Tracking

D.A. Forsyth, UIUC

Tracking: Why?

- Motion capture
  - build models of moving people from video
- Recognition from motion
  - eg cyclists move differently than runners
- Surveillance
  - who is doing what?
    - for security (eg keep people out of sensitive areas in airports)
    - for HCI (eg kinect, eyetoy, etc.)

Tracking

- Establish state of object using time sequence
  - state could be:
    - position; position+velocity; position+velocity+acceleration
    - or more complex, eg all joint angles for a person
  - Biggest problem -- Data Association
    - which image pixels are informative, which are not?
- Key ideas
  - Tracking by detection
    - if we know what an object looks like, that selects the pixels to use
  - Tracking through flow
    - if we know how an object moves, that selects the pixels to use

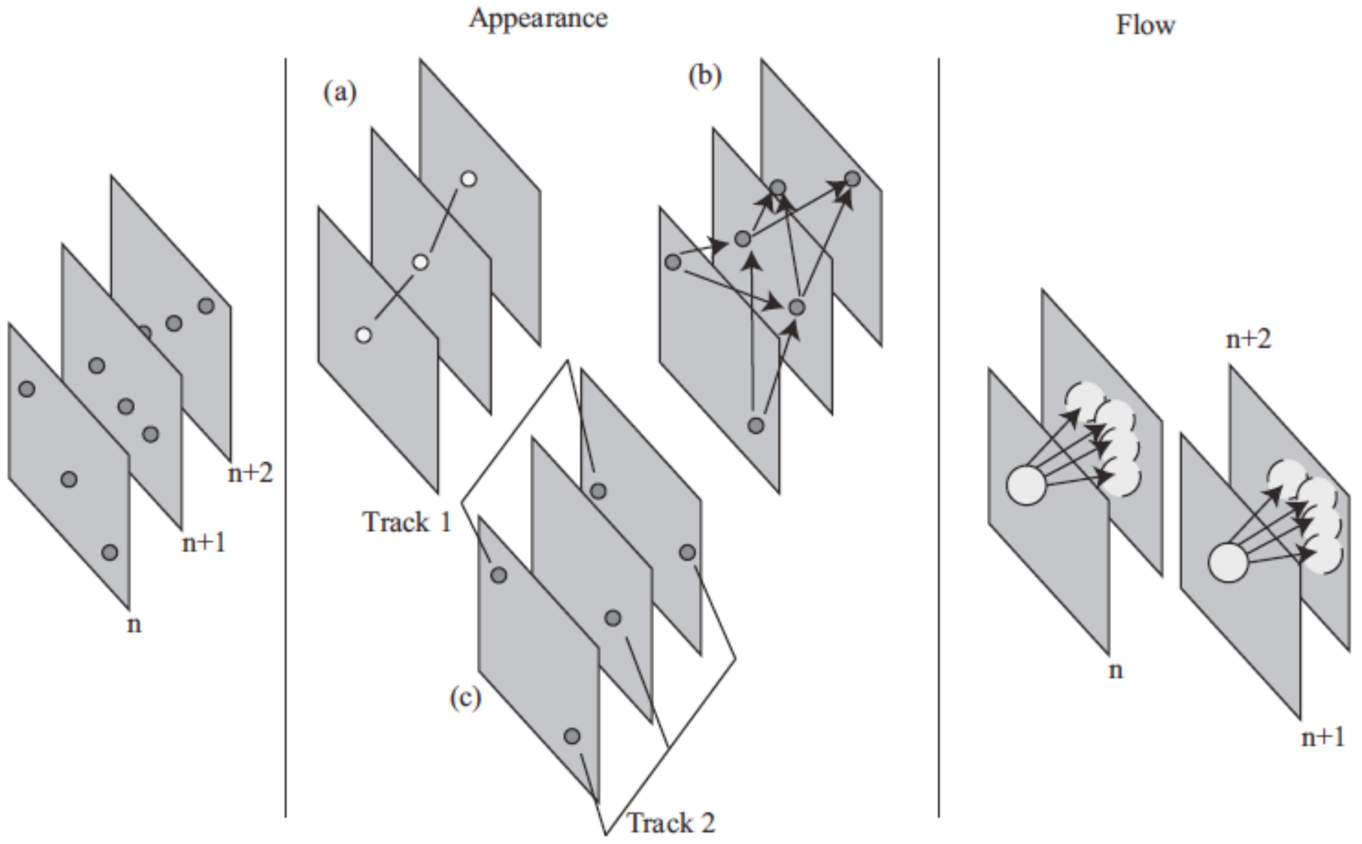
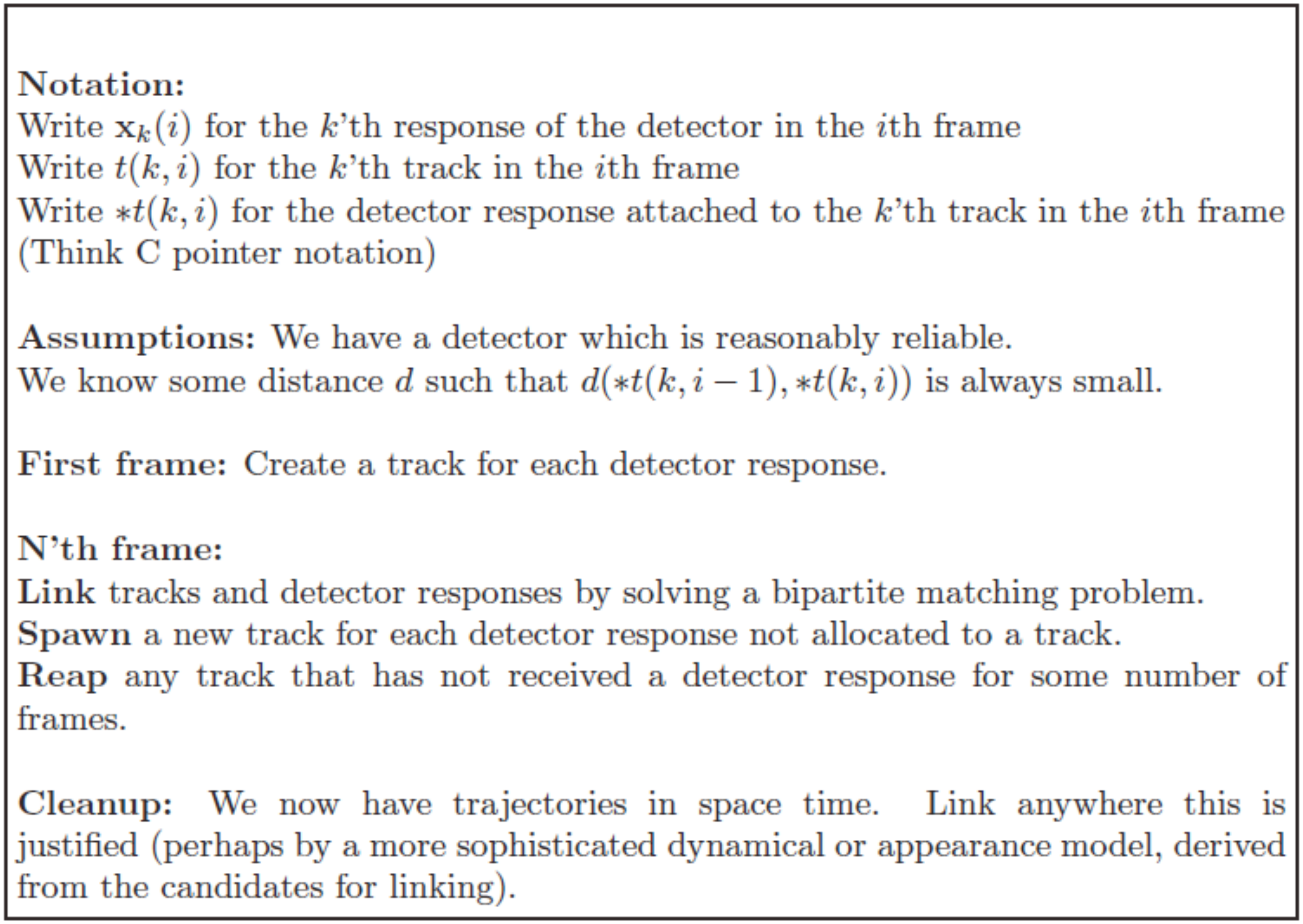


