UDRC Summer School 2019
Day 2 - Sensing and Tracking

David Cormack*, Mengwei Sun, James R. Hopgood
*drc9@hw.ac.uk

University of Edinburgh

25^{th} June 2019

Overview I

1. Overview of multi-target tracking (MTT)
   - Problem Formulation
   - Detection Methods
   - Motion Models
   - Correlation/Association
   - Observation Models
2. Single-Target Tracking
   - Introduction
   - Kalman Filter
   - Particle Filter

Overview II

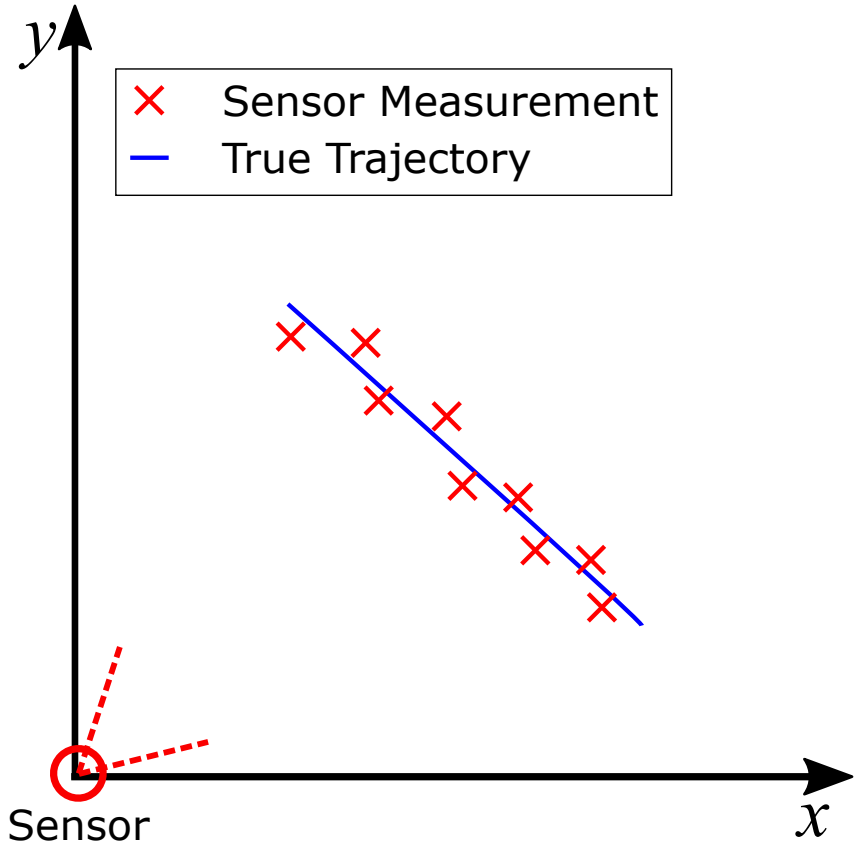
3. Data Association
   - Quick Recall
   - Munkres/Hungarian
   - Auction
   - PDA Filtering
4. Advanced MTT
   - Set Methods
   - Vector Methods
   - Interacting Multiple Models
   - Data Fusion
   - Performance Evaluation in MTT

Contents

1. Overview of MTT
   - Problem Formulation
   - Detection Methods
   - Motion Models
   - Correlation/Association
   - Observation Models
2. Single-Target Tracking
3. Data Association
4. Advanced MTT

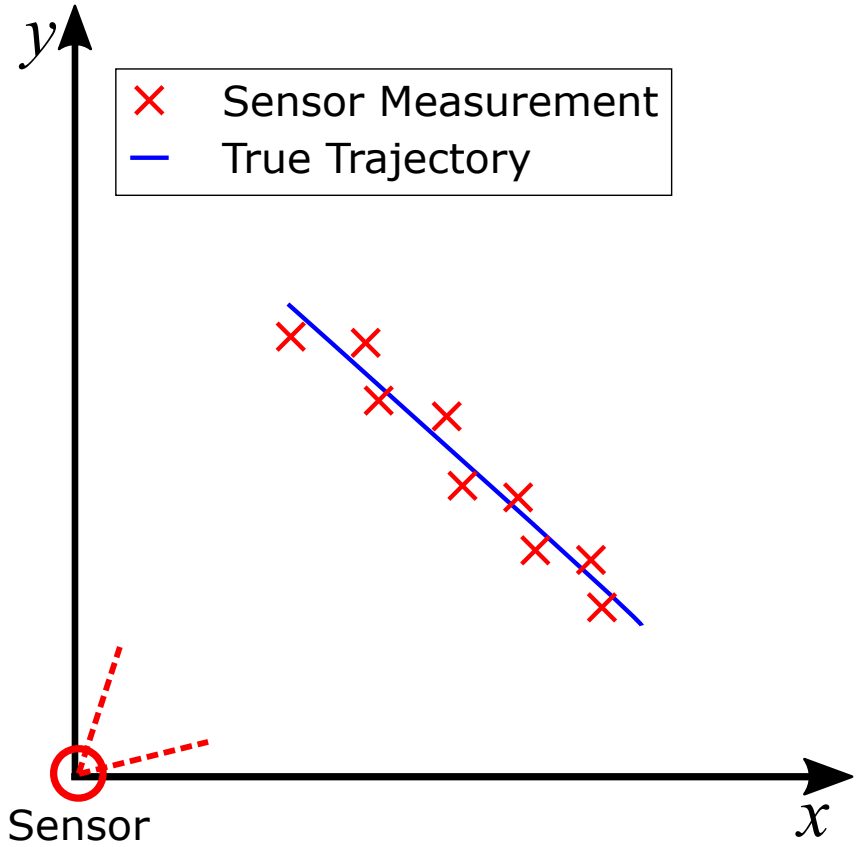
What is target tracking?

- Estimation of an object's *state* using sensor *measurements*
- Many design choices to be made!
- Let's start with this simple example...



What is target tracking?

- Consider a single target moving at a constant velocity in open space
- Our *sensor* measures the position of this target with very high accuracy
- This *sensor* has an excellent detection profile, and does not detect any other items in this space



What is target tracking?

- Sadly, all of these points occur very rarely in practice!

What is target tracking?

- Sadly, all of these points occur very rarely in practice!
- Typically find multiple targets of interest in a scene, following completely different trajectories

What is target tracking?

- Sadly, all of these points occur very rarely in practice!
- Typically find multiple targets of interest in a scene, following completely different trajectories
- Different types of sensor will measure different properties *(position, velocity, size, ...)* and to varying accuracies *(time, environment, ...)*

What is target tracking?

- Sadly, all of these points occur very rarely in practice!
- Typically find multiple targets of interest in a scene, following completely different trajectories
- Different types of sensor will measure different properties *(position, velocity, size, ...)* and to varying accuracies *(time, environment, ...)*
- Detection profiles are imperfect, and will often allow clutter and false alarms through, along with target measurements

Notation - your starter for 10!

Some definitions and symbols typically used in MTT, more will be introduced throughout as and when we need them!

t : the current time-step

$t-1$: the previous time-step

Δ_t : transition time (time between t and $t-1$)

$\mathbf{x} \in X_t$: the set of *true* target states, at time t typically formed with position and velocity information

$\mathbf{z} \in Z_t$: the set of sensor *measurements* collected at time t

$\hat{\mathbf{x}} \in \hat{X}_t$: the set of *estimated* target states provided as an output of the MTT algorithm at time t

How do we start structuring the problem?

Let's assume we're working in a *2-dimensional, flat Earth space* today. This gives us a *4-dimensional* space to work in. Our *arbitrary sensor* can only measure the target's position.

$$\mathbf{x}_k = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix} \qquad \mathbf{z}_m = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

$$X_t = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_K \end{bmatrix} \qquad Z_t = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_M \end{bmatrix}$$

Note

The number of targets $k \in \{1, \dots, K\}$ we are tracking, and the number of measurements $m \in \{1, \dots, M\}$ we receive at a given time-step can both vary!

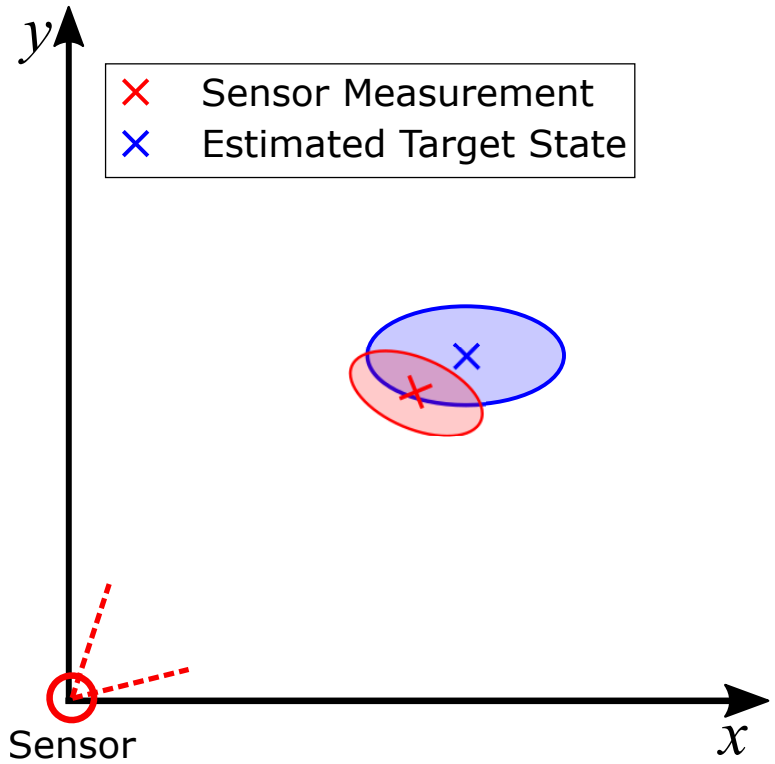
How do we start structuring the problem?

For now, we will represent our target states using *Gaussian distributions*. They are robust, and by using some algebra, are a very useful tool in target tracking.

Gaussian distributions are defined by two parameters

- mean μ
- variance σ^2

and are usually written as $\mathcal{N}(\mu, \sigma^2)$. The *noisy* measurements will also be represented in this way.

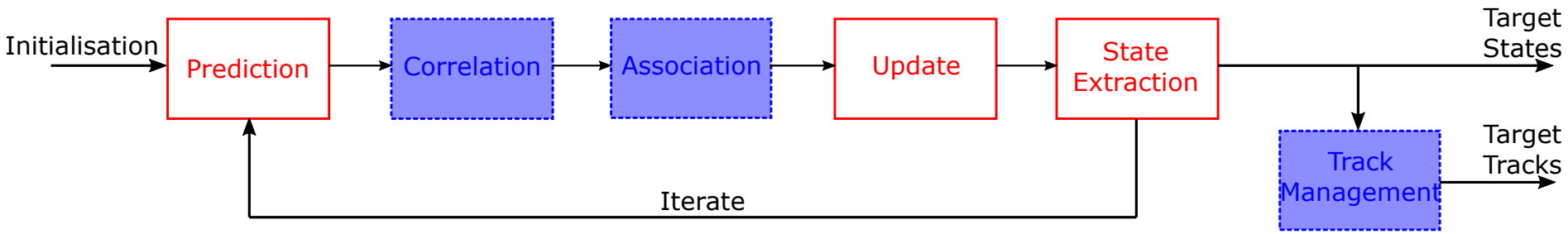


What blocks do we need?

There is no "golden algorithm" for MTT, there are lots of design options available. Certain algorithms are better at performing certain tasks as we'll see throughout the day!

- prior information - input
- motion model(s)
- correlation/association
- observation model(s)
- state extraction/track management - output

What blocks do we need?



- Red boxes are required functions!
- Blue boxes are optional design choices!

Prior Info. - How do we start?

There is a lot of *physical* and *contextual* information available to us that we can exploit as prior information to feed the tracking algorithm. These can be linked with the assumptions that follow on the next slide, to *constrain* the overall tracking problem.

Prior Info. - How do we start?

There is a lot of *physical* and *contextual* information available to us that we can exploit as prior information to feed the tracking algorithm. These can be linked with the assumptions that follow on the next slide, to *constrain* the overall tracking problem.

Physical

- Maximum detection range
- Platform dynamics
- Obscurations

Contextual

- Target dynamics
- Location
- Mission objectives

Prior Info. - How do we start?

There is a lot of *physical* and *contextual* information available to us that we can exploit as prior information to feed the tracking algorithm. These can be linked with the assumptions that follow on the next slide, to *constrain* the overall tracking problem.

Physical

- Maximum detection range
- Platform dynamics
- Obscurations

Contextual

- Target dynamics
- Location
- Mission objectives

Such information can help us set up an appropriate size of state-space, choosing a *birth model*, or restricting trajectories to portions of the state-space.

Prior Info. - Assumptions

1. The multi-target state X_t evolves according to a first-order Markov process.
2. The single-target states $\mathbf{x}_t$ evolve independently.
3. Each measurement $\mathbf{z}_m$ will originate from a target or from clutter; it cannot originate from more than one target.
4. Each target will be treated as a "point"; it can generate no more than one measurement $\mathbf{z}_m$.
5. The number of clutter measurements at time t will be *Poisson distributed* with mean λ_c.
6. The clutter measurements are independent of the target measurements.

Note

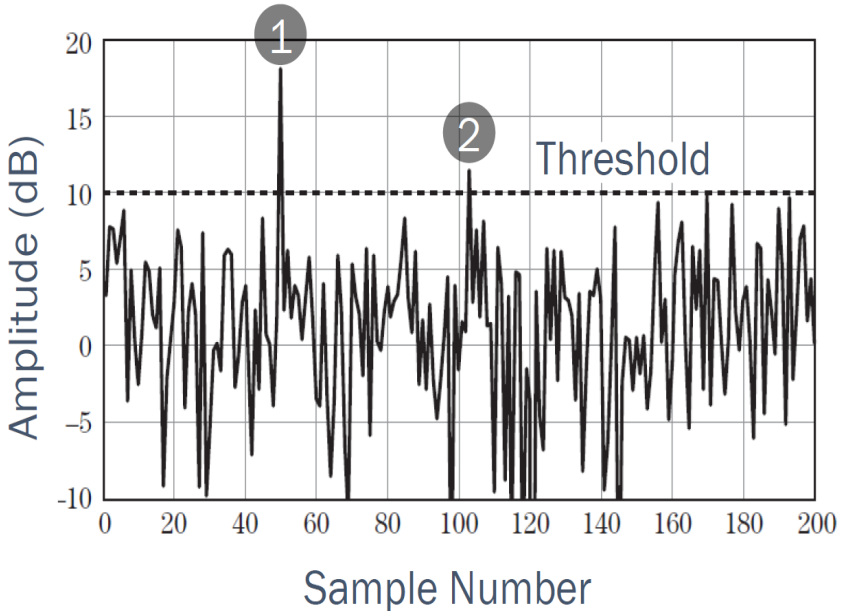
These will be *adapted or removed* as we progress through the material!

