State estimation and tracking

Jordi Barr*, Steve Hiscocks*, Lyudmil Vladimirov†, David Cormack‡

*Dstl, †University of Liverpool, ‡Leonardo Company

29 June 2020

What are we trying to do?

We want to *infer* the *state* of an entity

To do this we make *observations* or *measurements* using some device – generally referred to as a sensor

Some facts we have to cope with:

- the state of an entity is hidden,
- measurements are noisy,
- 'incorrect' measurements can happen – the sensor returns something that can be interpreted as coming from an entity despite there being nothing there (call these *false alarms*),
- attempts at observation can result in no measurement being returned, despite the existence of an entity *(missed detections)*

We're also interested in putting together multiple state estimates to estimate an entity's *track* in the past, present or future

Applications

Early-warning radar

Air defence

Acoustics

Ship defence

Complicated situation awareness

Air ISR

Objectives

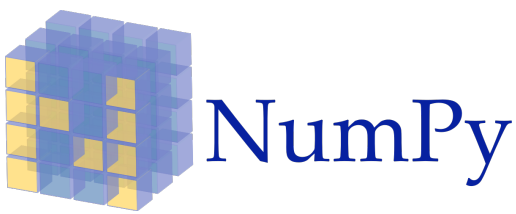
The aims of today's lesson are to give you:

- an *understanding* of the basics of state estimation and target tracking,
- an *appreciation* of the defence situations which benefit from these techniques,
- some *intuition* surrounding the strengths and weaknesses of various estimation, tracking and association schemes

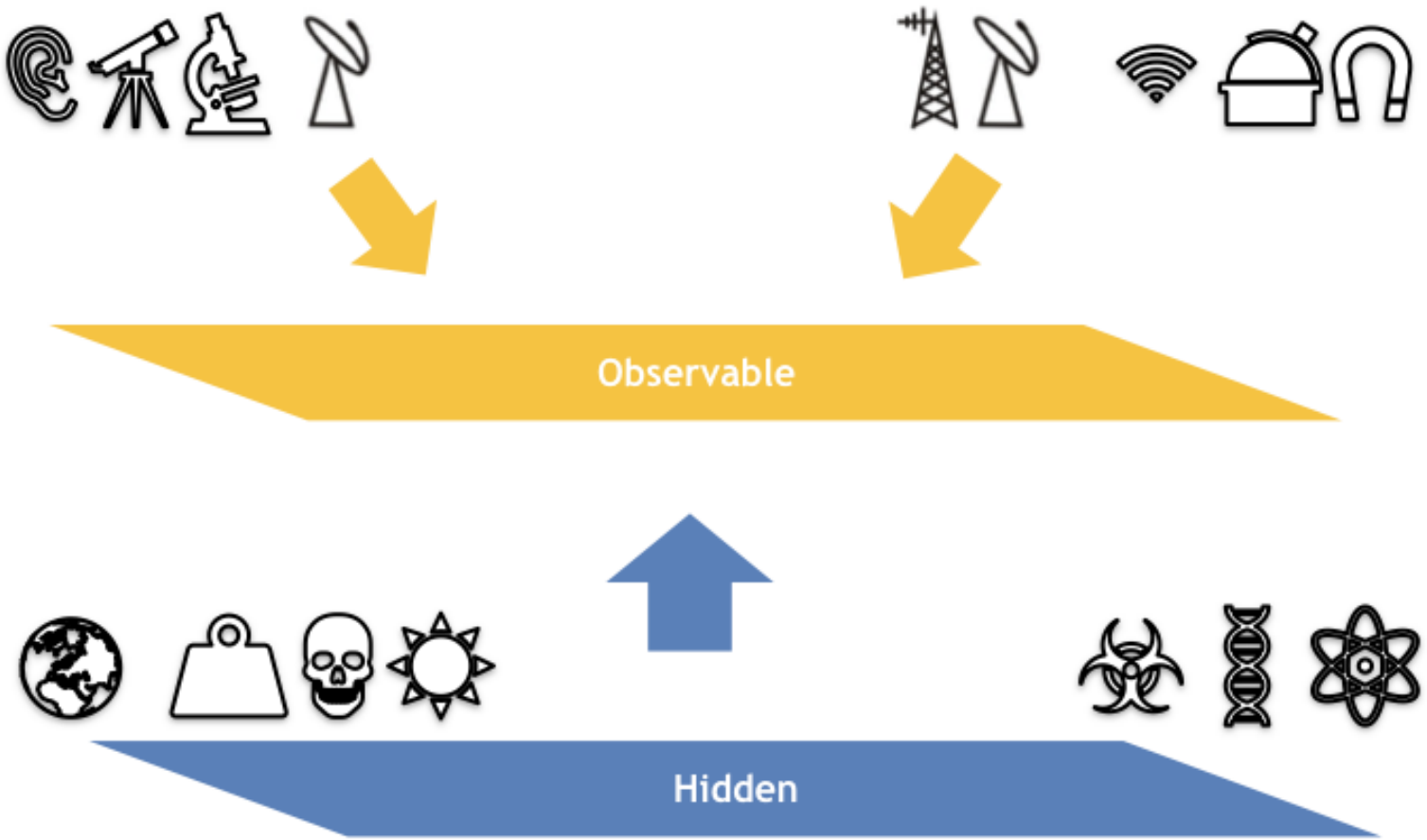
Stone Soup

We'll be making use of *Stone Soup*, an open-source tracking and state estimation framework

- Core modules built from Python
- Object oriented
- extensible through many libraries
- interactive - we'll be using Jupyter notebooks



An abstract view



Definitions

State estimation

the process of inferring the intrinsic and extrinsic properties of an entity, or entities, by way of data gathered using a sensor

Tracking

inferring the state of an entity over a period of time, or other sequence, using accumulated sensed data, processed synchronously or asynchronously

Multi-target tracking

jointly inferring the state of multiple entities over a period of time using accumulated sensed data from multiple sensors, processed synchronously or asynchronously

Notation

$\mathbf{x}_k$	a hidden *state vector* at (time) k
$\mathbf{z}_k$	a *measurement* at k
	Measurements can incorporate false detections, clutter, and may be missed

Our goal is often to infer $\mathbf{x}_k$ - the state of the thing we're interested in at some point from a sequence of measurements up to that point, $\mathbf{z}_{1:k}$

