# MULTI-TARGET, MULTI-MODEL TRACKING

## 8.1: Introduction and a long example

- A common application of Kalman filtering is to attempt to localize a stationary or mobile target in some coordinate system.
- This target may be friendly (search and rescue operation) or hostile (military application), but in neither case do we know the driving input $u_k$ to the target dynamics

$$x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \qquad \text{but, } u_{k-1} \text{ is unknown}$$

$$z_k = h_k(x_k, u_k, v_k) \qquad \text{but, } u_k \text{ is unknown.}$$

- Therefore, we must treat $u_k$ as a random signal where, arguably, the Gaussian assumption breaks down.
- So, we often rely on very simple models based on physical first principles of lumped objects. *e.g.*, NCV by Newton's first law

$$\begin{bmatrix} \xi_k \\ \dot{\xi}_k \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \xi_{k-1} \\ \dot{\xi}_{k-1} \end{bmatrix} + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} w_{k-1}$$

$$z_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \xi_k \\ \dot{\xi}_k \end{bmatrix} + v_k.$$

- Our position/velocity estimate is not as accurate as it might be
  - The model does not precisely describe the dynamics of the system being tracked;

  - We don't know $u_k$.

- But, covariance still has a geometric interpretation (hyper-ellipsoid) that is very helpful in pin-pointing geographic uncertainty of the object being tracked, and provides hints as to where to point sensors to look for the object at the next time step.

## Extended illustrative example

- Consider an infrared camera mounted in a ball gimbal on a military aircraft. We wish to detect and track missiles fired at our aircraft.[1]
- In each image where a missile is present, we want to compute:
  1. Our best estimate of the missile position in frame $k$ given $k$ observations, so that we have the best possible battlefield situational awareness.
  2. Our best estimate of where the missile position will be in frame $k+1$ given $k$ observations, so we can point the ball gimbal to maintain target lock.
- Assumptions
  1. We acquire the target when it is still far away. It will appear as a single very bright pixel, perhaps surrounded by several relatively dimmer pixels that are somewhat brighter than the background.
  2. The frame rate of our camera is very fast compared to the kinematics of the target. Thus, the target is capable of only negligible accelerations from frame to frame.

[1] Example adapted from J.P. Havlicek's course notes for *ECE5283: Kalman Filtering*, Ch. 5, University of Oklahoma, 2003.

- In general, we may wish to simultaneously track several different targets, maintaining independent "target tracks":
  - We will use a separate Kalman filter for each target.
  - We will mention different ways to associate measurements to targets (gating and different methods of data association).

## *Target extraction*

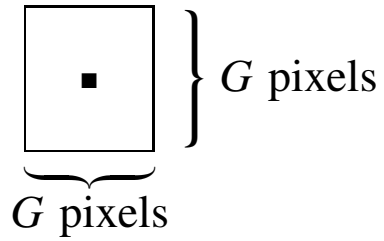
- We will first apply a nonlinear filter to detect the target(s).
- The main idea is this: At each pixel, formulate an estimate of the background. Subtract the background estimate from the pixel.

| | | |
|---|---|---|
| $I_{i-1,j-1}$ | $I_{i,j-1}$ | $I_{i-1,j+1}$ |
| $I_{i-1,j}$ | $I_{i,j}$ | $I_{i+1,j}$ |
| $I_{i-1,j+1}$ | $I_{i,j+1}$ | $I_{i+1,j+1}$ |

- Background estimate $=$ median of eight nearest neighbors $= M_{i,j}$.
- Filter output $Y_{i,j} = I_{i,j} - M_{i,j}$.
- The median-filtered image is then thresholded.
  - We could use an absolute threshold, a relative threshold (based, *e.g.*, on the variance of the neighborhood), or both.
  - Pixels that pass the threshold are called exceedances. They are candidate targets, denoted $E_{i,j}$.
- Each time we receive a new frame, we do two things:
  1. Associate exceedances with existing tracks and use the associated exceedances as observations to update existing tracks.

2. Consider starting new tracks on any exceedances that do not associate with already existing tracks.

*Gating*




- A "track gate" is a rectangular window centered on the estimated position of an existing track or a candidate new track.
- Let $G$ be the width and height of the gate.
- Typically, the gate size is varied during tracking.
- We will stipulate minimum and maximum values for $G$.
- Typically, for a new track, $G$ starts out at $G_{\max}$.
- Why vary the gate size?
  - For a candidate new track, our knowledge of the true track position is typically poor. So, we want a large, or "loose" gate to be sure we pick up all the exceedances associated with the track.
  - For an existing track, if the measurements have not been agreeing well with the predictions, we also want a large gate for the same reasons as above.
  - For an existing track where the measurements and predictions have been in good agreement, however, we want the gate to be small, or "tight."
  - Since only exceedances within the gate will be included in the observed threshold calculation, a tight gate means less noise in the centroid calculation.

- Note: Henceforth, we consider only tracking in the horizontal direction. Vertical tracking is analogous.

