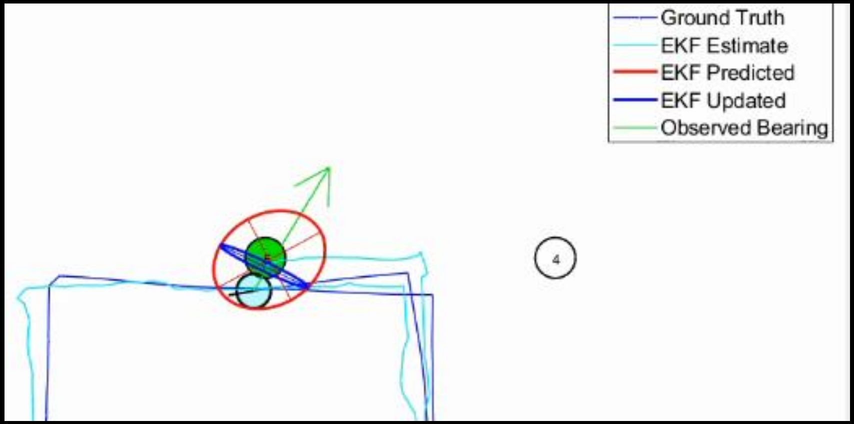
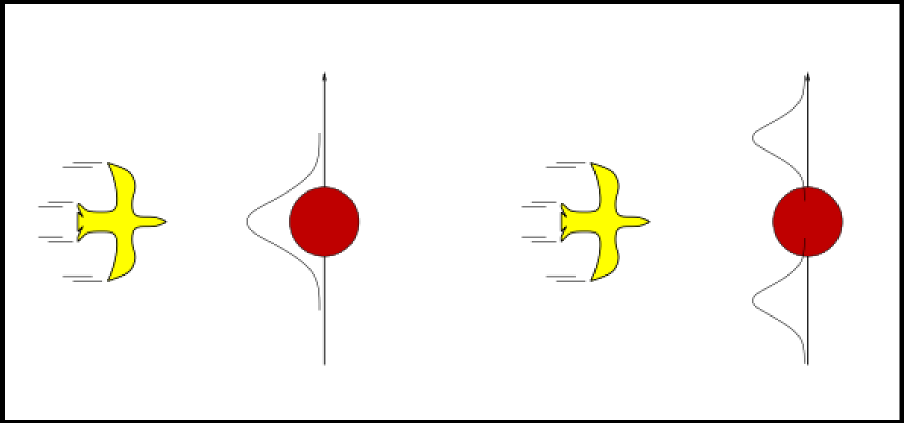
UKF and Particle Filters

Using materials from Probabilistic Robotics book and Cyrill Stachniss

Last time…

- We talked about Kalman filters:





- But even with Extended Kalman Filter (EKF), we have problems
  - Dynamics/sensors could be extremely non-linear
  - Still modeling state distribution as a Gaussian
    - What if there are two or more distinct parts of the distribution?

Outline

- KF and EKF review
- Unscented Kalman Filter
  - Handles non-linear dynamics better than EKF
- Particle filter
  - Handles arbitrary dynamics and state distributions

Review: Kalman Filter

Estimates the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

Current state

Previous state

Action

Actuation noise

with a sensor measurement

$$z_t = C_t x_t + \delta_t$$

Current measurement

Current state

Sensor noise

Kalman Filter Algorithm

Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Use dynamics (A_t and B_t) to predict what will happen

1. Prediction:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

Use sensor measurement z_t to correct prediction

2. Correction:

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

This is called the Kalman gain

$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$

$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

Return μ_t, Σ_t

Review: Nonlinear dynamic systems

- Most robotics problems involve nonlinear dynamics and sensors

$$\boxed{x_t = g(u_t, x_{t-1})}$$

$$\boxed{z_t = h(x_t)}$$

Review: The EKF trick

- Can't deal with non-linear functions directly
- But, if the change is small, we can use a *local linear approximation*
- How? Compute the Jacobians of *g* and *h*!

$$x_t = g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}(x_{t-1} - \mu_{t-1})$$

$$x_t = g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1})$$

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$$

$$z_t = h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_t - \bar{\mu}_t)$$

$$z_t = h(x_t) \approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t)$$

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$$

Algorithm **Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

1. Prediction:

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$$

2. Correction:

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$$

$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$$

Return μ_t, Σ_t

Kalman Filter

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$$

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

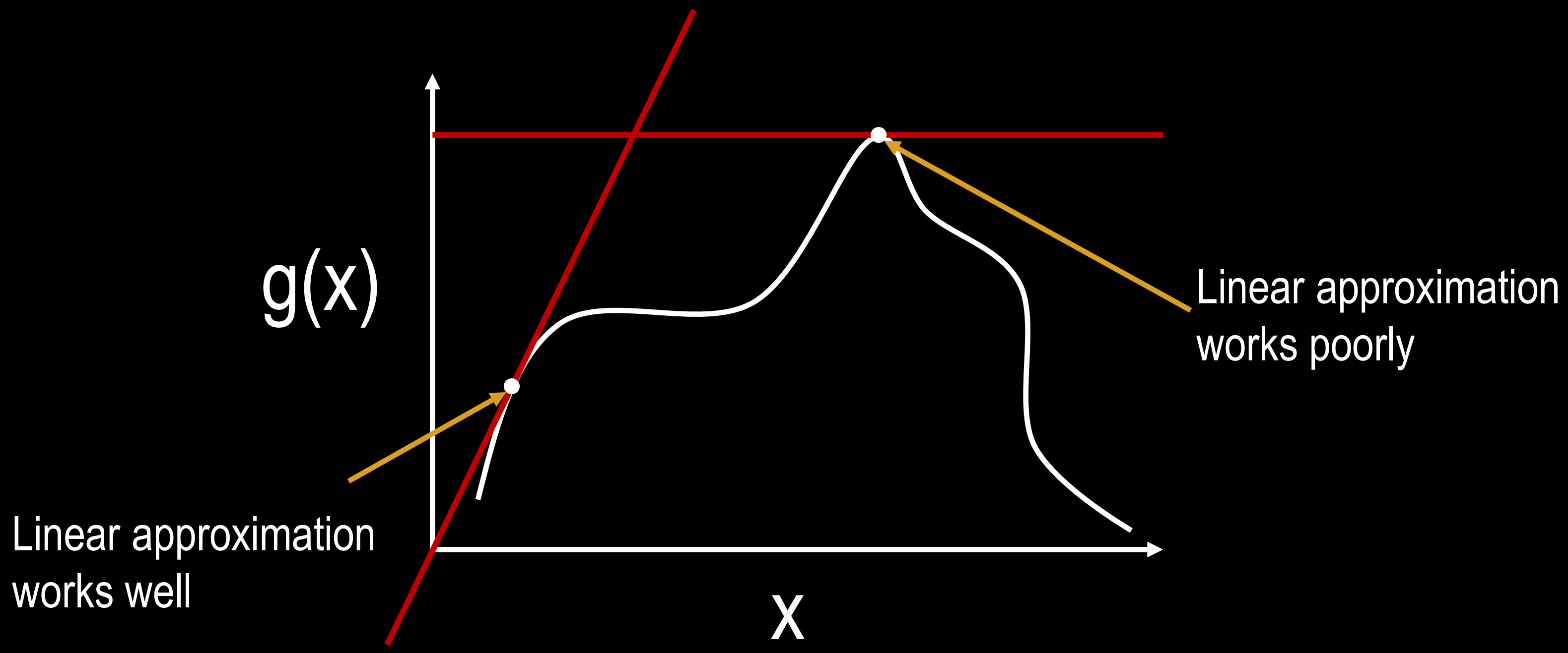
$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$$

$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$$

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t} \qquad G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$$

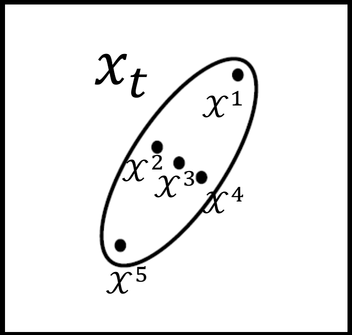
The problem with EKF

- Works well when function is close to linear
- Doesn't work well when the function is highly non-linear

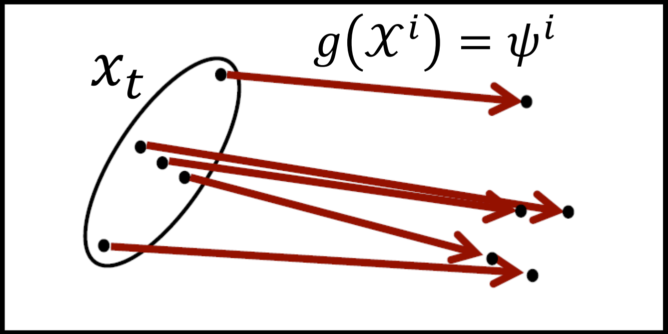


The Unscented Transform

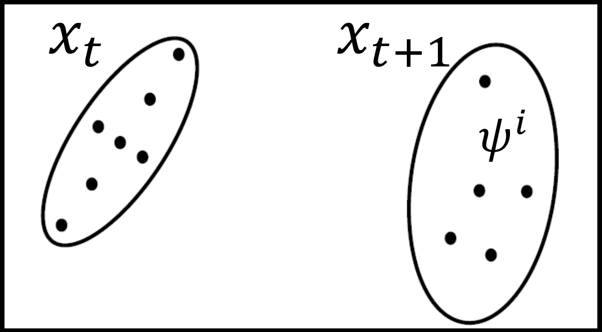
- Key idea:
  1. Sample a set of *sigma points* from the Gaussian distribution
  2. Pass sigma points through the function
  3. Re-estimate Gaussian



Step 1



Step 2

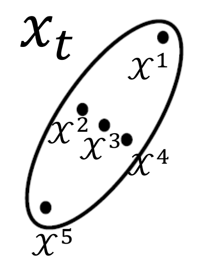


Step 3

Sigma Points

- Computing the sigma points

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



Step 1

matrix square root

Extracts the *(i-n)*th column from the matrix square root

dimensionality

scaling parameter $\lambda = \alpha^2(n+\kappa) - n$

- Each sigma point is assigned two weights:

$$
w_m^0 = \frac{\lambda}{n+\lambda} \qquad w_c^0 = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)
$$

$$
w_m^i = w_c^i = \frac{1}{2(n+\lambda)} \qquad \text{for } i = 1,\ldots,2n
$$

<u>Typical parameter values:</u>

$\alpha = 10^{-3}$

$\kappa = 0$

$\beta = 2$

Using Sigma Points

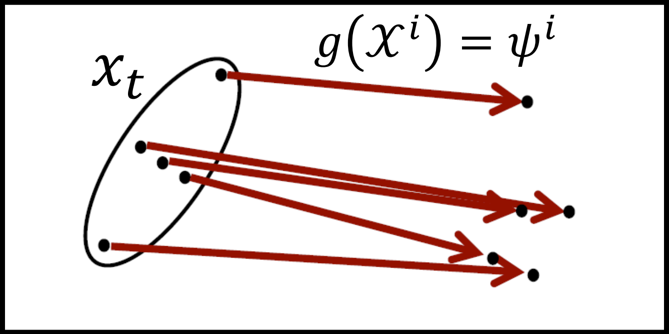
- Pass sigma points through nonlinear function

$$\psi^i = g(\chi^i)$$

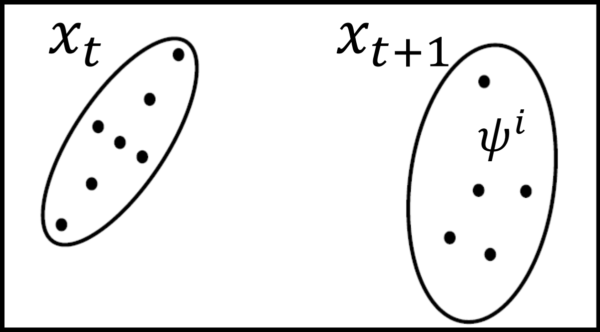
- Recover mean and covariance

$$\mu' = \sum_{i=0}^{2n} w_m^i \psi^i$$

$$\Sigma' = \sum_{i=0}^{2n} w_c^i (\psi^i - \mu)(\psi^i - \mu)^T$$

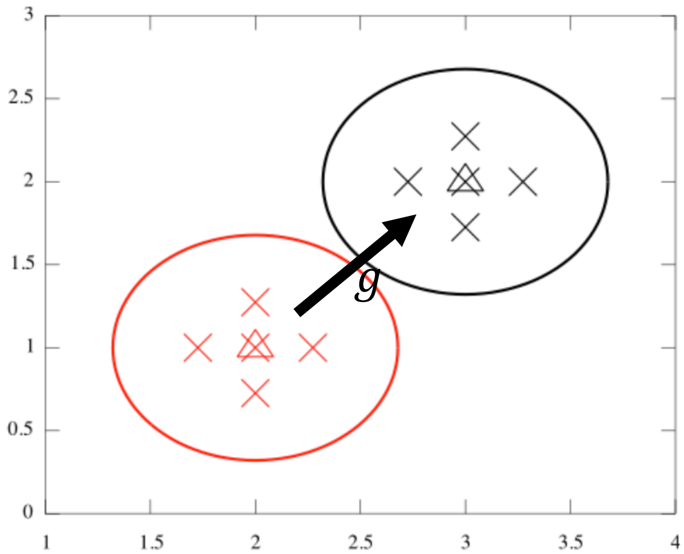


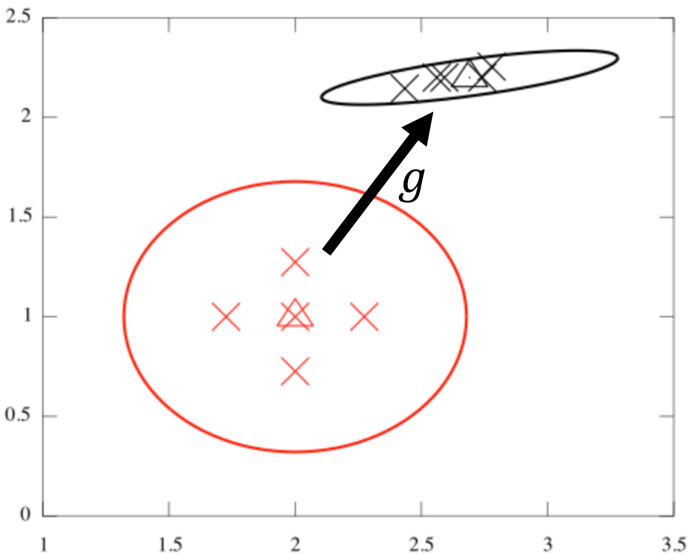
Step 2



Step 3

Sigma Points Examples





$$g((x,y)^T) = \begin{pmatrix} x+1 \\ y+1 \end{pmatrix}^T$$

$$g((x,y)^T) = \begin{pmatrix} 1+x+\sin(2x)+\cos(y) \\ 2+0.2y \end{pmatrix}^T$$

Unscented_Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

$\mathcal{X}_{t-1} =$ Compute sigma points using μ_{t-1} and Σ_{t-1}

$\bar{\mathcal{X}}_t^* = g(u_t, \mathcal{X}_{t-1})$

$$\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}$$

Process noise

$$\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^T + R_t$$

Prediction

$\bar{\mathcal{X}}_t =$ Compute sigma points using $\bar{\mu}_t$ and $\bar{\Sigma}_t$

$\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t)$

$$\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]}$$

Sensor noise

$$S_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)^T + Q_t$$

$$\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t)(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)^T$$

$K_t = \bar{\Sigma}_t^{x,z} \; S_t^{-1}$

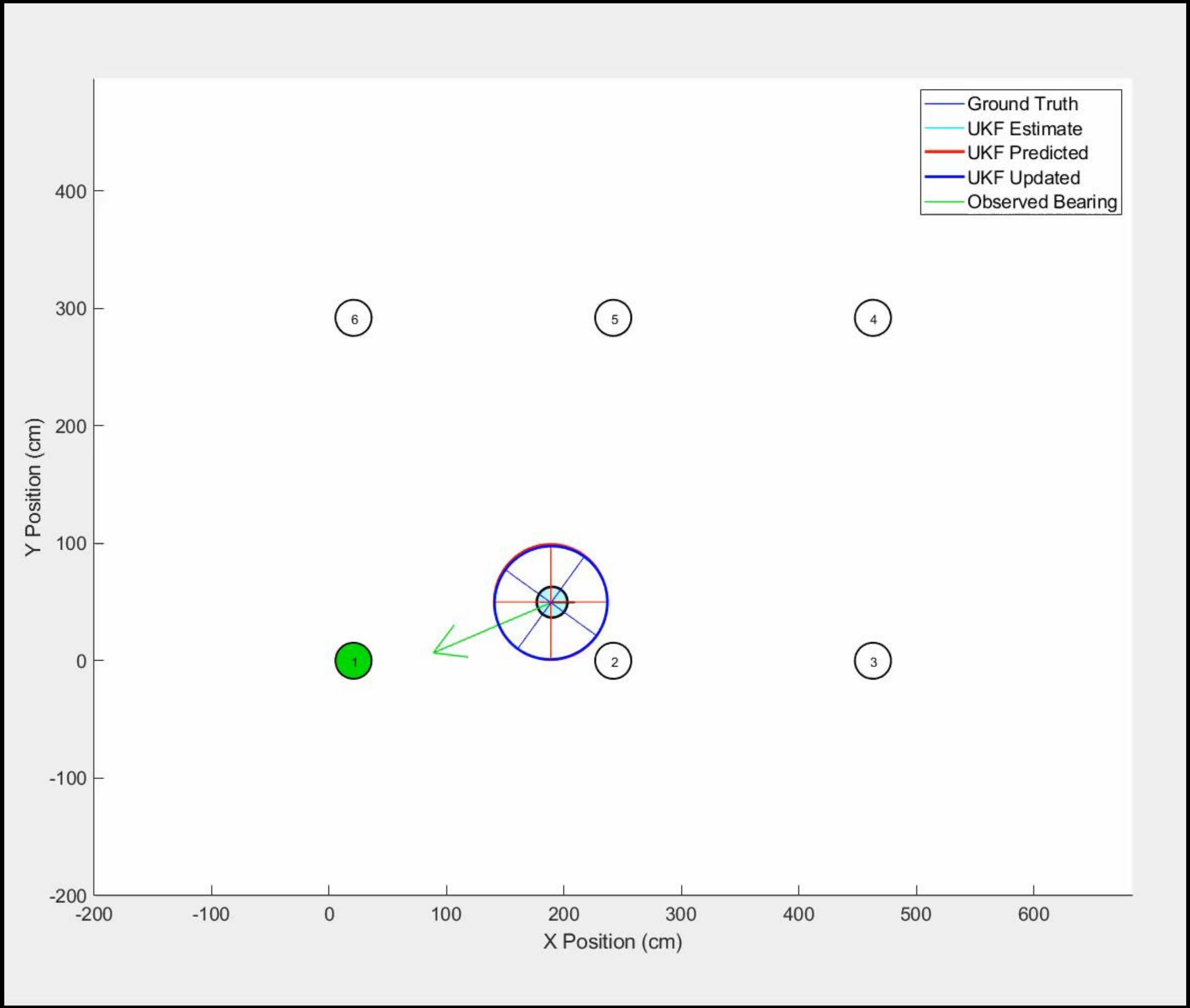
$\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$

