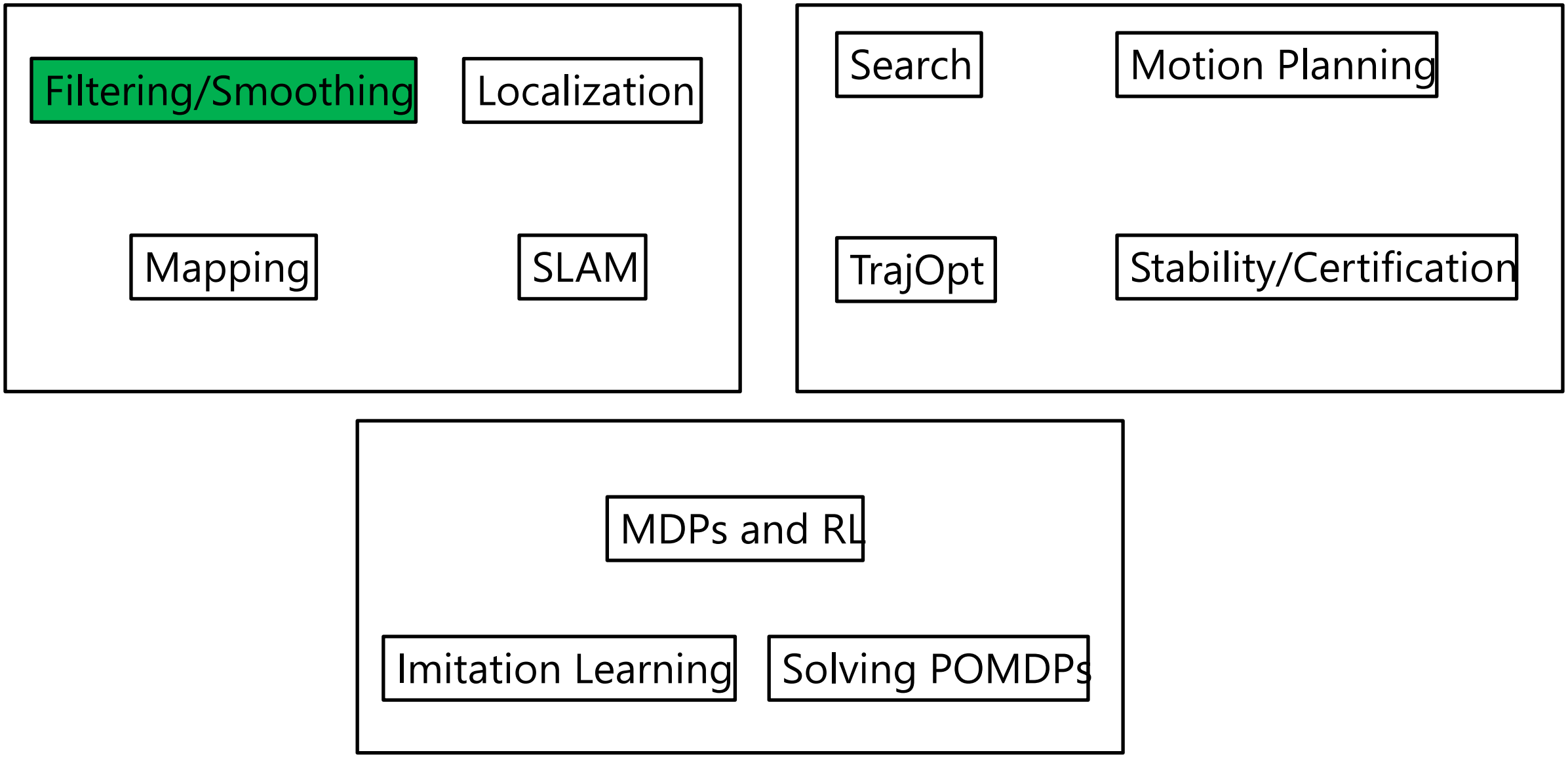
Robotics

Spring 2023

Abhishek Gupta

TAs: Yi Li, Srivatsa GS

Recap: Course Overview



Lecture Outline

Particle Filters

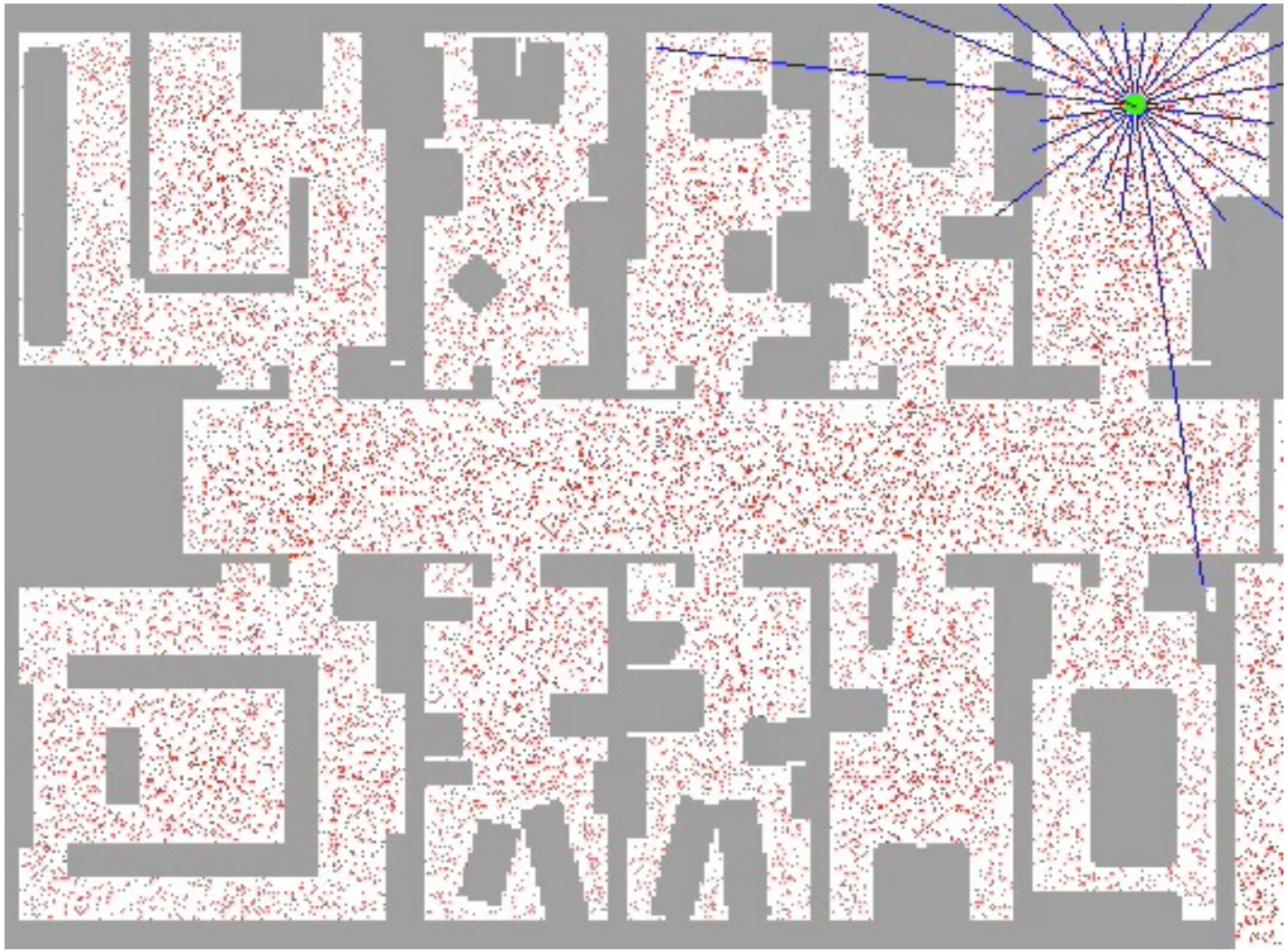
Motion Models

Sensor Models

Particle Filters: Motivation

- So far, we discussed the
  - Kalman filter: Gaussian, linearization problems, discrete Bayes filters (eg histogram filters)
- Histogram filters are great but they waste space and are non adaptive
- Particle filters are a way to **efficiently** represent **non-Gaussian distributions adaptively**
- Basic principle
  - Set of state hypotheses (“particles”)
  - Survival-of-the-fittest