*Gating and association: Update of existing tracks*

- For each existing track, we have a predicted position $\hat{\xi}_k^- = A\hat{\xi}_{k-1}^+$.
- When frame $k$ arrives, we apply the spatial filter and threshold(s) to detect exceedances.
- For each track, we place the track gate at the predicted location $\hat{\xi}_k^-$ and compute the centroid of the exceedances in the gate. This centroid is the observation $z_k$ for the track.
- The measurement noise arises from the discrete nature of the camera focal plane array detector, thermal noise, and imperfections in the gating and target extraction processing.
- Centroid calculation

$$z_k = \frac{\sum_{m,n\in\text{Gate}} m E_{m,n}}{\sum_{m,n\in\text{Gate}} E_{m,n}}$$

  where

$$E_{i,j} = \begin{cases} Y_{i,j}, & Y_{i,j} \geq \text{threshold}; \\ 0, & \text{else.} \end{cases}$$

- Note, the standard deviation of the exceedances in the gate is also sometimes used in the calculation of $G$, the gate size.
- For each existing track, its $z_k$ is used to update the Kalman filter.

*New track starts*

- In any given frame, there may be exceedances that do not fall near the gate of any existing track (*i.e.*, do not "associate.")

- A default sized gate is placed around such exceedances and they become candidates for new tracks.
- Since spurious bright spots may occur due to reflections, sun glints, and so forth, we don't start a new track right away.
- Instead, we require the candidate track to persist for some small number of frames before we actually start a new track. This is called the "persistence test" or the "temporal correspondence test."
- For the NCV model, we use the most recent two measurements to initialize a track, as we discussed in chapter 4 of the notes

$$\Sigma_{\tilde{x},0}^{+} = \begin{bmatrix} \Sigma_{\tilde{v}} & \dfrac{\Sigma_{\tilde{v}}}{T} \\ \dfrac{\Sigma_{\tilde{v}}}{T} & \dfrac{2\Sigma_{\tilde{v}} + \Sigma_{\tilde{w}} T^4/4}{T^2} \end{bmatrix}$$

$$\hat{x}_0 = \begin{bmatrix} z_0 \\ \dfrac{z_0 - z_{-1}}{T} \end{bmatrix}.$$

*Track coasting*

- Sometimes a target is temporarily obscured.
- In this case, there will be no exceedances in the track gate of the existing track.
- When this occurs, we need to "coast" the track.
- This involves two steps:
  1. Open the track gate up to the maximum size.
  2. Allow the state equation to evolve with no input.
- If the number of missed detections exceeds a set limit, then the track is terminated. This is called "track loss" or "track deletion."

- It may be helpful to restart the Kalman filter upon reacquisition.

*Track crossings*

- Sometimes two targets will cross one another.
- When this happens, the track gates overlap.
- In this case, it may not be possible to resolve the individual targets.
- The best thing we can do is coast both tracks and hope to reacquire the targets when they separate.
- It may be helpful to restart the Kalman gains upon reacquisition.

*Track merge*

- Sometimes two tracks will cross and never come apart again.
- In this case, we merge the tracks.
- The simplest way to do this is to keep the track that best agrees with the observations and delete the other track.
- This is called a “track merge.”
- It should be noted in the history of the deleted track that it was deleted due to a merge. Later offline analysis of both tracks may reveal insight into what actually happened.

## 8.2: Data association

- When tracking multiple targets, there is ambiguity as to which measurement corresponds to which target track.
- Making associations between measured target positions and target tracks is generally known as the data association problem.
- Associations are generally made according to some distance function between the predicted state of the target and the measured state.
- Note that distance is not limited to position; it may also consist of features such as color. This becomes especially important for targets that may come close to or cross one another.
- Popular association algorithms for single-target-at-a-time applications are based on the following schemes:
  - Nearest Neighbor: This algorithm always updates the tracking filter with the measurement closest to the predicted state.
  - Multi-Hypothesis Track Splitting: This scheme creates a new hypothesis track for every measurement that is in the validation region, and prunes unlikely tracks using a likelihood ratio.
  - Probabilistic Data Association: Each measurement affects the tracking filter to a degree based on the probability that it is the correct given the predicted state.
  - Optimal Bayesian Filter: This variation of Probabilistic Data Association splits multiple tracks, like the Multi-Hypothesis algorithm, and eliminates unlikely tracks.
- For multiple sensors and multiple targets, the problem becomes increasingly complex. Common association algorithms are:

- Joint Likelihood: This variation on the Multi-Hypothesis Track Splitting algorithm above extends to multiple tracks.
- Joint Probabilistic Data Association: This algorithm updates the filter for each track based on a joint probability of association between the latest set of measurements and each track.
- Multiple Hypothesis Joint Probabilistic: This variation of the Optimal Bayesian Filter uses joint probabilities among multiple track associations for multiple hypotheses. It is by far the most computationally complex algorithm, and requires intelligent pruning techniques. It is NP-complete, which provides considerable incentive to find non-exhaustive ways to search the space of possible associations to maximize the joint probability.

■ Sadly, these methods are beyond the scope of our class.

