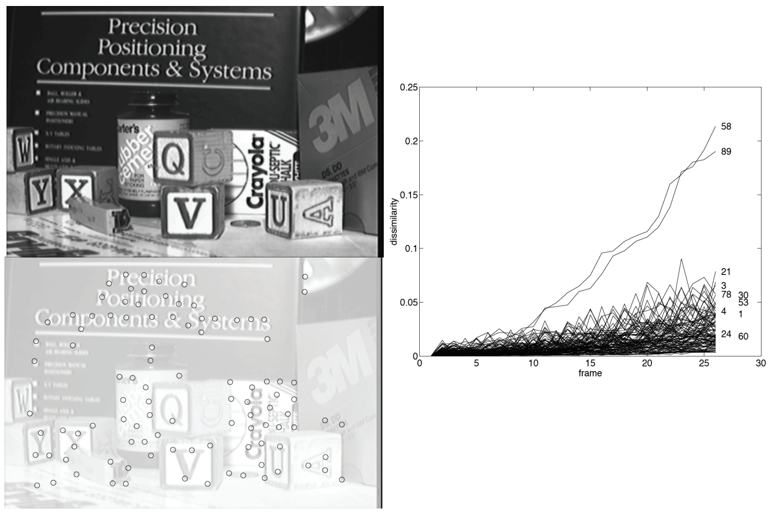

FIGURE 11.3: It is natural to track local neighborhoods, like those built in Section 5.3.2; however, for these neighborhoods to yield good tracks, they should pass a test of appearance complexity, shown in the text. This test checks that estimates of the translation of the neighborhood are stable. **Top left:** the first frame of an image sequence, with possible neighborhoods that pass this test shown on the **bottom left**. On the **right**, the sum-of-squared differences between the translated patch in frame n and the original in frame 1. Notice how this drifts up, meaning that the accumulated motion over many frames is *not* a translation; we need a better test to identify good tracks. *This figure was originally published as Figures 10, 11, 12 of "Good features to track," by J. Shi and C. Tomasi, Proc. IEEE CVPR 1994, © IEEE, 1994.*

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. As Shi and Tomasi (1994) point out, this means that we can test the smallest eigenvalue of this matrix to tell whether a local window is worth tracking.

11.1.3 Using Affine Transformations to Confirm a Match

Some patches are like the soccer player example in Figure 11.2: the patch just translates. For other patches, the movement from frame n to $n+1$ is quite like a translation, but when one compares frame 1 to frame $n+1$, a more complex model of deformation is required. This could occur because, for example, the surface