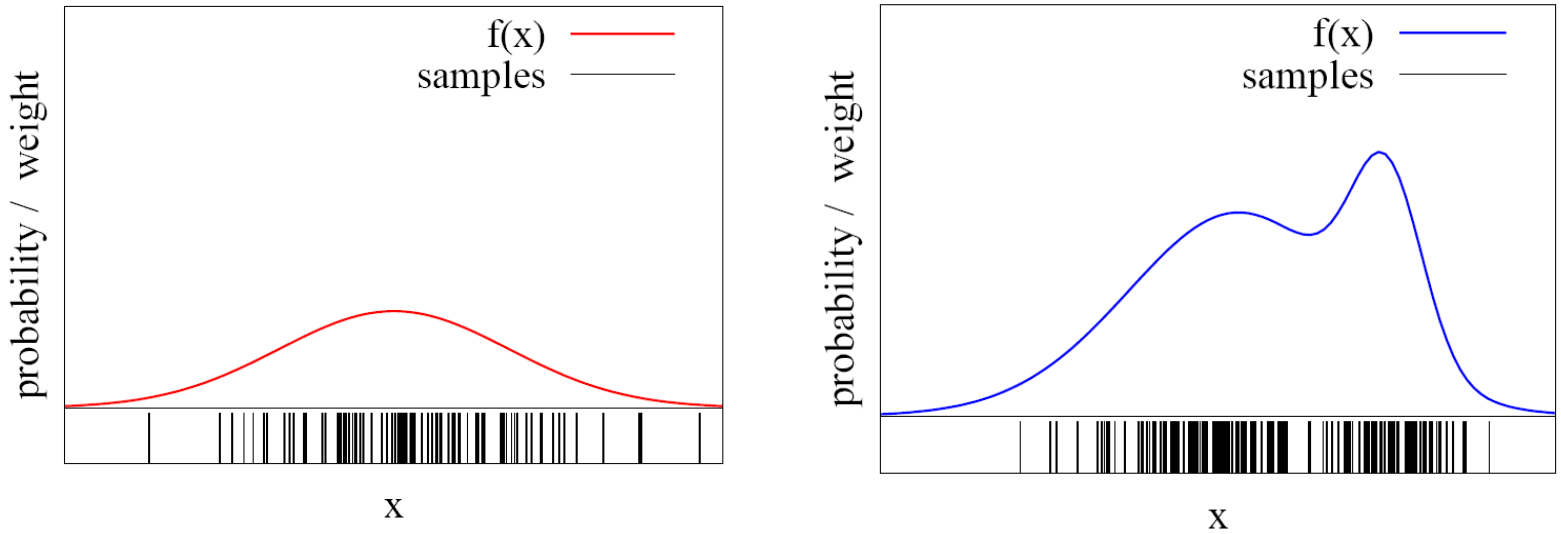
Sample-based Localization (sonar)



Let’s introduce some tools

Density Approximation

- Particle sets can be used to approximate densities

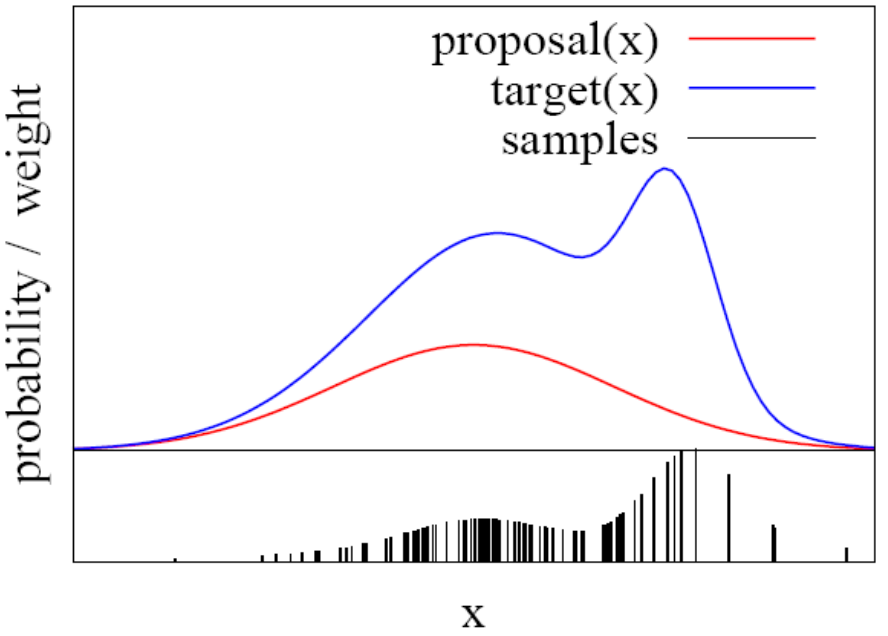


- The more particles fall into an interval, the higher the probability of that interval

- How to draw samples form a function/distribution?

Importance Sampling Principle

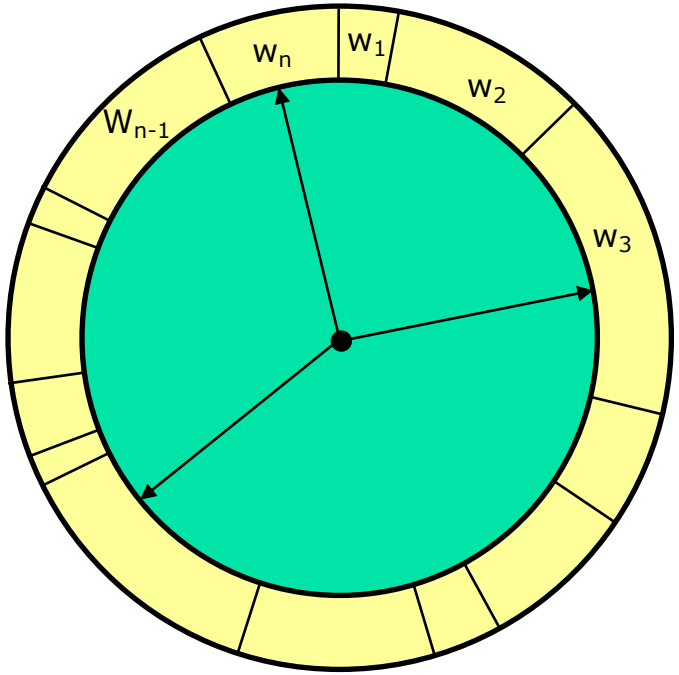
- We can even use a different distribution *g* to generate samples from *f*
- By introducing an importance weight w, we can account for the "differences between *g* and *f* "
- $w = f / g$
- *f* is often called target
- *g* is often called proposal