$\Sigma_t = \bar{\Sigma}_t - K_t \; S_t \; K_t^T$

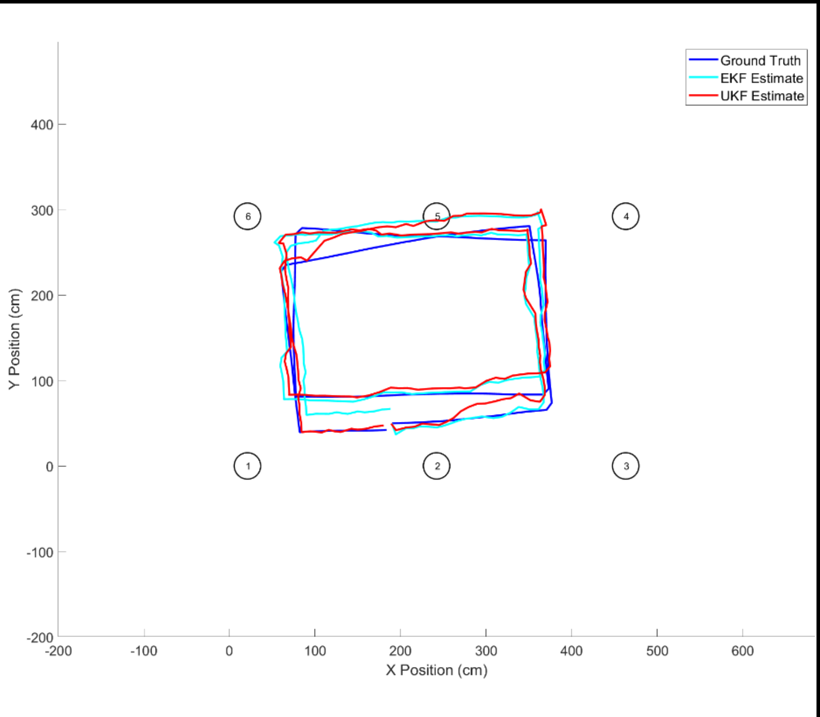
return μ_t, Σ_t

Correction

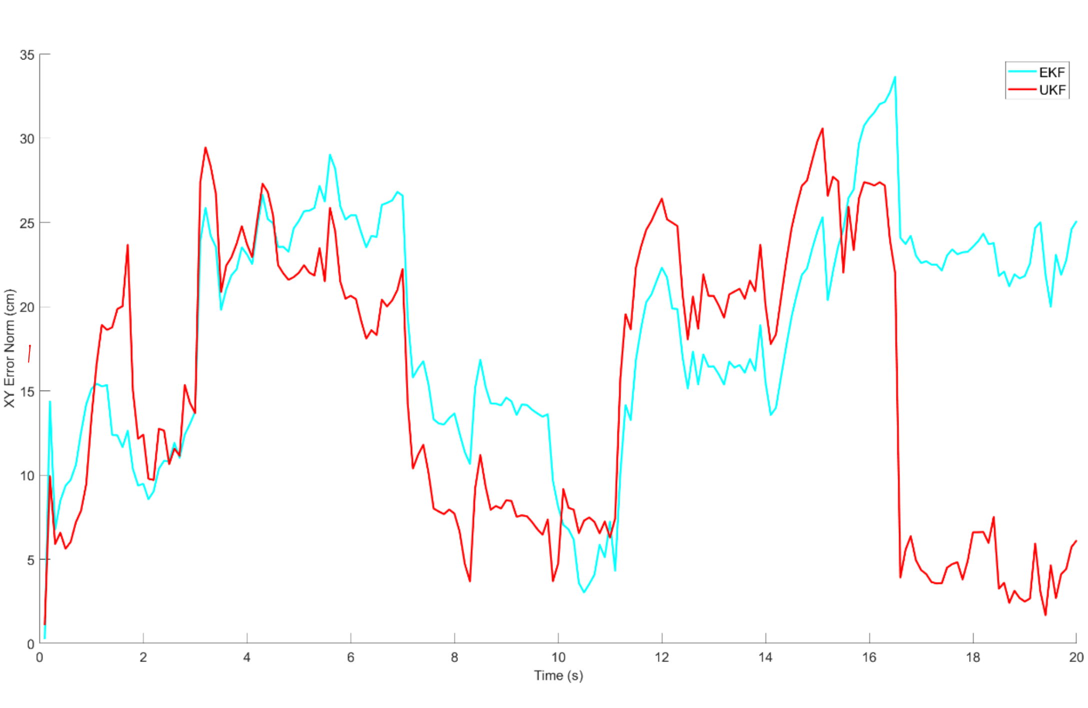
UKF Example: Localization with Landmarks



EKF vs. UKF



Ground Truth
EKF Estimate
UKF Estimate
Y Position (cm)
X Position (cm)
400
300
200
100
0
-100
-200
-200
-100
0
100
200
300
400
500
600
1
2
3
4
5
6



EKF
UKF
XY Error Norm (cm)
Time (s)
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12
14
16
18
20

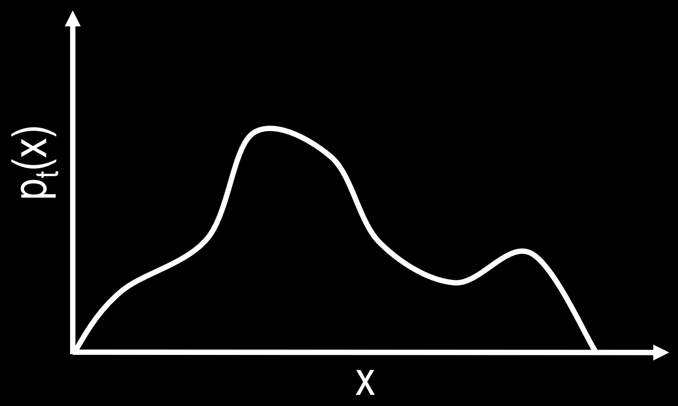
UKF Summary

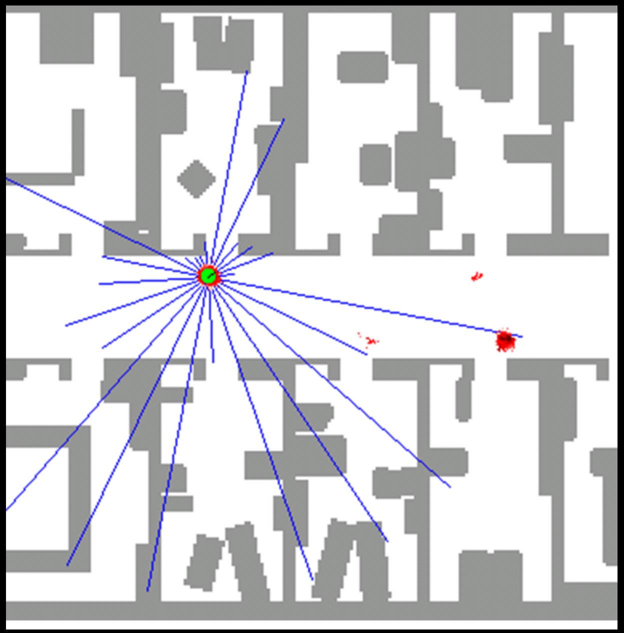
- Highly efficient: Same complexity as EKF, with a constant factor slower in typical practical applications
- Better approximation than EKF: Accurate in first two terms of Taylor expansion (EKF only first term)
- Derivative-free: No Jacobians needed
- Still not optimal!

Particle Filters

Particle Filters

- Kalman filter assumes all error is Gaussian
- Need a way to handle *arbitrary* probability distributions
- Distribution changes as you get new sensor data
  - E.g. as robot moves/senses
- Particle Filters main idea:
  - *sample* from the implicit distribution of the state at any given time
    - Each sample is called a *particle*
  - no need to make assumptions about distributions





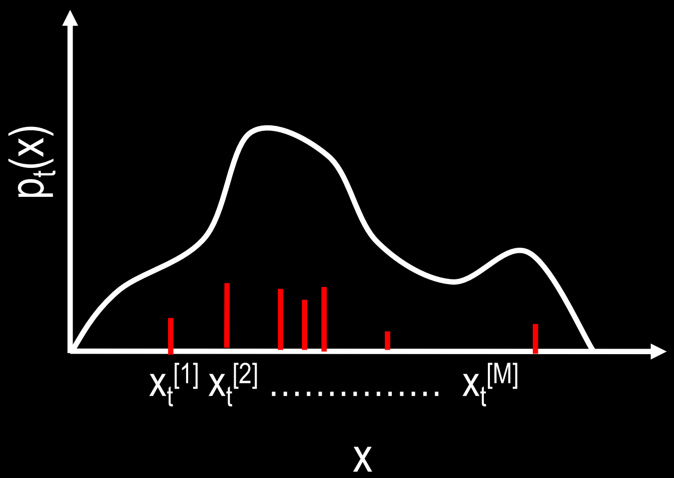
Representing a Distribution

- At any time t, distribution is represented by
  - M samples of the robot's state

    $$X_t = x_t^{[1]}, x_t^{[2]}, \ldots , x_t^{[M]}$$

  - Each sample has an associated weight

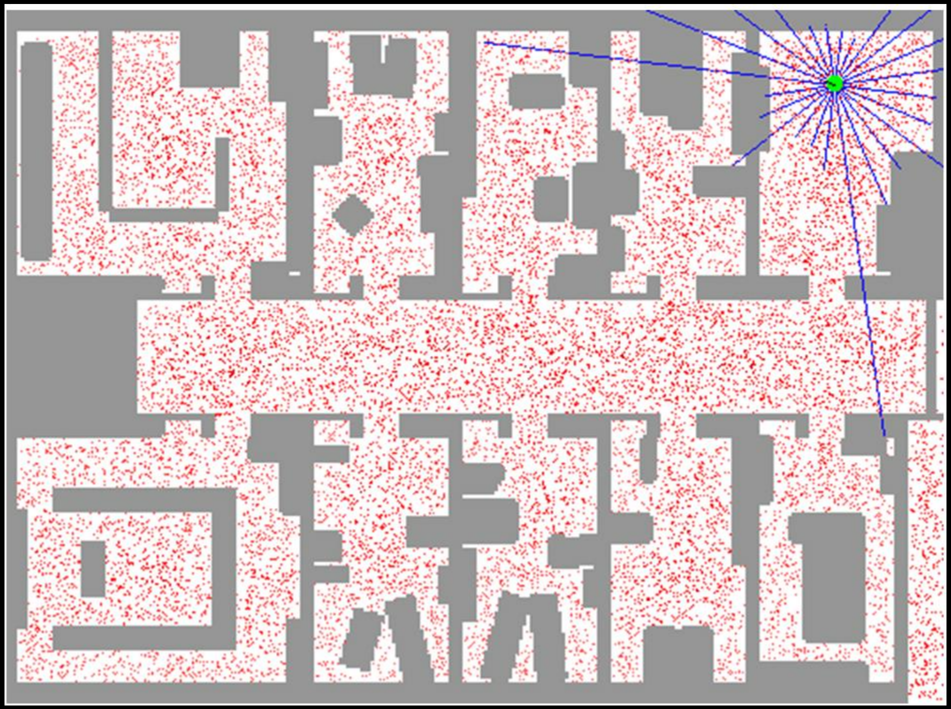
    $$W_t = w_t^{[1]}, w_t^{[2]}, \ldots , w_t^{[M]}$$



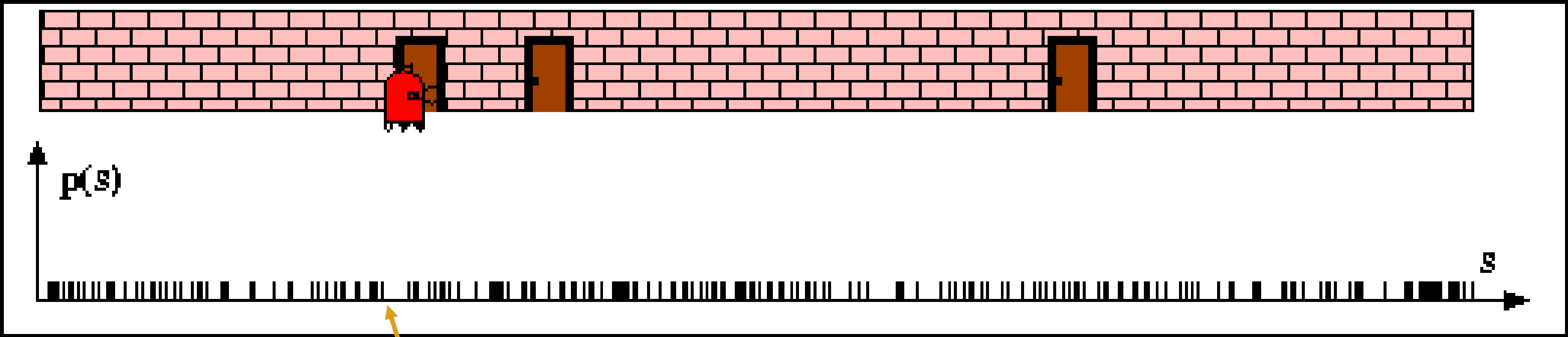
Heights of lines are the weights

Initialization

- If you roughly know the starting location, cluster particles around the start
- If starting location unknown, scatter particles through the environment

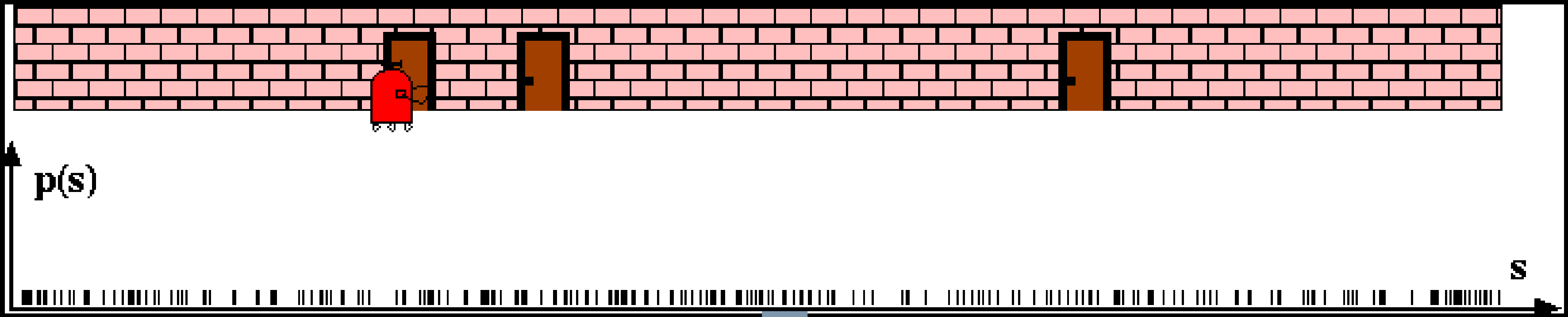


Particle Filter Example

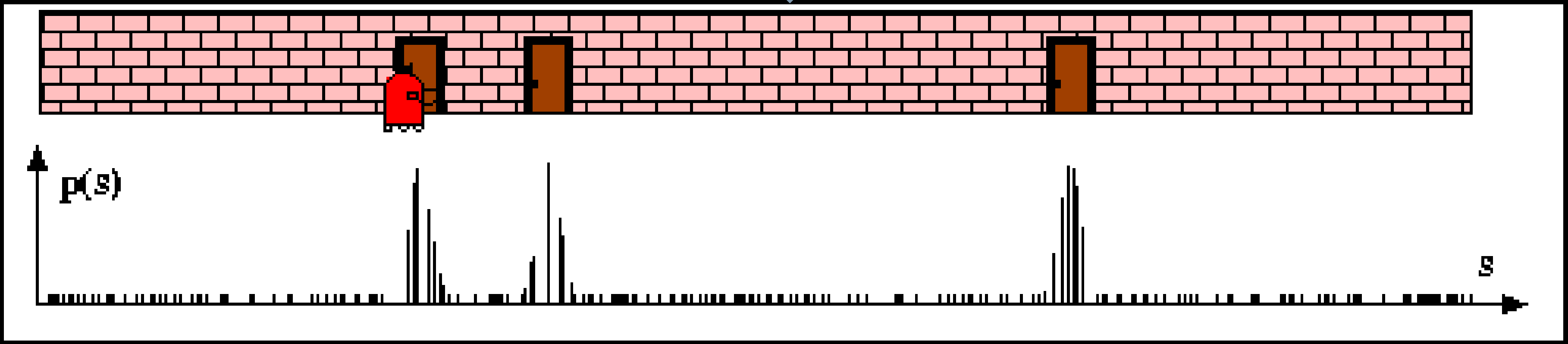


Each line is a particle
The height of a line is how likely it is to be the true position

