KFUPM

COE 584 – Robotics

Project Presentation 3

June 2008

Directions in Robot Localization

Ahmad Salam AlRefai, Computer Engineering dept.

Mohammad Shahab, Systems Engineering dept.

Outline

1. Bayes Filtering Review
2. Kalman Filter Review
3. Unscented Kalman Filter
4. Mixed Filter
5. Cooperative Localization
6. Particle Filter Review
7. PF Variations
8. Resampling Techniques
9. KF Simulation
10. PF Simulation

Bayes Filter

- Recall the recursive equation

$$Bel(x_k) = \boxed{\eta P(o_k | x_k)} \int \boxed{P(x_k | x_{k-1}, a_{k-1})} \boxed{Bel(x_{k-1})} dx_{k-1}$$

Observation Model

Motion Model

Previous ‘Belief’

x = pose of robot
o = robot observation (sensor information)
a = robot action (odometry information)

Probability Density (distribution) of the robot state

Kalman Filter

- At every Step

$$\hat{x}_k^- = A\hat{x}_{k-1}^+$$
$$P_k^- = AP_{k-1}^+A^T + Q_{k-1}.$$

Motion Update (Prediction)

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$
$$P_k^+ = (I - K_kH)P_k^-,$$

Measurement Update (Correction)

$$K_k = P_k^- H^T \left(H P_k^- H^T + R_k\right)^{-1}$$ Kalman Gain

$$\left.\begin{matrix} Q_{k-1}. \\ R_k \end{matrix}\right\}$$

Covariance Matrices:
Uncertainties in Actuators & Sensors

$$P_k \}$$

Covariance of States:
Uncertainty at each step

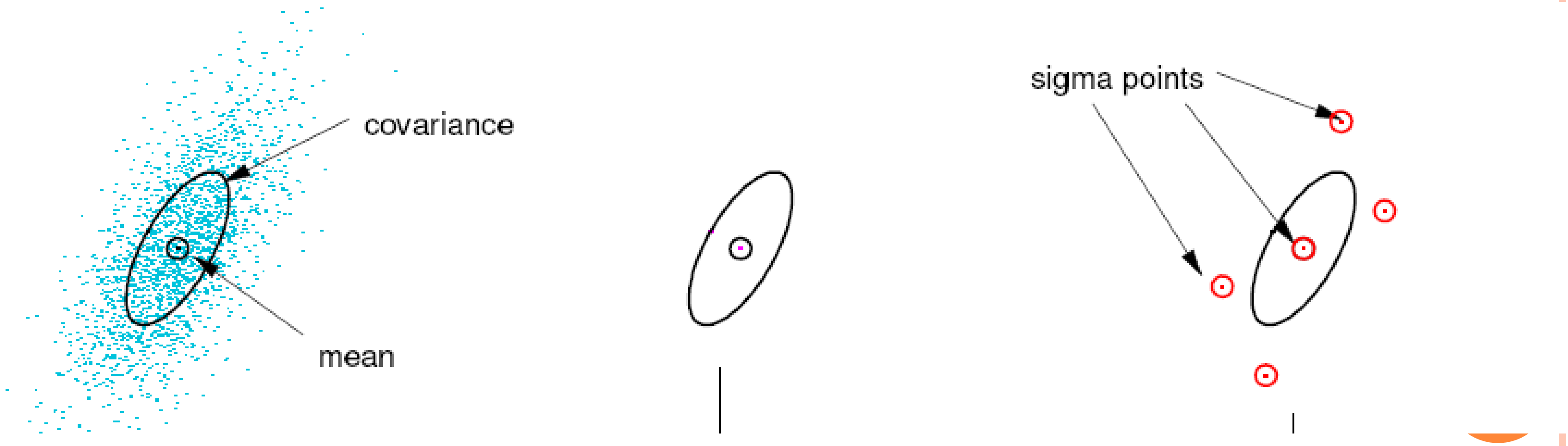
Kalman Filter

- At each step :
  - Computation is only one evaluation of the equations (no particles)
  - Belief is Gaussian (Normal), described *only* by Mean and Covariance

- However,
  - Applied to **linear** models (robot is not)
  - Environment is dynamic and **not gaussian**

UNSCENTED KALMAN FILTER

- We introduce the concept of '**Sigma-Points**'
  - They **approximate** the belief distribution
  - they capture the ***most*** important statistical properties of the prior belief



UNSCENTED KALMAN FILTER

- For 3-state pose, we should choose 2*3+1=7 **sigma-points**
- At each step,

$$\mathcal{X}_0 = \bar{\mathbf{x}}$$ **1 point**

$$\mathcal{X}_i = \bar{\mathbf{x}} + \left(\sqrt{(n_x + \lambda)\mathbf{P}_x}\right)_i$$ **3 points**

$$\mathcal{X}_i = \bar{\mathbf{x}} - \left(\sqrt{(n_x + \lambda)\mathbf{P}_x}\right)_i$$ **3 points**

n_x Number of states

Weights are chosen also (see paper)

P_x **Covariance of States:**
Uncertainty at each step

λ Spread Factor

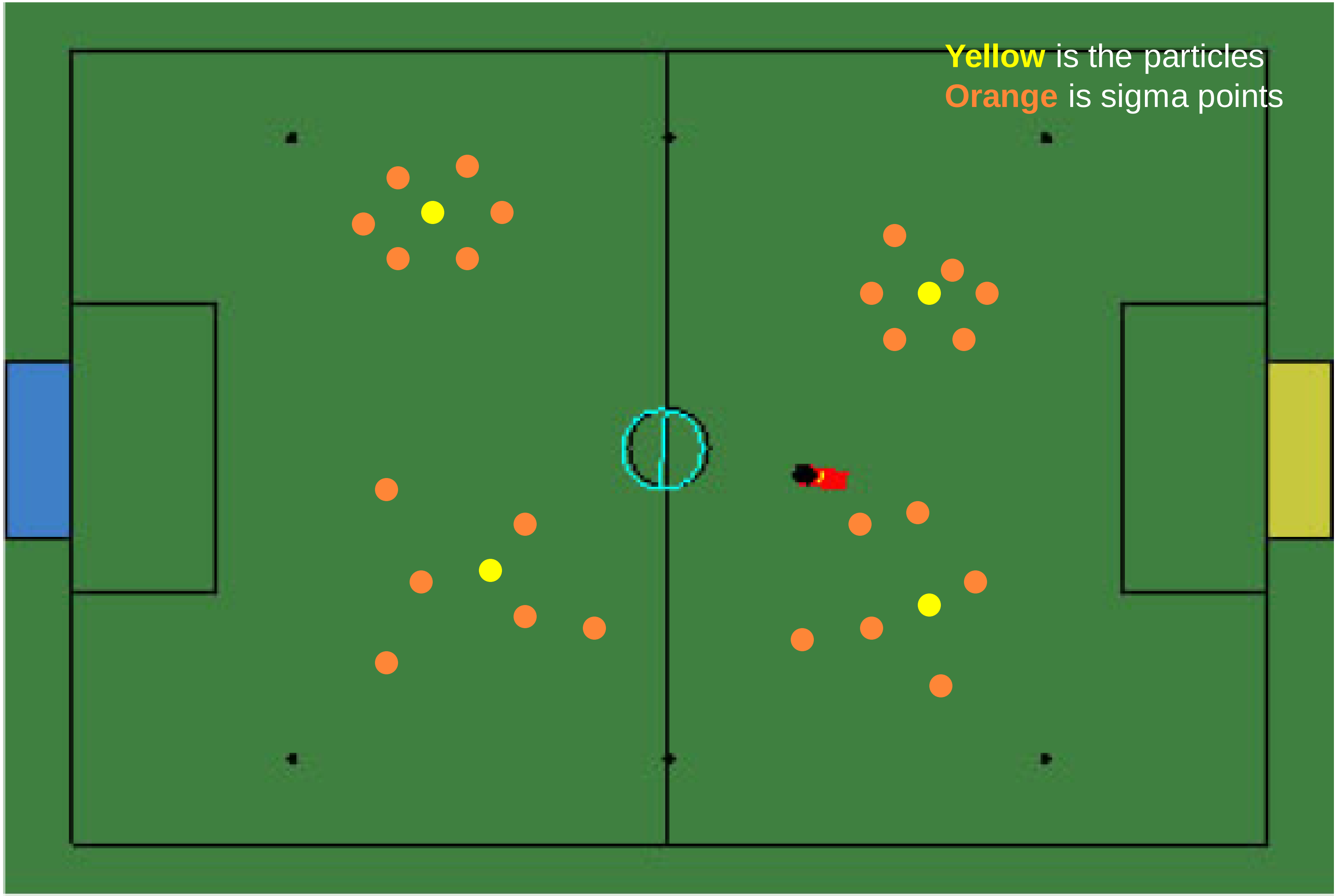
UNSCENTED PARTICLE FILTER

1. At each step for each particle:
   1. Calculate the Sigma Points
   2. Apply Kalman Update Equations
   3. Normalize and get mean and covariance for each particle

2. Continue the PF as known before

- It looks like it has more computation, but if particles number is small it will reduce computation and increase accuracy

UNSCENTED PARTICLE FILTER



MIXED PARTICLE FILTER

1. For p% of the N particles:
   - Apply Unscented Particle Filter

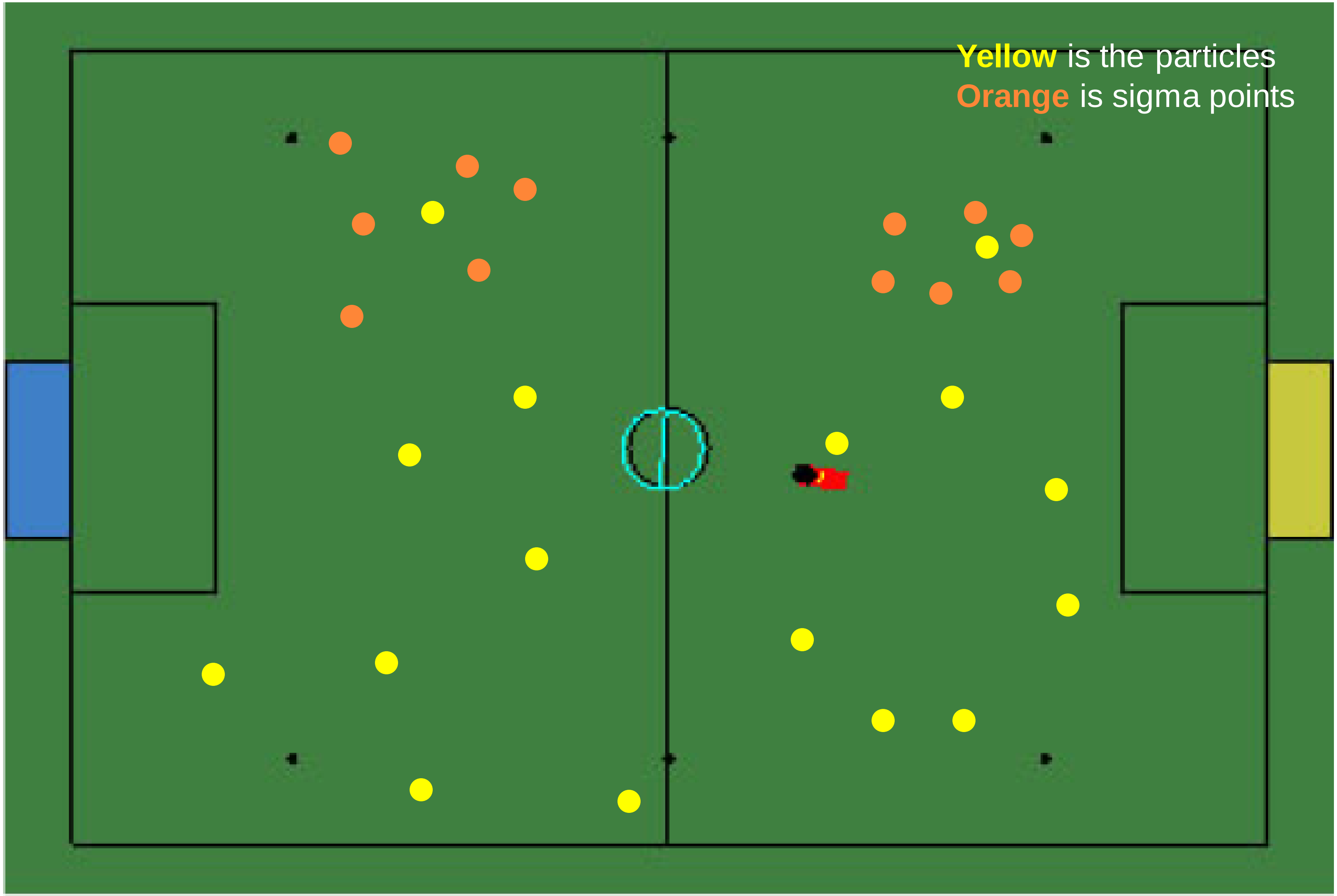
2. For (100-p)% of the particles:
   - Apply the normal particle filter

