CHAPTER 43

Tracking by Detection

Tracking involves finding correspondences across frames. You must construct trajectories followed by some things – interest points, objects, people, for example – that appear in some frames in a sequence of images. The images usually come from video, but not always. Things might disappear behind other things, or behind scene features, so a pedestrian walks behind a tree and becomes invisible for some frames. Things might appear without warning, usually but not always at the sides of the frame. Things might move quickly or slowly – what is important is how predictable their movement is. There may be very many things in view (the very large number of pedestrians in Figure 36.3) and you will usually wish to ensure that correspondence is based on identity.

Tracking problems are of great practical importance. There are very good reasons to want to, say, track aircraft using radar returns. Other applications include *surveillance*, *motion capture*, *reconstruction*, and *targeting*. In surveillance applications, you want to understand the behavior of people or vehicles in some environment. You might do so to ensure that, say, trucks do not stop on active runways; or to ensure that people do not go into dangerous locations; or to keep track of who is doing what when; or to ensure baggage isn't stolen at airports. In motion capture applications, you want to measure the motion of something for future use. The main case is watching the detailed movement of human joints. You could do this online, so that a person could control a CGI character. You could do this offline, to build a record of human movement. You could use this record to drive animations, or to diagnose diseases. Tracking can be useful in structure from motion applications because you could use tracks to build correspondences. Tracking is particularly useful when you are reconstructing deforming objects, because you could obtain detailed local information about the object by tracking points on its surface. Finally, a really significant fraction of the tracking literature is oriented toward (a) deciding what to shoot, and (b) hitting it. This fraction of the literature used to focus on tracking using radar. Mostly, what people do with armed aerial drones doesn't make it into the vision research literature, but it is likely the case that (a) such craft carry cameras, because cameras are light and cheap and (b) use various tracking technologies.

It is now relatively unusual to track just one thing in a video. Even in applications like motion capture, you can expect to have more than one subject in many videos. The objects that are tracked in *multi-object tracking* (sometimes multi-target tracking) can be very dense in the scene (Figure 43.1). By far the most important case involves tracking very large numbers of people in public.

A small number of important strategies apply to most of the cases covered by this very broad scope. It is really helpful to be good at detecting what you want to track (Section 43.1). Really detailed descriptions of the appearance of the things



FIGURE 43.1: *Frames from an evaluation dataset illustrate the density at which modern trackers must track objects. On the* **left**, *a frame from the MOT20 challenge dataset, showing a reasonably dense set of pedestrians in a scene. There are some other objects which must be tracked, too. The* **center** *shows the ground truth. Look closely, and notice that each box has a different number. The tracker must maintain the identity of distinct pedestrians. Further, notice that there are boxes on some umbrellas and other objects. On the* **right**, *reference boxes from a Faster-RCNN detector.*

you are tracking are very helpful. It is often very useful to know how the things you are tracking will move (Section **??**).

43.1 TRACKING BY DETECTION

Tracking by detection (STbD) is a simple tracking strategy that assumes you have an object detector (Chapter 36.3; Section 36.3). A straightforward procedure is to detect objects in frame 1. Now iterate:

- Detect objects in frame $i + 1$.
- Compute correspondences between the objects in frame i and those in frame $i + 1$. Each object in frame i is linked to at most one object in frame $i + 1$. Each object in frame $i + 1$ is linked to at most object in frame i.

The result is a sequence of links between object instances in frames. Each such path is a track. However, these paths are broken up by objects that are not detected for some frames (Figure 43.2). This is a significant nuisance. You expect objects to disappear on occasion – they might go behind an occluder, or the detector might produce a false negative. Similarly, objects might appear from behind an occluder, or just be a false positive response.

The procedure is easily modified to something extremely useful. Rather than linking detector responses across frames, you maintain a set of abstract tracks. Each track has some information associated with it. For example, it might contain the appearance of the object the last time you saw it, when you last saw it, and some motion information. The master recipe is in a box, below.

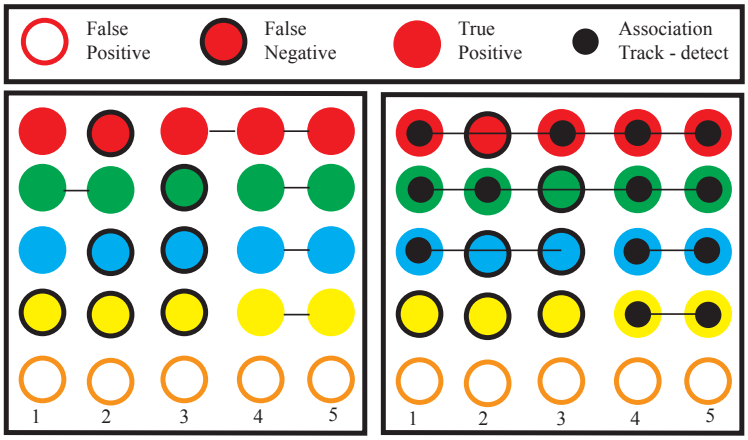


FIGURE 43.2: *The simplest tracking by detection procedure breaks up tracks when objects are occluded. Allowing abstract tracks solves this problem. Each column is a frame; each colored circle is a detector response. Full colored circles are detector responses that correctly identify an object. Colored circles with a black boundary are false negatives – the object is in the frame, but the detector didn't respond to it. Each object is represented by a consistent color. Empty circles with colored boundaries are false positives – there is no underlying object. For these, the color has no particular meaning because there is no underlying object. The simplest procedure (***left***) just links detector responses to the next frame. There is no way to represent an object that is missed for a frame, so the tracks break up. If you allow abstract tracks (***right***), tracks can be kept even if a measurement is missing. An association between a track and a detector response is given by a small dark circle. There isn't one on a false negative, because there is no measurement to link the track to. For this figure, the track is allowed to go one frame without a measurement, and is then reaped. A new measurement results in a new track being created (**blue**; **yellow**).*

Procedure: 43.1 *Tracking by detection links detector responses to abstract tracks.*

Detect objects in frame 1, and create one track per object. Now maintain tracks by iterating:

- Detect objects in frame $i + 1$.
- Compute correspondences between the objects in frame $i + 1$ and the tracks. Each track gets at most one object. Each object goes to at most one track.
- Update each track that received an object in frame $i + 1$.
- If a track does not have a corresponding detect, hold onto it for some number of frames. If it has had no associated detect for several frames, reap the track.
- If you have a detect that cannot be associated with any current track, create a new track.

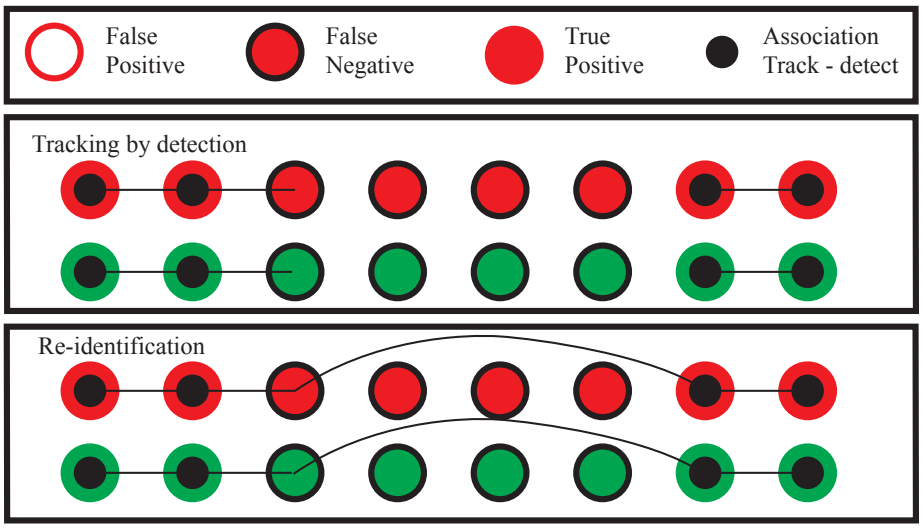


FIGURE 43.3: *If an object is occluded for long enough, procedure 43.1 will break up the track. Reidentification creates links between object detector responses where there is strong evidence the object is the same.*

The result is a set of tracks associated with object instances in many, but not all, frames. It is useful to draw the relations between detector responses and tracks, and Figure 43.2 shows one way to do this.

