CMPE58C: Sp. Tp. Mobile Location Tracking & Motion Sensing

Data Fusion

Can Tunca

-

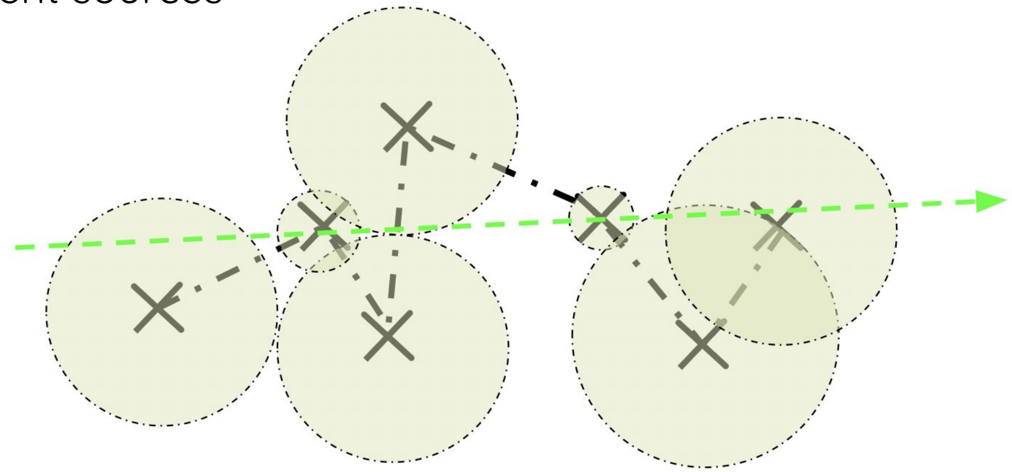
Fall 2022

Lecture Overview

- Why “Data Fusion” and not “Sensor Fusion”
  - Sensor fusion implies fusion of raw sensor data
  - Fusion can take place at higher levels, e.g. fusing the output of two positioning methods
- Some topics that will be covered:
  - Bayesian filters introduction
  - Kalman filter
  - Linearized Kalman filters
    - Extended Kalman filter
    - Unscented Kalman filter
  - Particle filters
  - Data fusion examples
    - 1D positioning (velocity + absolute measurements)
    - 2D positioning (odometry)
    - Pedestrian dead reckoning

Data Fusion: Reasons

- Handling uncertainty in raw measurements (noise/bias)
- Desire to fuse measurements from multiple sensors
- Incorporating accuracy estimates of measurements (if exists)
- Smoothing (preventing fluctuations)
- Modeling error and noise of different sources



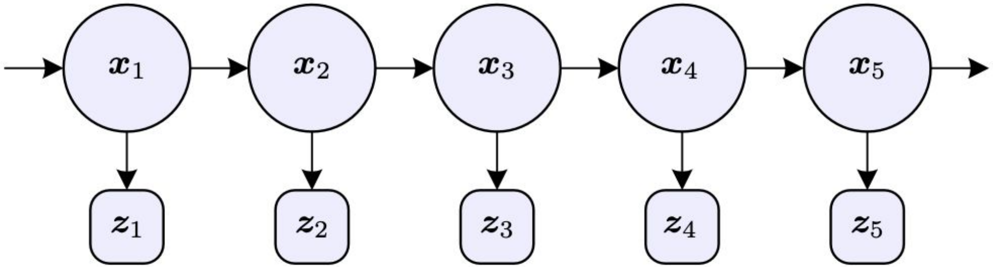
Bayesian Filters

- A probabilistic framework for recursive state estimation
- Estimating the probability distribution over the space of all possible states at time *t*

$$\underbrace{\mathrm{Bel}(\boldsymbol{x}_t)}_{\text{belief}} = p(\boldsymbol{x}_t | \underbrace{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t}_{\text{measurements made up to } t})$$

- Representing state as a probability distribution allows us to provide both an estimate and its uncertainty
- i.e. the belief can be queried to get the probability that $\boldsymbol{x}_t$ is the true state
- The complexity of belief increases as measurements accumulate, to combat this Bayesian filters assume the underlying dynamic system is a **Markov process**

Markov Process



- Two assumptions
  - The current state $\boldsymbol{x}_t$ depends on only the previous state $\boldsymbol{x}_{t-\delta t}$
  - A measurement $\boldsymbol{z}_t$ depends on only the current state $\boldsymbol{x}_t$
- In other words, all the information required to estimate state at time *t* is given by the information available at time *t* - 1: aka Markovian property
- Such systems are defined by two distributions:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-\delta t})$$

propagation model

$$p(\boldsymbol{z}_t|\boldsymbol{x}_t)$$

measurement model

Belief Propagation and Correction

- Under the Markov assumption current state updated based on previous state:

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

- This is called the estimated belief or *prior* distribution
- The prior is then is corrected to obtain *posterior* distribution:

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

(α_t is a normalization factor to make sure Bel is a prob. distribution)

- Different measurement models can be used to incorporate different measurement types (Bayesian filters naturally support sensor fusion)
- This is a probabilistic “framework”, the concrete implementation should choose how to represent the belief. We’ll see the most common implementations next…

Kalman Filter

- Assuming belief is Gaussian distributed for all time *t*, hence representable by a mean and covariance
- State transition (propagation) model is linear:

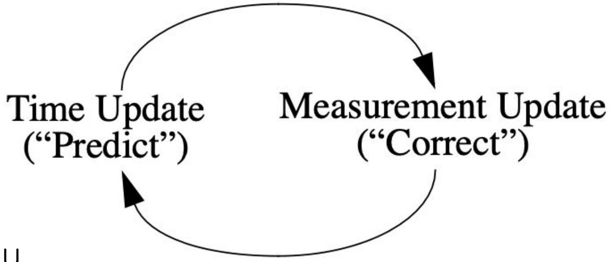
$$\underbrace{\boldsymbol{x}_t = \boldsymbol{F}_t}_{\text{state transition matrix}} \boldsymbol{x}_{t-\delta t} + \underbrace{\boldsymbol{w}_t}_{\text{zero mean Gaussian noise (process noise)}}$$

- Measurements are related to the state by a linear function:

$$\boldsymbol{z}_t = \underbrace{\boldsymbol{H}_t}_{\text{matrix relating measurement to state}} \boldsymbol{x}_t + \underbrace{\boldsymbol{v}_t}_{\text{zero mean Gaussian noise (measurement noise)}}$$

Kalman Filter

- How to compute belief mean and covariance?
- Two stages: **PREDICT** and **CORRECT**, a recursive process
- Correspond to state transition and measurement models, respectively



PREDICT

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t \hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}_t \mathbf{u}_t$$

$$\mathbf{P}_{t|t-1} = \mathbf{F}_t \mathbf{P}_{t-1|t-1} \mathbf{F}_t^T + \mathbf{Q}_t$$

CORRECT

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t \left(\mathbf{y}_t - \mathbf{H}_t \hat{\mathbf{x}}_{t|t-1} \right)$$

$$\mathbf{K}_t = \mathbf{P}_{t|t-1} \mathbf{H}_t^T \left(\mathbf{H}_t \mathbf{P}_{t|t-1} \mathbf{H}_t^T + \mathbf{R}_t \right)^{-1}$$

$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \mathbf{P}_{t|t-1}$$

$\hat{\mathbf{x}}$: Estimated state.

$\mathbf{F}$: State transition matrix (i.e., transition between states).

$\mathbf{u}$: Control variables.

$\mathbf{B}$: Control matrix (i.e., mapping control to state variables).

$\mathbf{P}$: State variance matrix (i.e., error of estimation).

$\mathbf{Q}$: Process variance matrix (i.e., error due to process).

$\mathbf{y}$: Measurement variables.

$\mathbf{H}$: Measurement matrix (i.e., mapping measurements onto state).

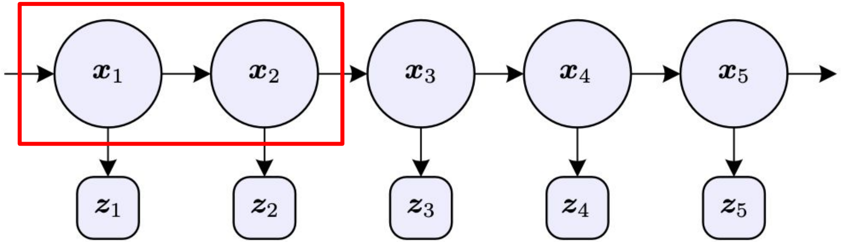
$\mathbf{K}$: Kalman gain.

$\mathbf{R}$: Measurement variance matrix (i.e., error from measurements).

Kalman Filter: Predict

1 $$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t \hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}_t \mathbf{u}_t$$

2 $$\mathbf{P}_{t|t-1} = \mathbf{F}_t \mathbf{P}_{t-1|t-1} \mathbf{F}_t^T + \mathbf{Q}_t$$



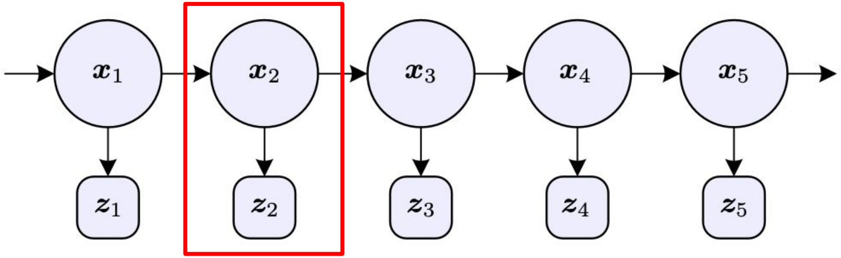
- Predict stage advances time (*t*-1 -> *t*)
- State transition to get an "a priori" estimate for the next timestep
- 1 computes the a priori mean (the actual estimate)
- 2 computes the a priori error covariance (the error of the estimate)
- Together they form a Gaussian representing the belief
- $\mathbf{u}_t$ is the optional control input, some external known factor affecting state
  - Either a controllable action (e.g. throttle in a car) or a measurable quantity advancing the state (e.g. acceleration), usually denoting a known **change** to the system - A possible place for data fusion!
  - $\mathbf{Q}_t$ embeds both error due to state transition and control input

Kalman Filter: Correct

1 $\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t \left(\mathbf{y}_t - \mathbf{H}_t \hat{\mathbf{x}}_{t|t-1}\right)$

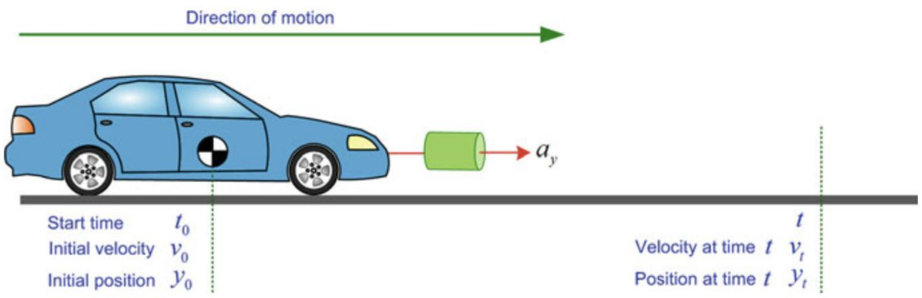
2 $\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^T \left(\mathbf{H}_t \mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t\right)^{-1}$

3 $\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t)\mathbf{P}_{t|t-1}$



- Corrects estimate at time *t* with a measurement
- Computes the “posterior” estimate for current time
- **1** computes the posterior mean (the actual estimate)
- **3** computes the posterior error covariance (the error of the estimate)
- **2** determines how to fuse the prior belief and measurement: **Kalman Gain**
  - We shouldn’t trust new measurement fully, otherwise we wouldn’t be using accumulated knowledge
  - We shouldn’t trust the previously predicted state, otherwise we won’t be correcting by new measurements
  - Quite like the blending weight in complementary filter
  - Computed optimally, error models $\mathbf{Q}_t$ and $\mathbf{R}_t$ influence the outcome

