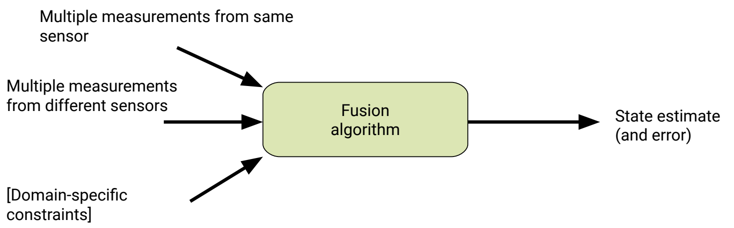
Sensor Fusion

Dr Robert Harle
Mobile and Sensor Systems
Lent 2020

Measurements are Noisy

A sensor measures some quantity with some accuracy. Whatever we do, noise will creep in

We therefore need to fuse multiple measurements to get a robust idea of what's happening



Algorithms

There are many fusion techniques and algorithms

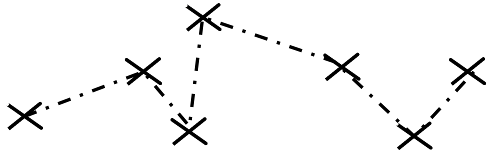
We will look at the two extremes: a very fast, very common algorithm that is limited in what it works with, and a general-purpose and flexible but more computationally demanding algorithm

Both are based on bayesian probability

We will use location **tracking** to illustrate the techniques because the problem is easy to relate to. But everything is general.

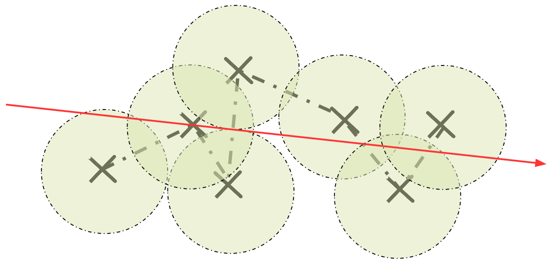
Simple Tracking Example

Consider a series of positions that come in a few seconds apart for a pedestrian. They will probably look rather unrealistic for a walking route:



Simple Tracking Example

But if we consider noise and error in the measurements we see that the data supports a more realistic hypothesis of straight line walking:



Probabilistic Approach

So what we want to do is to estimate our current state while incorporating knowledge of recent measurements and all of the associated errors. To do this we will use probability

$$\text{Bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|\boldsymbol{z}_1, ..., \boldsymbol{z}_t)$$

Belief (probability)

State at time t (e.g. position)

Measurements (e.g. from positioning system)

Filters and Smoothers

$$\text{Bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|\boldsymbol{z}_1, ..., \boldsymbol{z}_t)$$

This is known as a **filter** because it estimates the current state based on current and past measurements (only)

Sometimes you know the ‘future’ e.g. you may have logged data for post-processing rather than live processing

In that case you have a **smoother**

Recursive Bayesian Filters

Apply a Markov model (next state depends only on last) to recursively build up our probabilities

$$\text{Bel}^-(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-\delta t})\text{Bel}(\boldsymbol{x}_{t-\delta t})\, d\boldsymbol{x}_{t-\delta t}\,.$$

Prior

Propagation (motion) model

Evaluate over all previous states

This is the **propagation** or prediction step

We update the probabilities based on some model (e.g. constant velocity) → prior distribution

Recursive Bayesian Filters

Apply Bayes' theorem to incorporate measurements

$$\mathrm{Bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|\boldsymbol{z}_1, ..., \boldsymbol{z}_t)$$

$$\mathrm{Bel}(\boldsymbol{x}_t) = \alpha_t p(\boldsymbol{z}_t|\boldsymbol{x}_t)\mathrm{Bel}^-(\boldsymbol{x}_t)$$