3. *Normalize* all weights from 1,2

4. Resampling

"A Mixed Fast Particle Filter"
Fasheng Wang, Qingjie Zhao, and Hongbin Deng

Mixed Filter

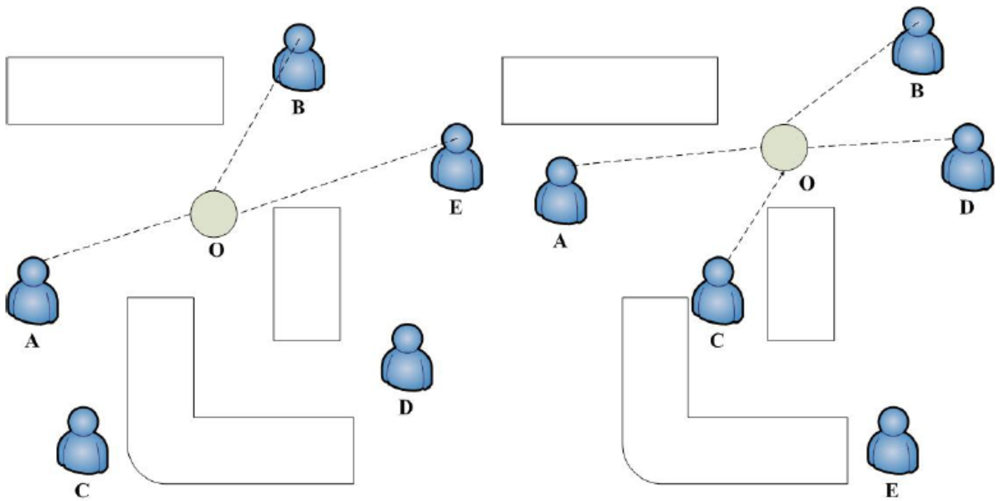


Cooperative Multi-Robot Localization

- In **Peking University**, China, they suggested the concept of "**Dynamic Object Reference**"
- Dynamic Object Reference:
  - A human can self-localize himself by putting, for example, special building as a reference (static)
  - However in Mobile Robots & Dynamic Environment, we need to have a Dynamic Reference
  - This dynamic reference object can be detected by all robots
  - For one robot: reliable self-localization => reliable object position
  - Normally, the dynamic object is the ball. So, "**ball localization**" is involved

Dynamic Reference Object

- So, for Multi Robots: they can all exchange a 'team message'
  - Object Position, Robot ID, Time, and Position Probability
  - For example



(a) t=t_1 (b) t=t_2

Calculated Position	Robot ID	Time	Position Possibility
(2388, 700)	A	t_1	0.71
(2264, 658)	B	t_1	0.92
(2530, 710)	E	t_1	0.86
(2368, 803)	A	t_2	0.81
(2401, 801)	B	t_2	0.91
(2103, 743)	C	t_2	0.32
(2215, 725)	D	t_2	0.43

- Then, Robot B can be reliable for the object position

Cooperative Multi-Robot Localization

- Common approaches for applying cooperation in multiple robots normally have the assumption that robot can identify other robots
- However, Robots only need to recognize the object instead of identify all the robots in the team
- **Algorithm**: Using Bayes Filter, like Particle Filter
  - After performing the usual PF
  - If robot B is reliable, then robot A belief about its own position would be modified by

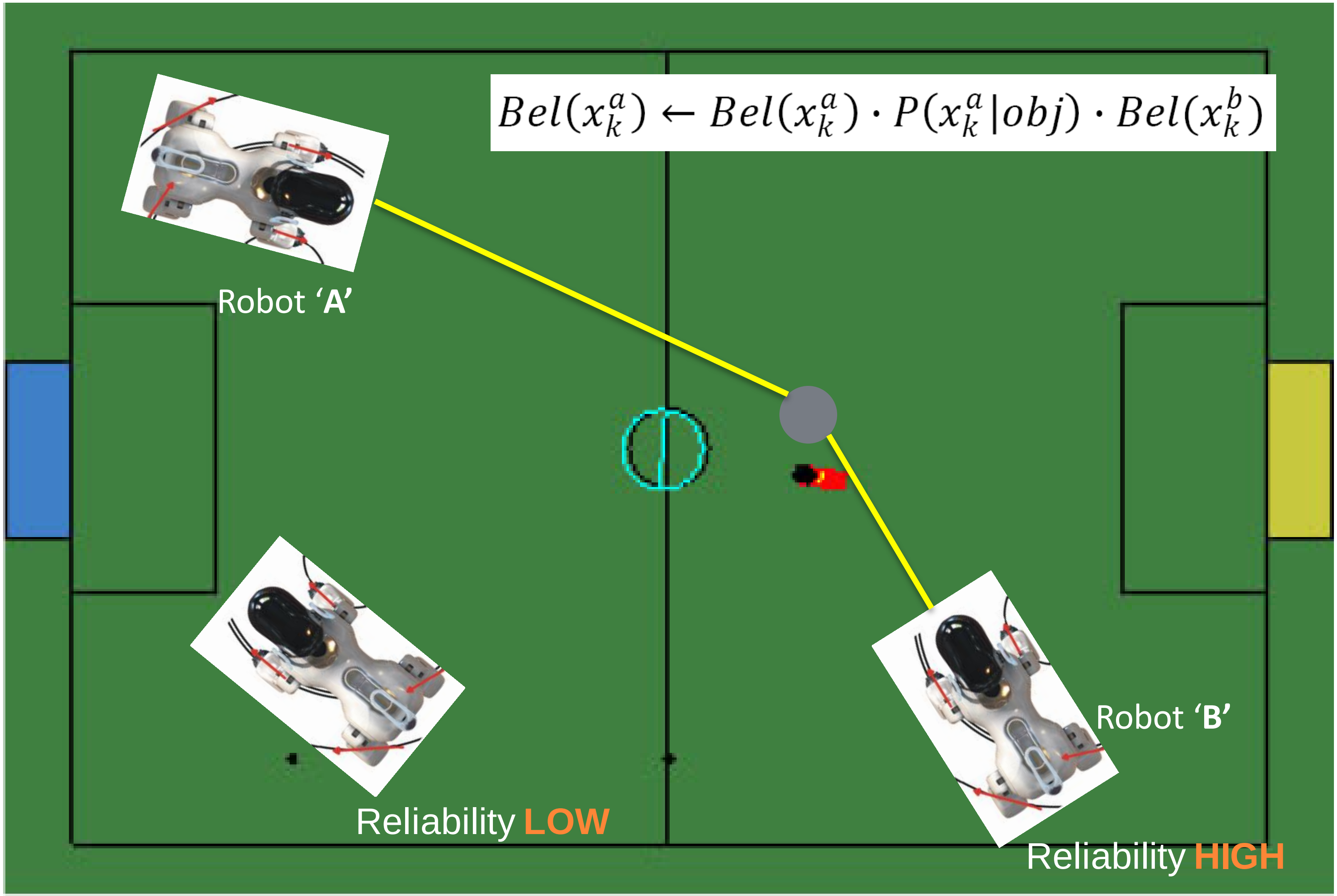
$$Bel(x_k^a) \leftarrow \boxed{Bel(x_k^a)} \cdot \boxed{P(x_k^a|obj)} \cdot \boxed{Bel(x_k^b)}$$

Normal Robot Self-Localization

Some Evaluation between the estimated object position and the estimated robot position

The reliable robot probability

Cooperative Robot Localization



PARTICLE FILTER

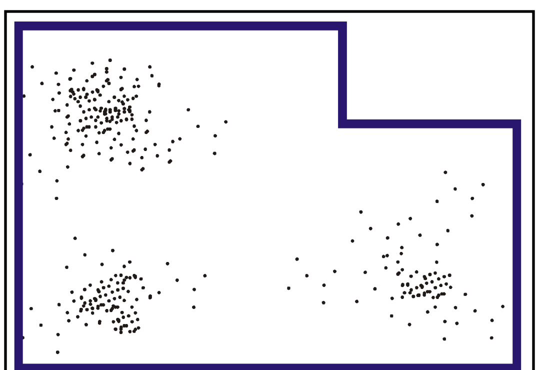
- Used where models are non-linear and noise is non-Gaussian.
- Use Particles to represent the distribution

$$P(x_t \mid y_{1:t}) = \frac{1}{c_t} P(y_t | x_t) \boxed{\int_Z P(x_t | x_{t-1} = z) P(x_{t-1} = z \mid y_{1:t-1}) dz}$$

Motion model

Observation model (=weight)

Proposal distribution



Sequential Importance Sampling

- Basis for most Monte Carlo Filters
- Technique for implementing recursive Bayesian Filter by Monte Carlo Simulation.
- Represent a set of required Posterior Probability by a set of random samples with weights
- As the number of samples becomes very large, the SIS approach optimality.

Sequential Importance Sampling

$\{\mathbf{x}_{0:k}^i\}$: set of support points (samples, particles)
$i = 1, \ldots, N_s$
(whole trajectory for each particle!)

w_k^i : associated weights, normalized to $\sum_i w_k^i = 1$

Then:

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(\mathbf{x}_{0:k} - \mathbf{x}_{0:k}^i)$$

(discrete weighted approximation to the true posterior)

Sequential Importance Sampling

- The weights are chosen based on the principle of importance sampling.
- $p(x) \propto \pi(x)$ difficult to draw samples
- $x^i \sim q(x),\ i = 1, \ldots, N_s$ (q: importance density easy to draw samples where

$$p(x) \approx \sum_{i=1}^{N_s} w^i \delta(x - x^i)$$

- If samples are drawn according to q then weights are defined as

$$w^i \propto \frac{\pi(x^i)}{q(x^i)} \qquad w_k^i \propto \frac{p(\mathbf{x}_{0:k}^i|\mathbf{z}_{1:k})}{q(\mathbf{x}_{0:k}^i|\mathbf{z}_{1:k})}$$

SEQUENTIAL IMPORTANCE SAMPLING

- If we choose q as $q(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) = q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{z}_{1:k})q(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1})$

  then we can update the sample using existing samples + new state

---

ALGORITHM 1: SIS PARTICLE FILTER

$[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}] = \text{SIS}\ [\{\mathbf{x}_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, \mathbf{z}_k]$

- FOR $i = 1 : N_s$
  - Draw $\mathbf{x}_k^i \sim q(\mathbf{x}_k|\mathbf{x}_{k-1}^i, \mathbf{z}_k)$
  - Assign the particle a weight, w_k^i, according to (48)
- END FOR

Degeneracy Problem

- All but one particle will have negligible weight
- This makes huge computation effort for samples whose contributions is almost zero.
- It can be solved either by good choice of importance function or resampling
- Good choice of importance function requires evaluation of the integrals and drawing samples from P, which is not possible in most cases.

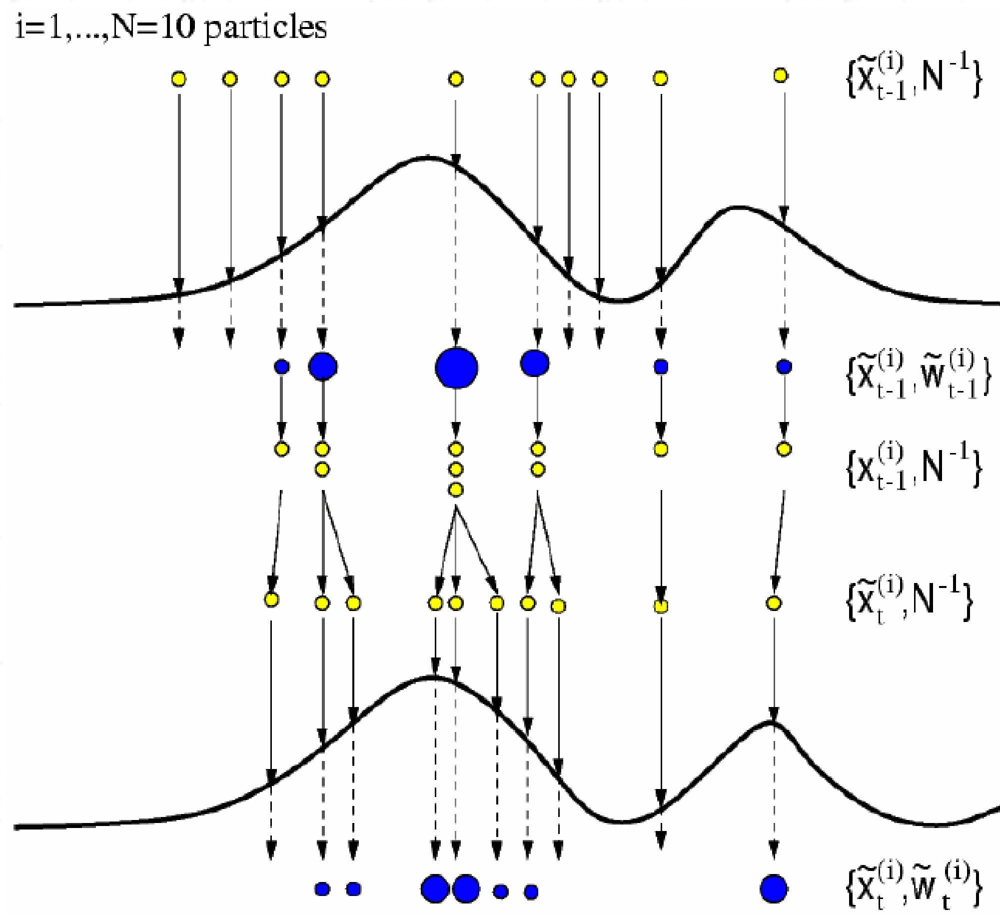
So…

- So at each time step
  - For each particle:
    - Use motion model to predict new pose (sample from transition priors)
    - Use observation model to assign a weight to each particle (posterior/proposal)
  - Create A new set of equally weighted particles by sampling the distribution of the weighted particles produced in the previous step.

Resampling

- The basic idea of resampling is to eliminate particles with small weights and concentrate on particles with large weights.
- Resample from: $$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i)$$
- Weights are reset $1/N_s$

Resampling



i=1,...,N=10 particles
$\{\tilde{x}_{t-1}^{(i)}, N^{-1}\}$
$\{\tilde{x}_{t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\}$
$\{x_{t-1}^{(i)}, N^{-1}\}$
$\{\tilde{x}_{t}^{(i)}, N^{-1}\}$
$\{\tilde{x}_{t}^{(i)}, \tilde{w}_{t}^{(i)}\}$

RESAMPLING

Systematic Resampling

- Simple to implement
- O(Ns)
- Minimize the variation.

