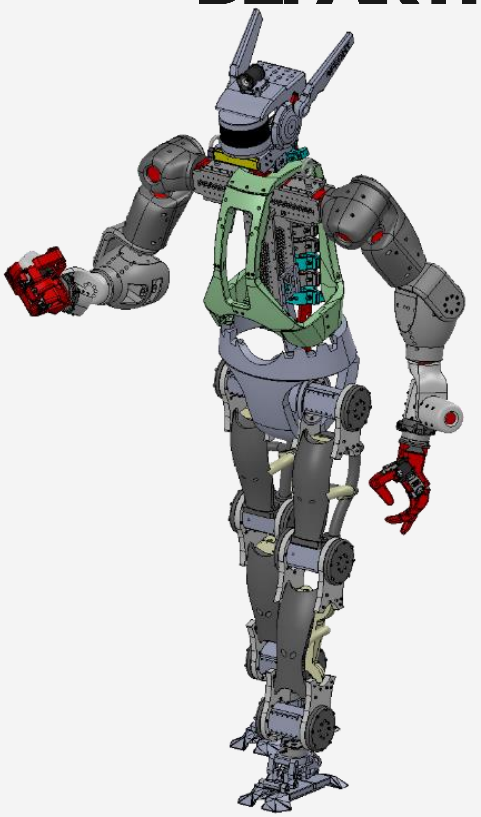
UNIVERSITY OF CALGARY

8: Estimation II

DEPARTMENT OF MECHANICAL ENGINEERING



Mobile Robotics

(ENME 650)

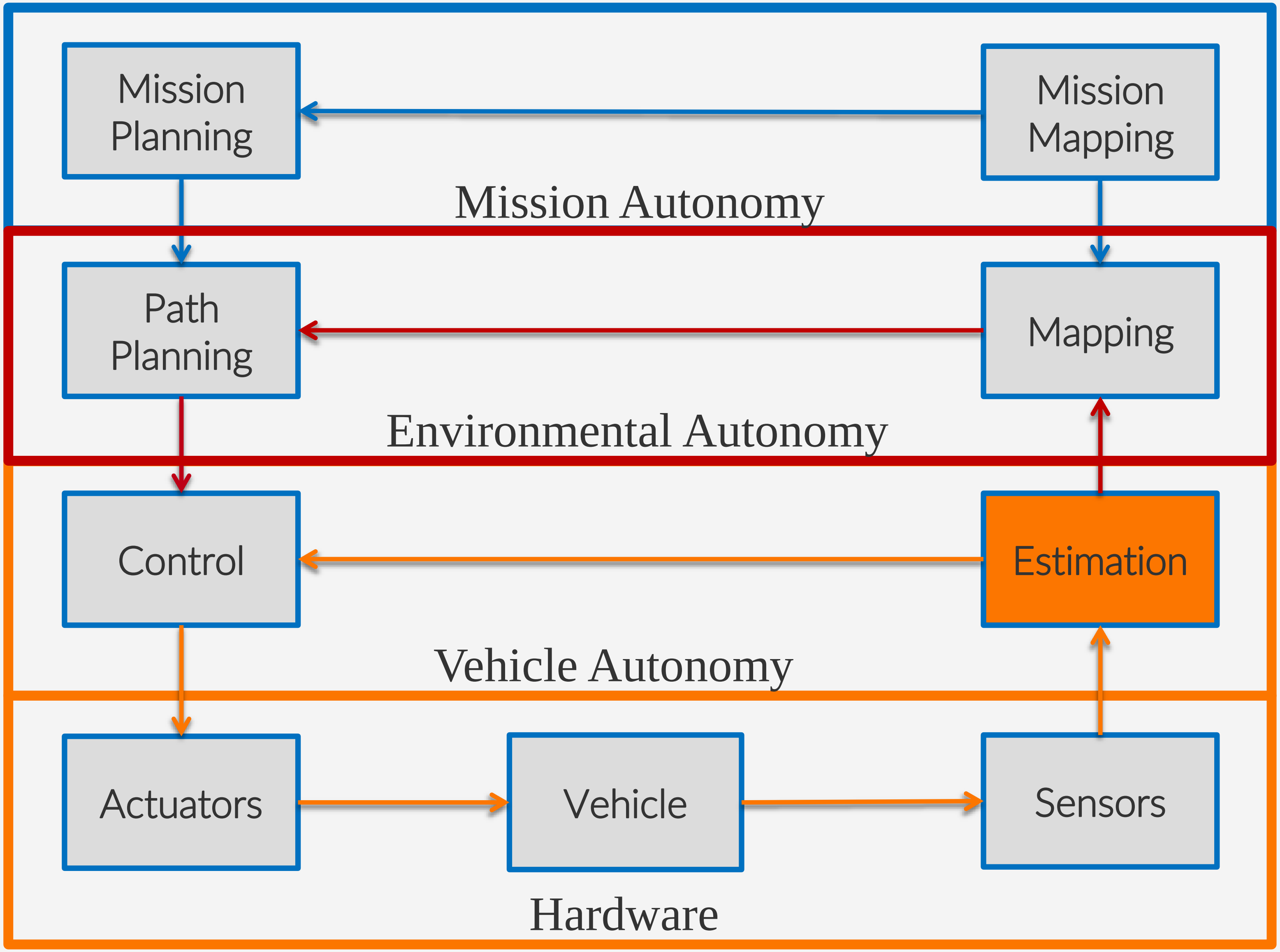
Instructor:

DR. ALEX RAMIREZ-SERRANO

Office: MEB 523
Phone: 220-3632
E-mail: aramirez@ucalgary.ca

CANADA

COMPONENTS


Mission Planning
Mission Mapping
Mission Autonomy
Path Planning
Mapping
Environmental Autonomy
Control
Estimation
Vehicle Autonomy
Actuators
Vehicle
Sensors
Hardware

OUTLINE

- Review of Probability
- Bayes Filter Framework
- Kalman Filter
- Extended Kalman Filter
- Unscented Particle Filter
- Particle Filter

PARTICLE FILTERS

- The Bayes Filter Framework has now been adapted to:
  - Kalman Filter (KF)
    - *Linear models with additive Gaussian Noise*
  - Extended Kalman Filter (EKF)
    - *Nonlinear models with additive Gaussian noise*

These filters are called ***Parametric filters*** *because we have parametrized the distribution with a mean and a covariance.*

- Both continuous Gaussian methods are computationally appealing (*i.e., can run in real-time*)
  - Even for large numbers of state & measurement variables
  - Benefit arises from ability to maintain Gaussian beliefs
    - *Track only mean and covariance throughout filtering process*

PARTICLE FILTERS

- In both cases, modeling requirements rule out a significant portion of real systems
  - *Nonlinear systems where linearization is a poor approximation over distribution range*
  - *Systems with multiple reasonable hypotheses*
- Alternatives include non-parametric filters (*e.g., when we cannot assume gaussian distributions, doesn't make sense to linearize models, etc.*):
  - Filters that do not track distribution parameters
  - Bayes/Histogram Filter
    - *Discrete state systems with known probabilities*
    - *Explodes computationally for higher dimensional models*
  - Particle Filter
    - *Maintain a sample set representation of beliefs*
    - *Results can be poor in higher dimensional models*
    - *Also called Sequential Monte Carlo methods*

PARTICLE FILTERS

- Modeling assumption(s):
  - Instead of assuming Gaussian, tracking μ_t, Σ_t , generate a set of sample states from each distribution
    - Each sample is a hypothesis about the current state
    - Properties of the whole collection of samples are used to generate estimates
  - Not possible to sample belief distribution directly, must apply Importance Sampling (*to find* ***a belief*** *from the sample set that indicates what the solution is - where the robot is)*

PARTICLE FILTERS

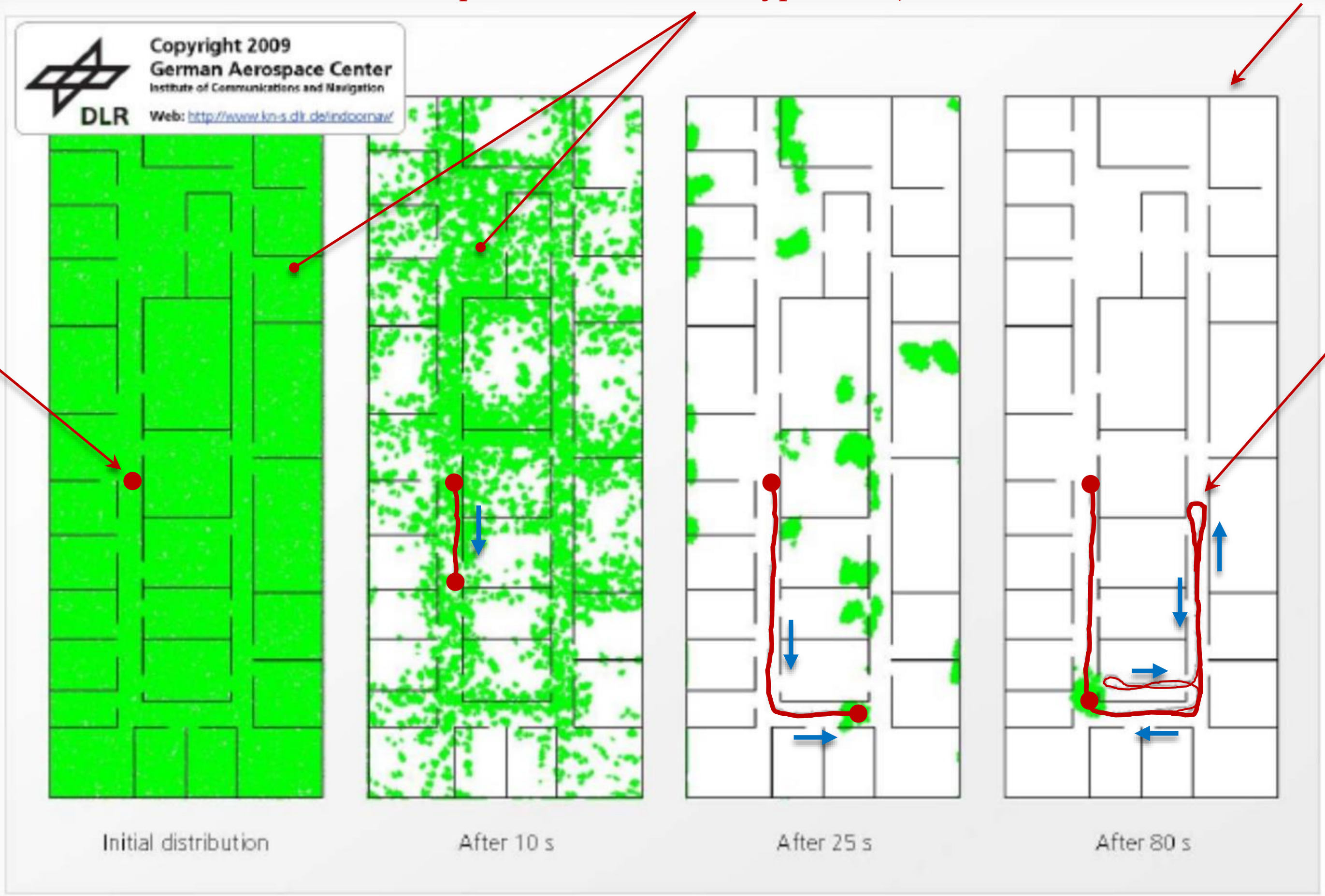
- Example particle sets – density of points defines probability

Set of particles (e.g., 10,000 possible solutions / hypothesis)

Map of the environment

Robot's location

Robot's path



PARTICLE FILTERS

- Generating samples from a known distribution
  - Given a probability density function, draw samples with the appropriate probability
  - Easy for uniform, Gaussian
    - Use built in Matlab functions
  - Harder for arbitrary pdfs (probability distribution functions), but approximation is possible
  - Needed in Particle filters to perform measurement update

PARTICLE FILTERS

- Generating samples from a distribution (*Steps to follow*):
  - Given a state $x \in \mathbb{R}$ and a distribution $g(x): \mathbb{R} \to [0,1]$