FIGURE 11.1: In tracking problems, we want to build space time paths followed by tokens—which might be objects, or regions, or interest points, or image windows—in an image sequence (**left**). There are two important sources of information; carefully used, they can resolve many tracking problems without 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**).

Track by detection (simple form)

- Assume
  - a very reliable detector (e.g. faces; back of heads)
  - detections that are well spaced in images (or have distinctive properties)
    - e.g. news anchors; heads in public
- Link detects across time
  - only one - easy
  - multiple - weighted bipartite matching
  - but what if one is missing?
- Better: create abstract tracks
  - link detects to track
  - create tracks, reap tracks as required
  - clean up spacetime paths



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.

Tracking by known appearance

- Even if we don't have a detector
- Know rectangle in image (n)
- Want to find corresponding rectangle in (n+1)
- Search over nearby rectangles
  - to find one that minimizes SSD error (Sum of Squared Differences)

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

  - where sum is over pixels in rectangle

- Application
  - stabilize players in TV sport

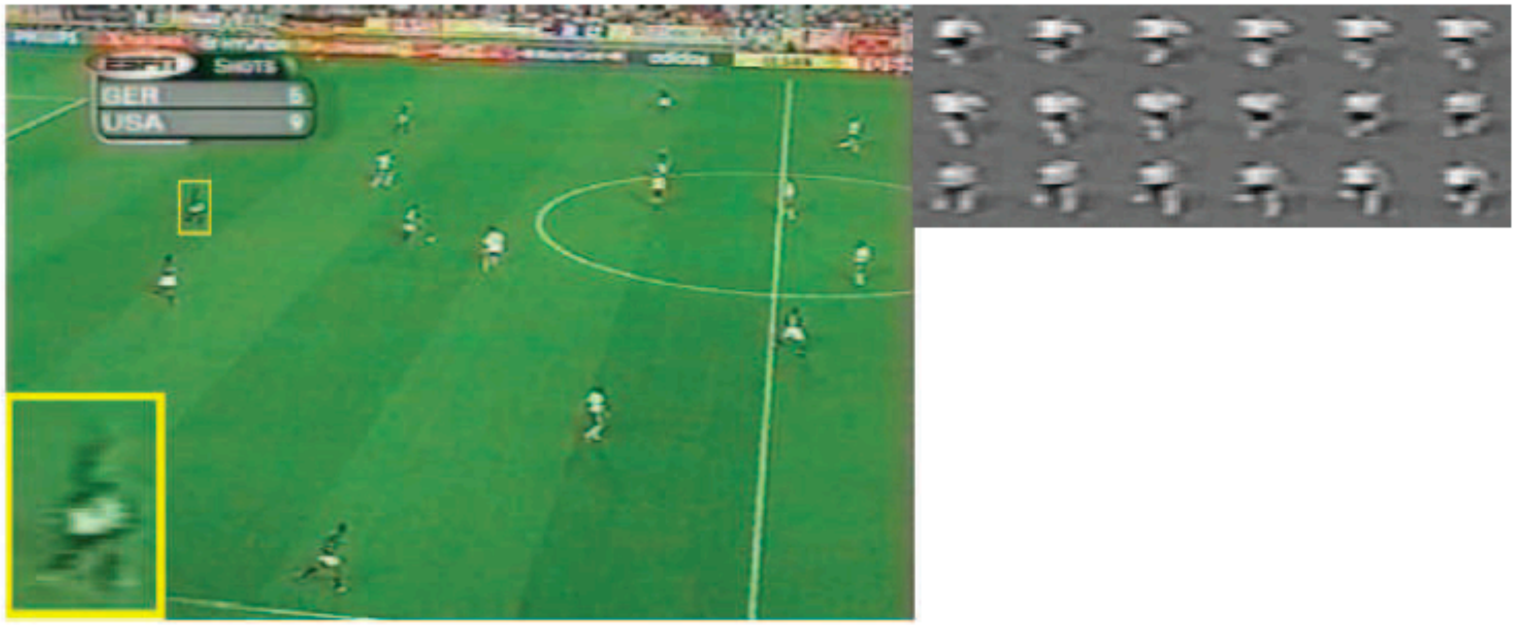