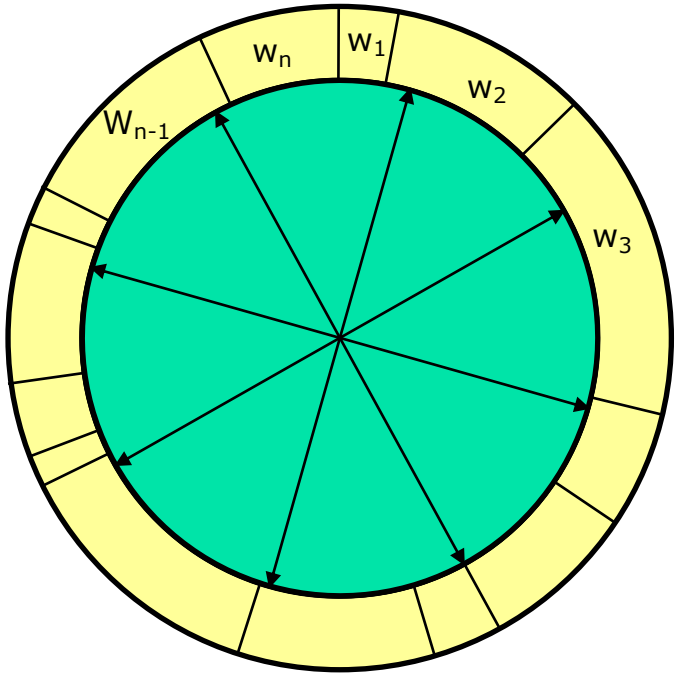
Resampling

- **Given**: Set *S* of weighted samples.
- **Wanted** : Random sample, where the probability of drawing x_i is given by w_i.
- Typically done *n* times with replacement to generate new sample set *S'*.

Resampling: Efficient Techniques



- Roulette wheel
- Binary search, n log n



- Stochastic universal sampling
- Systematic resampling
- Linear time complexity
- Easy to implement, low variance

Resampling: Efficient Techniques

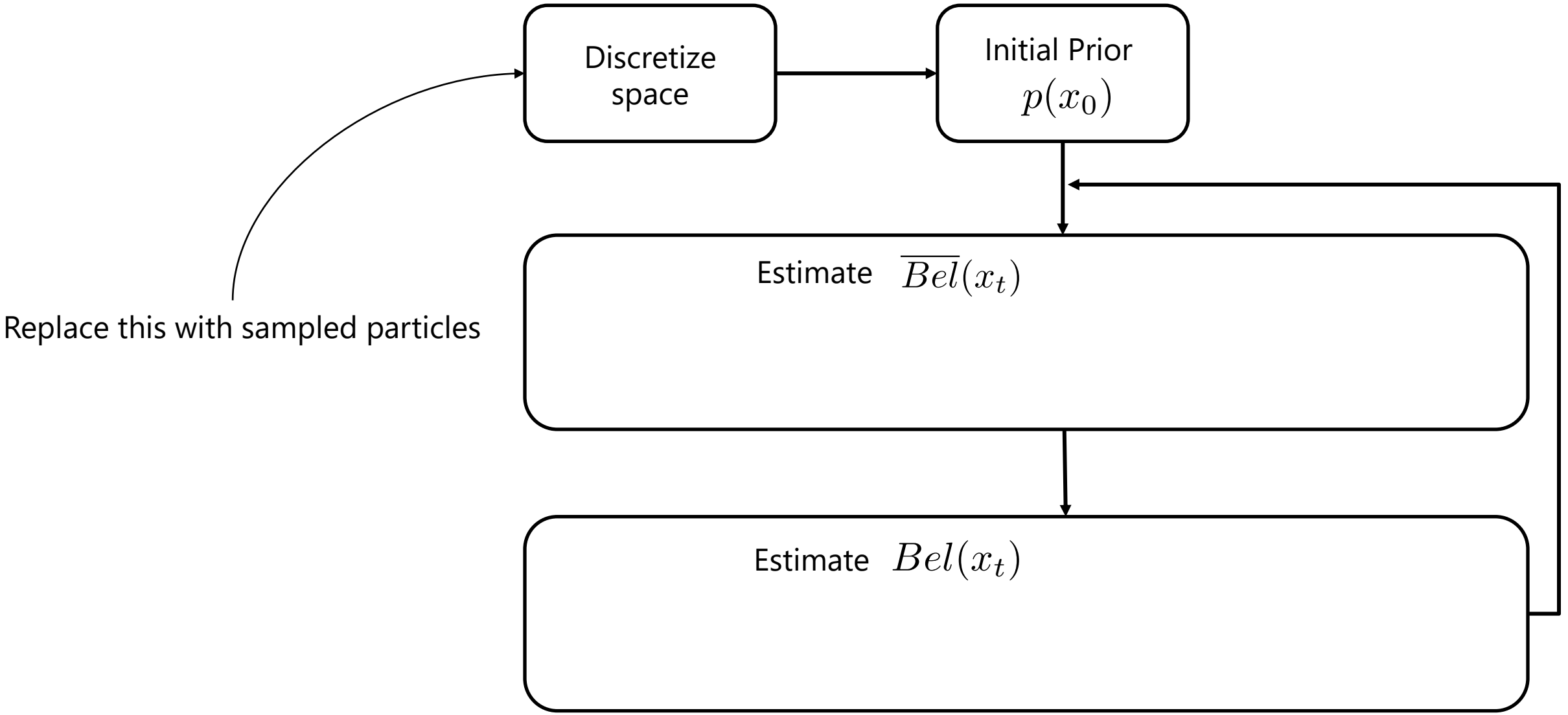
Pseudocode for low-variance sampling

```
1:   Algorithm Low_variance_sampler(X_t, W_t):
2:       X̄_t = ∅
3:       r = rand(0; M^-1)
4:       c = w_t^[1]
5:       i = 1
6:       for m = 1 to M do
7:           u = r + (m − 1) · M^-1
8:           while u > c
9:               i = i + 1
10:              c = c + w_t^[i]
11:          endwhile
12:          add x_t^[i] to X̄_t
13:      endfor
14:      return X̄_t
```

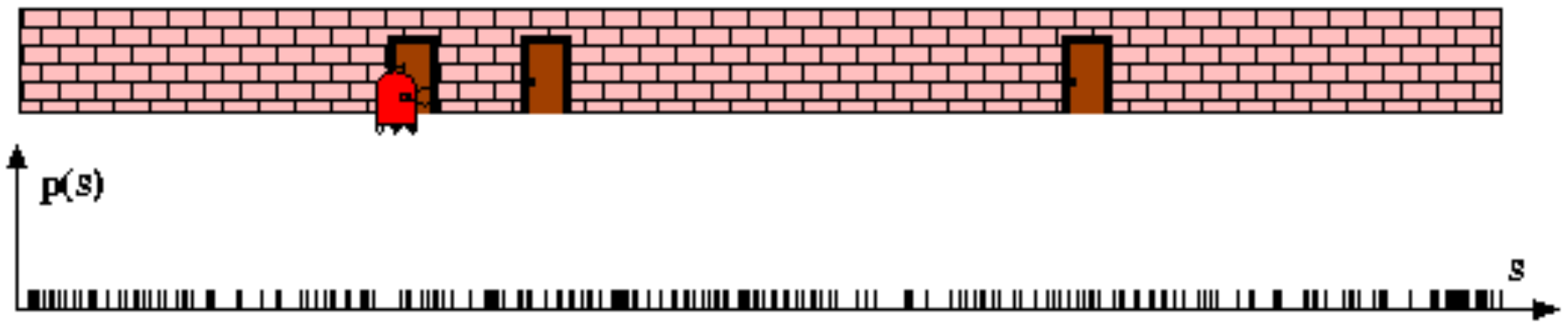
Table 4.4 Low variance resampling for the particle filter. This routine uses a single random number to sample from the particle set $\mathcal{X}$ with associated weights $\mathcal{W}$, yet the probability of a particle to be resampled is still proportional to its weight. Furthermore, the sampler is efficient: Sampling M particles requires $O(M)$ time.