Kalman Filter Example: 1D Positioning



$$y_t = y_{t-1} + v_{t-1}\Delta t$$

$$v_t = v_{t-1}$$

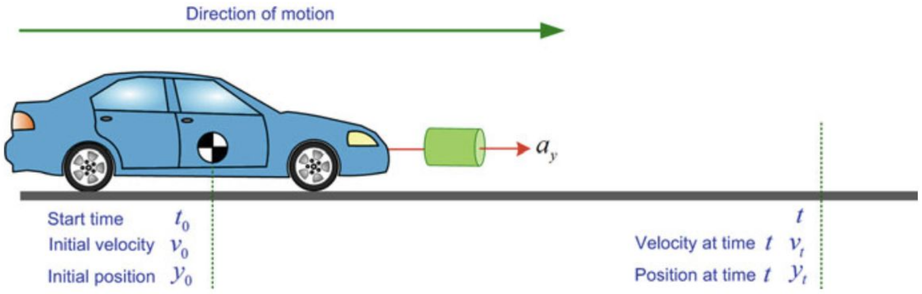
- Assume
  - A car travelling on a straight line
  - Intermittent GPS readings
  - No other measurements (no velocity, no accelerometer)
  - No fusion (only filtering)
  - Our best velocity estimate is the previous one (since no measurement)
- Can we get a smooth estimate of position?
  - Also continue tracking in case of no GPS (e.g. car going in a tunnel)

WE NEED TO DEFINE THESE MODELS

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

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

Kalman Filter Example: 1D Positioning



$$y_t = y_{t-1} + v_{t-1}\Delta t$$

$$v_t = v_{t-1}$$

WE NEED TO DEFINE
THESE MODELS

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

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

position

$$\boldsymbol{x}_t = \begin{pmatrix} y_t \\ v_t \end{pmatrix}$$

velocity

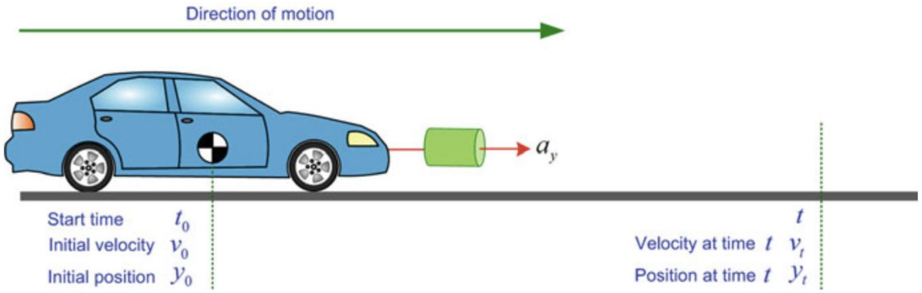
$$\boldsymbol{F}_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$$\boldsymbol{z}_t = \hat{y}_t$$

$$\boldsymbol{H}_t = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

position measurement (GPS)

Kalman Filter Example: 1D Positioning



$$y_t = y_{t-1} + v_{t-1}\Delta t$$
$$v_t = v_{t-1}$$

$$\boldsymbol{F}_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{H}_t = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

$\boldsymbol{x}_0$: Set to your initial estimate (could be zeroes): (2x1 matrix)

$\boldsymbol{P}_0$: Set to your initial uncertainty (could be big if unsure) (2x2 matrix)

$\boldsymbol{Q}_t$: How much do you trust your state transition model? (2x2 matrix)

$\boldsymbol{R}_t$: How much do you trust the measurements (GPS)? (1x1 matrix)

PREDICT

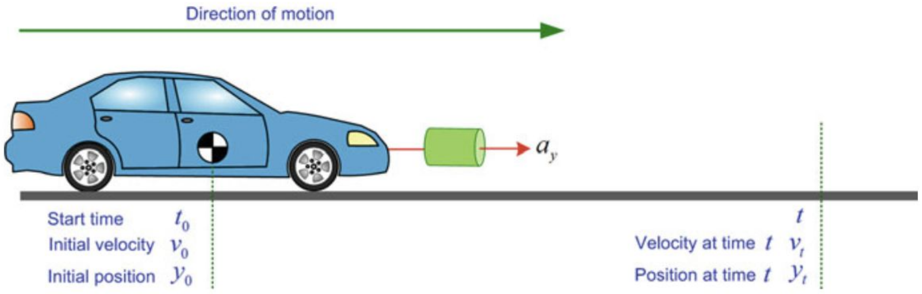
No control input in this example

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t \hat{\mathbf{x}}_{t-1|t-1} + \cancel{\mathbf{B}_t \mathbf{u}_t}$$
$$\mathbf{P}_{t|t-1} = \mathbf{F}_t \mathbf{P}_{t-1|t-1} \mathbf{F}_t^T + \mathbf{Q}_t$$

CORRECT

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t \left(\mathbf{y}_t - \mathbf{H}_t \hat{\mathbf{x}}_{t|t-1}\right)$$
$$\mathbf{K}_t = \mathbf{P}_{t|t-1} \mathbf{H}_t^T \left(\mathbf{H}_t \mathbf{P}_{t|t-1} \mathbf{H}_t^T + \mathbf{R}_t\right)^{-1}$$
$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \mathbf{P}_{t|t-1}$$

Kalman Filter Example: 1D Positioning with Acceleration



$$y_t = y_{t-1} + v_{t-1}\Delta t + \frac{1}{2}a_t\Delta t^2$$

$$v_t = v_{t-1} + a_t\Delta t$$

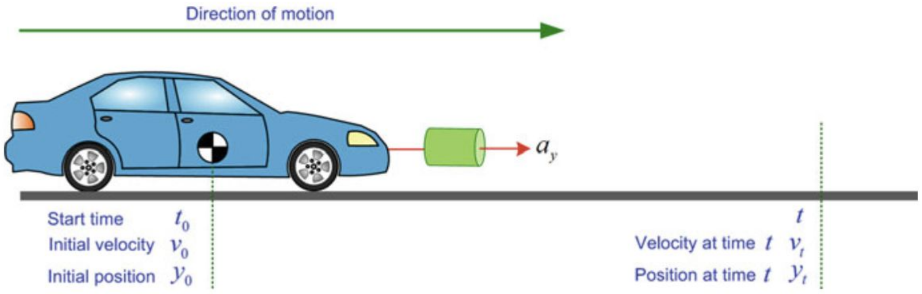
- Assume
  - We can also measure acceleration
  - Same state elements (position + velocity)
- Can we use it to improve our estimate?
- How to incorporate it into our model?
  - Control input is a good place

WE NEED TO DEFINE THESE MODELS

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

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

Kalman Filter Example: 1D Positioning with Acceleration



$$y_t = y_{t-1} + v_{t-1}\Delta t + \frac{1}{2}a_t\Delta t^2$$
$$v_t = v_{t-1} + a_t\Delta t$$

$$\boldsymbol{B}_t = \begin{bmatrix} \Delta t^2/2 \\ \Delta t \end{bmatrix} \qquad \boldsymbol{u}_t = a_t$$

PREDICT

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t\hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}_t\mathbf{u}_t$$
$$\mathbf{P}_{t|t-1} = \mathbf{F}_t\mathbf{P}_{t-1|t-1}\mathbf{F}_t^T + \mathbf{Q}_t$$

CORRECT

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t\left(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{x}}_{t|t-1}\right)$$
$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^T\left(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t\right)^{-1}$$
$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1}$$

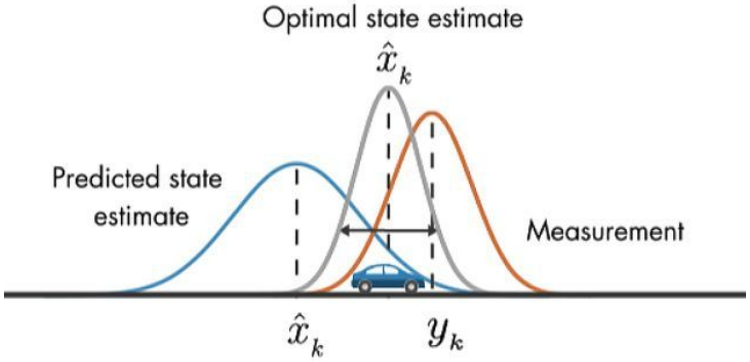
We should also adapt $\boldsymbol{Q}_t$ to incorporate acceleration error

Everything else is the same

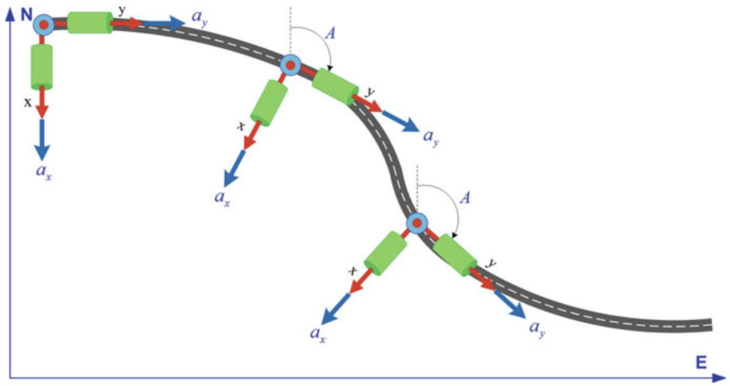
This is data fusion! (we've fused accelerometer and GPS)

Kalman Filter Examples: Some Observations

- Even if I get no GPS for a while, the velocity state element will advance the state
- Control input can be used to incorporate external measurements (first possibility for sensor fusion)
- Different measurement models may be used for incorporating different data (second possibility of sensor fusion)
  - Example: What if we have **speed** measurements? $\boldsymbol{H}_t = \begin{pmatrix} 0 & 1 \end{pmatrix}$
  - This will also correct position! (correlated state elements)
  - Can specify a different error model $\boldsymbol{R}_t$
  - Can be fed into the filter in an alternating fashion (or whenever data is available)
- In a way Kalman filter fuses two Gaussians optimally
  - Prediction error grows until a measurement comes in



Kalman Filter Example: 2D Positioning



$$x_t = x_{t-1} + v_{t-1}\Delta t \cos(\theta_{t-1} + \Delta\theta_t)$$

$$y_t = y_{t-1} + v_{t-1}\Delta t \sin(\theta_{t-1} + \Delta\theta_t)$$

$$\theta_t = \theta_{t-1} + \Delta\theta_t$$

- Assume
  - We can measure speed and heading change (e.g. car speedometer + steering wheel input)
  - Intermittent GPS readings
- We'll revisit this...

WE NEED TO DEFINE THESE MODELS, BUT **CAN WE**?

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

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

Kalman Filter Conclusion

- Assuming the assumptions hold (linear state transition and measurement models, Gaussian belief and noise), Kalman filter is an optimal filter
- i.e. we can compute the Bayesian filter framework equations exactly
- It is important to set noise covariances right
- Preferred solution for linear or near-linear systems where the belief is unimodal
- Computationally efficient (only a few matrix operations)
- A versatile tool that you can use in many areas (not just for positioning!)