$f(\mathbf{x}_k \vert \mathbf{x}_{k-1}, \mathbf{w}_k)$	a function determining the transition from state at (time) $k-1$ to state at (time) k (Markovian assumption) in the presence of some noise process $\mathbf{w}_k$
$h(\mathbf{z}_k \vert \mathbf{x}_k, \boldsymbol{\nu}_k)$:	a function telling us how the sensor responds to the state, $\mathbf{x}_k$ at k, with noise according to $\boldsymbol{\nu}_k$

Examples of $\mathbf{x}$, $\mathbf{z}$, $f(.)$ and $h(.)$

$\mathbf{x}$	$\mathbf{z}$
temperature	photon flux in the mid-IR
location	range, bearing
size	angular extent
radioactivity	radiation flux
etc	etc

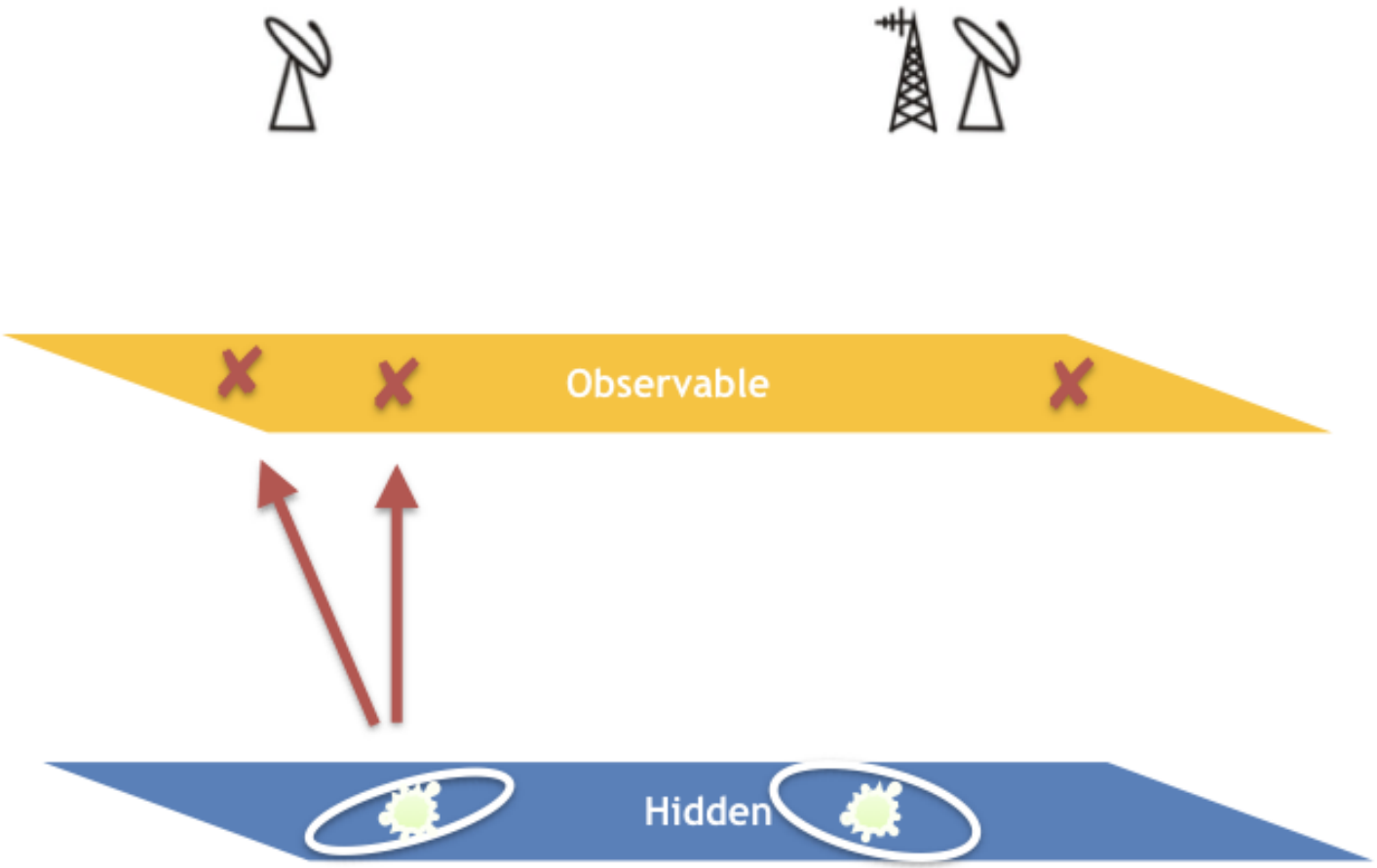
Examples of $f(.)$:

- $\mathbf{v} = \mathbf{u} + \mathbf{a}t$
- conservation laws (e.g. $m^{tot}\mathbf{v}_k^{tot} = m^1\mathbf{v}_{k-1}^1 + m_{k-1}^2\mathbf{v}_{k-1}^2$)
- solutions to diffusion equations