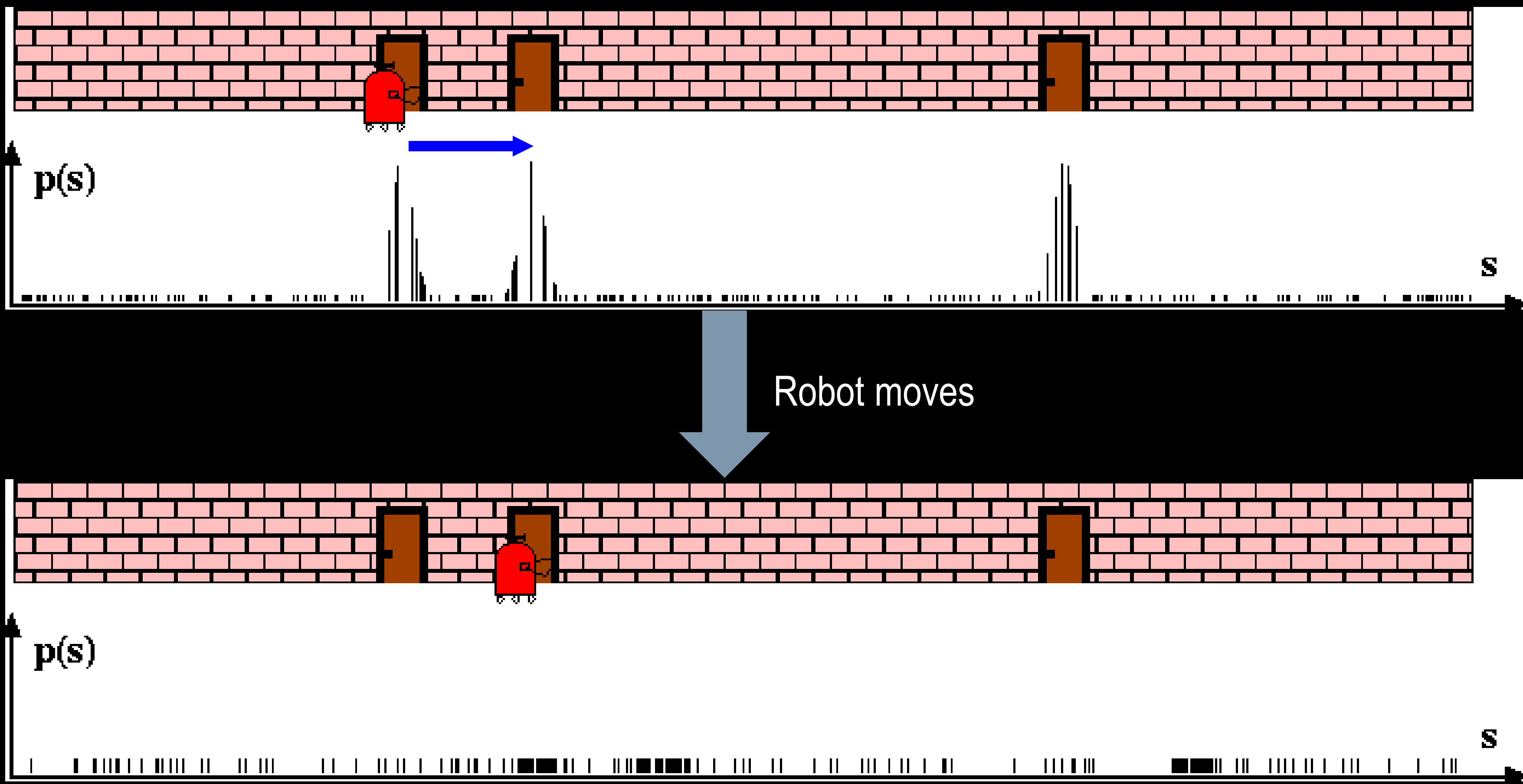
Particle Filter Example



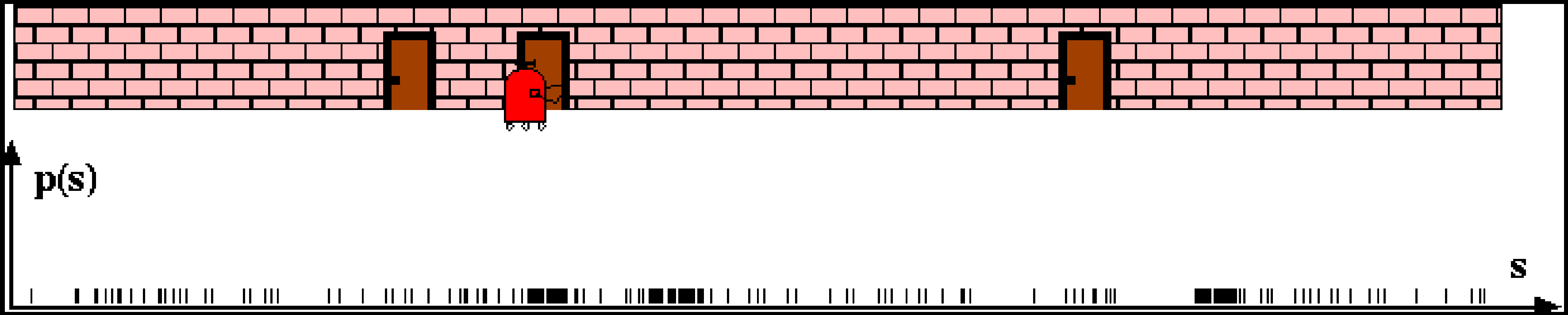
Get Sensor Data



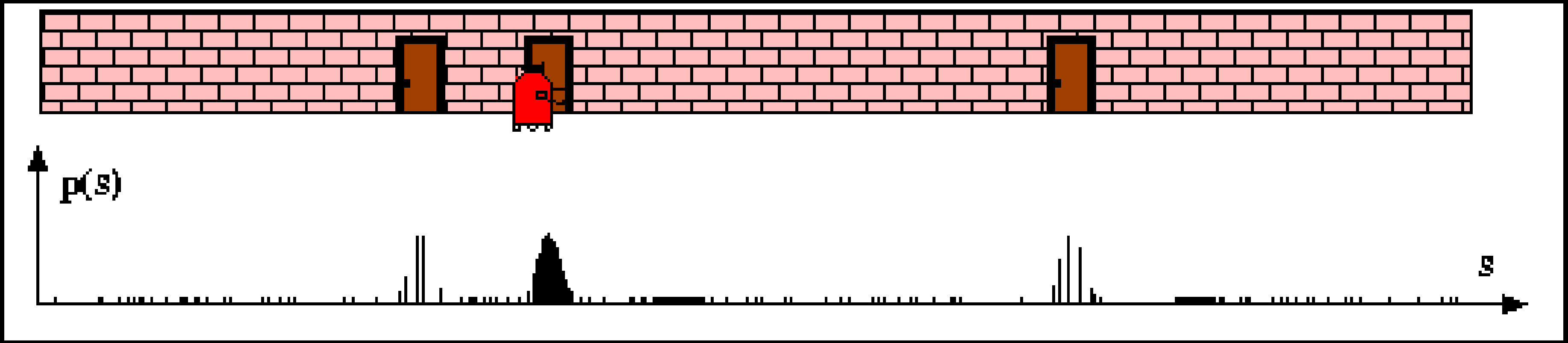
Particle Filter Example



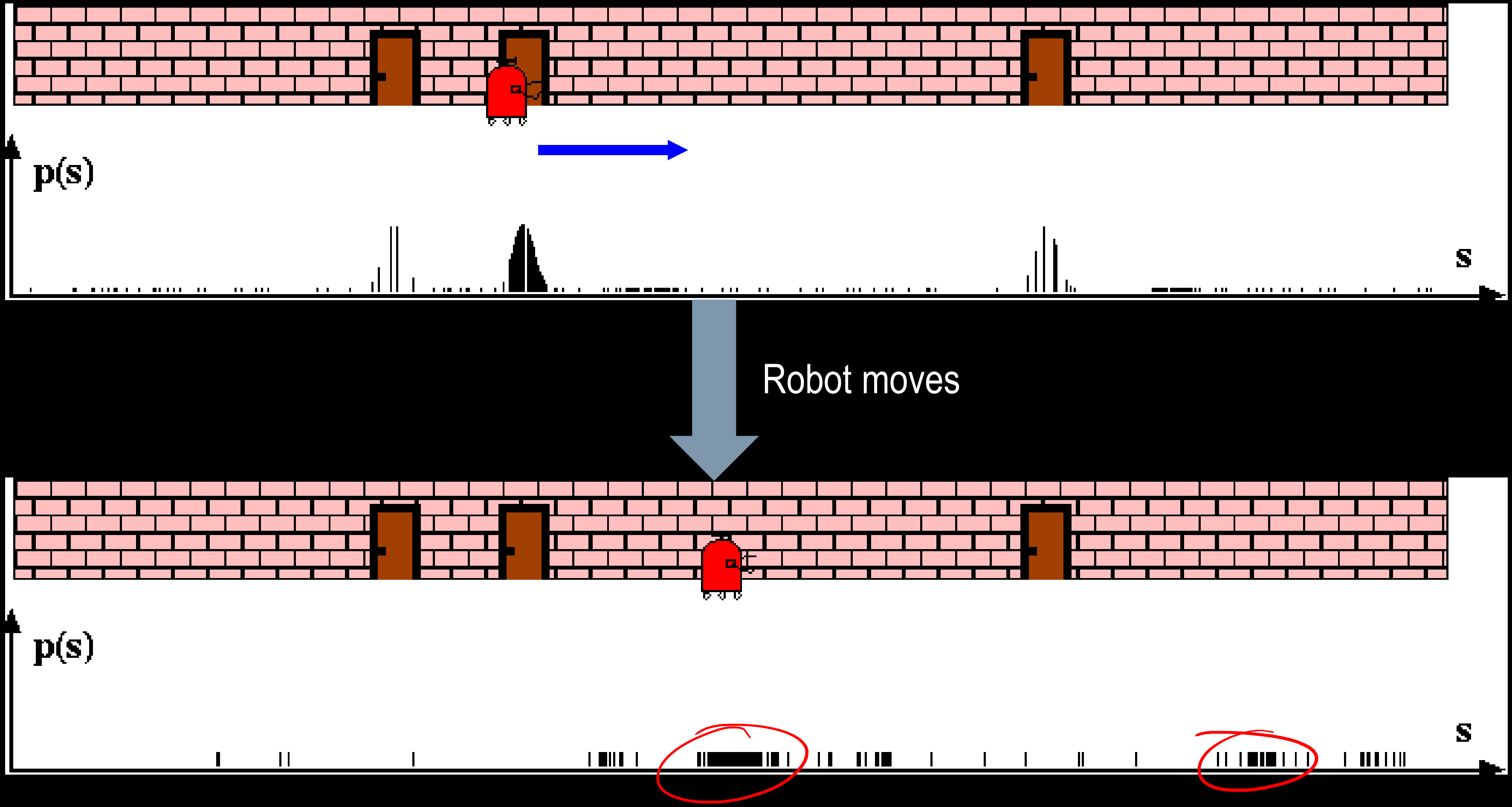
Particle Filter Example



Get Sensor Data



Particle Filter Example



Particle Filter Algorithm

$\mathbf{X}_0$ = Sample M times from $P(X_0)$
t = 0
While(1)
    t++;
    u_t = Action()
    z_t = Sensor()
    $\mathbf{S}_t = \mathbf{X}_t$ ={}
    for m = 1 to M
        sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]}$ from $\mathbf{X}_{t-1})$
        $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
        $\mathbf{S}_t = \mathbf{S}_t \cup (x_t^{[m]}, w_t^{[m]})$
    endfor

    for m = 1 to M
        draw i with probability $\propto w_t^{[i]}$
        add $x_t^{[i]}$ from $\mathbf{S}_t$ to $\mathbf{X}_t$
    endfor
endwhile

e_t

initialize

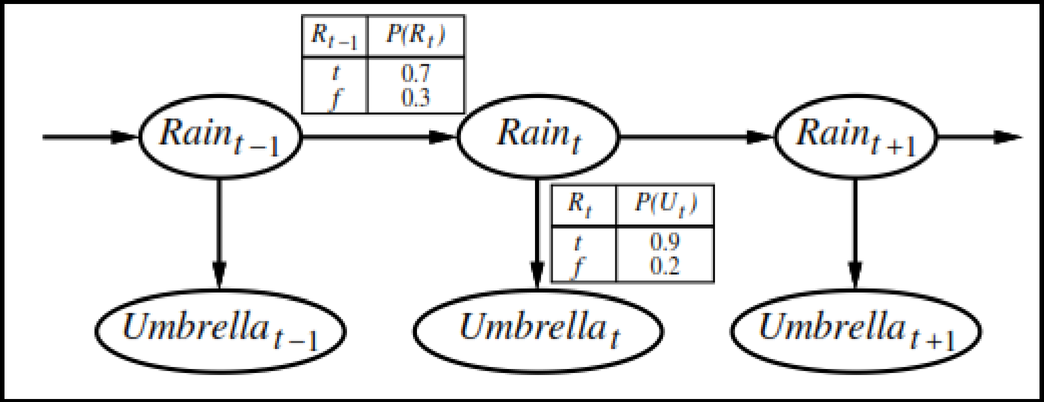
get new information

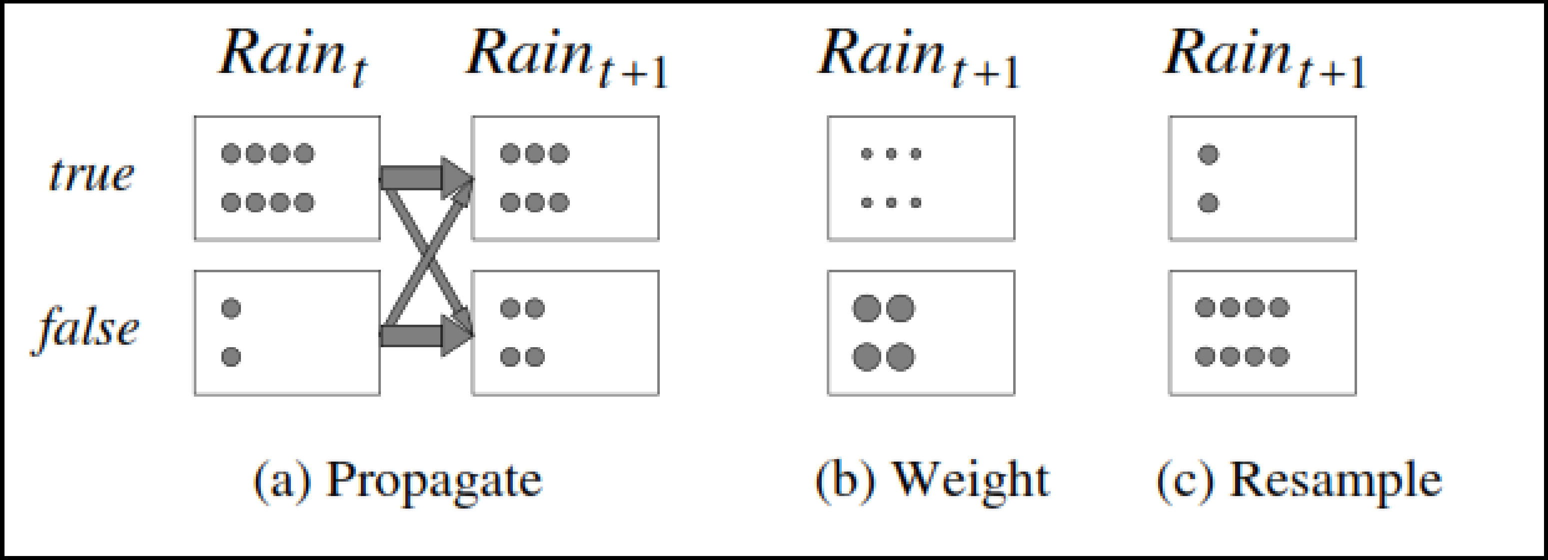
Apply u_t to particles
(if u_t is certain, no sampling necessary)

$w_t^{[m]}$ is how well sample m “explains” the sensor data z_t

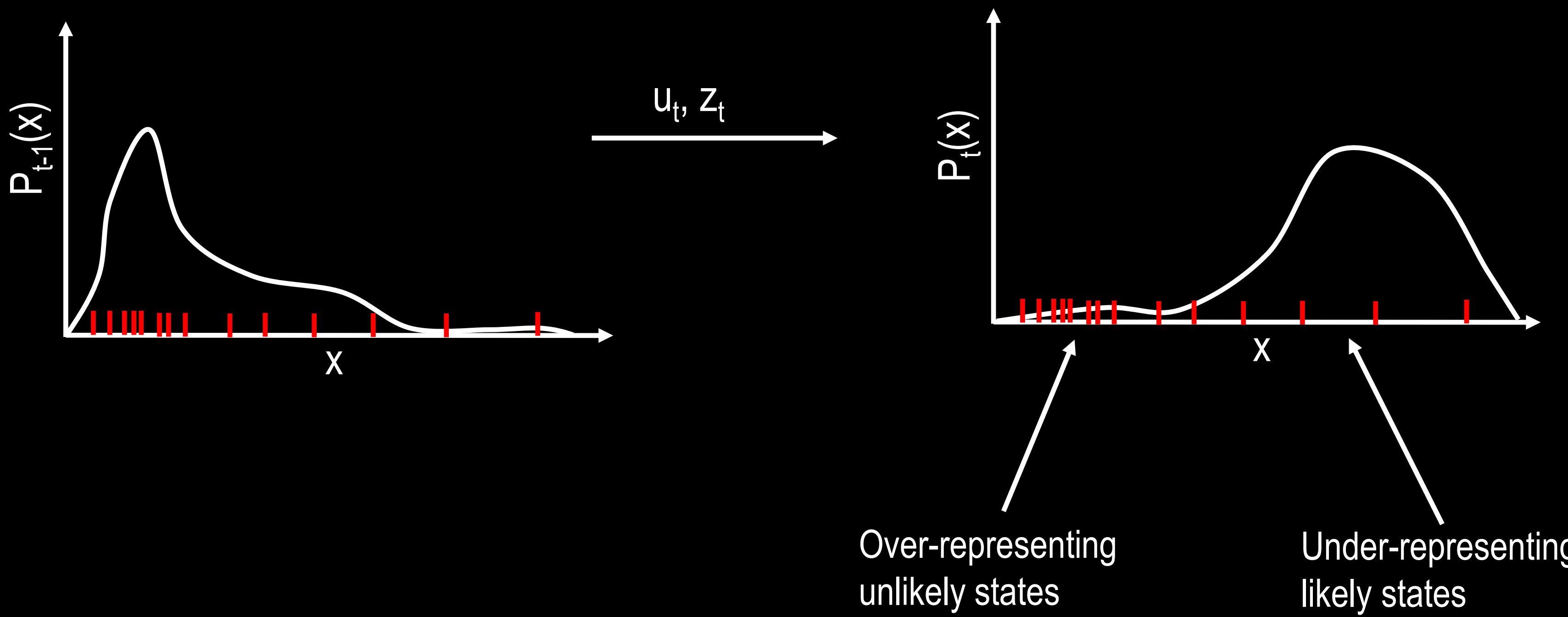
Re-sample particles
(Importance sampling using $w_t^{[i]}$)

Particle Filter Example: Discrete State Space

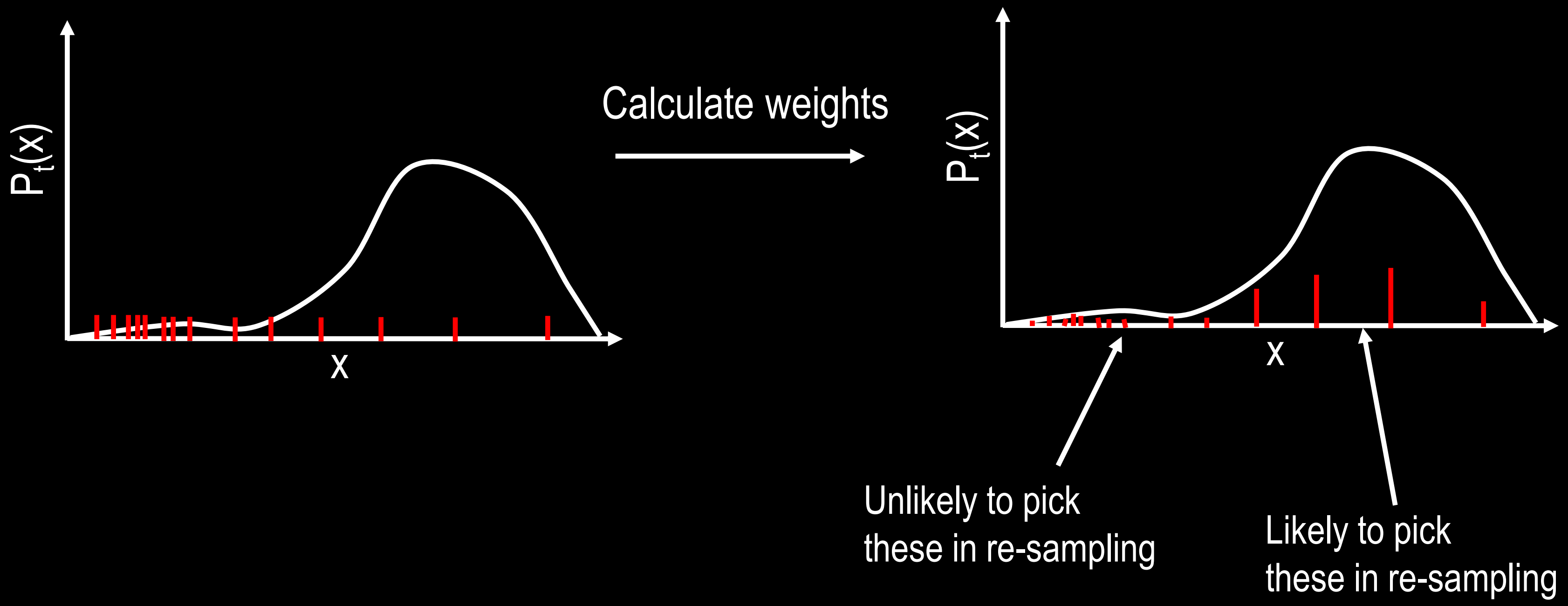




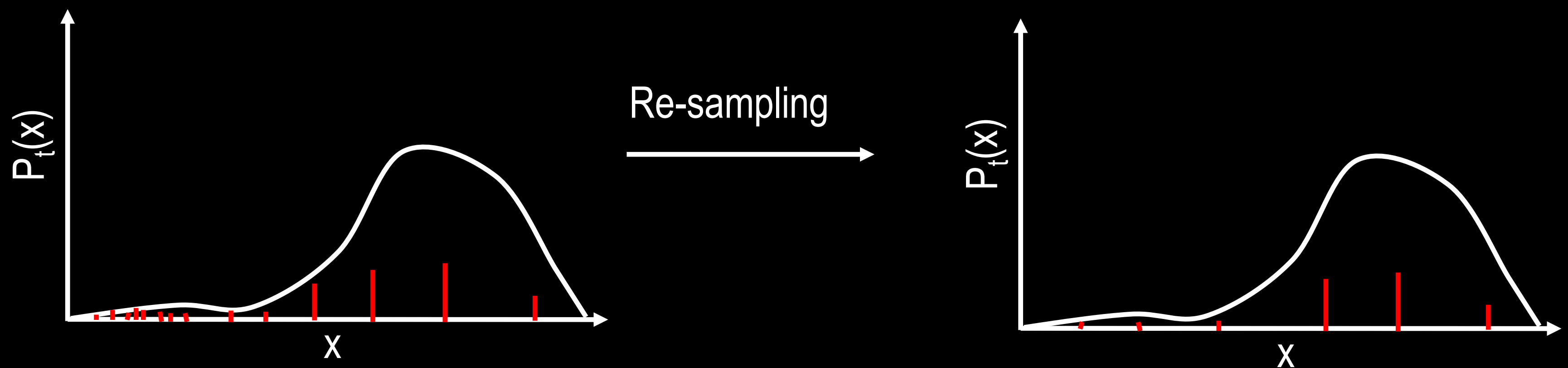
Why Re-sample?