FIGURE 11.2: A useful application of tracking is to stabilize an image box around a more interesting structure, in this case a football player in a television-resolution video. A frame from the video is shown on the **left**. Inset is a box around a player, zoomed to a higher resolution. Notice that 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.*

Tracking by known appearance

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

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

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

This matrix is part of the Harris corner detector

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

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.

Building tracks

- Start at scattered points in image 1
  - perhaps corner detector responses
- For each in image (n), compute position in (n+1)
  - as in previous slide
- Now check tracks
  - patch in (n+1) should look like an affine transform of patch in 1
  - Prune bad tracks

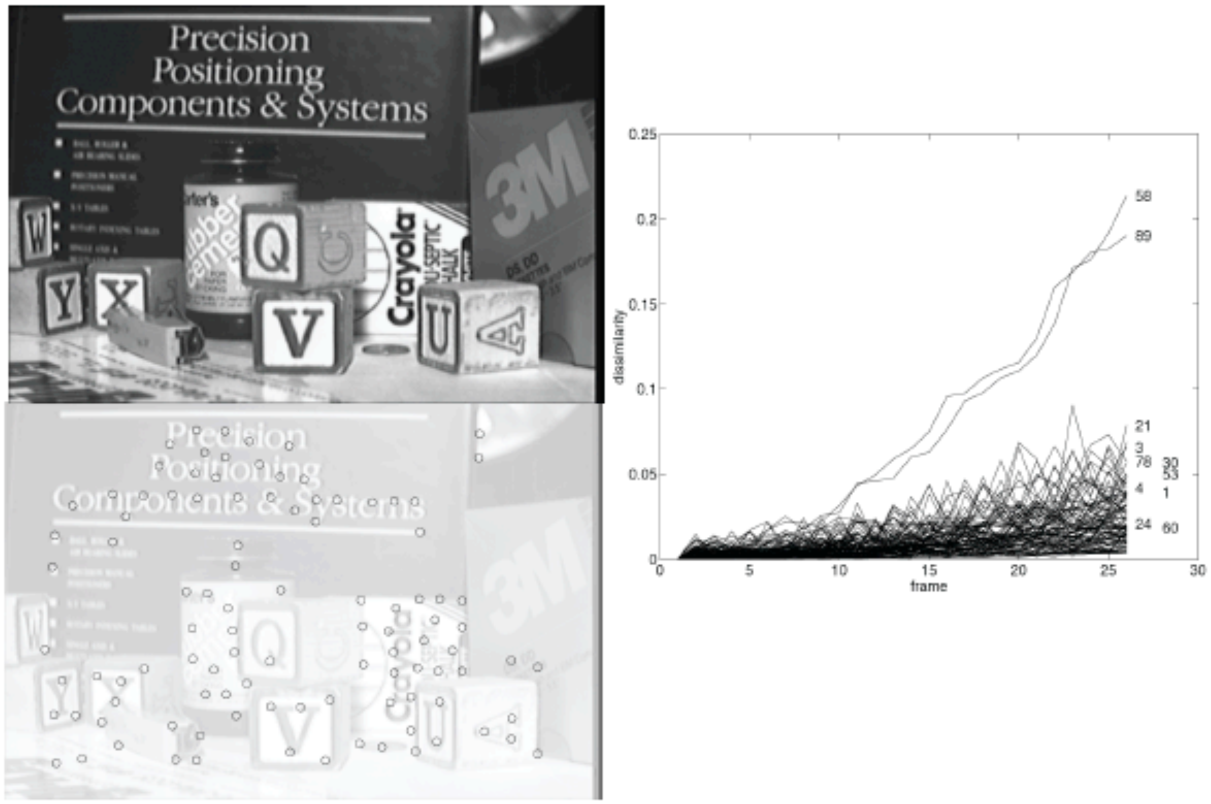


FIGURE 11.3: It is natural to track local neighborhoods, like those built in Section 5.3.2; however, for these neighborhoods to yield good tracks, they should pass a test of appearance complexity, shown in the text. This test checks that estimates of the translation of the neighborhood are stable. **Top 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.*