Prior Info. - Thresholding

The measurements that are received into the tracking algorithm are likely to be the result of some pre-processing performed during the detection step. This could be a simple threshold check; if the received signal power is above a preset level, record a measurement at that location.

Other types of pre-processing and schemes could include

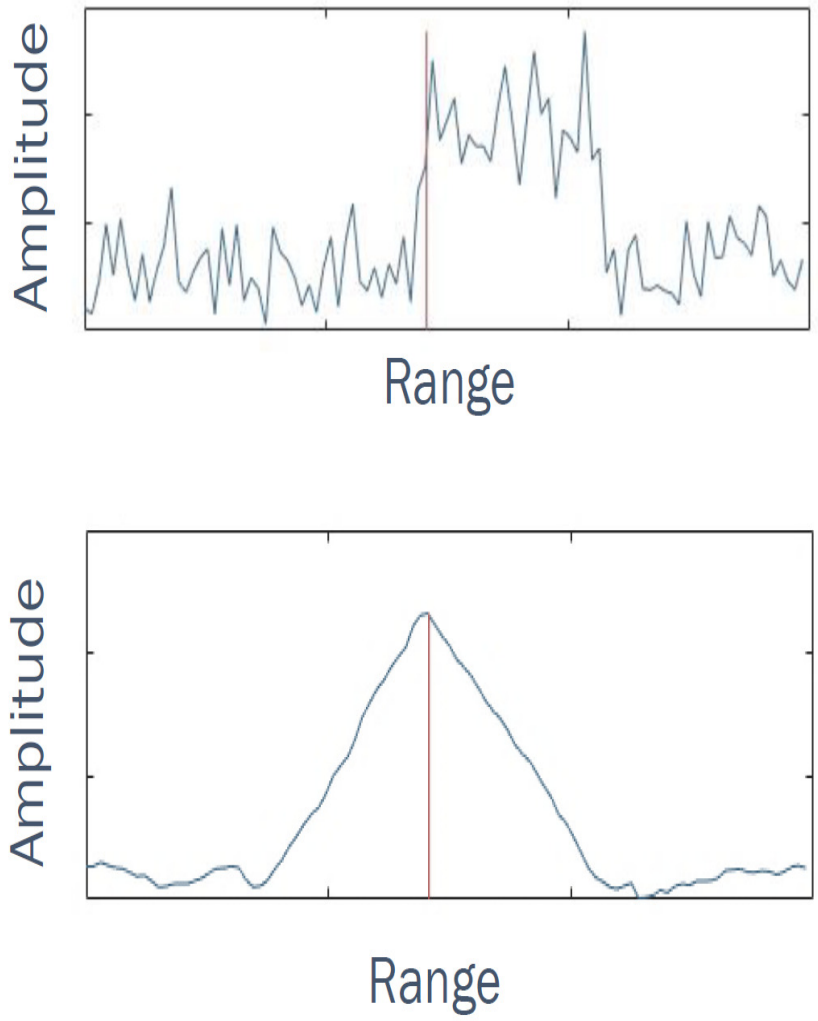
- CFAR (radar)
- M/N detection
- matched filter



Prior Info. - Matched Filter (Radar)

A typical scheme found in radar is the matched filter. This exploits knowledge of the transmitted radar waveform, to help maximise the Signal-to-Noise Ratio (SNR) in the received signal. When additive Gaussian noise is present, the optimal filter is a *time-reversed* version of the transmitted signal.

$$h(t) = x^*(\tau_{max} - t)$$



Motion Models

We need to start making some assumptions about how we expect our targets to move in time. This choice is then used to formulate the *PREDICTION* step in our MTT algorithm. Some commonly used models include

- Brownian motion - static targets
- constant velocity
- constant turn rate
- constant acceleration
- ...

New Definitions - State Prediction

F - transition matrix
Q - process noise covariance

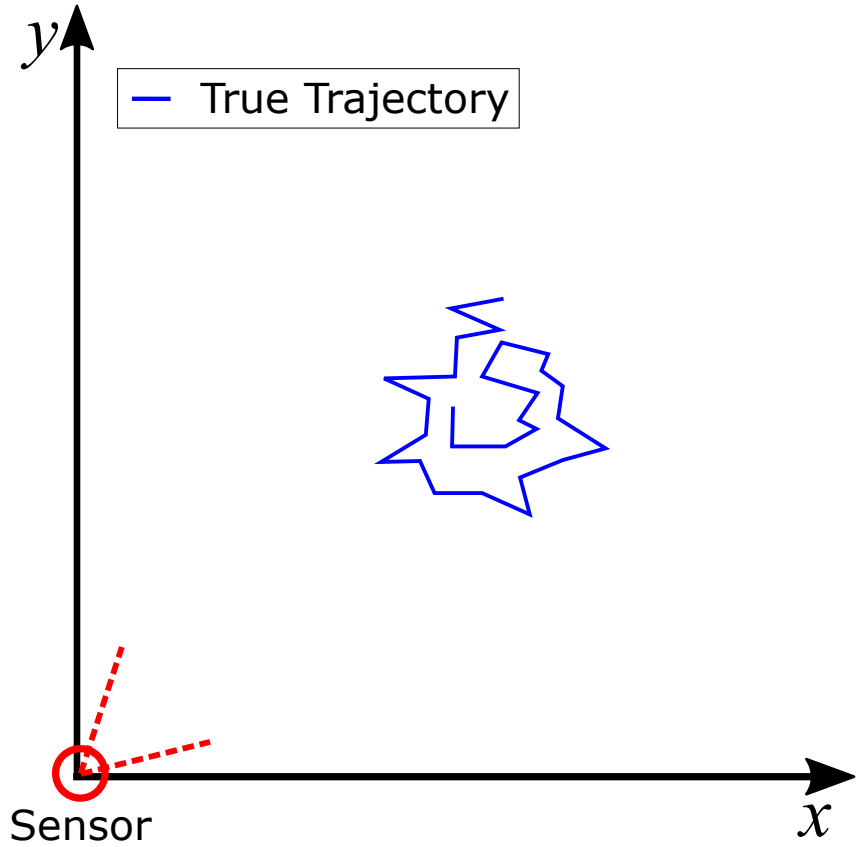
Example - Brownian Motion

Used when targets will either be static, or drift by small amounts. Examples include -

- boats/buoys bobbing on sea surface
- ground emitter localisation

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} \frac{1}{3}\Delta_t^3 & \frac{1}{2}\Delta_t^2 & 0 & 0 \\ \frac{1}{2}\Delta_t^2 & \Delta_t & 0 & 0 \\ 0 & 0 & \frac{1}{3}\Delta_t^3 & \frac{1}{2}\Delta_t^2 \\ 0 & 0 & \frac{1}{2}\Delta_t^2 & \Delta_t \end{bmatrix}$$



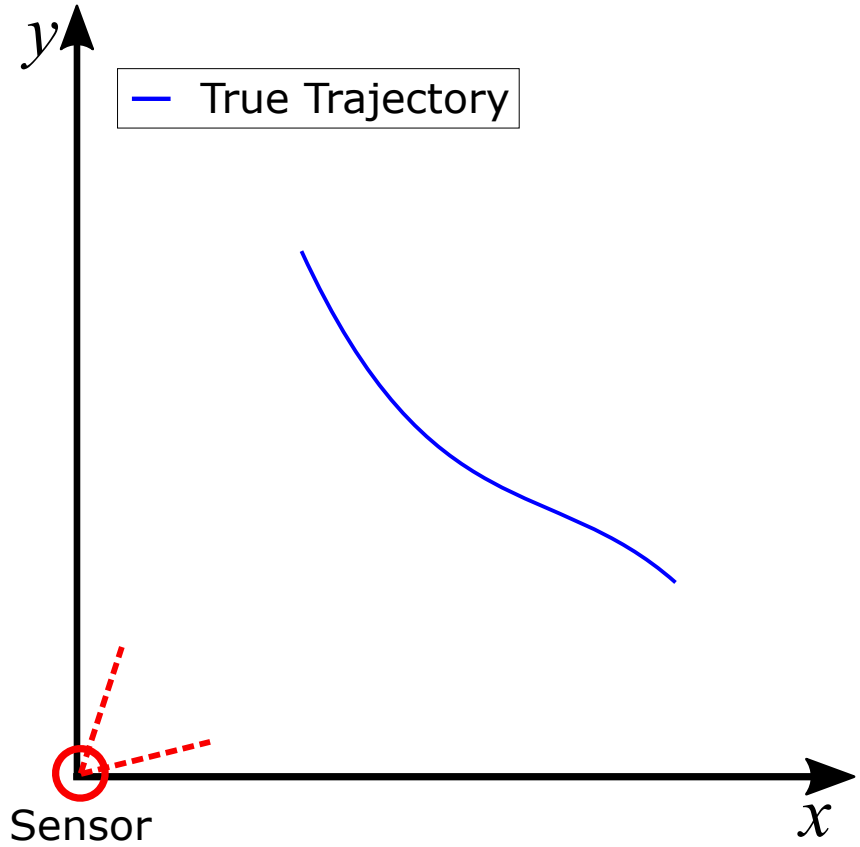
Example - Constant Velocity

Used when targets will be travelling in (almost) straight lines and maintaining velocity. Examples include -

- people walking
- passenger planes cruising

$$F = \begin{bmatrix} 1 & \Delta_t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

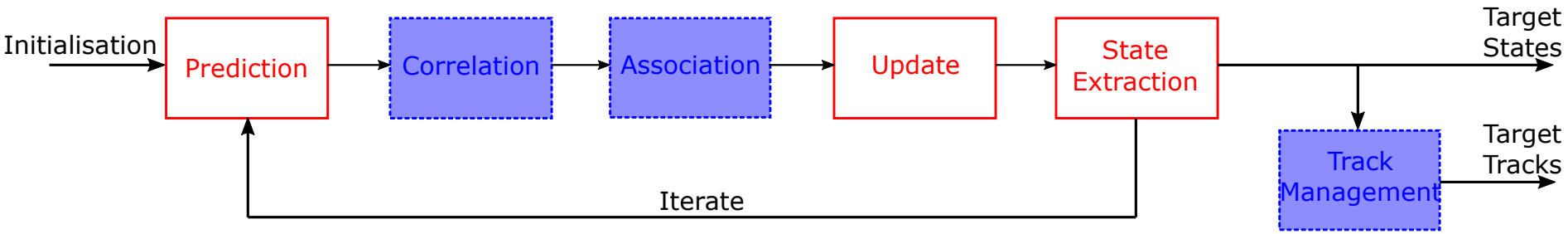
$$Q = \begin{bmatrix} \frac{1}{3}\Delta_t^3 & \frac{1}{2}\Delta_t^2 & 0 & 0 \\ \frac{1}{2}\Delta_t^2 & \Delta_t & 0 & 0 \\ 0 & 0 & \frac{1}{3}\Delta_t^3 & \frac{1}{2}\Delta_t^2 \\ 0 & 0 & \frac{1}{2}\Delta_t^2 & \Delta_t \end{bmatrix}$$



What is correlation (gating)?

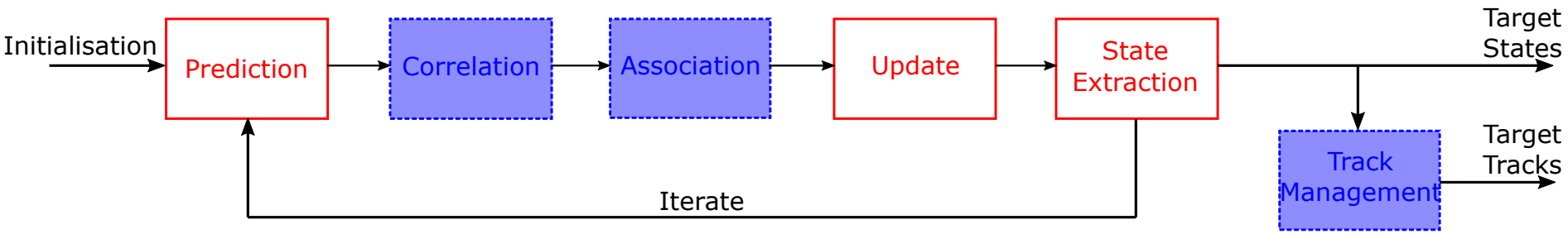
Correlation is an *optional* tool we can use in tracking algorithms, to help us reduce computational effort further down the chain! We don't want to try and associate a measurement to a track that is very far away.

Using a *window or threshold*, we can define a distance limit within which we will assume that it is possible for a measurement and target to correlate. This could be rectangular, or taken from some distribution...



What is data association?

Once we have our *feasible associations*, we now need to find the most likely assignment, based on the assumptions defined earlier e.g. only one measurement per target. We can see that for larger scenarios, there could be many permutations or possible solutions...



We will visit this problem again in much more detail later on today!

Observation Models

An observation model forms the majority of the *UPDATE* step in tracking targets. The model is dependent on the types of measurement $\mathbf{z}_m$ that the "sensor" makes, and how they relate to the the variables in the tracking state vector $\mathbf{x}_k$.

Observation Models

An observation model forms the majority of the *UPDATE* step in tracking targets. The model is dependent on the types of measurement $\mathbf{z}_m$ that the "sensor" makes, and how they relate to the the variables in the tracking state vector $\mathbf{x}_k$.

Sensor Types

- Radar
- Lidar
- Infrared
- Camera
- ...

Measured Variables

- Position
- Velocity
- SNR
- Size
- ...