Why Re-sample?



Why Re-sample?



for m = 1 to M

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

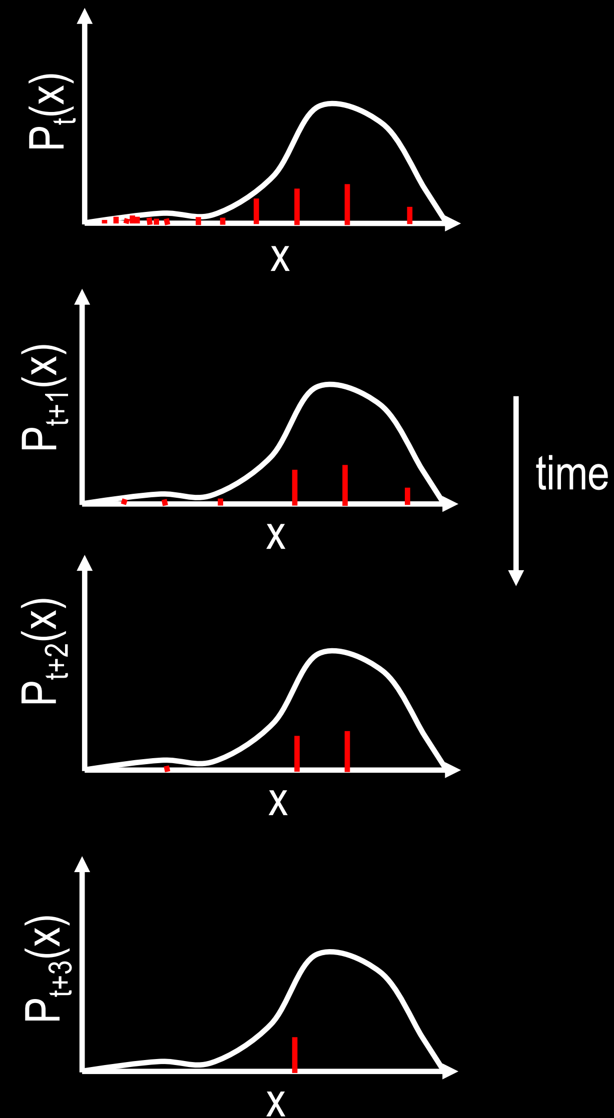
  add $x_t^{[i]}$ to X_t

endfor

Re-sampling Issues

- Sampling with replacement
  - Can have multiple copies of the same particle
- Sacrifices diversity of particles
  - Low-weight particles are likely to be destroyed
  - No way to regain lost particles
- Re-sampling occurs at ever time step, even if the agent is not moving!
  - If the robot stops, the filter will converge very quickly to the wrong answer (usually)

Re-sampling with no new measurements

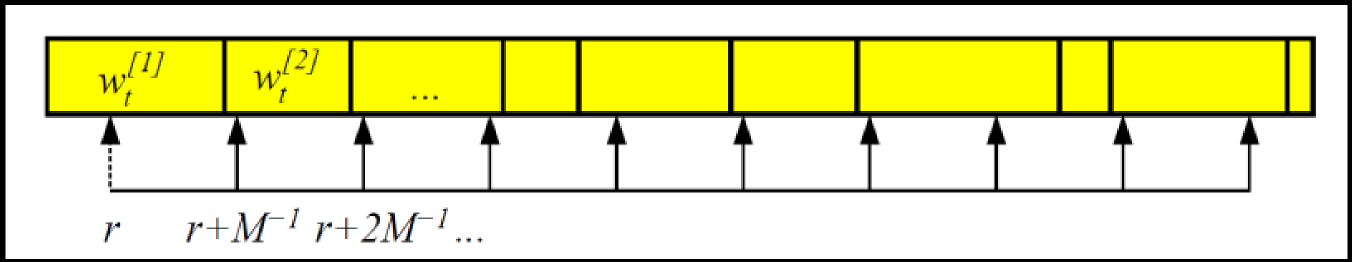


Modified Re-sampling

- Change when to re-sample
  - Never re-sample when the robot is stopped
  - Re-sample only when *n* new measurements have been taken

- Low-variance sampling
  - Choose a random number *r* and select those particles that correspond to

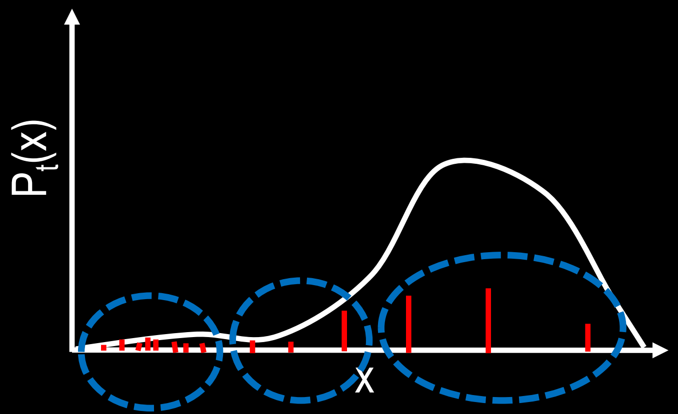
$$r + (m - 1)M^{-1} \text{ where } m = 1,\ldots,M \text{ and } r \in [0, \frac{1}{M}]$$



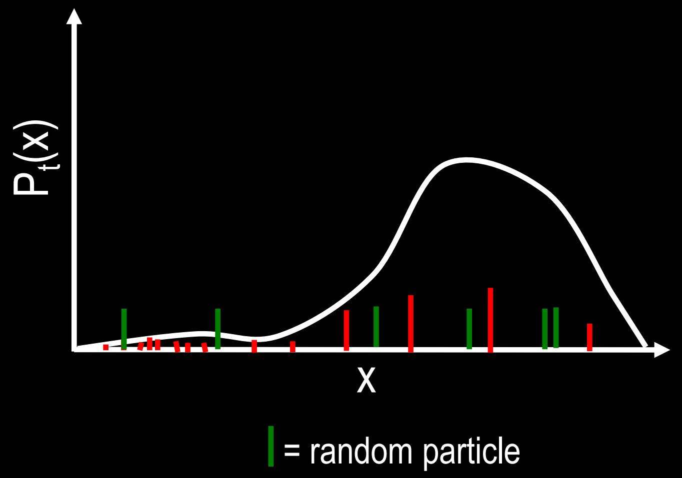
  - More systematic than independent random sample
  - If all samples have the same importance ($w_t^{[m]}$), all samples will be preserved

Modified Resampling

- Stratified sampling
  - Group particles into subsets
  - Number of samples from each subset is proportional to the number of particles in the set
  - Sample randomly within each subset
  - Good for tracking multi-modal distributions



- Preserving diversity
  - Inject random particles into re-sampling step



How many particles to use?

- The more particles, the more computational expense

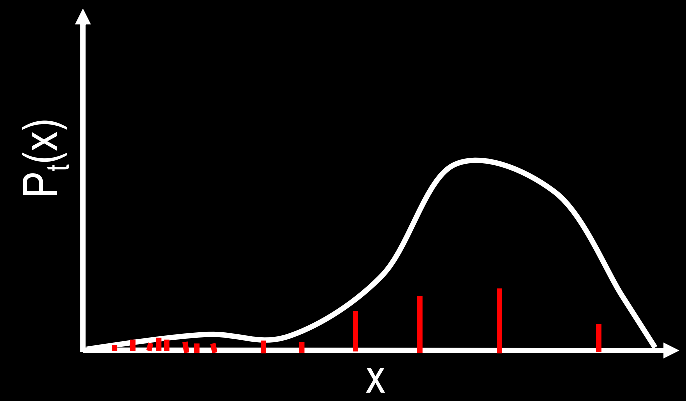
- But as M goes to infinity, the prominence of sampling issues decreases

Use as many particles as possible!

So what is the most likely state?

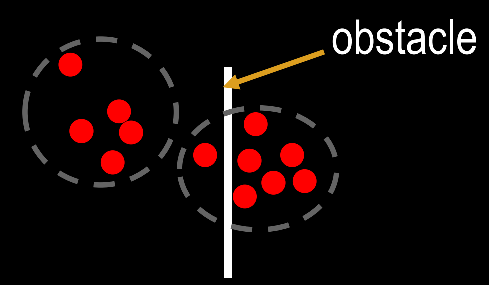
- Many ways to extract “most likely” state from particles

1. Use most likely particle
   - Ignoring much of the distribution



2. Weighted average of all particles
   - Won’t work for multi-modal distributions

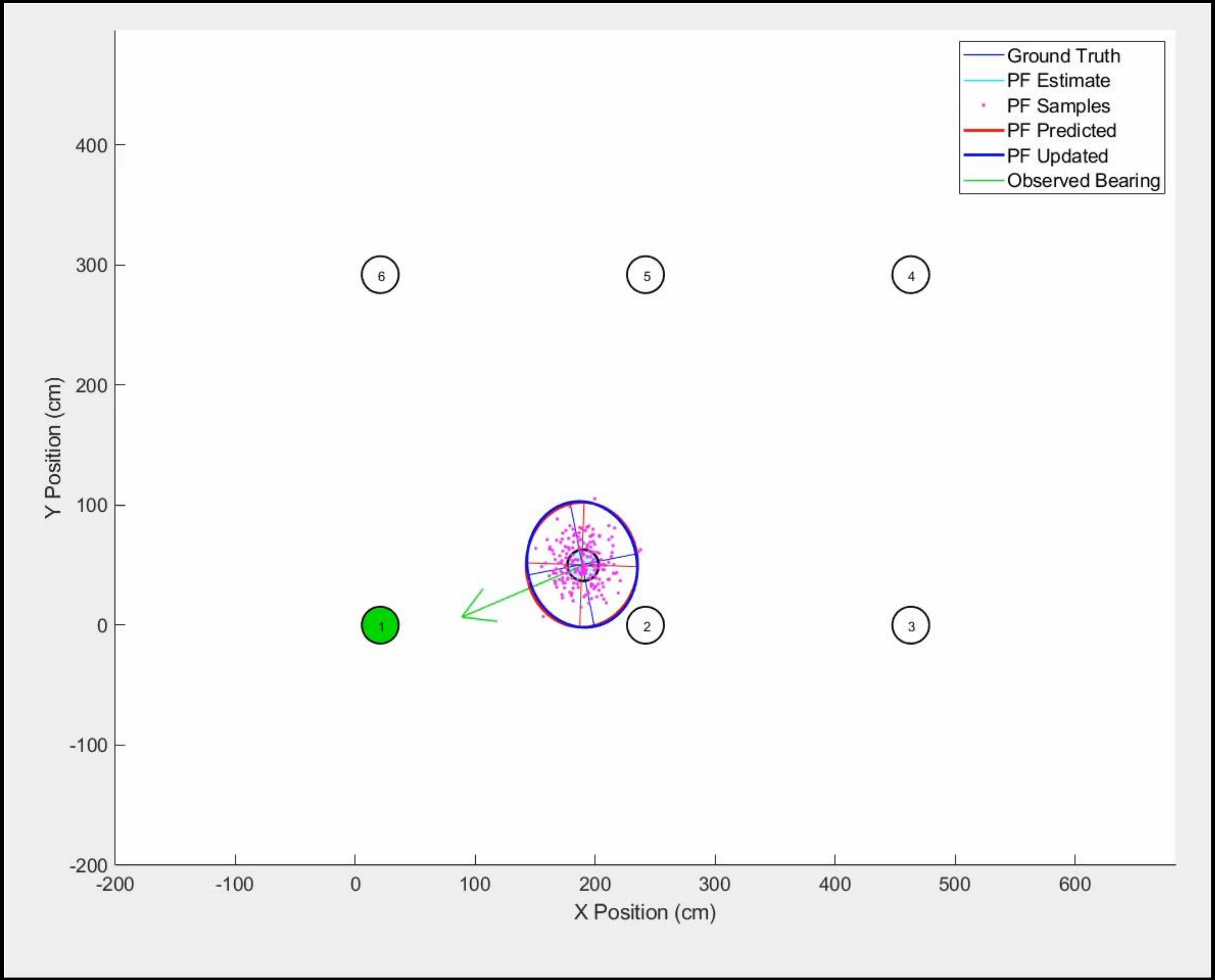
3. Cluster, then take center of largest cluster
   - What is the distance metric for clustering?



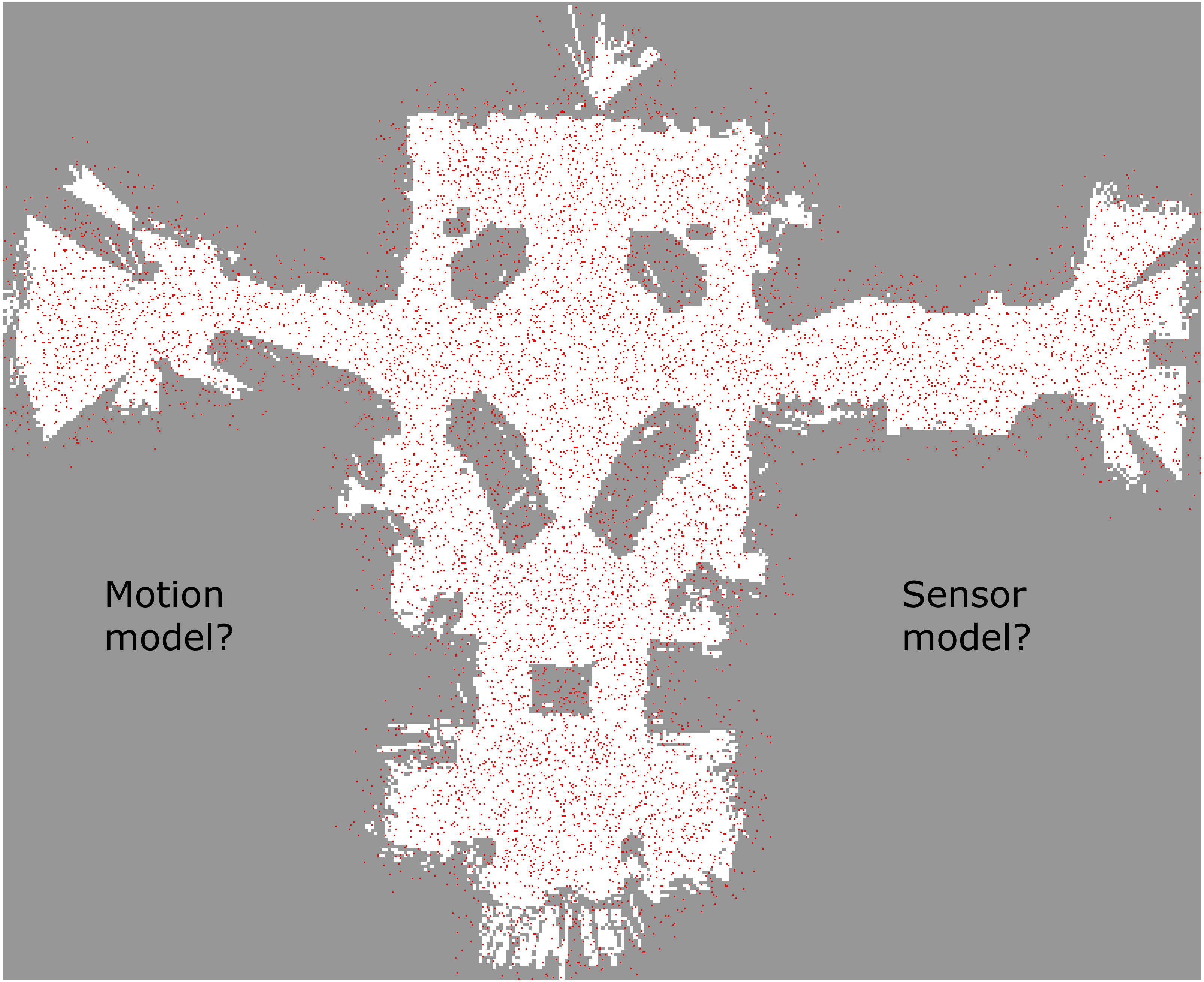
Clustering only on Euclidean distance may not make sense