One important algorithmic issue is *correspondence*. You need to be able to link the abstract track to the object detector responses. Some detector responses might be false positive, and so should not be linked. You expect that, with a modern detector, if there are two largely overlapping objects you will get two detector responses, so no detector response can be linked to more than one track. Similarly, the objects of interest might result in false negatives, so a track might not be linked to any detector response in a given frame. Objects of interest don't break up, so no track can be linked to more than one detector response. You must construct a correspondence that meets these constraints and ensures that detector responses are linked to the right track to the extent possible.

Another important algorithmic issue is *re-identification*. Assume that an object is not detected in some frames, then is detected. This could happen because it went behind something else, or because the detector produce some false negatives. You might then have two tracks associated with one object (Figure **??**). Re-identifying the object in when it reappears involves linking these up to form one track.

43.1.1 Computing Correspondences

The natural procedure is to construct a weighted bipartite graph. Assume you have N_i detections in frame i and N_T tracks. Draw the graph as in Figure 36.3.

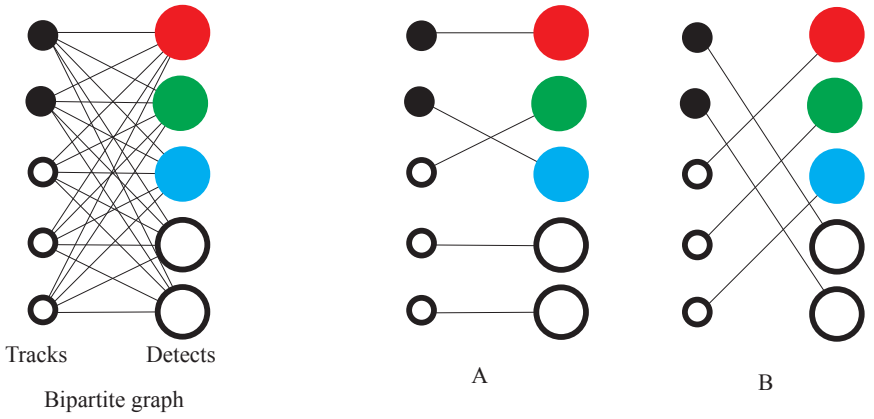


FIGURE 43.4: *Match a track to a set of object detector responses using bipartite graph matching. Vertices are tracks (***left, filled***) with missing tracks (***left, empty***) and responses (***right, filled***) and missing responses (***right, empty***). In matching A, the* **red** *response is linked to the first track, the* **blue** *to the second, and the* **green** *is not linked to a track (which would imply spawning a new track to accommodate the green response). In matching B, no track gets a response and no response gets a track, so three new tracks will need to be spawned.*

On the left, a column of $N_i + N_T$ vertices; N_T of these correspond to tracks, and the remainder are an accounting device that represents a missing track. On the right, a column of $N_i + N_T$ vertices; N_i of these correspond to detects, and the remainder are an accounting device that represents a detection failure. Construct all edges connecting left and right, *except* edges linking missing tracks to detection failures. Edges that connect a track to a detect are weighted with a measure of similarity between the track and the detect, where a smaller weight means greater similarity. Edges that connect a track to a detection failure get a large weight. Edges that connect a missing track to a detect get a small weight. Now compute the minimum weighted bipartite matching. This is a standard procedure that usually uses something often known as the *Hungarian algorithm*. You can find an implementation in good API's (SciPy has one, for example).

Assume you have a matching. For every edge linking a track to a detect, attach that detect to the track. Using that detect, update the appearance information in the track. For every edge linking a track to a detection failure, decide whether to reap the track – perhaps there have been no measurements for many frames – or to hold on to it. Finally, for every edge linking a missing track to a detect, create a new track and populate it with any information that is useful for improving detection.

Weighted bipartite graph matching is cubic in the number of vertices. This means you may need to be careful about reaping tracks. It is quite common to replace weighted bipartite matching with a greedy algorithm. The greedy algorithm initializes a running graph with the original bipartite graph, then iterates the following until there are no edges. Choose the edge in the running graph with highest

weight; insert this into the matching; then erase all edges that share a start point or an end point with this edge. Theoretical results guarantee the value of this approximation is good compared to the optimal solution []. This approximate procedure is in practice quite strong, particularly when the weights are seldom ambiguous.

There is a straightforward way to start TbD. Apply a detector to the first image and promote each detected object to a track. For some applications it is better to use an interactive procedure. For example, a user might identify the particular things to be tracked.

43.1.2 Computing Correspondence Weights using Appearance

In the straightforward tracking by detection framework, the quality of similarity weights is crucial. If weights are seldom ambiguous, then the procedure will work very well. One way to obtain good weights is to use a strong embedding. Image embeddings are an extremely effective source of appearance descriptions. The main question is managing the dimension of the description.

GHOST (for Good Old Hungarian Simple Tracker – authors assert "the order of the letters of the acronym does not change the product") is a very strong multiple object tracker that uses TbD and a carefully constructed appearance model []. This tracker tracks people. The appearance model is a 50 layer ResNet (Section 36.3) trained on static frames from a person re-identification dataset. The ResNet is followed by a batch normalization layer and a layer that reduces the dimension of the feature. At runtime, the batch normalization layer sees mean and standard deviation of features taken over the detector responses in the current frame. This successfully adjusts the appearance representation to account for frame specific effects. Tracks are *active* if they received a detector response at the previous frame, *inactive* otherwise, and different matching thresholds are used for these two cases. You should see this as a form of the hysteresis of Chapter 36.3. Now assume you wish to compute weights between tracks and detector responses at frame i. Write $\mathbf{f}_{i,j}$ for the feature vector describing the detector response associated with the j'th track in the i'th frame, if there is one. The appearance weight for an *active* track j to the k'th detector response in frame i described by $\mathbf{f}_k$ is the distance $d(\mathbf{f}_{i-1,j}, \mathbf{f}_k)$. If the track is *inactive*, form an average of the distances $d(\mathbf{f}_{u,j}, \mathbf{f}_k)$. Here u ranges over the N_k previous frames in the track where a measurement was associated with the track.

43.1.3 Exploiting Dynamical Models in Correspondence Weights

Imagine the things you are tracking move significant distances from frame to frame. If each has a distinctive appearance, then this may present no problems. Well constructed appearance weights in the matcher should be enough to match track to object. If they do not, you may need to rely on what you know about dynamics. It is (relatively!) straightforward to model the movement of objects (Section 36.3). If you have a movement model, you can use it to predict the location of the object in frame i. If the movement model is sufficiently accurate, and if there is a sufficiently small stochastic component, this prediction will be accurate. In turn, it can be incorporated into the correspondence weight. Do this by forming the sum of appearance and dynamical weights.