New Definitions - State Update

H - observation matrix

R - measurement noise covariance

Example - Linear Relationship

Consider a situation where we want to track an object moving across an image. Our state vector $\mathbf{x}$ will contain the same variables, but the units are in pixels, and pixels per second. Having performed some object detection and thresholding on the image, we generate *noisy* position measurements.

$$\mathbf{x}_k = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix} \qquad \mathbf{z}_m = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

Example - Linear Relationship

Consider a situation where we want to track an object moving across an image. Our state vector $\mathbf{x}$ will contain the same variables, but the units are in pixels, and pixels per second. Having performed some object detection and thresholding on the image, we generate *noisy* position measurements.

$$\mathbf{x}_k = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix} \qquad \mathbf{z}_m = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

In order to perform a linear update of the state, we will use the H matrix to "strip out" the appropriate variables from the state. In this case -

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Example - Linear Relationship

Performing the matrix multiplication $H\mathbf{x}_k$, the resultant matrix will be

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

The sensor (and it's output measurements) will be subjected to system losses and noise, making them less accurate and adding some extra *uncertainty* into our model. We can account for this using the diagonal R matrix, which is made up of the expected uncertainty in each measured variable. In this example, the R matrix would contain -

$$R = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

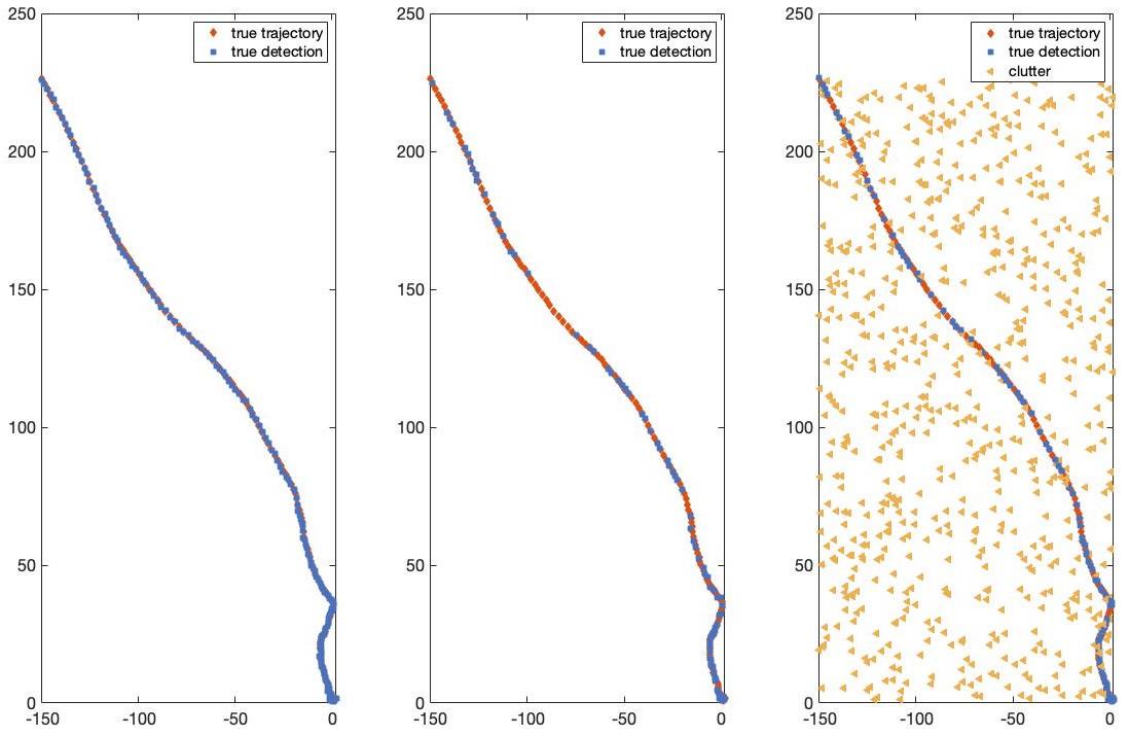
where σ_x^2 is the variance in the x-dimension, and σ_y^2 is the variance in the y-dimension.

Contents

1. Overview of MTT
2. Single-Target Tracking
   - Introduction
   - Kalman Filter
   - Particle Filter
3. Data Association
4. Advanced MTT

Introduction

- Single-target tracking: Processing of measurements obtained from one target to estimate its current state. e.g., position, velocity, acceleration, and turn rate.
- Single-target tracking in practice: missing detection, clutter and non-linearity



Introduction

- Linear model with no missing detection or clutter $\Rightarrow$ Kalman filter (KF) with prediction and update of states and covariance
- Linear model with missing detection but no clutter $\Rightarrow$ KF with prediction and update of only covariance (track coasting)
- Linear model with missing detection and clutter $\Rightarrow$ Probabilistic Data Association filter
- Nonlinear model $\Rightarrow$ Extended Kalman filter, Unscented Kalman Filter (UKF), Cubature Kalman Filter (CKF and its variants), Particle filter, and more!

Kalman Filter

What is a Kalman Filter and What Can It Do?

A Kalman filter is an optimal estimator - uses the measurements to learn about the unobservable state variables.

Optimal in What Sense?

For linear system and white Gaussian errors, the Kalman filter minimises the mean square error of the estimated parameters.

Formulating a Kalman Filter

KF models dynamically measurement z_t and the state x_t.

$$x_t = g(x_{t-1}, v_t), \quad \text{state equation}$$
$$z_t = f(x_t, w_t), \quad \text{measurement equation}$$

How does the KF work?

- It is a recursive data processing algorithm. As new information arrives, it **predicts** $(x_{t|t-1}, P_{t|t-1})$.
  - Use previous estimate and historical measurements to make prediction. Integrating over x_{t-1} gives the *Chapman-Kolmogorov equation*:

$$p(x_t|\{z_{t-1}\}) = \int p(x_t|x_{t-1})p(x_{t-1}|\{z_{t-1}\})dx_{t-1}$$

- It **updates** using given measurements $\{z_t\}$ to give us x_t.
  - Use measurement to update prediction by blending prediction and residual
  - Optimal estimate with smaller variance

Chapman-Kolmogorov equation

- Suppose that f_i is an indexed collection of random variables, that is, a stochastic process.
- Joint probability density function $p_{i_1,\ldots,i_n}(f_1, \ldots, f_n)$
- The Chapman-Kolmogorov equation is

$$p_{i_1,\ldots,i_{n-1}}(f_1, \ldots, f_{n-1}) = \int_{-\infty}^{\infty} p_{i_1,\ldots,i_n}(f_1, \ldots, f_n)\, df_n$$

- When the stochastic process under consideration is Markovian, the Chapman-Kolmogorov equation is equivalent to an identity on transition densities.

$$p_{i_1,\ldots,i_n}(f_1, \ldots, f_n) = p_{i_1}(f_1) p_{i_2;i_1}(f_2 \mid f_1) \cdots p_{i_n;i_{n-1}}(f_n \mid f_{n-1}),$$

KF - Linear dynamic system

Continuous White Noise Acceleration Model

- Discrete time state equation, $\mathbf{x}_{t-1} = [x_t, \dot{x}_t, y_t, \dot{y}_t]^T$

$$\mathbf{x}_{t|t-1} = F\mathbf{x}_{t-1} + \mathbf{v}_t$$

The transition matrix is

$$F = I_{2\times 2} \otimes \begin{bmatrix} 1 & \Delta_t \\ 0 & 1 \end{bmatrix}$$

The discrete time process noise, $\mathbf{v}_t \sim \mathcal{N}(0, Q)$,

$$Q = I_{2\times 2} \otimes \begin{bmatrix} \frac{1}{3}\Delta_t^3 & \frac{1}{2}\Delta_t^2 \\ \frac{1}{2}\Delta_t^2 & \Delta_t \end{bmatrix} q$$

- Measurement, $\mathbf{z}_t = [x_t, y_t]^T$

$$\mathbf{z}_t = H\mathbf{x}_t + \mathbf{w}_t.$$

$H = I_{2\times 2} \otimes [1, 0]$, $\mathbf{w}_t \sim \mathcal{N}(0, R)$ and $R = rI_{2\times 2}$.

KF - algorithm

- **Predict**

$$\begin{aligned} \text{State:} \quad & \mathbf{x}_{t|t-1} = F\mathbf{x}_{t-1}, \\ \text{Covariance:} \quad & P_{t|t-1} = FP_{t-1}F' + Q, \\ \text{Measurement:} \quad & \hat{\mathbf{z}}_{t|t-1} = H\mathbf{x}_{t|t-1} \end{aligned}$$

- **Update**

$$\begin{aligned} \text{State:} \quad & \hat{\mathbf{x}}_t = \mathbf{x}_{t|t-1} + W_t\tilde{\mathbf{e}}_t, \\ \text{Covariance:} \quad & P_t = (I - W_tH)P_{t|t-1}. \end{aligned}$$

$$\begin{aligned} \text{Kalman gain:} \quad & W_t = P_{t|t-1}H'S_t^{-1}, \\ \text{Innovation:} \quad & \tilde{\mathbf{e}}_t = \mathbf{z}_t - \hat{\mathbf{z}}_{t|t-1} = \mathbf{z}_t - H\hat{\mathbf{x}}_{t|t-1}, \\ \text{Innovation Covariance:} \quad & S_t = HP_{t|t-1}H' + R. \end{aligned}$$

Nonlinear Model — CT Model

- The turn of most vehicles usually follows a pattern known as Coordinated Turn (CT), characterised by a (nearly) constant turn and a (nearly) constant velocity.
- CT is nonlinear if the turn rate is not a known constant. The state vector $\mathbf{x}_{t-1} = [x_t, \dot{x}_t, y_t, \dot{y}_t, \Omega_t]^T$.

$$\mathbf{x}_t = \begin{bmatrix} 1 & \frac{\sin \Omega_t \Delta_t}{\Omega_t} & 0 & -\frac{1-\cos \Omega_t \Delta_t}{\Omega_t} & 0 \\ 0 & \cos \Omega_t \Delta_t & 0 & -\sin \Omega_t \Delta_t & 0 \\ 0 & \frac{1-\cos \Omega_t \Delta_t}{\Omega_t} & 1 & \frac{\sin \Omega_t \Delta_t}{\Omega_t} & 0 \\ 0 & \sin \Omega_t \Delta_t & 0 & \cos \Omega_t \Delta_t & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} \frac{1}{2}\Delta_t^2 & 0 & 0 \\ \Delta_t & 0 & 0 \\ 0 & \frac{1}{2}\Delta_t^2 & 0 \\ 0 & \Delta_t & 0 \\ 0 & 0 & \Delta_t \end{bmatrix} \mathbf{v}_t$$

- The zero-mean white process noise $\mathbf{v}_t$ is 3-dimensional: the first two dimensions model the linear acceleration while the third one models the turn acceleration.

Particle Filtering - Introduction

The particle filter is an alternative approach to performing target tracking, where the distributions are represented using sets of samples (*particles*) rather than a Gaussian.

Previously, these Monte Carlo (MC) types of filter were less favourable, due to issues with high complexity, and longer computation times. However with advances in processing capability, these methods can now be implemented in real-time.

The most basic, and first feasible implementation of MC methods for target tracking is the *bootstrap filter*.

Note

There are thankfully a number of built-in functions in MATLAB (other tracking tools/simulators are available...) to make sampling methods simple!

Particle Filtering - Formulation

To develop this filter, start by thinking about the change in target state representation. Rather than being represented by $\mathcal{N}(\mu, \sigma^2)$, it is now represented using a weighted set of particles $\left\{\mathbf{x}_t^i, w_t^i\right\}_{i=1}^N$. Each $\mathbf{x}_t^i$ is a state vector, drawn from a prior distribution.

The likelihood that the sample contains the true target state, conditioned on measurement data is then evaluated as the *update* step.

Note

We will assume that the distributions/likelihood functions here are Gaussian for simplicity, but other types could be used in practice!

Particle Filtering - Implementation

Initialisation:

At $t = 1$, draw an initial set of N samples from $\mathcal{N}(\mu, \sigma^2)$ centred on the location of the first *noisy* measurement (measurement-driven birth).

$$\mathbf{x}_1^i \sim \mathcal{N}(\mathbf{z}_1, R), \quad w_1^i = \frac{1}{N}$$

Prediction:

From $t = 2$, apply the chosen *motion model* to each individual particle state, plus some zero-mean Gaussian noise.

$$\mathbf{x}_{t|t-1}^i = F\mathbf{x}_{t-1}^i + u(t), \quad u(t) = \mathcal{N}(0, Q)$$

Update:

Generate new weights for each of the predicted samples, conditioned on the newly-received measurement.

$$w_t^i = \mathcal{N}(\mathbf{x}_{t|t-1}^i; \mathbf{z}_t, R)$$

Particle Filtering - Implementation

We must normalise these weights such that $\sum_{i=1}^{N} w_t^i = 1$ (rules of PDFs!).

We now have a further design choice to make; do we extract the Maximum A Posteriori (MAP) sample from the distribution? Or perform a weighted mean (WM) over the whole set?

MAP or WM?

MAP - sample with highest weight, could be erratic over time
WM - combination of all samples, likely to be more stable

Using either scheme, we can extract the new state estimate $\hat{\mathbf{x}}_t$.

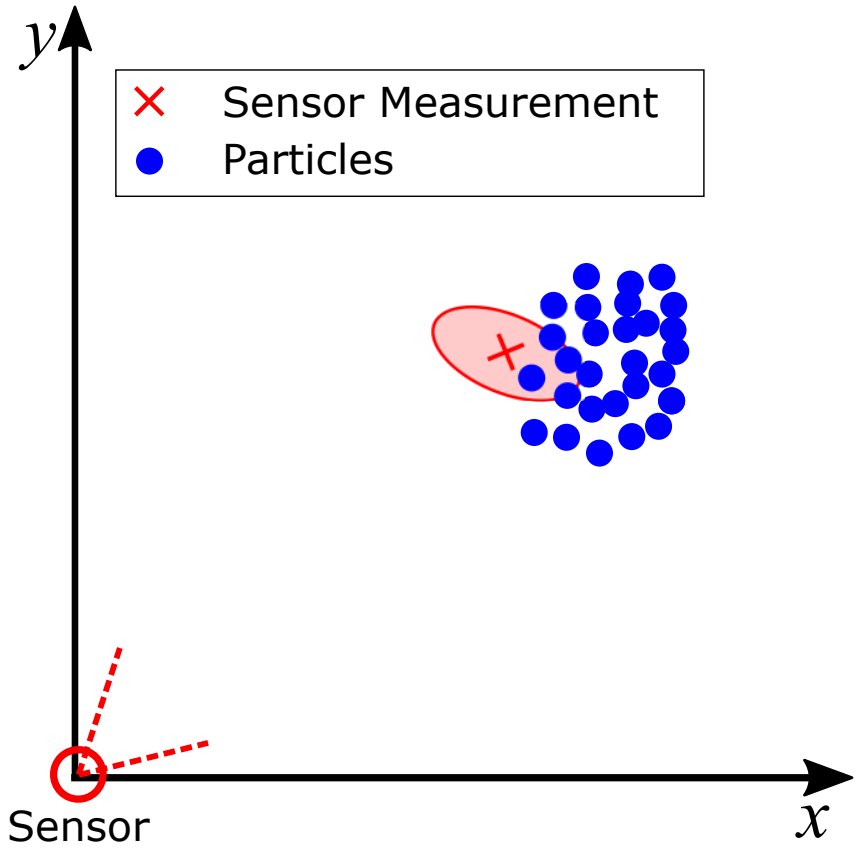
Particle Filtering - Resampling

One last step in the process is that of *resampling*, and arguably, it is the most important! Over time, some of the samples are likely to move away from the true trajectory being tracked. This can lead to samples with very low weights which contribute very little to the final estimation. This is called *particle degeneracy*.