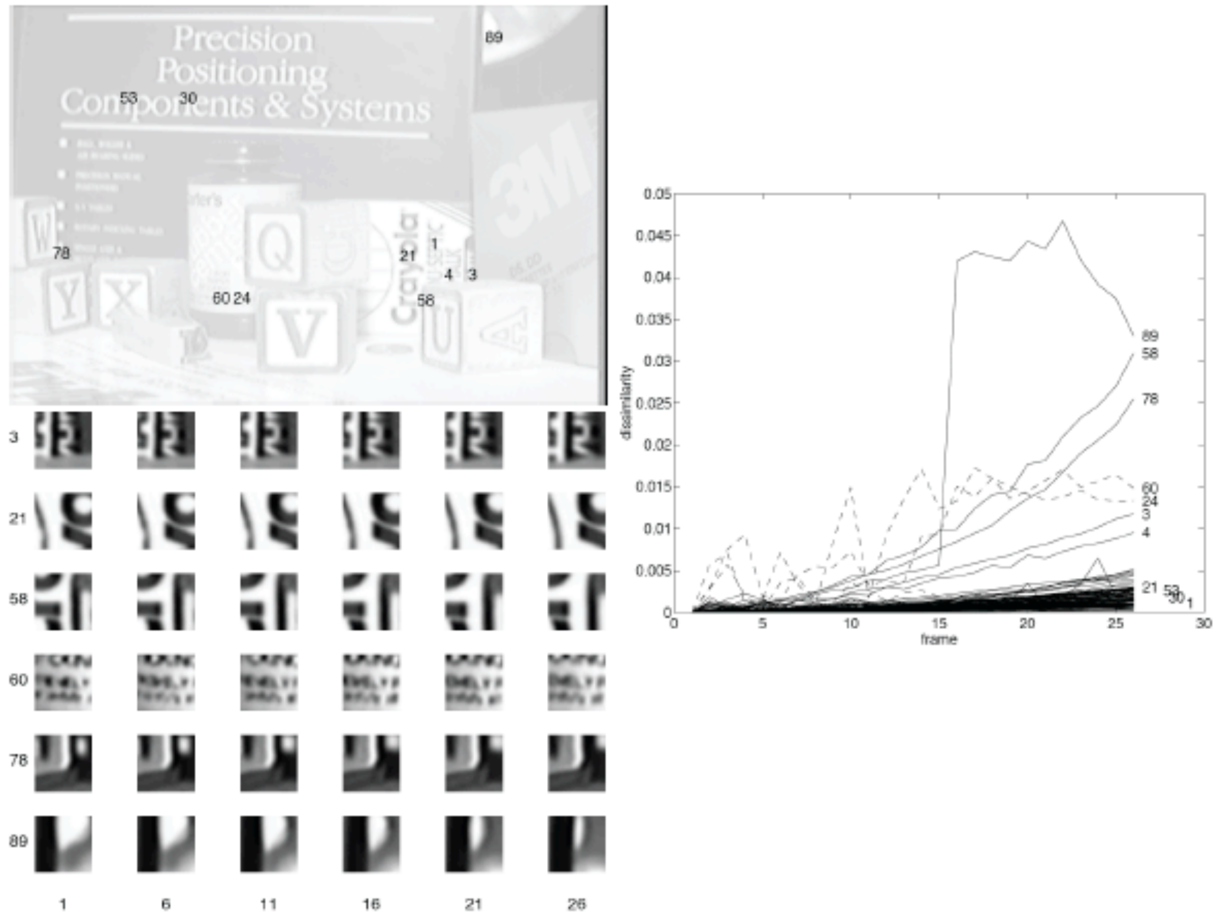


FIGURE 11.4: On the **top left**, the first frame of the sequence shown in Figure 11.3, with some neighborhoods overlaid. On the **bottom left**, the neighborhoods associated with these features (vertical) for different frames (horizontal). Notice how the pattern in the neighborhood deforms, perhaps 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.*

But what if the patch deforms?