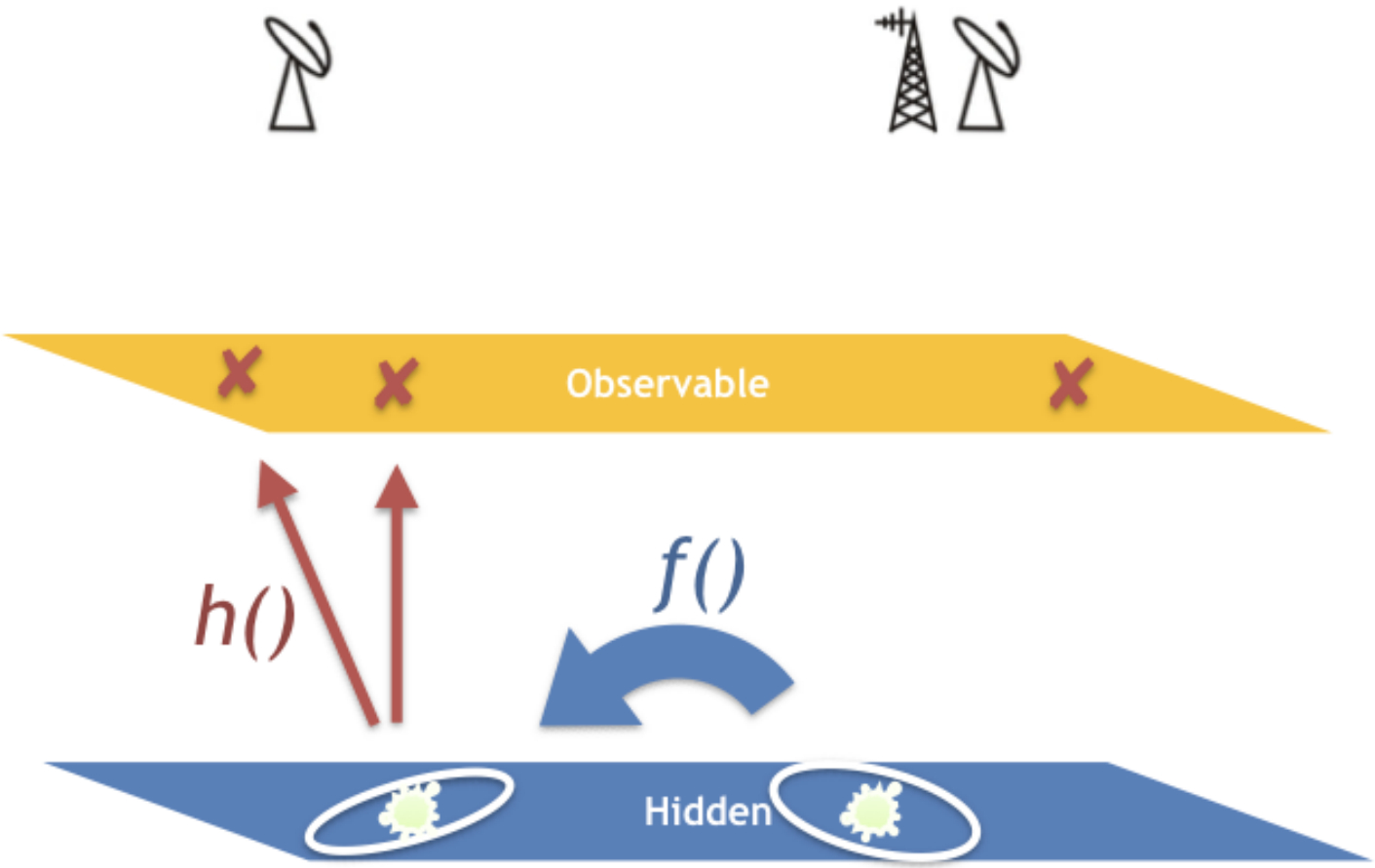
Examples of $h(.)$:

- in a 2-d range-bearing sensor $\mathbf{z} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan(\frac{y}{x}) \end{bmatrix}$

That abstract view, again



That abstract view, again

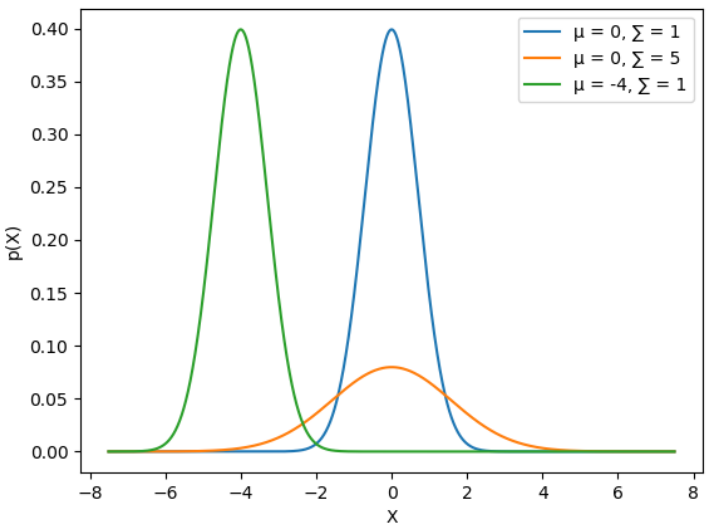


Probability distributions

We are looking for a way of estimating an inherently *stochastic* process

It's often not possible to know, or measure, the precise state of an object – the transition and sensing processes may be very noisy.

Probability density functions (pdfs) are extremely useful in that they give us access to more state information than just the 'most likely true' value (e.g. uncertainty, multi-modality, etc). They also come with a handy algebra we can use.



Some key tools

Recall from yesterday, or your previous statistics learning

Law of total probability (marginalisation)

$$P(X) = \sum_{i=1}^{N} P(X|Y_i)P(Y_i) \text{ or } p(x) = \int_X p(x|x')p(x')dx'$$

Bayes' rule

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

More probability

There's much power and depth to stochastic calculus. For the moment we merely note some notational equivalence:

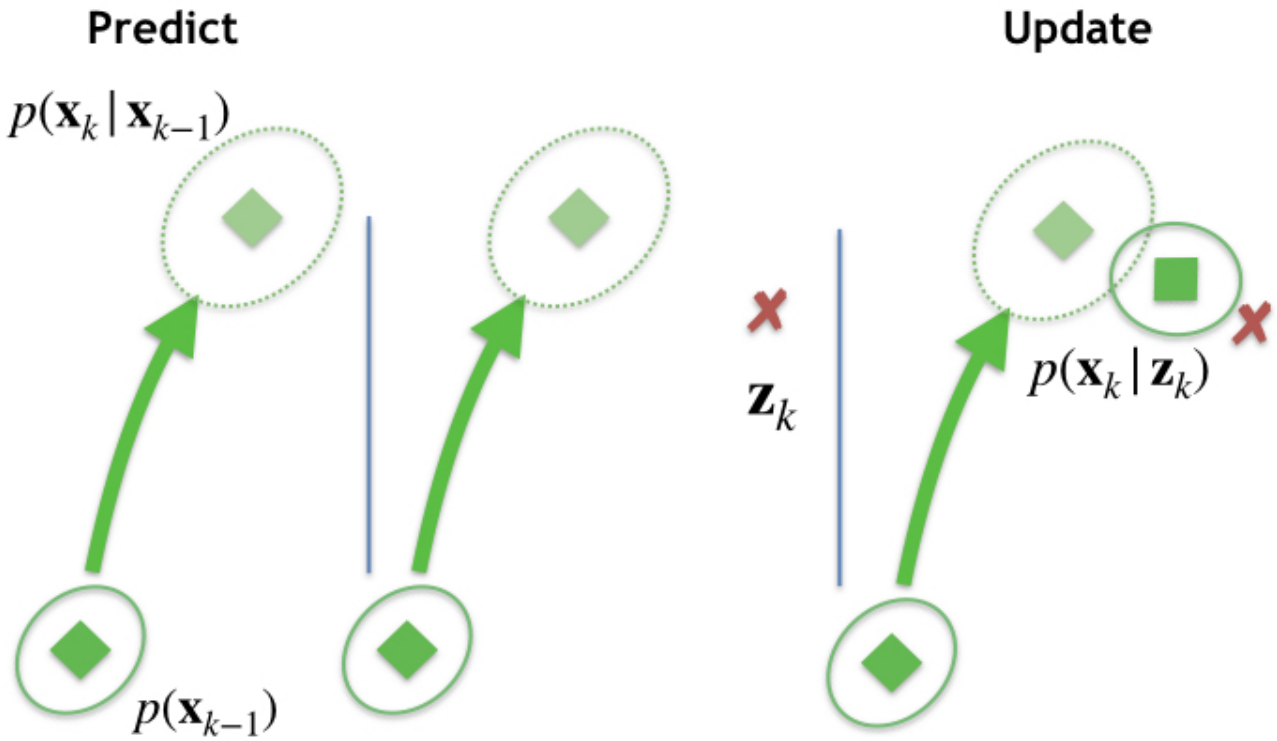
$\mathbf{x}_{k|k-1} = f_k(\mathbf{x}_{k-1}, \mathbf{w}_k)$, $\mathbf{w}_k \sim \mathcal{Q}_k(\eta_1, \eta_2, ...)$,
or $\mathbf{x}_{k|k-1} \sim p(\mathbf{x}_k|\mathbf{x}_{k-1}, \boldsymbol{\eta}_k)$