## 8.3: Tracking with polar measurements and a Cartesian state

- It is often most convenient to express the state dynamics of a target in Cartesian coordinates.
- However, in active sonar and radar systems, the measurement is returned in polar coordinates.
- To handle this case, we can either
  1. Use a nonlinear KF (with output equation in polar coordinates), or
  2. Compute an "equivalent" measurement in Cartesian coordinates from the true measurement in polar coordinates.
- Here, we handle the second case, which would be trivial except for the noises involved.[2]
- We measure (where $\mathbb{E}[\tilde{r}] = 0$, $\mathbb{E}[\tilde{\theta}] = 0$, and $\tilde{r}$ and $\tilde{\theta}$ are uncorrelated)

$$r_m = r + \tilde{r} \qquad \text{and} \qquad \theta_m = \theta + \tilde{\theta}.$$

- A standard conversion to Cartesian coordinates gives

$$x_m = r_m \cos\theta_m \qquad \text{and} \qquad y_m = r_m \sin\theta_m.$$

- The errors in each coordinate can be found by expanding

$$x_m = x + \tilde{x} = (r + \tilde{r})\cos(\theta + \tilde{\theta})$$

$$y_m = y + \tilde{y} = (r + \tilde{r})\sin(\theta + \tilde{\theta}).$$

- Using trigonometric identities, we obtain

$$\tilde{x} = r\cos\theta(\cos\tilde{\theta} - 1) - \tilde{r}\sin\theta\sin\tilde{\theta} - r\sin\theta\sin\tilde{\theta} + \tilde{r}\cos\theta\cos\tilde{\theta}$$

[2] From: Lerro, D. and Bar-Shalom, Y., "Tracking with debiased consistent converted measurements versus EKF" *IEEE Trans. Aerospace and Electronic Systems,* 29(3), July 1993, 1015–22.

$$\tilde{y} = r\sin\theta(\cos\tilde{\theta} - 1) + \tilde{r}\cos\theta\sin\tilde{\theta} + r\cos\theta\sin\tilde{\theta} + \tilde{r}\sin\theta\cos\tilde{\theta},$$

which are not independent, and each coordinate error depends on the true range and bearing as well as the errors in range and bearing.

- The mean and variance of the errors can be found by assuming zero-mean Gaussian errors in the polar measurements. In this case

$$\mathbb{E}[\cos\tilde{\theta}] = e^{-\sigma_\theta^2/2}, \qquad \mathbb{E}[\sin\tilde{\theta}] = 0, \qquad \mathbb{E}[\sin\tilde{\theta}\cos\tilde{\theta}] = 0;$$

$$\mathbb{E}[\cos^2\tilde{\theta}] = \frac{1 + e^{-2\sigma_\theta^2}}{2}, \qquad \mathbb{E}[\sin^2\tilde{\theta}] = \frac{1 - e^{-2\sigma_\theta^2}}{2}.$$

- The true mean error of the $(x_m, y_m)$ conversion is then

$$\mu_t(r,\theta) = \begin{bmatrix} \mathbb{E}[\tilde{x} \mid r,\theta] \\ \mathbb{E}[\tilde{y} \mid r,\theta] \end{bmatrix} = \begin{bmatrix} r\cos\theta(e^{-\sigma_\theta^2/2} - 1) \\ r\sin\theta(e^{-\sigma_\theta^2/2} - 1) \end{bmatrix}.$$

- The true values of the elements of the converted measurement covariance are (much unpleasant trig manipulation omitted)

$$\begin{aligned}
\Sigma_{\tilde{v},t}^{11} &= \mathrm{var}(\tilde{x} \mid r,\theta) \\
&= r^2 e^{-\sigma_\theta^2}[\cos^2\theta(\cosh(\sigma_\theta^2) - 1) + \sin^2\theta\sinh(\sigma_\theta^2)] \\
&\quad + \sigma_r^2 e^{-\sigma_\theta^2}[\cos^2\theta\cosh(\sigma_\theta^2) + \sin^2\theta\sinh(\sigma_\theta^2)] \\
\Sigma_{\tilde{v},t}^{22} &= \mathrm{var}(\tilde{y} \mid r,\theta) \\
&= r^2 e^{-\sigma_\theta^2}[\sin^2\theta(\cosh(\sigma_\theta^2) - 1) + \cos^2\theta\sinh(\sigma_\theta^2)] \\
&\quad + \sigma_r^2 e^{-\sigma_\theta^2}[\sin^2\theta\cosh(\sigma_\theta^2) + \cos^2\theta\sinh(\sigma_\theta^2)] \\
\Sigma_{\tilde{v},t}^{12} &= \mathrm{cov}(\tilde{x}, \tilde{y} \mid r,\theta) \\
&= \sin\theta\cos\theta e^{-2\sigma_\theta^2}[\sigma_r^2 + r^2(1 - e^{\sigma_\theta^2})].
\end{aligned}$$

- Ideally, we would subtract the mean error from the conversion, and use the $\Sigma_{\tilde{v},t}$ matrices in the measurement update.