Posterior

Norm factor

Measurement model

Prior

This is the **correction** or update step

We correct the probabilities on a measurement → posterior distribution

Implementation

$$\mathrm{Bel}^-(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-\delta t})\mathrm{Bel}(\boldsymbol{x}_{t-\delta t})\, d\boldsymbol{x}_{t-\delta t}\,.$$ Propagation/predict

$$\mathrm{Bel}(\boldsymbol{x}_t) = \alpha_t p(\boldsymbol{z}_t|\boldsymbol{x}_t)\mathrm{Bel}^-(\boldsymbol{x}_t)$$ Correction/update

There are broadly two classes of techniques to implement these filters

1. Model all the probability distributions using mathematical models. This keeps everything continuous. But it's not always easy to do this (the distributions get complex). E.g. Use Gaussians everywhere → “Kalman Filter”

2. Represent arbitrary distributions by sampling them. Nice and general but much more work involved.

The Kalman Filter

The Kalman Filter

The simplest recursive Bayesian filter

It is used everywhere: very important

Requires that you can write the dynamics of your system using linear algebra (matrices etc)

Boils down to 3 equations:

Motion model

Last state

New state

$$\boldsymbol{x}_t = \boldsymbol{F}_t\boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$

$$\boldsymbol{P}_t^- = \boldsymbol{F}_t\boldsymbol{P}_{t-\delta t}\boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

Propagation

$$\boldsymbol{z}_t = \boldsymbol{H}_t\boldsymbol{x}_t + \boldsymbol{v}_t$$

Correction

Motion Model Example: Constant Velocity

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$

$$F = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$$

The Kalman Filter

The simplest recursive Bayesian filter

It is used everywhere: very important

Requires that you can write the dynamics of your system using linear algebra (matrices etc)

Boils down to 3 equations:

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$ Propagation

$$\boldsymbol{P}_t^- = \boldsymbol{F}_t \boldsymbol{P}_{t-\delta t} \boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

Noise terms

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t$$ Correction

The Kalman Filter

The simplest recursive Bayesian filter

It is used everywhere: very important

Requires that you can write the dynamics of your system using linear algebra (matrices etc)

Boils down to 3 equations:

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$ Propagation

Covariance ("error")

$$\boldsymbol{P}_t^- = \boldsymbol{F}_t \boldsymbol{P}_{t-\delta t} \boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t$$ Correction

The Kalman Filter

The simplest recursive Bayesian filter

It is used everywhere: very important

Requires that you can write the dynamics of your system using linear algebra (matrices etc)

Boils down to 3 equations:

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-\delta t} + \boldsymbol{w}_t$$ Propagation

$$\boldsymbol{P}_t^- = \boldsymbol{F}_t \boldsymbol{P}_{t-\delta t} \boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

Measurement model

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t$$ Correction

Measurement Model Example

Just measure the position directly

$$z_t = H_t x_t + v_t$$

$$H = (1 \quad 0)$$

The Nitty Gritty

Predict [edit]

Predicted (*a priori*) state estimate	$\hat{\mathbf{x}}_{k\mid k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1\mid k-1} + \mathbf{B}_k \mathbf{u}_k$
Predicted (*a priori*) estimate covariance	$\mathbf{P}_{k\mid k-1} = \mathbf{F}_k \mathbf{P}_{k-1\mid k-1} \mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k$

Update [edit]

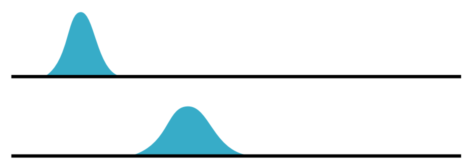
Innovation or measurement residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k\mid k-1}$
Innovation (or residual) covariance	$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k\mid k-1} \mathbf{H}_k^T + \mathbf{R}_k$
Optimal Kalman gain	$\mathbf{K}_k = \mathbf{P}_{k\mid k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$
Updated (*a posteriori*) state estimate	$\hat{\mathbf{x}}_{k\mid k} = \hat{\mathbf{x}}_{k\mid k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$
Updated (*a posteriori*) estimate covariance	$\mathbf{P}_{k\mid k} = (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k\mid k-1}$