Extended Kalman Filter (EKF)

- What if we can linearize state transition and/or measurement models locally?
- EKF: A common variant of linearized Kalman filters
- It is no longer an optimal filter, an approximation
- The approximation is still representable by a Gaussian
- Works well if the approximation is good enough (i.e. non-linearity is not too much)
- We can then use arbitrary functions for state transition and measurement models, as long as they are differentiable:

$$\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) + \boldsymbol{w}_t$$

$$\boldsymbol{z}_t = h(\boldsymbol{x}_t) + \boldsymbol{v}_t$$

Extended Kalman Filter

PREDICT

$$\hat{\boldsymbol{x}}_{t|t-1} = \underline{f(\hat{\boldsymbol{x}}_{t-1|t-1}, \boldsymbol{u}_t)} + \boldsymbol{v}_t$$

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^T + \boldsymbol{Q}_t$$

CORRECT

$$\hat{\boldsymbol{x}}_{t|t} = \hat{\boldsymbol{x}}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{y}_t - \underline{h(\hat{\boldsymbol{x}}_{t|t-1})})$$

$$\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T (\boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T + \boldsymbol{R}_t)^{-1}$$

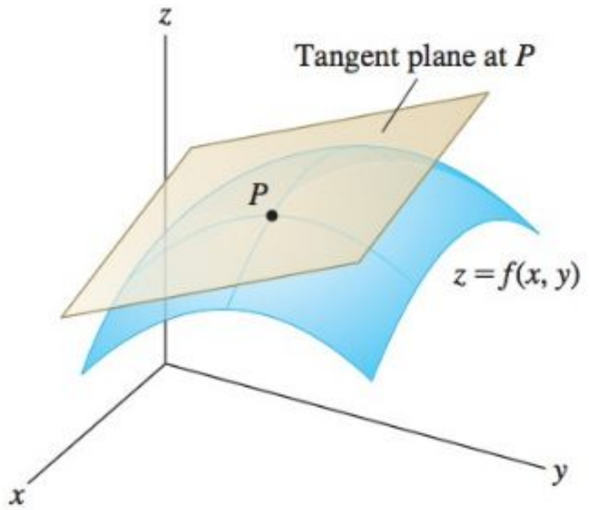
$$\boldsymbol{P}_{t|t} = (\mathbf{I} - \boldsymbol{K}_t \boldsymbol{H}_t) \boldsymbol{P}_{t|t-1}$$

- The only differences are the models marked by red, rest are the same as KF
- $\boldsymbol{F}_t$ and $\boldsymbol{H}_t$ are Jacobian matrices, i.e. first order partial derivatives of $\boldsymbol{f}$ and $\boldsymbol{h}$ (see next slide)

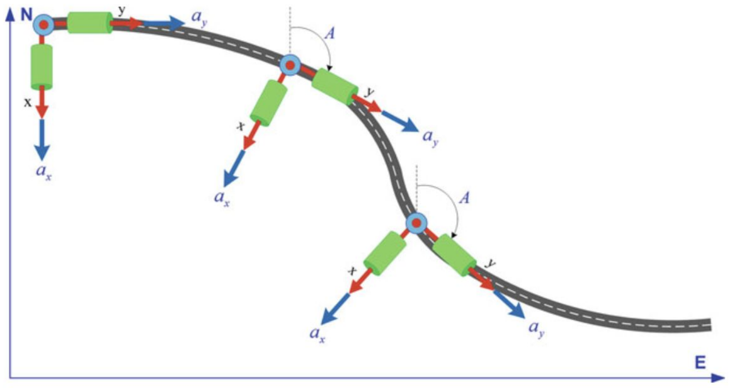
Jacobian Matrix

- The matrix of first-order derivatives of a vector function
- First order Taylor expansion
- The orientation of the plane tangent at a given point

$$\boldsymbol{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} \quad \boldsymbol{F} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$



EKF Example: 2D Positioning Revisited



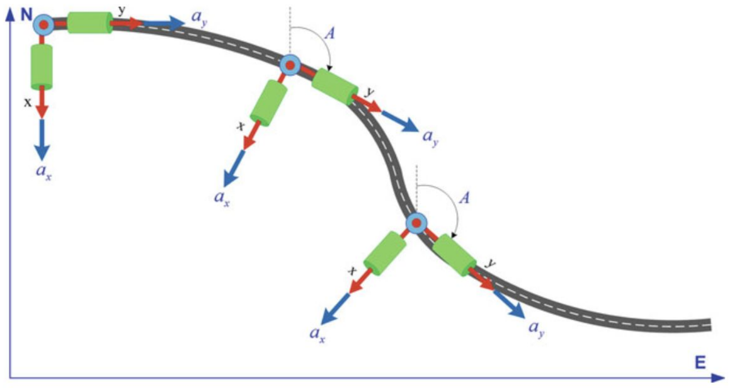
$$x_t = x_{t-1} + v_{t-1}\Delta t \cos(\theta_{t-1} + \Delta\theta_t)$$
$$y_t = y_{t-1} + v_{t-1}\Delta t \sin(\theta_{t-1} + \Delta\theta_t)$$
$$\theta_t = \theta_{t-1} + \Delta\theta_t$$

- Assume
  - We can measure speed and heading change (e.g. car speedometer + steering wheel input)
  - Intermittent GPS readings
- This is a non-linear state transition model, but measurement model is still linear!
  - So we'll need to only compute the Jacobian $\boldsymbol{F}_t$

$$\boldsymbol{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix}$$

$$f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) = \begin{pmatrix} x_{t-1} + v_{t-1}\Delta t \cos(\theta_{t-1} + \Delta\theta_t) \\ y_{t-1} + v_{t-1}\Delta t \sin(\theta_{t-1} + \Delta\theta_t) \\ \theta_{t-1} + \Delta\theta_t \end{pmatrix}$$

EKF Example: 2D Positioning Revisited



$$x_t = x_{t-1} + v_{t-1}\Delta t \cos(\theta_{t-1} + \Delta\theta_t)$$

$$y_t = y_{t-1} + v_{t-1}\Delta t \sin(\theta_{t-1} + \Delta\theta_t)$$

$$\theta_t = \theta_{t-1} + \Delta\theta_t$$

$$\boldsymbol{x}_{t-1} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} \qquad f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) = \begin{pmatrix} x_{t-1} + v_{t-1}\Delta t \cos(\theta_{t-1} + \Delta\theta_t) \\ y_{t-1} + v_{t-1}\Delta t \sin(\theta_{t-1} + \Delta\theta_t) \\ \theta_{t-1} + \Delta\theta_t \end{pmatrix}$$

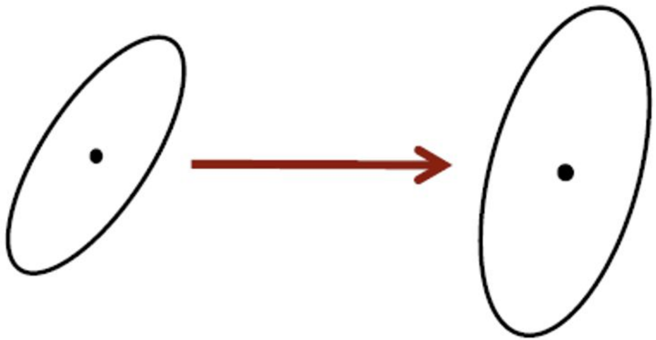
$$\boldsymbol{F}_t = \frac{\partial f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)}{\partial \boldsymbol{x}_{t-1}} = \begin{pmatrix} 1 & 0 & -v_{t-1}\Delta t \sin(\theta_{t-1} + \Delta\theta_t) \\ 0 & 1 & v_{t-1}\Delta t \cos(\theta_{t-1} + \Delta\theta_t) \\ 0 & 0 & 1 \end{pmatrix}$$

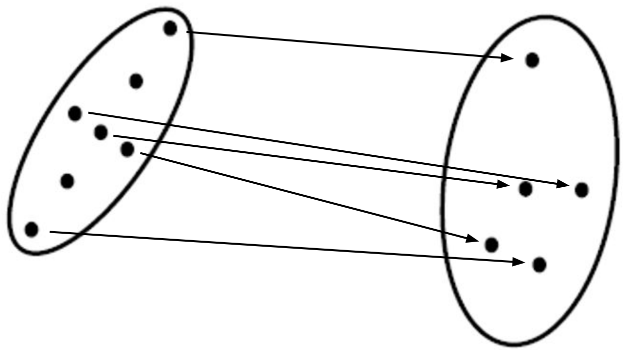
Extended Kalman Filter Conclusion

- Very powerful - allows us to use Kalman filters for many problems!
- But still an approximation...
- Best suited for functions that are near-linear in the short term
- Not suitable for multi-model belief distributions (non-linearity usually leads to such distributions)
- Initial state is more important
  - Re: the previous example: What if we do not know the initial heading of the car?
  - EKF may have trouble estimating it (too much non-linearity)
- Can we do a better approximation?

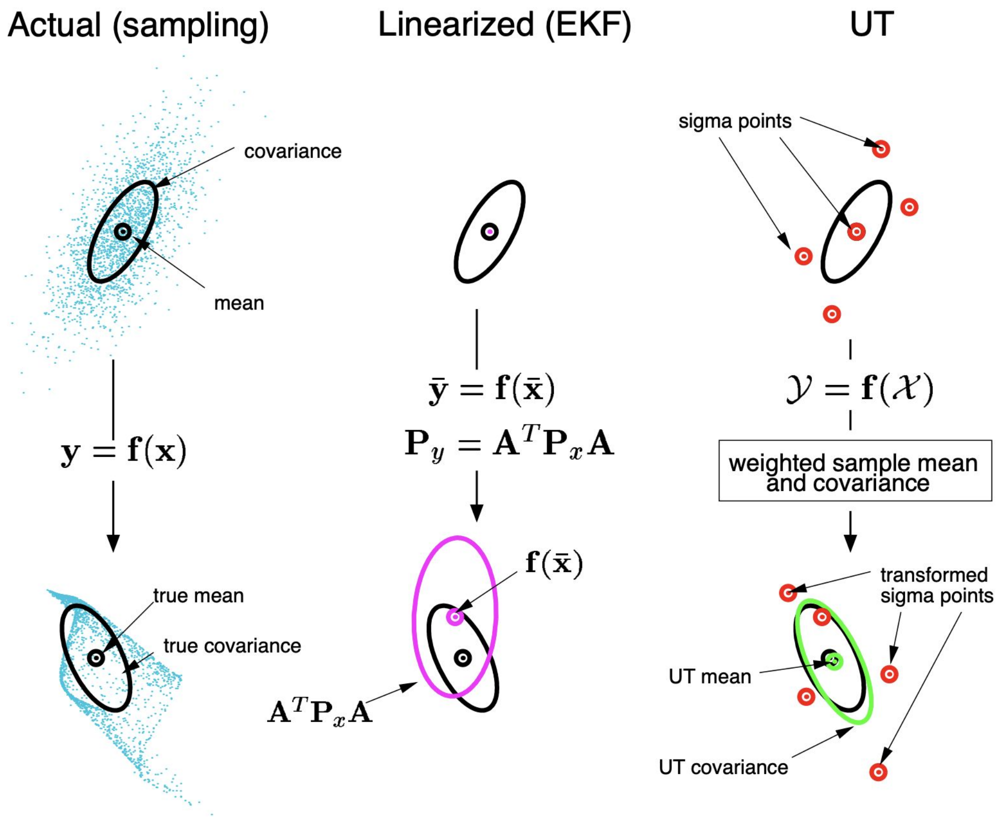
Unscented Kalman Filter (UKF)

- Another linearized Kalman filter, with a different approximation method
- Rather than linearizing around a single point, use **sigma points**
- Transform each sigma point through the non-linear function (aka Unscented Transform)
- Recompute Gaussian from the transformed and weighted points
- The output is similarly a Gaussian, but potentially a better one