- However, they are uncomputable since we do not know $r$ and $\theta$ (we know only $r_m$ and $\theta_m$).
- We can, however, compute the average true bias and the average true covariance.
- The average true bias is (unpleasant trig again omitted)

$$\mathbb{E}[\mu_t(r,\theta) \mid r_m, \theta_m] = \begin{bmatrix} r_m \cos\theta_m (e^{-\sigma_\theta^2} - e^{-\sigma_\theta^2/2}) \\ r_m \sin\theta_m (e^{-\sigma_\theta^2} - e^{-\sigma_\theta^2/2}) \end{bmatrix}.$$

- The average true covariance matrix has elements

$$\begin{aligned}
\Sigma_{\tilde{v},a}^{11} &= r_m^2 e^{-2\sigma_\theta^2}[\cos^2\theta_m(\cosh 2\sigma_\theta^2 - \cosh\sigma_\theta^2) + \sin^2\theta_m(\sinh 2\sigma_\theta^2 - \sinh\sigma_\theta^2)] \\
&\quad + \sigma_r^2 e^{-2\sigma_\theta^2}[\cos^2\theta_m(2\cosh 2\sigma_\theta^2 - \cosh\sigma_\theta^2) \\
&\quad + \sin^2\theta_m(2\sinh 2\sigma_\theta^2 - \sinh\sigma_\theta^2)] \\
\Sigma_{\tilde{v},a}^{22} &= r_m^2 e^{-2\sigma_\theta^2}[\sin^2\theta_m(\cosh 2\sigma_\theta^2 - \cosh\sigma_\theta^2) + \cos^2\theta_m(\sinh 2\sigma_\theta^2 - \sinh\sigma_\theta^2)] \\
&\quad + \sigma_r^2 e^{-2\sigma_\theta^2}[\sin^2\theta_m(2\cosh 2\sigma_\theta^2 - \cosh\sigma_\theta^2) \\
&\quad + \cos^2\theta_m(2\sinh 2\sigma_\theta^2 - \sinh\sigma_\theta^2)] \\
\Sigma_{\tilde{v},a}^{12} &= \sin\theta_m\cos\theta_m e^{-4\sigma_\theta^2}[\sigma_r^2 + (r_m^2 + \sigma_r^2)(1 - e^{\sigma_\theta^2})].
\end{aligned}$$

- Note that the average covariance is *larger* than the true covariance conditioned on the exact position as it takes into account the additional errors incurred by evaluating it at the measured position.
- Thus, the final polar-to-Cartesian unbiased consistent conversion which corrects for the average bias is

$$z = \begin{bmatrix} r_m \cos\theta_m (1 - e^{-\sigma_\theta^2} + e^{-\sigma_\theta^2/2}) \\ r_m \sin\theta_m (1 - e^{-\sigma_\theta^2} + e^{-\sigma_\theta^2/2}) \end{bmatrix},$$

and

$$\Sigma_{\tilde{v}} = \Sigma_{\tilde{v},a} = \begin{bmatrix} \Sigma_{\tilde{v},a}^{11} & \Sigma_{\tilde{v},a}^{12} \\ \Sigma_{\tilde{v},a}^{12} & \Sigma_{\tilde{v},a}^{22} \end{bmatrix}.$$

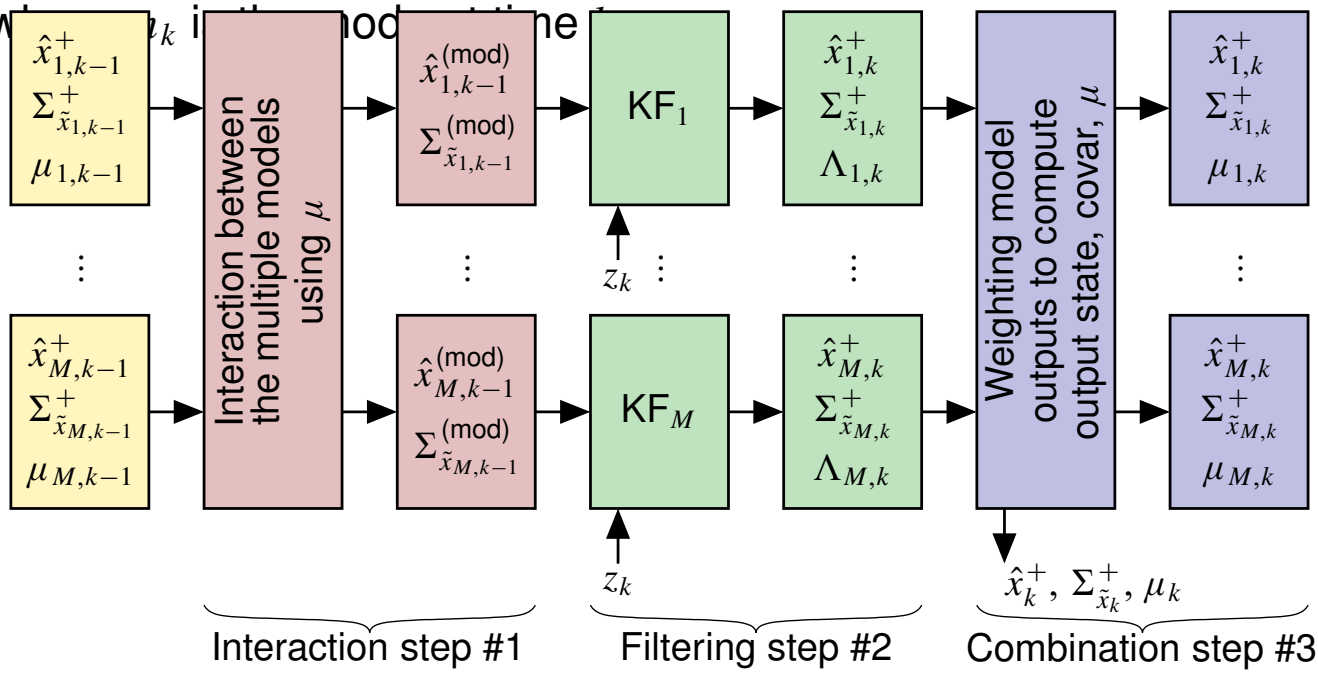
- Note that the corrections to $\Sigma_{\tilde{v}}$ are time varying, so must be computed for every measurement.
- Ironically, a true nonlinear Kalman filter may be less complex than applying these corrections to a linear Kalman filter!