Let’s put these pieces together

Particle Filters



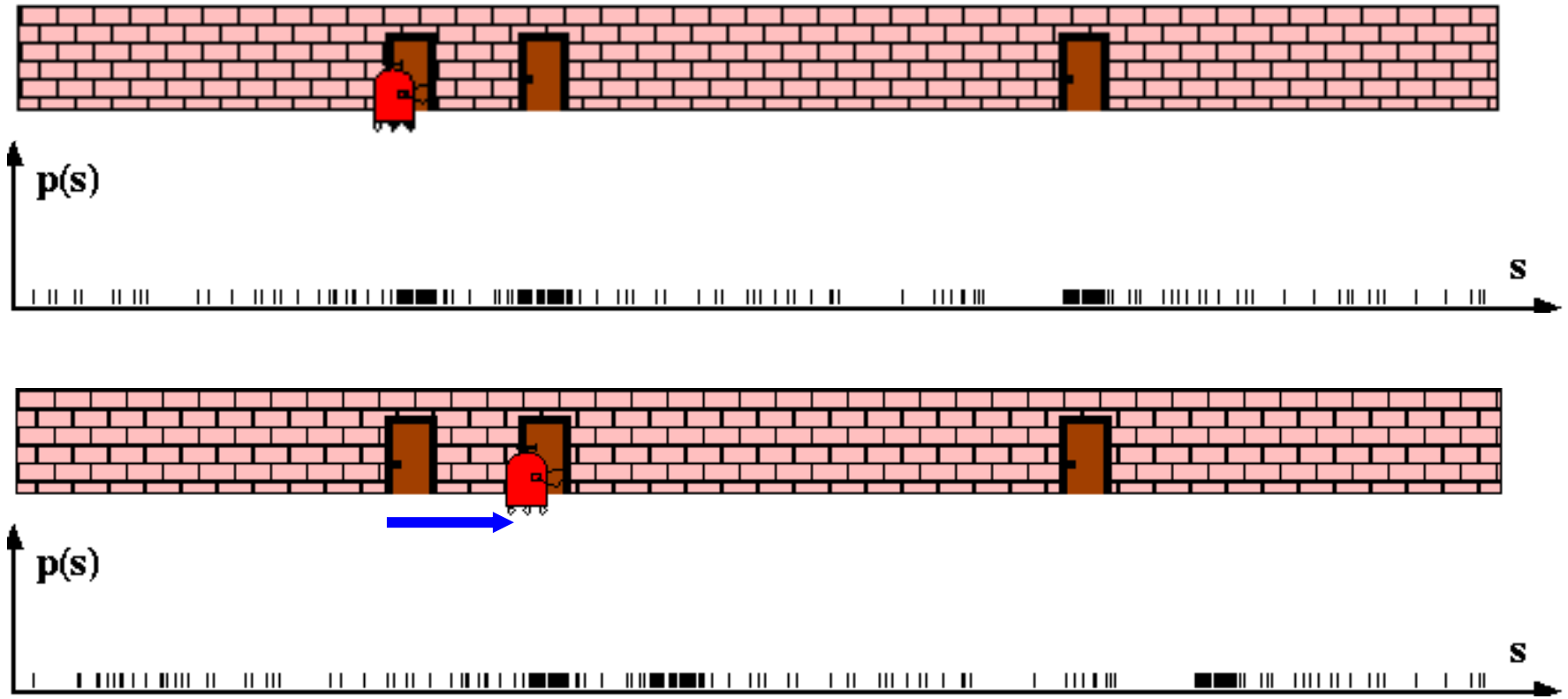
Particle Filters



Robot Motion

$$\overline{Bel}(x_t) = \int P(x_t|u_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

Push samples forward according to dynamics

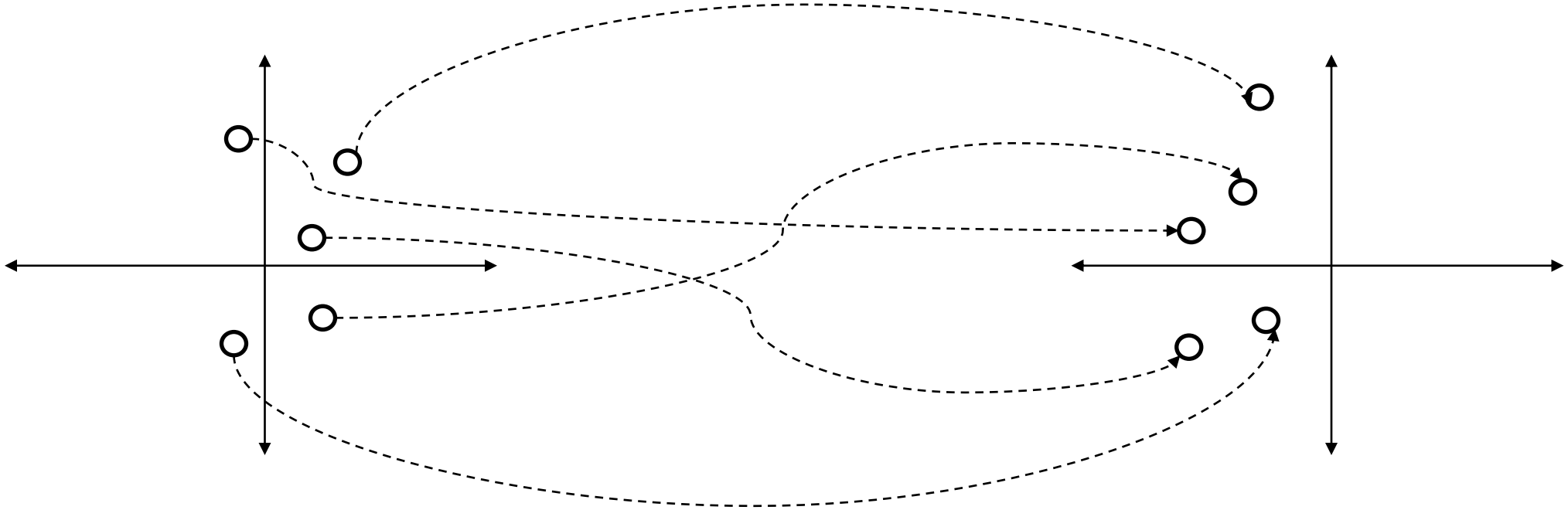


Dynamics Update:

$$\overline{Bel}(x_t) = \int P(x_t|u_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

Sample forward using the dynamics model:

1. No gaussian requirement
2. No linearity requirement, just push forward distribution



Sensor Information: Measurement Update

Can no longer just push forward with evidence, need to normalize

$$Bel(x_t) = \eta P(z_t|x_t)\overline{Bel}(x_t)$$

$$Bel(x_t) = \frac{P(z_t|x_t)\overline{Bel}(x_t)}{\int P(z_t|x_t)\overline{Bel}(x_t)dx_t}$$

Looks a lot like importance sampling!