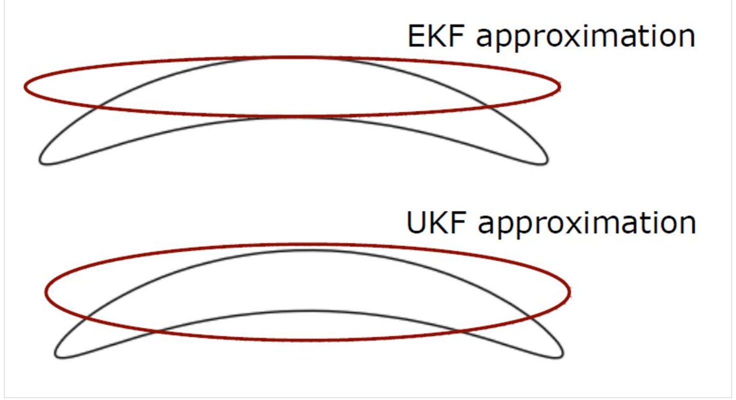
Unscented Transform Example



UKF Considerations

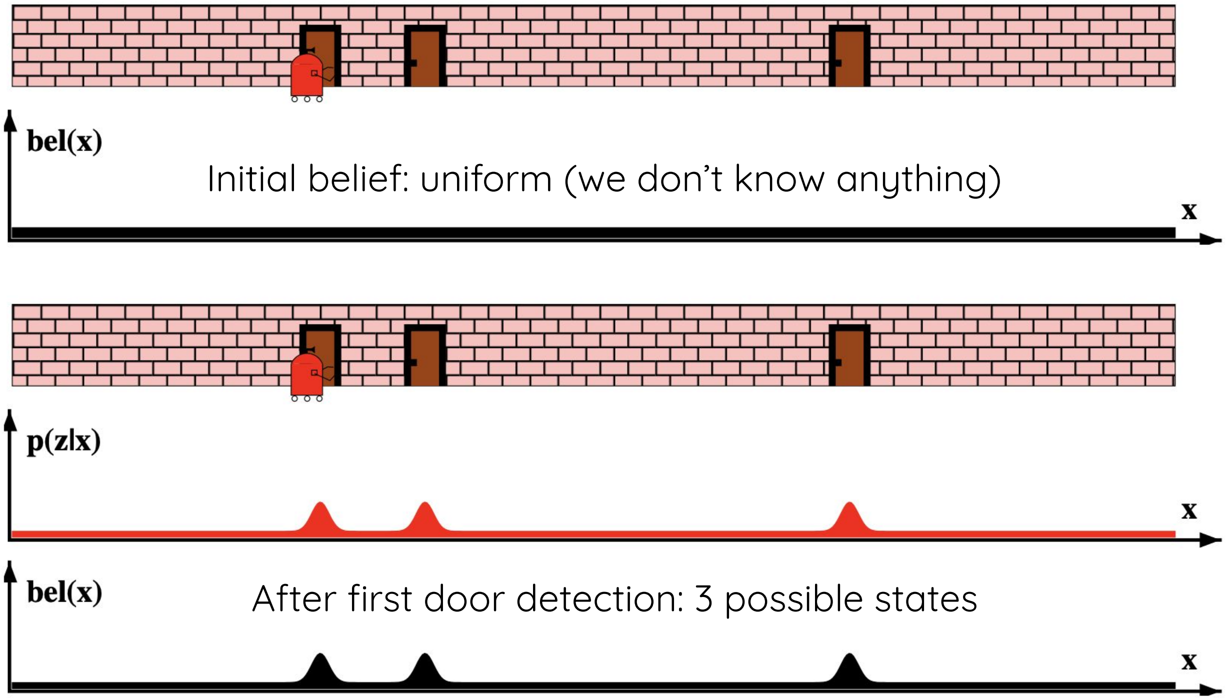
- Sigma points are deterministically selected to represent the Gaussian
- $\mathcal{X}$ is the matrix defining sigma points (i's are its columns)
- λ influences how far the sigma points are from the mean
- Points are weighted according to their distance from the mean
- UKF deals with non-linearity better, but it's not foolproof
- We still assume Gaussians and unimodal belief distribution

$$
\begin{aligned}
\mathcal{X}^{[0]} &= \mu \\
\mathcal{X}^{[i]} &= \mu + \left(\sqrt{(n+\lambda)\,\Sigma}\right)_i \qquad \text{for } i = 1, \ldots, n \\
\mathcal{X}^{[i]} &= \mu - \left(\sqrt{(n+\lambda)\,\Sigma}\right)_{i-n} \qquad \text{for } i = n+1, \ldots, 2n
\end{aligned}
$$

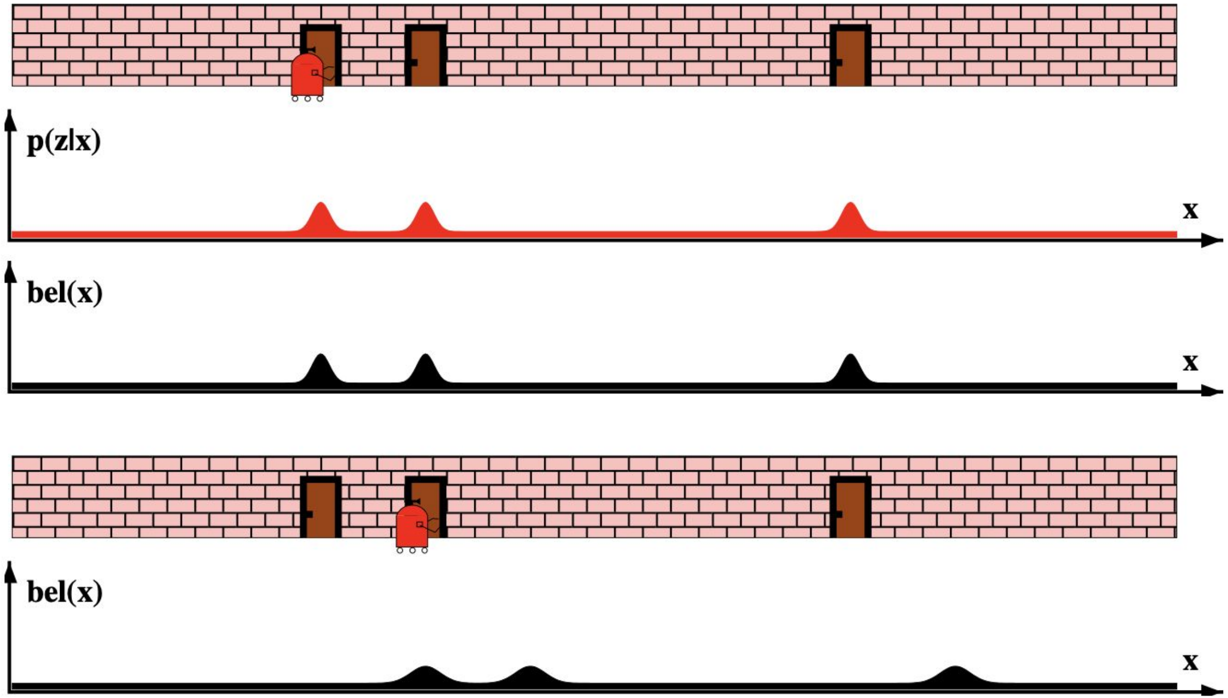


Multimodal Belief Example

Imagine a robot which can detect doors, but it doesn't know which door it is

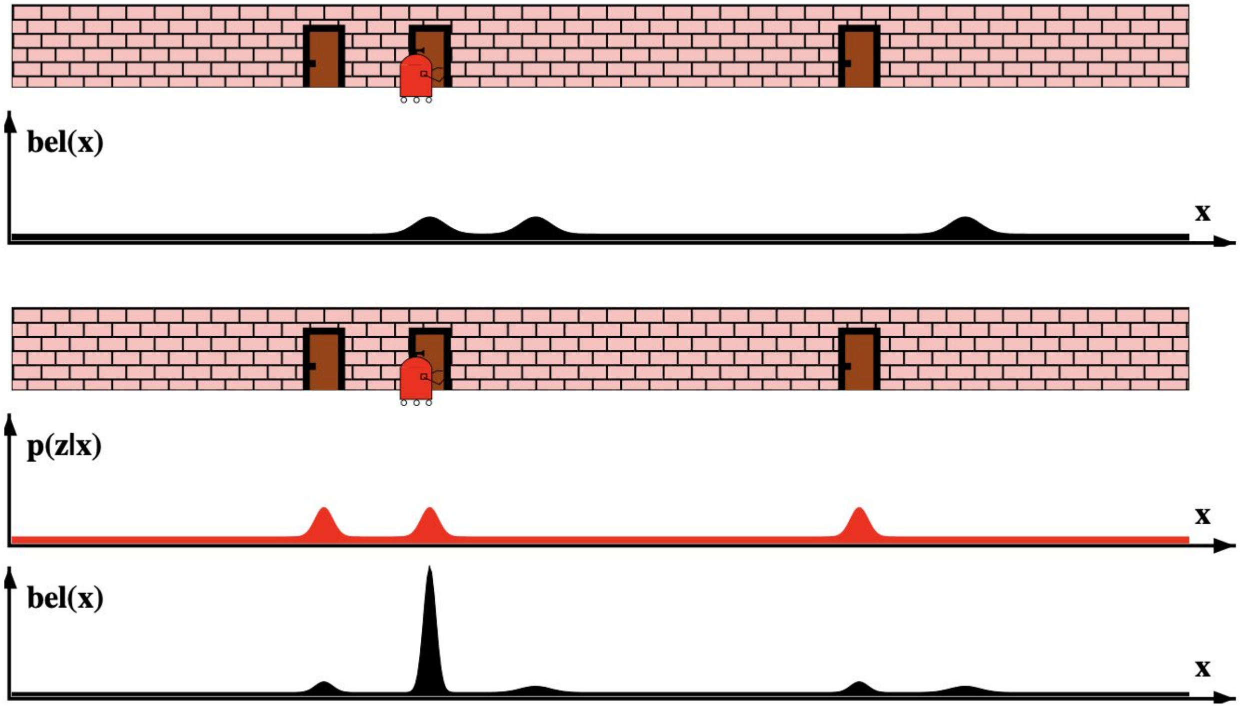


Multimodal Belief Example



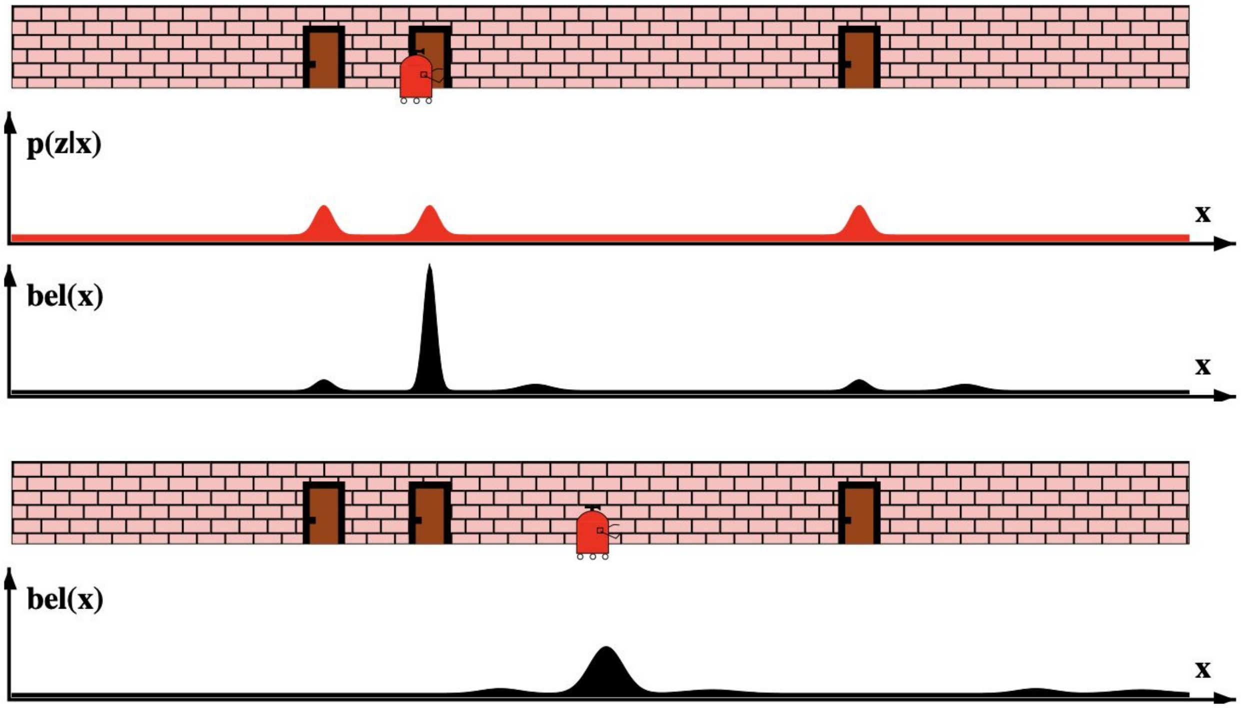
After some movement: Note how the estimates widen
(incorporating potential error due to motion)