$\mathbf{z}_k = h_k(\mathbf{x}_k, \boldsymbol{\nu}_k)$, $\boldsymbol{\nu}_k \sim \mathcal{R}_k(\zeta_1, \zeta_2, ...)$,
or $\mathbf{z}_k \sim p(\mathbf{z}_k|\mathbf{x}_k, \boldsymbol{\zeta}_k)$

where $\mathcal{Q}$ and $\mathcal{R}$ are stochastic processes with parameters $\boldsymbol{\eta}$ and $\boldsymbol{\zeta}$ respectively, which generate instances of transition noise and measurement noise represented by vectors $\mathbf{w}$ and $\boldsymbol{\nu}$.

Application to state estimation

State estimation operates by way of a recursive process. Starting with the *prior* estimate from a previous step, a *prediction* is used to estimate the state at the current step. This is then *updated* with a measurement to yield the *posterior* state estimate.



Application to state estimation

Adapting the previous equations slightly:

The Chapman-Kolmogorov equation (predict)

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int_{-\infty}^{\infty} p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})\mathrm{d}\mathbf{x}_{k-1}$$

Bayes' rule (update)

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k)}$$

where:
$p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$: the probability of the current state given (conditioned on) all observations up to some previous step,
$p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is the probability of the current state given the set of observations up to k.

The Kalman Filter

Due to Rudolph E. Kálmán; but also Bucy, Swerling, Stratonovich. The key insight is that in linear-Gaussian regimes the solution to the recursive equations is analytic. (There are various derivations – feel free.)

Predict

$$\mathbf{x}_{k|k-1} = F_k \mathbf{x}_{k-1} + B_k \mathbf{u}_k, \; P_{k|k-1} = F_k P_{k-1} F_k^T + Q_k$$

Update

$$\mathbf{y}_k = \mathbf{z}_k - H_k \mathbf{x}_{k|k-1}, \; S_k = H_k P_{k-1} H_k^T + R_k$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1}$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + K_k \mathbf{y}_k, \; P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

The Kalman Filter

We refer to $\mathbf{x}_{k-1}$, P_{k-1} as the prior state and covariance (or error, or uncertainty) at step $k-1$. (These are sometimes rendered as $\mathbf{x}_{k-1|k-1}$, $P_{k-1|k-1}$.)

The predicted values of these quantities at k are $\mathbf{x}_{k|k-1}$ and $P_{k|k-1}$ where the subscripts can be taken to mean, "the value at k given all information collected up to step $k-1$." The update step yields the posterior values $\mathbf{x}_{k|k}$, $P_{k|k}$.

$\mathbf{y}$ and S are known as the *innovation* and *innovation covariance* respectively. K is called the *Kalman gain* and can be interpreted as a measure of how much weight is placed on the measurement, as opposed to the prediction. For low values of K the posterior value tends to the prediction (when R is large you don't trust the measurement too much). When R is small (measurements are more accurate) K approaches a limit governed by the uncertainty in the dynamics only.

The Kalman Filter

Note that the transition model $f(\cdot)$ and the observation model $h(\cdot)$ are now matrix multiplications of the state vector $\mathbf{x}$ and observation $\mathbf{z}$ via F and H.

The stochasticity due to the transition ($\mathcal{Q}$) and observation ($\mathcal{R}$) is accounted for by the covariance matrices Q and R, reflecting the fact that we are now assumeing these are multivariate normal distributed processes.

A term to account for control input is included ($B_k\mathbf{u}_k$). This is assumed noiseless and included for completeness. We drop it in subsequent analysis as for many tracking examples the target of interest isn't something we can manipulate directly. Clearly it's of more direct concern in the control theory literature.

It's interesting to note that whereas the posterior mean estimate is a function of the measurement, the posterior uncertainty isn't.

Practical

Jupyter notebook 1: Kalman filter

Non-linear estimation

The Kalman filter is used in a surprising array of applications. It's often quite convenient to linearise the problem, rather than the solution. There are, however, situations where you simply can't apply a linear function.
Consider a simple two dimensional range-bearing sensor. How would one apply the Kalman filter here?

A simple range-bearing sensor (Kalman version)

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}, \mathbf{z} = \begin{bmatrix} r \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{(x^2 + y^2)} \\ \arctan(\frac{y}{x}) \end{bmatrix}$$

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} (\text{for example}), H = \text{er...}$$

The Extended Kalman Filter

Recall that in the Kalman filter,
$\mathbf{x}_{k|k-1} = f(\mathbf{x}_{k-1}, \mathbf{w}_k) = F_k \mathbf{x}_{k-1} + \mathbf{w}_k$ and
$\mathbf{z}_k = h(\mathbf{x}_k, \boldsymbol{\nu}_k) = H_k \mathbf{x}_k + \boldsymbol{\nu}_k$.