ALGORITHM 2: RESAMPLING ALGORITHM

$[\{\mathbf{x}_k^{j*}, w_k^j, i^j\}_{j=1}^{N_s}] = \text{RESAMPLE}\ [\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$

- Initialise the CDF: $c_1 = 0$
- FOR $i = 2 : N_s$
  - Construct CDF: $c_i = c_{i-1} + w_k^i$
- END FOR
- Start at the bottom of the CDF: $i = 1$
- Draw a starting point: $u_1 \sim \mathbb{U}\left[0, N_s^{-1}\right]$
- FOR $j = 1 : N_s$
  - Move along the CDF: $u_j = u_1 + N_s^{-1}(j-1)$
  - WHILE $u_j > c_i$
    - $i = i + 1$
  - END WHILE
  - Assign sample: $\mathbf{x}_k^{j*} = \mathbf{x}_k^i$
  - Assign weight: $w_k^j = N_s^{-1}$
  - Assign parent: $i^j = i$
- END FOR

PARTICLE FILTER

ALGORITHM 3: GENERIC PARTICLE FILTER

$[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}] = \text{PF}\ [\{\mathbf{x}_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, \mathbf{z}_k]$

- FOR $i = 1 : N_s$
  - Draw $\mathbf{x}_k^i \sim q(\mathbf{x}_k | \mathbf{x}_{k-1}^i, \mathbf{z}_k)$
  - Assign the particle a weight, w_k^i, according to (48)
- END FOR
- Calculate total weight: $t = \text{SUM}\ [\{w_k^i\}_{i=1}^{N_s}]$
- FOR $i = 1 : N_s$
  - Normalise: $w_k^i = t^{-1} w_k^i$
- END FOR
- Calculate $\widehat{N_{eff}}$ using (51)
- IF $\widehat{N_{eff}} < N_T$
  - Resample using algorithm 2:
    * $[\{\mathbf{x}_k^i, w_k^i, -\}_{i=1}^{N_s}] = \text{RESAMPLE}\ [\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$
- END IF

PROBLEMS WITH RESAMPLING

- Limits the opportunity to paralelize since all particles need to be combined.
- Loss of Diversity among samples, we have many repeated points. (sample impoverishment).
- In the case of very small noise, all the particles will collapse to a single point within a few iteration.

Other Resampling Techniques

- Resample-move algorithm avoid sample impoverishment through rigorous manner that ensures particles asymptotically approximate samples from posterior.
- Regularization less rigorous

Sampling Importance Resampling

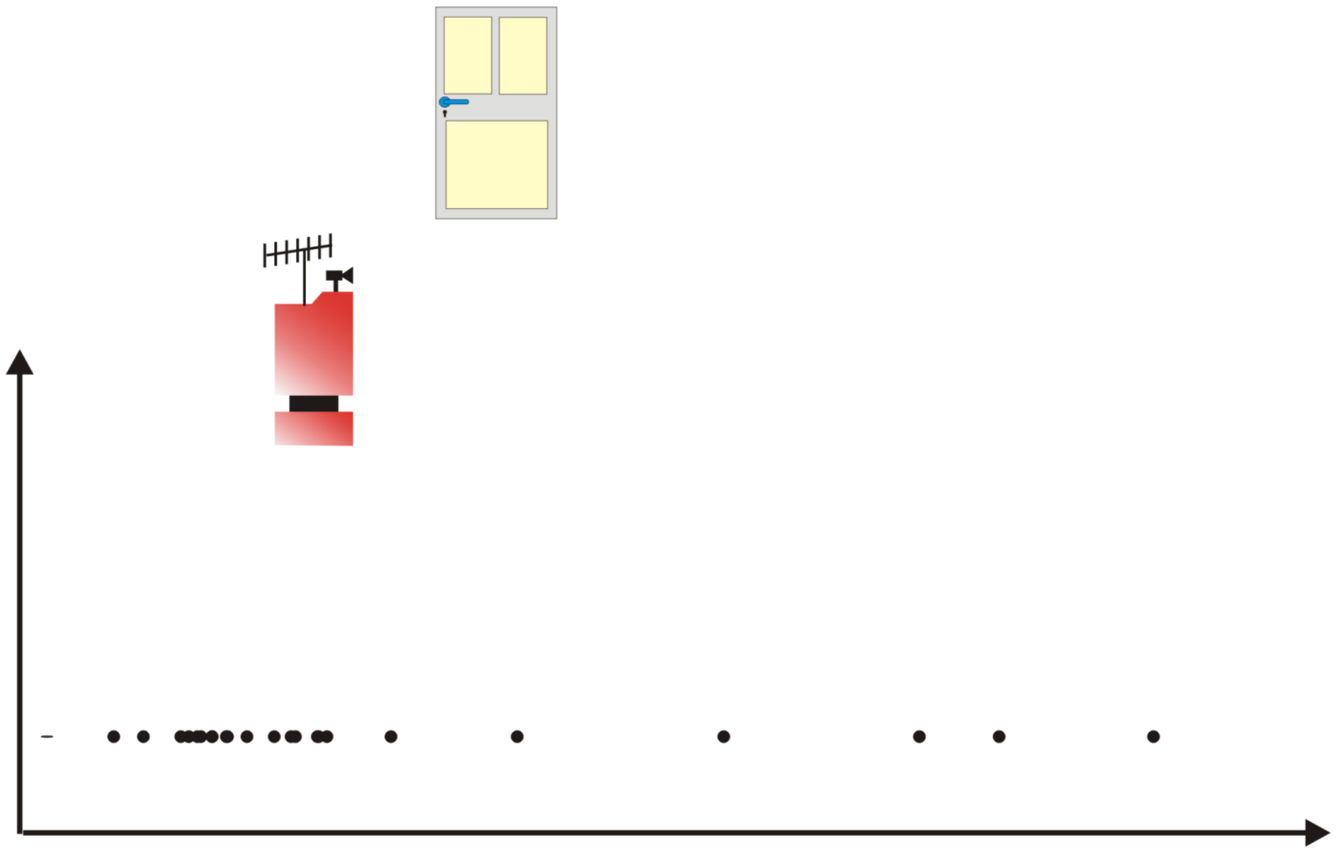
- Choice of
  - Importance density to be the prior density
  - Resampling step to be applied at every index
- Independence of measurements
  - Inefficient and sensitive to outliers
- Loss of diversity due to resampling
- Easy evaluation of importance weight and easy sampling of importance density.

ALGORITHM 4: SIR PARTICLE FILTER

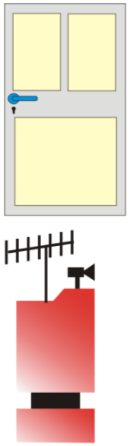
$[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}] = \text{SIR}\ [\{\mathbf{x}_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, \mathbf{z}_k]$

- FOR $i = 1 : N_s$
  - Draw $\mathbf{x}_k^i \sim p(\mathbf{x}_k | \mathbf{x}_{k-1}^i)$
  - Calculate $w_k^i = p(\mathbf{z}_k | \mathbf{x}_k^i)$
- END FOR
- Calculate total weight: $t = \text{SUM}\ [\{w_k^i\}_{i=1}^{N_s}]$
- FOR $i = 1 : N_s$
  - Normalise: $w_k^i = t^{-1} w_k^i$
- END FOR
- Resample using algorithm 2:
  - $[\{\mathbf{x}_k^i, w_k^i, -\}_{i=1}^{N_s}] = \text{RESAMPLE}\ [\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$

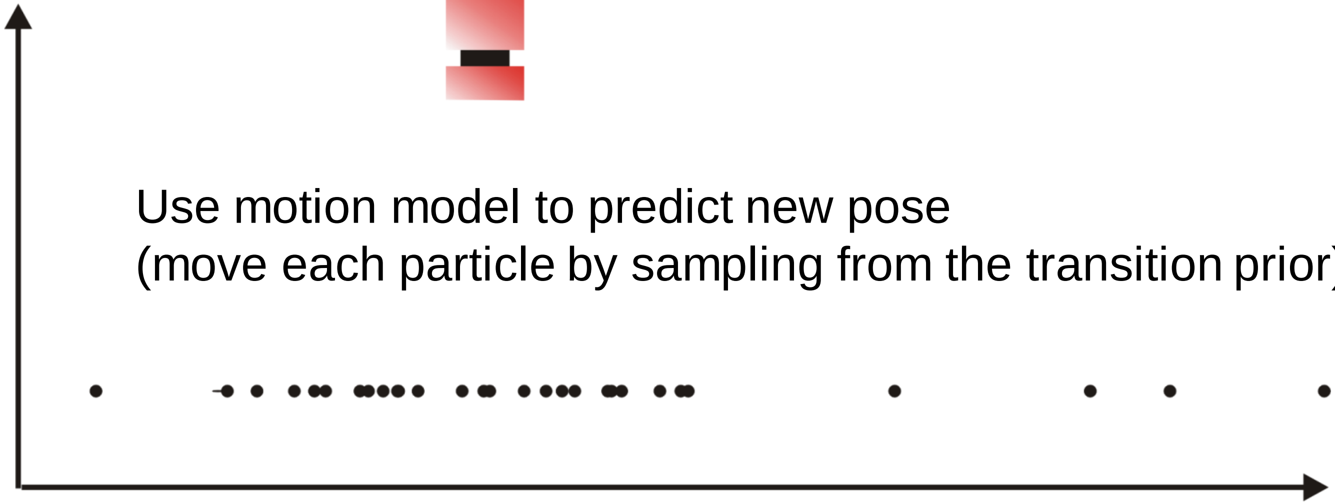
Particle Filters – Example 1



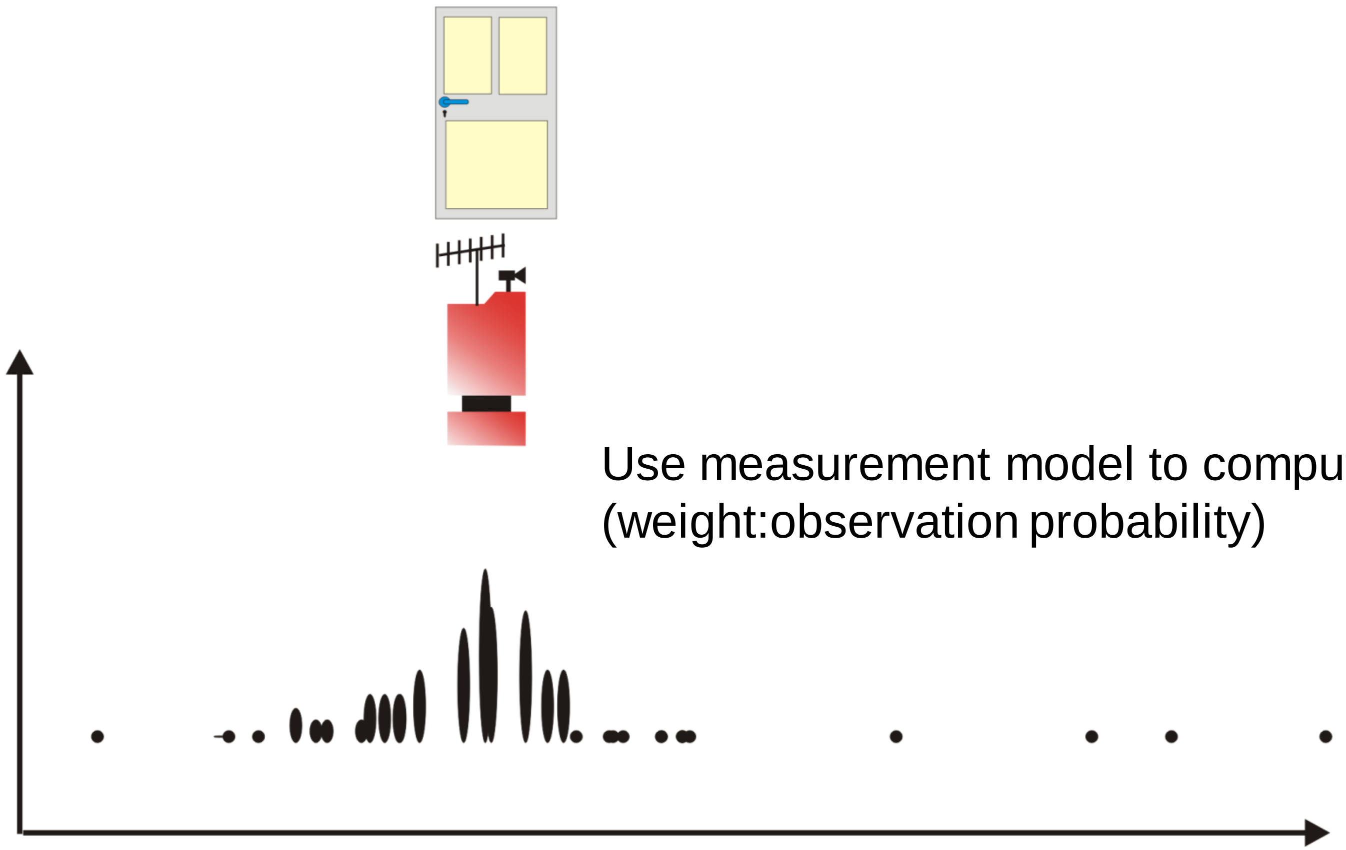
PARTICLE FILTERS – EXAMPLE 1



Use motion model to predict new pose
(move each particle by sampling from the transition prior)