Multimodal Belief Example



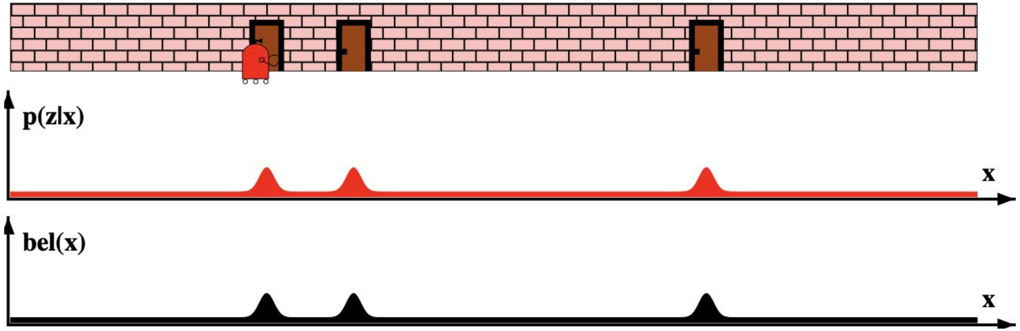
After second door detection: Estimate gets sharper
(the peak at the first door is low due to our prior belief)

Multimodal Belief Example



We have now converged on a state we can be confident in

How to represent multimodal belief?



- Such a multimodal distribution cannot be represented accurately by a single Gaussian
- Maybe mixture of Gaussians, but what if measurement model is not Gaussian as well?
  (or a complex motion model may disrupt the individual Gaussians)
- We need a generalizable solution for arbitrary distributions
- Next: Particle filter!

Particle Filter

- aka Sequential Monte Carlo
- Represent belief by a set of samples also called "particles"

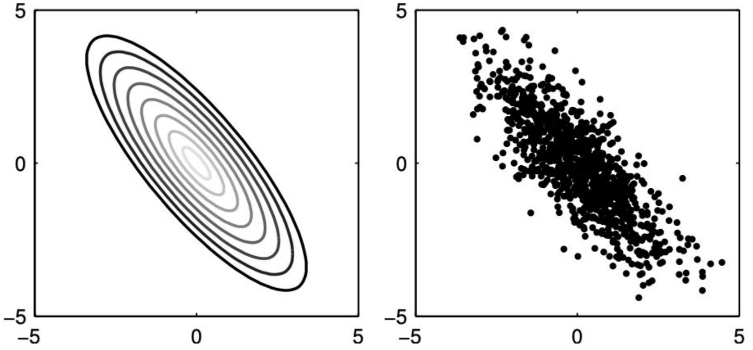
$$\mathcal{X}_t \quad := \quad x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}$$

M is the number of particles

- Particles are individual "hypotheses", the likelihood of including a particle in the set should ideally be proportional to the posterior belief

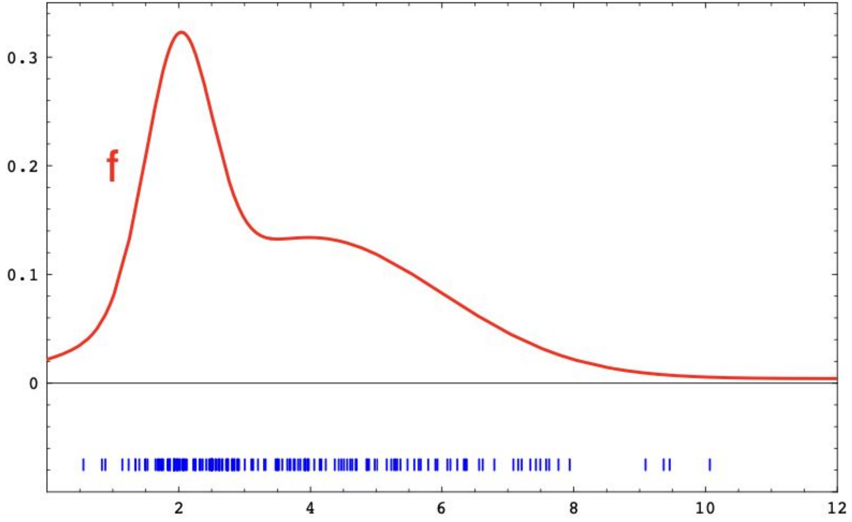
$$x_t^{[m]} \quad \sim \quad p(x_t \mid z_{1:t}, u_{1:t})$$

- Implication: Denser regions should be populated by more particles
- As M goes to infinity we represent the belief better (practically we don't need as much)



How to sample from Belief?

- Belief is an arbitrary distribution, could be multimodal and complex
- So, sampling from such a distribution is not a straightforward task



How do we generate samples from $\boldsymbol{f}$?

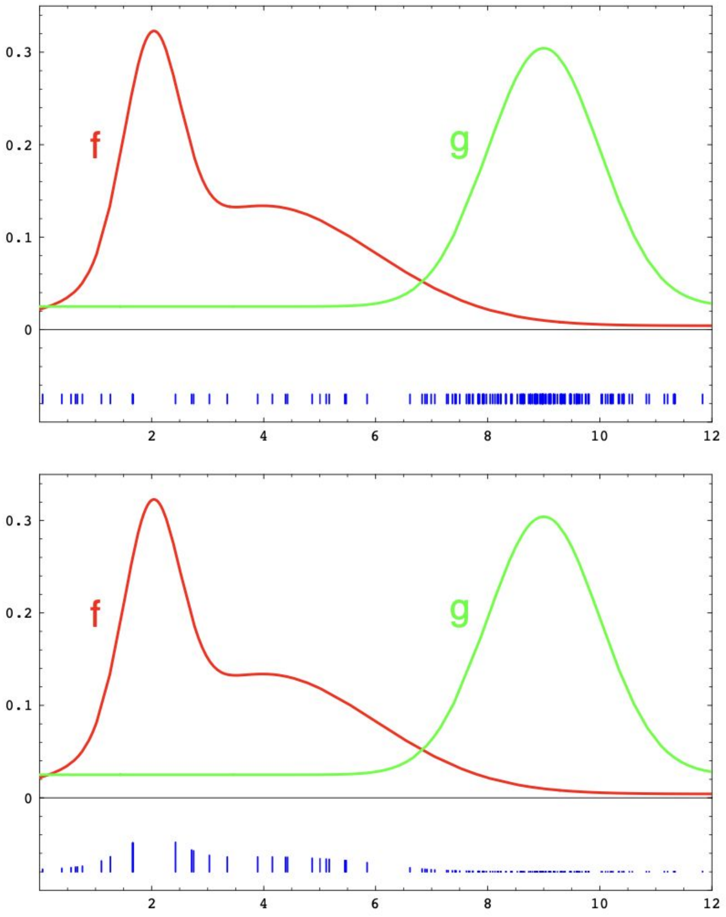
Importance Sampling

- Instead of sampling from target distribution ***f*** directly, sample from a proposal distribution ***g***
- Then, the samples are weighted according to ***f***

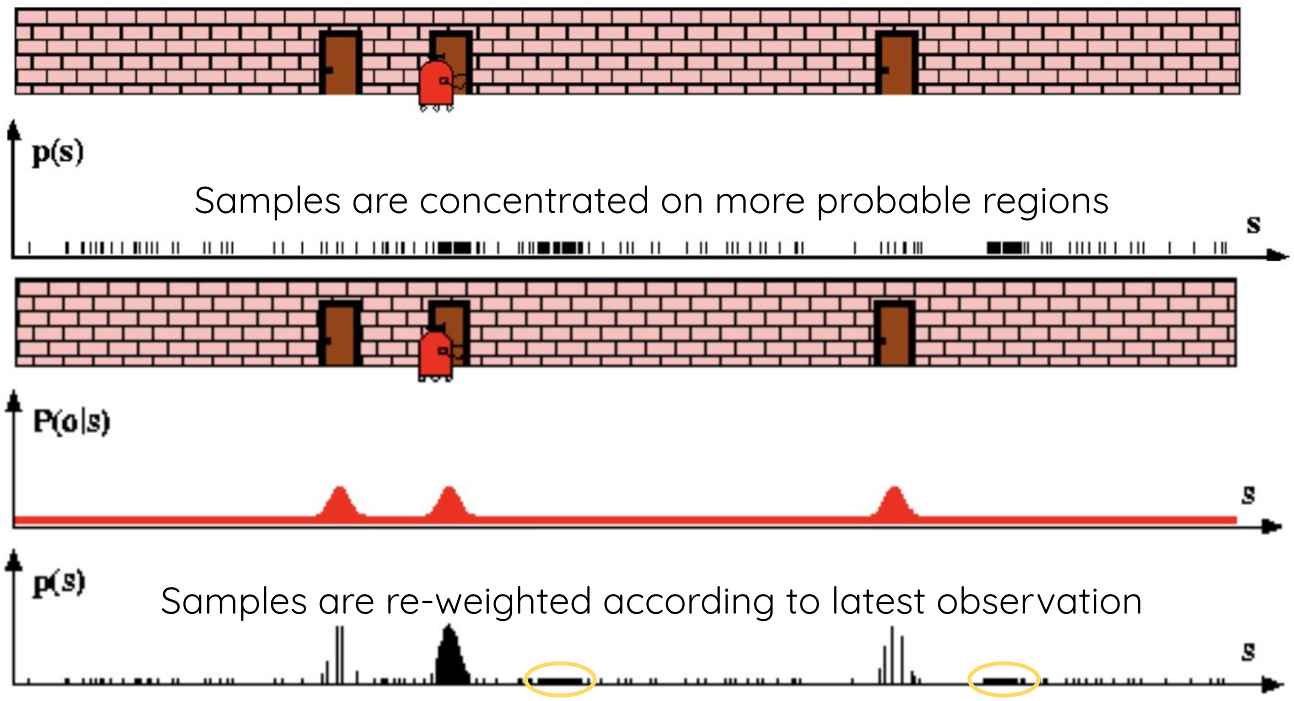
$$w^{[m]} = \frac{f(x^{[m]})}{g(x^{[m]})}$$

- Selection of ***g*** is important, closer to ***f*** the better
- But, even for non-ideal ***g***, the final weights resemble ***f***
- There is one problem: Improbable regions may have the majority of the particles!

(it gets worse and worse after many iterations)



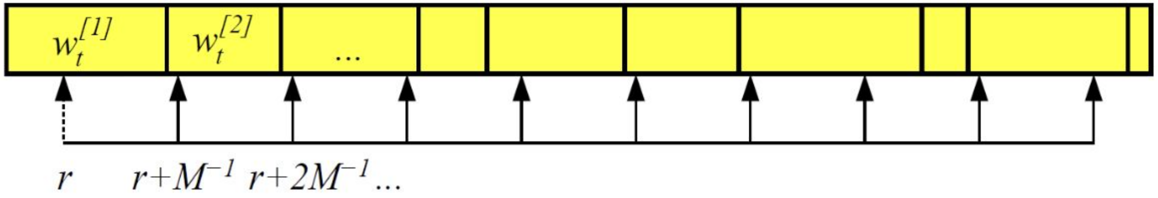
Multimodal Belief Example Revisited



Yellow regions contain a lot of samples even though not very probable, what to do?