There are a number of ways to perform resampling -

- Replacement
- Systematic
- Stratified
- ...

each having it's positives and negatives!



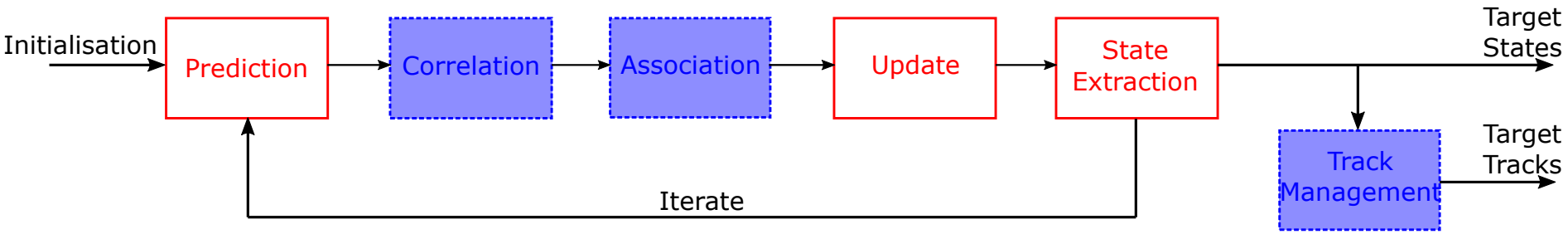
Contents

1. Overview of MTT
2. Single-Target Tracking
3. Data Association
   - Quick Recall
   - Munkres/Hungarian
   - Auction
   - PDA Filtering
4. Advanced MTT

Recall - Correlation

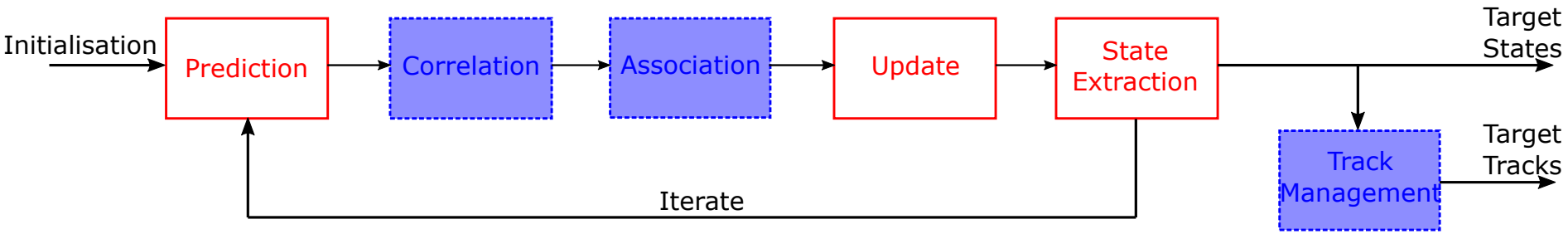
Correlation is an *optional* tool we can use in tracking algorithms, to help us reduce computational effort further down the chain! We don't want to try and associate a measurement to a track that is very far away.

Using a *window or threshold*, we can define a distance limit within which we will assume that it is possible for a measurement and target to correlate. This could be rectangular, or taken from some distribution...



Recall - Data Association

Once we have our *feasible associations*, we now need to find the most likely assignment, based on the assumptions defined earlier e.g. only one measurement per target. We can see that for larger scenarios, there could be many permutations or possible solutions...



Let's look at some algorithms that will help solve this problem!

Munkres/Hungarian Algorithm

The Munkres (or Hungarian) algorithm is a very common method for resolving the association/assignment problem. It was developed in the 1950's by Kuhn, but he named it after two Hungarian mathematicians who laid the ground work previously!

Munkres/Hungarian Algorithm

The Munkres (or Hungarian) algorithm is a very common method for resolving the association/assignment problem. It was developed in the 1950's by Kuhn, but he named it after two Hungarian mathematicians who laid the ground work previously!

Let's consider a case where we have three people, and three different tasks.

People

- David
- Mengwei
- James

Tasks

- A
- B
- C

——	A	B	C
David	2	3	4
Mengwei	4	2	3
James	3	4	2

Munkres/Hungarian Algorithm

Without going in to heavy detail, the Munkres algorithm performs a number of repeated row and column operations to this cost matrix to find the optimal assignment.

This works well for small matrices with only a few people/tasks to assign, but *scalability* is an issue. We are also constrained to *matrices with equal dimensions* i.e. square matrices.

Auction Algorithm

The auction algorithm works in a contrasting fashion. Imagine that all of the tasks are now up for sale, and the people involved have to *bid* for these tasks.

Auction Algorithm

The auction algorithm works in a contrasting fashion. Imagine that all of the tasks are now up for sale, and the people involved have to *bid* for these tasks.

Each iteration of the algorithm has two stages:

- Bidding
- Assignment

Auction Algorithm

The auction algorithm works in a contrasting fashion. Imagine that all of the tasks are now up for sale, and the people involved have to *bid* for these tasks.

Each iteration of the algorithm has two stages:

- Bidding
- Assignment

Example -

Consider the scenario from the previous slide. For some reason, myself and James both really want to perform task C. Based on some financial constraints, we start a bidding frenzy and Mengwei is the *auctioneer*.

Auction Algorithm

Both myself and James have an *alternative choice*, so if we don't win the auction, we can fall back onto this (assuming Mengwei isn't waiting to bid for it...).

Auction Algorithm

Both myself and James have an *alternative choice*, so if we don't win the auction, we can fall back onto this (assuming Mengwei isn't waiting to bid for it...).

Let's assume James wins and gets task C. I then fall back to task A as my next preferred choice. Two things can now happen -

Auction Algorithm

Both myself and James have an *alternative choice*, so if we don't win the auction, we can fall back onto this (assuming Mengwei isn't waiting to bid for it...).

Let's assume James wins and gets task C. I then fall back to task A as my next preferred choice. Two things can now happen -

- I get task A and Mengwei gets task B, OR

Auction Algorithm

Both myself and James have an *alternative choice*, so if we don't win the auction, we can fall back onto this (assuming Mengwei isn't waiting to bid for it...).

Let's assume James wins and gets task C. I then fall back to task A as my next preferred choice. Two things can now happen -

- I get task A and Mengwei gets task B, OR
- Mengwei and myself start bidding on task A.

Auction Algorithm

Both myself and James have an *alternative choice*, so if we don't win the auction, we can fall back onto this (assuming Mengwei isn't waiting to bid for it...).

Let's assume James wins and gets task C. I then fall back to task A as my next preferred choice. Two things can now happen -

- I get task A and Mengwei gets task B, OR
- Mengwei and myself start bidding on task A.

This process continues until everyone has a task, and therefore we've solved the assignment problem! But can anyone see a problem with this?

Auction Algorithm

Both myself and James have an *alternative choice*, so if we don't win the auction, we can fall back onto this (assuming Mengwei isn't waiting to bid for it...).

Let's assume James wins and gets task C. I then fall back to task A as my next preferred choice. Two things can now happen -

- I get task A and Mengwei gets task B, OR
- Mengwei and myself start bidding on task A.

This process continues until everyone has a task, and therefore we've solved the assignment problem! But can anyone see a problem with this?

Initial conditions are very important!

Introducing clutter!

It is usually assumed that clutter measurements typically follow a *Poisson* point process or distribution.

Introducing clutter!

It is usually assumed that clutter measurements typically follow a *Poisson* point process or distribution.

- The mean number of clutter points per time-step is set to be λ_{fa}.

Introducing clutter!

It is usually assumed that clutter measurements typically follow a *Poisson* point process or distribution.

- The mean number of clutter points per time-step is set to be λ_{fa}.
- Specifically, at each time, the measurement area is set to be $V = (\bar{x} - \underline{x}) \times (\bar{y} - \underline{y})$, and the number of false alarm is generated by Matlab code '$N_{fa} = \text{poissrnd}(\lambda_{fa} * V)$'.

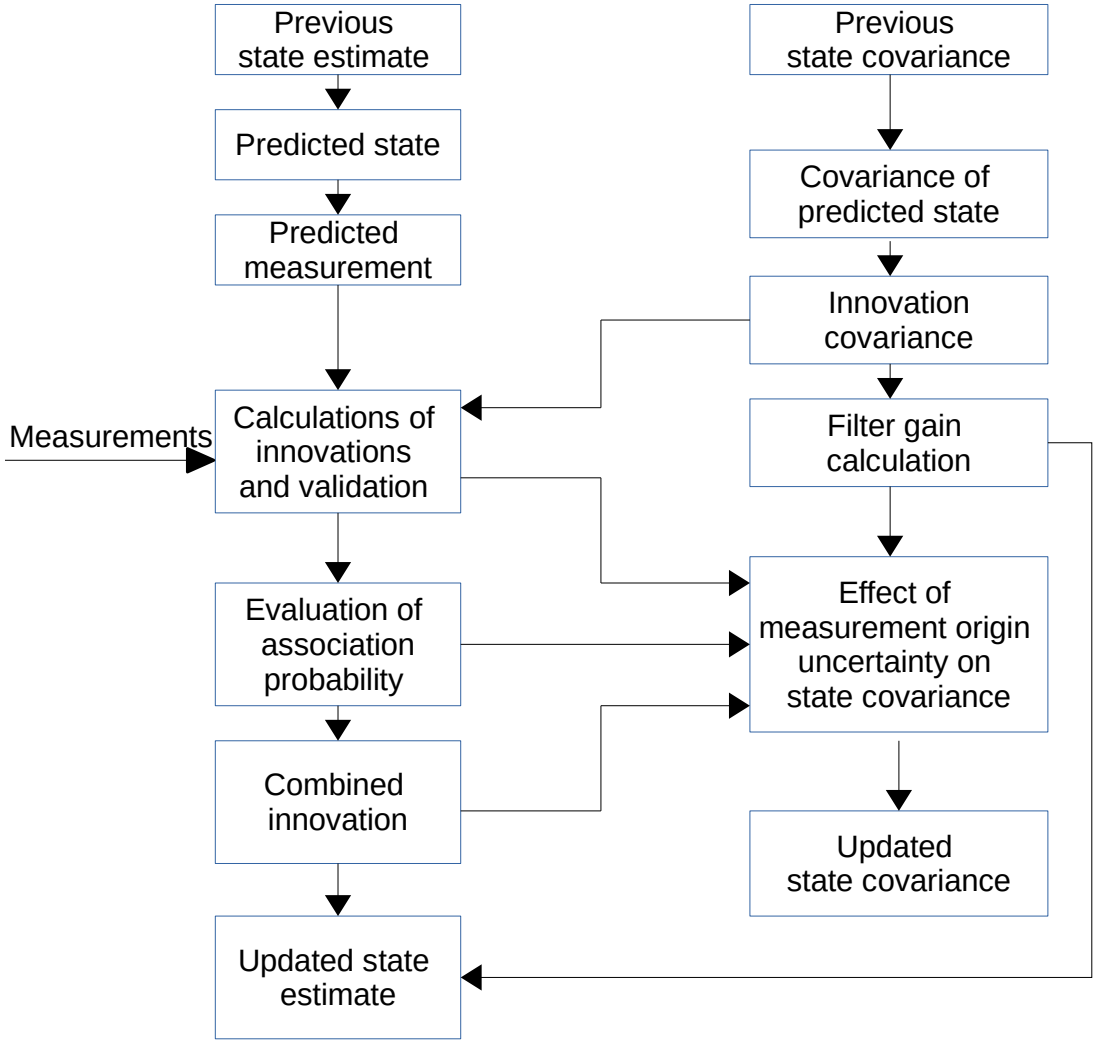
Introducing clutter!

It is usually assumed that clutter measurements typically follow a *Poisson* point process or distribution.

- The mean number of clutter points per time-step is set to be λ_{fa}.
- Specifically, at each time, the measurement area is set to be $V = (\bar{x} - \underline{x}) \times (\bar{y} - \underline{y})$, and the number of false alarm is generated by Matlab code '$N_{fa} = \text{poissrnd}(\lambda_{fa} * V)$'.
- The x-axis of each false alarm follows uniform distribution with region $[\underline{x}, \bar{x}]$. Similarly, the y-axis of each false alarm follows uniform distribution with region $[\underline{y}, \bar{y}]$.

Probabilistic Data Association (PDA) filter

The PDA filter mainly includes four steps, i.e., prediction, measurement validation, data association and state estimation.



Probabilistic data association (PDA) filter

- The prediction of state $\mathbf{x}_{t|t-1}$ and covariance $P_{t|t-1}$ are the same as the traditional KF.
- By setting a certain gate probability p_g, delete those measurements beyond the validation region $\mathcal{V}(t, \tau_g)$.

$$\mathcal{V}(t, \tau_g) = \left\{ \mathbf{z} : \left[\mathbf{z}_{t,m} - \hat{\mathbf{z}}_{t|t-1} \right]' S_t^{-1} \left[\mathbf{z}_{t,m} - \hat{\mathbf{z}}_{t|t-1} \right] \leq \tau_g \right\},$$

where τ_g is the gate threshold corresponding to a certain gate probability.