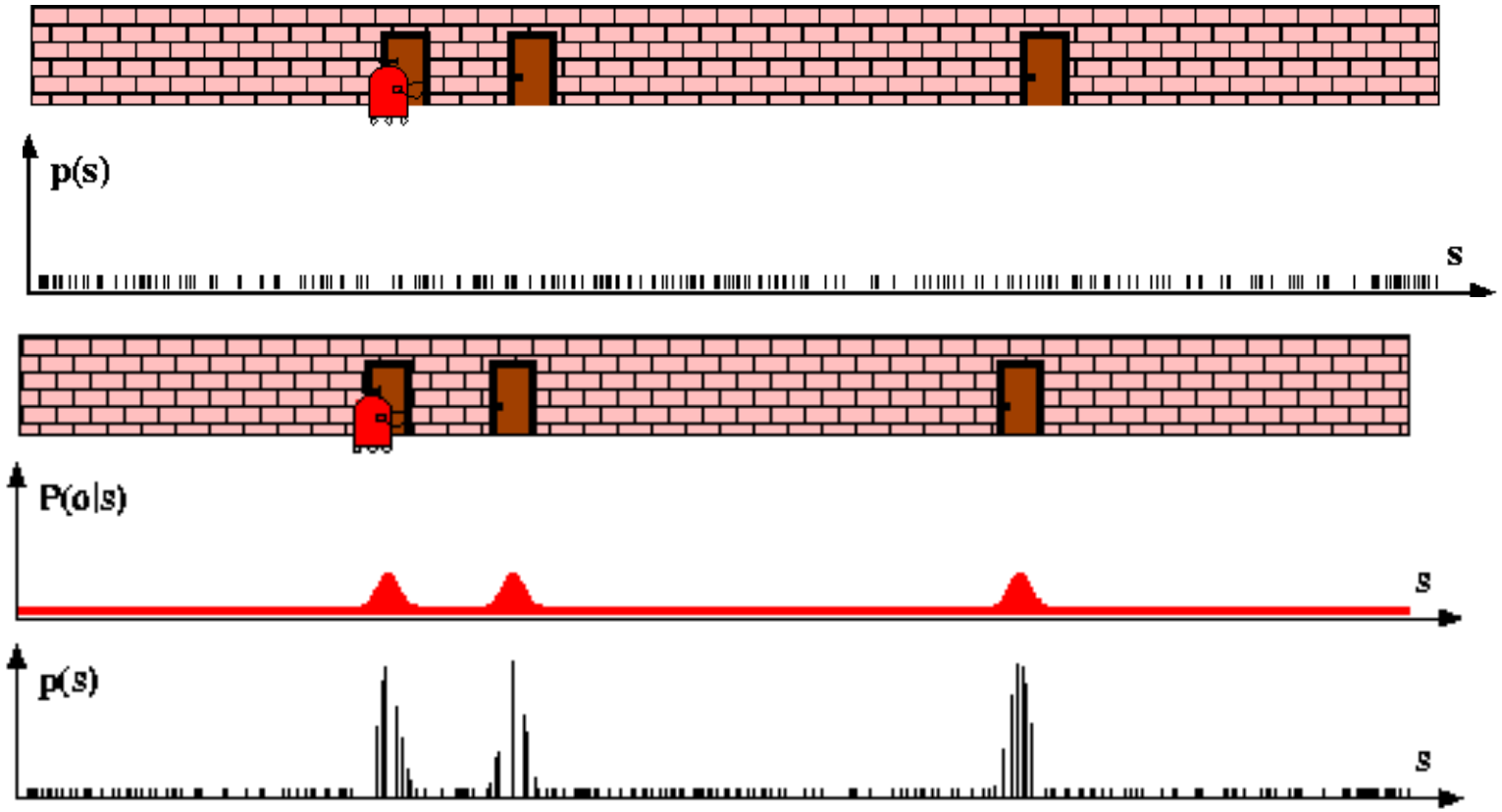
Can compute a per sample importance weight $$w_i = \frac{P(z_t|x_t^i)}{\sum_j P(z_t|x_t^j)}$$

Distribution can be represented as a set of weighted samples

Sampling

Can compute a weighted set of samples by weighting by (normalized) evidence

$$Bel(x_t) = \eta P(z_t|x_t)\overline{Bel}(x_t) \qquad w_i = \frac{P(z_t|x_t^i)}{\sum_j P(z_t|x_t^j)}$$