We can approximate the non-linear $f(\mathbf{x}_{k-1})$ and,or $h(\mathbf{x}_k)$ as a Taylor expansion about $\mathbf{x}_{k-1} = \boldsymbol{\mu}_{k-1}$ or $\mathbf{x}_{k|k-1} = \boldsymbol{\mu}_{k|k-1}$,

General n-d Taylor series approximation

$$g(\mathbf{x}) \approx g(\boldsymbol{\mu}) + (\mathbf{x} - \boldsymbol{\mu})^T Dg(\boldsymbol{\mu}) + \frac{1}{2!}(\mathbf{x} - \boldsymbol{\mu})^T D^2 g(\boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu}) + \ldots$$

The Extended Kalman Filter

This series is most often truncated at the first term which means that the task boils down to finding the gradient D using the transpose of the Jacobian matrix. The functions $f(.)$ and $h(.)$ can then be approximated as,

The Extended Kalman Filter

$$F(\mathbf{x_{k-1}}) \approx \mathbf{J}(\boldsymbol{f})|_{\mathbf{x}=\boldsymbol{\mu}_{k-1}}$$

$$H(\mathbf{x_{k|k-1}}) \approx \mathbf{J}(\boldsymbol{h})|_{\mathbf{x}=\boldsymbol{\mu}_{k|k-1}}$$

where

$$\mathbf{J}(\mathbf{g}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \cdots & \frac{\partial g_m}{\partial x_n} \end{bmatrix}$$

The predict and update stages proceed as with the Kalman filter

Practical

Jupyter notebook 2: extended Kalman filter, range-bearing exercise

The Particle Filter

Also known as *Sequential Monte Carlo (SMC) sampling* methods, particle filters seek to approximate (posterior) probability distributions using samples, or particles

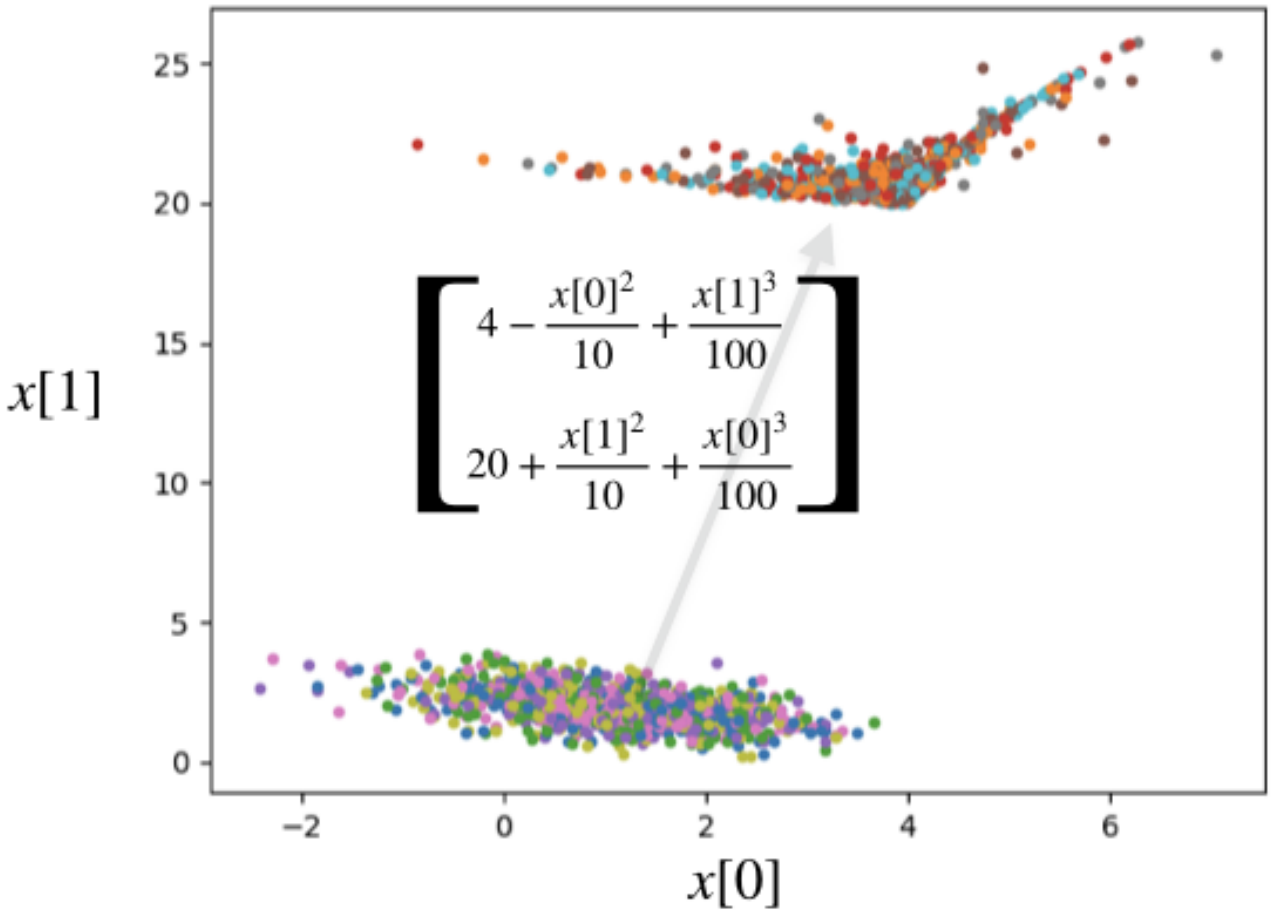
Sample-based probability distribution

We make the following approximation,

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i)$$

where w_k^i *are weights such that* $\sum_i w_k^i = 1$

Particle filter - pictorial intuition



Particle filter

- Each particle goes through the predict/update process as with the Kalman filters
- The predict step 'moves' the particles, and the update step recalculates the weights
- Benefits:
  - Points only: no need to consider higher moments (like those pesky covariances)
  - Easily copes with non-linearity in the process and sensor models
- Drawbacks:
  - computational effort
  - sample sparsity

Importance Sampling

Sequential Importance Sampling

$$w_k^i = w_{k-1}^i \frac{p(\mathbf{z}_k|\mathbf{x}_k^i)p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)}{q(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_{1:k})}$$

where $q(.)$ is the so-called importance density *which is easy to sample from, while approximating the posterior distribution.*

Degeneracy

After a few iterations, all but a small number of the particles will have negligible weight. This affects accuracy, wastes computation on particles with little effect on the estimate. Resampling schemes *– many exist – are designed to redistribute particles to areas where the posterior probability is higher.*