PDA filter

- The data association is described by association probability $\beta_{t,m}$ based on likelihood ratio (LR).

$$\beta_{t,m} = \begin{cases} \frac{\mathcal{L}_{t,m}}{1-p_d p_g+\sum_{j=1}^{M} \mathcal{L}_{t,j}}, & m = 1, ..., M \\ \frac{1-p_d p_g}{1-p_d p_g+\sum_{j=1}^{M} \mathcal{L}_{t,j}}, & m = 0 \end{cases}$$

where $m = 0$ stands for 'none is correct', i.e., missing detection. And the LR is calculated as:

$$\mathcal{L}_{t,m} = \frac{\mathcal{N}[\mathbf{z}_{t,m}; \hat{\mathbf{z}}_{t|t-1}, S_t] p_d}{\lambda_{fa}}.$$

PDA filter

- The state update of the PDAF is

$$\mathbf{x}_{t|t} = \mathbf{x}_{t|t-1} + W_t \nu_t,$$

- W_t is the gain of standard KF. ν_t is the combined innovation (considering maybe more than one validated measurement exists) and is calculated as:

$$\nu_t = \sum_{m=1}^{M} \beta_{t,m} \nu_{t,m},$$

PDA filter

- The covariance of the updated state is:

$$P_{t|t} = \beta_{t,0} P_{t|t-1} + [1 - \beta_{t,0}] P^c_{t|t} + \tilde{P}_t$$

- $P^c_{t|t}$ is the covariance of the state updated with the correct measurement.

$$P^c_{t|t} = P_{t|t-1} - W_t S_t W_t',$$

- $\tilde{P}_t$ means the spread of the innovation:

$$\tilde{P}_t = W_t \left\| \sum_{m=1}^{M} \beta_{t,m} \nu_{t,m} \nu_{t,m}' - \nu_t \nu_t' \right\| W_t'.$$

Joint Probabilistic Data Association (JPDA)

The Joint Probabilistic Data Association (JPDA) filter is a direct extension to the PDA filter. With this approach -

- The measurement to target association probabilities are computed jointly across the targets.
- State estimation is performed separately for each target - unique identifiers!

We start with the typical prediction for each individual target, and then proceed to the correlation step...

Joint Probabilistic Data Association (JPDA)

Gating is performed in much the same way as before; we want to rule out some of the joint association events as they are very unlikely to happen (negligible probability) and will not contribute to the final result.

For all of the events, we compute the likelihood of that event happening, such that

$$p(\mathbf{z}_{t,m}|\theta_t, Z_{1:t-1}) = \begin{cases} \ell & \text{if inside } \tau_g \\ p_{FA} & \text{if outside } \tau_g \end{cases}$$

$$\ell = \mathcal{N}\left(\mathbf{z}_{t,m}; \mathbf{x}_{t,k}, S_{\mathbf{x}_{t,k}}\right)$$

Notes!

τ_g - gating threshold
θ - association event(s)
$S = HPH^T + R$
$p_{FA} = \frac{\lambda_{FA}}{V}$ - probability of false alarm

Joint Probabilistic Data Association (JPDA)

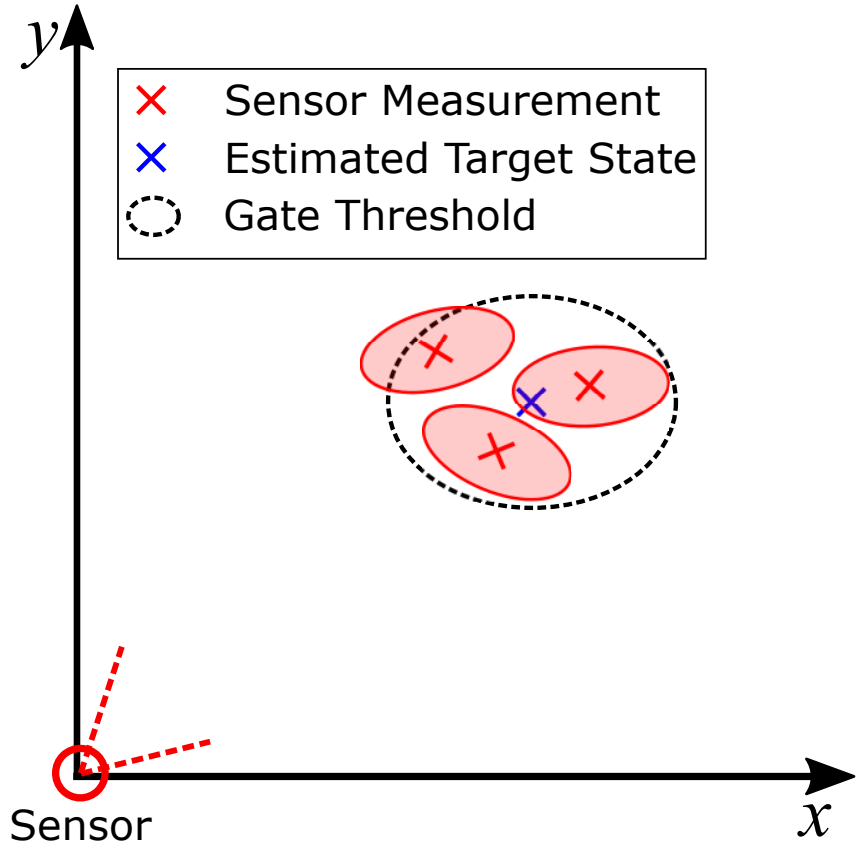
We can perform a *combinatorial* update, using all of the measurements that are feasible associations for a given target state $\mathbf{x}_{t,k}$.

Joint Probabilistic Data Association (JPDA)

We can perform a *combinatorial* update, using all of the measurements that are feasible associations for a given target state $\mathbf{x}_{t,k}$.

Let's consider this example, where three measurements fall inside τ_g for a target. Using the likelihoods computed in the previous slides, we can perform a form of weighted update. This effectively says -

"I'll take a pinch of all the measurements and not just stick to one!"



We can then perform state estimation in the same way as PDA filtering.

Joint Probabilistic Data Association (JPDA)

Advantages

- No hard decision made on association; a combination of all possibilities.
- Much less costly than GNN.
- Very simple to expand to from the single-target PDA filter.

Disadvantages

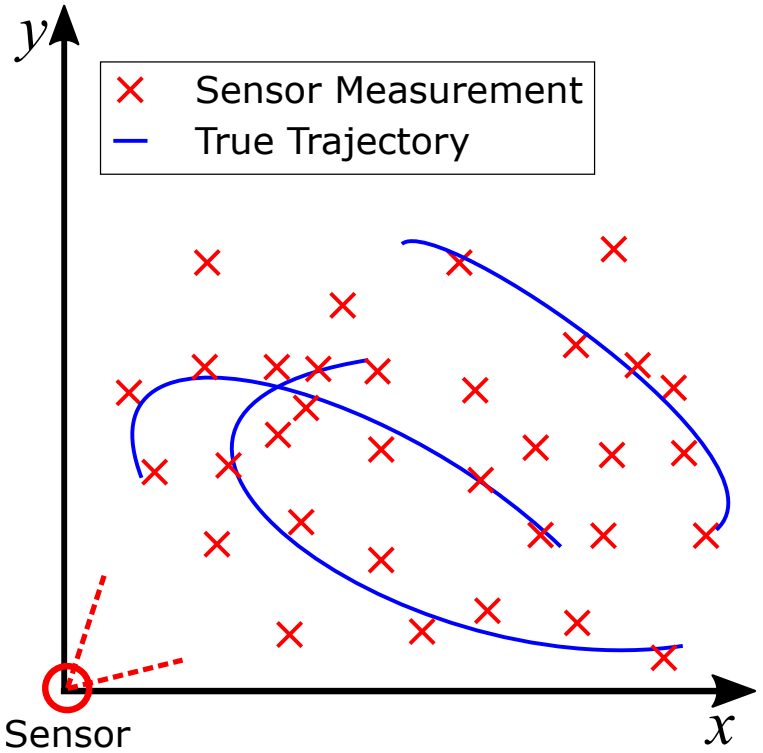
- Combinatorially expensive!
- Only designed for a fixed, known number of targets (needs reworked to JIPDA to allow for this).
- Poor in scenarios with closely-spaced targets.

Contents

1. Overview of MTT
2. Single-Target Tracking
3. Data Association
4. Advanced MTT
   - Set Methods
   - Vector Methods
   - Interacting Multiple Models
   - Data Fusion
   - Performance Evaluation in MTT

Introduction

Now that we have all of the components we need, we can extend our single-target examples to track multiple targets simultaneously. MTT brings together many of the topics we have covered this morning, including dealing with imperfect detection of targets, the potential for false alarms and clutter, and the unknown association between the received measurements and the target states.



Introduction

In MTT problems, not only do the target state variables change over time, the number of targets being tracked may change. This could be due to a number of reasons -

- a previously undetected target appears in the scene
- a new target enters from the edge of the scene
- a current target leaves the scene
- a current target can no longer be detected
- a current target *spawns* a new target (not covered today)

Objective

To jointly estimate, at each time-step t, the number of targets and their trajectories, from the *noisy, imperfect* sensor measurements.

Notation - Recall (and some extensions...)

Recall from Session 1:

t : the current time-step

$t-1$: the previous time-step

Δ_t : transition time (time between t and $t-1$)

$\mathbf{x} \in X_t$: the set of *true* target states, at time t typically formed with position and velocity information

$\mathbf{z} \in Z_t$: the set of sensor *measurements* collected at time t

$\hat{\mathbf{x}} \in \hat{X}_t$: the set of *estimated* target states provided as an output of the MTT algorithm at time t

Notation - Recall (and some extensions...)

We need to include some new variables to allow us to fully represent the MTT problem.

$p_d(\mathbf{x}_k)$: the probability of detection for target $\mathbf{x}_k$

$p_s(\mathbf{x}_k)$: the probability of a target's survival

p_b : the probability of a new target being born in the scene

$g_t(\mathbf{z}_{t,m}|\mathbf{x}_{t,k})$: single target likelihood function

The standard MTT model

Most of the MTT algorithms we will explore follow two standard models.

Motion Model

- An existing target $\mathbf{x}_{t-1,k}$ survives to time t with probability $p_s(\mathbf{x}_k)$, and moves to a new state $\mathbf{x}_{t,k}$ through a motion model.
- An existing target $\mathbf{x}_{t-1,k}$ dies at time t with probability $1 - p_s(\mathbf{x}_k)$.
- All surviving targets appear and evolve independently of one another.
- Incorporates a target birth process.

Observation Model

- A target $\mathbf{x}_{t,k}$ is detected with probability $p_d(\mathbf{x}_k)$ and generates an observation $\mathbf{z}_{t,m}$ with likelihood $g_t(\mathbf{z}_{t,m}|\mathbf{x}_{t,k})$.
- A target $\mathbf{x}_{t,k}$ is missed with probability $1 - p_d(\mathbf{x}_k)$.
- False alarm and clutter modelling.
- Observations generated independently, and all "point" targets.

Let's start off simple!

The simplest possible MTT algorithm, is the Global Nearest Neighbour (GNN) tracker, effectively using a Kalman filter on the associated measurements.

Advantages

- Intuitive solution, just run a simple Kalman filter!
- Simple to implement, we already have Kalman filter and association code!

Disadvantages

- Susceptible to losing tracks
- Very poor performance in closely-spaced scenarios
- Very computationally expensive!

For a scenario with any sort of challenging trajectories, this solution just won't cut it. We need some more robust algorithms that can perform more accurately, and more efficiently.

What are set-type methods?

MTT algorithms typically fall in to one of two categories, the first of which are *set-type methods*.

What are set-type methods?

MTT algorithms typically fall in to one of two categories, the first of which are *set-type methods*.

A Random Finite Set (RFS) is a random variable that takes values as *finite sets*. These are useful when analysing observed patterns of points, where the points represent the locations of some objects, e.g. measurements on a radar screen.

What are set-type methods?

MTT algorithms typically fall in to one of two categories, the first of which are *set-type methods*.

A Random Finite Set (RFS) is a random variable that takes values as *finite sets*. These are useful when analysing observed patterns of points, where the points represent the locations of some objects, e.g. measurements on a radar screen.

An RFS is completely specified by a discrete distribution, e.g. for a *Poisson* RFS, the cardinality is Poisson distributed with a given mean, and the points will be independently and identically distributed according to a chosen distribution (uniform, Gaussian, ...).

What are set-type methods?

MTT algorithms typically fall in to one of two categories, the first of which are *set-type methods*.

A Random Finite Set (RFS) is a random variable that takes values as *finite sets*. These are useful when analysing observed patterns of points, where the points represent the locations of some objects, e.g. measurements on a radar screen.

An RFS is completely specified by a discrete distribution, e.g. for a *Poisson* RFS, the cardinality is Poisson distributed with a given mean, and the points will be independently and identically distributed according to a chosen distribution (uniform, Gaussian, ...).

Fundamentally, like any random variable, an RFS is completely described by it's probability distribution.

The PHD filter

The Probability Hypothesis Density (PHD) filter is a first-moment approximation which alleviates the computational intractability of the multi-target Bayes filter. By propagating just the first moment (mean number of targets), the PHD filter operates on a single-object state space, and *avoids dealing with the data association problem!*

The recursion is given by -

$$v_{t|t-1}(X_t) = \int p_s(\mathbf{x}_k) f_{t|t-1}(X_t|X_{t-1}) v_{t-1}(X_{t-1}) \mathrm{d}X_{t-1} + \gamma_t(X_t)$$

$$v_t(X_t) = [1 - p_d(\mathbf{x}_k)]\, v_{t|t-1}(X_t) + \sum_{\mathbf{z}\in Z_t} \frac{p_d(\mathbf{x}_k) g_t(\mathbf{z}|\mathbf{x}_k) v_{t|t-1}(\mathbf{x}_t)}{\kappa_t(\mathbf{z}) + \int p_d(\mathbf{x}_k) g_t(\mathbf{z}|\mathbf{x}_k) v_{t|t-1}(\mathbf{x}_t)}$$

Set-type assumptions

Definitions

$v_{t|t-1}(X_t)$ - predicted PHD intensity
$v_t(X_t)$ - updated PHD intensity
$\gamma_t(X_t)$ - birth RFS
$\kappa_t(\mathbf{z})$ - clutter RFS

- The birth RFS is a Poisson RFS and independent of the surviving objects RFSs.
- The clutter RFS is a Poisson RFS and independent of the object generated measurement RFSs.
- The predicted and updated multi-object RFSs are approximated by Poisson RFSs.