## 8.4: The interacting-multiple-model Kalman filter

- A target that we wish to track may have a number of significantly distinct modes of operation, numbered $1 \ldots M$.
- For example, a target may act like NCV for a period of time, then NCP for another period of time, then follow a coordinated turn, etc.
- The IMM-KF operates multiple Kalman filters in parallel, and carefully blends state and covariance from each filter to make a composite state estimate and covariance.[3]
- The IMM-KF also computes a PMF corresponding to the filter's estimate of the likelihood that the target is operating in modes $1 \ldots M$ .
- An additional input required by IMM is a matrix containing the probability of transitioning from mode $i$ to mode $j$:

$$p_{ij} = \Pr(m_k = j \mid m_{k-1} = i),$$

where $\mu_k$ is the mode at time $k$




[3] A good external reference is: Mazor, E., Averbuch, A., Bar-Shalom, Y., and Dayan, J., "Interacting multiple model methods in target tracking: A survey," *IEEE Trans. Aerospace and Electronic Systems,* 34(1), Jan. 1998, 103–123.

- The IMM repeatedly executes three steps per iteration: interaction, filtering, and combination. This is shown in the diagram:
  - Step 1 is the interaction between the filters;
  - Step 2 is the individual filter update; and
  - Step 3 combines filter information to create the output information.

*Interaction*

- The estimates and covariances from the $M$ independent KFs (from the prior time step) are blended together to produce the inputs to the $M$ independent KFs for this time step.
- We first compute the conditional probability

$$\mu_{i|j,k-1} = \Pr(m_{k-1} = i \mid m_k = j, \mathbb{Z}_{k-1})$$

- Note that a version of Bayes' rule that applies is:

$$\Pr(A \mid B, C) = \frac{\Pr(B \mid A, C)\Pr(A \mid C)}{\Pr(B \mid C)}.$$

- So, we have

$$\begin{aligned}\mu_{i|j,k-1} &= \Pr(m_{k-1} = i \mid m_k = j, \mathbb{Z}_{k-1}) \\ &= \frac{\Pr(m_k = j \mid m_{k-1} = i, \mathbb{Z}_{k-1})\Pr(m_{k-1} = i \mid \mathbb{Z}_{k-1})}{\Pr(m_k = j \mid \mathbb{Z}_{k-1})} \\ &= \frac{1}{\bar{c}_j} p_{ij} \Pr(m_{k-1} = i \mid \mathbb{Z}_{k-1}) = \frac{1}{\bar{c}_j} p_{ij}\mu_{i,k-1},\end{aligned}$$

where $\bar{c}_j = \Pr(m_k = j \mid \mathbb{Z}_{k-1}) = \sum_{i=1}^{M} p_{ij}\mu_{i,k-1}$.

- Then we blend together all state outputs from the prior step according to the probability that they could have contributed to the state at this time step.
- The modified input state to filter $j$ at time step $k$ is

$$\begin{aligned}\hat{x}_{j,k-1}^{(\text{mod})} &= \mathbb{E}\left[x_{k-1} \mid m_k = j, \mathbb{Z}_{k-1}\right] \\ &= \sum_{i=1}^{M} \hat{x}_{i,k-1}^{+} \Pr(m_{k-1} = i \mid m_k = j, \mathbb{Z}_{k-1}) \\ &= \sum_{i=1}^{M} \hat{x}_{i,k-1}^{+} \mu_{i|j,k-1}.\end{aligned}$$

- The covariance of $\hat{x}_{j,k-1}^{(\text{mod})}$ can be found as

$$\begin{aligned}\Sigma_{\tilde{x}_{j,k-1}}^{(\text{mod})} &= \mathbb{E}\left[(x_{k-1} - \hat{x}_{j,k-1}^{(\text{mod})})(x_{k-1} - \hat{x}_{j,k-1}^{(\text{mod})})^T\right] \\ &= \sum_{i=1}^{M} \mathbb{E}\left[(x_{k-1} - \hat{x}_{j,k-1}^{(\text{mod})})(x_{k-1} - \hat{x}_{j,k-1}^{(\text{mod})})^T \mid m_k = j\right] \mu_{i|j,k-1} \\ &= \sum_{i=1}^{M} \mathbb{E}\Big[(x_{k-1} - \hat{x}_{i,k-1}^{+} + \hat{x}_{i,k-1}^{+} - \hat{x}_{j,k-1}^{(\text{mod})}) \times \\ &\qquad (x_{k-1} - \hat{x}_{i,k-1}^{+} + \hat{x}_{i,k-1}^{+} - \hat{x}_{j,k-1}^{(\text{mod})})^T \mid m_k = j\Big] \mu_{i|j,k-1} \\ &= \sum_{i=1}^{M} \mathbb{E}\left[(x_{k-1} - \hat{x}_{i,k-1}^{+})(x_{k-1} - \hat{x}_{i,k-1}^{+})^T \mid m_k = j\right] \mu_{i|j,k-1} \\ &\qquad + \sum_{i=1}^{M} (\hat{x}_{i,k-1}^{+} - \hat{x}_{j,k-1}^{(\text{mod})}) \underbrace{\mathbb{E}\left[x_{k-1} - \hat{x}_{i,k-1}^{+} \mid m_k = j\right]}_{0} \mu_{i|j,k-1}\end{aligned}$$