- Eg a football player’s jersey
  - Colors are “similar” but SSD won’t work
- Idea: patch histogram is stable

- To track:
  - repeat
    - predict location of new patch
    - search nearby for patch whose histogram matches original the best

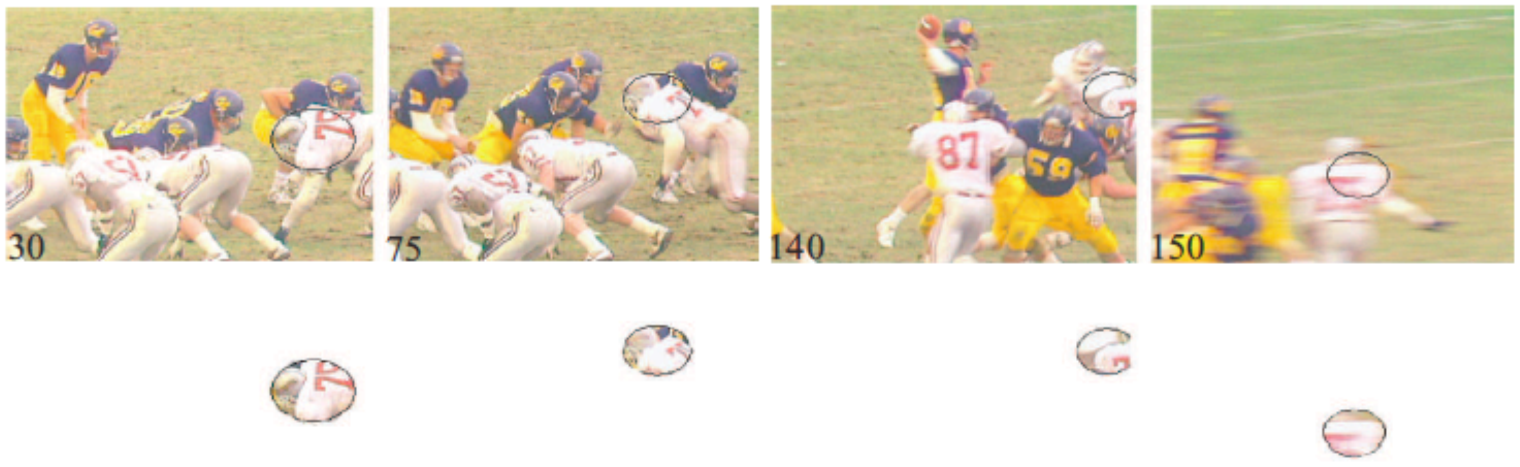


FIGURE 11.5: Four frames from a sequence depicting football players, with superimposed domains. The object to be tracked is the blob on top of player 78 (at the center right in frame 30). We have masked off these blobs (**below**) to emphasize just how strongly the pixels move around in the domain. Notice 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.*

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.

Q: Why no deep network methods?

- Silly A: I wrote these slides in 2011
- Better A: They've changed some feature mechanics
  - but nothing major
  - I'll work through some examples later

When are motions “easy”?

- Current procedure
  - predict state
  - obtaining measurement from prediction by search
  - correct state
- Easy
  - When the object is close to where you expect it to be
    - eg Object guaranteed to move a little
- Large motions can be easy
  - When they’re “predictable”
    - e.g. ballistic motion
    - e.g. constant velocity
- Need a theory to fuse this procedure with motion model

Q: why care about dynamics?

- Surely deep network methods make this go away?
- A: No they didn't
  - Some problems are "easy" with dynamical models, hard without
  - Example:
    - many similar, fast-moving cars