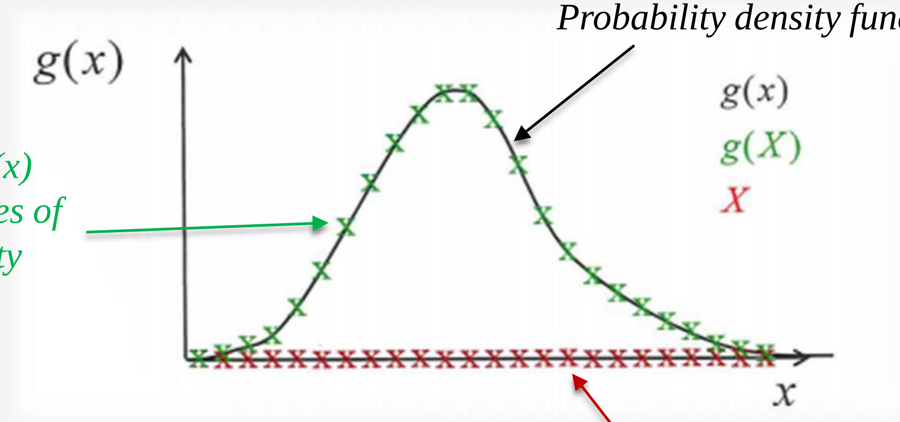
1. Create a vector $\boldsymbol{X}$ of evenly spaced values of x over the range of interest
   - e.g., If $g(x)$ is Gaussian create $\boldsymbol{X}$ to span $\pm 5\sigma$ about mean
2. Create an exact/approximate cumulative distribution vector, $G(\boldsymbol{X})$
   - Integrate probability distribution $g(x)$ to get a cumulative $G(x)$, and create the vector $G(\boldsymbol{X})$
   - Or sum probabilities $g(\boldsymbol{X})$ and normalize to get vector $G(\boldsymbol{X})$
3. Draw samples from a uniform distribution over [0,1]
4. Find closest value to sample in $G(\boldsymbol{X})$
5. Corresponding value of x is a sample, denoted $x_g^{[i]}$

PARTICLE FILTERS

- Sampling of $g(x)$ ($\textit{graphical representation of steps in previous slide}$)
  - Step 1:

Probability density function (Gaussian)

Vector of the function g(x) evaluated over the values of X (creating a probability distribution function)

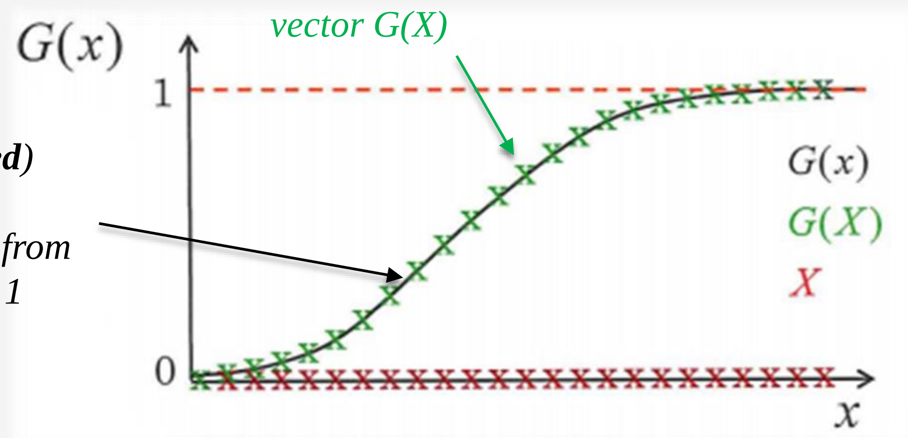


Vector X (evenly spaced)

  - Step 2:

Cumulative distribution vector G(X)

*Cumulative (**normalized**) density function (black curve) obtained from vector of g(x) in Step # 1*



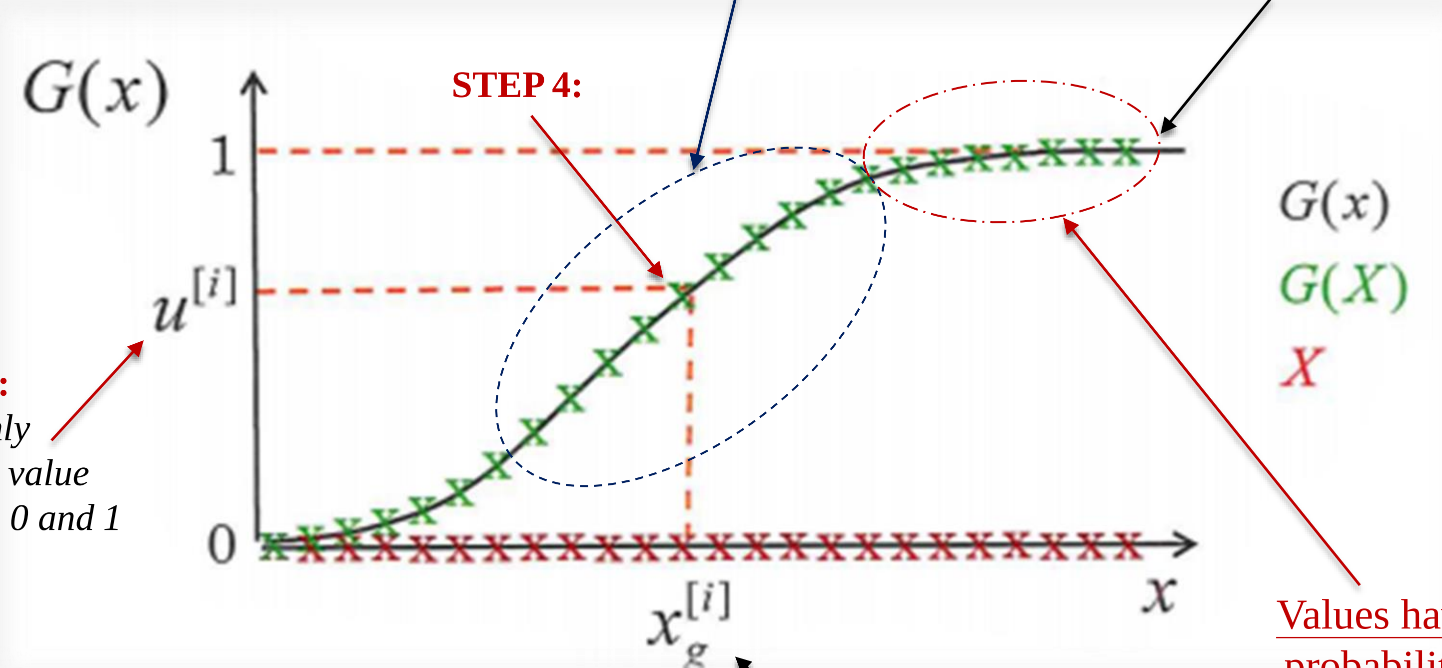
PARTICLE FILTERS

- Sampling of $g(x)$
  - Steps 3 to 5:

Values having **HIGH** probability of being selected

*Cumulative (**normalized**) density function*

STEP 4:



STEP 3:
Randomly selected value between 0 and 1

Values having **LOW** probability of being selected

STEP 5:
*Select the closets sample in the vector **X** that corresponds to the randomly selected value, $u^{[i]}$*

Good sample (particle) that describes the hypothesis

PARTICLE FILTERS

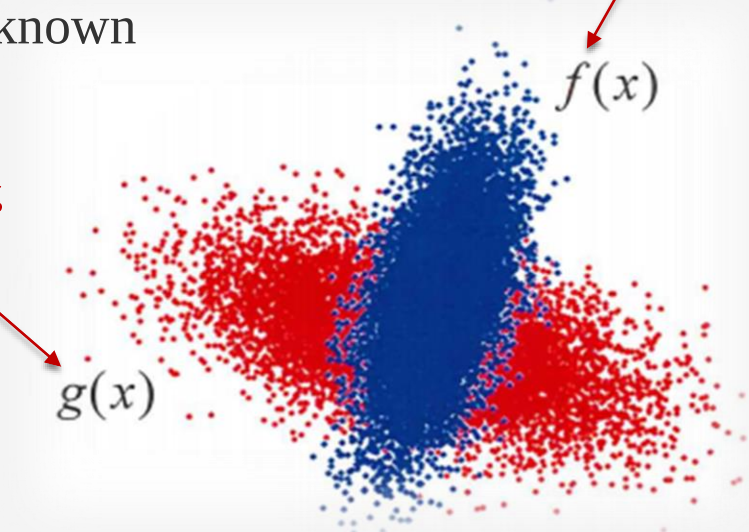
○ Importance Sampling

- However, we do not know $f(x)$, so assuming we have another distribution, $g(x)$, that is directly related to $f(x)$, we can use such function to find $f(x)$.
- **Goal:** perform a calculation using a distribution, $f(x)$, but without being able to sample it directly
  - ○ $f(x) =$ Target distribution, unknown
- Can first sample a different distribution, $g(x)$,
  - ○ $g(x) =$ Proposal distribution, known

Resampling function (*incorporating new information*)

$$f(x) = F\big(g(x)\big)$$

Sampling function



$$f(x) = \frac{f(x)}{g(x)} g(x)$$

PARTICLE FILTERS

- Importance Sampling
  - Then use relationship between distribution if known to define the weighting factor as

$$\boxed{w(x) = \frac{f(x)}{g(x)}}$$

- Finally, resample from $g(x)$, with weights $w(x)$ to generate samples of $f(x)$

- If weighting factor is known, can perform this calculation without knowing $f(x)$
  - Note that $g(x) > 0$ wherever $f(x) > 0$ for this to be valid

PARTICLE FILTERS

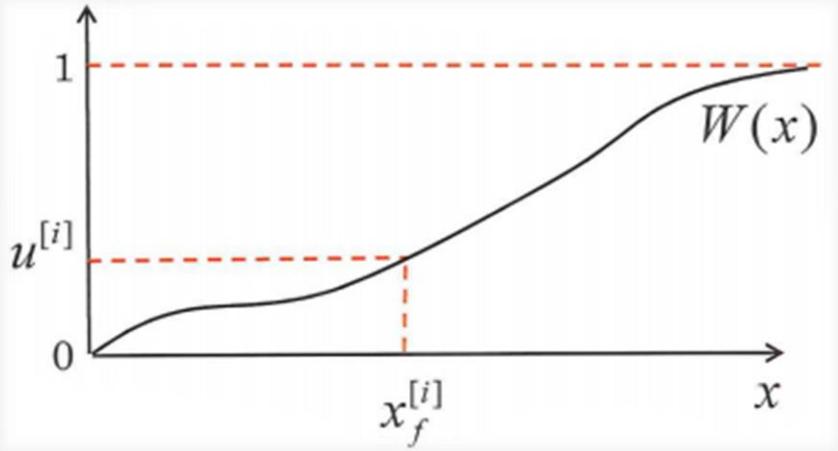
- Importance Sampling
  - Define weights for each sample $x_g^{[i]}$ in S

$$w(x) = \frac{f(x)}{g(x)}$$

    - Weights are the probability that we should include sample $x_g^{[i]}$ in our final sample set
      - The importance of sample $x_g^{[i]}$
    - Not obvious how to calculate the weight at this point, will become clear in derivation of particle filter
    - For now, found by dividing $f(x_g^{[\mathrm{i}]})$ by $g(x_g^{[i]})$
      - This assumes complete knowledge of $f(x)$,
        - (yes, cheating)