The PHD filter

The typical method for implementing the PHD filter is to use a *Gaussian mixture*, basically a straight-forward expansion from the single Gaussian used in the single-target filters seen earlier!

$$v_{t-1}(X) = \sum_{i=1}^{J_{t-1}} w_{t-1}^{(i)} \mathcal{N}(X; m_{t-1}^{(i)}, P_{t-1}^{(i)})$$

Note

J_{t-1} - number of Gaussian components
$w_{t-1}^{(i)}$ - component weight
$m_{t-1}^{(i)}$ - Gaussian mean
$P_{t-1}^{(i)}$ - Gaussian (co)variance

The PHD filter

We need to keep an eye on the number of Gaussian components being used to approximate the multi-target density. We could be limited by memory allocation, or processing time available. Two methods are built in to the PHD filter to help with this -

The PHD filter

We need to keep an eye on the number of Gaussian components being used to approximate the multi-target density. We could be limited by memory allocation, or processing time available. Two methods are built in to the PHD filter to help with this -

- Pruning - Gaussian components with low weight are deleted from the mixture

The PHD filter

We need to keep an eye on the number of Gaussian components being used to approximate the multi-target density. We could be limited by memory allocation, or processing time available. Two methods are built in to the PHD filter to help with this -

- Pruning - Gaussian components with low weight are deleted from the mixture
- Merging - Gaussians that are close to one another or substantially overlap are merged together

The PHD filter

We need to keep an eye on the number of Gaussian components being used to approximate the multi-target density. We could be limited by memory allocation, or processing time available. Two methods are built in to the PHD filter to help with this -

- Pruning - Gaussian components with low weight are deleted from the mixture
- Merging - Gaussians that are close to one another or substantially overlap are merged together

Important!

Avoiding the data association problem gives us a very fast algorithm, but a major drawback in information! Without developing some extra modules to "bolt on" at the end, we won't have a history available to us and *target tracks cannot be drawn!*

Variants and extensions

- EK-PHD and UK-PHD - same as their single-target counterparts, allow for non-linearities in the tracking problem

Variants and extensions

- EK-PHD and UK-PHD - same as their single-target counterparts, allow for non-linearities in the tracking problem
- SMC-PHD - replacing the Gaussian mixture with a large set of particles. The particles are clustered at each time-step for state extraction.

Variants and extensions

- EK-PHD and UK-PHD - same as their single-target counterparts, allow for non-linearities in the tracking problem
- SMC-PHD - replacing the Gaussian mixture with a large set of particles. The particles are clustered at each time-step for state extraction.
- Labelled-RFS methods - first RFS-type filters to have labelled components allowing for association and tracking, but are however *computationally expensive*!

Variants and extensions

- EK-PHD and UK-PHD - same as their single-target counterparts, allow for non-linearities in the tracking problem
- SMC-PHD - replacing the Gaussian mixture with a large set of particles. The particles are clustered at each time-step for state extraction.
- Labelled-RFS methods - first RFS-type filters to have labelled components allowing for association and tracking, but are however *computationally expensive*!
- Higher-order filters (Panjer PHD, CPHD) - more than just the first-order moment propagated, can also include variance information for the cardinality. Also allows *flexibility in birth and clutter modelling*.

What are vector-type methods?

In contrast to the RFS methods discussed previously, we now represent the state vectors and sensor measurements as *random vectors*. This immediately gives us an advantage over the *unlabelled* approaches found in RFS, as we can explicitly define target IDs and labels in a simple way.

Vector-type methods -

- explicitly resolve the data association problem
- inherently have a track history available
- allow for *deferred logic decisions*

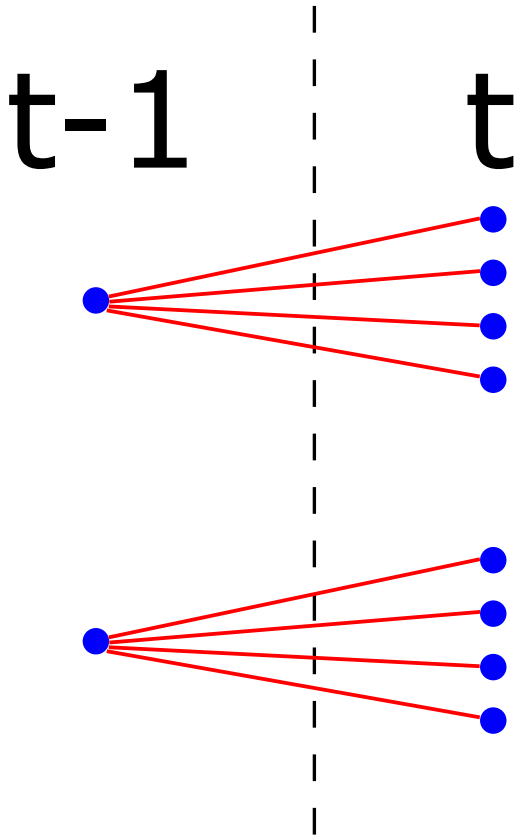
Multiple Hypothesis Tracking (MHT)

Multiple Hypothesis Tracking is one of the oldest MTT algorithms, and is also one of the most robust. It considers the association of *sequences* of measurements, and evaluates the probability, or likelihood, of *all possible* association hypotheses.

There are two distinct versions of Multiple Hypothesis Tracking (MHT) -

- Hypothesis Oriented MHT (HOMHT)
- Track Oriented MHT (TOMHT)

We will work with HOMHT today; TOMHT is distinctly non-Bayesian and non-convex!



Multiple Hypothesis Tracking (MHT)

HOMHT builds hypotheses directly from the measurements that it receives from the processing chain. Tracks are then extracted from the (probably much larger!) set of hypotheses.

Multiple Hypothesis Tracking (MHT)

HOMHT builds hypotheses directly from the measurements that it receives from the processing chain. Tracks are then extracted from the (probably much larger!) set of hypotheses.

The *set of hypotheses* Ω_t at time t is generated by augmenting the previous set of hypotheses with each new measurement, and with all *feasible* associations.

Multiple Hypothesis Tracking (MHT)

HOMHT builds hypotheses directly from the measurements that it receives from the processing chain. Tracks are then extracted from the (probably much larger!) set of hypotheses.

The *set of hypotheses* Ω_t at time t is generated by augmenting the previous set of hypotheses with each new measurement, and with all *feasible* associations.

Feasible associations for a current measurement are -

1. A continuation of a previous target or track
2. A false alarm
3. A newly-detected target

New Definition

Ω_t - set of hypotheses at time t

Multiple Hypothesis Tracking (MHT)

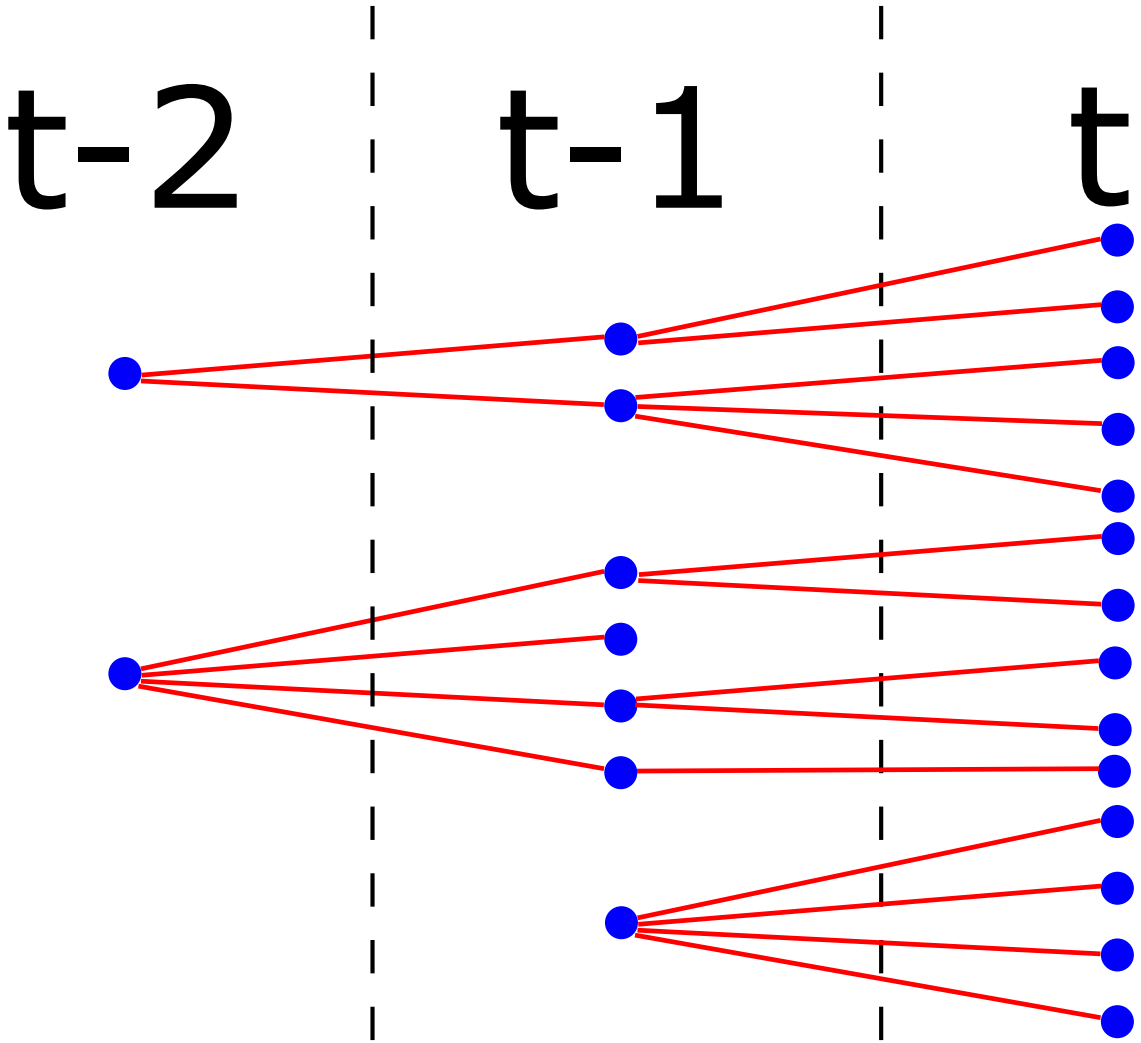
Recall

Recall the correlation gating procedure discussed earlier; a very important step for MHT!

Consider one output track from the previous time $t-1$. At time t, we will first perform *gating* to see which new measurements lie inside the threshold for this track. A new *hypothesis* is created between the track, and all of the measurements that lie inside the threshold, giving us a new set of hypotheses $\Omega_{t,k}$.

Multiple Hypothesis Tracking (MHT)

By expanding all of the hypotheses over time, the result is a tree-like structure, with many branches.



Multiple Hypothesis Tracking (MHT)

For each of our branches (hypotheses), we then -

Multiple Hypothesis Tracking (MHT)

For each of our branches (hypotheses), we then -

1. Update each of the branches using a *standard filter* (KF, EKF etc...)

Multiple Hypothesis Tracking (MHT)

For each of our branches (hypotheses), we then -

1. Update each of the branches using a *standard filter* (KF, EKF etc...)
2. Perform hypothesis management - very important!
   - Pruning hypotheses with low probability
   - Merging hypotheses with states very close to one another
   - Time windowing to deal with only $t - s$ time-steps

Multiple Hypothesis Tracking (MHT)

For each of our branches (hypotheses), we then -

1. Update each of the branches using a *standard filter* (KF, EKF etc...)
2. Perform hypothesis management - very important!
   - Pruning hypotheses with low probability
   - Merging hypotheses with states very close to one another
   - Time windowing to deal with only $t - s$ time-steps
3. Selection of the most probable hypotheses - time consuming!
   - Exhaustive search across all hypotheses...

Multiple Hypothesis Tracking (MHT)

For each of our branches (hypotheses), we then -

1. Update each of the branches using a *standard filter* (KF, EKF etc...)
2. Perform hypothesis management - very important!
   - Pruning hypotheses with low probability
   - Merging hypotheses with states very close to one another
   - Time windowing to deal with only $t - s$ time-steps
3. Selection of the most probable hypotheses - time consuming!
   - Exhaustive search across all hypotheses...

Overall, the complexity of MHT scales exponentially with time.

Note

s - tunable parameter for length of sliding window; how far back in time should we be looking at?

Message Passing (MP)

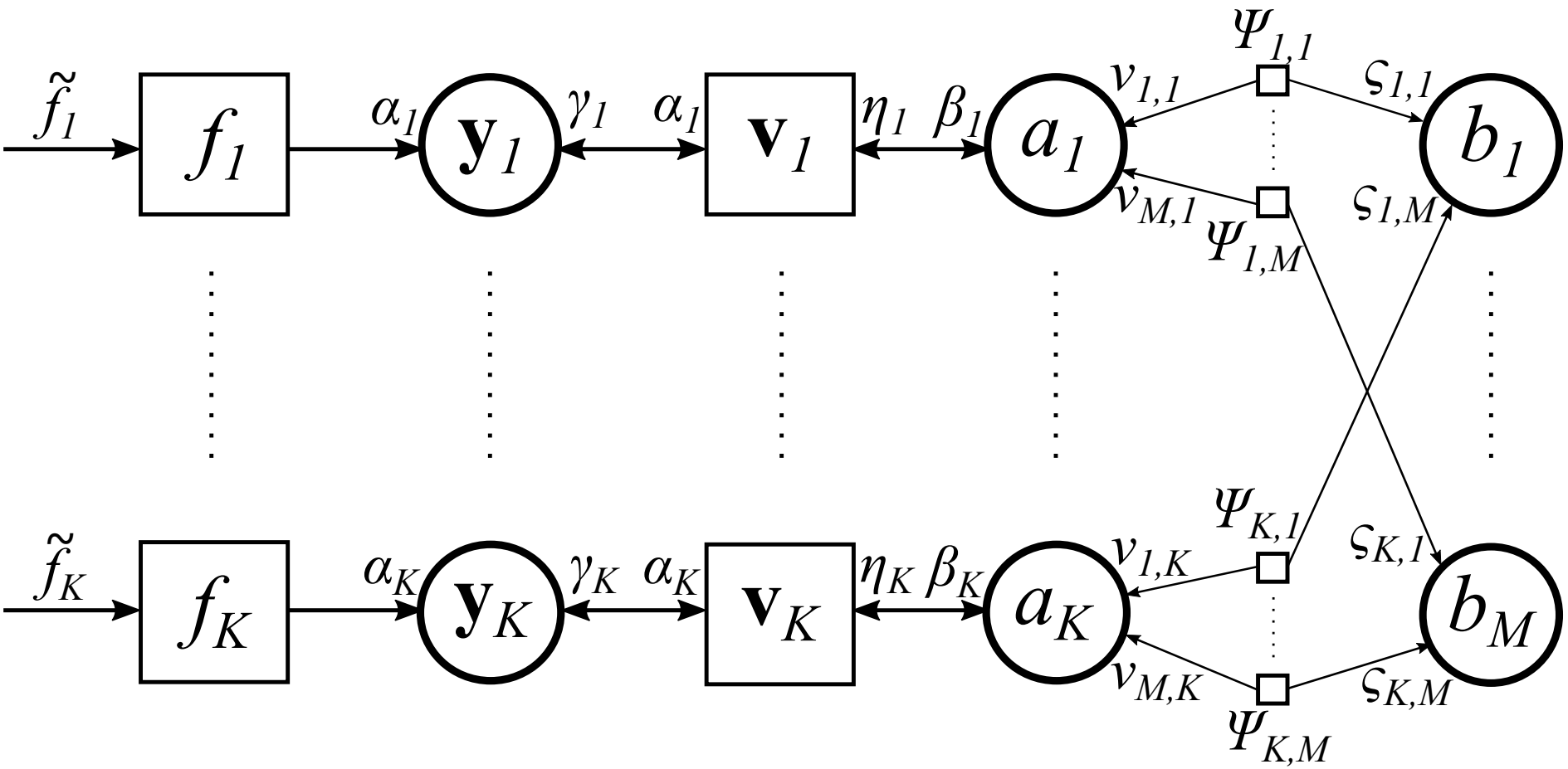
Many of the techniques we have covered so far have their limitations and can be computationally expensive. A recent development in the area of MTT has revolved around Message Passing (MP), or Belief Propagation (BP).