$$+\sum_{i=1}^{M}\underbrace{\mathbb{E}\left[x_{k-1}-\hat{x}_{i,k-1}^{+}\mid m_k=j\right]}_{0}(\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})})^T\mu_{i|j,k-1}$$

$$+\sum_{i=1}^{M}(\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})})(\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})})^T\mu_{i|j,k-1}$$

$$=\sum_{i=1}^{M}\mathbb{E}\left[(x_{k-1}-\hat{x}_{i,k-1}^{+})(x_{k-1}-\hat{x}_{i,k-1}^{+})^T\mid m_k=j\right]\mu_{i|j,k-1}$$

$$+\sum_{i=1}^{M}[\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})}][\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})}]^T\mu_{i|j,k-1}$$

$$=\sum_{i=1}^{M}\left\{\Sigma_{\tilde{x}_{i,k-1}}^{+}+[\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})}][\hat{x}_{i,k-1}^{+}-\hat{x}_{j,k-1}^{(\text{mod})}]^T\right\}\mu_{i|j,k-1}.$$

- The first term is the mixture of the prior covariances, and the second term is due to the "spread of the means" of the individual filters.

*Filtering*

- After the interaction step, one time step for each of the $M$ Kalman filters is then executed.
- Each filter outputs the present state and covariance estimate for that mode, as well as the likelihood of the present measurement given that the target is in mode $j$:

$$\Lambda_{j,k}=\mathcal{N}(z_k-\hat{z}_{j,k},\,\Sigma_{\tilde{z}_{j,k}})$$

$$=\frac{1}{(2\pi)^{n/2}\left|\Sigma_{\tilde{z}_{j,k}}\right|^{1/2}}\exp\left(-\frac{1}{2}(z_k-\hat{z}_{j,k})^T\Sigma_{\tilde{z}_{j,k}}^{-1}(z_k-\hat{z}_{j,k})\right).$$

## *Combination*

- The final step is to combine the results of the $M$ filters to get an overall target state estimate and a probability distribution on the mode random variable.
- We first compute the *a posteriori* probability of being in mode $j$ at time $k$

$$\begin{aligned}
\mu_{j,k} &= \Pr(m_k = j \mid \mathbb{Z}_k) = \Pr(m_k = j \mid \mathbb{Z}_{k-1}, z_k) \\
&= \frac{f(z_k \mid m_k = j, \mathbb{Z}_{k-1}) \Pr(m_k = j \mid \mathbb{Z}_{k-1})}{\Pr(z_k)} \\
&= \frac{1}{c}\Lambda_{j,k} \sum_{i=1}^{M} \Pr(m_k = j \mid m_{k-1} = i, \mathbb{Z}_{k-1}) \Pr(m_{k-1} = i \mid \mathbb{Z}_{k-1}) \\
&= \frac{1}{c}\Lambda_{j,k} \sum_{i=1}^{M} p_{ij}\mu_{i,k-1} \\
&= \frac{1}{c}\Lambda_{j,k}\bar{c}_j,
\end{aligned}$$

  where $c$ is a normalizing constant to ensure that $\mu_{j,k}$ sums to 1.
- Then, the filter output is computed as a mixture of the individual filter estimates, according to the likelihood of the target being in the mode of that filter

$$\hat{x}_k^+ = \sum_{j=1}^{M} \hat{x}_{j,k}^+ \mu_{j,k}$$

$$\Sigma_{\tilde{x},k}^+ = \sum_{j=1}^{M} \left\{ \Sigma_{\tilde{x}_{j,k}}^+ + [\hat{x}_{j,k}^+ - \hat{x}_k^+][\hat{x}_{j,k}^+ - \hat{x}_k^+]^T \right\} \mu_{j,k}.$$

- Then, the state is computed as a weighted combination of the individual filter state estimates.

- Likewise, the covariance is computed as a weighted combination of the individual filter covariance estimates.