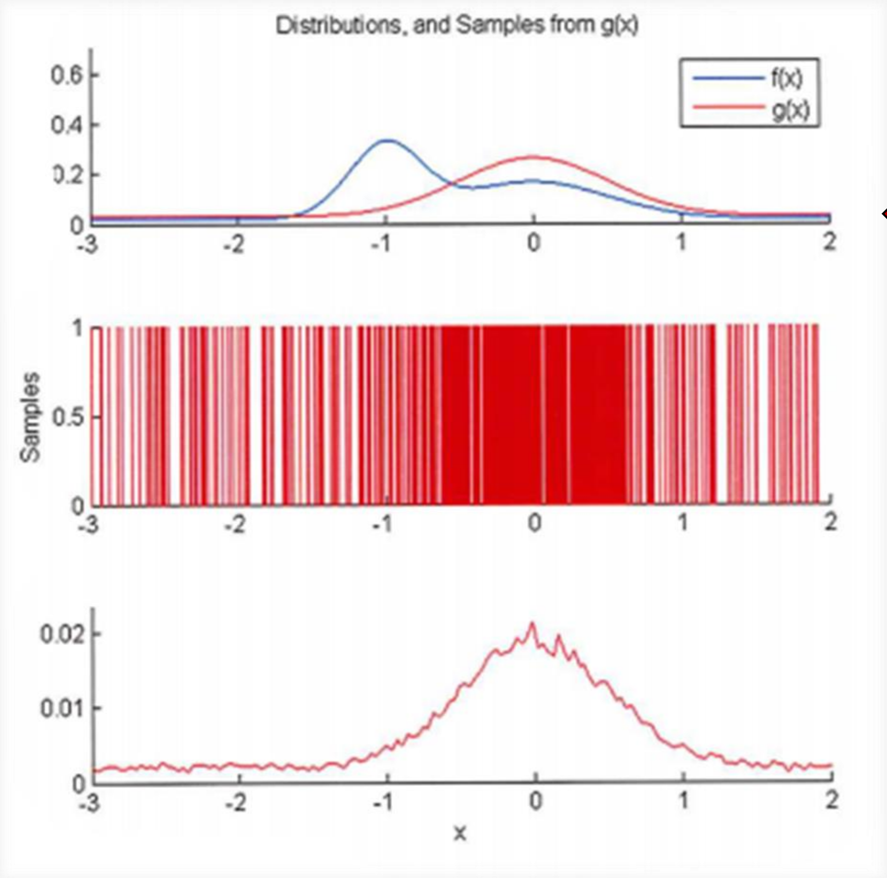
PARTICLE FILTERS

- Importance Sampling
  - Importance sampling of $f(x)$
    1. Define cumulative distribution $W(x)$ based on weights $w(x)$ as before (*samples need not be ordered*)
    2. For each sample:
       a) Take uniform sample, $u^{[i]}$
       b) Find first element of $W(x)$ that exceeds current sample
       c) Add corresponding value of $x_g^{[i]}$ as a sample-to-sample set, S'



PARTICLE FILTERS

- Example (*Theoretical example: we know f (x) and g(x)*)
  - Target distribution $f(x)$, proposal distribution $g(x)$
  - 20000 samples drawn from $g(x)$ (1/25th of samples shown)

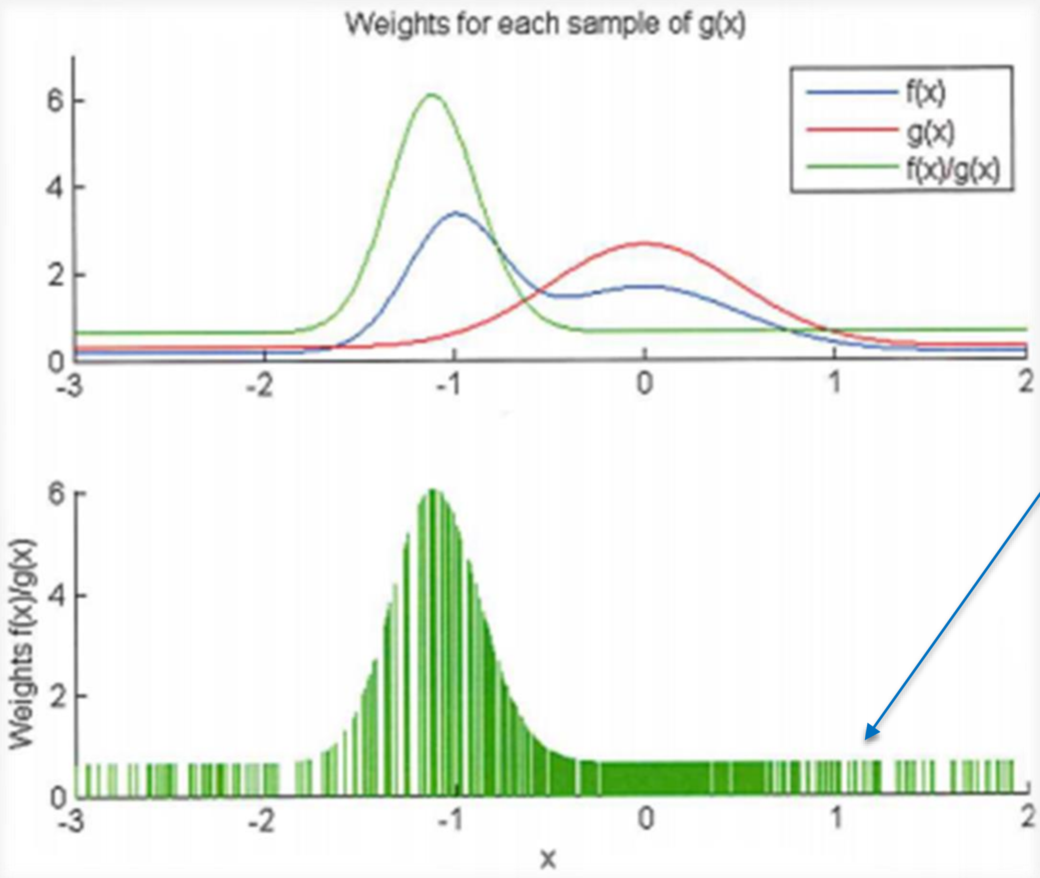


We satisfy $g(x) > 0$ wherever $f(x) > 0$

We want to reconstruct $f(x)$ from $g(x)$

PARTICLE FILTERS

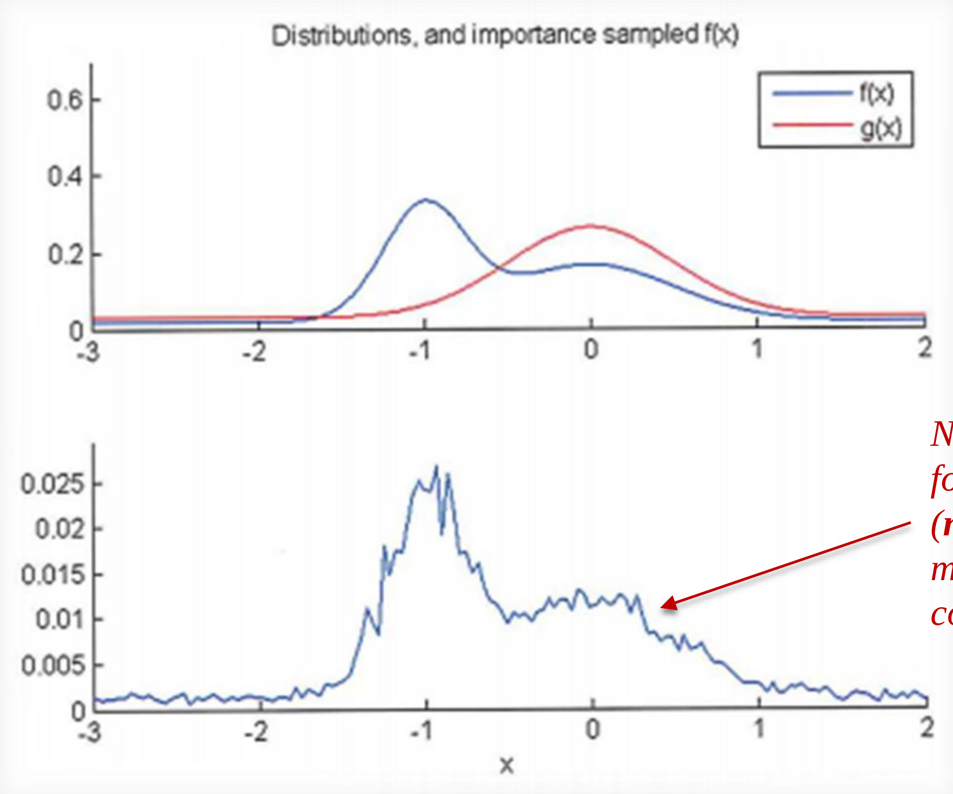
- Example
  - We now calculate the weights for all *x* and sample weights for each sample



$$w(x) = \frac{f(x)}{g(x)}$$

PARTICLE FILTERS

- Example
  - Importance sample points to generate new set
  - New set is distributed according to $f(x)$



*New distribution obtained by forming the cumulative (**normalized**) density function and making our selection from there to construct f(x)*

PARTICLE FILTERS

- The Particle Set
  - A sample can be drawn from a *proposal distribution*

$$x_p^{[i]}$$

  - The sample is assigned a weight

$$w_p^{[i]}$$

  - The combination of sample and weight is a particle

$$s^{[i]} = \left\{x_p^{[i]}, w_p^{[i]}\right\}$$

  - The particle set is used to generate an approximation to the *target distribution*

$$s^{[i]} = \left\{x^{[i]}, w^{[i]}\right\} \qquad S = \{s^{[1]}, \dots, s^{[I]}\}$$

  - I is the total number of particles in the set
  - The approximation improves as $I \rightarrow \infty$

PARTICLE FILTERS

- Defining the usual model elements, in general probabilistic form (*Derivation*)
  - State prior:
    $$p(x_0)$$
  - Motion Model:
    $$p(x_t|x_{t-1}, u_t)$$
  - Measurement Model:
    $$p(y_t|x_t)$$
  - Only restriction on model elements are that samples can be drawn from them, (*probabilities known for all conditional values*)

PARTICLE FILTERS

- Beliefs
  - In particle filters, the belief distributions will be represented by particle sets
    - The belief

$$x_t^{[i]} \sim bel(x_t) = p(x_t | y_{1:t}, u_{1:t})$$

$$S_t = \{s_t^{[1]}, \ldots, s_t^{[I]}\}$$

    - The predicted belief

$$\bar{x}_t^{[i]} \sim \overline{bel}(x_t) = p(x_t | y_{1:t-1}, u_{1:t})$$

$$\bar{S}_t = \{\bar{s}_t^{[1]}, \ldots, \bar{s}_t^{[I]}\}$$

PARTICLE FILTERS

- Particle Filter Algorithm:
  1. Prediction Transformation
     - Transform prior belief particle set to predicted belief through sampling
  2. Importance factor
     - Using measurement, calculate particle importance factor
       - *Probability of the measurement occurring, given the state was defined by the current particle*
  3. Resampling
     - Transform predicted belief particle set to belief using importance sampling

Note: Steps 1, 2 can be combined into a single loop, if prediction and measurement steps are combined.

PARTICLE FILTERS

- Particle Filter Components

  1. Prediction Update
     - The samples $x_{t-1}^{[i]}$ are known from previous iteration
     - The motion model and input are known
     - It is therefore possible to generate samples of $\overline{bel}(x_t)$

$$\bar{x}_t^{[i]} \sim p\left(x_t \middle| x_{t-1}^{[i]}, u_t\right)$$

     - One new sample is drawn from each distribution defined by the prior samples
     - The set of I new samples defines an approximation to $\overline{bel}(x_t)$
     - Unit weighting on each particle

$$\bar{S}_t = \{\bar{s}_t^{[1]}, \dots, \bar{s}_t^{[I]}\}$$

PARTICLE FILTERS

- Particle Filter Components

  2. Measurement update

     - The measurement is known but the state is not
     - Would like to generate a particle set to represent $bel(x_t)$
       - Target distribution
     - Have particle set representation of prediction belief $\overline{bel}(x_t)$
       - Proposal distribution
     - Use importance sampling to generate belief update

measurement model

$$w_i^{[i]} = \eta \frac{bel(x_t)}{\overline{bel}(x_t)}$$

prior belief

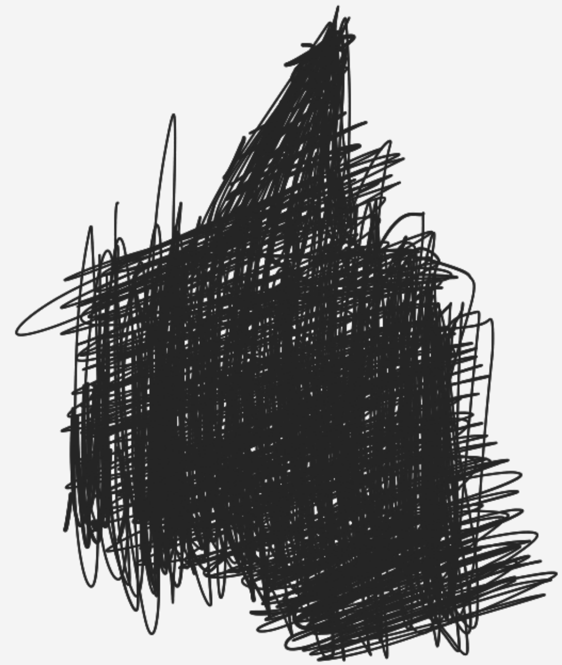
     - The proper weighting to use turns out to be

$$w_t^{[i]} = p(y_t | \bar{x}_t^{[i]})$$

PARTICLE FILTERS

- Particle Filter Expanded Algorithm
  1. Prediction update:
     a) Sample $\bar{x}_t^{[i]} \sim p(x_t | x_{t-1}^{[i]}, u_t)$
     b) Weight $\bar{w}_t^{[i]} = 1$
     c) Add to $\bar{S}_t$