BREAK

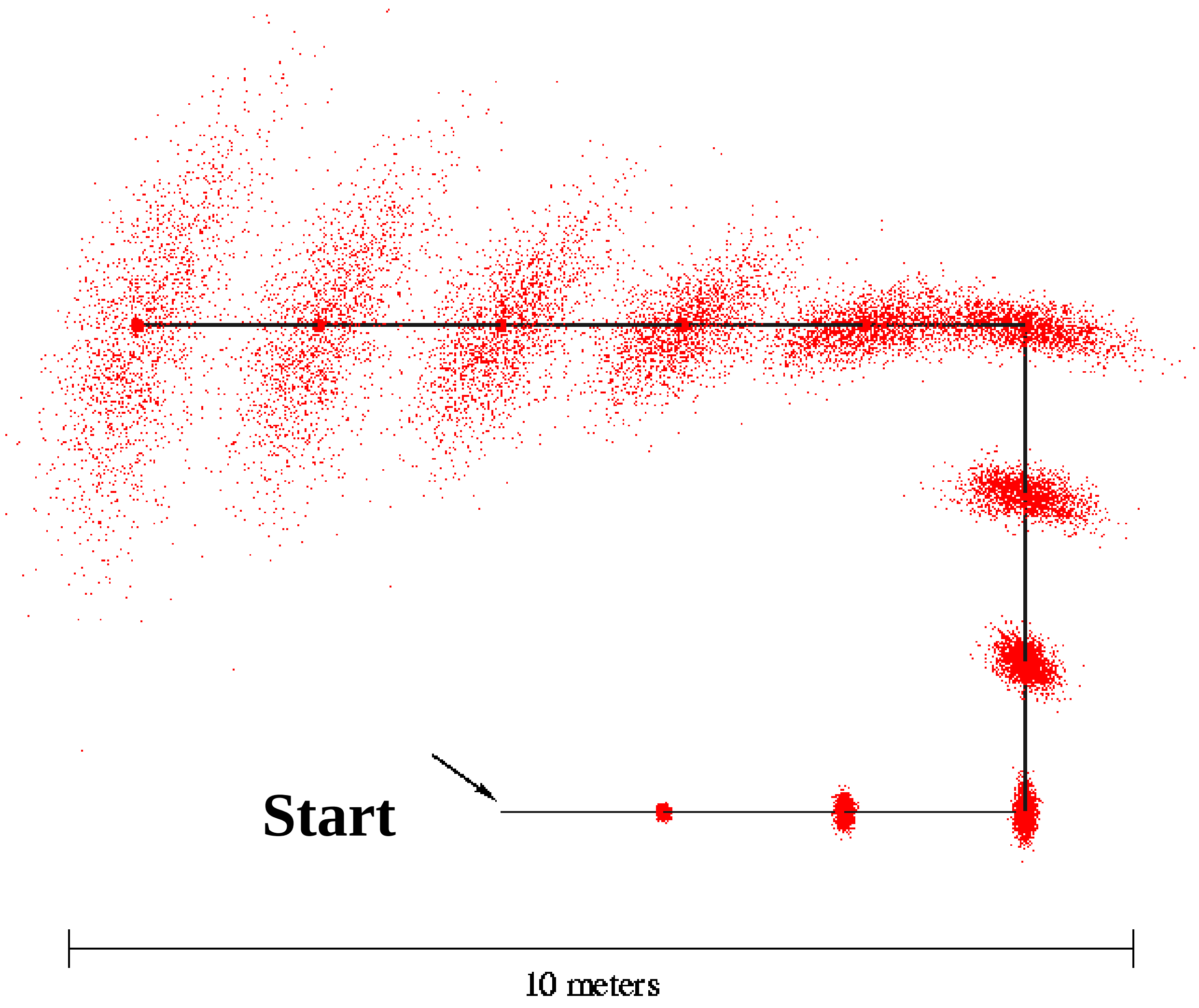
Particle Filter Example: Localization with Landmarks