FIGURE 43.5: *On the* **left**, *an identity switch. A true positive (the* **red** *responses) has a different predicted identity (or track number) in the two frames indicated. MOTA counts identity switches. On the* **right**, *an identity transfer, where the same track has different ground truth identities in two successive frames. MOTA does not count identity transfers.*

Once you have constructed correspondence between tracks and objects, you have measurements of the location of the object in the image. These measurements – and, perhaps, others – can be fed into the dynamical model. Depending on the model, quite noisy initial measurements of position can, over time, turn into quite accurate predictions. Details appear in Section 36.3.

GHOST uses a very simple dynamical model. Assume you wish to compute weights at frame i linking track j to detector response k. Compute the velocity of that object at N_v previous frames. Do so by computing a difference between locations, doing the natural thing for frames where no object is associated with the track **exercises**. Now average these velocities, and use the average to predict the location of the object in frame i. Now use this information to compute a dynamical weight. You could use, for example, the distance between predicted and true locations. GHOST uses the boxes predicted by the detections, and computes

$$1 - \text{IoU}(\text{box predicted by dynamics on track } j, \text{box of response } k)$$

as a dynamical weight.

43.1.4 ReIdentification

Either detection failure or occlusion might mean an object does not produce detector responses for some frames. Not knowing where an object is in some frames does not mean it cannot be tracked. If you keep a representation of the object's appearance for each frame in the past where you did link the track to a response, you can use this information to find the object the next time it appears.

This is one example of reidentification. In the easy case, the object disappears for one or two frames, then reappears. For example, a person might disappear behind a lamppost. Here the most recent observations are very recent, and so are likely a fair guide to what the object looks like. In the harder case, the object has disappeared for many frames. The track may even have been reaped. For example, the camera might be viewing a baggage claim area in an airport. A person may

pick up a bag, then go into a bathroom. Some time later, the person emerges from the bathroom, and you would like the tracker to know that this is the same person, whatever the interval. As another example, the camera sees a person walking with a backpack. Some time later, that person is seen in a different camera, now without a backpack. Ideally, you would like the tracker to know that these are the same person.

Reidentification typically involves training an appearance feature using example detector results. You could train the feature using a classification approach – are these boxes the same thing? Alternatively, you could use a metric learning approach, and so train the feature so that boxes that do match produce features that are closer together than features from boxes that don't match.

Reidentification is particularly important in many human tracking applications. For example, reidentification procedures can be used to link tracks in one camera to tracks in another camera. Many surveillance applications need to produce a result that follows a particular person over a long interval of time. As a result, reidentification can be a controversial technology. A person tracking application might be used in a reasonable attempt to document and prosecute illegal behavior. In the backpack example, the tracker might have identified someone who placed an explosive device. Reasonable societies should and do regulate that sort of behavior. The same application might be used in a way that is intrusive, authoritarian or creepy. In the backpack example, the tracker might be used to identify someone who delivered political literature that a government finds inconvenient. There is no universal standard to distinguish between these cases, and research conducted for benevolent reasons might have quite unattractive outcomes. As a result, human reidentification research is controversial.

43.1.5 Evaluation

Evaluating multi-object trackers is delicate. You must evaluate whether objects that should have been found, were found. You must also evaluate whether objects are assigned to tracks coherently. As in evaluating detectors (Section **??**), there is a correspondence problem. You must determine which, if any, prediction corresponds to which, if any, ground truth instance. This requires a notion of similarity. If predictions and ground truth instances are represented as boxes, use IoU as similarity (cf Section **??**). On occasion, predictions and ground truth are represented as points; in this case, use one minus the Euclidean distance between the points as similarity. No correspondence is allowed if the similarity is less than some threshold α.

MOTA (Multiple Object Tracking Association) emphasizes accurate detection. This uses the idea of an identity switch. Each object detector response inherits an identity from its track. This is the number of the track. An identity switch occurs when a true positive has different predicted identity in frame i and frame $i+1$ (Figure 43.5). For frame i, compute the number of false positive detections $FP(i)$, the number of false negative detections $FN(i)$, the ground truth number of objects $G(i)$, and the number of identity switches $IS(i)$. Then the MOTA is

$$1 - \frac{\sum_{i \in \text{frames}} \left(FP(i) + FN(i) + IS(i) \right)}{\sum_{i \in \text{frames}} G(i)}.$$

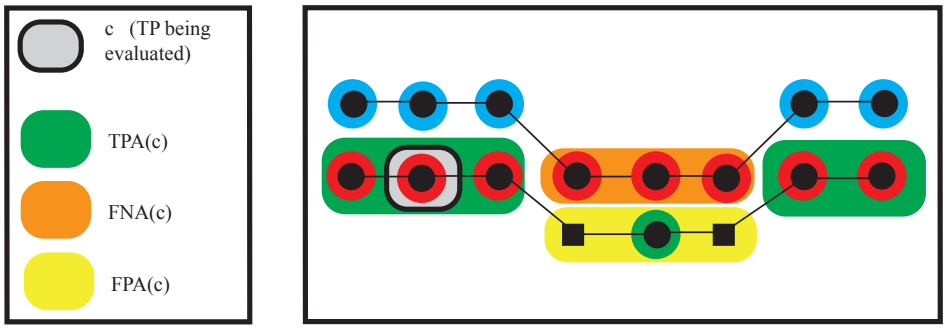


FIGURE 43.6: *MOTA evaluates associations for each true positive in each frame. At a true positive c in a frame (the* **gray** *window), MOTA uses the ground truth identity ($gti(c)$) and the predicted identity ($pri(c)$ – which track the true positive is assigned to). True positive associations are all pairs (gti, pri) such that $gti = gti(c)$ and $pri = pri(c)$ (the* **green** *window). False positive associations are all pairs (gti, pri) such that $gti = gti(c)$ and $pri \neq pri(c)$ (the* **orange** *window – here the tracker assigned the object a new identity compared to c). False negative associations are all pairs (gti, pri) such that $gti \neq gti(c)$ and $pri = pri(c)$ (the* **yellow** *window – here the tracker assigned the identity a new object compared to c).*

Scoring well on a MOTA evaluation requires strong detector performance. However, MOTA is not a sensitive measure of accurate identity management, because it does not measure identity transfers. An identity transfer occurs when the same track has different ground truth identities in successive frames (Figure 43.5). Further, MOTA does not measure localization of the detector responses.

The tracker of Procedure 43.1 constructs correspondences between tracks and identities. These are not used directly in evaluating MOTA. Instead, the matching of predictions to ground truth is performed to optimize the final MOTA score. In each frame, fix matches that (a) have similarity greater than α and (b) don't cause an identity switch (which you compute using the previous frame's matching from the tracker). For any remaining potential matches, use weighted bipartite matching to select a set of matches that maximizes the number of true positives. Break ties by maximizing the mean of similarity across the true positives.