Prior belief



  2. Measurement update:
     1. For each particle in $\bar{S}_t$
        a) Calculate weighting

$$w_t^{[i]} = p(y_t | \bar{x}_t^{[i]})$$

     2. For $j = 1$ to I

Belief

        a) Draw particle $\bar{x}_t^{[i]}$ with probability $\propto w_t^{[i]}$
        b) Add to S_t as $s_t^{[i]} = \left\{x_t^{[i]}, 1\right\}$

PARTICLE FILTERS

- Particle Filter Expanded Algorithm (*simplified*)

1. For each particle in S_{t-1}
   a) Propagate sample forward using motion model (*sampling*)

   $$\bar{x}_t^{[i]} \sim p(x_t | x_{t-1}^{[i]}, u_t)$$

   b) Calculate weight (*importance*)

   $$w_t^{[i]} = p(y_t | \bar{x}_t^{[i]})$$

   b) Store in interim particle set

   $$S_t' = S_t' + \left\{ s_t^{[i]} \right\}$$

2. Normalize weights
3. For j = 1 to I
   a) Draw index i with probability $\propto w_t^{[i]}$ (*resampling*)
      - Add to final particle set

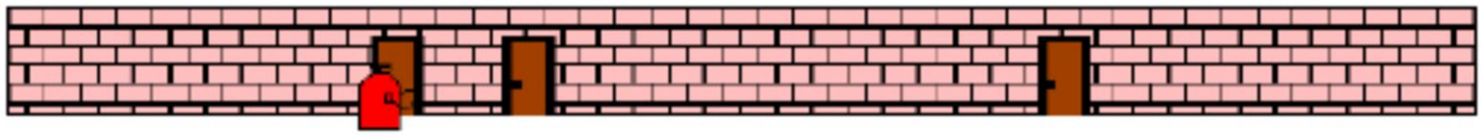
        $$S_t = S_t + \left\{ s_t^{[i]} \right\}$$

PARTICLE FILTERS

- Example:
  - Robot Localization
    - Robot travels along a hallway, can detect doors within a range with a noisy sensor
    - Knows probability of detecting a door, given a specific location

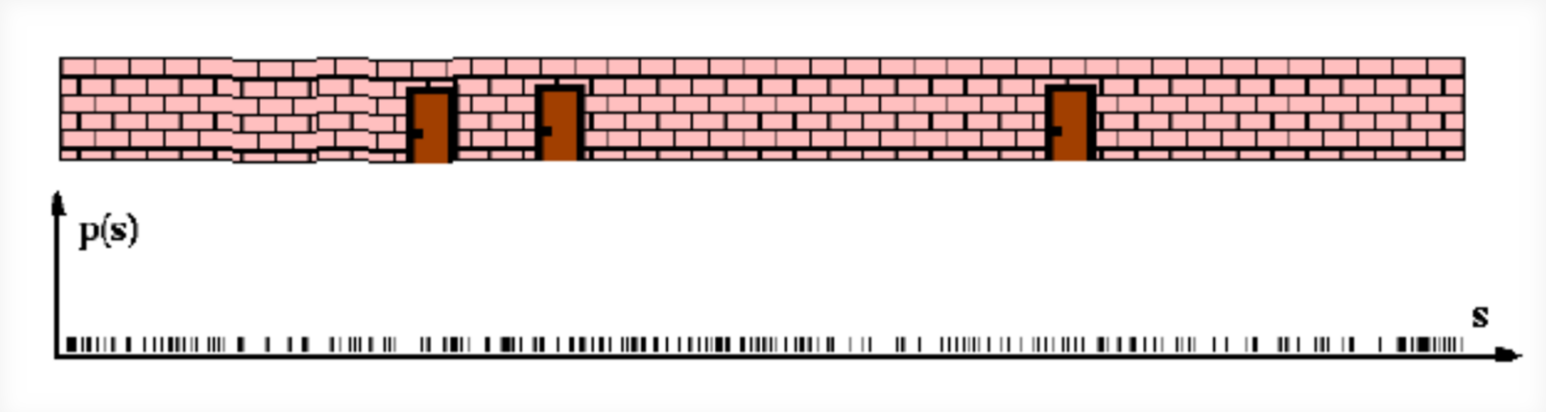
$$p(y_t|x_t)$$

    - Knows motion model, and has uniform initial belief (*has no idea where it is*)



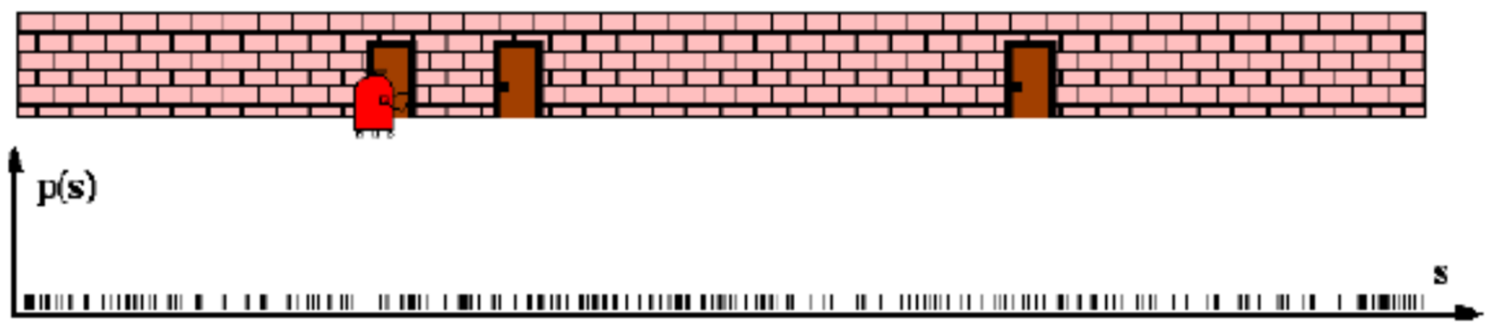
PARTICLE FILTERS

- Example
  - Step One:
    - Sample uniform over state space



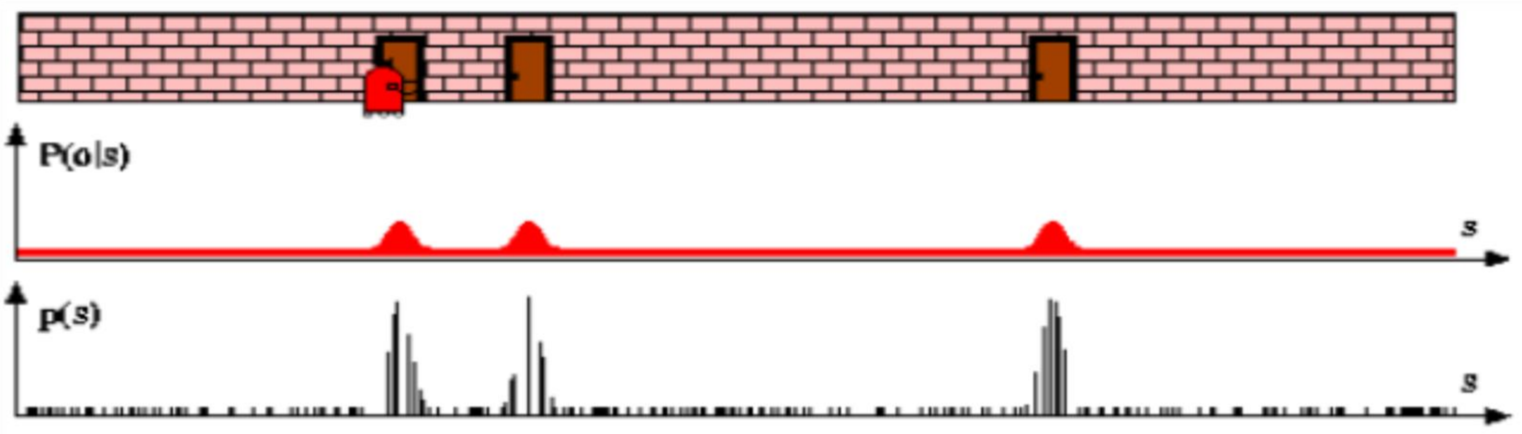
PARTICLE FILTERS

- Example
  - Step Two:
    - Propagate samples through motion model (*i.e., provide an input, u_t, and add disturbances*)



PARTICLE FILTERS

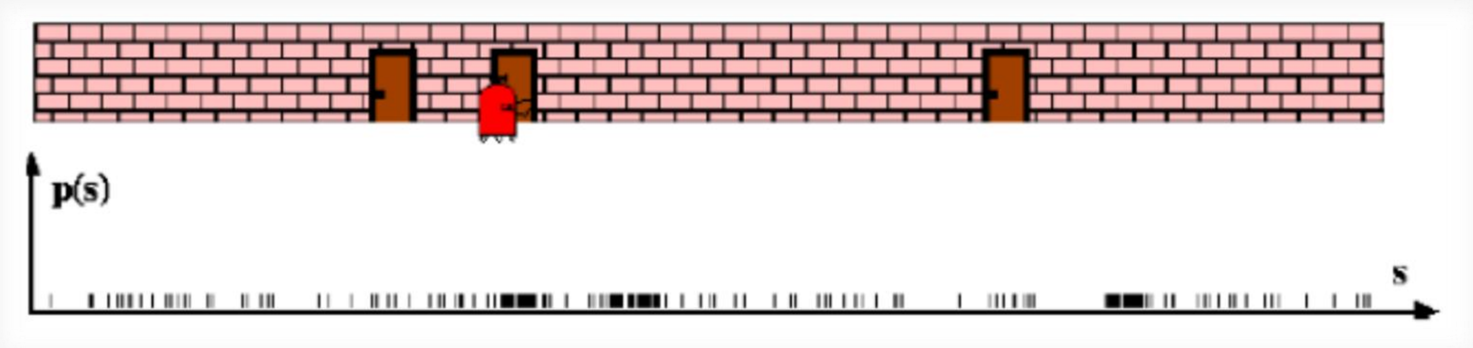
- Example
  - Step Three:
    - Take a measurement, and use $p(y_t|x_t)$ to calculate weights



- Particles that are more likely have higher weights
  - Starting to narrow down options
  - Still difficult to estimate state (mean?)