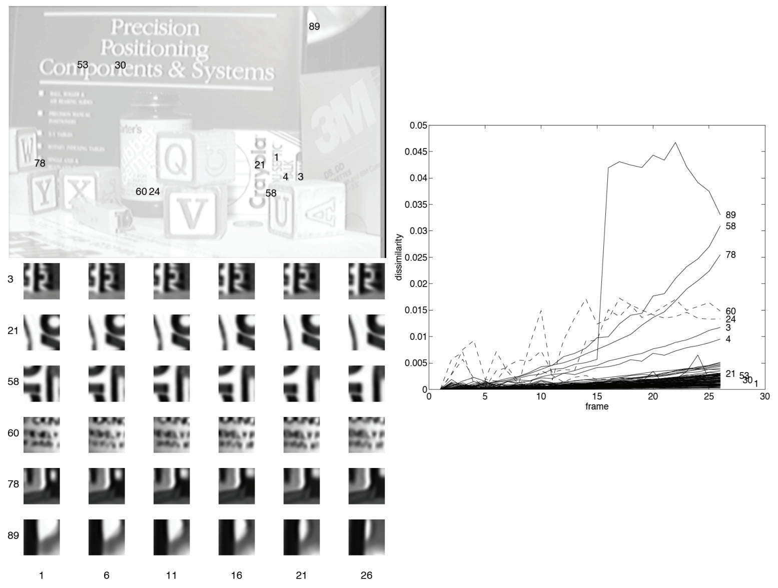

FIGURE 11.4: On the **top left**, the first frame of the sequence shown in Figure 11.3, with some neighborhoods overlaid. On the **bottom left**, the neighborhoods associated with these features (vertical) for different frames (horizontal). Notice how the pattern in the neighborhood deforms, perhaps because the object is rotating in 3D. This means that a translation model is good for the movement from frame n to frame $n+1$, but does not explain the movement from frame 1 to frame $n+1$. For this, we need to use an affine model. On the **right**, the value of the sum-of-squared differences between neighborhoods on a track in frame n and in frame 1, plotted against n. In this case, the neighborhood has been rectified by an affine transform, as in Section 11.1.3, before computing the SSD. Notice how some tracks are obviously good and others can be seen to have drifted. We could use this property to prune tracks. *This figure was originally published as Figures 13, 14, 15 of "Good features to track," by J. Shi and C. Tomasi, Proc. IEEE CVPR 1994, © IEEE, 1994.*

on which the patch lies is rotating in 3D. In cases such as this, we should use a translation model to build a track, and then prune tracks by checking the patch in frame $n+1$ against frame 1. Because the image patch is small, an affine model is appropriate. The affine model means that the point $\boldsymbol{x}$ in frame 1 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 $\boldsymbol{u}\in\mathcal{P}_1$. 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.

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

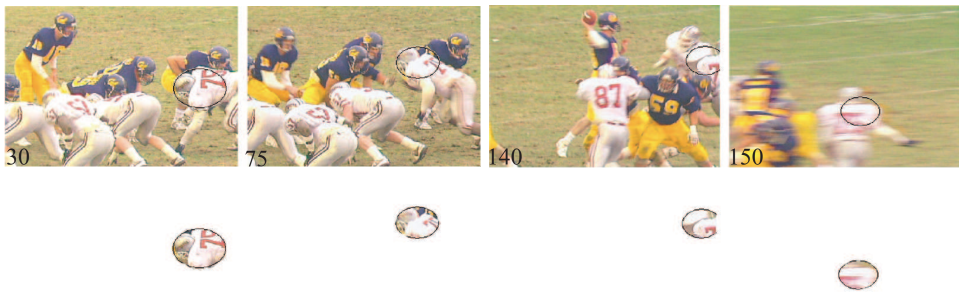

FIGURE 11.5: Four frames from a sequence depicting football players, with superimposed domains. The object to be tracked is the blob on top of player 78 (at the center right in frame 30). We have masked off these blobs (**below**) to emphasize just how strongly the pixels move around in the domain. Notice 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. *This figure was originally published as Figure 1 of "Kernel-Based Object Tracking" by D. Comaniciu, V. Ramesh, and P. Meer, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003, © IEEE 2003.*

create and reap tracks as necessary, we find all interest points in frame 1. 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 11.4.1, we show how to link this procedure with a dynamical model built from a Kalman filter (Kalman filters are described in Section 11.3).

11.2 TRACKING USING MATCHING

Imagine tracking a face in a webcam. 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 blobby, tends to have coherent appearance, and tends only to translate and rotate. As with the strategy of Section 11.1.2, we have a domain of interest in the nth image, $\mathcal{D}_n$, and we must search for a matching domain $\mathcal{D}_{n+1}$ in the $n+1$st image, but 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 11.2.1). 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 10.6.2).

```
Assume we have a sequence of $N$ images; a domain $\mathcal{D}_1$,
in the first image represented by parameters
$\boldsymbol{y}_1$ (for a circular domain of fixed size, these would be the
location of the center; for a square, the center and edge length; and so on);
a kernel function $k$; a scale $h$; and a feature representation $\boldsymbol{f}$ of each pixel.

For $n \in [1, \ldots, N-1]$
    Obtain an initial estimate $\boldsymbol{y}_{n+1}^{(0)}$ of the next domain
        either from a Kalman filter, or using $\boldsymbol{y}_n$
    Iterate until convergence

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

        where $p_u$, $k$, $g$ are as given in the text

The track is the sequence of converged estimates $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$.
```

Algorithm 11.2: Tracking with the Mean Shift Algorithm.

11.2.1 Matching Summary Representations

Look at the football player's uniform in Figure 9.3.4. From frame to frame, we see the player's back at different viewing angles. Individual pixels in one domain might 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 might 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 $\mathcal{D}_n$ might 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 might 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 when 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 $\mathcal{D}_n$.