A more sensitive and flexible measure is HOTA (Higher Order Tracking Accuracy). MOTA evaluates associations for each true positive in each frame. At a true positive c in a frame (the **gray** window), MOTA uses the ground truth identity ($gti(c)$) and the predicted identity ($pri(c)$ – which track the true positive is assigned to). In a true positive association, the tracker assigned the object the same identity compared to c. Write $TPA(c)$ for the number of true positive associations for c, which are all pairs (gti, pri) such that $gti = gti(c)$ and $pri = pri(c)$. In a false positive association, the tracker assigned the object a different identity compared to c. Write $FPA(c)$ for the number of false positive associations for c, which are all pairs (gti, pri) such that $gti = gti(c)$ and $pri \neq pri(c)$. In a false negative association, the tracker assigned the object a different identity compared to c. Write $TPA(c)$

for the number of false negative associations for c, which are all pairs (gti, pri) such that $gti \neq gti(c)$ and $pri = pri(c)$. Now write

$$A(c) = \frac{TPA(c)}{TPA(c) + FPA(c) + FNA(c)}.$$

The MOTA for a given α is then

$$\sqrt{\frac{\sum_{c \in \text{true positives}} A(c)}{TP + FP + FN}}.$$

As in MOTA, HOTA is computed using a matcher that adjusts associations to obtain the best score. In, for example, forensic applications, preserving identity is extremely important. In other applications, it may be more important to detect everything. An attractive featureof MOTA is that straightforward variants emphasize either identity preservation or detection. Details appear in [].

43.2 TBD VARIANTS

It should be clear from the TbD recipe that tracking can be very tightly coupled to detection. In fact, quite simple modifications of existing learned object detectors can produce very good trackers. I give two examples from a substantial literature below. Object detectors can be adjusted to produce accurate object masks (Section 36.3), and so trackers can, too. Finally, because a region proposal network can predict whether an object is present without knowing the name of the object, you can track objects whose category is not known to the detector.

43.2.1 Exploiting FasterRCNN

FasterRCNN can be exploited to track in a very natural way. Assume you have a set of boxes in frame i. You wish to find those boxes, and any other objects that have appeared, in frame $i+1$. Tracktor [] applies ROIpool to the features for frame $i+1$ using each box from frame i for locations. The resulting features are then fed to the bounding box regression network. If the object hasn't moved too much, the regression network should be able to determine its new location accurately. The box is then scored using the classification network. New boxes come from the region proposal network applied to frame $i+1$, and proceed through FasterRCNN as one would expect. There is now a pool of new boxes that are over threshold and old boxes that have been propagated. Apply non-maximum suppression to this pool, and you have the final set of detects in frame $i+1$ together with links to boxes in frame i. Boxes that do not have links are associated with new tracks, and boxes that do have links are associated with the track they had in frame i. You can improve performance somewhat with a simple reidentification network.

43.2.2 Exploiting Detr

Recall that Detr is an object detector built using a transformer (Section 36.3). Image tokens are transformed into detection tokens using learned object query tokens. In turn, the detection tokens are decoded into either "no object" or a bounding box and category scores.

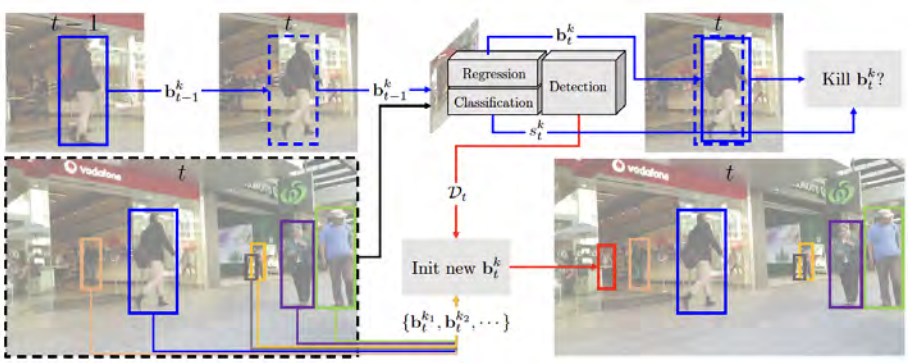


FIGURE 43.7: *Clever exploitation of the structure of FasterRCNN (Section 36.3) can produce a strong multi-object tracker. Tracktor [] uses the existing bounding box regression machinery by ROIPooling to refine each detect in frame $t-1$ into a box in frame t (***blue** *pathway in the figure). The classification head computes a score for the refined box (***blue** *pathway). FasterRCNN's region proposal network proposes further boxes in frame t. The resulting pool of boxes is pruned using (a) classification score and (b) non-maximum suppression (***red** *pathway).* Image credit: *Figure 1 of "Tracking without bells and whistles", Bergmann et al, 2019.*

Detr can be exploited for tracking by detection. At frame i, TbD needs to (a) resolve what happened to objects present in frame $i-1$ and (b) detect objects that were not in $i-1$ and have appeared for the first time in frame $i-1$. Adjust Detr to do this by taking tokens that decoded into objects in frame $i-1$, then attaching them to the query tokens for frame i (Figure 43.8). There will be a variable number of query tokens, because the number of objects in frame $i-1$ varies. Some training is required to ensure that tokens that have been passed on can be interpreted properly. The result is Trackformer.

Trackformer can be trained with consecutive pairs of frames from a labelled sequence. The overall scheme is as in Detr – you must solve a matching problem to match tokens with ground truth. The matching regime can be changed somewhat because you expect that tokens from the second frame produced by objects in the first frame are relatively easy to identify. Tokens corresponding to new objects require more detailed matching.

43.2.3 Tracking in an Open World

Master recipe 43.1 requires (a) you know something about where an object is and (b) you can connect responses from frame to frame. It *does not* require that you know the name of the object. Further, region proposal networks are very successful at identifying boxes that are likely to contain an object, independent of the name of the object. This means that it can be applied quite successfully to track objects that cannot be classified by the detector. The result is an *open world tracker*.

An open world tracker can be built on MaskRCNN (Section 36.3). Doing

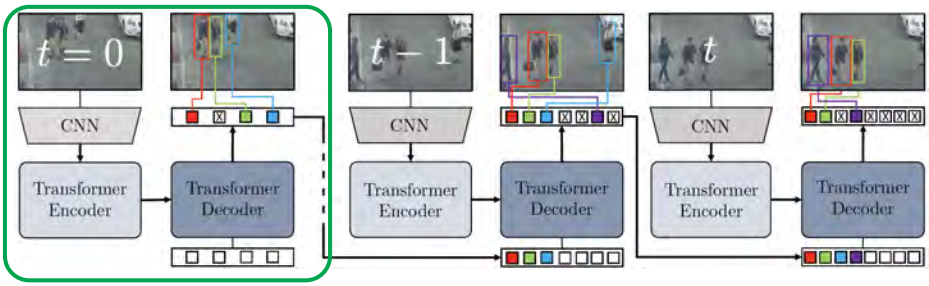


FIGURE 43.8: *Detr uses a transformer to combine image tokens and object query tokens into detection tokens, that are then decoded into either "no object" or a box location and category scores (***green** *box). In Trackformer [], the detection tokens are adjusted and passed on as query tokens. These are combined with tokens for the next frame to produce detection tokens, and so on.* Image credit: *Figure 1 of "Tracking without bells and whistles'TrackFormer: Multi-Object Tracking with Transformers", Meinhardt et al, 2021*

so requires knowing which of the 1000 proposed boxes should be accepted as an unknown object, and which rejected. It turns out that combining the objectness score computed by the region proposal network and the background score from the classification head yields the most reliable procedure. Testing the combined score against a threshold identifies the boxes of interest. A number of tests can be used to identify which object in frame $i + 1$ corresponds to which in frame i. You could test IoU between proposal boxes; IoU between object masks obtained from MaskRCNN; warp the object region in frame i using optic flow, then test IoU against the region in IoU; or test appearance. Appearance tests are difficult to get right, because you need an appearance model that is capable of telling when two pictures of an unknown object are the same. Here the fact you don't know the object creates difficulties. A strong baseline tracker can be built using optic flow and box IoU [].