Message Passing (MP)

Many of the techniques we have covered so far have their limitations and can be computationally expensive. A recent development in the area of MTT has revolved around MP, or BP.

MP algorithms have been around for many years, but it's only been recently that they have been discovered for MTT applications. The fundamental parts of the framework are factor graphs (see next slide!) and the SPA.

Message Passing (MP)



Message Passing (MP)

The SPA is an *approximation* of Bayesian inference, and allows for flexibility between accuracy and execution time. A number of tunable parameters are available in the implementation such as -

Message Passing (MP)

The SPA is an *approximation* of Bayesian inference, and allows for flexibility between accuracy and execution time. A number of tunable parameters are available in the implementation such as -

- a *maximum* number of targets to be tracked
- a *variable* number of particles representing each target state
- optional resampling strategies at each time-step to refresh distributions
- the number of *message passing iterations* to perform

Message Passing (MP)

The SPA is an *approximation* of Bayesian inference, and allows for flexibility between accuracy and execution time. A number of tunable parameters are available in the implementation such as -

- a *maximum* number of targets to be tracked
- a *variable* number of particles representing each target state
- optional resampling strategies at each time-step to refresh distributions
- the number of *message passing iterations* to perform

We use particle representations of all *messages and beliefs*, such that non-linear, non-Gaussian scenarios can be accounted for directly!

Main advantage!

The SPA can calculate the marginal PDFs much faster than performing direct marginalisation! This is where the main speed-up comes from.

Message Passing (MP)

To get this major speed-up, we need to assume that the joint posterior pdf $f(X|Z_{1:t})$ can be seen as a product of C lower-dimensional factors -

$$f(X|Z_{1:t}) = \prod_{c=1}^{C} \psi_c(X^{(c)})$$

Message Passing (MP)

To get this major speed-up, we need to assume that the joint posterior pdf $f(X|Z_{1:t})$ can be seen as a product of C lower-dimensional factors -

$$f(X|Z_{1:t}) = \prod_{c=1}^{C} \psi_c(X^{(c)})$$

The overall implementation follows these steps (not too dissimilar to other previous techniques!)

1. Initialisation
2. Prediction
3. Measurement Evaluation
4. Data Association
5. Measurement Update
6. State Extraction

Questions and Answers!

Based on what you know now, are there any challenges you still see in MTT?

- Specific problems?
- Other algorithms?
- Novel applications?

What are IMMs?

Implementing a multiple model approach gives us a versatile tool for adaptive systems where the behaviour pattern can change significantly over time.

What are IMMs?

Implementing a multiple model approach gives us a versatile tool for adaptive systems where the behaviour pattern can change significantly over time.

Imagine a target that is travelling in a straight line with NCV, and then starts making a sharp turn. A typical tracker would lose the target after this point, as the motion does not fit the model any more!

What are IMMs?

Implementing a multiple model approach gives us a versatile tool for adaptive systems where the behaviour pattern can change significantly over time.

Imagine a target that is travelling in a straight line with NCV, and then starts making a sharp turn. A typical tracker would lose the target after this point, as the motion does not fit the model any more!

Let's now introduce a solution to this problem, that allows us some more flexibility in the choice of motion models, namely the Interacting Multiple Model (IMM).

What are IMMs?

IMMs are a cost-effective method for tracking targets that are likely to switch between different motion models as the scenario develops. The algorithm is broken down in to four main steps -

What are IMMs?

IMMs are a cost-effective method for tracking targets that are likely to switch between different motion models as the scenario develops. The algorithm is broken down in to four main steps -

1. Interaction - Mixing of previous mode-conditioned state estimates and covariances using the mixing probabilities, to initialise the next cycle of each mode-conditioned filter.

What are IMMs?

IMMs are a cost-effective method for tracking targets that are likely to switch between different motion models as the scenario develops. The algorithm is broken down in to four main steps -

1. Interaction - Mixing of previous mode-conditioned state estimates and covariances using the mixing probabilities, to initialise the next cycle of each mode-conditioned filter.
2. Mode-conditioned filtering - Calculation of state estimates and covariances conditioned on a chosen mode.

What are IMMs?

IMMs are a cost-effective method for tracking targets that are likely to switch between different motion models as the scenario develops. The algorithm is broken down in to four main steps -

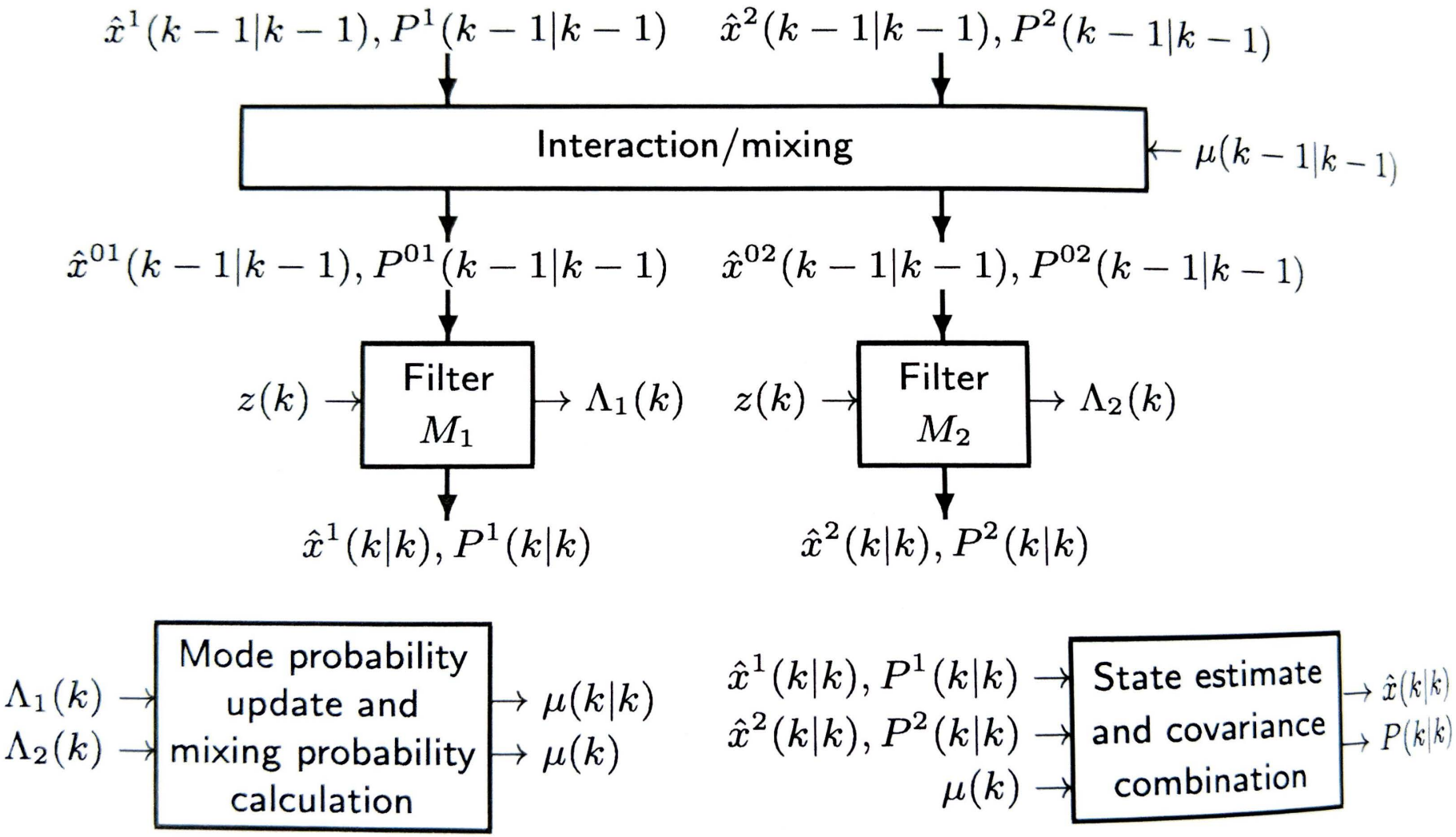
1. Interaction - Mixing of previous mode-conditioned state estimates and covariances using the mixing probabilities, to initialise the next cycle of each mode-conditioned filter.
2. Mode-conditioned filtering - Calculation of state estimates and covariances conditioned on a chosen mode.
3. Probability evaluations - Computation of the mixing and updated mode probabilities.

What are IMMs?

IMMs are a cost-effective method for tracking targets that are likely to switch between different motion models as the scenario develops. The algorithm is broken down in to four main steps -

1. Interaction - Mixing of previous mode-conditioned state estimates and covariances using the mixing probabilities, to initialise the next cycle of each mode-conditioned filter.
2. Mode-conditioned filtering - Calculation of state estimates and covariances conditioned on a chosen mode.
3. Probability evaluations - Computation of the mixing and updated mode probabilities.
4. State estimation - Combination of the latest mode-conditioned state estimates and covariances.

IMM Structure



What is data fusion?

Modern sensor platforms often carry a large number, and variety, of sensors onboard. Having more sensors available can be very useful for improving the accuracy, timeliness and robustness of our MTT. The idea of combining the different information available to us is generalised to the idea of *data fusion*.

What is data fusion?

Modern sensor platforms often carry a large number, and variety, of sensors onboard. Having more sensors available can be very useful for improving the accuracy, timeliness and robustness of our MTT. The idea of combining the different information available to us is generalised to the idea of *data fusion*.

The main advantages for data fusion include -

- Increase in tracking accuracy

What is data fusion?

Modern sensor platforms often carry a large number, and variety, of sensors onboard. Having more sensors available can be very useful for improving the accuracy, timeliness and robustness of our MTT. The idea of combining the different information available to us is generalised to the idea of *data fusion*.

The main advantages for data fusion include -

- Increase in tracking accuracy
- Target tracks can be updated more often

What is data fusion?

Modern sensor platforms often carry a large number, and variety, of sensors onboard. Having more sensors available can be very useful for improving the accuracy, timeliness and robustness of our MTT. The idea of combining the different information available to us is generalised to the idea of *data fusion*.

The main advantages for data fusion include -

- Increase in tracking accuracy
- Target tracks can be updated more often
- Missed detections can be recognised*
- False tracks stopped*

The JDL Data Fusion Levels

In 2004, four levels were defined by a US Department of Defence committee, based on the levels of abstraction, and problem space complexity.

The JDL Data Fusion Levels

In 2004, four levels were defined by a US Department of Defence committee, based on the levels of abstraction, and problem space complexity.

- *Level 0:* Estimating states of sub-object entities (waveforms, ...)

The JDL Data Fusion Levels

In 2004, four levels were defined by a US Department of Defence committee, based on the levels of abstraction, and problem space complexity.

- *Level 0:* Estimating states of sub-object entities (waveforms, ...)
- *Level 1:* Estimating states of discrete physical objects (target track)

The JDL Data Fusion Levels

In 2004, four levels were defined by a US Department of Defence committee, based on the levels of abstraction, and problem space complexity.

- *Level 0:* Estimating states of sub-object entities (waveforms, ...)
- *Level 1:* Estimating states of discrete physical objects (target track)
- *Level 2:* Estimating relationships between objects (group tracking)

The JDL Data Fusion Levels

In 2004, four levels were defined by a US Department of Defence committee, based on the levels of abstraction, and problem space complexity.

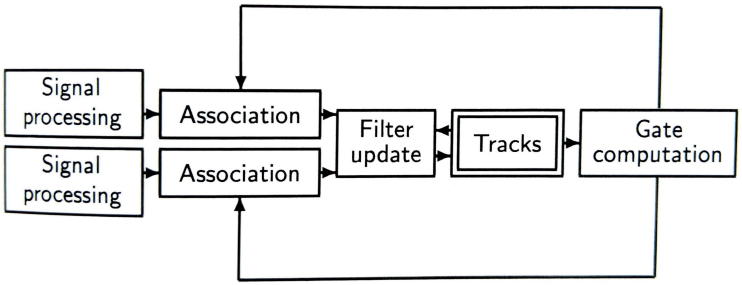
- *Level 0:* Estimating states of sub-object entities (waveforms, ...)
- *Level 1:* Estimating states of discrete physical objects (target track)
- *Level 2:* Estimating relationships between objects (group tracking)
- *Level 3:* Estimating overall impact (mission planning, objectives)

What is data fusion?

We can perform data fusion in two ways; *centralised* or *decentralised*.