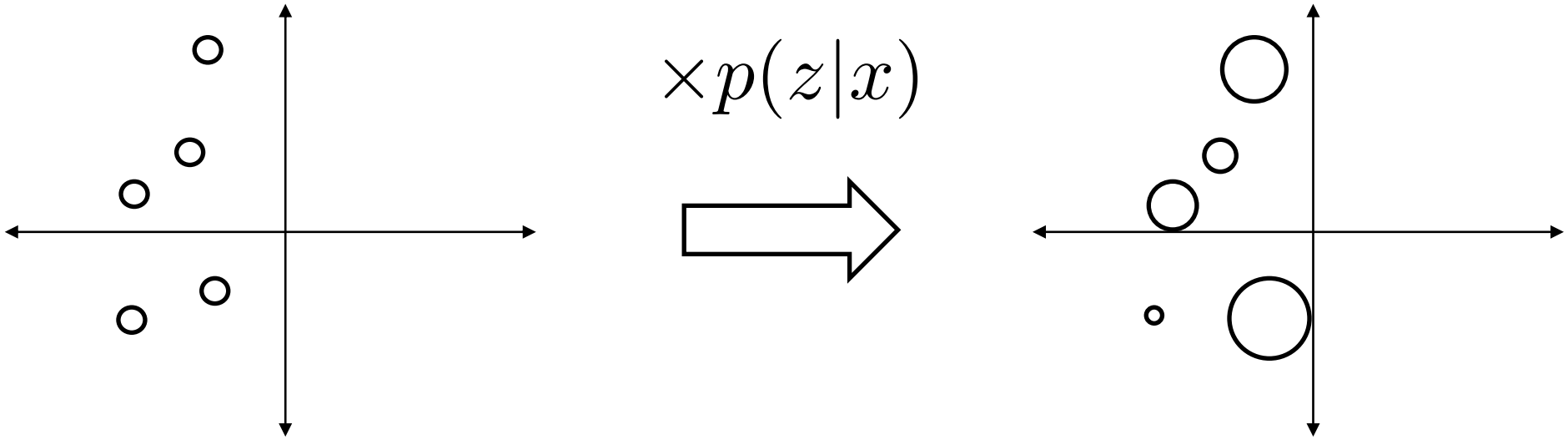


Measurement Update

$$Bel(x_t) = \eta P(z_t|x_t)\overline{Bel}(x_t)$$

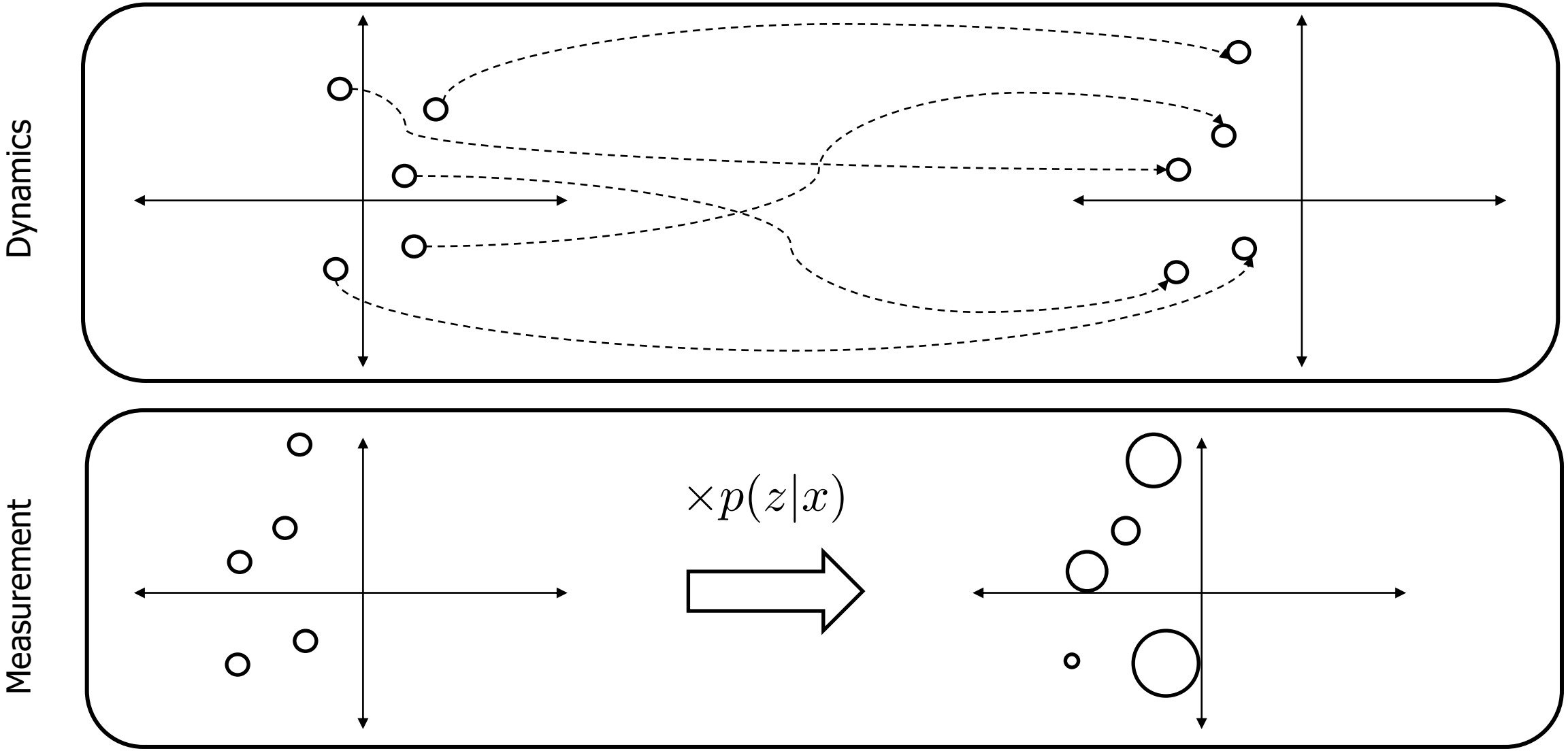
$$Bel(x_t) = \frac{P(z_t|x_t)\overline{Bel}(x_t)}{\int P(z_t|x_t)\overline{Bel}(x_t)dx_t}$$

$$w_i = \frac{P(z_t|x_t^i)}{\sum_j P(z_t|x_t^j)}$$



Reweight particles according to measurement likelihood

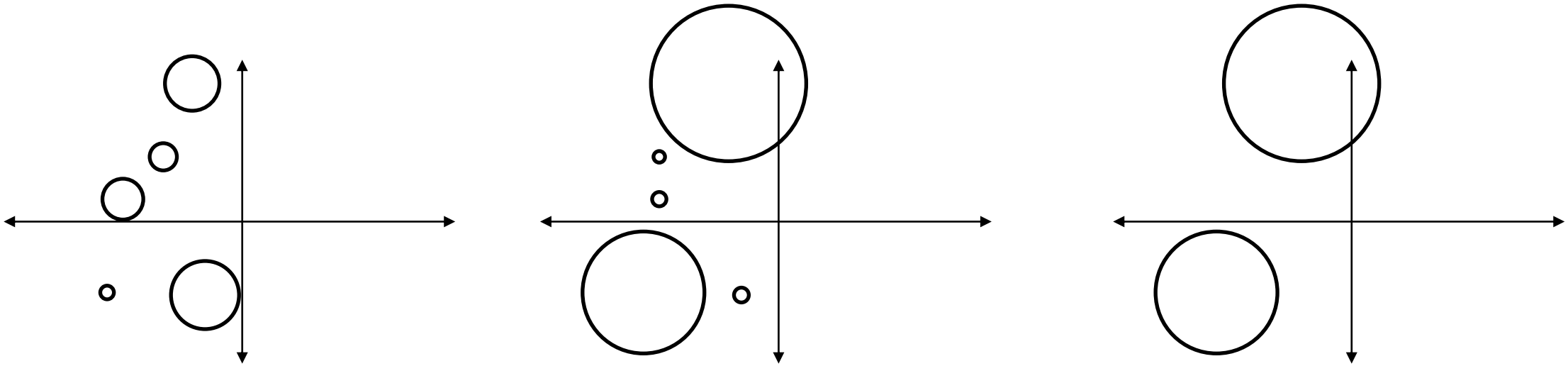
What happens across multiple steps?



Importance weights get multiplied at each step

Why might this be bad?

Importance weights get multiplied at each step



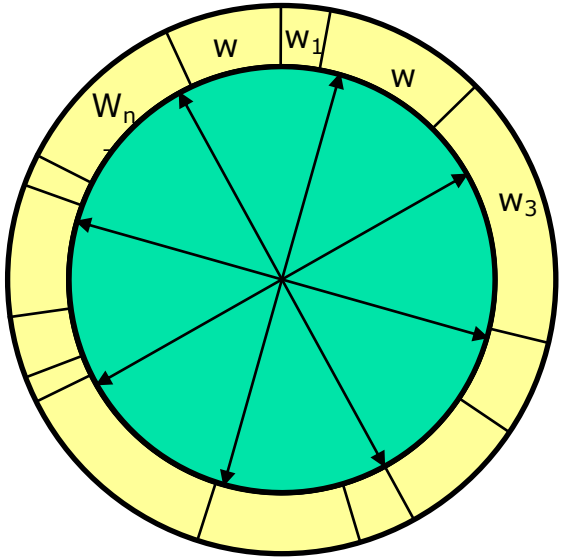
1. May blow up and get numerically unstable over many steps
2. Evidence doesn't affect samples themselves, just weights