Resampling

- The problem is also called "particle degeneracy": Lack of particles corresponding to the true state (since most particles are in improbable regions, i.e. their weights are close to zero)
- The solution: **Resampling**!



- Resample particles according to their weights! ("survival of the fittest")
- A particle with a high weight has a higher chance to be selected (even duplicated)
- Particles with lower weights won't be selected at all (we'll get rid of improbable particles)
- There are many resampling techniques: Low Variance Sampling (above example), Stochastic Universal Sampling, Roulette Wheel… (we won't go into details)

Particle Filter Algorithm

Algorithm Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$

for $m = 1$ to M do

sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$

$w_t^{[m]} = p(z_t \mid x_t^{[m]})$

$\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$

endfor

for $m = 1$ to M do

draw i with probability $\propto w_t^{[i]}$

add $x_t^{[i]}$ to $\mathcal{X}_t$

endfor

return $\mathcal{X}_t$

Prediction: Advance particle to its next a priori state

Correction: Compute particle weight according to measurement

Resampling
(we don’t have to do it at each iteration, in which case the weights are reset to 1/M here)

When to resample?

- We may do it at every step, but what if we lose an important particle by chance?
- Resampling is a destructive operation, improves our estimate but alters it
- Resampling is also a computationally heavy operation
- A better strategy is to do it based on **effective sample size**:

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}_k^{(i)})^2}$$

- All weights equal to each other: N_{eff} = N
- All weights zero except one particle with weight 1: N_{eff} = 1
- Idea: Resample only when effective sample size is lower than a threshold

Particle Depletion

- Even if we do resampling, we may still end up with no particles around the true state: Maybe due an unlucky streak of bad measurements…
- The solution: **Particle Injection**
- Idea: Add a small amount of additional particles randomly
  - Could be generated around the latest measurement
  - Could be generated totally randomly (similar to prior belief)
- Advantage is its simplicity, but theoretically we are altering our posterior belief (but negligible if injected particles are few)
- A famous problem: Kidnapped Robot Problem
  - What if a robot is carried to a different location?
  - We won't be able to trust the motion model (since robot is carried)
  - We want to be able to correct our estimate towards the true state: particle injection is a solution

Practical Applications of Data Fusion

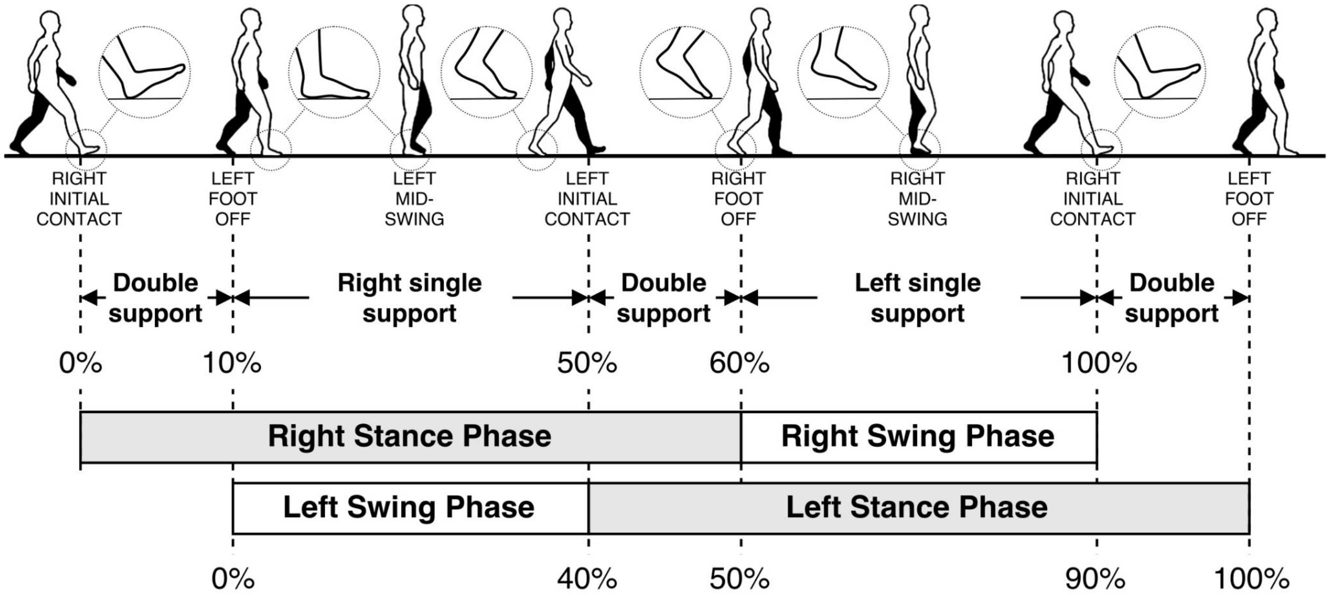
- We will now explore some applications of fusion methods for positioning
- We've already given some examples that can be readily applied to the real world (1D, 2D positioning…), but there are some additional considerations
- How to convert inertial sensor measurements to viable inputs?
- How to cope with inertial sensor errors?

Using Inertial Sensors for the Motion Model

- Inertial navigation equations can be used as is for gyroscopes
  - Errors are manageable, only one integration required
- We can’t say the same thing for accelerometer…
  - If you have frequent and reliable correction measurements, you can still do
    Example: RTK (Real-Time Kinematics) systems -> High quality, high frequency GPS + inertial navigation equations (double integration…)
  - Frequent corrections mean small delta-time, which means double integration error remains bounded
  - Otherwise, not feasible
  - But there are some options for **pedestrian dead reckoning**…

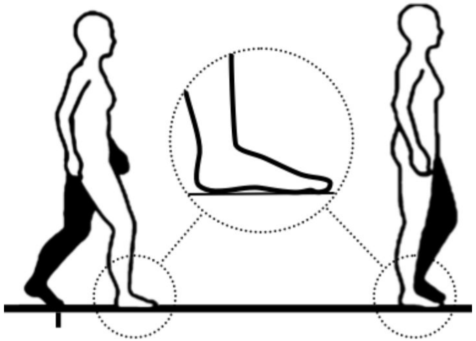
Pedestrian Dead Reckoning

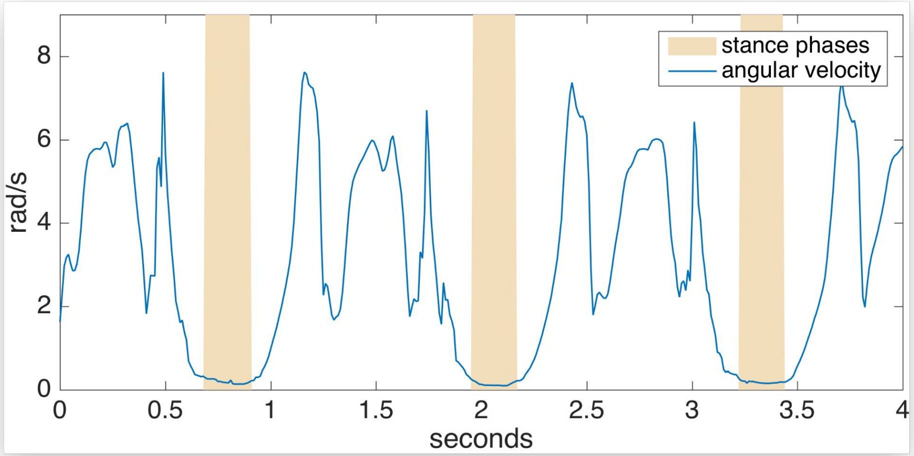
- Assuming we are tracking walking humans
- Human gait has some properties that we can exploit



Zero-Velocity Updates (ZUP)

- Applicable for sensors fixed to the foot (e.g. sensors embedded in shoes, or simply IMUs strapped to the feet)
- There is a brief phase where foot is stationary within the gait cycle
  - Foot is completely flat on the floor, velocity is known to be zero
- Detectable via raw gyro readings via thresholding (also possible with accelerometer)





Error-State Kalman Filter

- Rather than tracking the actual state via KF, track the error
- Actual state (position, velocity, attitude) is tracked via standard navigation equations
- ZUP measurements are used to correct velocity
- KF automatically corrects position as well!
  (e.g. if velocity overshot, position will be corrected backwards)

PREDICT

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^{\mathsf{T}} + \boldsymbol{\Sigma}_w$$

CORRECT

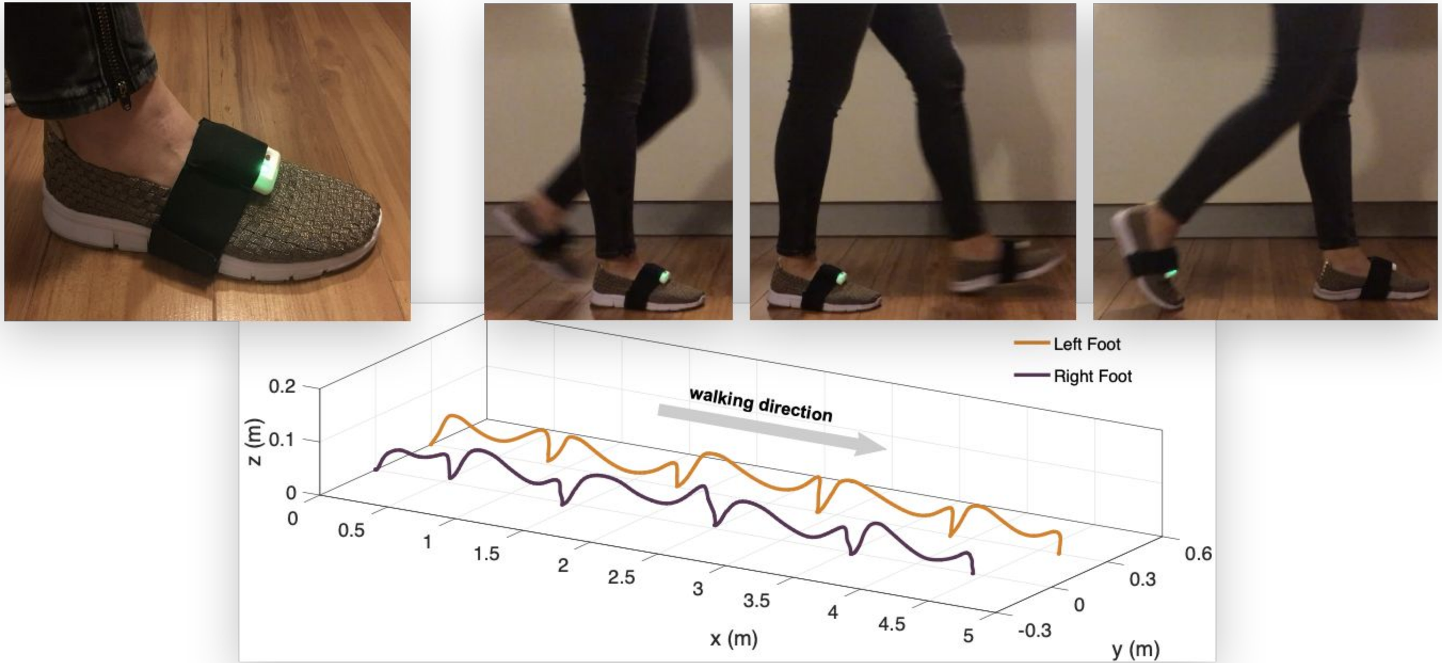
$$\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1} \boldsymbol{H}^{\mathsf{T}} \left(\boldsymbol{H} \boldsymbol{P}_{t|t-1} \boldsymbol{H}^{\mathsf{T}} + \boldsymbol{\Sigma}_v \right)^{-1}$$

$$\delta \hat{\boldsymbol{x}}_t = \boldsymbol{K}_t \left(\boxed{\mathbf{0}} - \boldsymbol{H} \hat{\boldsymbol{x}}_t \right)$$

$$\boldsymbol{P}_{t|t} = \boldsymbol{P}_{t|t-1} - \boldsymbol{K}_t \boldsymbol{H} \boldsymbol{P}_{t|t-1}$$