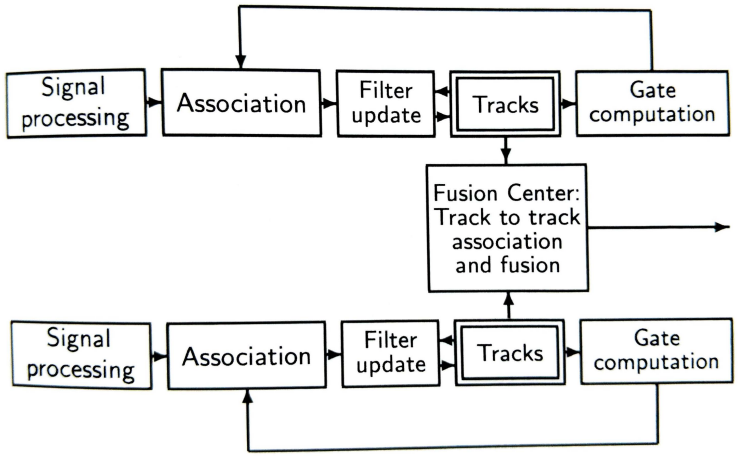
Centralised

The Fusion Centre (FC) has access to all of the raw measurement data available from all sensors.



Decentralised

The FC has access to the tracks generated by each of the individual sensors.



What do we need?

In order to perform fusion effectively, we need to construct the problem appropriately, in much the same way as we did previously with the single-sensor examples. Ideally, we should have -

What do we need?

In order to perform fusion effectively, we need to construct the problem appropriately, in much the same way as we did previously with the single-sensor examples. Ideally, we should have -

- A well-defined and *consistent* state-space representation across all sensors;

What do we need?

In order to perform fusion effectively, we need to construct the problem appropriately, in much the same way as we did previously with the single-sensor examples. Ideally, we should have -

- A well-defined and *consistent* state-space representation across all sensors;
- Highly accurate knowledge of all sensor positions and orientations;

What do we need?

In order to perform fusion effectively, we need to construct the problem appropriately, in much the same way as we did previously with the single-sensor examples. Ideally, we should have -

- A well-defined and *consistent* state-space representation across all sensors;
- Highly accurate knowledge of all sensor positions and orientations;
- Observation models for all sensors;

What do we need?

In order to perform fusion effectively, we need to construct the problem appropriately, in much the same way as we did previously with the single-sensor examples. Ideally, we should have -

- A well-defined and *consistent* state-space representation across all sensors;
- Highly accurate knowledge of all sensor positions and orientations;
- Observation models for all sensors;
- No communication/timing delays;

What do we need?

In order to perform fusion effectively, we need to construct the problem appropriately, in much the same way as we did previously with the single-sensor examples. Ideally, we should have -

- A well-defined and *consistent* state-space representation across all sensors;
- Highly accurate knowledge of all sensor positions and orientations;
- Observation models for all sensors;
- No communication/timing delays;
- A fully synchronised network of sensors.

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

1. Initialise a common MTT distribution using the first set of measurements available.

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

1. Initialise a common MTT distribution using the first set of measurements available.
2. Predict ahead to the next time-step that measurements are available.

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

1. Initialise a common MTT distribution using the first set of measurements available.
2. Predict ahead to the next time-step that measurements are available.
3. Correlation gating/association (algorithm dependent!).

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

1. Initialise a common MTT distribution using the first set of measurements available.
2. Predict ahead to the next time-step that measurements are available.
3. Correlation gating/association (algorithm dependent!).
4. Update using appropriate observation model for sensor.

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

1. Initialise a common MTT distribution using the first set of measurements available.
2. Predict ahead to the next time-step that measurements are available.
3. Correlation gating/association (algorithm dependent!).
4. Update using appropriate observation model for sensor.
5. Extract any declared target states.

How do we do it?

For centralised fusion, we need to have a form of database, containing knowledge of each of the sensors we wish to perform fusion between. This includes observation models, expected false alarm rates and so forth.

A simple way to perform data fusion is to perform a typical predict/update cycle on a common GM or particle set for whichever sensor is ready. We end up with the following steps -

1. Initialise a common MTT distribution using the first set of measurements available.
2. Predict ahead to the next time-step that measurements are available.
3. Correlation gating/association (algorithm dependent!).
4. Update using appropriate observation model for sensor.
5. Extract any declared target states.
6. Track management (algorithm dependent!).

How do we do it?

Decentralised fusion works in a slightly different way, in that each individual sensor performs it's own tracking routine, and the outputs from each MTT algorithm are fused. Much less information is shared between sensors, and we don't need a common distribution.

How do we do it?

Decentralised fusion works in a slightly different way, in that each individual sensor performs it's own tracking routine, and the outputs from each MTT algorithm are fused. Much less information is shared between sensors, and we don't need a common distribution.

1. Perform your favourite MTT routine in each of the sensors.

How do we do it?

Decentralised fusion works in a slightly different way, in that each individual sensor performs it's own tracking routine, and the outputs from each MTT algorithm are fused. Much less information is shared between sensors, and we don't need a common distribution.

1. Perform your favourite MTT routine in each of the sensors.
2. Target/track information is shared to a central node.

How do we do it?

Decentralised fusion works in a slightly different way, in that each individual sensor performs it's own tracking routine, and the outputs from each MTT algorithm are fused. Much less information is shared between sensors, and we don't need a common distribution.

1. Perform your favourite MTT routine in each of the sensors.
2. Target/track information is shared to a central node.
3. Perform track-to-track association/fusion to get common picture.

So which is better?

Yet again, there is no straight-forward answer! It comes down to *design choice* and *physical limitations*.

Centralised

- Largely suited for *single platform* scenarios.
- Lots of data required to be transmitted.
- Registration issues can be resolved.

Decentralised

- Suited for larger-scale scenarios, or over long distances.
- Less communication costs involved.
- Potential for systematic bias!

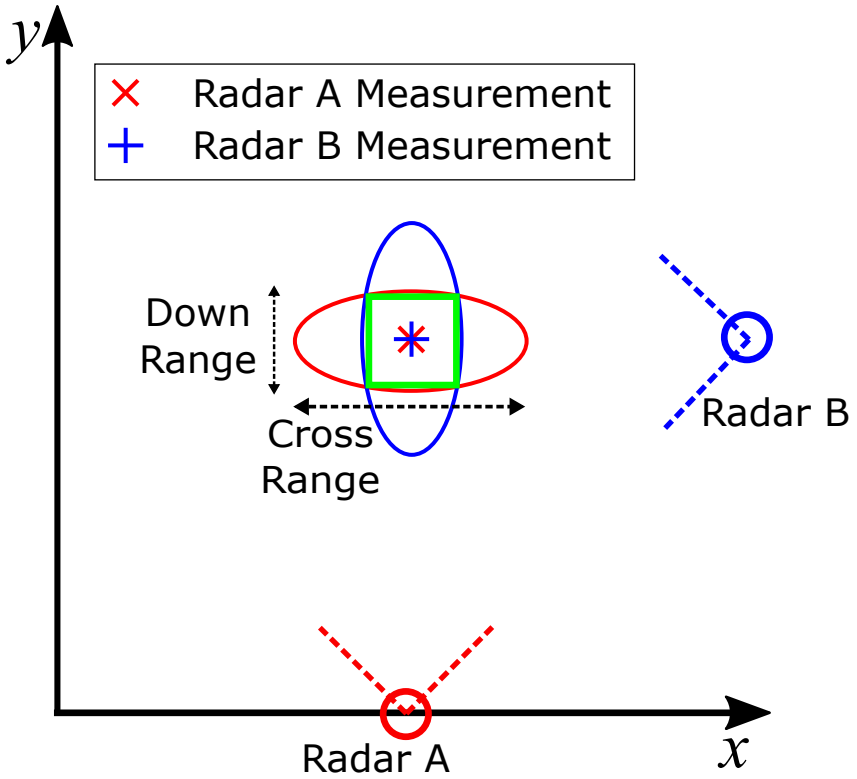
Ongoing Research

Our research currently focuses on *scalable, dynamic and distributed inference*, and looks to attack the problems from both theoretical and practical perspectives. Message Passing (MP) on graphical models is a powerful framework for solving many different problems, and is suitable for solving MTT as we've seen today!

MTT and data fusion problems can also be expanded to allow estimation of other parameters that are related to the problem.

Message Passing for Joint Registration and Tracking in Multistatic Radar

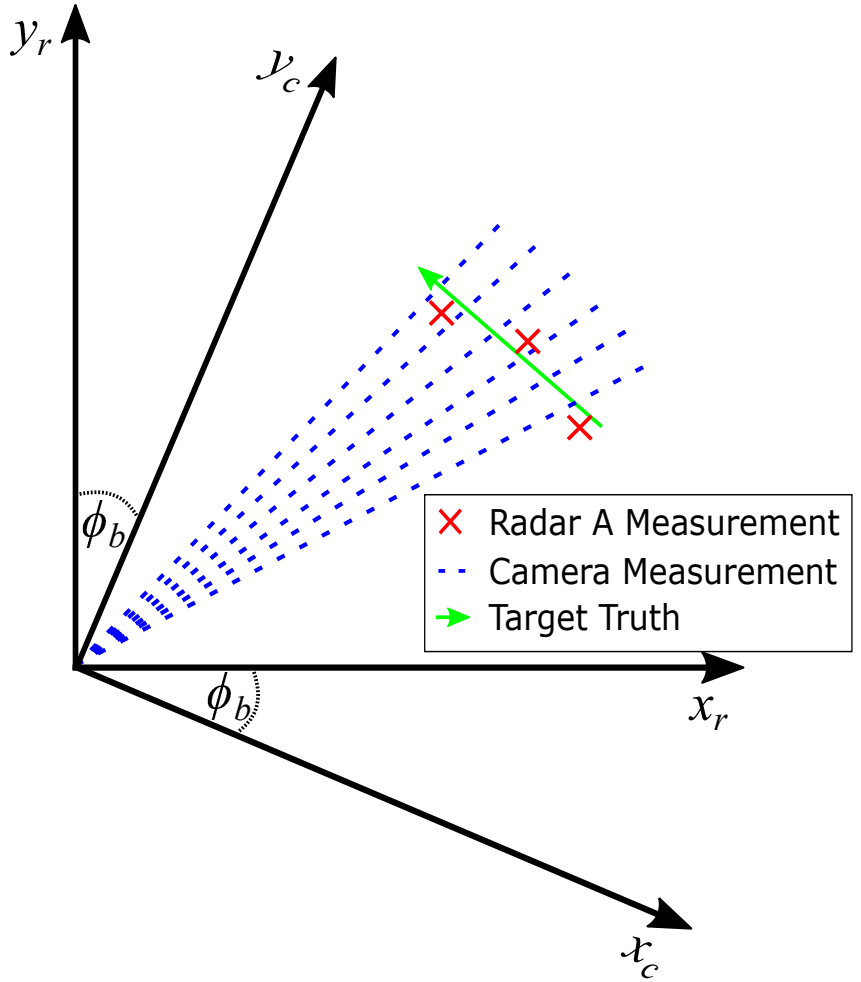
- Performing data fusion using two identical, non-colocated radars, while estimating correct registration parameters.
- Radar A is reference sensor, Radar B has range and azimuth biases.



Fusion 2019 - Next week!

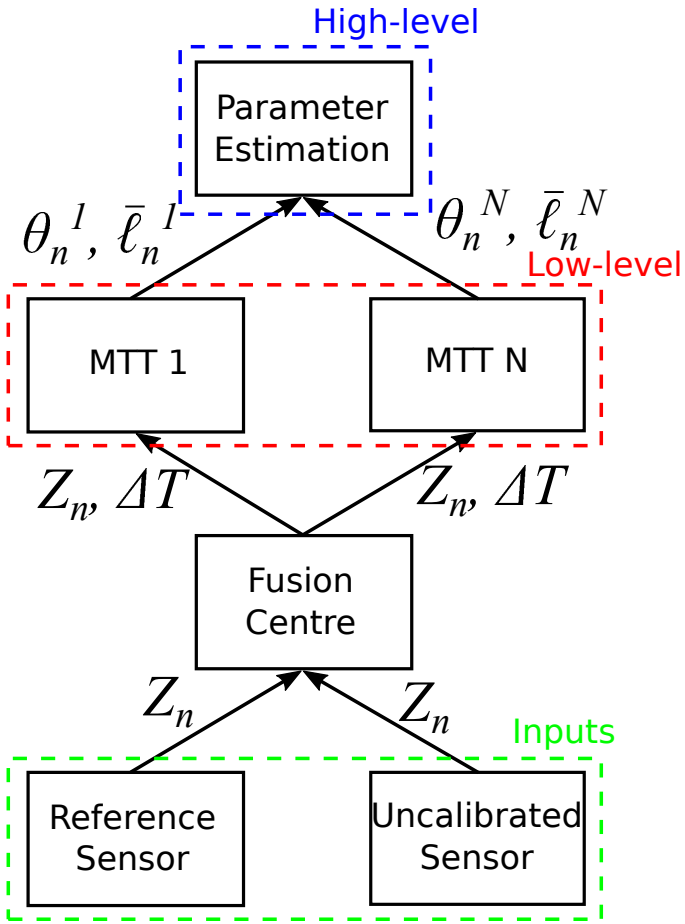
Sensor Registration and Tracking from Heterogeneous Sensors with Belief Propagation

- Performing data fusion between a radar and a camera, both attached to the same platform.
- Radar is reference sensor, camera has azimuth bias.
- Angles-only tracking; notoriously difficult problem!



How did we solve them?

We can formulate a joint estimation problem, and exploit a technique known as *hierarchical models* to solve both simultaneously.



OSPA

There are a few common methods for evaluating the accuracy of estimates an MTT system provides. One such method that appears in many pieces of literature in this area is the Optimal Subpattern Assignment (OSPA) metric. The metric is made up of a localisation error and a cardinality error between two sets X and Y, with cardinalities m and n.

$$d_p^{(c)}(X,Y) = \left[\min_{\pi \in \Pi_n} \sum_{i=1}^{m} d^{(c)}(x_i, y_{\pi(i)})^p + c^p(n-m) \right]^{\frac{1}{p}}$$

Definitions

p - order parameter, typically set at $p = 2$ for smoother distance curves.
c - cut-off distance, determines trade-off between penalising cardinality error or localisation error.

GOSPA

The OSPA metric has since been generalised, or superseded by the Generalised OSPA (GOSPA) metric which introduces an additional parameter. This allows a choice of cardinality mismatch cost, hence giving a sum of localisation errors for detected targets, and penalising missed and false targets.

$$d_p^{(c,\alpha)}(X,Y) = \left[\min_{\pi \in \Pi_n} \sum_{i=1}^{m} d^{(c)}(x_i, y_{\pi(i)})^p + \frac{c^p}{\alpha}(n-m) \right]^{\frac{1}{p}},$$

Definitions

α - tunable parameter for cardinality mismatch cost, usually set $\alpha = 2$

The End

Please send any questions to drc9@hw.ac.uk!