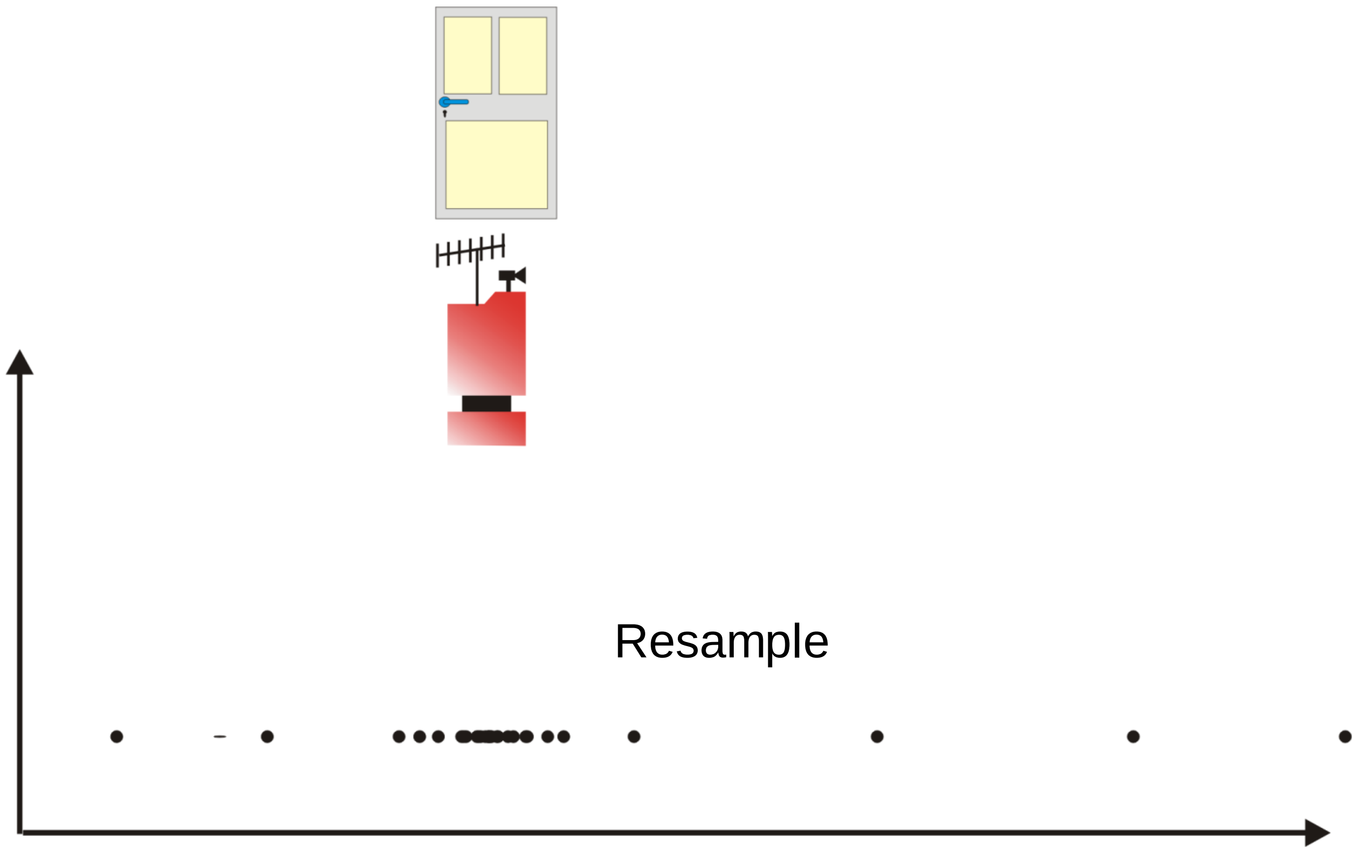


Particle Filters – Example 1

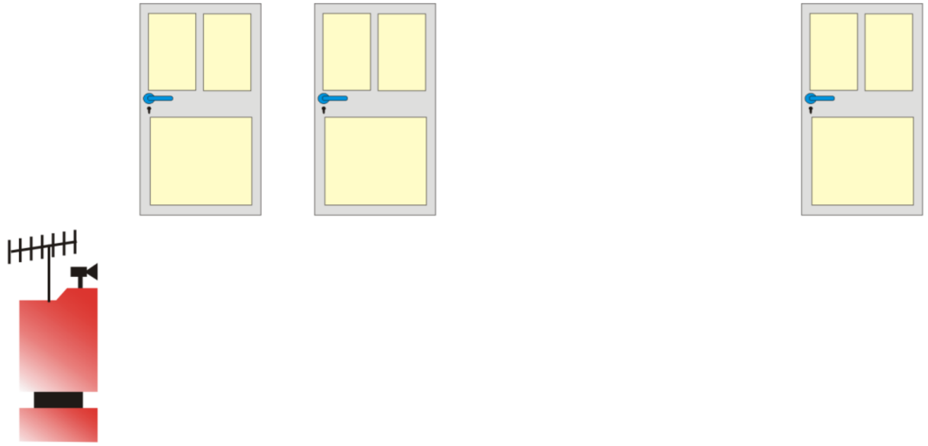


Use measurement model to compute weights
(weight:observation probability)