(Thanks to wikipedia. No, you aren't expected to learn these)

Key to the Kalman Filter

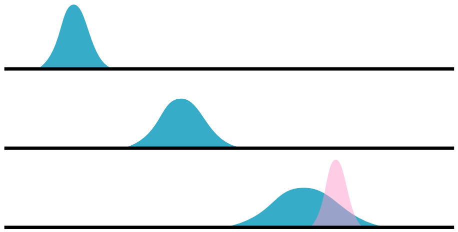
Initially we have some position estimate that is associated with a normal distribution

Key to the Kalman Filter



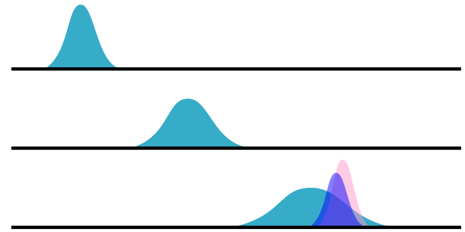
We propagate the state, meaning we use the motion model to move it forward. Since we had no actual input, we increase the error (→ Gaussian gets shorter and fatter)

Key to the Kalman Filter



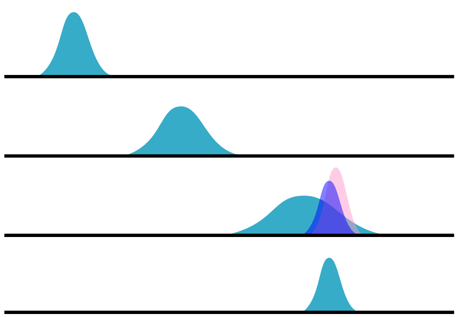
We repeat the propagation but then a measurement comes in. This is associated with another Gaussian, although thinner because it's an OK estimate

Key to the Kalman Filter



The beauty of a Gaussian is that when you multiply two together you get another Gaussian. Thus we always finish a cycle with a new Gaussian estimate → we can represent it using just two parameters, making it amenable to linear algebra

Key to the Kalman Filter

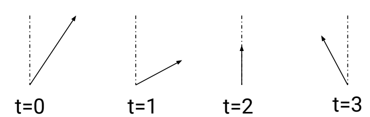


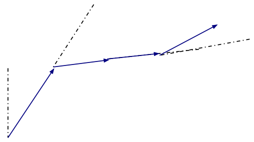
A more complex example

Consider the Inertial GPS systems you find in vehicles. They need to estimate where the car is at all times between GPS measurements.

We compute position by concatenating a series of displacements and headings (dead reckoning).

We use inertial sensors to estimate the displacements (wheel encoders) and headings (gyroscopes) since the last state estimate

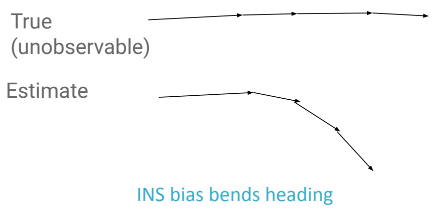




Inertial Nav

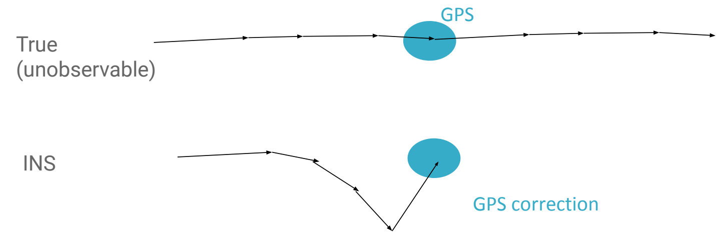
We integrate the gyroscope signal to estimate the heading change (note the motion model uses the inertial inputs)

But gyros are subject to bias errors (a bias is a bogus offset reported when it's not rotating) and we often see erroneous bending:



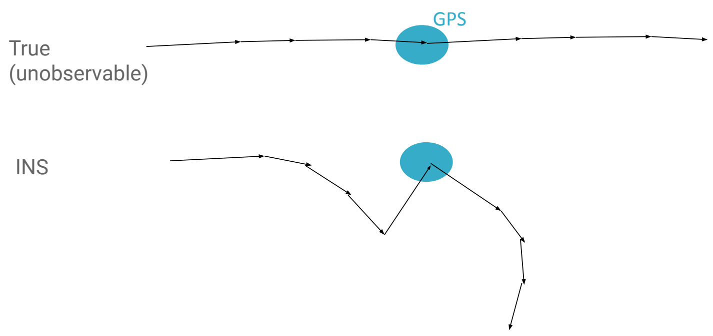
Inertial Nav

When a GPS measurement comes in we can fix things



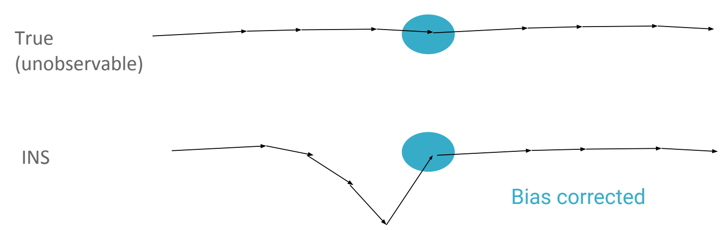
Inertial Nav

But if we just correct position, it starts to go wrong again



Inertial Nav

But if we add the bias to the state in the kalman filter, it will estimate that for us too



KF Limitations

$$\mathrm{Bel}^-(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-\delta t})\mathrm{Bel}(\boldsymbol{x}_{t-\delta t})\, d\boldsymbol{x}_{t-\delta t}\,.$$

Propagation

$$\mathrm{Bel}(\boldsymbol{x}_t) = \alpha_t p(\boldsymbol{z}_t|\boldsymbol{x}_t)\mathrm{Bel}^-(\boldsymbol{x}_t)$$

Correction

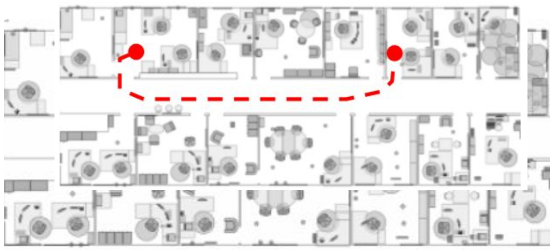
What if those probability distributions don't lend themselves to being normal?

Our example will be constraining movement to be on a building floorplan. How could you build a motion model matrix that incorporated a floorplan??!

The Particle Filter

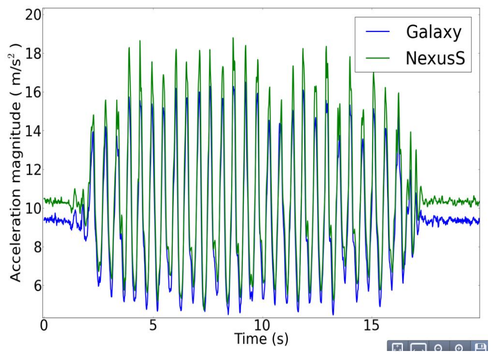
Our Goal

Figure out where someone walked indoors



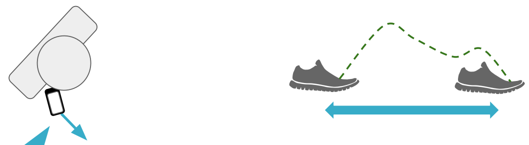
Step Detection

Phones detect step events pretty well from the accelerometer



Step-and-Heading Systems

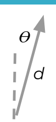
IMUs in phonescan detect steps quite well, and orientation changes reasonably well (from a KF on the gyro)



Step-and-Heading Systems

IMUs in phones can detect steps quite well, and orientation changes reasonably well (from a KF on the gyro)

Noisy Dead Reckoning



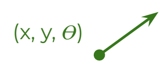
We end up with a lot of step vectors, with noise on both orientation and length.