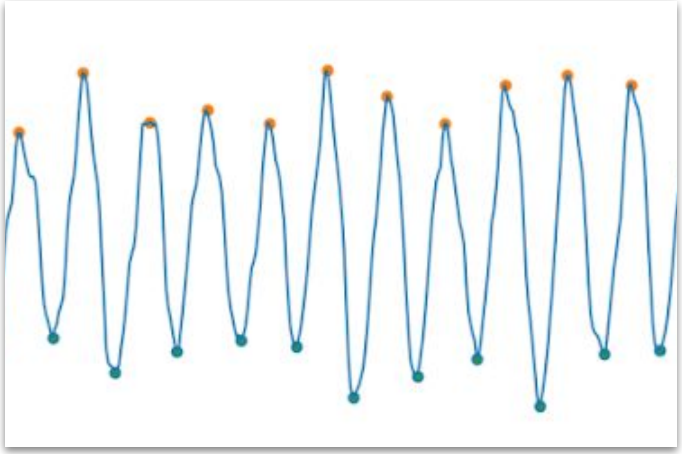
- ZUP is marked with red: just a vector of zeros! (in 3 axes)
- NOTE: No need to track mean ($\boldsymbol{x}$), since mean error is always zero, all we need is covariance
- $\delta \hat{\boldsymbol{x}}_t$ is then applied to the separately tracked actual state

Zero-Velocity Updates Example

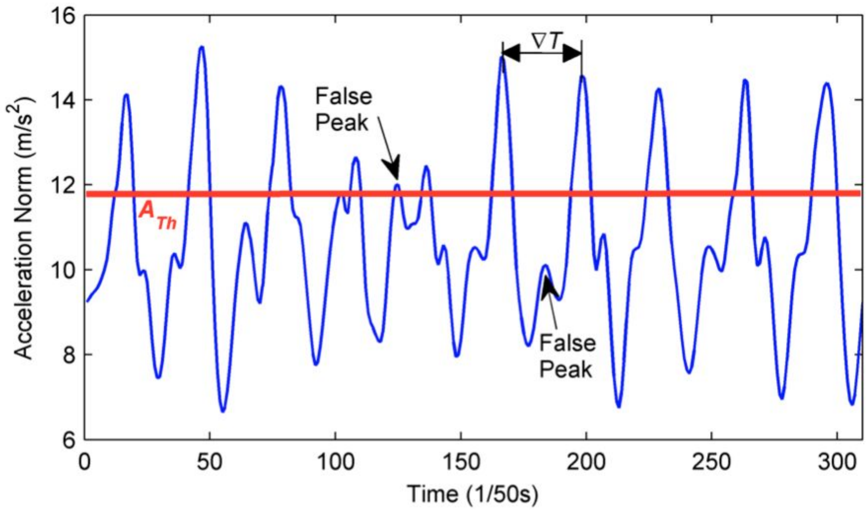


Inertial Tracking for Mobile Phones

- ZUP is not possible, since phones are usually not strapped to foot...
- Assuming the phone remains more or less stationary with respect the the user (e.g. handheld, strapped to a belt, in the pocket)
- Attitude can be projected to the Earth-tangent plane (converted to heading/yaw)
- How to estimate position?
- **Step detection** to the rescue!



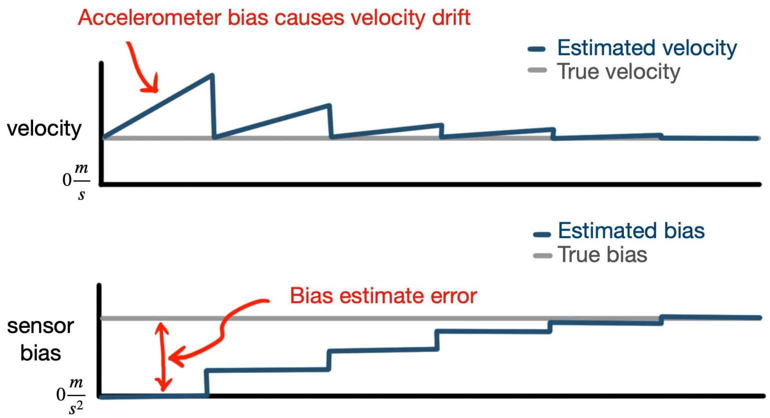
Step Detection



- Human gait is periodic
- Naive approach: Peak/valley detection
  - Typically performed on the magnitude of acceleration (no need for attitude estimation even!)
  - If we can compute attitude, we can extract only the z component (may be more reliable)
  - Works really well, but we've gotta be careful about false positives!
- Detecting steps is not enough, we also have to estimate step length...

Step Length Estimation

- All we can do is approximate, but hopefully typical step length remains in a limited range
  - We can assume a fixed step length
  - May be good enough provided we have accurate correction measurements
- Filters can learn the undershoot/overshoot!
  - Start with a fixed estimate, it gets better over time
  - We should include a term for this in the filter state vector (aka **bias term**)
  - The same concept can be applied to learn other sensor bias errors!
  - Bias estimation is a difficult task though
  - Correction measurements should be accurate
- Not ideal initially, can we do better?



Step Length Estimation

- There are existing approximation models that can give us a somewhat better estimate
- Intuition: Step length influences acceleration (e.g. longer steps tend to produce more acceleration)
- Two very commonly used models:

WEINBERG

$$k \cdot \sqrt[4]{a_{\max} - a_{\min}}$$

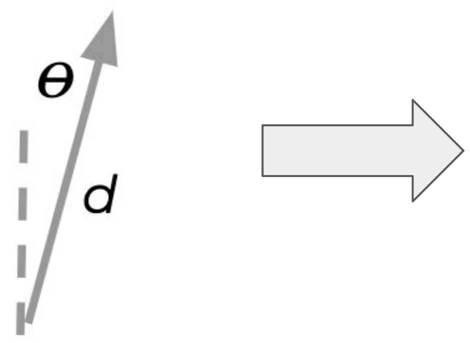
SCARLET

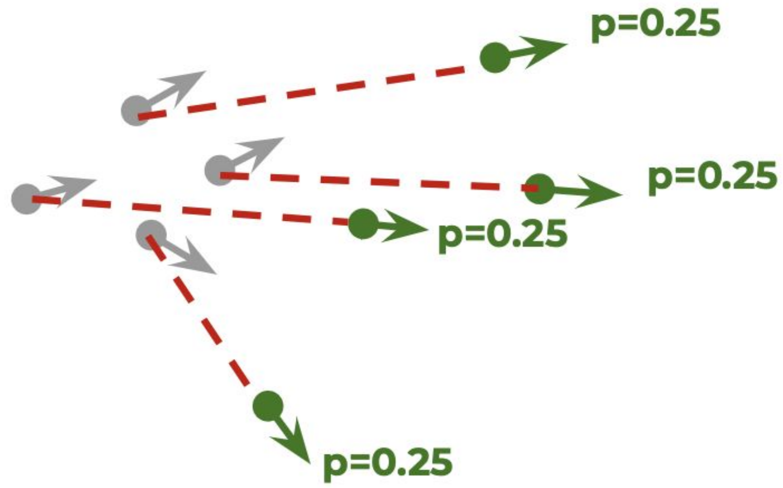
$$k \cdot \frac{\left(\sum_{i=1}^{N} \left|a_i\right| / N\right) - a_{\min}}{a_{\max} - a_{\min}}$$

- max/min acceleration are determined within the window of a detected step
- ***k*** is a tunable coefficient
- Time could also be incorporated into the model (e.g. longer steps tend to be quicker)
- Not foolproof, but may be better than a fixed estimate
- Could be similarly augmented via bias estimation (for instance we could estimate ***k***)

Particle Filter Positioning Example

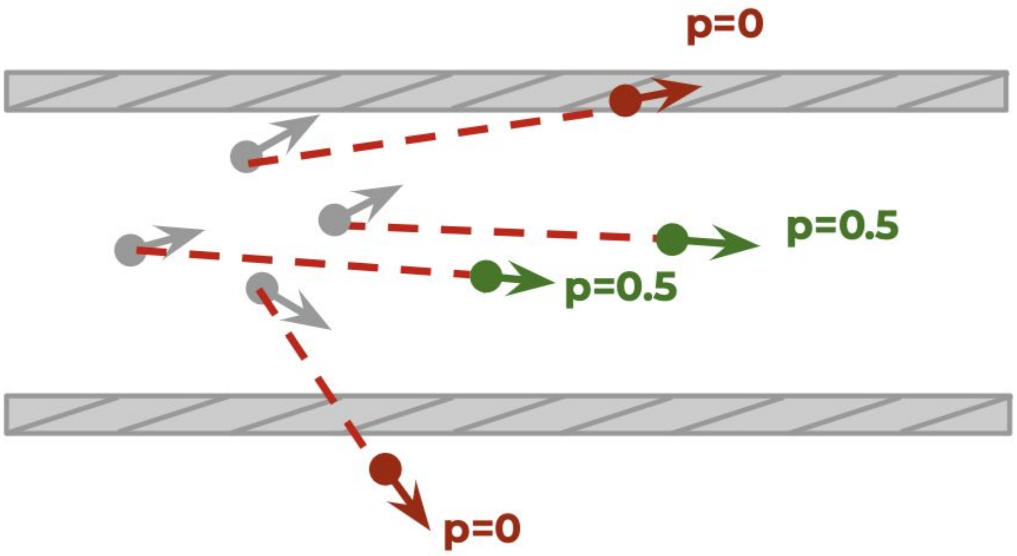
- A simple motion model consisting of displacement + heading change
  - Similar to velocity + heading change model, could easily be converted to/from
  - Suitable for step detector output (step length = displacement)
- Particles are propagated with these relative measurements (+ some noise)





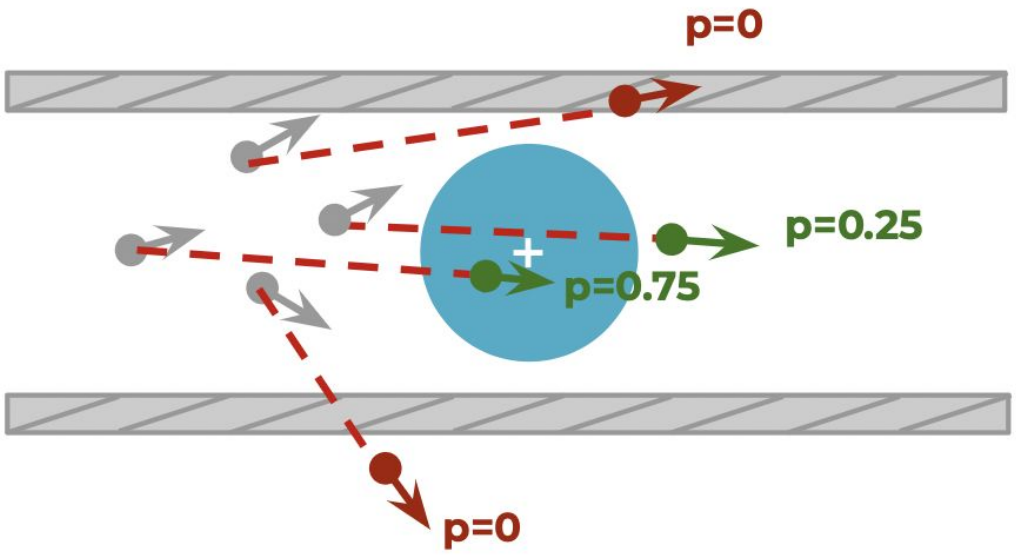
Particle Filter: How to incorporate map info?