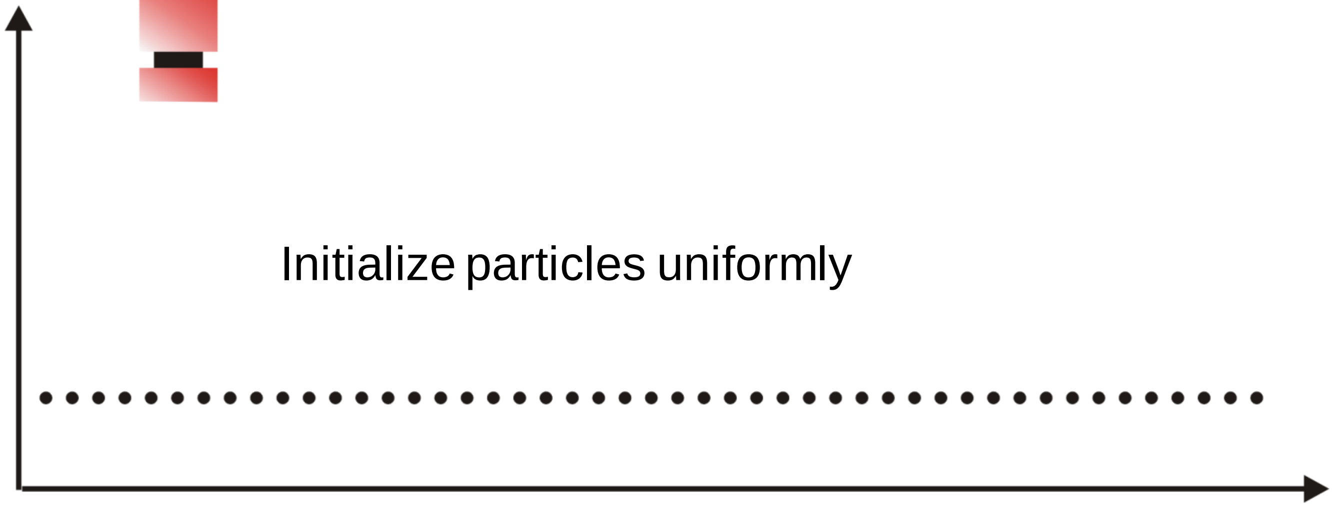
Particle Filters – Example 1



Particle Filters – Example 2



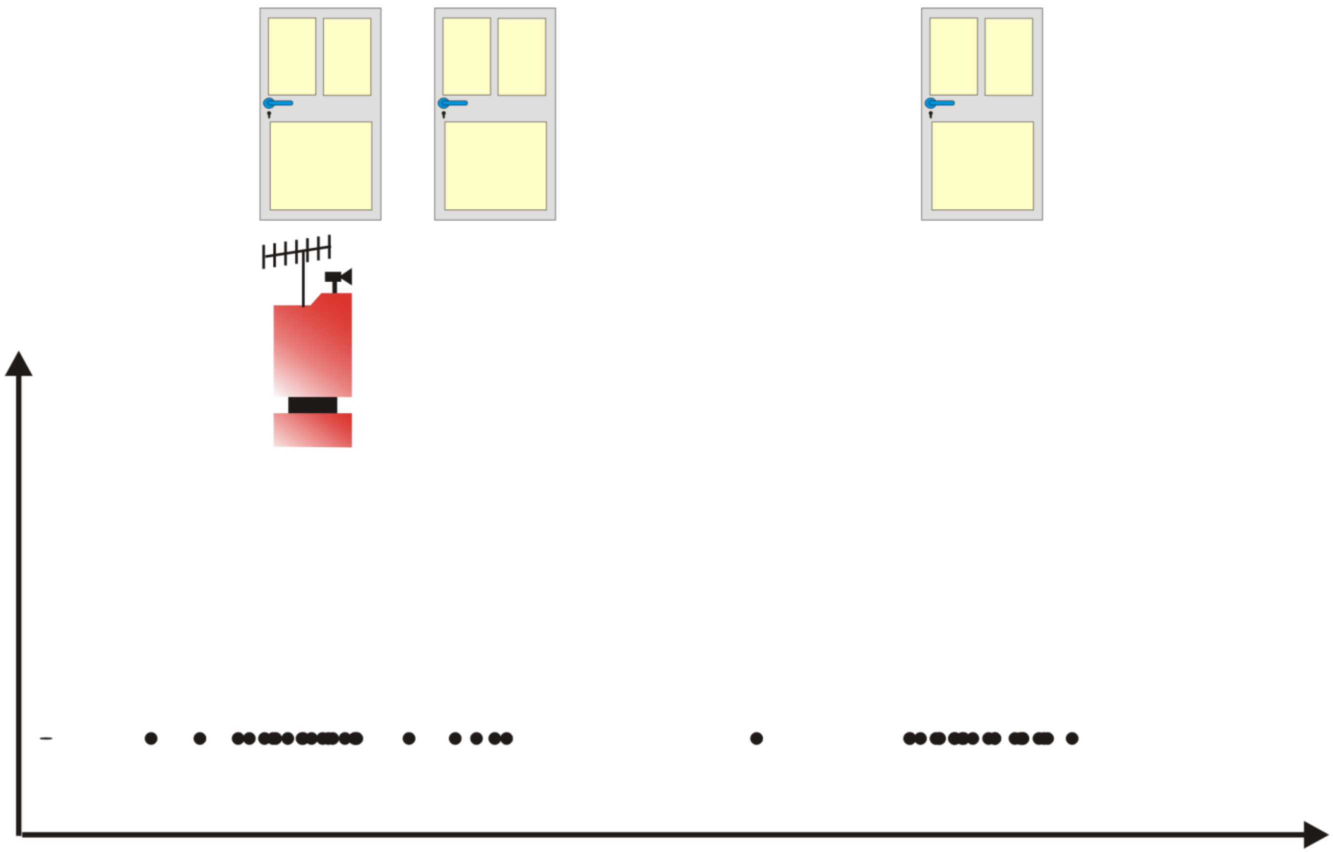
Initialize particles uniformly



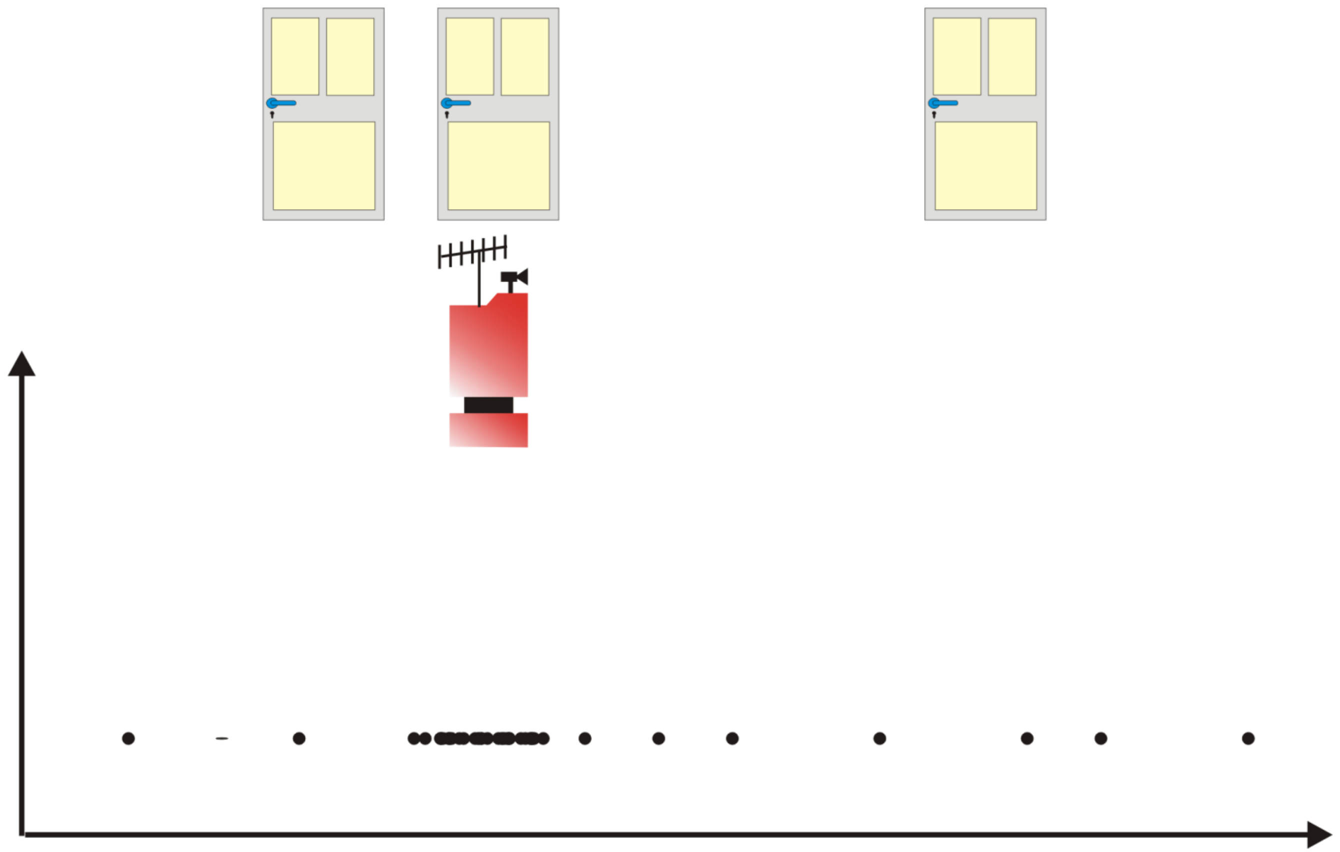
Particle Filters – Example 2

Particle Filters – Example 2

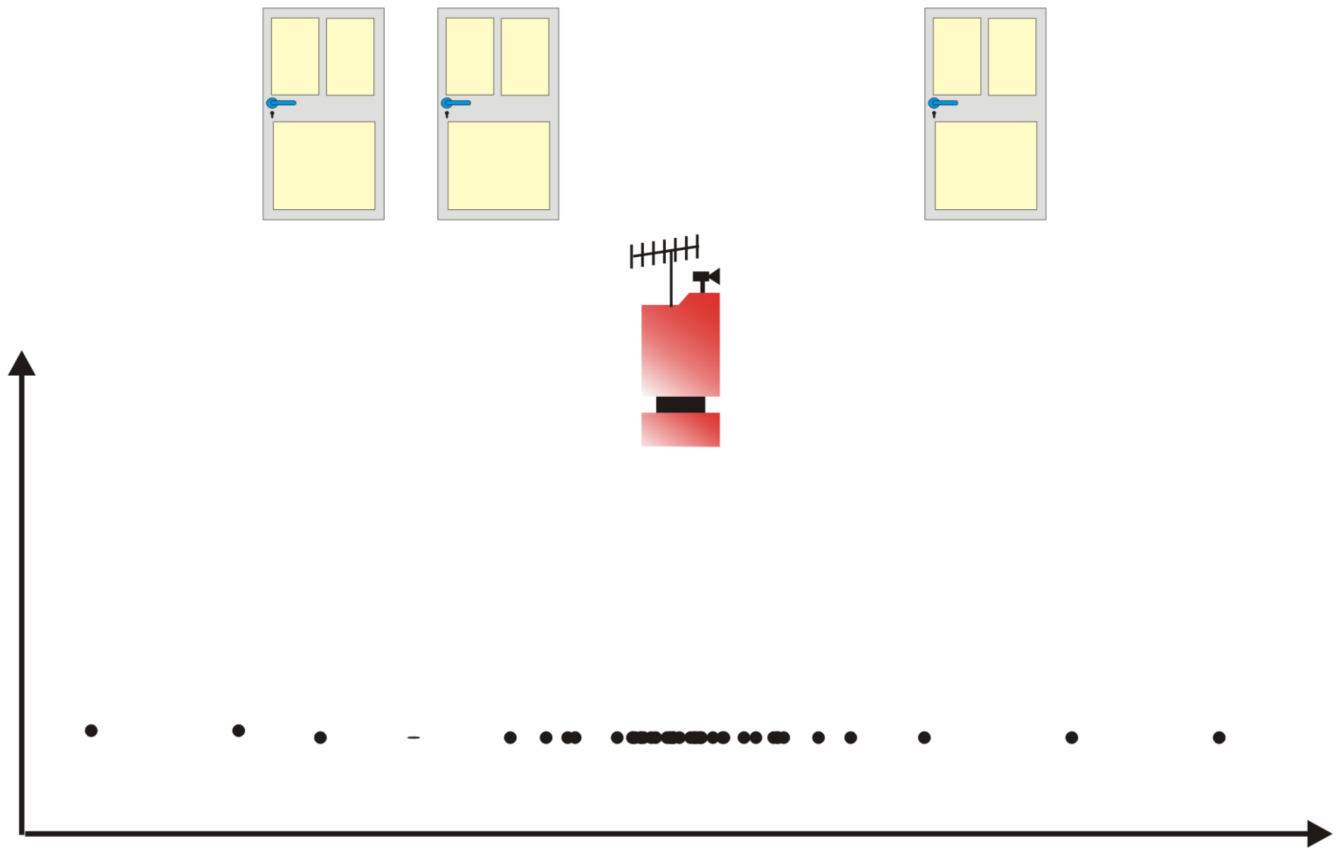
June 2008



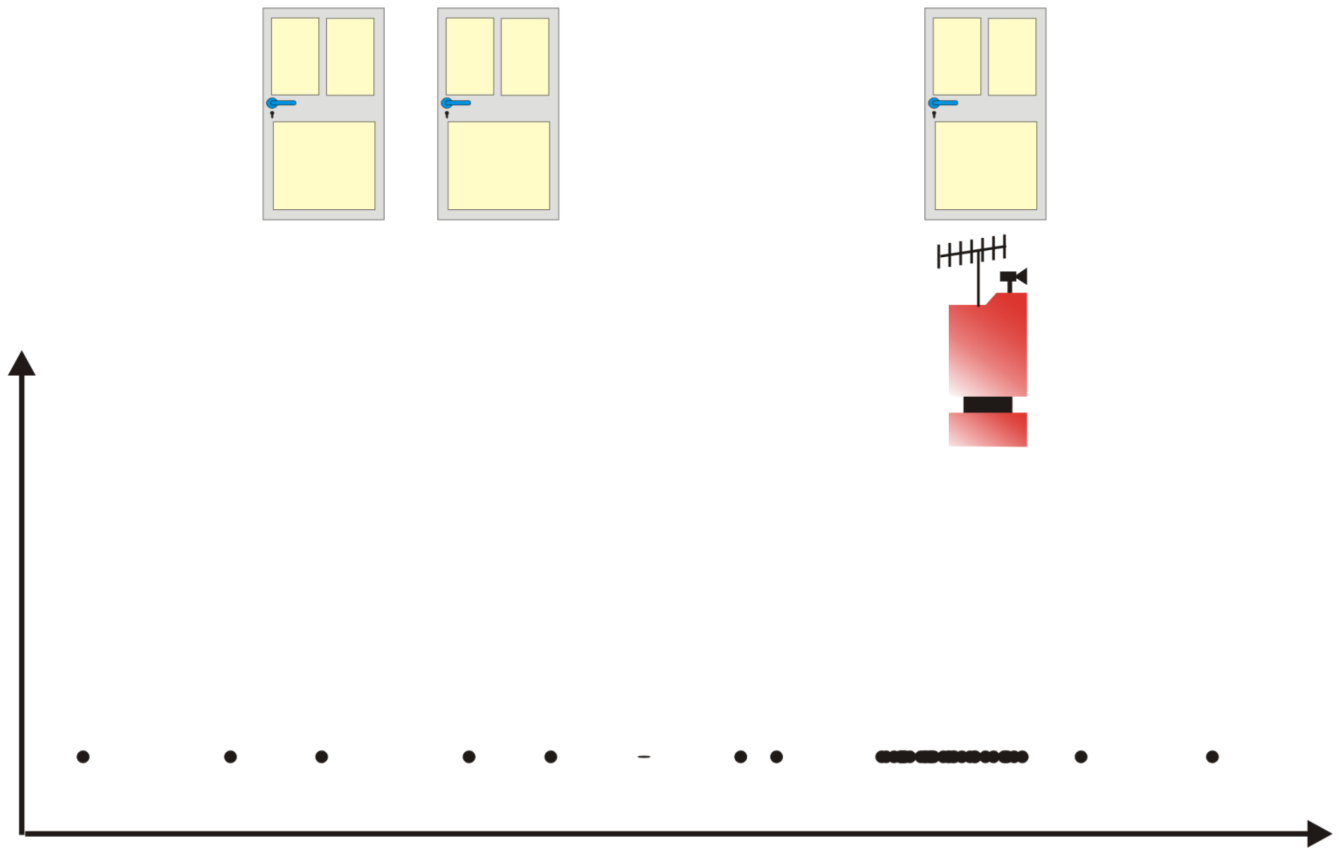
Particle Filters – Example 2



Particle Filters – Example 2



Particle Filters – Example 2



Continuous State Approaches Eg. Kalman

- Perform very accurately if the inputs are precise (performance is optimal with respect to any criterion in the linear case).
- Computational efficiency.

- Requirement that the initial state is known.
- Inability to recover from catastrophic failures
- Inability to track Multiple Hypotheses the state (Gaussians have only one mode)

Discrete State Approaches Eg. Particle

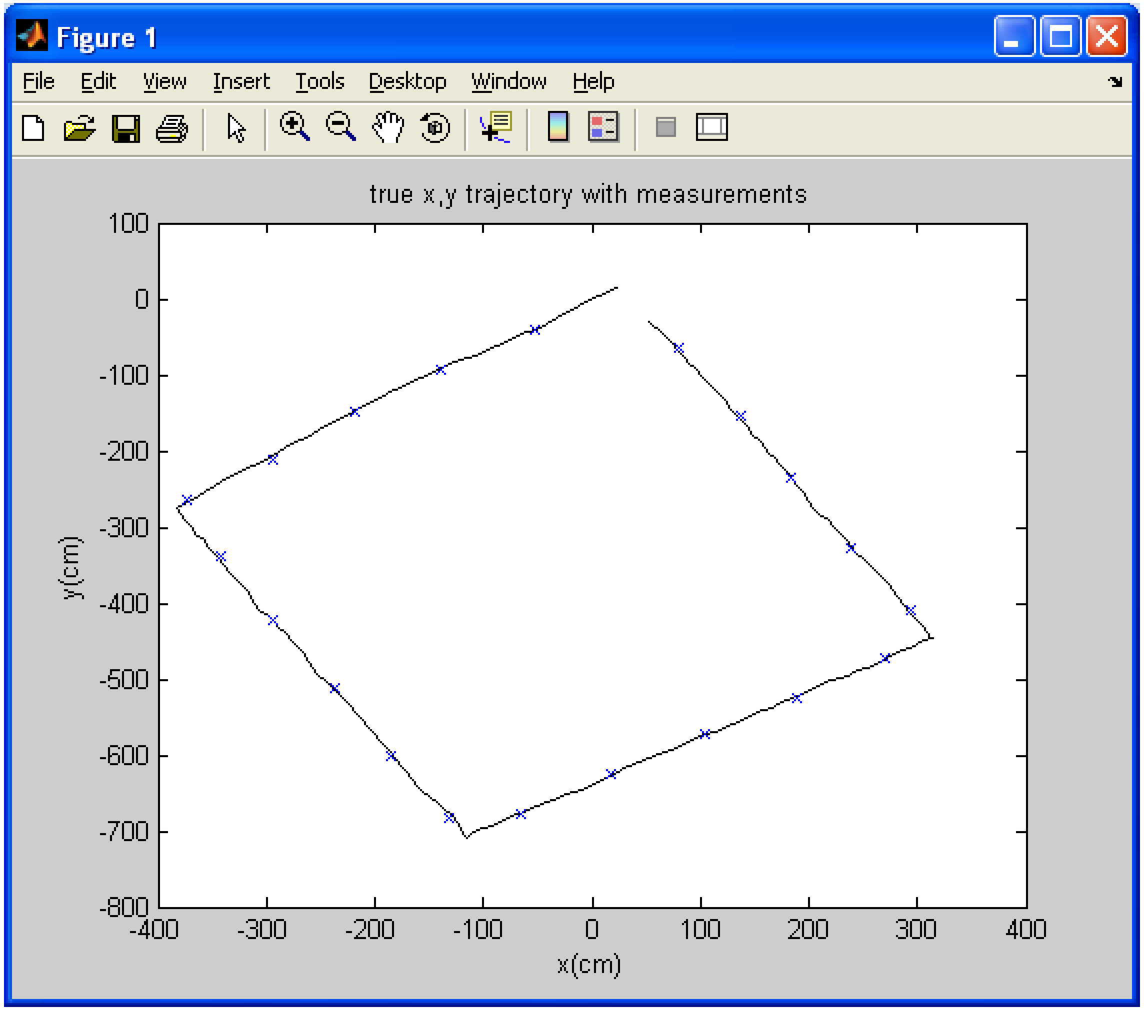
- Ability (to some degree) to operate even when its initial pose is unknown (start from uniform distribution).
- Ability to deal with noisy measurements.
- Ability to represent ambiguities (multi modal distributions).
- Computational time scales heavily with the number of possible states (dimensionality of the grid, number of samples, size of the map).
- Accuracy is limited by the size of the grid cells/number of particles-sampling method.
- Required number of particles is unknown

PARTICLE FILTER ADV. & DISADV.