43.2.4 The Uses of Memory

Master recipe 43.1 attaches detector responses to tracks, but does not specify what is attached, how long it is kept, and so on. An explicit memory of past tracks is useful, because it allows you to build a kind of model of how things change in appearance. Trackformer contains such a model (because it is trained on pairs of consecutive frames), but the model is parametric and is constrained to what has been seen in training. In contrast, a memory could supply an non-parametric model specialized to each object being tracked. So, for example, if a particular person has a tendency to put on and take off a unique hat, the memory might record this fact for this person. Such a model could improve association between tracks and detector responses. However, a long memory may force a model to reproduce mistakes in association it has made in the past.

MeMOT maintains an explicit memory of past appearances in each track. If there is no instance associated with a particular track in a particular frame, that is



FIGURE 43.9: *Tracking by detection can track objects which a detector has been trained to find and name (***left** *part of each pair, labelled "known objects"),* and *objects that can be found by a region proposal network (***right** *part of each pair, labelled "unknown objects"). This particular detector does not know about bulls, or cloaks, and so can detect and label only matador (masked orange) and some of the audience in the* **top left** *frame. But the region proposal network can find other objects, and so identify bull and cloak and other audience members. Propagation of proposals across time is straightforward, details in text.* Image credit: *Figure 1 of "Opening up OpenWorld Tracking", Liu et al, 2022.*

recorded too. A transformer encoder/decoder produces a set of tokens representing the possible objects in the image (as in DeTR, Section 36.3). These tokens are concatenated with a set of tokens from memory. The resulting set is decoded to identify instances that should be attached to tracks and those that are new objects. Some image tokens that represent objects are suppressed, because they are explained by memory tokens. Others are background, or novel instances. Tokens that represent objects are then used to update the memory. Memory imposes hardware costs (you have to keep it somewhere) and for this case can stretch to 24 frames in the past. There is good evidence that a longer memory produces improvements in association.

43.2.5 Tracking Segmentation

Now assume you have a very strong image segmenter, as in Section **??**. You could apply the tracking by detection procedure to segments as well. If you have a memory, segments in frame i, $i-1$, ..., $i-k$ can be advanced to frame $i+1$ by (for example) warping them along optical flow. This provides a pool of segment predictions from the past that can be compared to the current set of segments.

Tracking by detection as I have described it is *causal* (or online) – you use information from the past to link the current detector responses to a track, but you do not use information from the future. However, you can use information from the future if you have it. For example, you might be working offline, or you might be willing to tolerate a degree of latency. In segment tracking, you could warp segmentations of future frames backward in time using optical flow as well. This would increase the pool of possible segment predictions for the current frame.

Now take the pool of segments (past, and if you have them, future) and compute a prediction of which current segments correspond to which in the pool.

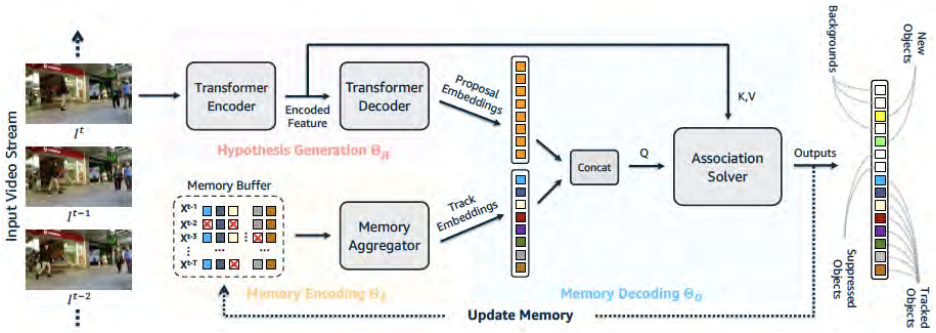


FIGURE 43.10: *MeMOT maintains a memory buffer containing image embeddings associated with tracks seen in the past. This memory buffer is used to support allocating object windows to tracks and to identify instances that have not been seen before. A transformer encoder/decoder produces a set of tokens representing the possible objects in the image (as in DeTR, Section 36.3). These tokens are concatenated with a set of tokens from memory. The resulting set is decoded to identify instances that should be attached to tracks and those that are new objects. Some image tokens that represent objects are suppressed, because they are explained by memory tokens. Others are background, or novel instances. Tokens that represent objects are then used to update the memory.* Image credit: *Figure 2 of "MeMOT: Multi-Object Tracking with Memory", Cai et al, 2023.*



FIGURE 43.11: *You can track segments using a variant of the TbD recipe. Assume you have a very strong image segmenter. Apply this to each frame to break it into regions corresponding to objects. Relatively straightforward methods can be used to link segments across frames. More sophisticated procedures are required to use past frames to refine the segmentation of the current frame. Even future frames can be used. In the three frames from two sequences, shown above, the color of the masks shows the correspondence from frame to frame.* Image credit: *Figure 1 of "Tracking Anything with Decoupled Video Segmentation", Liu et al, 2022.*

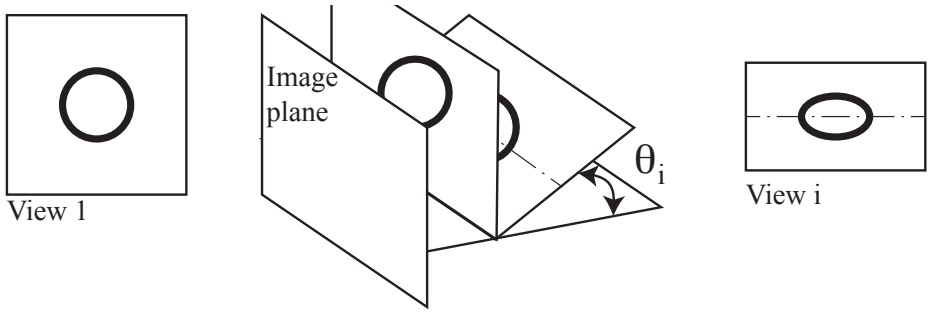


FIGURE 43.12: *Look at a circle on a plane in a perspective camera. If the plane is parallel to the image plane, you see a circle (***left***); if the plane is tilted (***center***), the circle is foreshortened (***right***). I have suppressed the focal point because the circle is too small to display perspective effects. This foreshortening is not modelled by rotation, translation or scale. The correct window to compute the interest point descriptor is a circle in image 1, but an ellipse in image i. This means that the descriptor computed using the algorithms of Chapter 36.3 is likely different for frame i and frame 1.*

Liu et al frame this as an integer programming problem []. Ideally, you want to attach a current segment to a track where it has a very large overlap with many past segments from that track and which is supported by many future segments.

43.3 POINT TRACKERS

The tracking by detection recipe could be applied to the interest points of Section 36.3. Recall that you find corners (points that can be localized); at each corner, you estimate a scale and rotation; and in the resulting window, you obtain some feature (SIFT in that section, but others are available). Now regard each corner as an object detector response. You can now use the description of the feature to link points to a track, and the master recipe for tracking applies. A version of this approach is known as the *Lucas-Kanade method*.