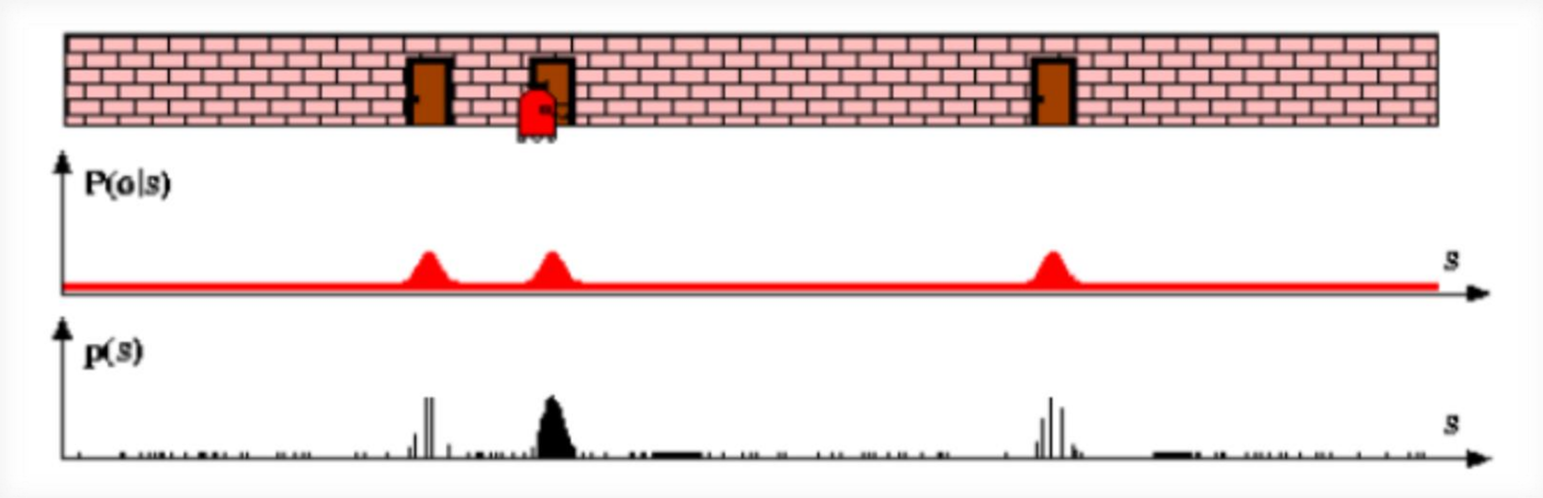
PARTICLE FILTERS

- Example
  - Step Four:
    - Perform resampling to get more particles in areas of higher probability
    - Reset weights to 1, as particle locations capture probability information
  - Repeat
  - The following particle set shows how the motion model distributes the identical particles that result from resampling



PARTICLE FILTERS

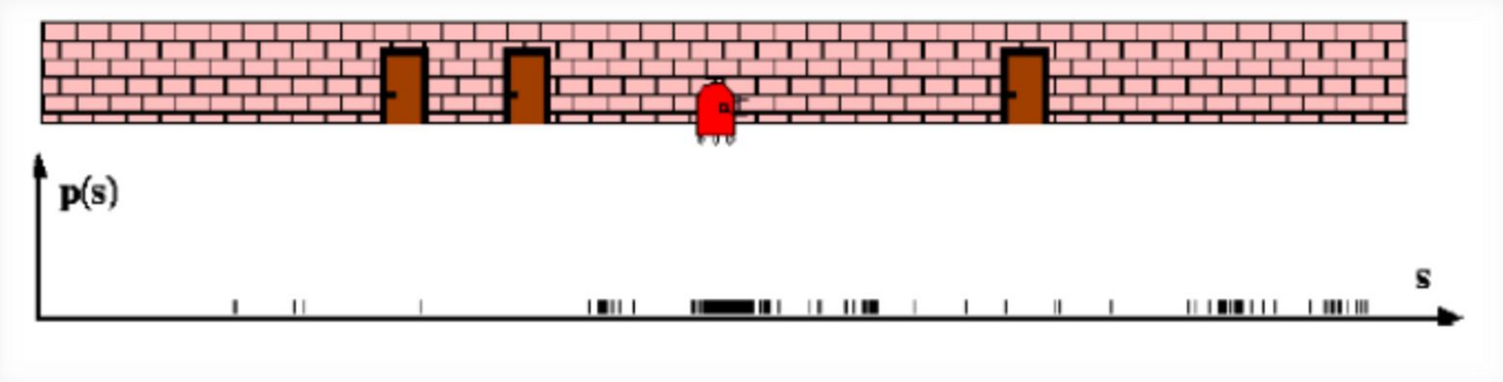
- Example
  - After a second measurement, weights are again assigned to the particles



  - True state starts to become apparent

PARTICLE FILTERS

- Example
  - After resampling again, propagate with motion model sampling



PARTICLE FILTERS

- Derivation (*why the measurement model is the weight assignment*):
  - Consider the particles as state sequence samples

$$x_{(0:t)}^{[i]} = x_0^{[i]}, x_1^{[i]}, \dots, x_t^{[i]}$$

  - Form belief over entire sequence

$$bel(x_{0:t}) = p(x_{0:t}|u_{1:t}, y_{1:t})$$

    - Instead of just

$$bel(x_t) = p(x_t|u_{1:t}, y_{1:t})$$

  - This is an enormous state to approximate with a set of particle, but no matter, for derivation only

PARTICLE FILTERS

- Derivation
  - Using Bayes Theorem, expand belief about last measurement

$$bel(x_{0:t}) = p(x_{0:t} | u_{1:t}, y_{1:t})$$

$$= \eta \, p(y_t | x_{0:t}, u_{1:t}, y_{1:t-1}) \, p(x_{0:t} | u_{1:t}, y_{1:t-1})$$

  - The Markov assumption remains valid

$$= \eta \, p(y_t | x_t) \, p(x_{0:t} | u_{1:t}, y_{1:t-1})$$

Normalizer

PARTICLE FILTERS

- Derivation
  - Conditional probability can be used to expand the last distribution

$$bel(x_{0:t})$$
$$= \eta\, p(y_t|x_t\,)p\left(x_t\middle|x_{0:(t-1)}, u_{1:t}, y_{1:t-1}\right) p(x_{0:t-1}|u_{1:t}, y_{1:t-1})$$

  - Apply the Markov assumption again yields

~ current belief at time t-1

$$bel(x_{0:t}) = \eta\, p(y_t|x_t)\, p(x_t|x_{t-1}, u_t\,)\, p(x_{0:t-1}|u_{1:t}, y_{1:t-1})$$

Measurement model

Motion model

PARTICLE FILTERS

- Derivation
  - The sequence $x_{0:t-1}$ does not depend on u_t

$$bel(x_{0:t}) = \eta\, p(y_t|x_t)\, p(x_t|x_{t-1}, u_t)\, p(x_{0:t-1}|u_{1:t-1}, y_{1:t-1})$$

$$= \eta\, p(y_t|x_t)\, p(x_t|x_{t-1}, u_t)\, bel(x_{0:t-1})$$

  - Breaking into two steps
    - Prediction

$$\overline{bel}(x_{0:t}) = p(x_t|x_{t-1}, u_t)\, bel(x_{0:t-1})$$

    - i^{th} particle generated by this distribution is an element of the predicted belief particle set

PARTICLE FILTERS

- Derivation
  - The measurement update uses importance sampling to generate a particle set representation of belief
    - Weighting, based on relation to predicted belief is

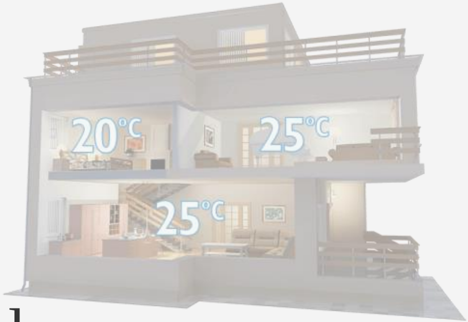
$$w_t^{[i]} = \eta \frac{bel(x_{0:t})}{\overline{bel}(x_{0:t})}$$

$$= \frac{\eta\, p(y_t|x_t)\, p(x_t|x_{t-1}, u_t)\, bel(x_{0:t-1})}{p(x_t|x_{t-1}, u_t)\, bel(x_{0:t-1})}$$

$$= \eta\, p(y_t|x_t)$$

    - Which confirms the use of the measurement model as the weighting parameter
    - This confirms that particles sets are distributed according to full belief sequences, which means must hold for current state too

PARTICLE FILTERS



- Example (*Single variable*)
  - Returning to the temperature control problem
    - State is current temperature
    - One dimensional example
    - Prior: Uniform over temperature range

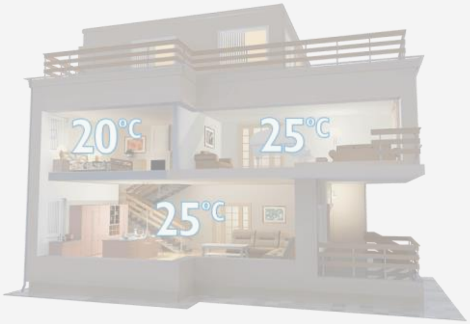
    - Motion Model: Decaying temperature +furnace input + disturbances (opening doors, outside effects)

$$x_t = 0.8x_{t-1} + 3u_t + r_t$$

$$A = 0.8, \quad \mathrm{B} = 3$$

$$r_t \sim N(0,2)$$

PARTICLE FILTERS



- Example
  - Measurement Model:
    - Directly measure the current temperature

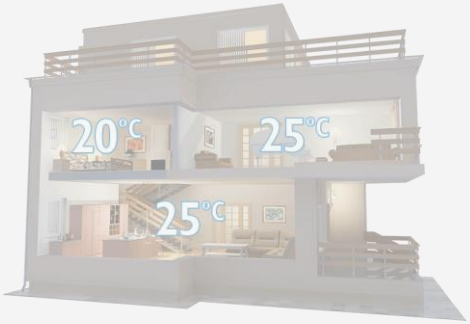
$$y(t) = x(t) + \delta_t$$

$$\delta_t \sim N(0,4)$$

  - Controller design:
    - Bang bang control, based on current estimate of temperature

$$u(t) = \begin{cases} 1 & \mu_t < 2 \\ 0 & \mu_t > 10 \\ u(t-1) & \text{otherwise} \end{cases}$$

PARTICLE FILTERS



- Particle filter calculations
  1. Transform and Sample from Gaussian

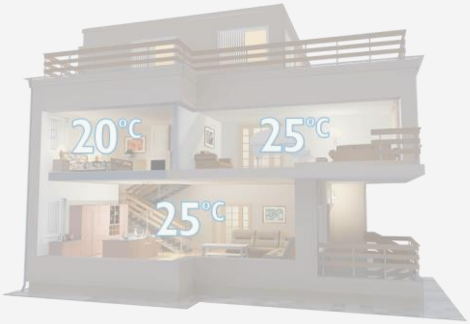
$$\bar{x}_t^{[i]} \sim p\left(x_t \middle| x_{t-1}^{[i]}, u_t\right) = 0.8x_{t-1}^{[i]} + 3u_t + r_t$$

  2. Define weights from measurement model, with Gaussian noise centered at predicted measurement location

$$w_t^{[i]} = p(y_t | \bar{x}_t^{[i]})$$

  3. Resample from predicted belief particle set using weights to generate belief particle set

PARTICLE FILTERS



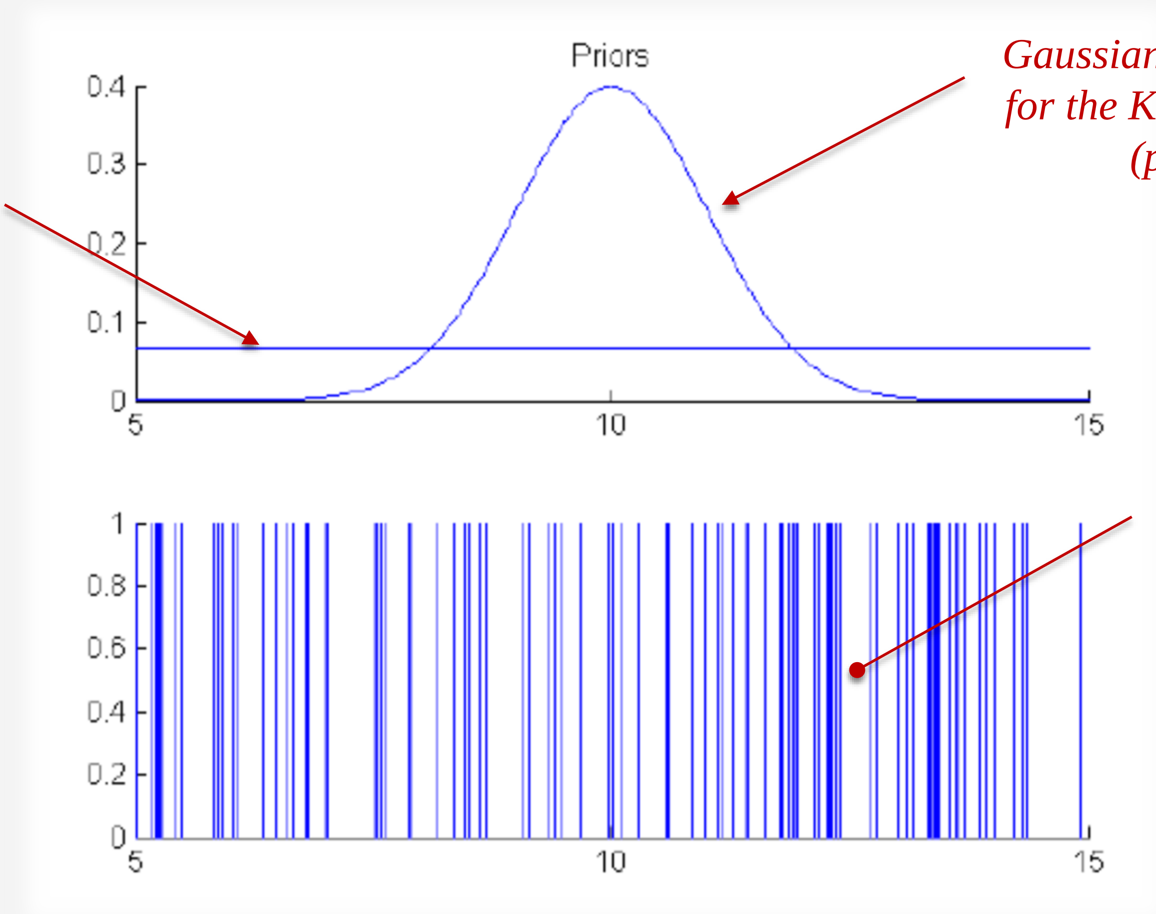
- Resulting particle filter Matlab code

```
%Particle filter estimation
 for i=1:I
      e = sqrt(R)*randn(1);
      Xp(i) = A*X(i) + B*u(t) + e;
      w(i) = normpdf(y(t),C*Xp(i),Q);
 end
 W = cumsum(w);
 W = W/max(W);
 for j=1:I
      i = find(W>rand(1),1);
      X(j) = Xp(i);
 end
```