- Maps have unreachable regions such as walls, can we use this to improve our estimate?
- A highly non-linear problem, posterior distribution can become very complex
- Solution: Kill particles that bump into walls (or make their weights low)



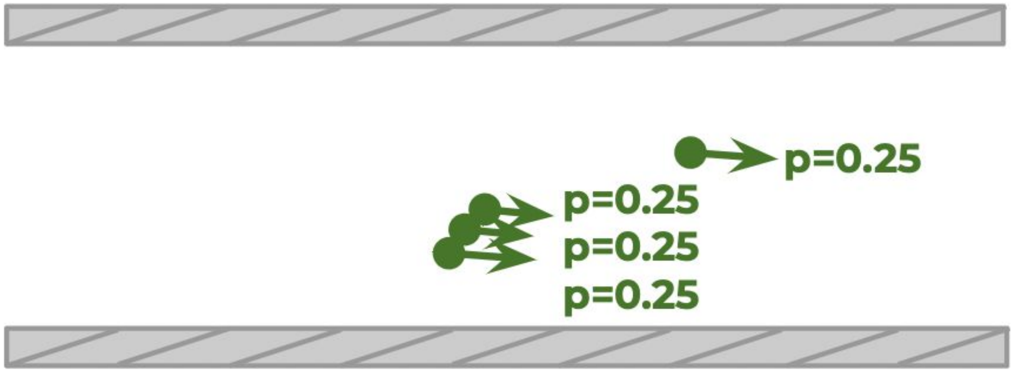
Particle Filter: Correction Example

- An absolute position measurement can then be used for correction
- Particle weights are updated according to the proximity to the new measurement
- Measurement model could be Gaussian, or anything really



Particle Filter: Resampling Example

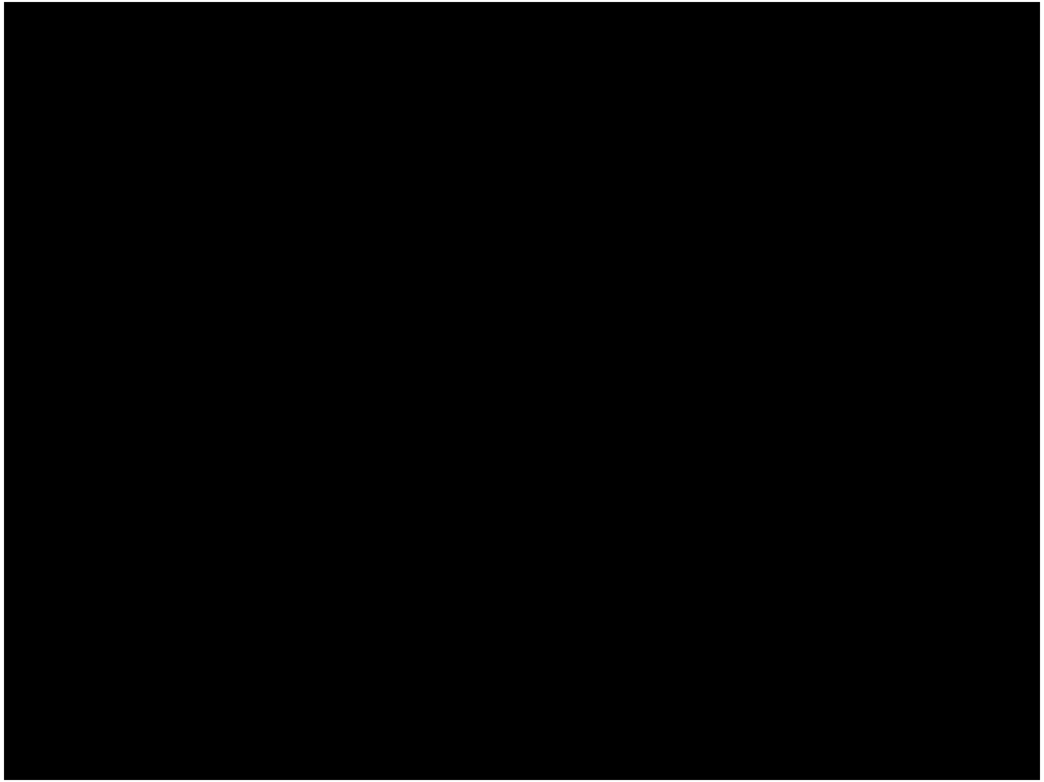
- Particles are then resampled; the most probable particles are duplicated
- Weights are reset to be equal
- Initially the duplicated particles are in the same position, but in the next predict iteration, random noise will separate them



Real World Data Fusion Example

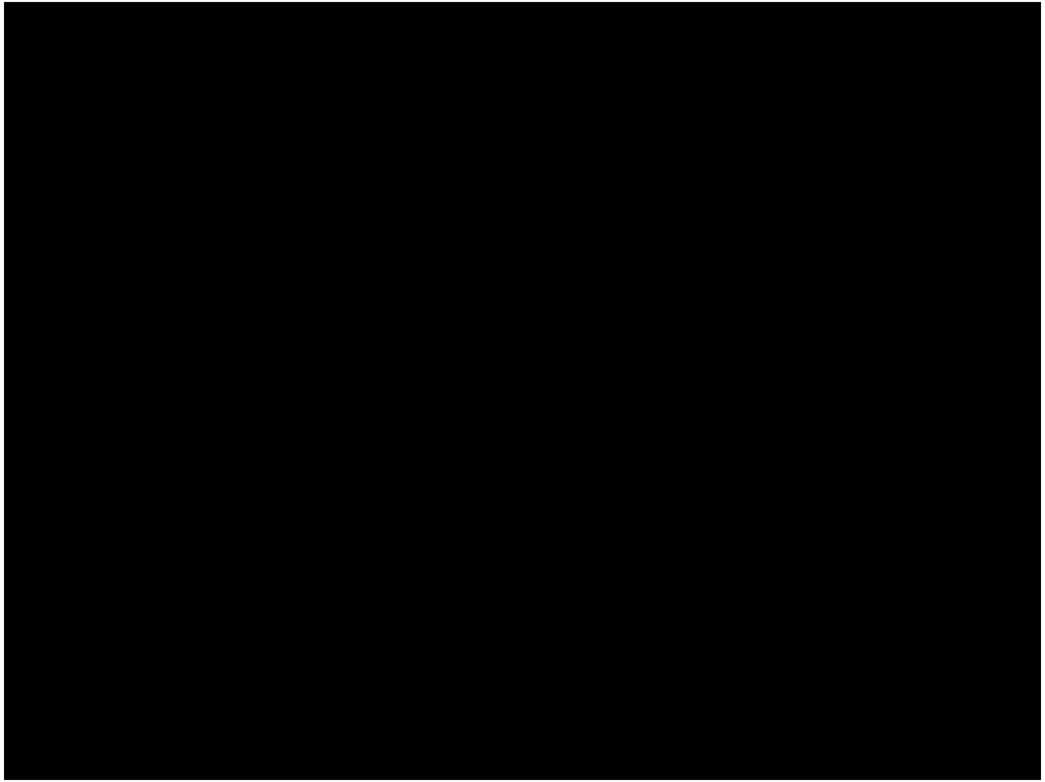
- Tracking a hand-held phone
- Inertial data: Step detector + gyro-based heading estimation (no magneto)
- Absolute position measurements: Bluetooth beacons trilateration
- Particle filter for fusion (state: x, y, heading)
- Also uses map information (obstacles) to improve estimate
- We'll see **inertial only**, **Bluetooth beacons only**, and finally **fusion**

Inertial Only



Bluetooth Beacons Only

Fusion



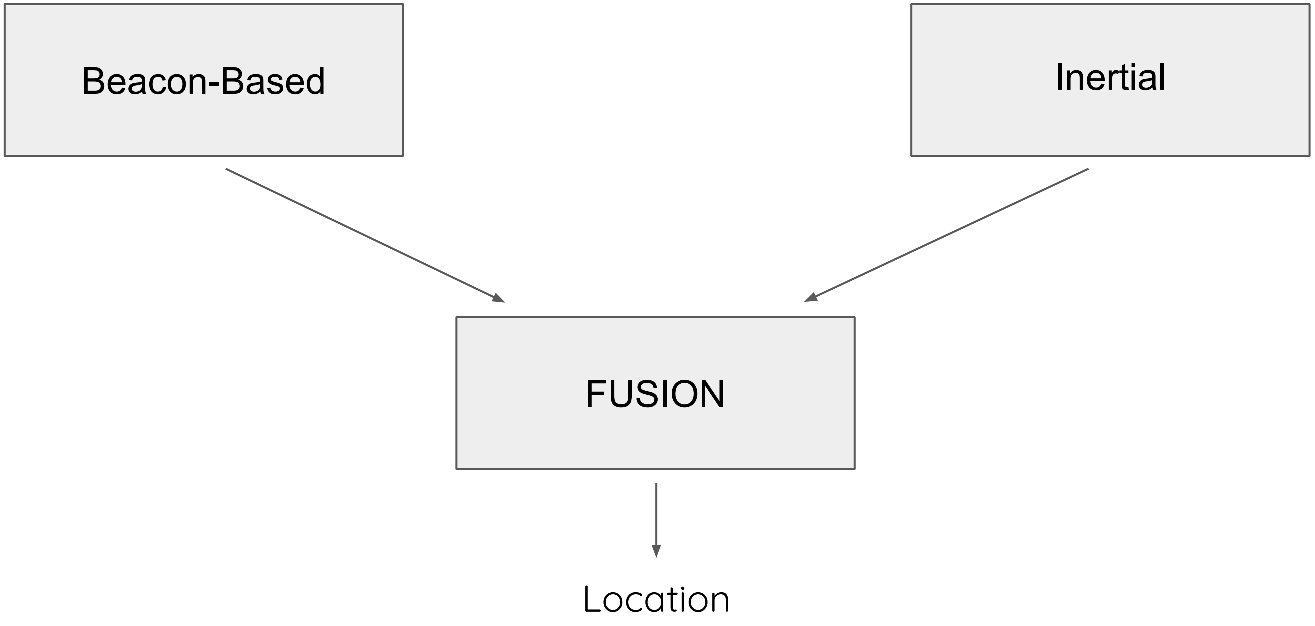
Fusion

Absolute

- Good for long-term (> 1 minute)
- Anchored on the map

Relative

- Good for short-term (seconds)
- Not anchored on the map



Detecting Floor

- Indoor positioning demands floor detection, as buildings have multiple floors
    - A single coordinate may correspond to multiple floors, which is it?
- Radio-based positioning can give an idea
    - Lateration/angulation can operate on 3D (if you are using a suitable tech)
    - The floor info can be embedded into the fingerprinting map
    - The receiver tends to pick up beacons/AP on the same floor more strongly
- Indoor spaces can be very challenging: Mezzanines, vertical spaces, inclined surfaces…
    - Radio-based methods may be insufficient
    - Fluctuations may make it difficult to converge on a solution (especially for RSSI-based methods)
- An augmenting solution: **Barometer**!

Barometer

- Most phones have a digital barometer nowadays
- Measures atmospheric pressure
  - Measurements are absolute, however pressure at the same location/altitude changes according to weather, time of day...
- Surprisingly accurate for measuring relative altitude changes
  - We can roughly convert the difference between two pressure values to relative altitude change
  - Accurate on the order of < 1 meter
- Relative altitude change can be incorporated into a motion model!
  - Can then be fused with other radio-based data
  - Can handle momentary fluctuations and bias
  - Should be careful on how much to trust it (what if our first estimate is wrong?)

Conclusion

- Data fusion methods can be used to combine complementary data (one filling in the weaknesses of the other)
- Fusion can take place either in the motion model (as control input), or in the measurement model (multiple measurement models are possible!)
- Non-linearity and the posterior distribution characteristics are important for method selection
- Fusion methods are only as powerful as how accurate you select your error models!
- Particle filters are very powerful, but one should be mindful of the computation cost; for many problems a linearized KF may be just as good!