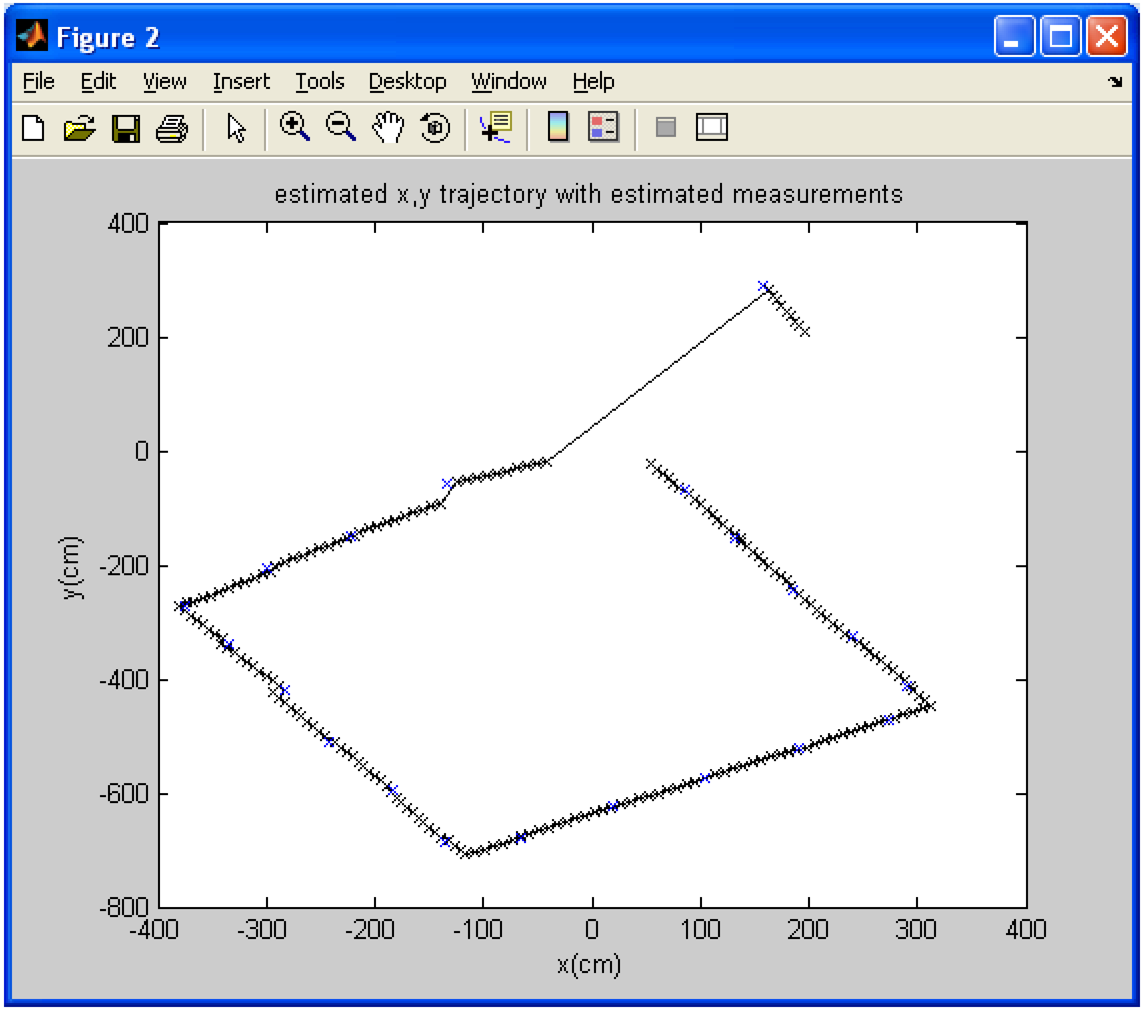
- Can deal with non-linearities.
- Can deal with non-Gaussian noise
- Can be implemented in O(Ns)
- Mostly parallelizable
- Easy to implement
- PFs Focus adaptively on probable regions of state space

- Included random element, they only convergence to posterior pdf if Ns → inf.
- If the assumptions of Kalman filters are valid, no PF can outperform it.
- Depending on the dynamic model, Gaussian sum filters, uncented kalman, or extended Kalman may produce satisfactory results at lower computation cost.

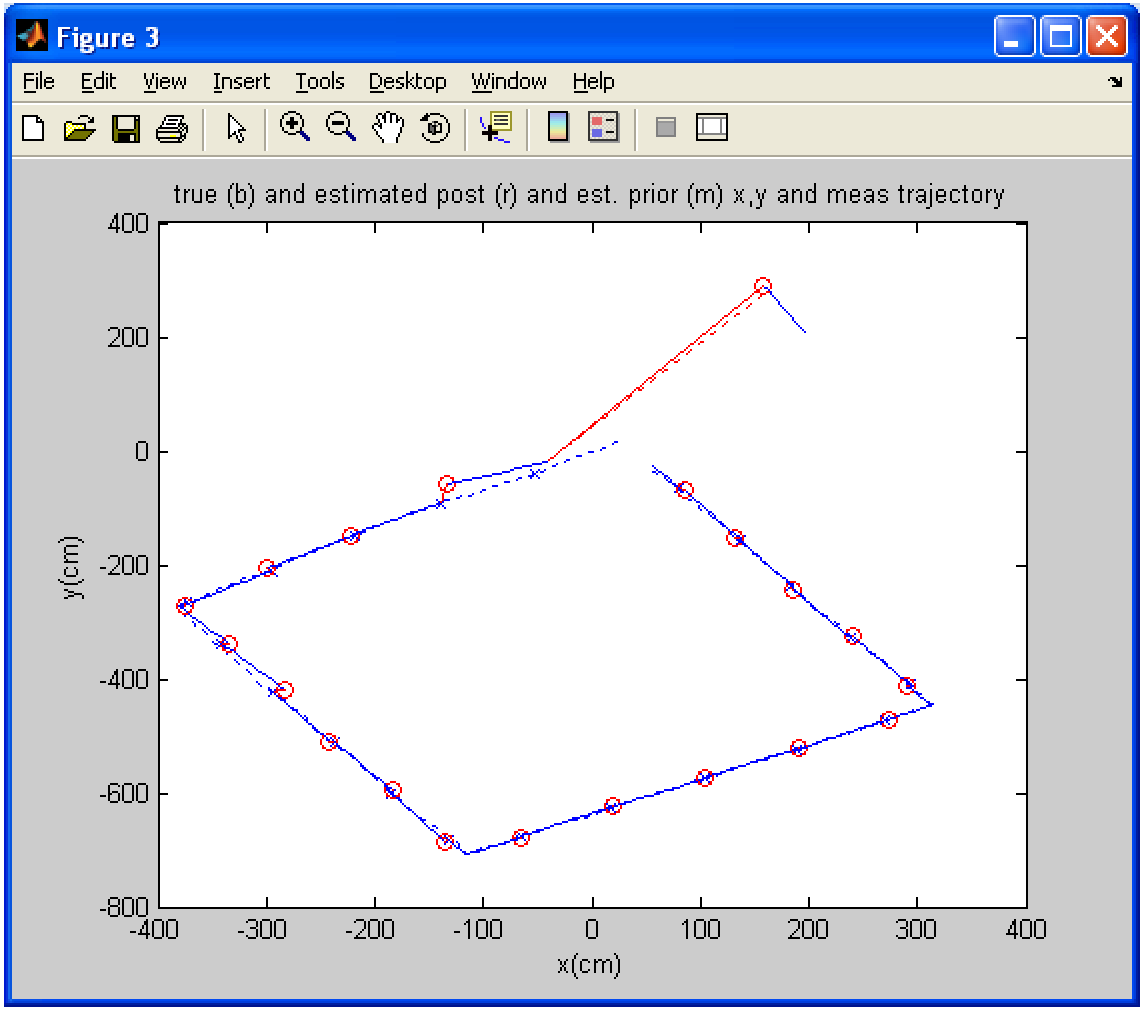
SIMULATION KALMAN FILTER – REAL PATH



Simulation Kalman Filter – estimated path



SIMULATION KALMAN FILTER – REAL ESTIMATED PATH



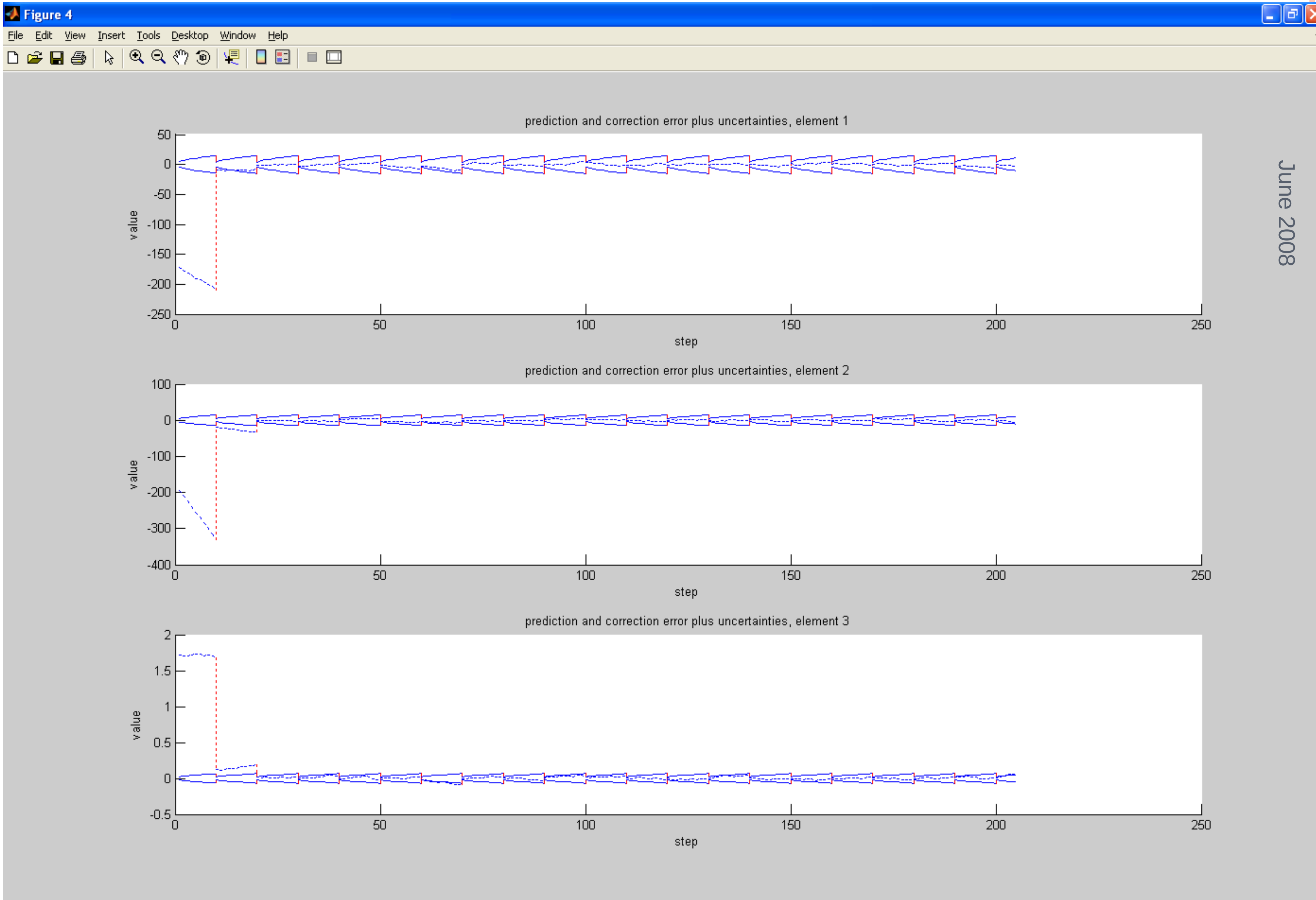
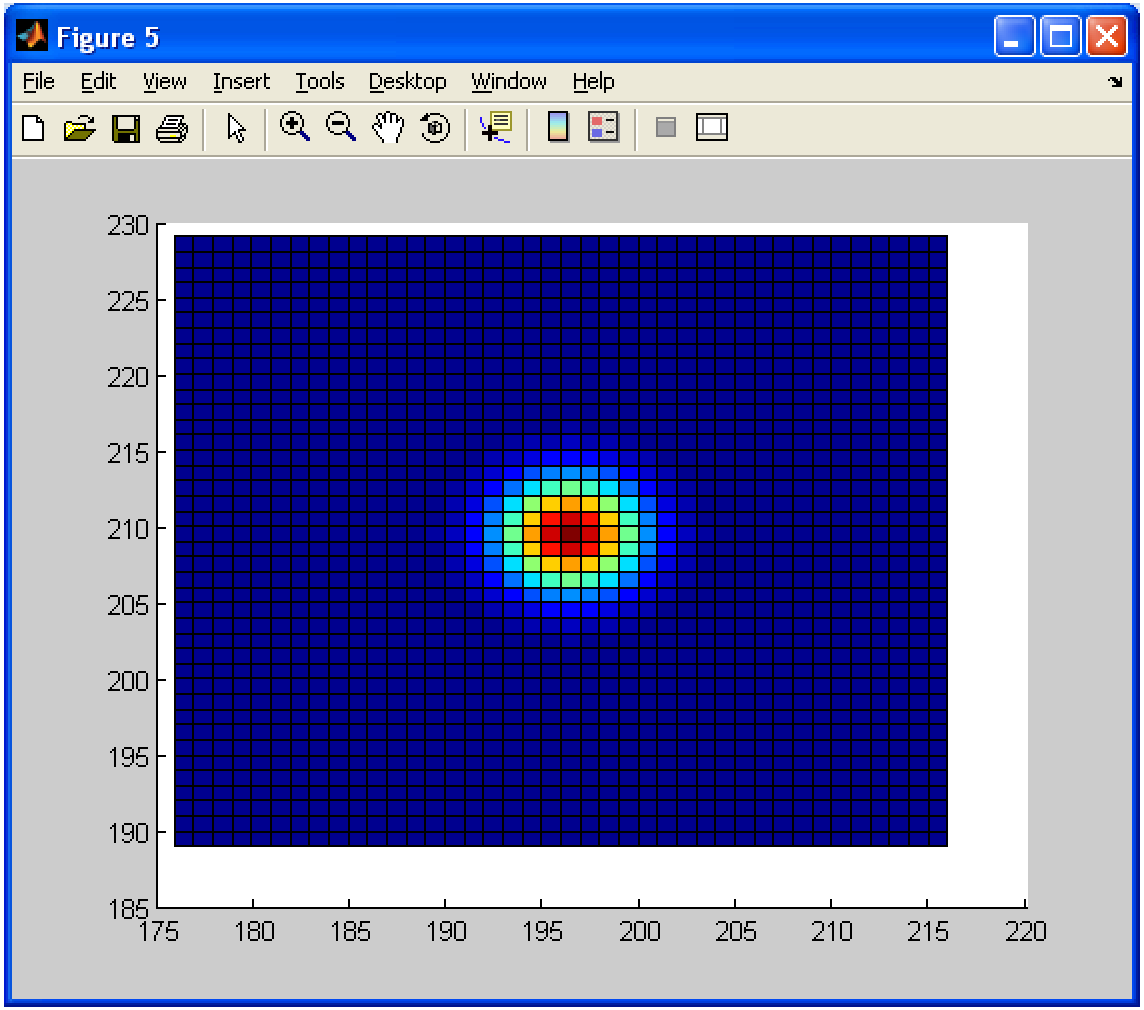