43.3.1 Managing Foreshortening for Interest Point Tracking

If you use the master recipe directly, you will discover that your ability to track a point decays with time. The feature construction is covariant to rotation (because of the orientation estimate); translation (because the center is localized); and scale (because you estimate scale using a laplacian of gaussian). Now imagine a small circle on a plane. The foreshortening caused by tilting this plane backward is not not covered by rotation, translation and scale. If the circle is small, there is no notable effect of perspective. But one diameter is foreshortened, and the other is not (Figure **??**). In the right coordinate system, you can model the effect of tilt as

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

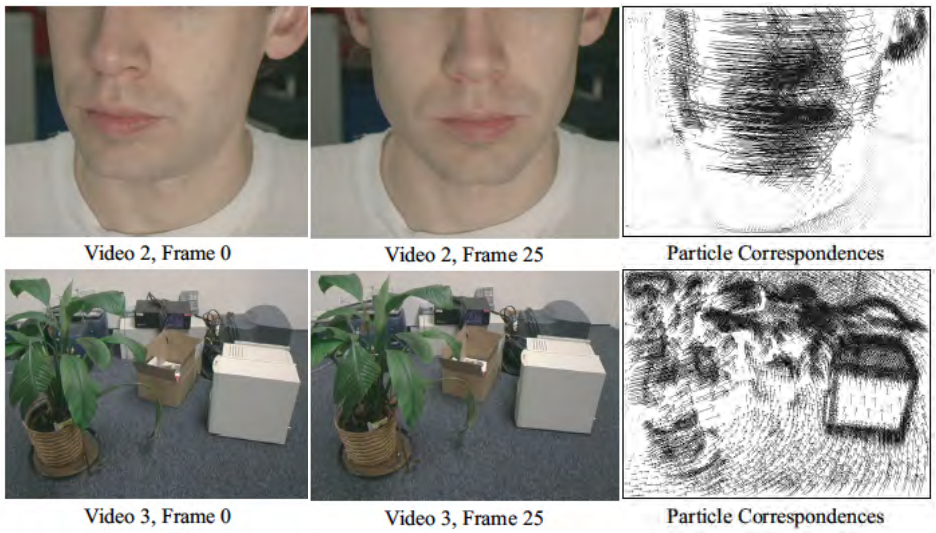


FIGURE 43.13: *You can build relatively dense, relatively long term, tracked representations of points in images using optical flow. Choose a collection of points in the first frame, then allow points to flow along the flow field. Maintain the representation at each frame using two procedures. Reap points if the flow is unreliable, and add points in regions where the point density is too low. The result is a collection of short to medium time point tracks that are relatively dense in the image.* **Left** *column shows first frame of two sequences;* **center** *column shows late frames in those sequences;* **right** *column shows the resulting point tracks. Some tracks survive for long periods and others for shorter periods. The points have fairly high density. Figure constructed using parts of Figure 6 of "Particle Video: Long-Range Motion Estimation using Point Trajectories", Sand and Teller, 06.*

This means that there is an anisotropic scaling, which isn't covered by the scale estimate. Now choose some interest point in frame 1, and assume that it is frontal (lies on a plane parallel to the image plane). Foreshortening means that the descriptor you compute for this point in frame i may not match the descriptor you computed in frame 1. You could improve behavior by adjusting the matching procedure. To compute the similarity in the bipartite matcher, take the window at frame i and compute an affine transformation that transforms the intensities to that in frame 1 **exercises**, then compute the goodness of the match. Maintaining a memory of the track will help, too. Preserve the descriptor computed for each window along the track. If the memory is long enough, you can expect that the descriptor for a new window will match one of the windows seen in the past.

43.3.2 Particle Video

The problem with interest points is that they may be quite sparse. A dense set of point tracks is useful. For example, if the tracks are accurate they provide



FIGURE 43.14: *TAPIR tracks query points in the first frame (***top** *two frames, from two distinct sequences), and can predict whether points have been occluded in a frame. Points with in the frames seen below correspond to points in the first frame with the same color. If the point is filled, then TAPIR predicts that it is visible; if empty, that it is invisible. When a point has a line segment attached, the far end of the segment is the ground truth, so a visible segment indicates the point is in the wrong place. A point indicated with an X has been identified as occluded in that frame. Figure constructed using parts of Figure 4 of "TAPIR: Tracking Any Point with per-frame Initialization and temporal Refinement", Doersch et al, 2023*

detailed information on how a surface is deforming. As another example, dense long term point tracks can be used to evaluate image enhancement algorithms. You track something in a sequence; then you enhance first and last frames, and see if corresponding points produce the same result.

One way to build dense point tracks is to put a set of point samples – perhaps, but not necessarily, a grid – in the first frame. Now move each point down the optic flow field for the first frame to get the corresponding point in the second frame. Continue moving points down the optic flow field. Reap points whenever the flow at a point appears untrustworthy, and insert new points into a frame if the set of points is not dense enough. The resulting set of tracks is known as a *particle video*. The detailed method of [] is now obsolete, but the underlying insight has been used to produce a number of point trackers.

43.3.3 TAPIR

As Section 36.3 pointed out, good flow models have three essential components: comprehensive comparisons of very distinctive local descriptors; coarse to fine estimation; and iterative refinement. These components yield very good point trackers as well. TAPIR (Track Any Point with Iterative Refinement) is one example.

TAPIR computes a comprehensive descriptor at each location in each frame using a convolutional encoder. The initial descriptor is computed on a coarse grid. The descriptor at each location is compared to that at every other location using a dot-product. This volume of data is then post-processed into a heatmap for each query point at each frame. The heatmap shows where the likely match is. The heatmap peaks are linked across time, using whatever dynamical model should apply. The matches using coarse features are then refined by comparing features computed on a finer grid in a range of locations around the coarse matches. The

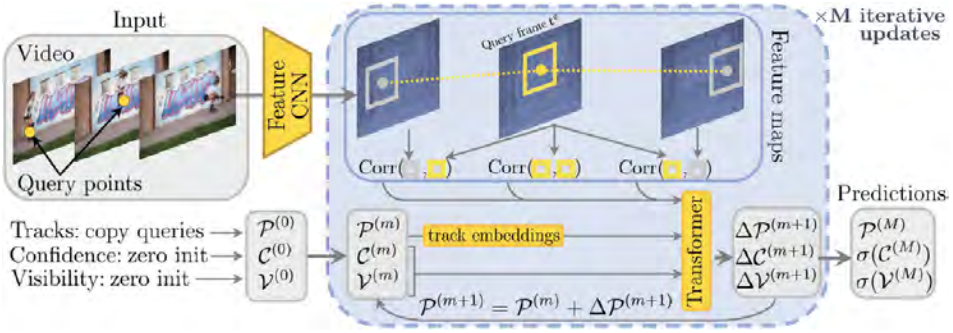


FIGURE 43.15: *CoTracker3 trains a transformer to update estimates of position (P), confidence (C), and visibility (V) for a query point at each frame. The transformer accepts $(\mathcal{P}^i, \mathcal{C}^i, \mathcal{V}^i)$ and image features for frame i, and produces an update to each to form these estimates for the next frame. The transformer is trained with labelled data, including synthetic data sets constructed with known point correspondences, and real datasets labelled by "teacher" tracker models. Figure constructed using parts of Figure 2 of "CoTracker3: Simpler and Better Point Tracking by Pseudo-Labelling Real Videos", Karaev et al, 2024*

tracks are refined using a learned predictor. The procedure is iterated to obtain localizations at a finer scale that are themselves more refined estimates.

TAPIR predicts a location for every query point in every frame of a sequence, but also predicts whether that point can be seen in that frame. TAPIR also predicts its confidence in each points location, using a score obtained from the matching process.