We could add these up (dead reckoning) but the noise will accumulate fast and we'll have big errors.

Need to model the errors and constrain them based on the floorplan.

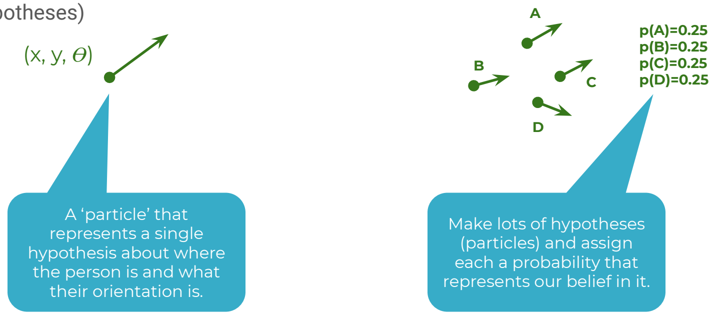
Particle Filter Approach

Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)



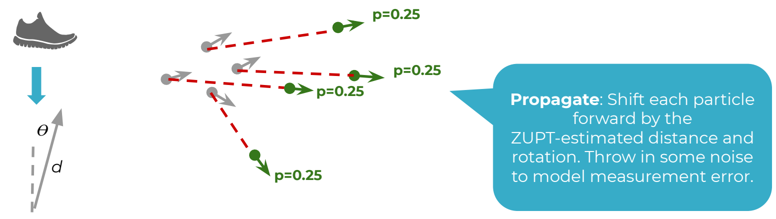
Particle Filter Approach

Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)



Particle Filter Approach

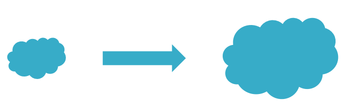
Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)



Growing uncertainty

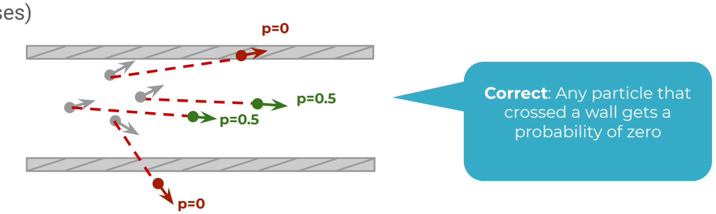
With every step, the noise I put in means the particle cloud will spread.

This is correct: I have not included any measurement to constrain it. Compare to the Gaussian getting fatter and fatter in the Kalman filter before the correct stage.



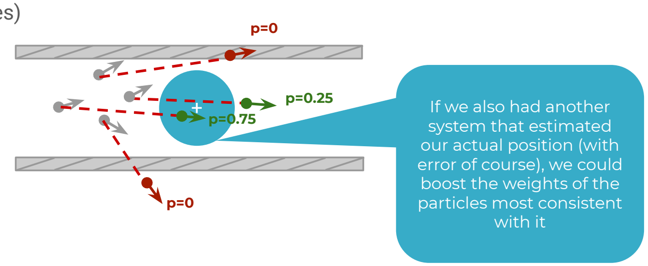
Particle Filter Approach

Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)



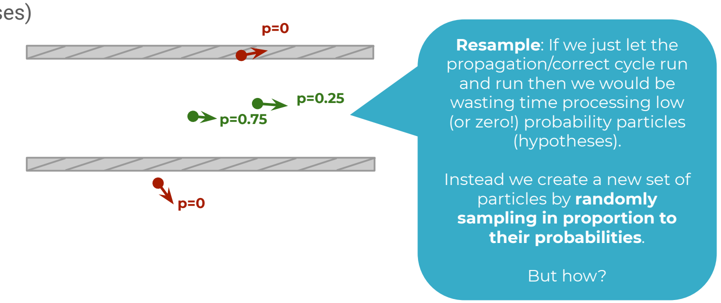
Particle Filter Approach

Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)