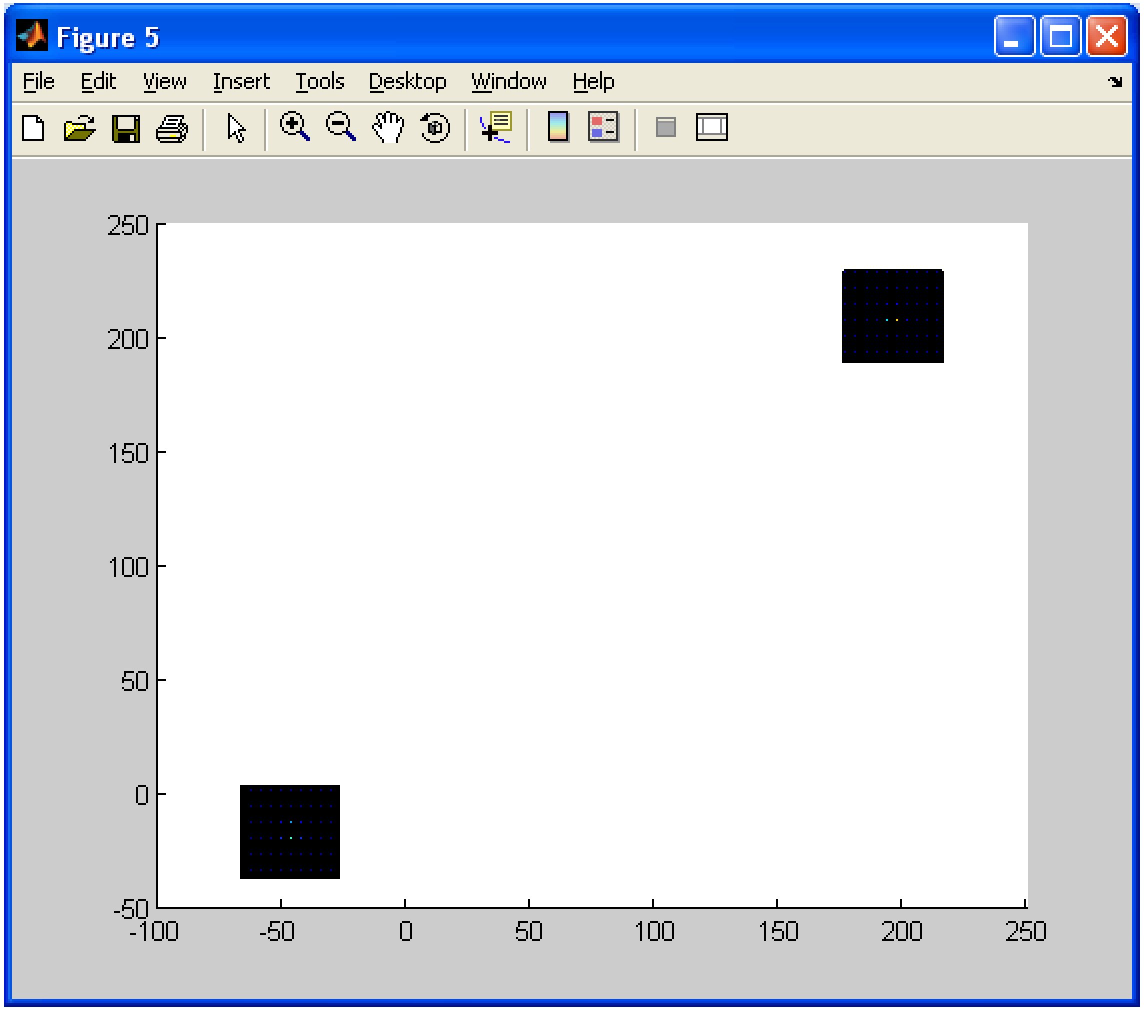
Figure 4
File Edit View Insert Tools Desktop Window Help
prediction and correction error plus uncertainties, element 1
value
step
prediction and correction error plus uncertainties, element 2
value
step
prediction and correction error plus uncertainties, element 3
value
step