43.3.4 CoTracker3

CoTracker3 obtains significant improvements in accuracy by refining tracks using a transformer. For a query point at the 0'th frame, obtain its position ($\mathcal{P}^0$), its confidence ($\mathcal{C}^0$) and its visibility ($\mathcal{V}^0$). Now use a transformer to predict updates $(\Delta\mathcal{P}^1, \Delta\mathcal{C}^1, \Delta\mathcal{V}^1)$ so that ($\mathcal{P}^1 = \mathcal{P}^0 + \Delta\mathcal{P}^1$, and so on. Do so using features computed using a convolutional encoder and fed to the transformer.

Training a transformer requires a large volume of labelled data. Accurately labelling real data at the required scale is intractable. CoTracker3 uses synthetic data in part. Synthetic data is relatively easy to create – you render the locations of assorted points on surface models, and also render the images – but presents difficulties. There is no strong evidence that physically exact rendering (which is expensive) is required, but it isn't clear what phenomena need to be accurately represented in simulations. There is some evidence that unexpected biases in rendered data may create problems in point tracking. CoTracker3 uses other point tracking models applied to real sequences as teacher models. The underlying assumption, which is strongly supported by the results, is that errors will "average out" across teacher models.

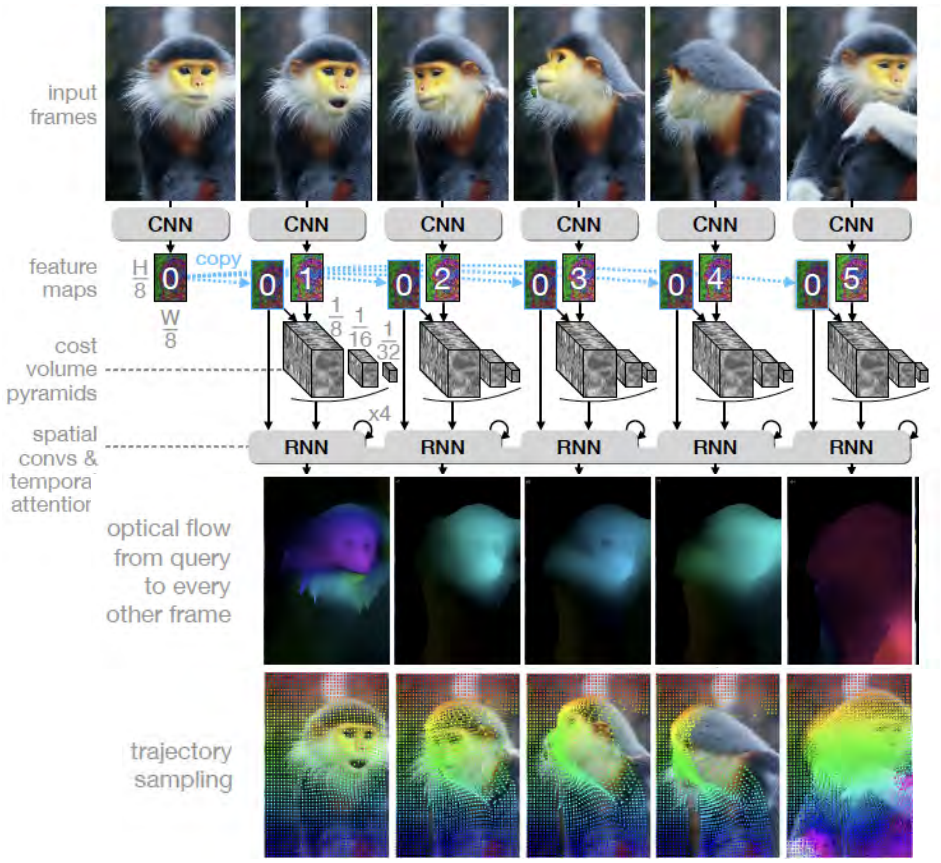


FIGURE 43.16: *Figure constructed using parts of Figure 2 of "AllTracker: Efficient Dense Point Tracking at High Resolution", Harley et al, 202X*

43.3.5 AllTracker

Point trackers track query points. These are typically identified in the first frame. On occasion, it is helpful to track query points obtained from a user in an interactive process. Alternatively, you could start with a grid of points. Ensuring that the set of tracked points remains dense requires some maintenance, because points that are visible in the first frame may disappear from view (the man in the blue and red striped shirt in Figure 43.15 occludes a number of points, for example). New points may appear, particularly if the camera pans or zooms out.

AllTracker tracks every point in a given frame in video, as an alternative to specifying query points. It accepts a tensor of $T \times H \times W \times 3$ representing a video (frames, height, width, and colorchannels) and a frame index which identifies the query frame. It produces a tensor of $T \times H \times W \times 4$. Two of the four values for each location in each frame give the offset that takes the pixel at that location in the query frame to the corresponding pixel in this frame. The other two give

whether the corresponding pixel is visible, and the confidence in the offset. You should think of AllTracker as making flow estimates for pairs of images that could be separated in time by a significant interval, where there may be quite large flows. As Section **??** shows, modern learned methods are extremely good at producing such flow fields. As you should expect, AllTracker proceeds from coarse to fine in space and time. It is trained entirely with synthetic data.

43.4 RESOURCES AND DATASETS

An open source version of TAPIR appears at `https://github.com/deepmind/tapnet`.

43.5 YOU SHOULD

43.5.1 remember these facts:

43.5.2 remember these procedures:

Tracking by detection links detector responses to abstract tracks. . . 797

43.5.3 be able to:

- Build a simple TbD application using a detector.

EXERCISES

QUICK CHECKS

43.1. Use Bayes' rule to construct $P(\mathbf{X}_0|\mathbf{Y}_0 = \mathbf{y}_0)$ from $P(\mathbf{X}_0)$ and $P(\mathbf{Y}_0 = \mathbf{y}_0|\mathbf{X}_0)$.

43.2. Check that

$$\int P(\mathbf{X}_i|\mathbf{X}_{i-1}, \mathbf{y}_0, \dots \mathbf{y}_{i-1})P(\mathbf{X}_{i-1}|\mathbf{y}_0, \dots, \mathbf{y}_{i-1})d\mathbf{X}_{i-1} = \int P(\mathbf{X}_i|\mathbf{X}_{i-1})P(\mathbf{X}_{i-1}|\mathbf{y}_0, \dots, \mathbf{y}_{i-1})d\mathbf{X}_{i-1}.$$

43.3. Show that

$$P(\mathbf{X}_i|\mathbf{y}_0, \dots, \mathbf{y}_i) \propto P(\mathbf{y}_i|\mathbf{X}_i)P(\mathbf{X}_i|\mathbf{y}_0, \dots, \mathbf{y}_{i-1})$$

43.4. Give a (reasonably practical) example of a case where the dynamical matrix $\mathcal{D}_i$ depends on the clock tick.

43.5. Show that if $x_{i-1} \sim N(\overline{\mathbf{x}}_{i-1}^-, d^2(\sigma_{i-1}^-)^2)$, then $d_i x_{i-1} \sim N(d_i\overline{\mathbf{x}}_{i-1}^-, d^2(\sigma_{i-1}^-)^2)$.