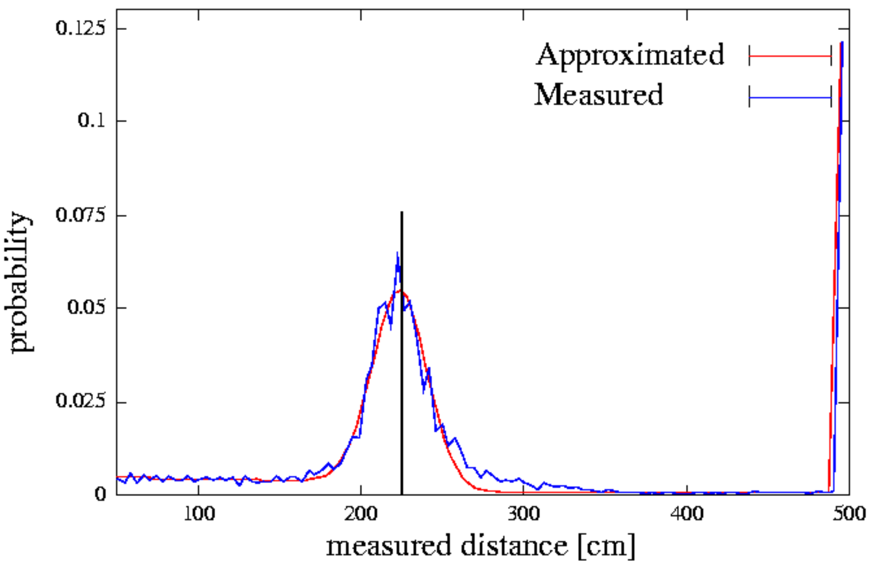
Particle Filter for Robot Localization



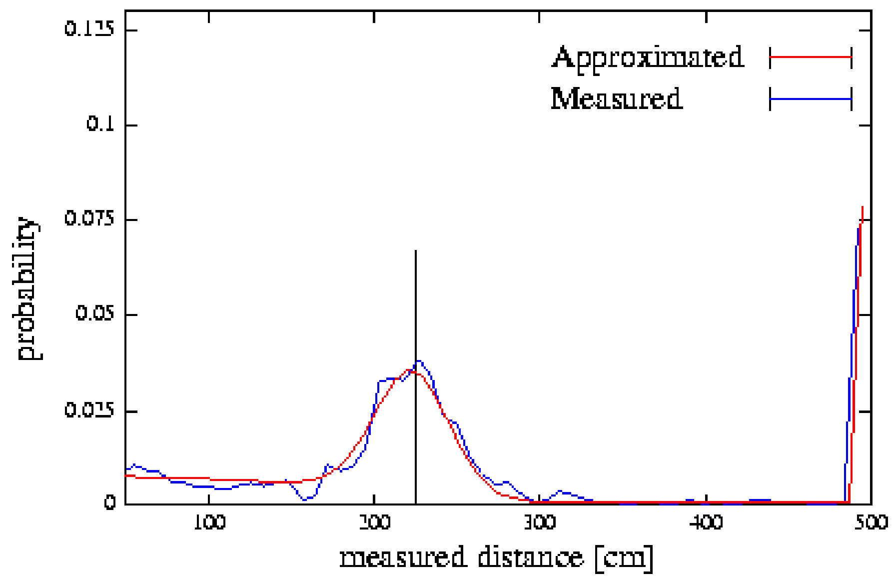
Example Motion Model



Example Proximity Sensor Model

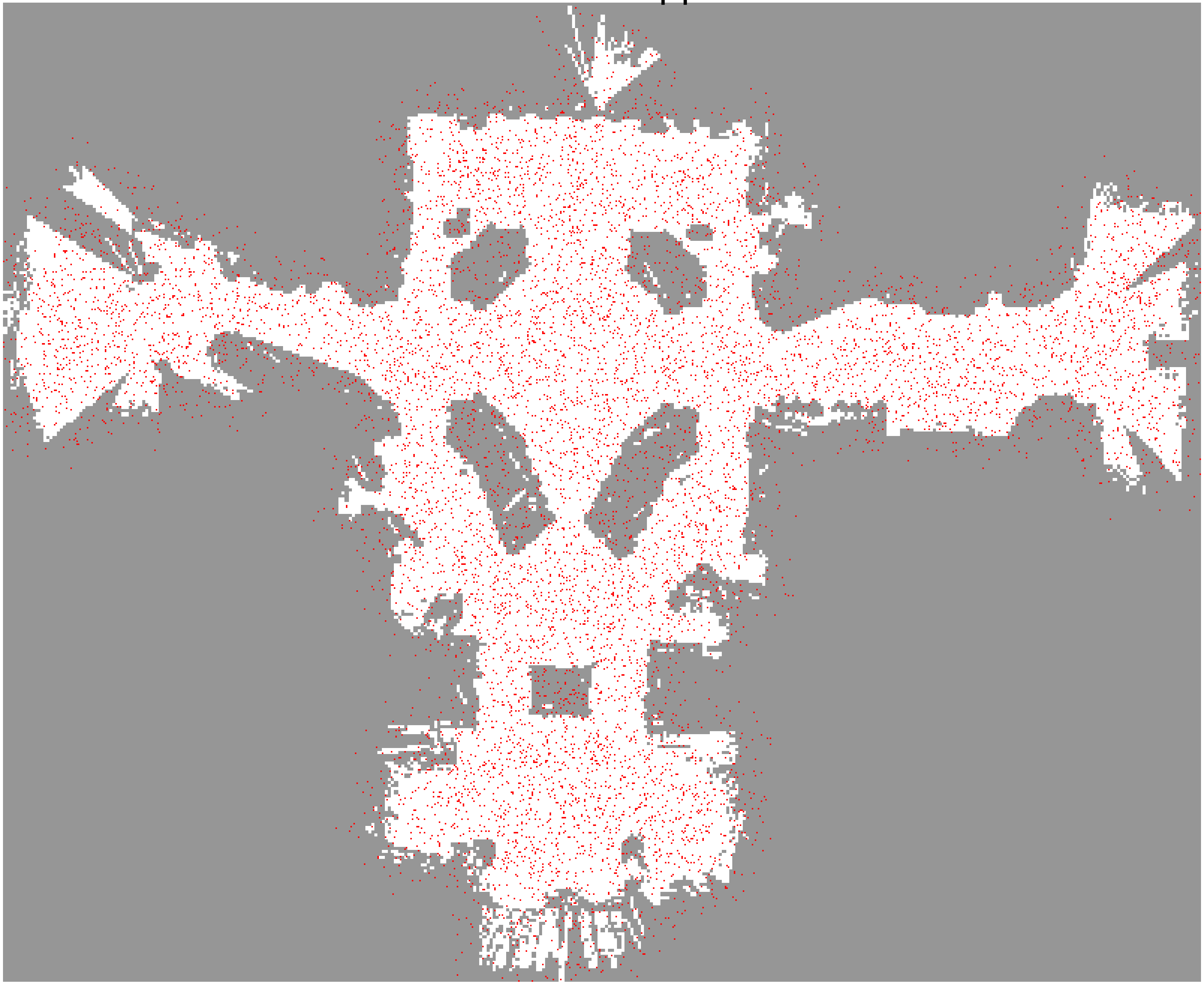


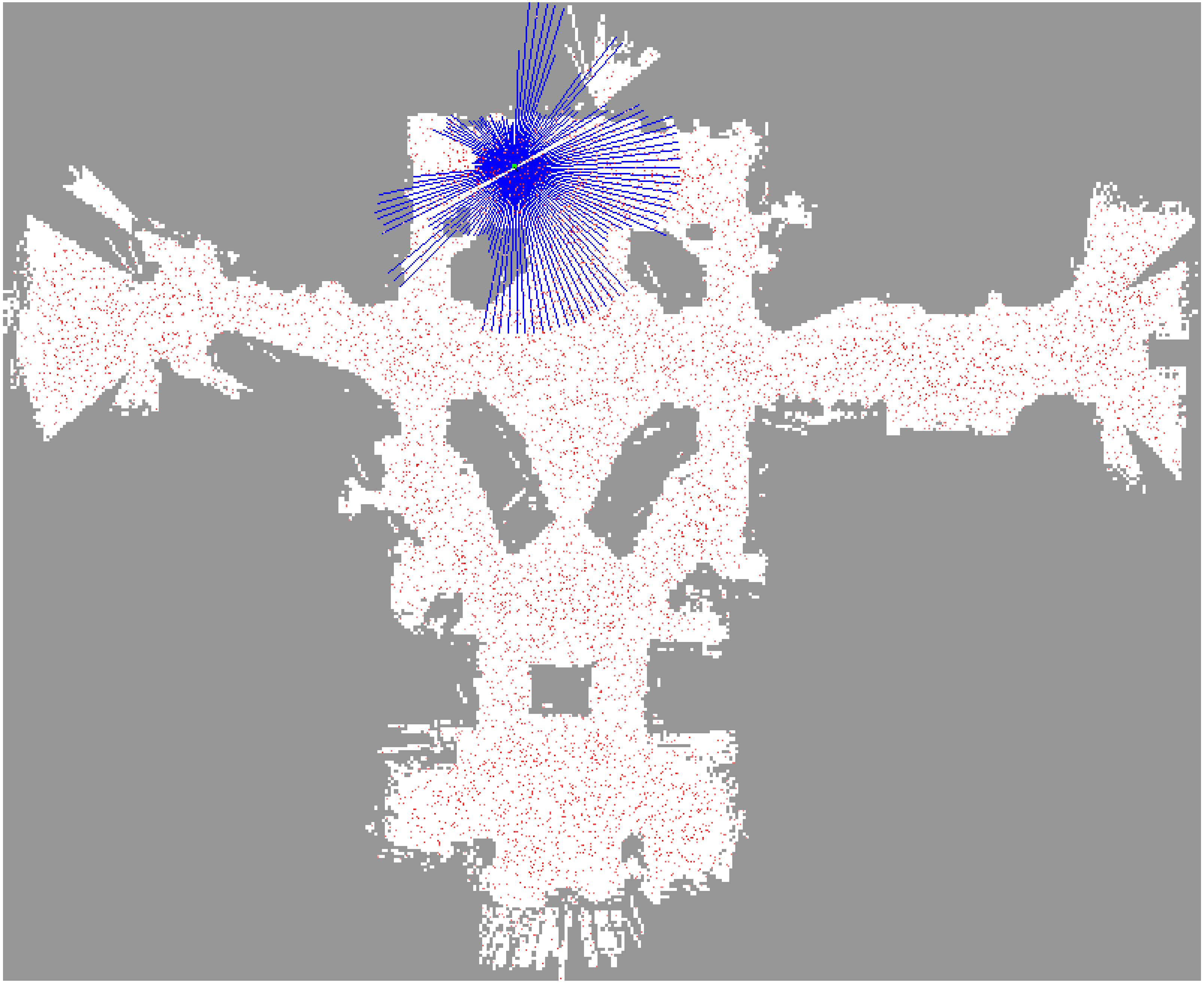
Laser sensor

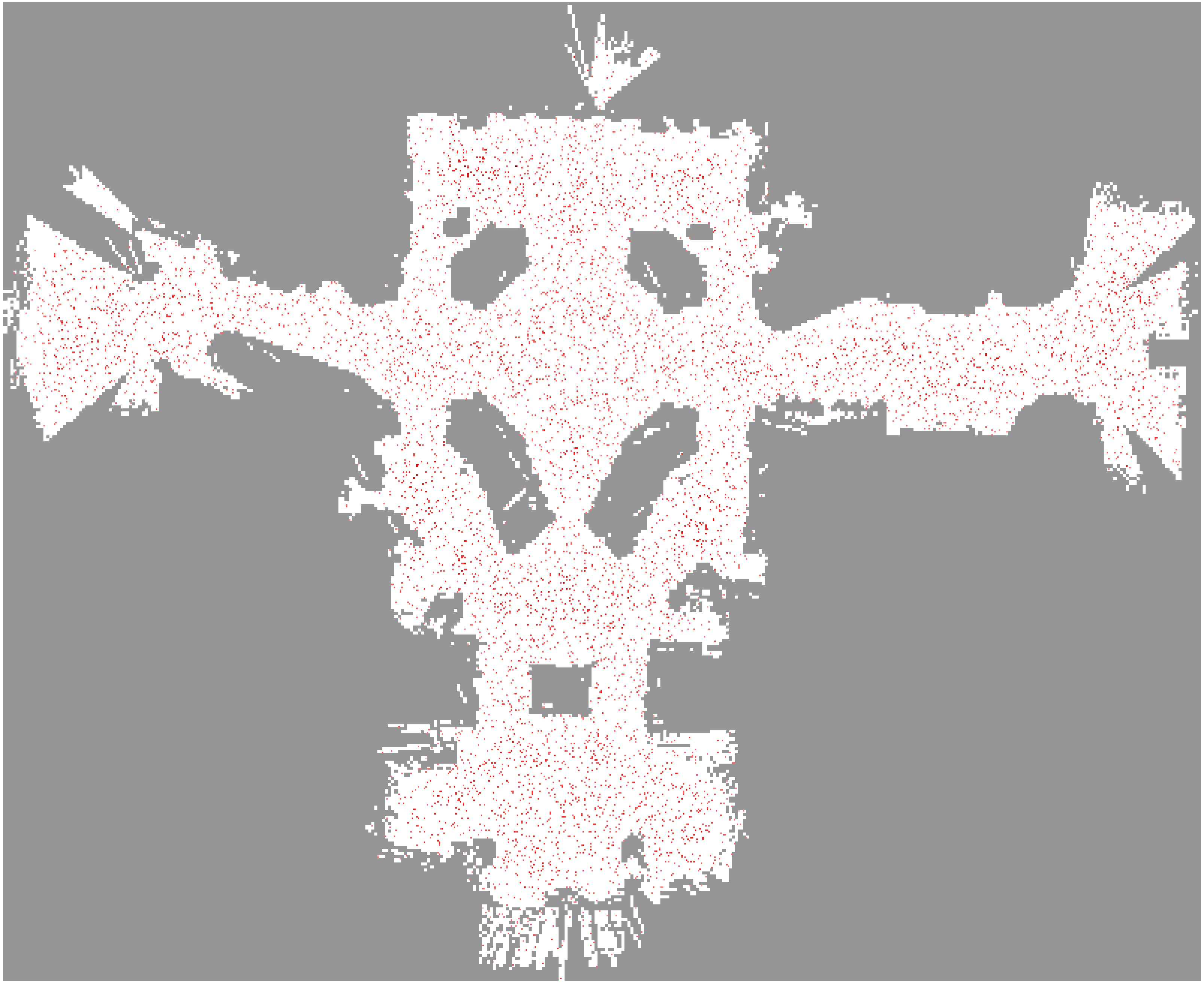


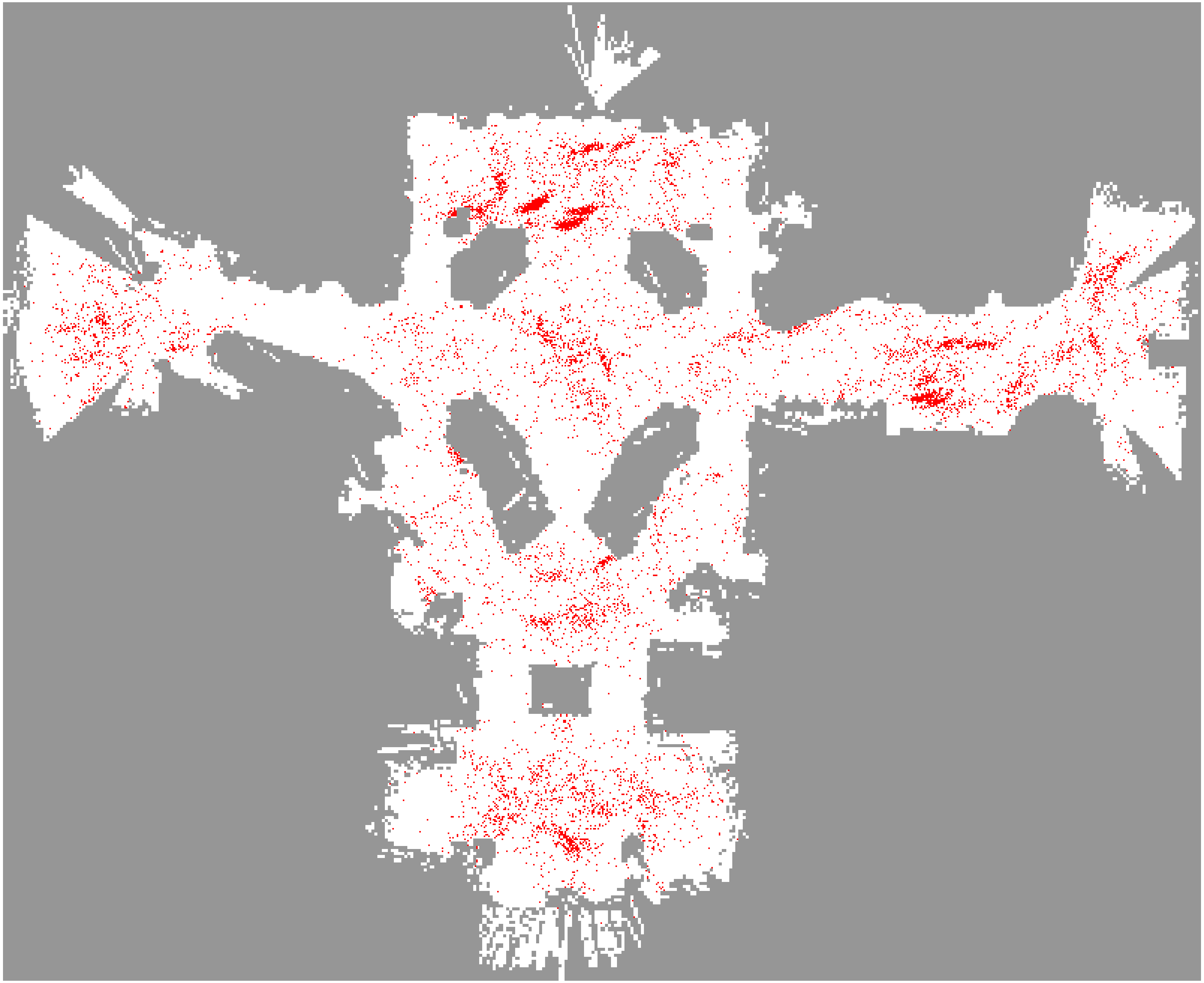
Sonar sensor

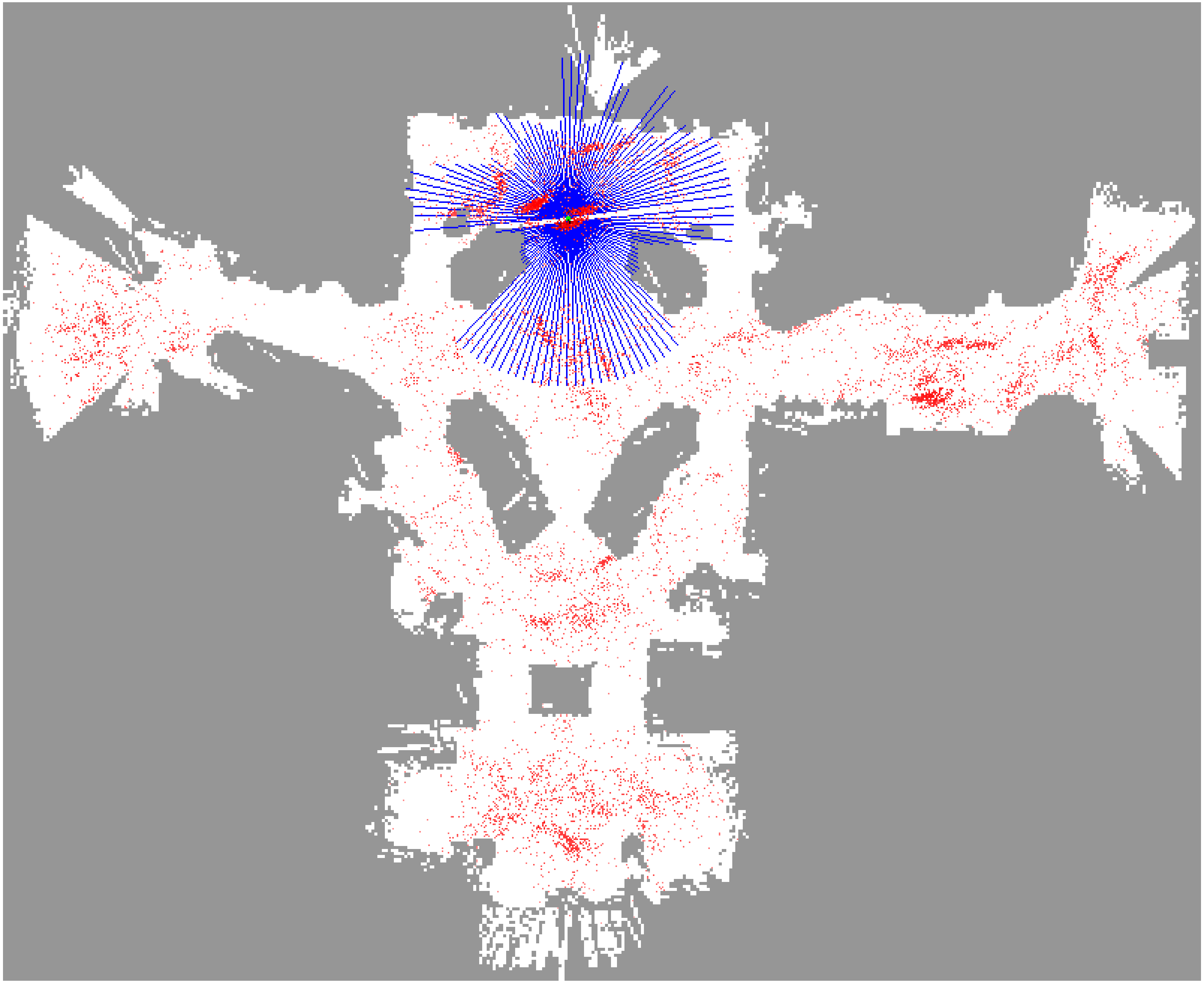
Particle Filter for Localization: Kidnapped Robot