Practical

Jupyter notebook 4: particle filter

Data Association

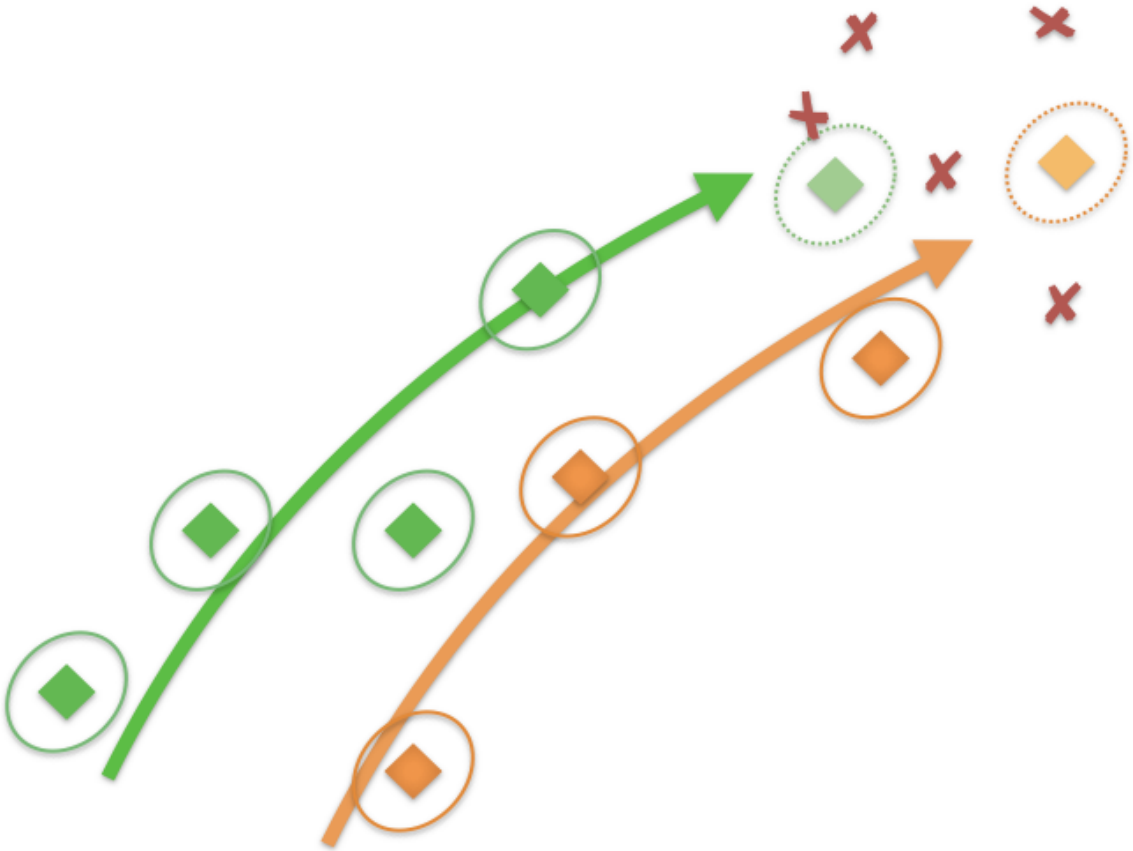
More often than not, the difficult part of state estimation concerns the ambiguous association of predicted states with measurements. This happens whenever,

- there is more that one target under consideration
- there are false alarms
- there is clutter
- targets can appear and disappear.

So it happens everywhere.

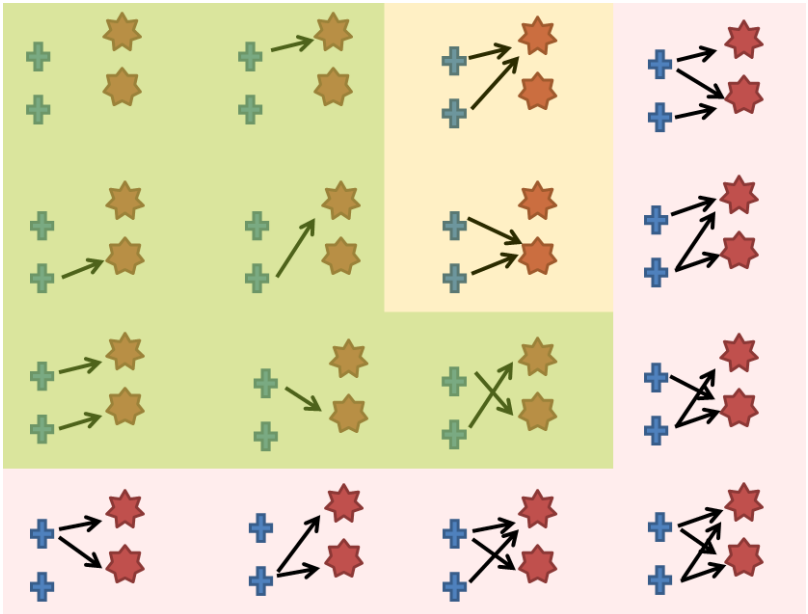
Association schemes

How do we map observations to predicted state?

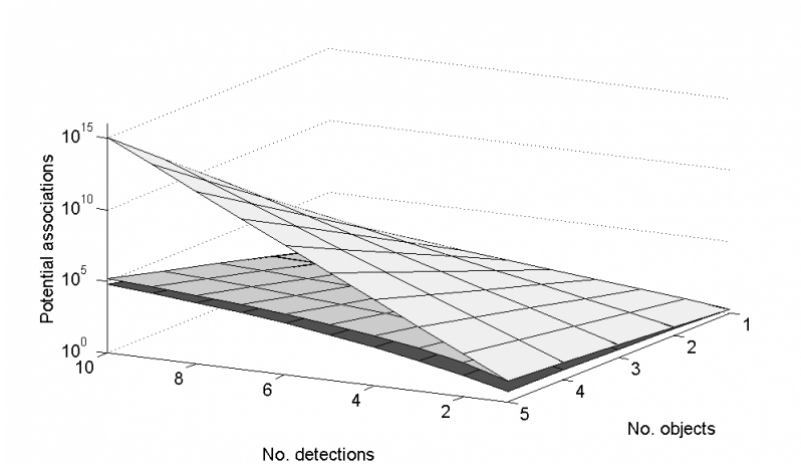


(with liberties taken regarding state-observation space)

Data Association - combinatorics



Exhaustive example of associating two targets (crosses) with two measurements (stars). Green: measurement generated by one target only and target capable of generating one measurement only; Yellow: includes measurements generated by more than one target; Pink: includes targets generating more than one measurement.