June 2008

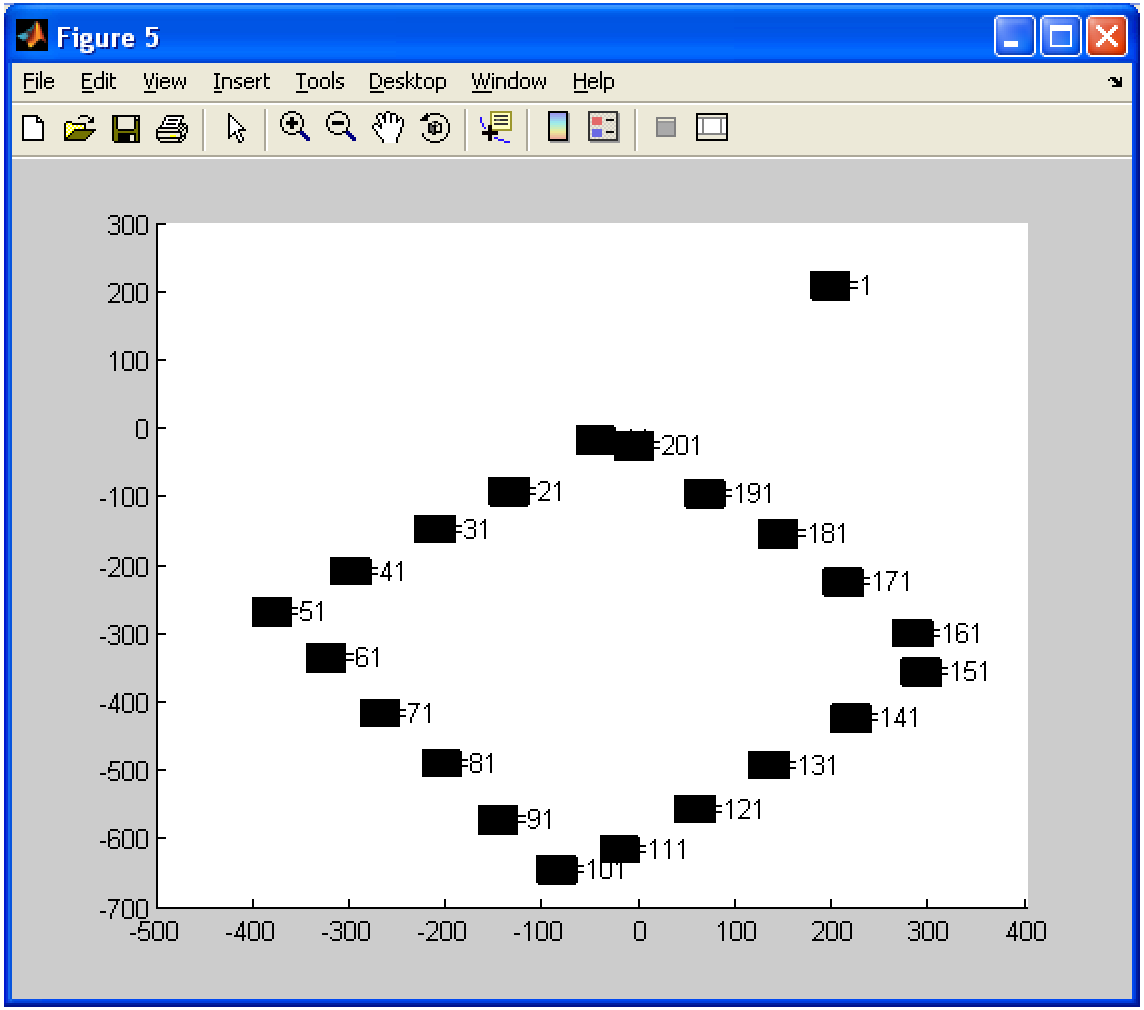
STATE STATE EVOLUTION



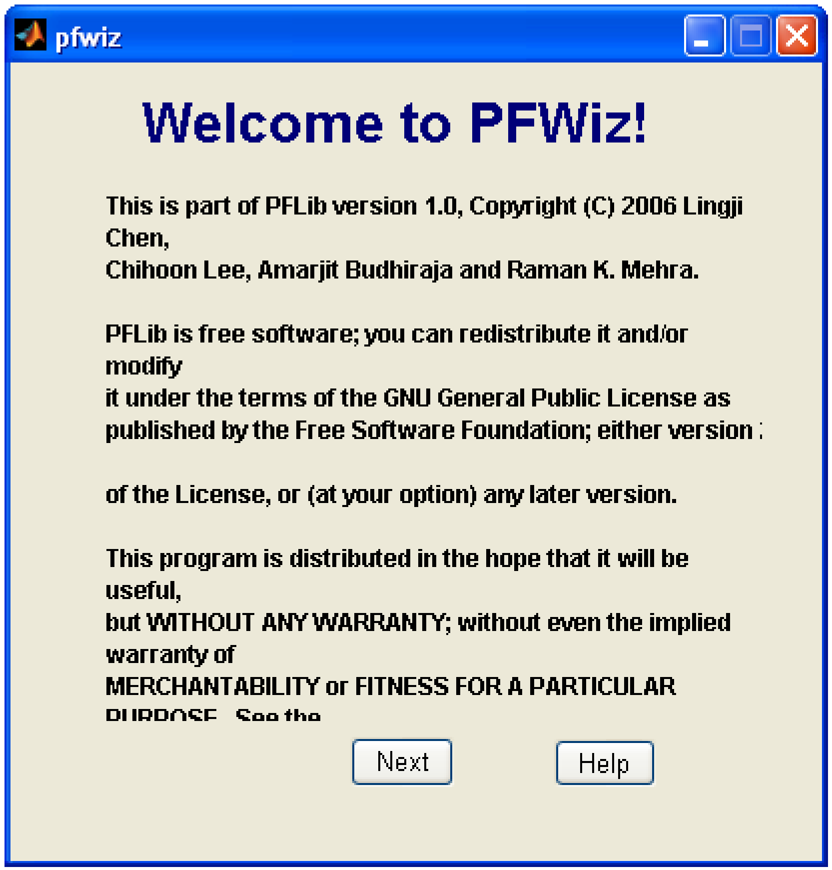
State Estimate Evolution



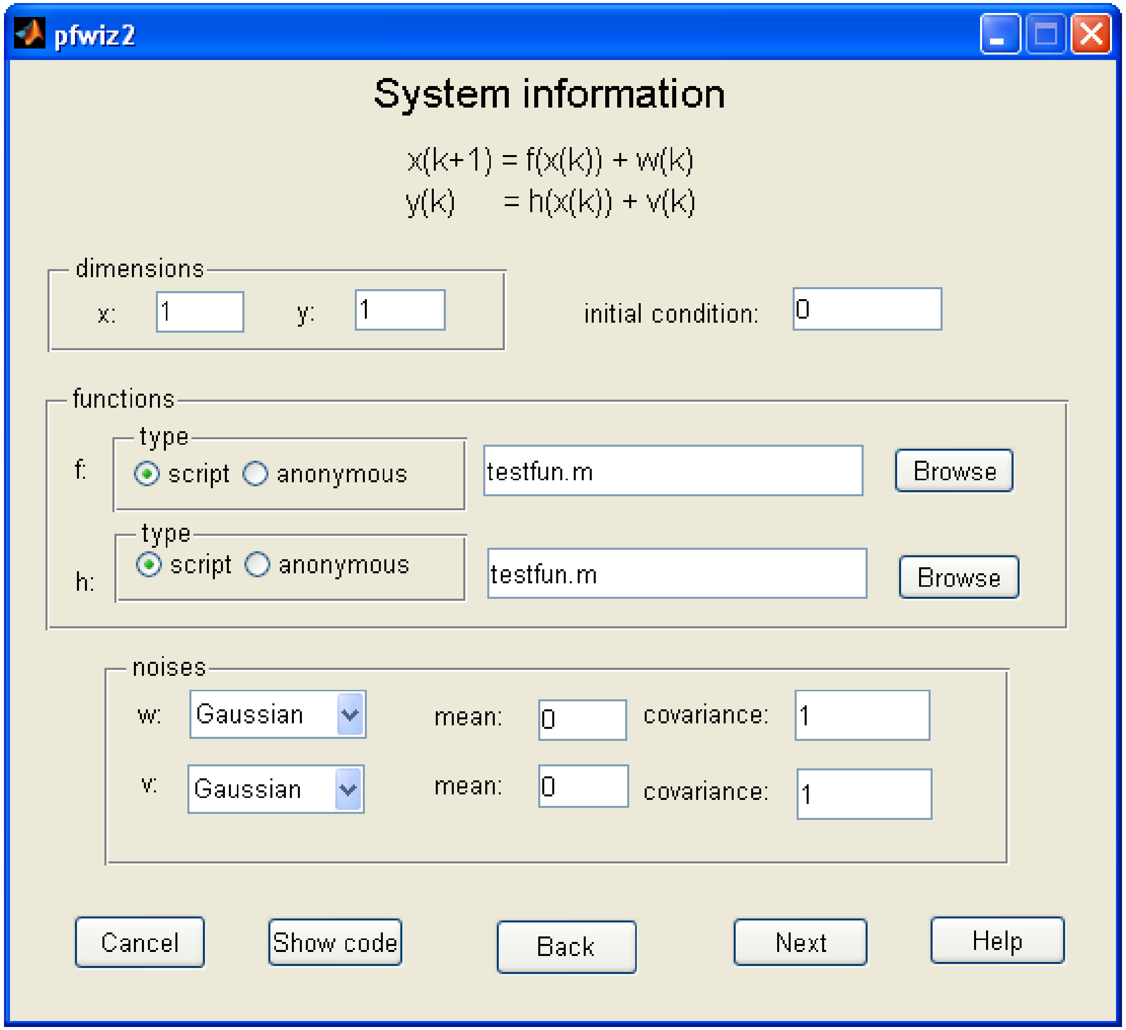
State Estimate Evolution



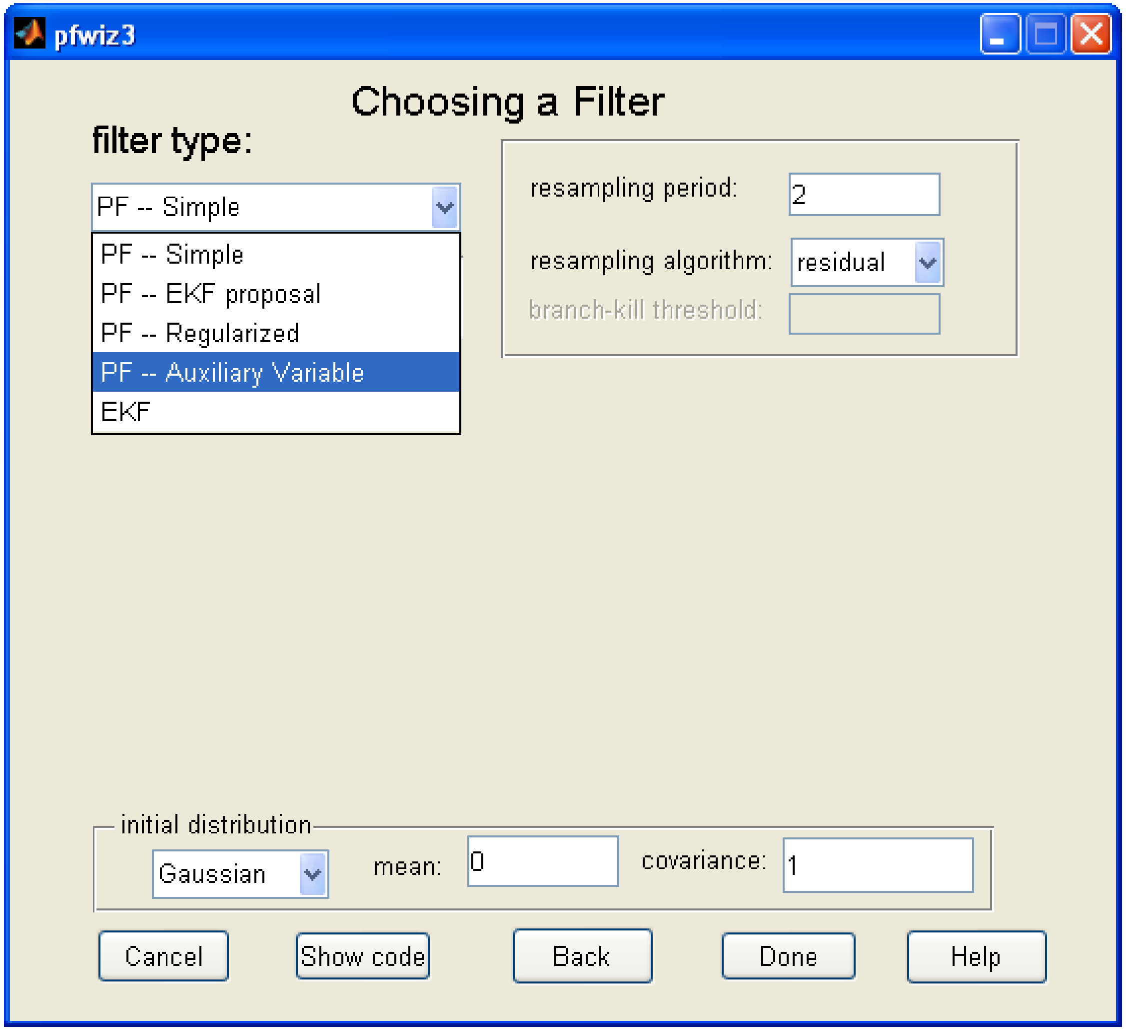
PARTICLE FILTER



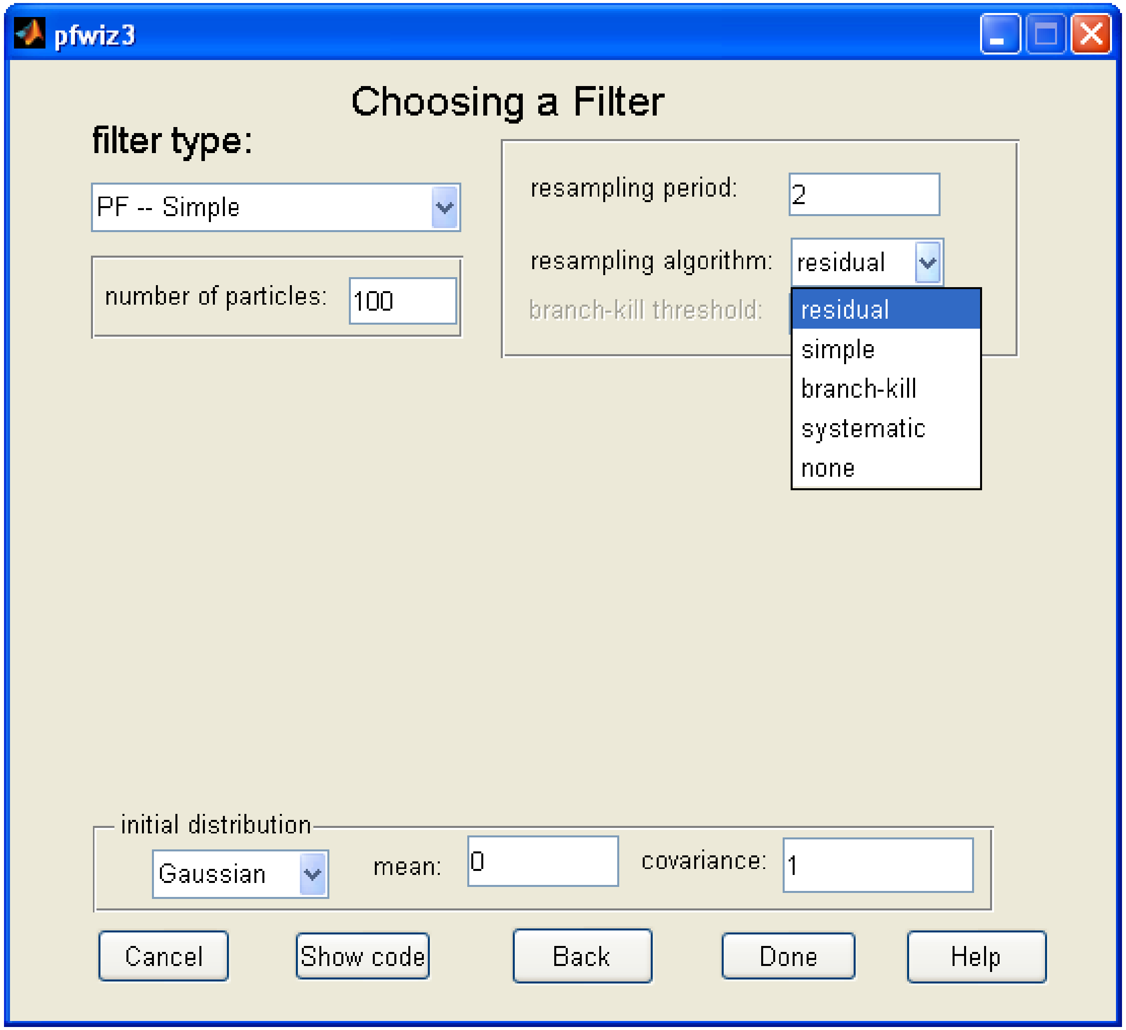
PARTICLE FILTER



Particle Filter



Particle Filter



Code…

C:\Documents and Settings\Ahmad Salam AlRefai\My Documents\MATLAB\PFLib\tmpCodeFile0001.m

```matlab
% The dimension variables were used by PFWiz for error checking
dimX = 1;
dimY = 1;
% initial condition of the system; NOT used to initialize the filter
initX = 0;
% function handle for state transition
f = @testfun;
% function handle for observation
h = @testfun;
meanW = 0;
covW = 1;
% constructing a Gaussian distribution
wDistr =  GaussianDistr(meanW, covW);
meanV = 0;
covV = 1;
vDistr =  GaussianDistr(meanV, covV);
% Constructing the system postulated by the filter.
```

June 2008

Results

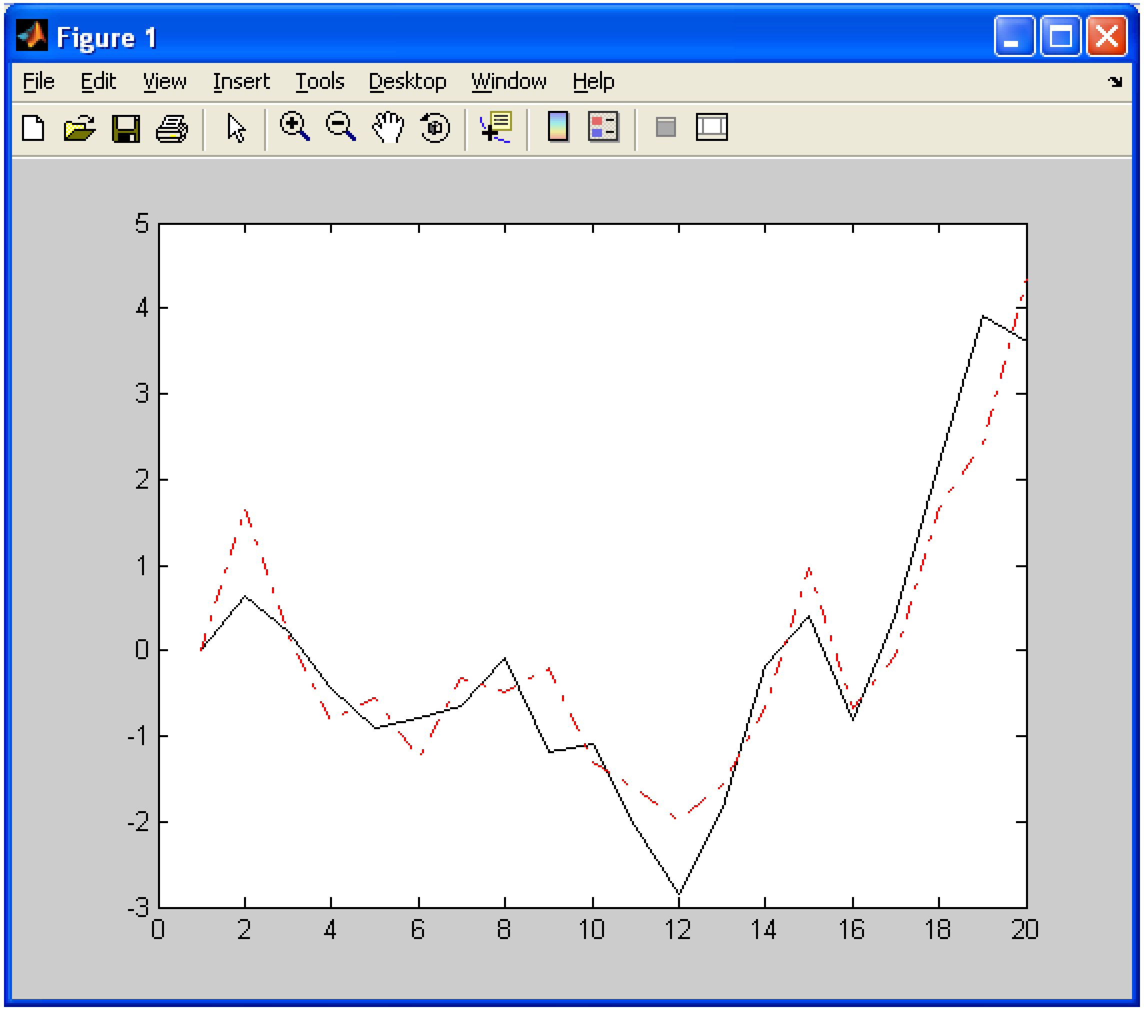


Figure 1
File Edit View Insert Tools Desktop Window Help
5
4
3
2
1
0
-1
-2
-3
0 2 4 6 8 10 12 14 16 18 20