## 8.5: Code for IMM

- The code in this section implements one iteration of the IMM.
- The inputs comprise the present system input `u` $= u_k$ (which is often zero), the present measurement `z` $= z_k$, `immState`, and `immData`.
- The `immData` variable has fields:
  - `txprob`: This is the state transition matrix $p_{ij}$.
  - `A`, `B`, `C`, `D`: The system's discrete-time $A$, $B$, $C$, and $D$ matrices, in a three-dimensional array, with the third dimension being the mode.
  - `Sw`, `Sv`: The system's discrete time noise matrices $\Sigma_{\tilde{w}}$ and $\Sigma_{\tilde{v}}$, in a three-dimensional array, with the third dimension being the mode.
- The `immState` variable has fields:
  - `mode`: An $M \times 1$ vector comprising the probability of being in any particular mode at this point in time. Corresponds to $\mu_{i,k}$ for all values of $i \in \{1 \cdots M\}$ at this time step $k$.
  - `X`: An $n \times M$ matrix comprising the state estimate of each KF. Each column of `X` contains one KF state estimate.
  - `SX`: An $n^2 \times M$ matrix comprising the covariance matrices of each KF. Each column contains one covariance matrix in columnar form.
  - `xhat`: The output combined state estimate $\hat{x}_k^+$ from the filter for this time step.
  - `SigmaX`: The output combined covariance estimate $\Sigma_{\tilde{x},k}^+$ from the filter for this time step.
- Some of the code is pretty tricky and bears closer examination.
- Some very efficient MATLAB matrix operations replace loops.

```
function immState = iterIMM(uold,unew,z,immState,immData)
  modes = size(immData.A,3);
  nx = size(immData.A,1);

  % Interaction step
  % 1) Compute cbar = sum(p(i,j)*mu(i,k-1))
  cbar = immData.txprob'*immState.mode;
  % 2) Compute mu(i|j,k-1) = 1/cbar * p(i,j)*mu(i,k-1)
  modeij = immData.txprob.*(immState.mode*(cbar.^(-1))');
  modeij(isnan(modeij)) = 0; % take care of impossible final states
  % 3) Compute xhat(mod)
  x0 = immState.X*modeij;
  % 4) Compute Sigma(mod). [Fixed error in Bar-Shalom IMM code here]
  Sx0 = immState.SX; Sx1 = immState.SX; % reserve space for Sx0,Sx1
  for j = 1:modes,
    xk1 = immState.X(:,j); Sx = xk1*xk1'; Sx0(:,j) = Sx(:);
    xk1 = x0(:,j); Sx = xk1*xk1'; Sx1(:,j) = Sx(:);
  end
  Sx0 = (immState.SX + Sx0)*modeij - Sx1; % verified

  % Filtering step
  Lambda = zeros(size(cbar)); Sx = zeros(nx);
  for j = 1:modes,
    % 0) Set up variables for this filter
    xhat = x0(:,j); Sx(:) = Sx0(:,j);
    A = immData.A(:,:,j); B = immData.B(:,:,j);
    C = immData.C(:,:,j); D = immData.D(:,:,j);
    Sw = immData.Sw(:,:,j); Sv = immData.Sv(:,:,j);
    % 1a) State estimate time update
    xhat = A*xhat + B*uold;
    % 1b) State covariance time update
    Sx = A*Sx*A' + Sw;
    % 1c) Output estimate
    zhat = C*xhat + D*unew;
    % 2a) Filter gain matrix
    Py = (C*Sx*C'+Sv); L = Sx*C'/Py;
    % 2b) State estimate measurement update
    immState.X(:,j) = xhat + L*(z-zhat);
    % 2c) State covariance measurement update
    Sx = Sx - L*Py*L'; immState.SX(:,j) = Sx(:);
    Lambda(j) = max(1e-9,exp(-0.5 * ((z-zhat)^2/Py)) / ...
        sqrt(2*pi*Py)); % much faster than normpdf
```

```
  end

  % Combination step
  % 1) Compute PMF of being in mode j: mu(j,k)
  immState.mode = Lambda.*cbar;
  immState.mode = immState.mode/sum(immState.mode);
  % 2) Compute composite state estimate
  immState.xhat = immState.X*immState.mode;
  % 3) Compute composite covariance estimate
  modeSx = immState.SX;
  for j=1:modes
    xk1=immState.X(:,j)-immState.xhat;
    Sx=xk1*xk1'; modeSx(:,j)=Sx(:);
  end
  immState.SigmaX(:) = (immState.SX+modeSx)*immState.mode;
```

- The following is example driver code for this subroutine:
  - A one-dimensional tracking case is considered
  - The true system uses an NCV model for 100 iterations, then an NCP model for 100 iterations.

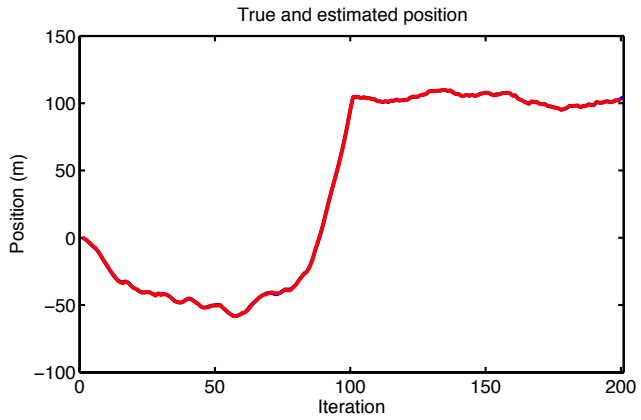
```
% Set up model 1 for mode 1: NCV model, one dimensional, T=1
% Start with continuous-time; convert to discrete-time
Ac = [0 1; 0 0]; Bc = [0; 1]; Swc = 1;
Z = [-Ac Bc*Swc*Bc'; zeros(size(Ac)) Ac'];
C = expm(Z*1); A1 = C(3:4,3:4)'; Sw1 = A1*C(1:2,3:4);
B1 = [1^2/2; 1]; C1 = [1 0]; Sv1 = 0.2;

% Set up model 2 for mode 2: NCP model, one dimensional, T=1
% Start with continuous-time; convert to discrete-time
Ac = [0 0; 0 0]; Bc = [1; 0]; Swc = 1;
Z = [-Ac Bc*Swc*Bc'; zeros(size(Ac)) Ac'];
C = expm(Z*1); A2 = C(3:4,3:4)'; Sw2 = A2*C(1:2,3:4);
B2 = [1; 0]; C2 = [1 0 ]; Sv2 = 0.1;

% Populate the immData and immState structures
immData = [];
immData.txprob = [0.95 0.05; 0.05 0.95];
immData.A(:,:,1) = A1; immData.A(:,:,2) = A2;
immData.B(:,:,1) = B1; immData.B(:,:,2) = B2;
```