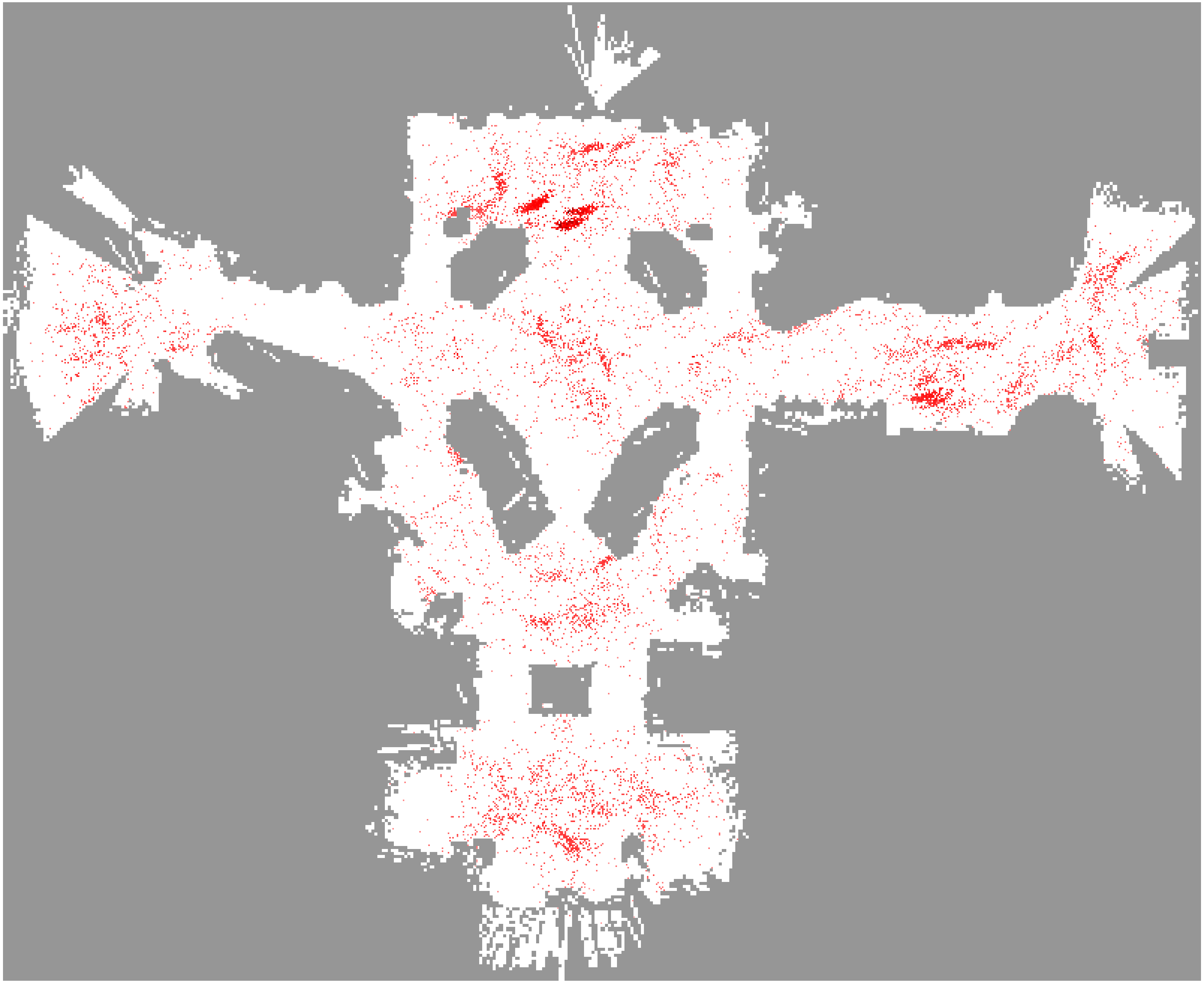


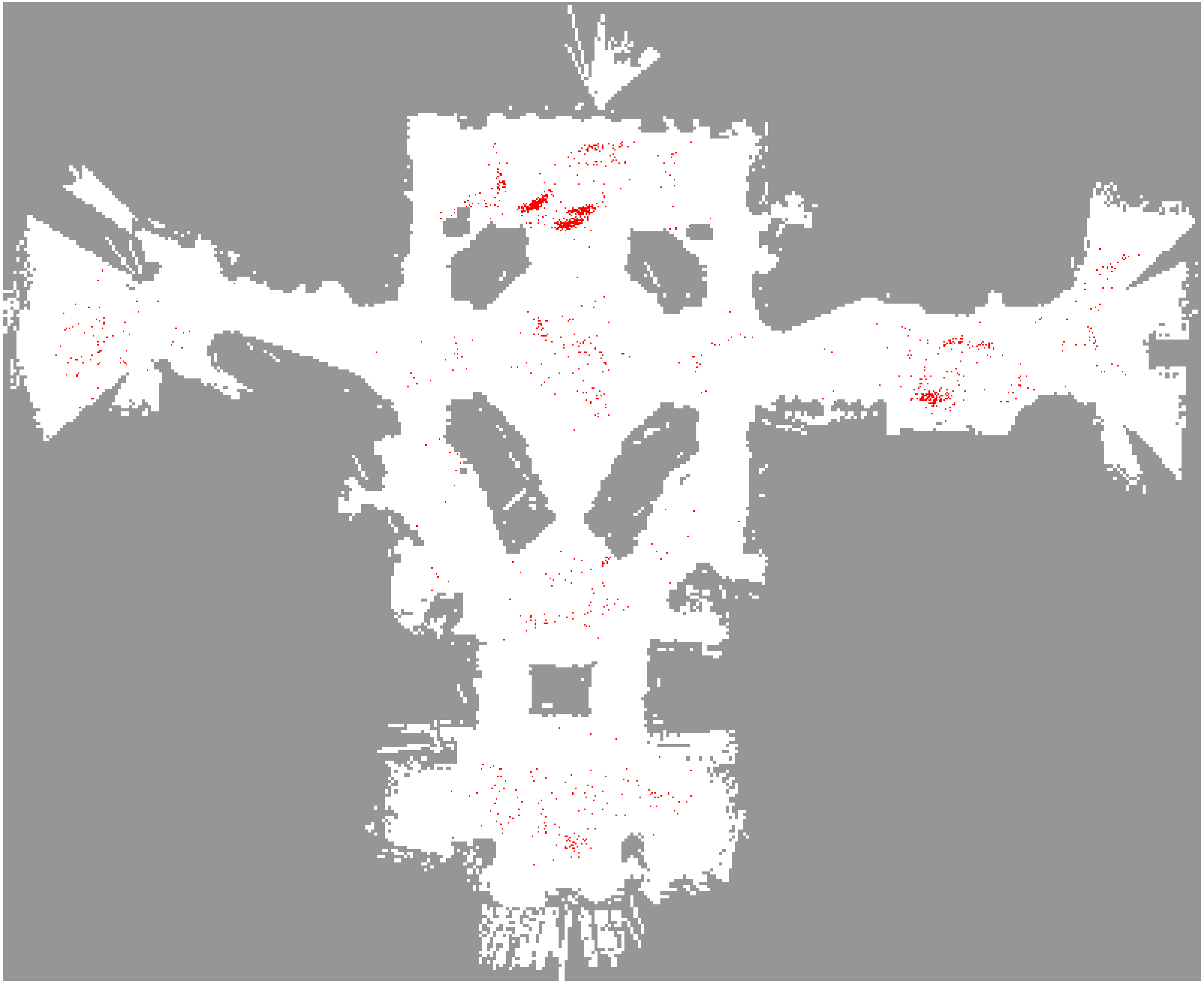


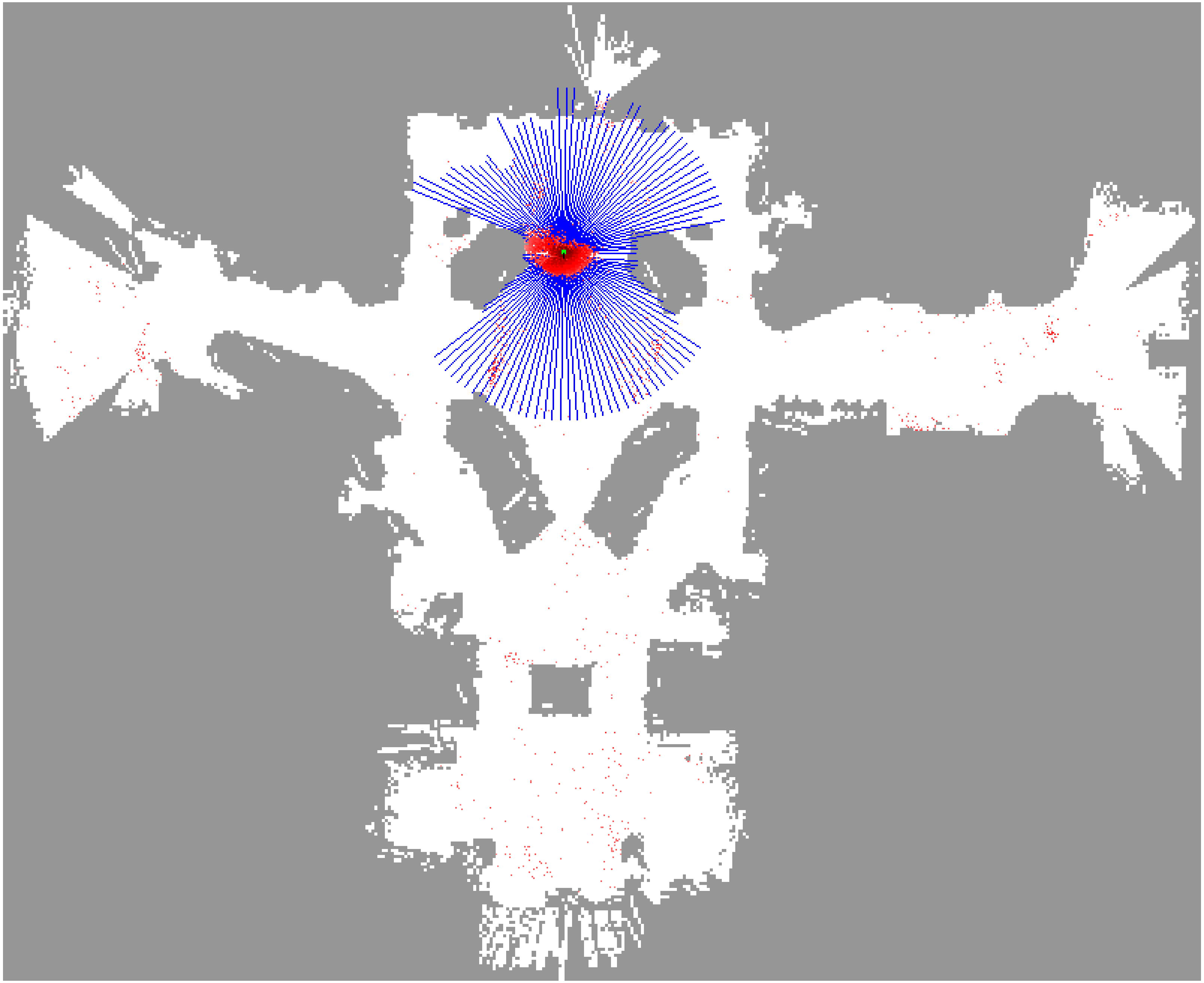


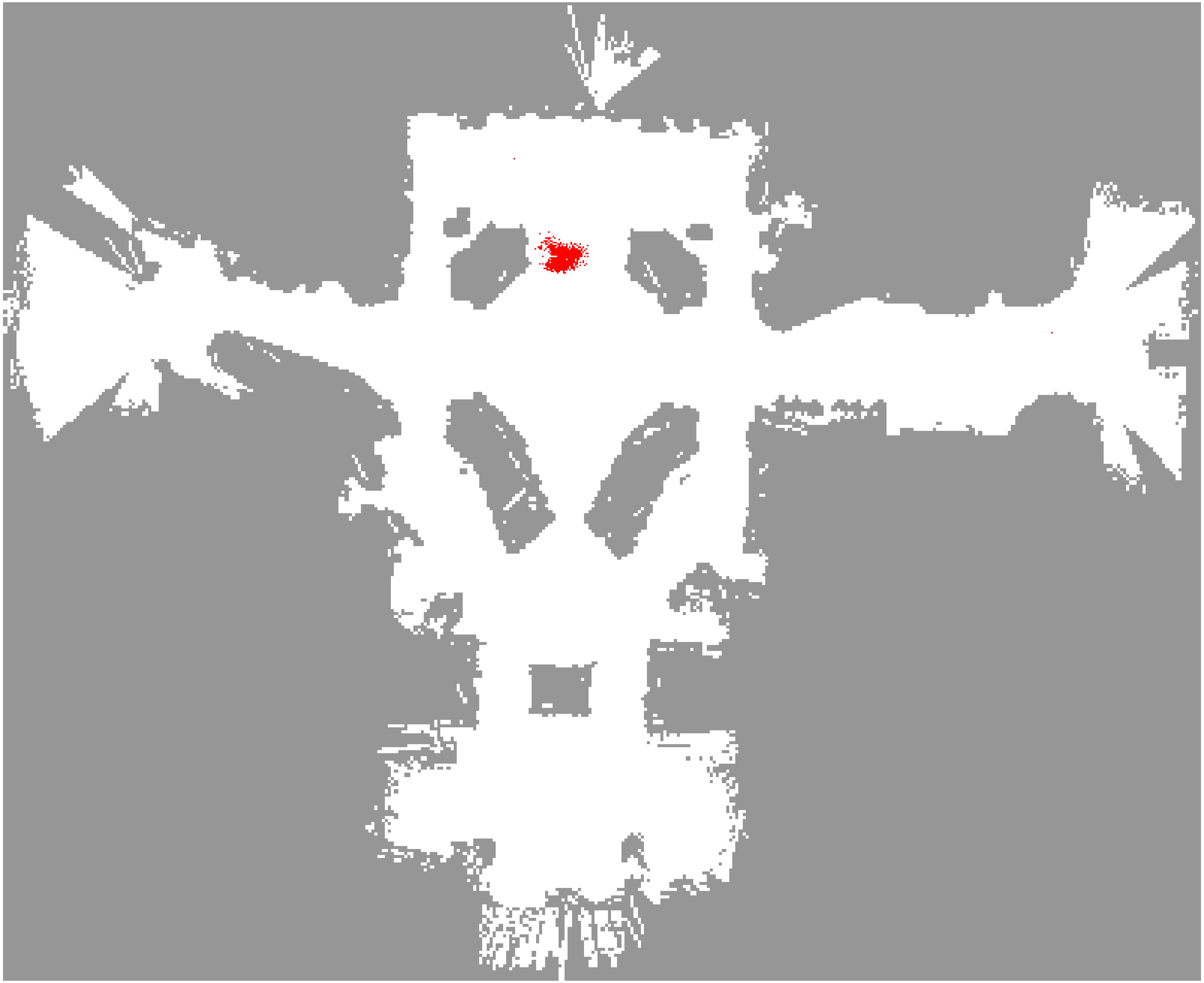


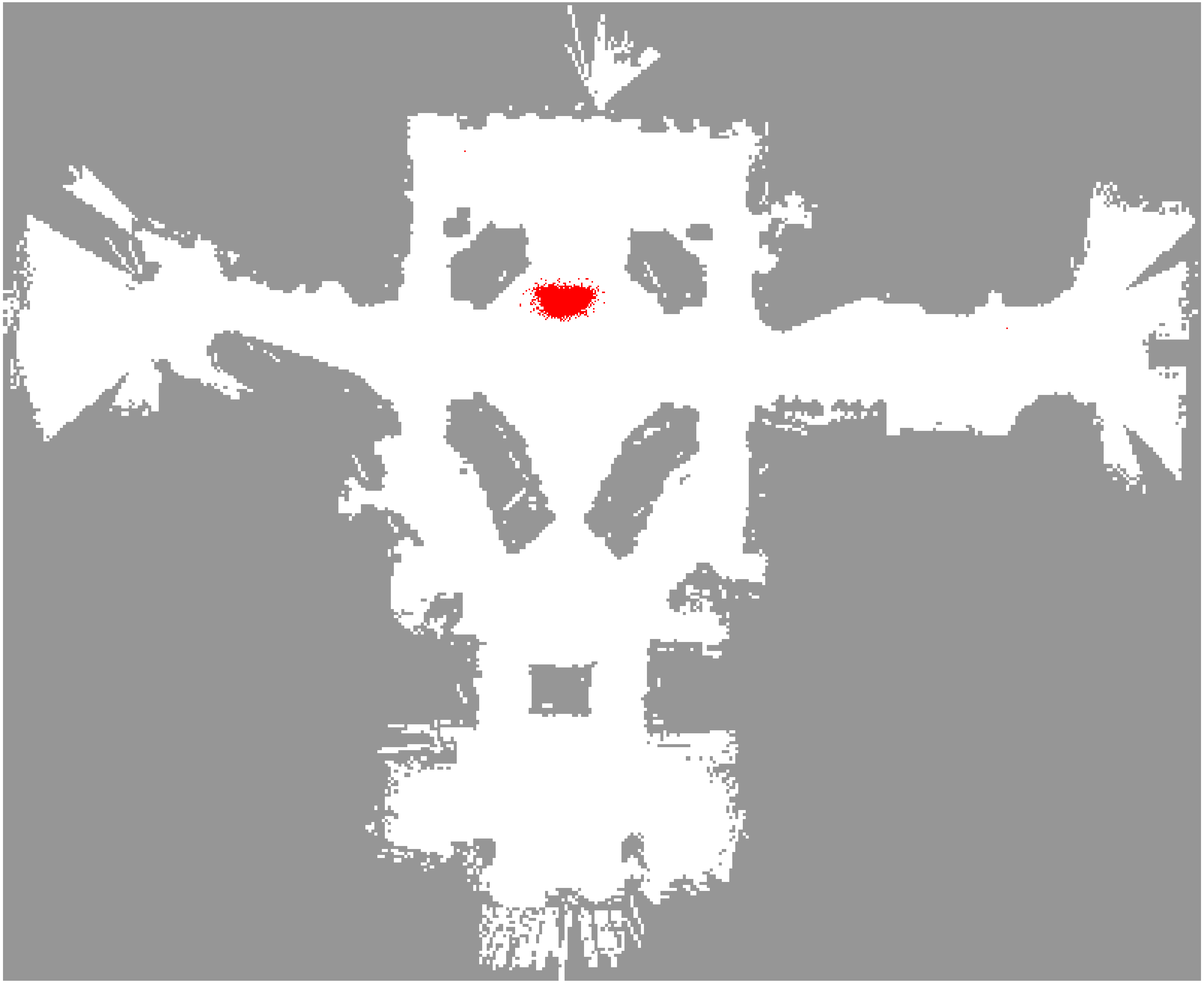


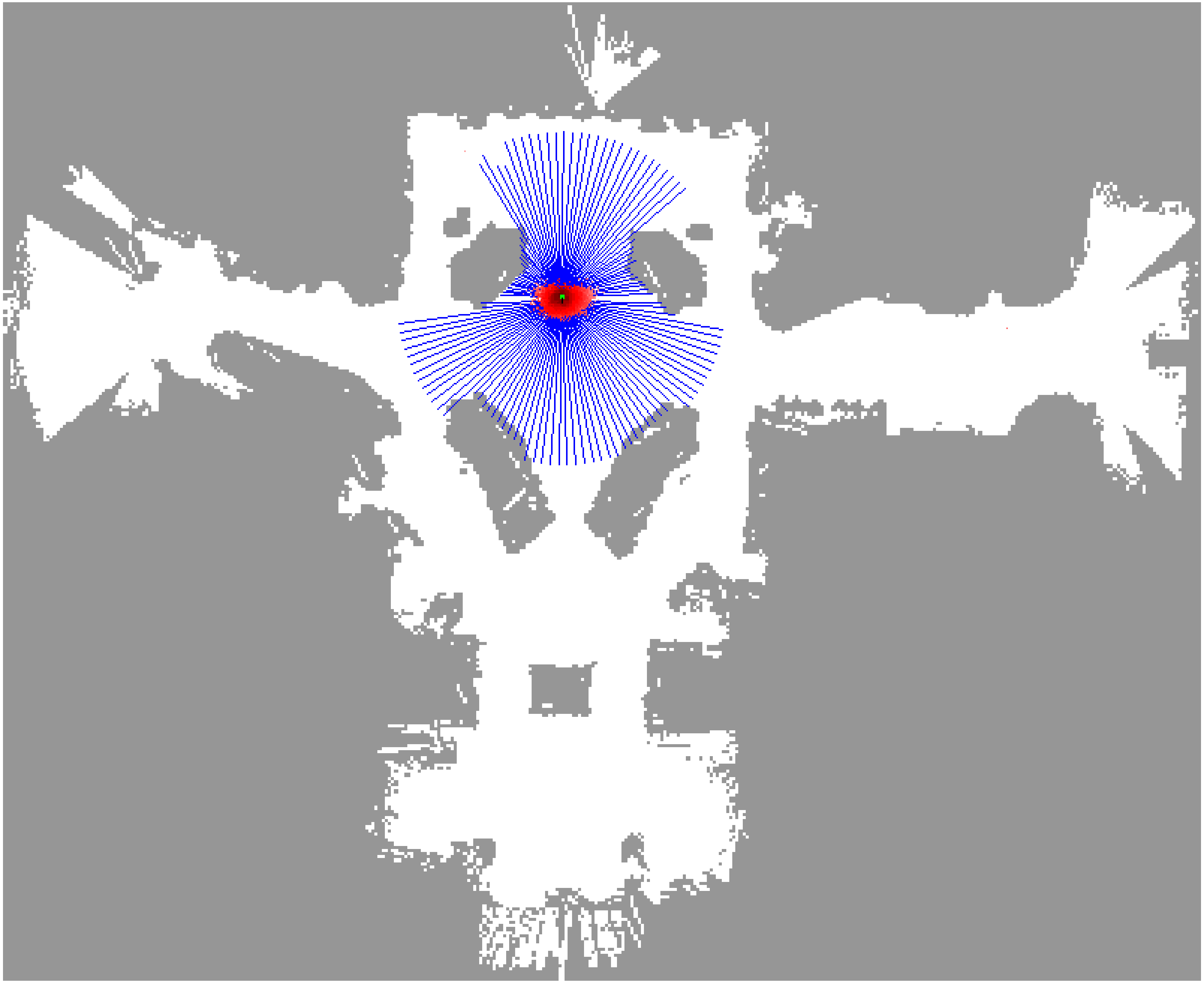


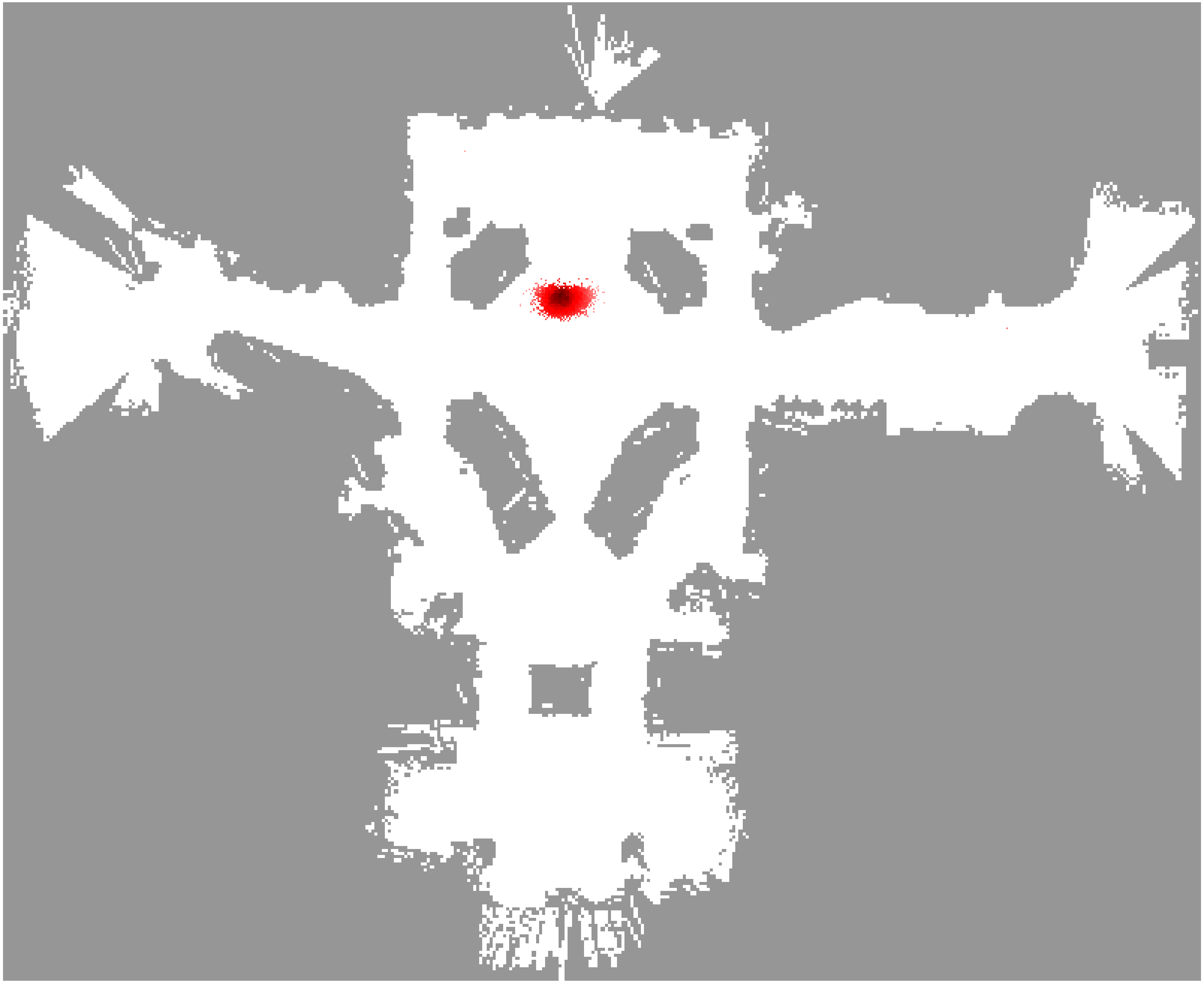


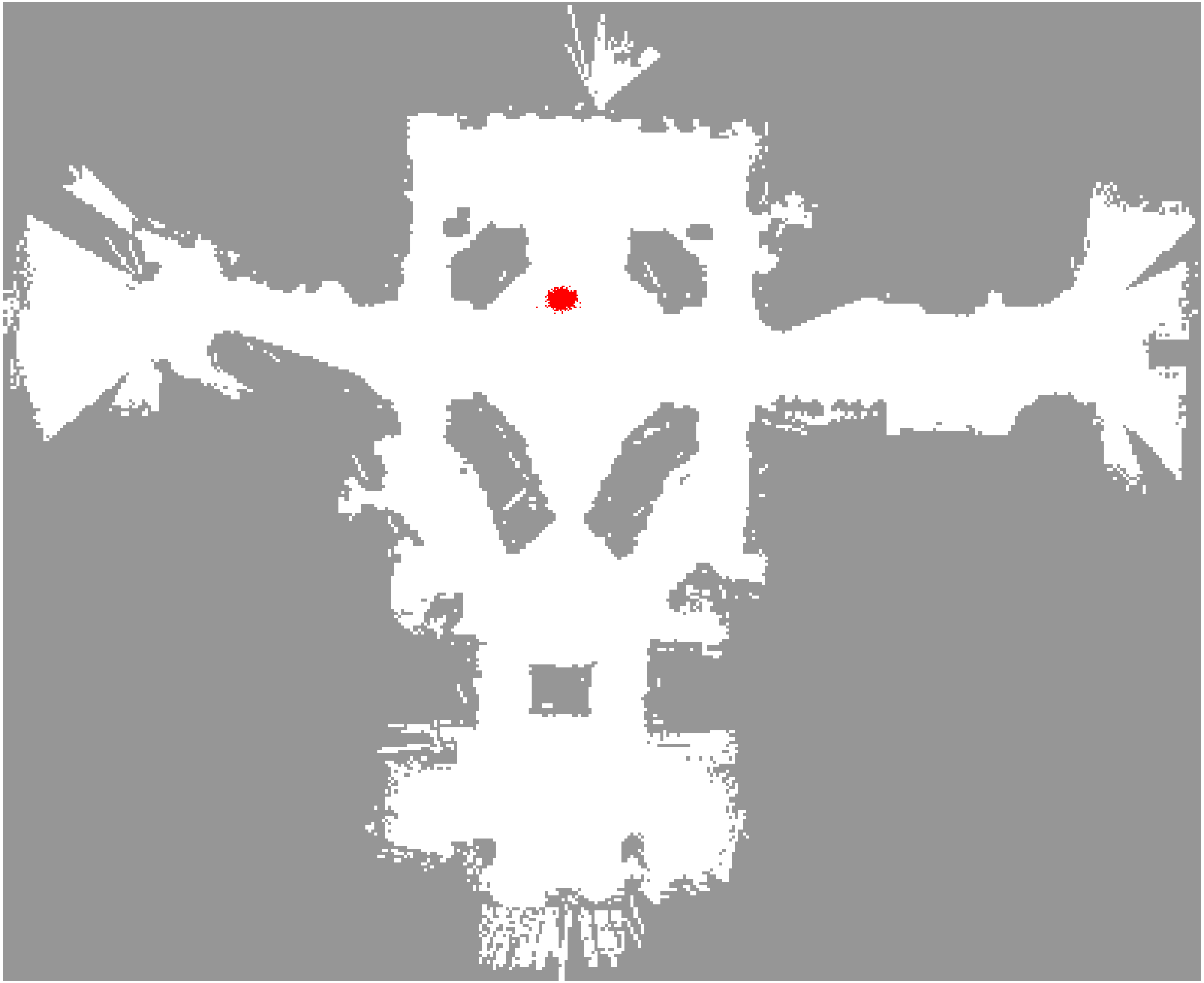


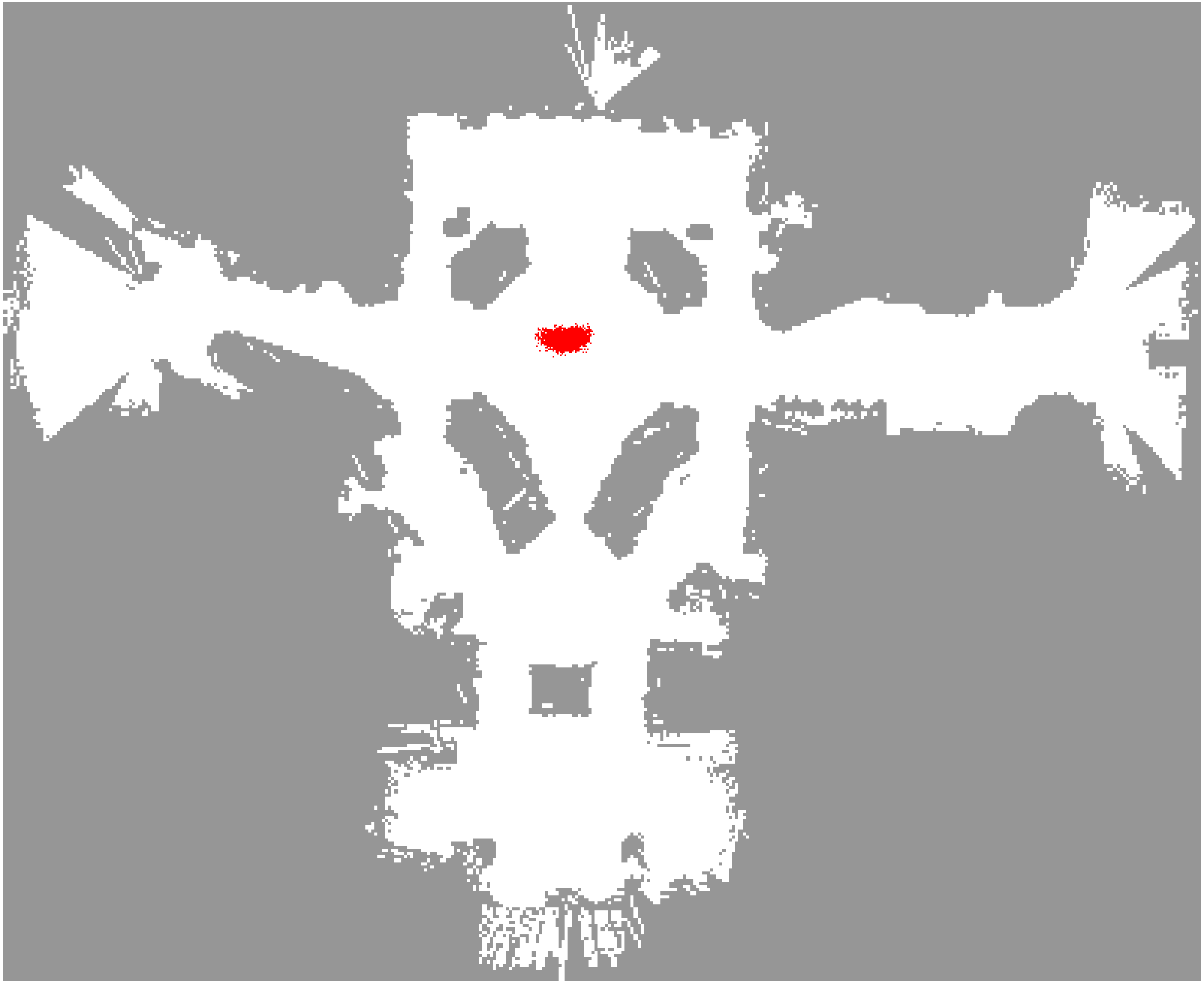


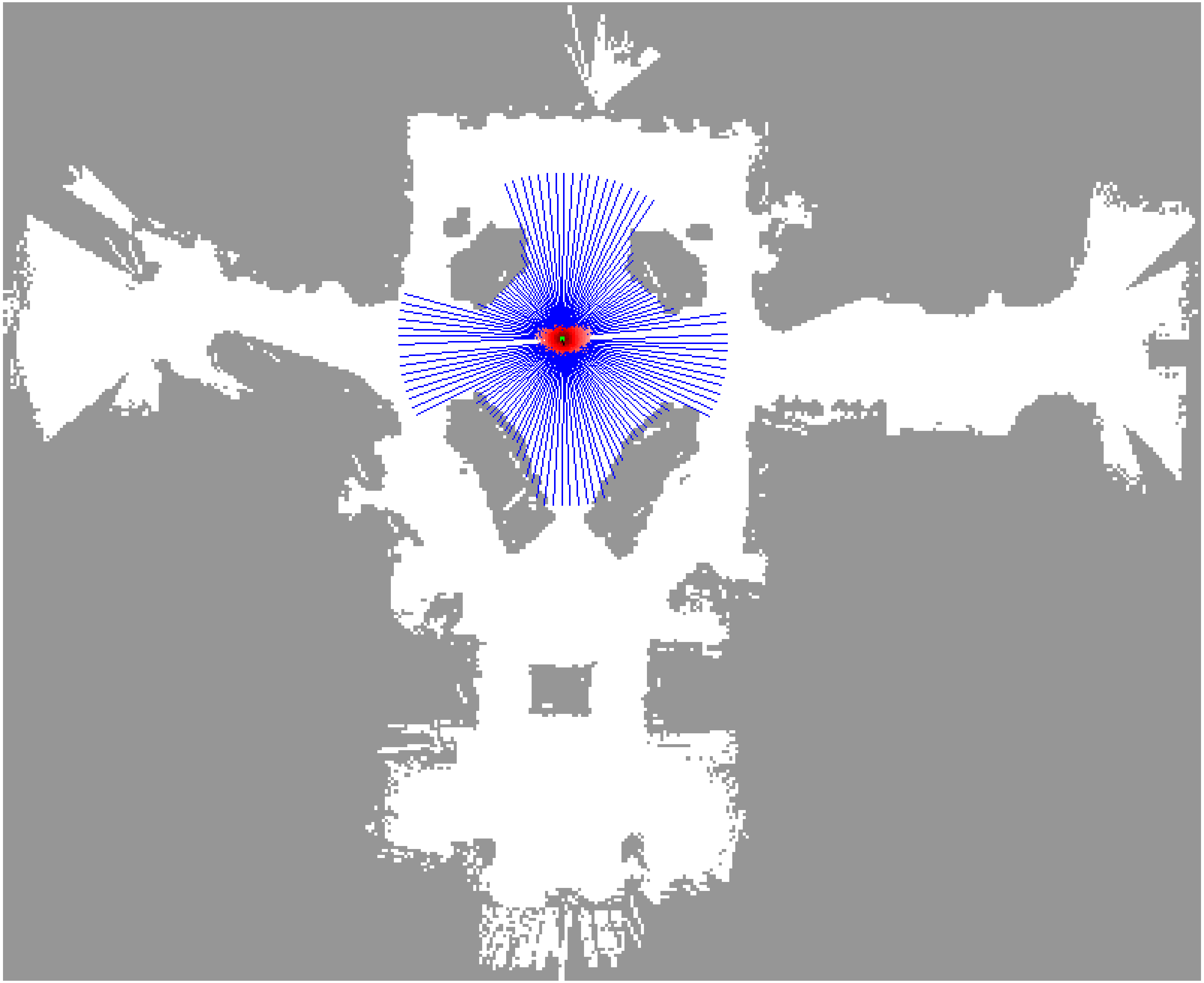


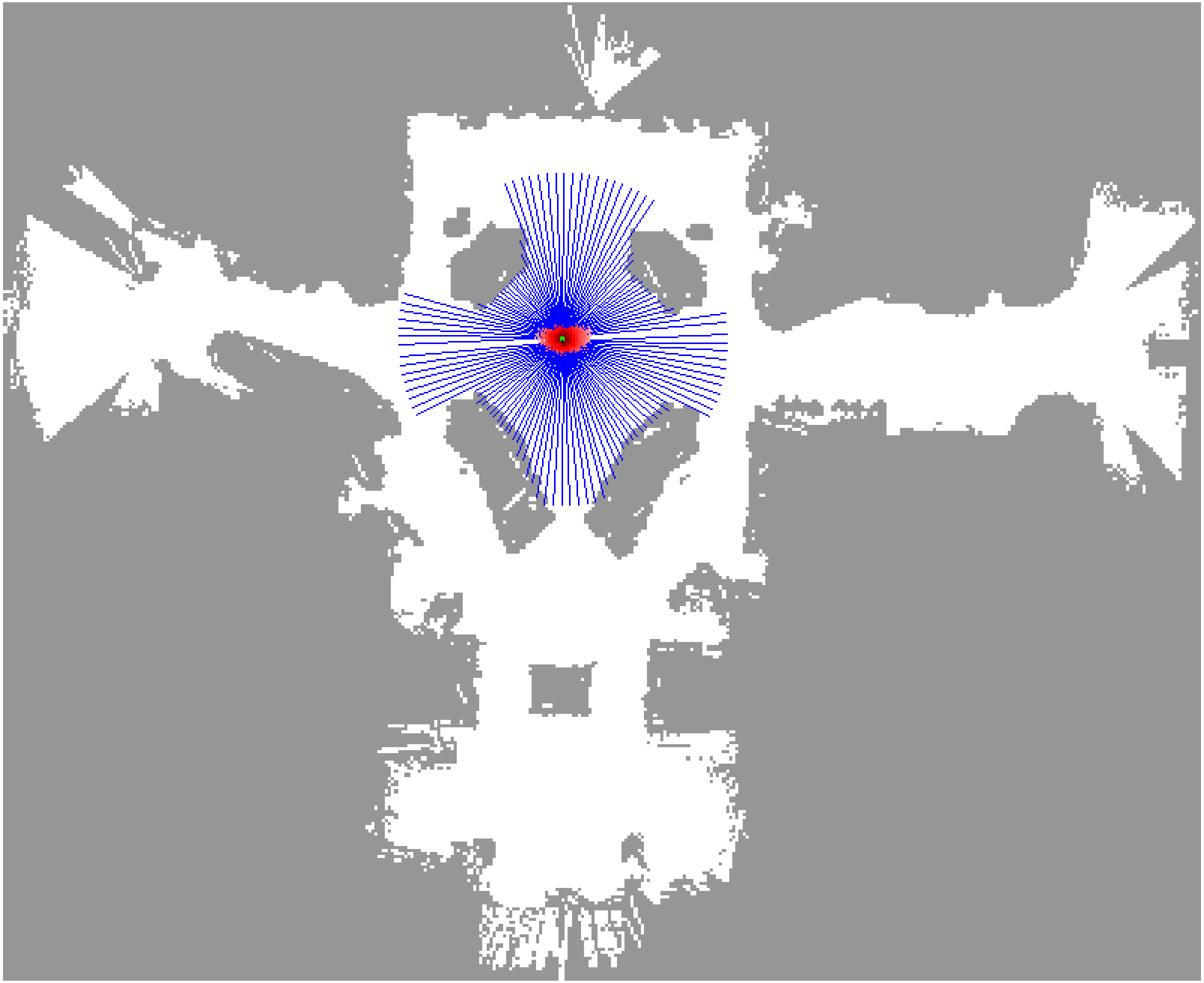




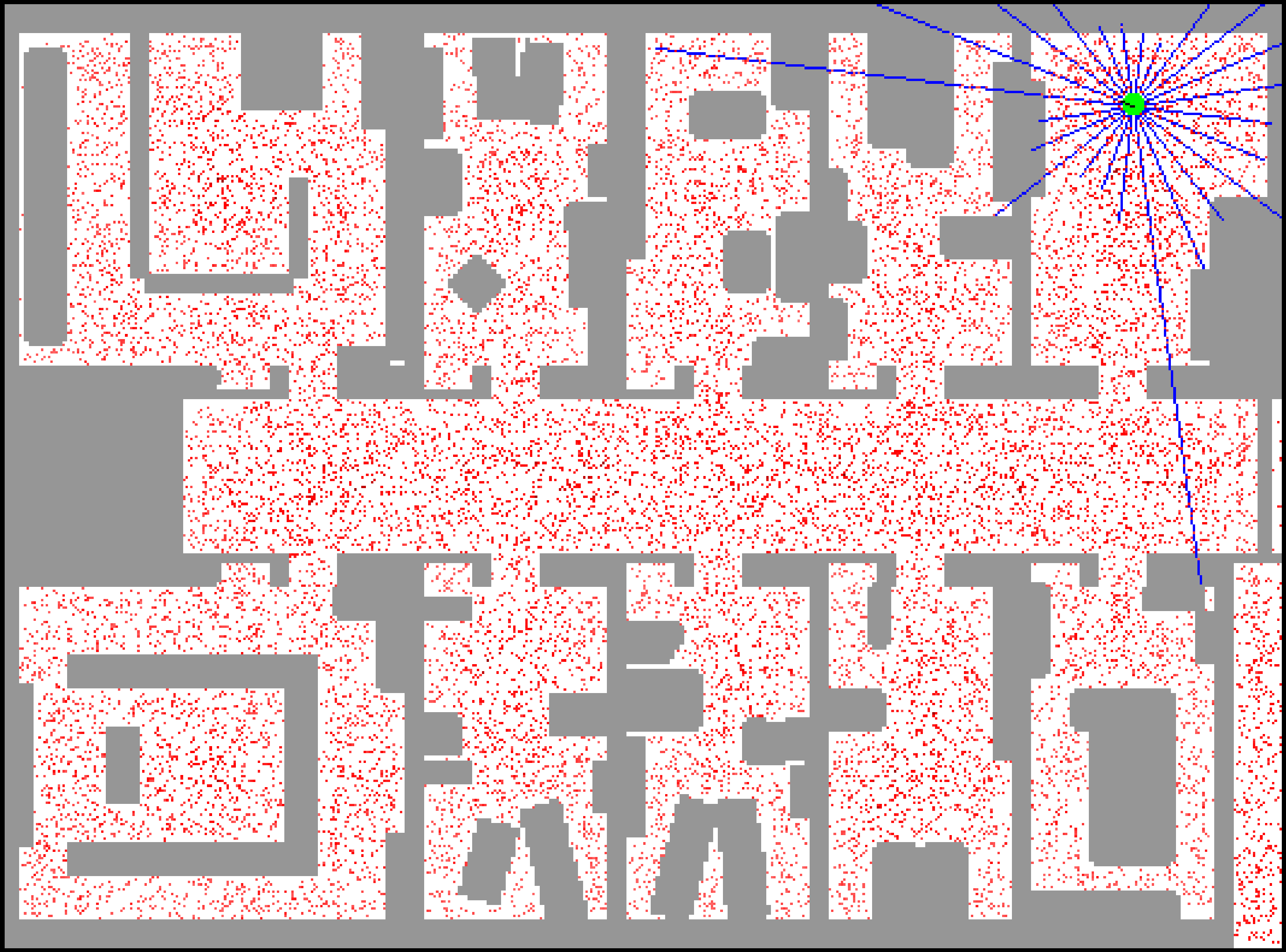








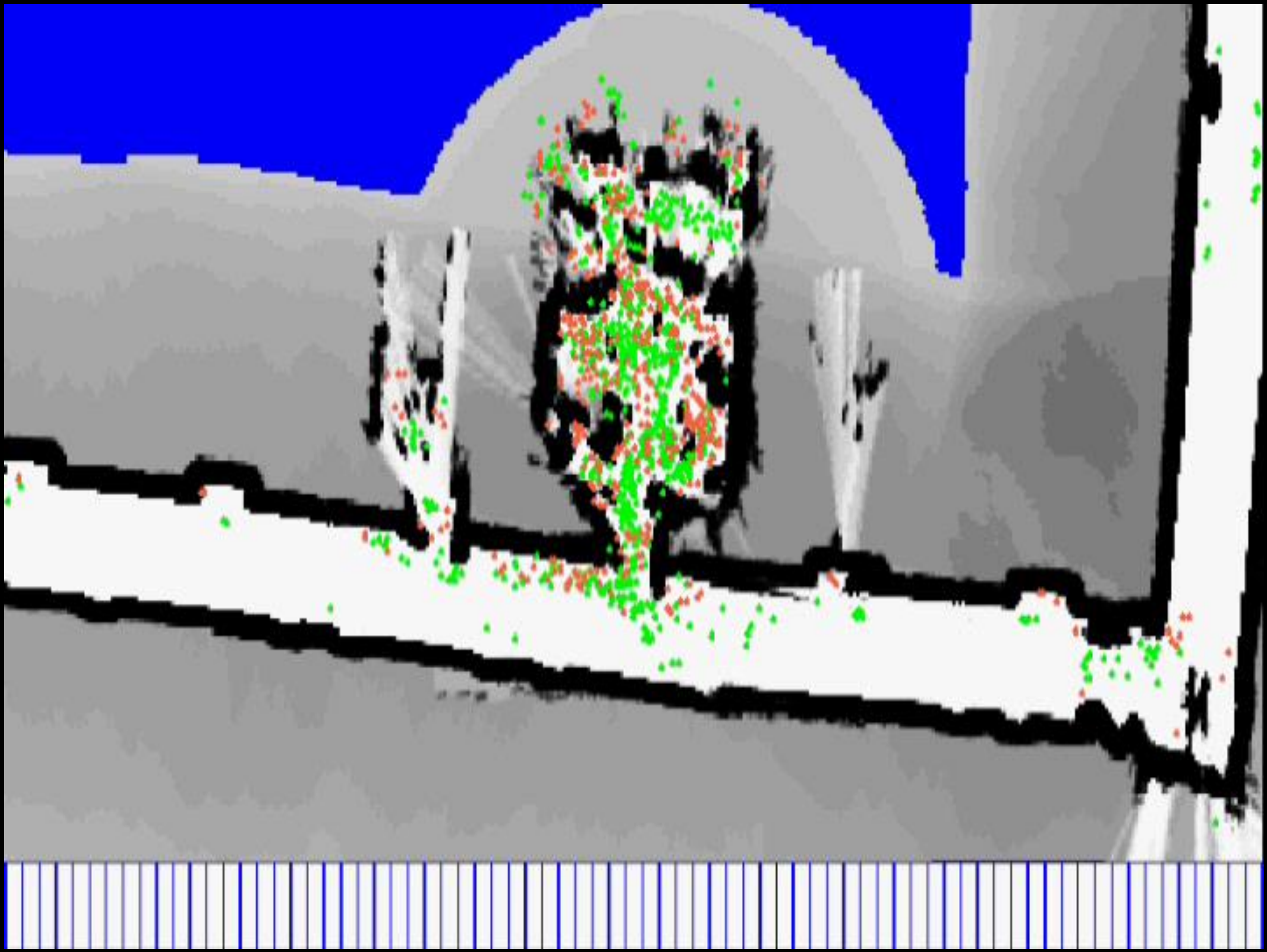
Example: Particle filter for a mobile robot



Fox et al.: Mobile robot localization with 24 sonar sensors

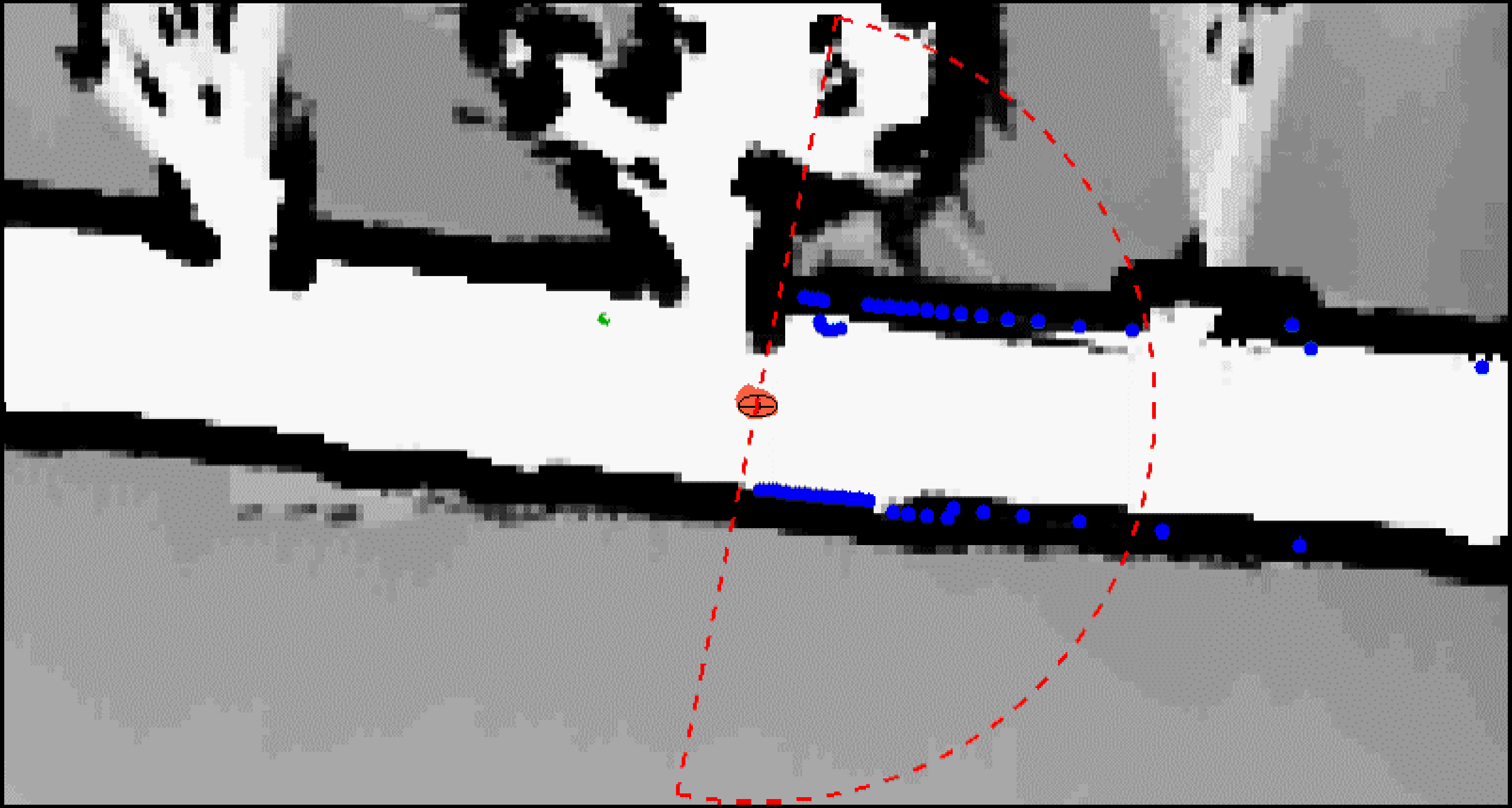
http://www.cs.washington.edu/ai/Mobile_Robotics/mcl/

Montemerlo et al.: Simultaneous localization and people tracking



Orange: robot
Green: person
http://www.cs.cmu.edu/~mmde/

Montemerlo et al.: Multiple people tracking



Orange: robot
Green: person

http://www.cs.cmu.edu/~mmde/

Summary

- Unscented Kalman Filter (UKF) can handle non-linear dynamics better than EKF
  - But there is some added computational cost
  - Still can't handle non-guassian state uncertainty
- Particle Filters represent a distribution (continuous or discrete) with a set of particles
  - The key is to re-sample the particles after propagating them
  - Can approximate arbitrary probability distributions
    - Generality at the expense of fast updates
  - The more particles, the better the approximation!

Homework

- Read AI book Ch. 17.1-17.4
- Remember to start HW5!
- Remember to register your final project choice by midnight tonight!