PARTICLE FILTERS

- Priors, comparing KF and particle filter:

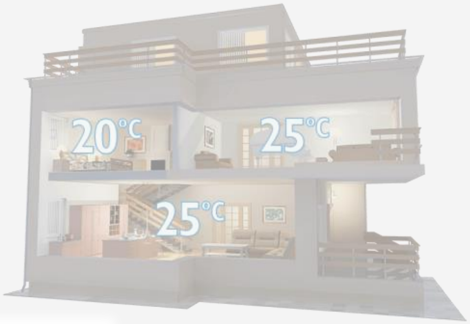
Uniform distribution (prior)



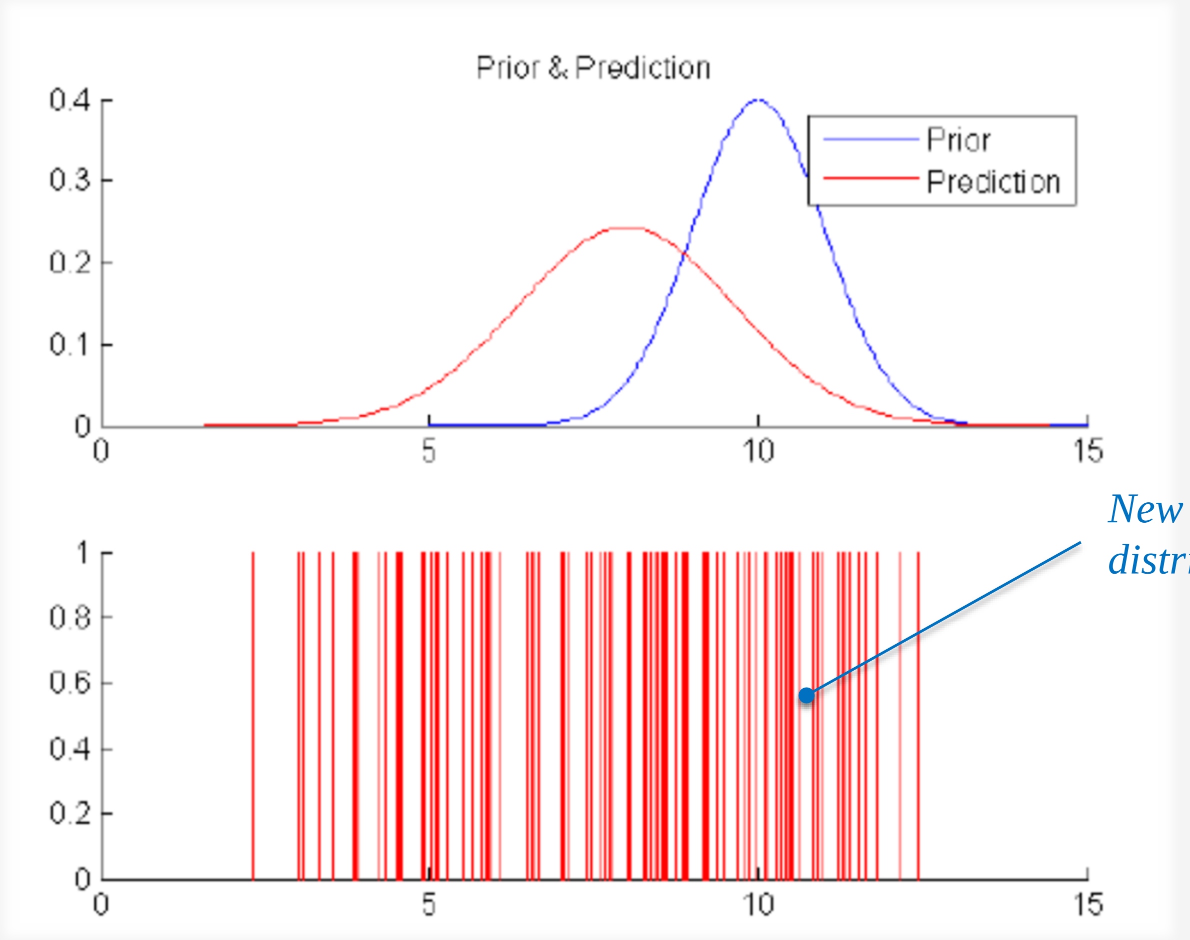
Gaussian distribution for the Kalman Filter (prior)

Sample set for the first stage

PARTICLE FILTERS

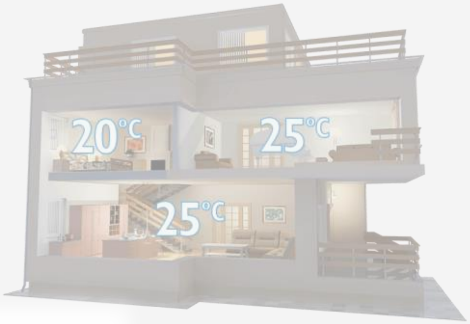


- Prediction:

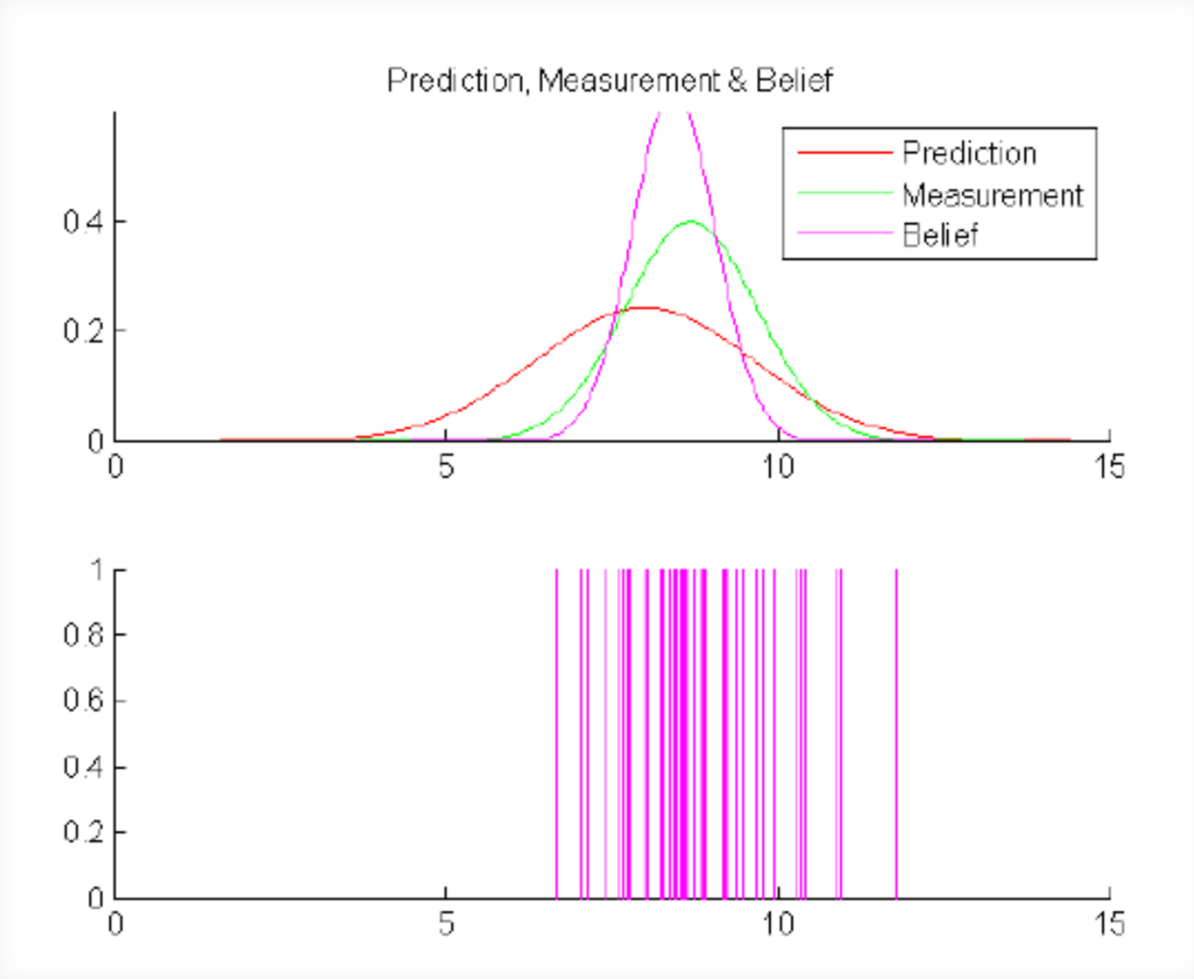


New formed distribution

PARTICLE FILTERS



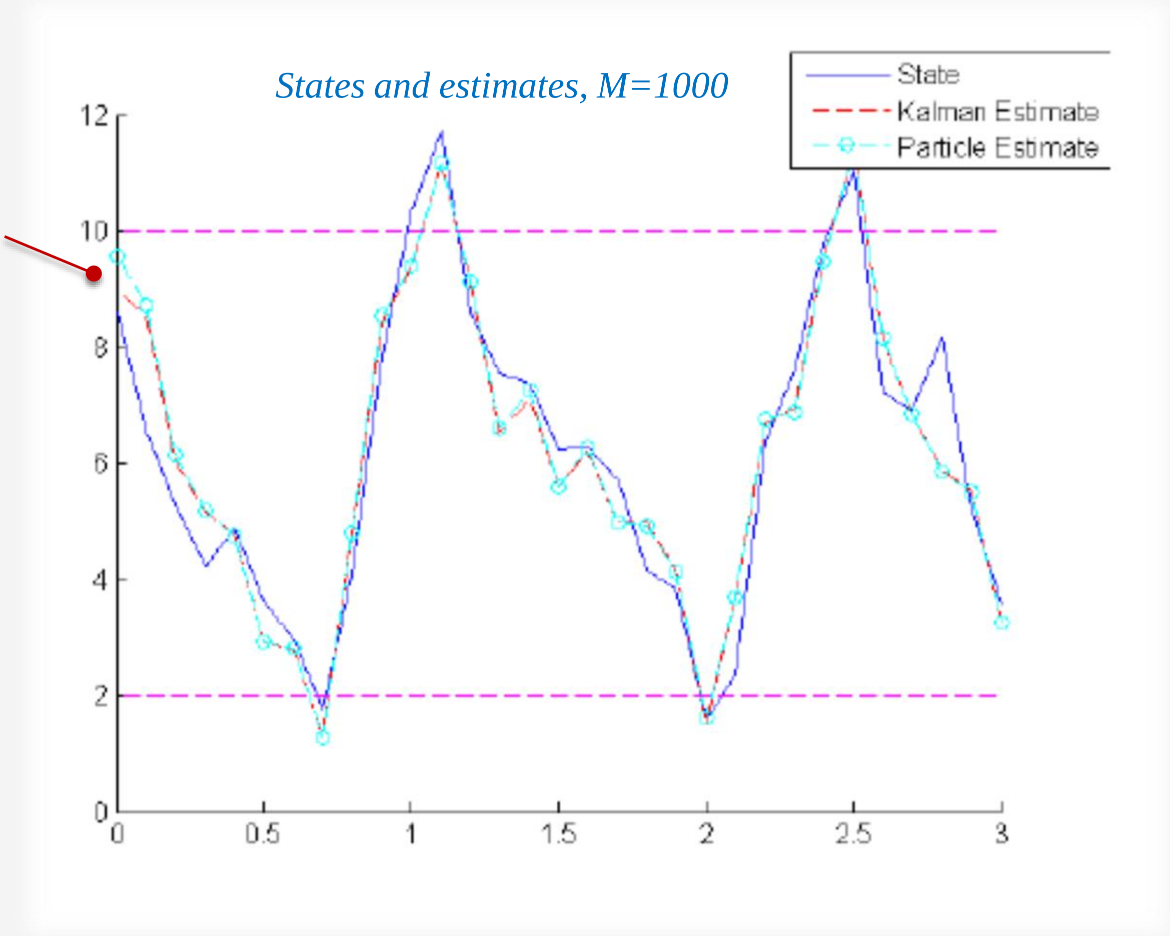
- Belief:



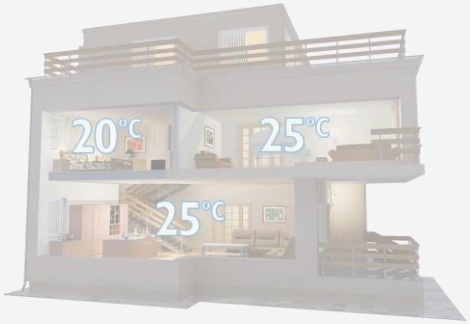
PARTICLE FILTERS

- Comparison of Gaussian parameters
  - KF vs PF (1000, 100, 10)

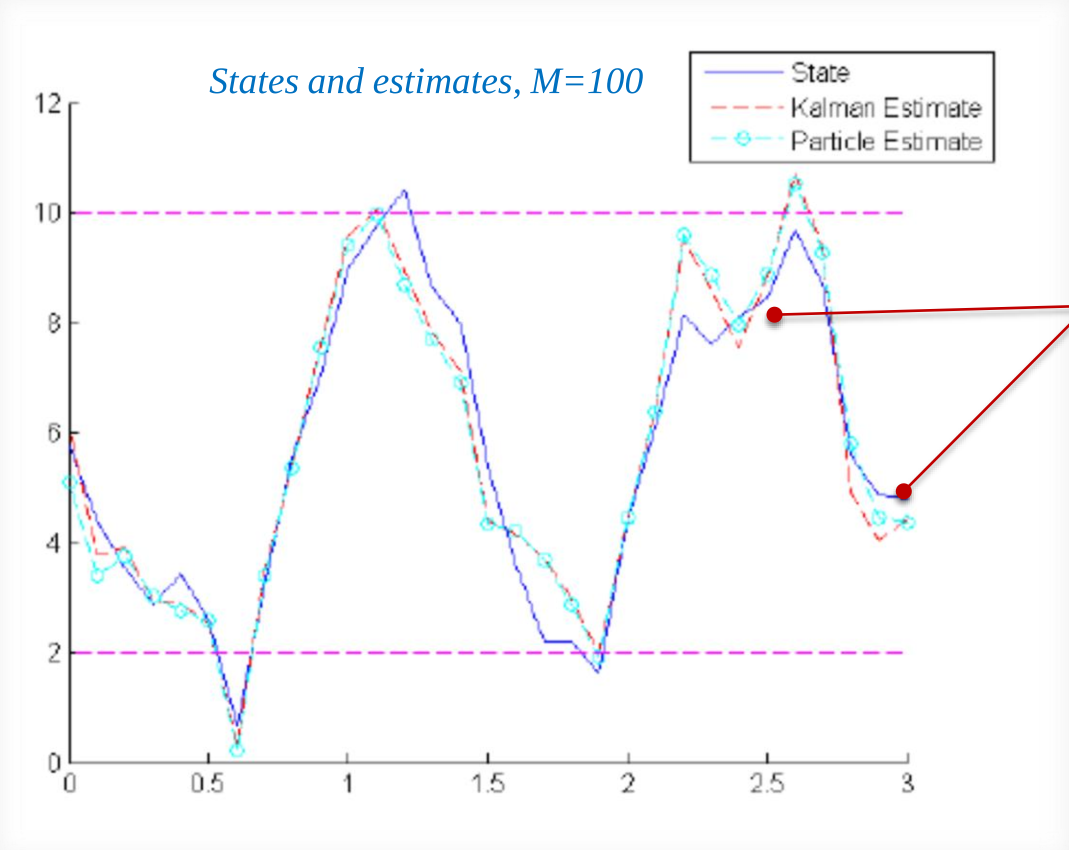
Some minor deviations based on the initial belief



PARTICLE FILTERS

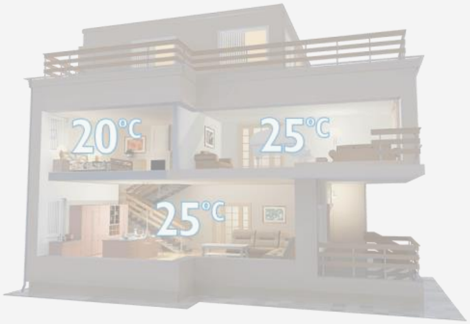


- Comparison of Gaussian parameters
  - KF vs PF (1000, 100, 10)

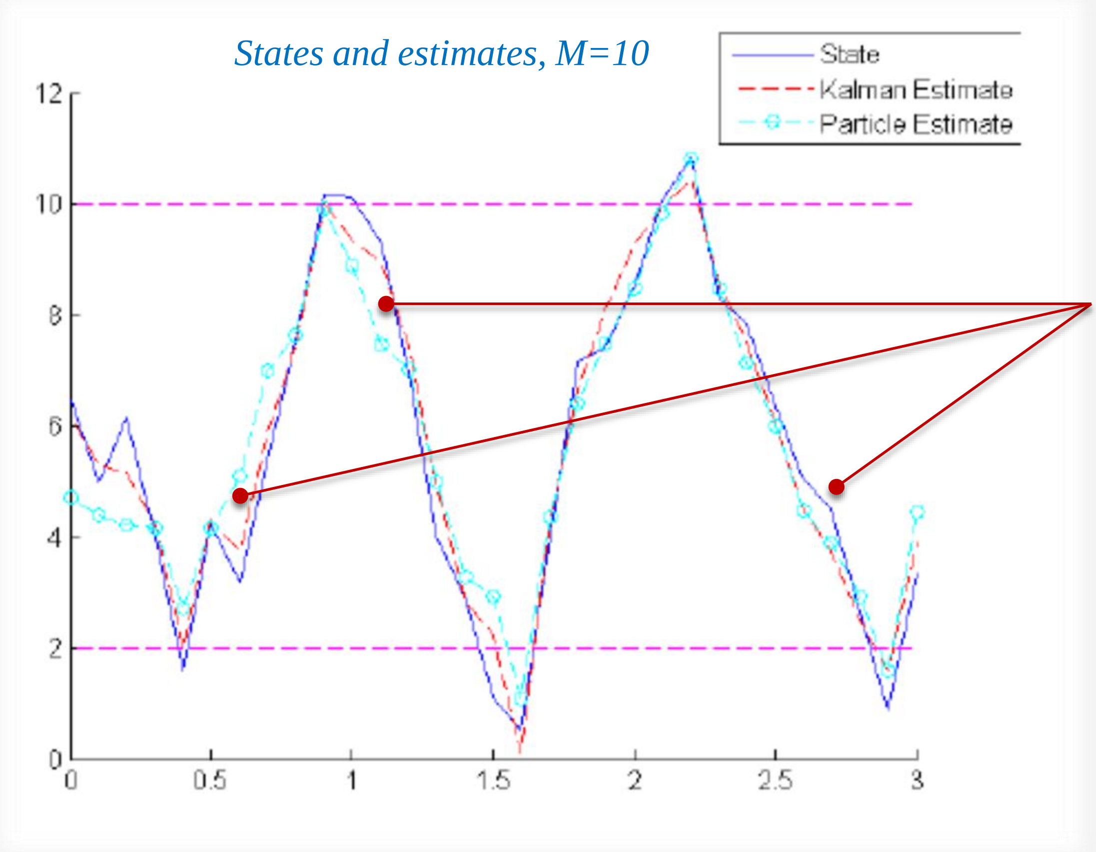


More variation relive to the KF

PARTICLE FILTERS



- Comparison of Gaussian parameters
  - KF vs PF (1000, 100, 10)



Significant deviation w.r.t. the KF

PARTICLE FILTERS

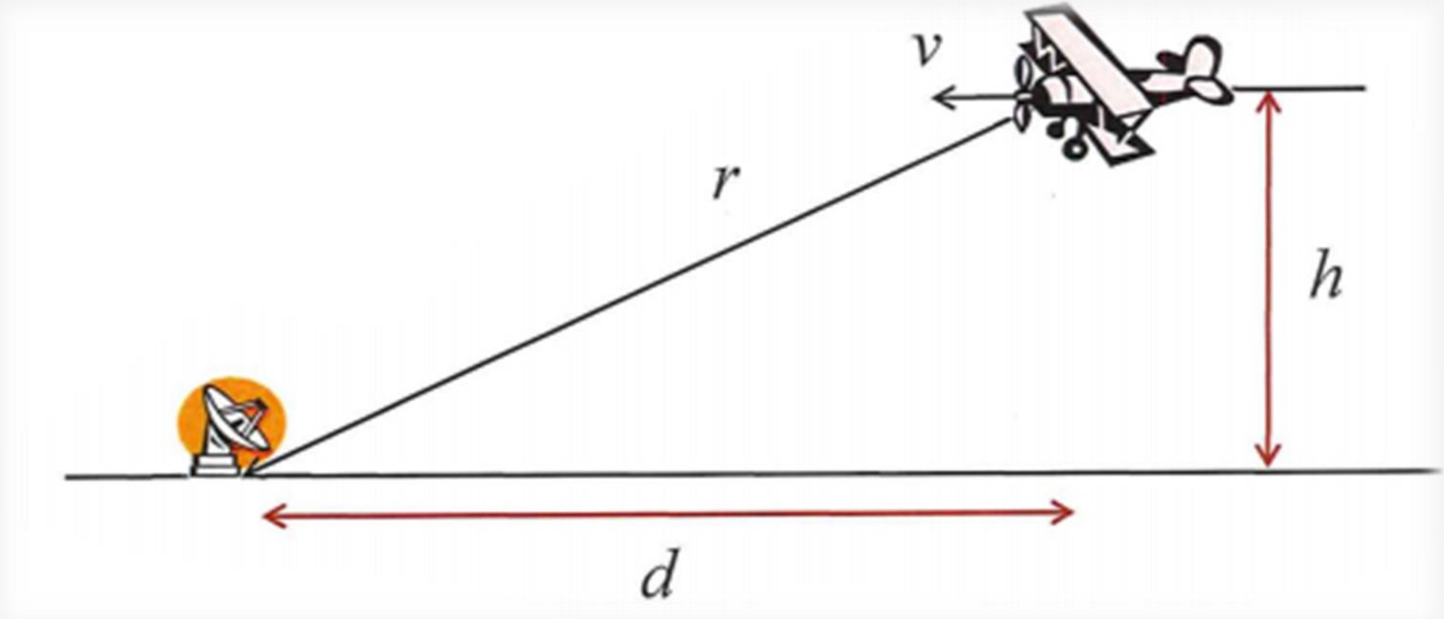
- Comparison of run times (in Matlab):

Algorithm	**Run (*computation*) Time**
Kalman Filter	0.005164
Particle Filter – *10 particles*	0.043191
Particle Filter – *20 particles*	0.06965
Particle Filter – *100 particles*	0.2188
Particle Filter – *1000 particles*	1.8740