Number of ways of associating up to 5 targets with up to 10 measurements depending on whether you allow one-to-one (bottom), many-to-one (middle) or many-to-many (top) association

Absolute assignment schemes

Make the assumption, for the moment, that each target generates, at most, one measurement, and that one measurement is generated by a single target, or is a clutter point.

- Nearest neighbour (or *greedy* assignment): simply pick the measurement closest to that of the predicted *measurement*, i.e. pick the minimum innovation, or pick none. (Note that this potentially violates the many-to-one measurement to target assignment rule.)

Absolute assignment schemes

- Global nearest neighbour (GNN): assign measurements to predicted measurements to minimise some total (global) cost, this cost being a function of the sum of innovations. This is an example of an *assignment problem* in combinatorial optimisation. Solutions to these *Linear Programming* problems abound. For example, the *Hungarian algorithm* (also the Munkres algorithm), auction algorithms, the *simplex algorithm*.
- Practical considerations are usually invoked, e.g.
  - Gating
  - Track initiation

Practical

Jupyter notebook 5 and 6: multi-target tracking using GNN

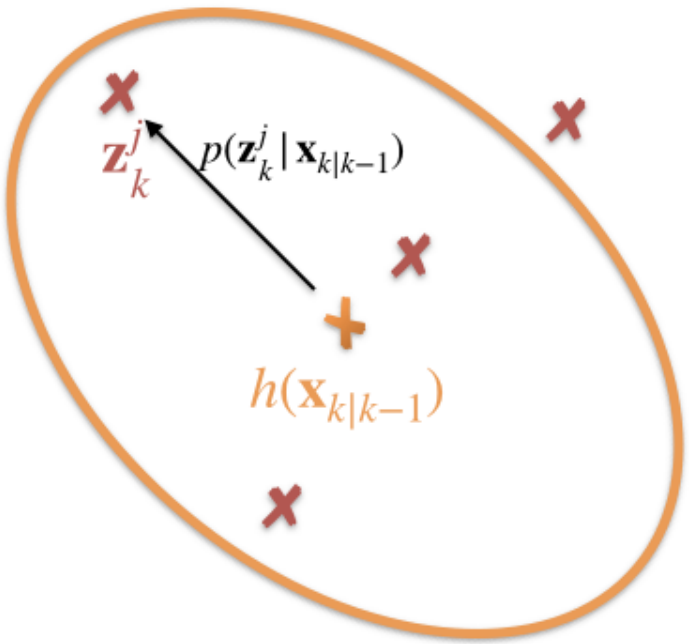
Issues

- Examine the output of the previous notebook. What happened?
- Is that your final decision?
  - There's a problem with making a firm decision at each time step
- Running time
  - Surprisingly often some form of brute-force algorithm will be used
  - Gate can be 'tuned'

Probabilistic Data Association

Rather than make a firm assignment at each timestep, we could work out the probability that each measurement should be assigned to a particular target.

Then, assign *a bit* of each measurement to a target based on some measure of those probabilities.



Probabilistic Data Association filter

$$\mathbf{Z}_k = \{\mathbf{z}_k^j\}, j = 1 \dots m_k$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + K_k \sum_{j=1}^{m_k} \beta^j \mathbf{y}_k^j$$

$$P_{k|k} = \beta_k^0 P_{k|k-1} + (1 - \beta_k^0) P_{k|k}^c + \tilde{P}_k$$

where,

$$\tilde{P}_k \triangleq K_k \left[\sum_{j=1}^{m_k} \beta_k^j \mathbf{y}_k^j \mathbf{y}_k^{j\,T} - \mathbf{y}_k^j \mathbf{y}_k^{j\,T} \right] K_k^T$$

Probabilistic Data Association

The symbols have the same meanings as they do in the description of the Kalman filter earlier, but now we have to cope with a set $\mathbf{Z}$ of m measurements. So $\mathbf{y}$ is still innovation but is now computed for each measurement as denoted by the superscript j. The new quantity β is the probability that the jth measurement is associated with the state under consideration.
The posterior covariance estimate is composed of three terms (broadly) interpreted as the sum of covariances representing:

- measurements all being false alarms,
- the target is associated correctly (hence P^c),
- uncertainties associated with incorrect associations, $\tilde{P}$.

Note that, as with the Kalman filter, we don't need to have the innovations to determine the correct posterior state covariance. We do, however, need them in order to determine the uncertainty due to incorrect associations.

Joint Probabilistic Data Association

Extending the PDA in a similar way as the GNN algorithm extends the naive nearest neighbour algorithm seems sensible.
But is that all that's missing from PDA? What about the notion of other targets/tracks? Interferers are no longer confined merely to clutter, and that can change the statistics.

Further to the PDA, assume:

- The number of targets we are tracking is known
- The transition and sensor models are linear-Gaussian distributed or can be approximated as such (via the EKF, say)

Joint Probabilistic Data Association

- We can enumerate the *joint conditional* probabilities of all assignments at time k (i.e. if the probability of the assignment of measurement j to target i ($\alpha^{i \leftarrow j}$) is β^{ij} then the probabilities of $\alpha^{k \leftarrow l}$ and $\alpha^{m \leftarrow n}$ are β^{kl} and β^{mn}, etc, etc.) These assignments may be statistically independent, or not.

JPDA marginal assignment probabilities

$$\begin{aligned} \beta_k^{ij} &\triangleq P(\alpha_k^{i \leftarrow j} | \mathbf{Z}_k) \\ &= \sum_{\alpha : \alpha^{i \leftarrow j} \in A_k} P(\alpha | \mathbf{Z}_k) \end{aligned}$$

where A_k are all possible assignments at time k

As with the GNN, we're likely to need efficient assignment.

Jupyter notebook 7, 8: joint probabilistic data association - compare with GNN

Practical considerations

In the real world we need to be able to:

- start a track in response to *new* measurements,
- delete tracks when we no longer want to (or can't) consider the object.

We might also want to use some practical measures to make data association easier:

- exclude certain combinations from consideration to reduce the computational complexity.

And if we're concerned with measuring how well we've done

- we'll want to compare against other methods by deriving measures of performance which are consistent across scenarios.

Practical considerations

Initiating and deleting tracks

This usually leans heavily toward the specifics of the problem being addressed. That may be to do with how target detectability, clutter, sensor performance, etc.