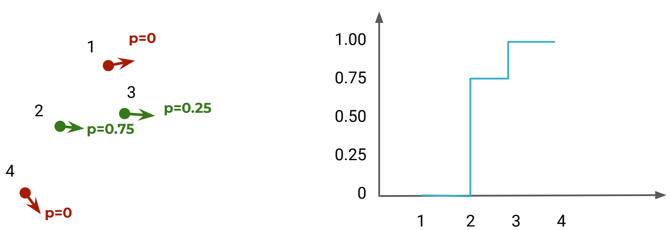
Particle Filter Approach

Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)



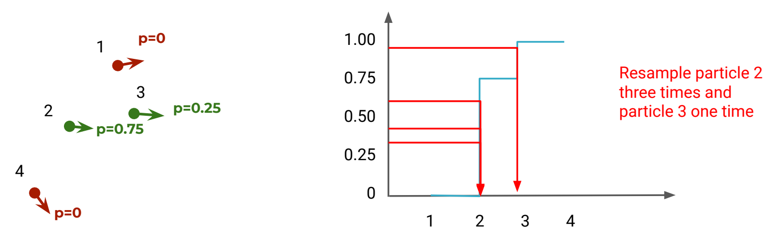
Sampling in Proportion

Number each particle (order is irrelevant) and form the cumulative weight distribution



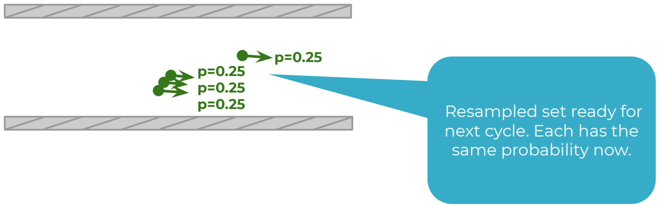
Sampling in Proportion

Generate uniform random number between 0.0 and 1.0 and read across and down to find out which particle to clone into the next generation ("resample")



Particle Filter Approach

Floorplan constraints are hard to incorporate because they are so nonlinear in nature. Instead we apply monte-carlo technique (effectively simulating multiple hypotheses)

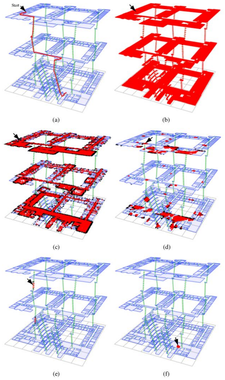


A Note on Performance

Update and correct steps are nicely parallelisable

But forming the cumulative weight for resampling is fundamentally sequential...

Localisation vs Tracking

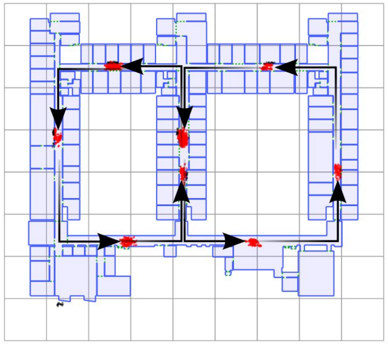


Initially we have no knowledge of the user's position

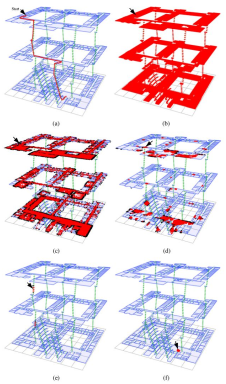
Lots of particles

"Localisation Phase"

Symmetry Problem



Localisation vs Tracking



Eventually we figure out where they are and the problem becomes easier

Fewer particles needed

"Tracking Phase"

(For interest, we got ~ 0.75m accuracy 95% of the time in this building)

In General

Particle filters are easy to implement and highly flexible

But:

- Every particle you add costs you in terms of computation
- The results are not deterministic
- Too few particles gives bad/failed results, while too many wastes precious CPU cycles. You need to ensure your system adequately represents the real uncertainty without going overboard!