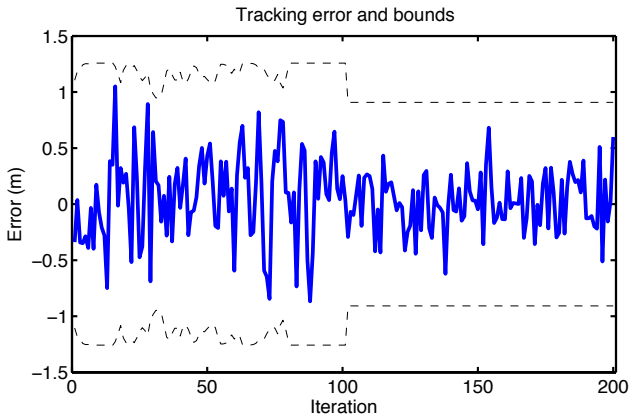
 

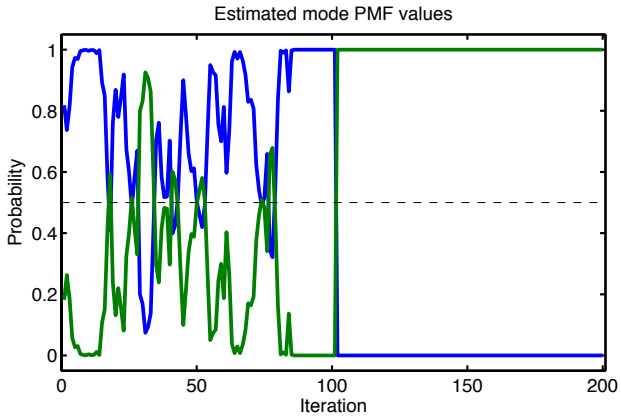
```
immData.C(:,:,1) = C1; immData.C(:,:,2) = C2;
immData.D(:,:,1) = 0;  immData.D(:,:,2) = 0;
immData.Sw(:,:,1) = Sw1; immData.Sw(:,:,2) = Sw2;
immData.Sv(:,:,1) = Sv1; immData.Sv(:,:,2) = Sv2;

immState = [];
immState.mode = [0.8; 0.2]; % a-priori likelihood for each mode
immState.X = [0 0; 0.3 0.3]; % initialize both modes to same estimate
immState.SX = [0.1 0.1; 0 0; 0 0; 0.1 0.1];
immState.xhat = [0; 0]; % these two are outputs, not inputs, but I'll set
immState.SigmaX = [0.1 0; 0 0.1]; % them anyway

% Generate the true system data
xtrue = zeros([2,201]);
z = zeros(1,200);
for k = 1:100,
  xtrue(:,k+1) = A1*xtrue(:,k) + chol(Sw1,'lower')*randn([2 1]);
  z(k) = C1*xtrue(:,k) + sqrt(Sv1)*randn(1);
end
for k = 101:200,
  xtrue(:,k+1) = A2*xtrue(:,k) + sqrt(Sw2)*randn([2 1]);
  z(k) = C2*xtrue(:,k) + sqrt(Sv2)*randn(1);
end
figure(1); clf; plot(xtrue(1,:)); hold on;

% Run the IMM on this data
xhatstore = zeros([2,200]);
modestore = zeros([2,200]);
sigmastore = zeros([4,200]);
for k = 1:200,
  immState = iterIMM(0,0,z(k),immState,immData);
  xhatstore(:,k) = immState.xhat;
  sigmastore(:,k) = immState.SigmaX(:);
  modestore(:,k) = immState.mode;
end
hold on; plot(xhatstore(1,:),'r');
title('True and estimated position'); xlabel('Iteration');
ylabel('Position (m)'); a = axis; axis([0 201 a(3) a(4)]);

figure(2); clf; plot(xtrue(1,1:200)-xhatstore(1,:)); hold on;
plot(3*sqrt(sigmastore(1,:)),'k--','linewidth',0.5);
plot(-3*sqrt(sigmastore(1,:)),'k--','linewidth',0.5);
```

```
title('Tracking error and bounds');
xlabel('Iteration'); ylabel('Error (m)'); axis([0 201 -1.5 1.5]);

figure(3); clf; plot(modestore');
hold on; plot([0 201],[0.5 0.5],'k--','linewidth',0.5);
title('Estimated mode PMF values'); xlabel('Iteration');
ylabel('Probability'); axis([0 201 -0.05 1.05]);
```

- The following graphs show sample output from this code




- Some comments:
  - The state estimate is always very good;
  - The error bounds improve during NCP (calmer dynamics);
  - The mode tracking is very good: Errors only when NCV is actually stopped, or NCP is actually moving more than a negligible amount.