Track initiation

- M-consecutive unassociated detections
- M-of-N unassociated detections

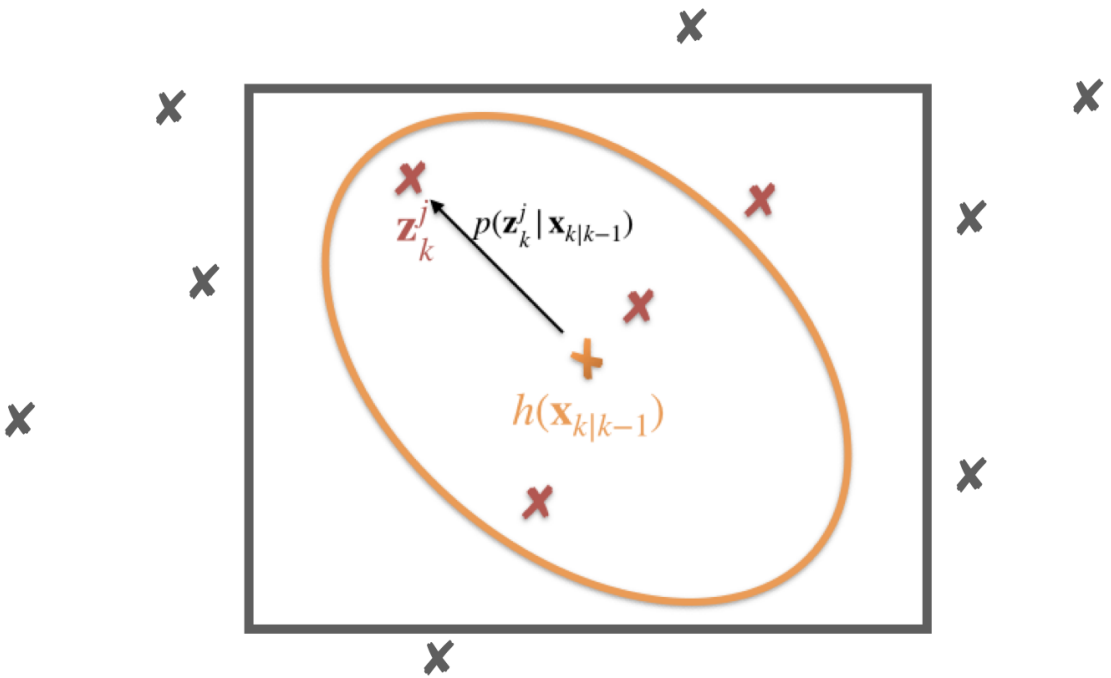
Track deletion

- Covariance limited
- Time-based

Practical considerations

Association

Why waste your time enumerating (and calculating the probability of) associations which have virtually no chance of being credible? A *Gate* can be employed to reduce the number of potential associations we are willing to consider. While conceptually easy to understand, it's ultimately up to the operator to choose a gate of appropriate size and shape; and that's usually scenario-specific.



Metrics

We want to be able to judge how well we've done in our tracking. But does *good* tracking mean

- a small average difference between a state's estimate and ground truth,
- an uncertainty representative of the difference between mean estimate and ground truth,
- consistent assignment of track to truth,
- maintaining tracks on a large number of objects in the scene of interest,
- one-to-one track to truth assignment,
- low number of track breaks,
- or something else, or combination thereof?

Metrics

It's worth distinguishing between true metrics and figures of merit. Strictly speaking, a metric is a function which returns a non-negative scalar and satisfies three properties:

- identity, $d(x_1, x_2) \implies x_1 = x_2$
- symmetry, $d(x_1, x_2) = d(x_2, x_1)$
- triangle inequality, $d(x_1, x_2) \leq d(x_1, x_3) + d(x_3, x_2)$

A figure of merit is more loosely defined as a numerical representation of a tracker's performance. Whilst not being as mathematically rigorous as metrics, they can capture practical aspects of a problem (for example if one prioritised persistent track association over assigning the correct number of tracks).

Metrics

Optimal Sub-Pattern Assignment

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

Generalized Optimal Sub-Pattern Assignment

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

Metrics and figures of merit

Single Integrated Air Picture (SIAP) 'metrics'
Notable figures of merit include,

- Completeness: the fraction of true objects that are tracked
- Clarity: subdivided into
  - Ambiguity: the fraction of true objects for which more than one track can be associated
  - Spuriousness: the fraction of tracks with no associated true object
- Continuity: the fraction of time for which the track's ID does not change
- Kinematic accuracy: how well track position and velocity (more generally, the state vector) matches the position and velocity of the associated object

And there are others (see references) associated with, among other things, how well and consistently an object's identification is maintained and how similar estimates of the tracking picture are across a set of sensors.

Practical

Jupyter notebooks 9 and 10: practical tracking, and metrics example

Time did not permit us to cover

- Other filtering schemes
  - Unscented Kalman filter
  - Continuous-time filter
  - Information filter
- Other association schemes
  - Many flavours of (J)PDA
  - Multi-hypothesis tracker (MHT)
- Multiple models
  - Interacting multiple model filter
  - Multiple sensors
- Non-sequential estimation
  - Asynchronous and out-of-sequence measurements
  - Smoothing
- Association-free filtering
  - The Probability Hypothesis Density (PHD) filter and its derivatives
  - Finite-set statistics-based filtering
  - Track-before-detect

Summary

We have covered:

- The abstract state-estimation and tracking problem and (some of the many) defence scenarios where this is of interest
- Single-target state estimation and tracking, linear and non-linear examples
- Multi-target tracking, the association problem and some schemes to address target-measurement assignment
- (Some) practical issues and how to measure tracking performance

Resources

Software

Stone Soup: `https://github.com/dstl/Stone-Soup`

Textbooks, tutorial papers, other references

Blackman S. S. 1986, Multiple-target tracking with radar applications, *Dedham, MA, Artech House, Inc.*

Koch W. 2014, Tracking and Sensor Data Fusion, *ISBN 978-3-642-39271-9*

Sanjeev Arulampalam M., Maskell S., Gordon N., Clapp T. 2002, Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking, *IEEE transactions on signal processing, vol. 50, no. 2*

Daum F, Huang, F. 2009, The Probabilistic Data Association Filter, Bar-Shalom Y, *IEEE Control Systems Magazine*

Votruba P., Nisley R., Rothrock R. et al. 2001-2003 (revised), SIAP SE TF Technical Reports 2001-001–003, Arlington VA: SIAP SE TF

Resources

People
Open Source Tracking and Estimation Working Group `isif-ostewg.org`

Social media
Stone Soup on Gitter
`https://gitter.im/dstl/Stone-Soup`