43.6. Show that if $x_{i-1} \sim N(\overline{\mathbf{x}}_{i-1}^-, d^2(\sigma_{i-1}^-)^2)$; and $x_i = d_i x_{i-1} + \xi$; and $\xi \sim N(0, \sigma_{n,i}^2)$; then

$$\overline{\mathbf{x}}_i^- = d_i\overline{\mathbf{x}}_{i-1}^- \text{ and } (\sigma_i^-)^2 = d^2(\sigma_{i-1}^-)^2 + \sigma_{n,i}^2.$$

43.7. Pattern match to the expression for a normal distribution to derive

$$\overline{\mathbf{x}}_i^+ = \left[\frac{m_i y_i(\sigma_i^-)^2 + \overline{\mathbf{x}}_i^-(\sigma_{m,i})^2}{m_i^2(\sigma_i^-)^2 + (\sigma_{m,i})^2}\right] \text{ and } (\sigma_i^+)^2 = \left[\frac{(\sigma_{m,i})^2(\sigma_i^-)^2}{m_i^2(\sigma_i^-)^2 + (\sigma_{m,i})^2}.\right]$$

43.8. Rearrange the result of the last exercise to find

$$x_i^+ = \overline{x}_i^- + k_i\left(y_i - m_i\overline{x}_i^-\right)$$

where

$$k_i = \frac{m_i(\sigma_i^-)^2}{m_i^2(\sigma_i^-)^2 + (\sigma_{m,i})^2}.$$

43.9. Check that if the measurement is wholly unreliable – so $\sigma_{m,i} >> \sigma_i^-$ – the weight k_i will be very small.

43.10. Further, check that if $\sigma_i^- >> \sigma_{m,i}$, the posterior mean will be very largely the measurement corrected for units.

43.11. Section 44.1.8 has: "The posterior is much better, because the filter received a velocity measurement." Explain.

43.12. Check that if

$$\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{n}) \text{ and } \mathbf{x}_i \sim N(\overline{\mathbf{x}}_i, \Sigma_{\mathbf{x}_i}),$$

$\mathbf{n}$ is noise, $\mathbf{n} \sim N(\mathbf{0}, \Sigma_{s,i})$, then linearizing with a Taylor series yields the approximation:

$$\mathbf{x}_{i+1} \sim N(\mathbf{f}(\overline{\mathbf{x}}_i), \mathcal{J}_{\mathbf{f},\mathbf{x}}\Sigma_{\mathbf{x}_i} + \mathcal{J}_{\mathbf{f},\mathbf{x}}^T + \mathcal{J}_{\mathbf{f},\mathbf{n}}\Sigma_{s,i}\mathcal{J}_{\mathbf{f},\mathbf{n}}^T)$$

43.13. Section 44.2.3 has the following expressions for prediction in an extended Kalman filter:

$$\overline{\mathbf{x}}_i^- = \mathbf{f}(\overline{\mathbf{x}}_{i-1}^+, \mathbf{0}) \text{ and } \Sigma_i^- = \mathcal{J}_{\mathbf{f},\mathbf{x}}\Sigma_{i-1}^+\mathcal{J}_{\mathbf{f},\mathbf{x}}^T + \mathcal{J}_{\mathbf{f},\mathbf{n}}\Sigma_{s,i}\mathcal{J}_{\mathbf{f},\mathbf{n}}^T.$$

Check this expression is correct.

43.14. Check that, for the EKF, $\mathcal{J}_{\mathbf{f},\mathbf{x}}$

LONGER PROBLEMS

43.15. This exercises shows that models of points moving with constant velocity or constant acceleration fit into the framework where $\mathbf{p}_i = \sum_{u=1}^{u=k} a_u \mathbf{p}_{i-u} +$ gaussian noise for some constants a_u. Write $\mathbf{p}$ for position, $\mathbf{v}$ for velocity and $\mathbf{a}$ for acceleration.

(a) In a constant velocity model, $\mathbf{v}_i \sim N(\mathbf{v}_{i-1}, \Sigma_{v,i-1})$. Show that this means that $\mathbf{v}_i \sim N(\overline{\mathbf{v}}_0, \sum_{u=0}^{u=i-1} \Sigma_{v,u})$.

(b) In a constant velocity model, $\mathbf{v}_i \sim N(\mathbf{v}_{i-1}, \Sigma_{v,i-1})$ and $\mathbf{x}_i \sim N(\mathbf{x}_{i-1} + \delta t \mathbf{v}_{i-1}, \Sigma_{x,i-1})$. Show that $\mathbf{x}_i \sim N(\mathbf{x}_{i-1} + \delta t \overline{\mathbf{v}}_0, \Sigma_{x,i-1} + \sum_{u=0}^{u=i-1} \Sigma_{v,u})$. Conclude that, for a given constant velocity model, there is an equivalent model in the framework where $\mathbf{p}_i = \sum_{u=1}^{u=k} a_u \mathbf{p}_{i-u} +$ gaussian noise.

(c) Repeat the previous construction for a constant acceleration model.

43.16. You have a non-linear function $\mathbf{h}$ of a random variable $\mathbf{x}$. The random variable is normal, so

$$\mathbf{x} \sim N(\mu_{\mathbf{x}}, \Sigma_{\mathbf{x}}).$$

The function is d-dimensional, and accepts a second argument $\mathbf{n}$. Here $\mathbf{n} \sim N(\mathbf{0}, \Sigma_{\mathbf{n},i})$. Write

$$\mathbf{u} = \mathbf{h}(\mathbf{x}, \mathbf{n}).$$

Write $\mathcal{J}_{\mathbf{h},\mathbf{x}}$ for the matrix of first partial derivatives of $\mathbf{h}$ with respect to the first argument, and $\mathcal{J}_{\mathbf{h},\mathbf{n}}$ for the matrix of first partial derivatives of $\mathbf{h}$ with respect to the second argument.

(a) Use a Taylor series to show that, for sufficiently small Σ_x and Σ_n, the approximation

$$\mathbf{u} \sim N(\mathbf{h}(\mu_{\mathbf{x}}, \mathbf{0}), \mathcal{J}_{\mathbf{h},\mathbf{x}} \Sigma_{\mathbf{x}} + \mathcal{J}_{\mathbf{h},\mathbf{x}}^T + \mathcal{J}_{\mathbf{h},\mathbf{n}} \Sigma_{\mathbf{n}} \mathcal{J}_{\mathbf{h},\mathbf{n}}^T).$$

is reasonable.

(b) The previous exercise uses "sufficiently small Σ_x and Σ_n", but is vague about what this means. Assume that each of these covariances is diagonal. What could go wrong with the approximation if a term on the diagonal was large?

PROGRAMMING EXERCISES

This exercise investigates the effects of non-linearity in dynamical models.

(c) Consider the dynamical model $x_i = x_{i-1} + \epsilon \sin(2\pi x_{i-1})$. There is no noise, and the model appears to be only "slightly" non-linear ($(x_i - x_{i-1})^2 \leq \epsilon^2$). For a range of intervals, $x_i > x_{i-1}$; for another range of intervals, $x_i < x_{i-1}$; and for a set of discrete points, $x_i = x_{i-1}$. Write down expressions for each of these three sets.

(d) Build a simulation of this model, where $x_0 \sim N(0, 9)$. Do this by drawing samples for x_0, then propagating them forward. Simulate until $i = 10$. What does the resulting distribution look like? How would you model it? You will find $\epsilon = 0.001$ and $\epsilon = 0.1$ informative.

(e) Now consider $x_i = x_{i-1} + \epsilon \sin(2\pi x_{i-1}) + \delta\xi$, where $\xi \sim N(0, 1)$. Simulate this model until $i = 10$ for various values of ϵ and δ. How big does δ have to be so that the resulting distribution looks normal? You will find $(\epsilon, \delta) = (0.1, 0.1)$ and $(\epsilon, \delta) = (0.1, 0.01)$ informative.