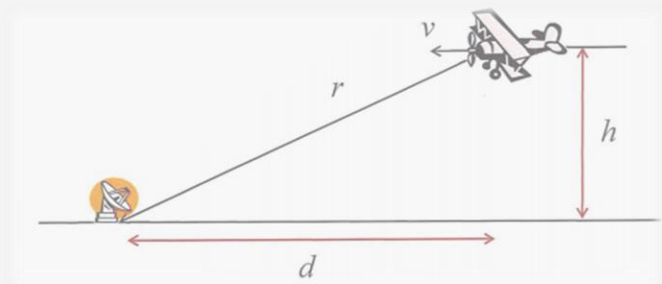
Particle filters: *Due to high computational cost we only use them when we need them (**in very complex problems**).*

PARTICLE FILTERS

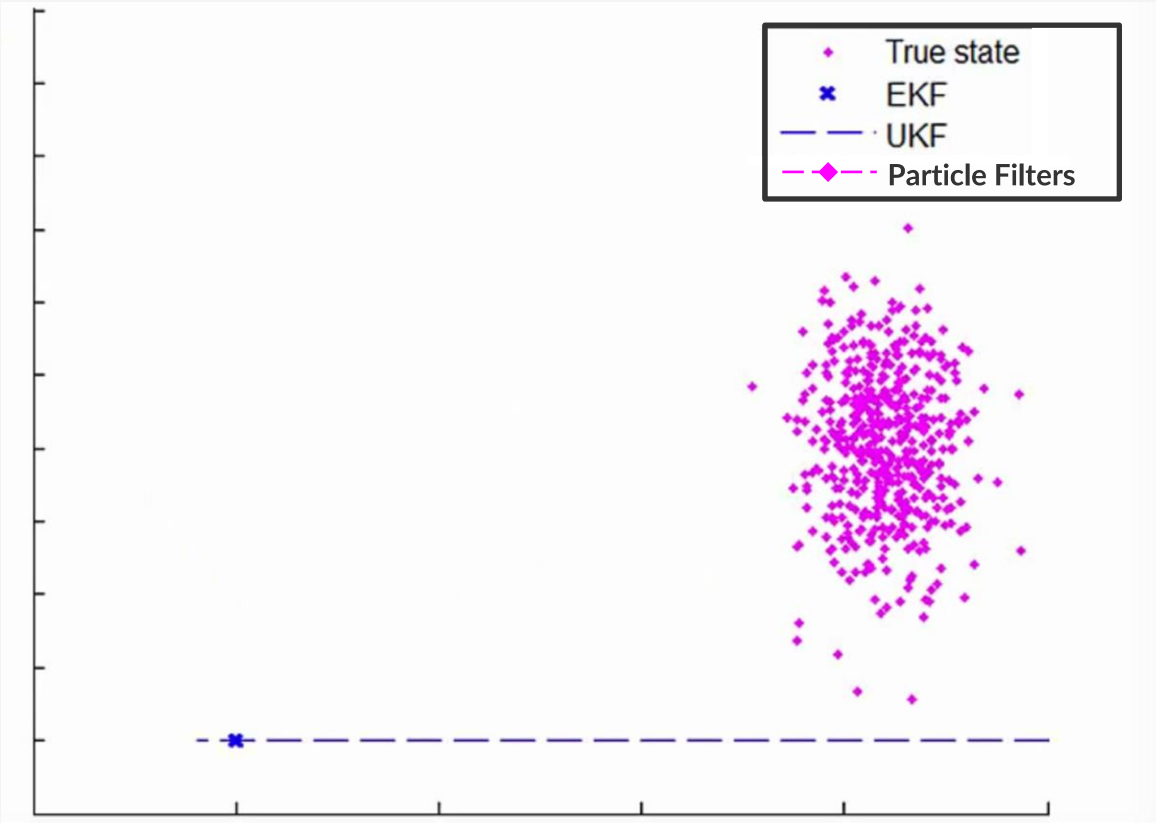
- EKF/UKF/Particle showdown
  - Aircraft flyover example



PARTICLE FILTERS

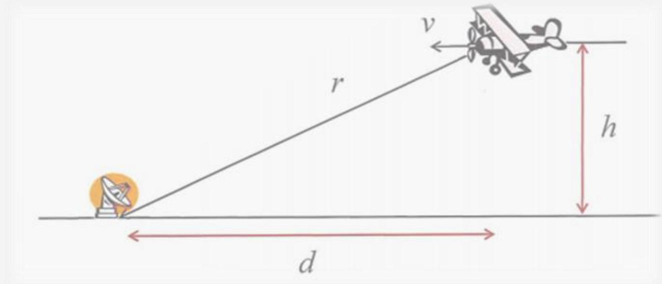


- Results (*Video*)

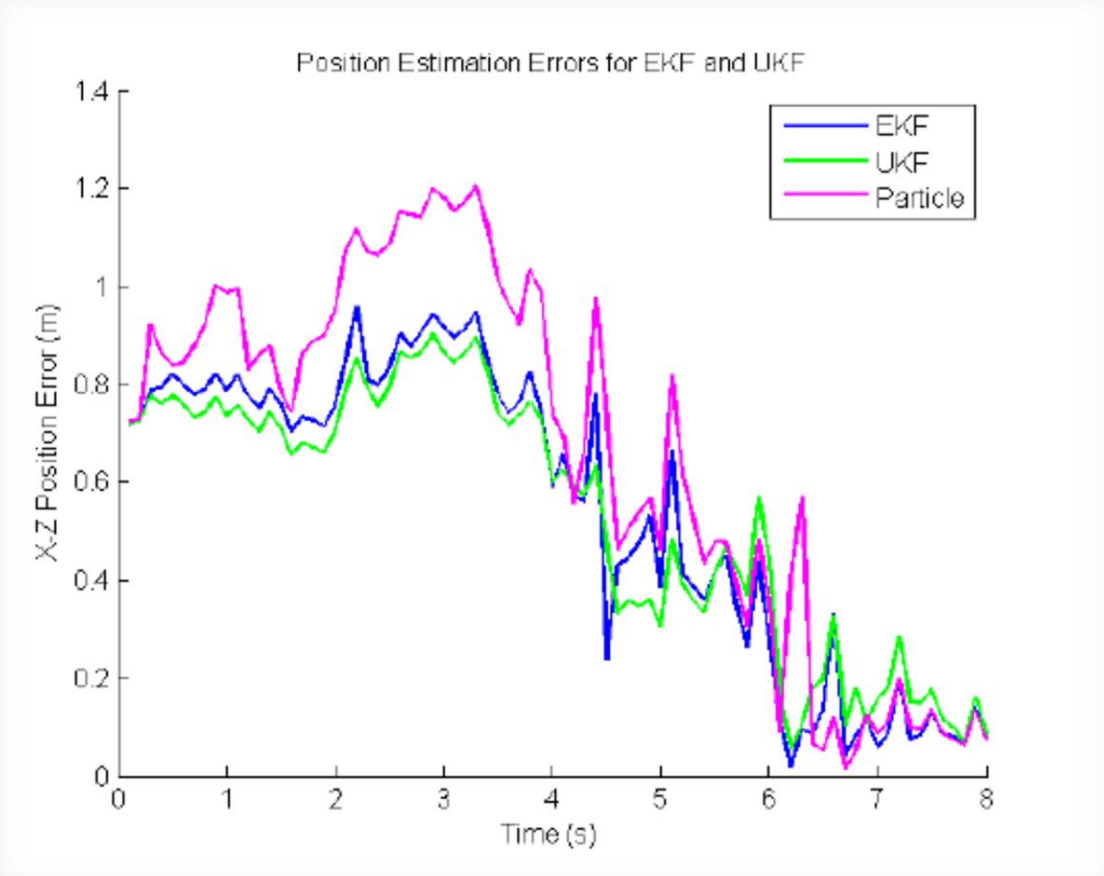


Aircraft flyover results using 500 particles (each representing a possible solution on where the aircraft is)

PARTICLE FILTERS



- Not always better, hard to tune



PARTICLE FILTERS

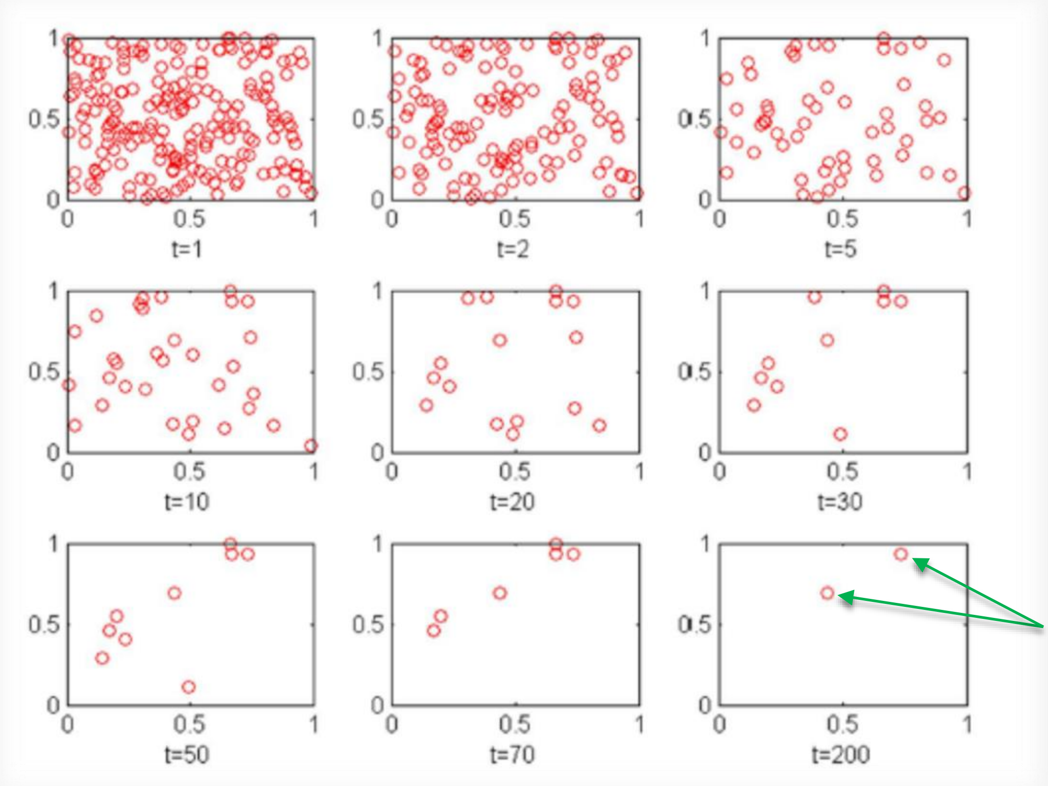
- Interpreting particle sets
  - In order to use a particle filter in robotics, we must somehow extract relevant information from particles that we can use to control robots.
  - **Density Extraction**
    - Gaussian approximation:
      - Simply calculate mean and covariance of set
      - Only really useful for unimodal distributions
      - Used most often for control applications
    - K-means algorithm:
      - Approximate density with a mixture of K Gaussians
      - Requires clustering of particles
    - Kernel density estimation:
      - Use each particle as the center of a continuous kernel function
      - Add all kernels together to generate a pdf
      - Linear in the number of particles

PARTICLE FILTERS

- Sample Variance
  - Since continuous distributions are approximated by a discrete set of samples, errors occur
  - Each time a particle filter is run (with random sampling) a different particle set will result
  - Extreme case:
    - No motion: $x_t = x_{t-1}$
    - No measurements, uniform weights on each particle
    - Uniform prior over 2D space
    - What will happen to the particle set as we update the particle filter?
      - Essentially repeating the resampling step with uniform weight on all particles

PARTICLE FILTERS

- Example
  - Particle deprivation (*the richness of the particle set is important to prevent deprivation*)



Each of these 2 particles include many copies, maintaining the size of the original sample set

PARTICLE FILTERS

- Excessive resampling can lead to particle deprivation
  - Motion sampling adds variety to particle set
  - Do not resample when no motion occurs
    - *Instead update weights multiplicatively for each measurement*
  - If problems arise
    - *Apply low variance sampling*
    - *Artificially disperse samples as well*
    - *Add random samples after resampling*
  - Referred to as variance reduction
    - *Reducing the variance in the particle set approximation*

PARTICLE FILTERS

- **Summary:**
  - Use particle sets instead of parameterizations to represent distributions
  - Inherently an approximation, introduces errors
  - Propagate samples through motion model by sampling from model distribution
  - Weight samples using measurement probability given sample state as true state
  - Define belief distribution through samples and weights (particles) or post resampling
  - Many extensions, nuances, issues, advanced techniques