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 $\boldsymbol{p}(\boldsymbol{y})$; its uth 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*:

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

This will be one if the two distributions 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 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

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

This means that, to minimize the distance, we must maximize

$$\frac{1}{2} \sum_u p_u(\boldsymbol{y}) \sqrt{\frac{p_u(\boldsymbol{y}_n)}{p_u(\boldsymbol{y}_{n+1}^{(0)})}}. \tag{11.1}$$

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 $\| \boldsymbol{x}_i - \boldsymbol{y} \|$ according to a kernel profile k (compare Section 9.3.4). Using this approach, the fraction of total votes in bin u produced by all features is

$$p_u(\boldsymbol{y}) = C_h \sum_{i \in \mathcal{D}_n} k(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2) \delta \left[b(\boldsymbol{f}_i - u) \right]. \tag{11.2}$$

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. Substituting Equation 11.2

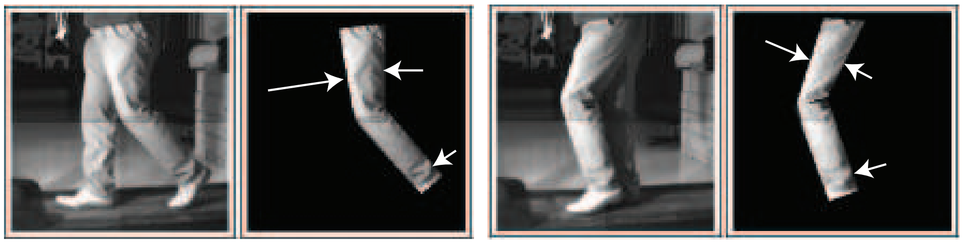

FIGURE 11.6: An important pragmatic difficulty with flow-based tracking is that appearance is not always fixed. The folds in loose clothing depend on body configuration, as these images of trousers indicate. The trousers were tracked using a flow-based tracker, but enforcing equality between pixel values 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. *This figure was originally published as Figure 4 of "Cardboard People: A Parameterized Model of Articulated Image Motion," by S. Ju, M. Black, and Y. Yacoob, IEEE Int. Conf. Face and Gesture, 1996 © IEEE, 1996.*

into Equation 11.1, we must maximize

$$f(\boldsymbol{y}) = \frac{C_h}{2} \sum_i w_i k(\| \frac{\boldsymbol{x}_i - \boldsymbol{y}}{h} \|^2), \tag{11.3}$$

where

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

We can use the mean shift procedure of Section 9.3.4 to maximize equation 11.3. Following the derivation there, the mean shift procedure involves producing a series of estimates $\boldsymbol{y}^{(j)}$ where

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

The procedure gets its name from the fact that we are shifting to a point that has the form of a weighted mean. The complete algorithm appears in Algorithm 11.2.

11.2.2 Tracking Using Flow

We can generalize the methods of Section 11.1.2 in a straightforward way. There we found the best matching translated version of an image domain. Instead, we could have a family of flow models, as in Section 10.6.1, and find the best matching domain resulting from a flow model. 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 nth 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 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 10.4.1).

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.

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. Section 11.1.3 showed how to prune tracks by testing against a model of appearance. If we have few tracks, we cannot just prune, but must correct the drift. This requires a fixed, global model of appearance, like those of Section 20.3.

Another important pragmatic difficulty is that an object's appearance is often not 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 11.6). Although this signal almost certainly contains some cues to configuration, they appear to be very difficult to exploit.

11.3 TRACKING LINEAR DYNAMICAL MODELS WITH KALMAN FILTERS

In Section 11.1.1, 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 letter indicates 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 $P(X_k|Y_0, \ldots, Y_k)$. In **smoothing** (sometimes called filtering), we wish to determine some representation of $P(X_k|Y_0, \ldots, Y_N)$ (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 on the hidden state, that is, that $P(Y_k|X_0, \ldots, X_N, Y_0, \ldots, Y_N) = P(Y_k|X_k)$.
- We assume that the probability density for a new state is a function only of the previous state; that is, $P(X_k|X_0, \ldots, X_{k-1}) = P(X_k|X_{k-1})$ 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