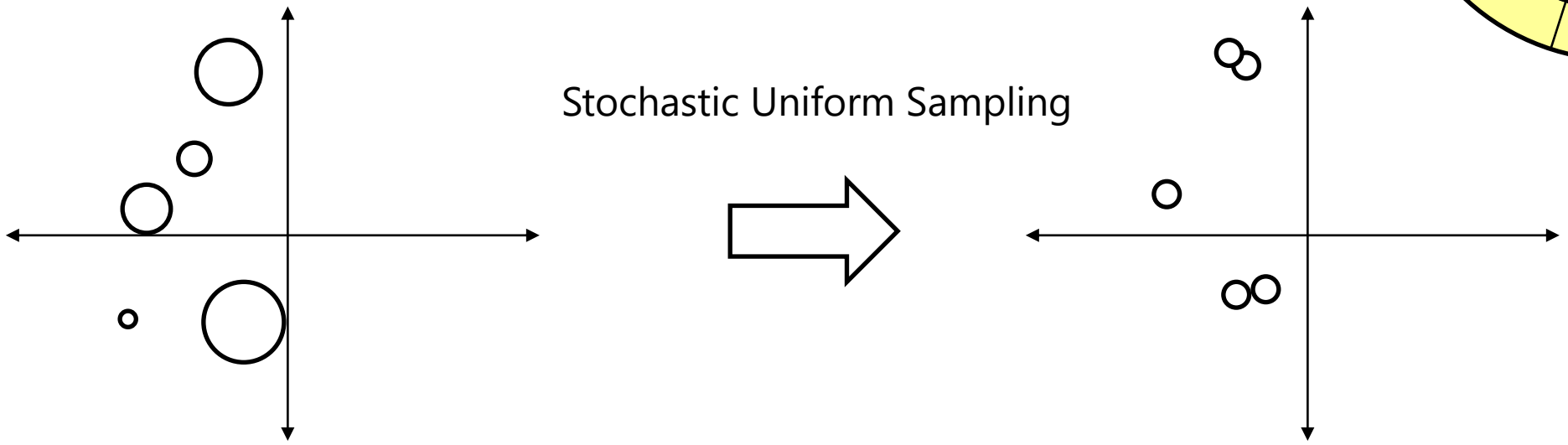
Measurement Update: Resampling

$$Bel(x_t) = \eta P(z_t|x_t)\overline{Bel}(x_t)$$

$$Bel(x_t) = \frac{P(z_t|x_t)\overline{Bel}(x_t)}{\int P(z_t|x_t)\overline{Bel}(x_t)dx_t}$$

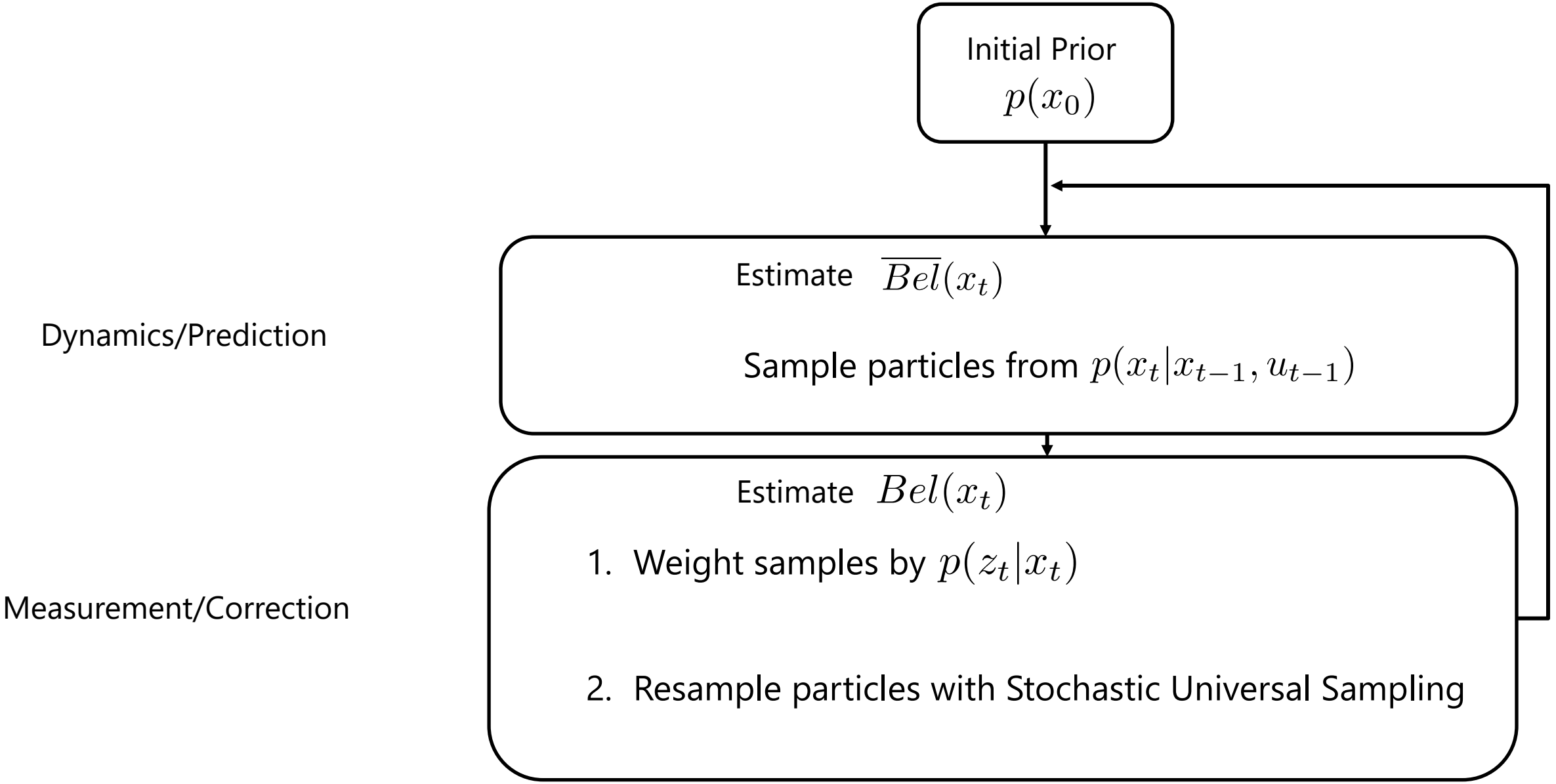
$$w_i = \frac{P(z_t|x_t^i)}{\sum_j P(z_t|x_t^j)}$$





Resample particles from weighted distribution with SUS

Overall Particle Filter algorithm



Particle Filter Algorithm

$$Bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

draw x^i_{t-1} from $Bel(x_{t-1})$

draw x^i_t from $p(x_t \mid x^i_{t-1}, u_{t-1})$

Importance factor for x^i_t:

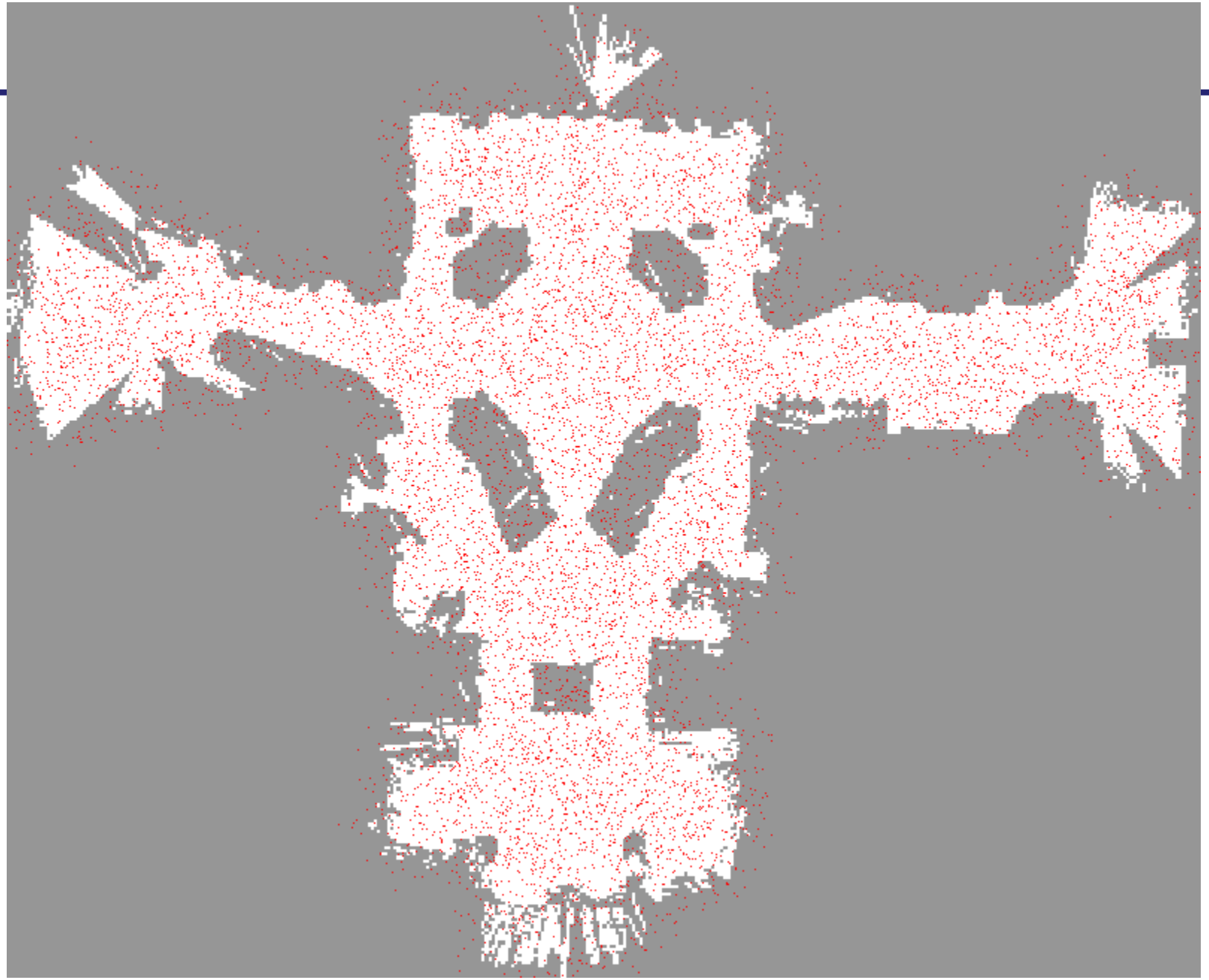
$$w^i_t = \frac{\text{target distribution}}{\text{proposal distribution}}$$

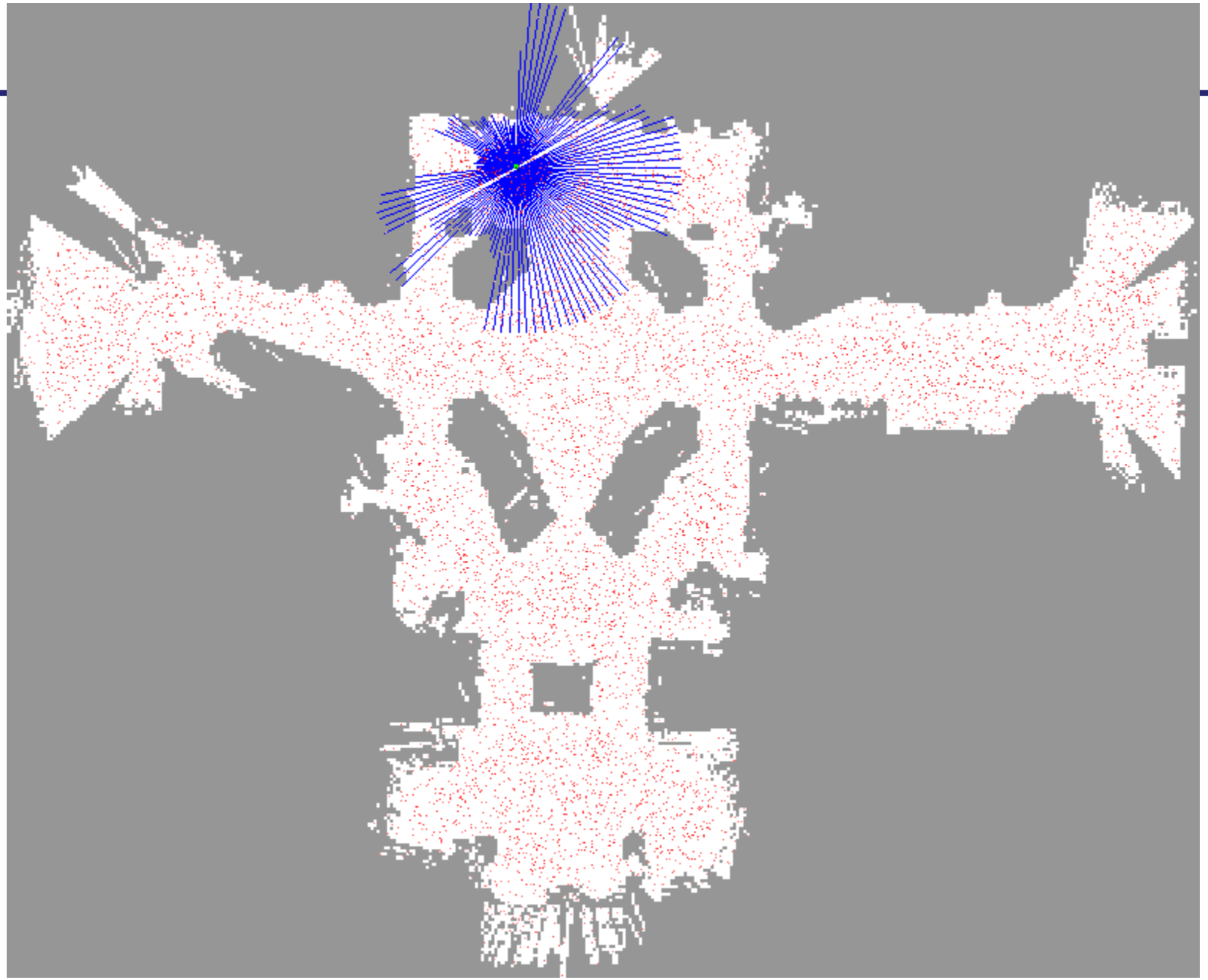
$$= \frac{\eta\; p(z_t \mid x_t)\; p(x_t \mid x_{t-1}, u_{t-1})\; Bel(x_{t-1})}{p(x_t \mid x_{t-1}, u_{t-1})\; Bel(x_{t-1})}$$

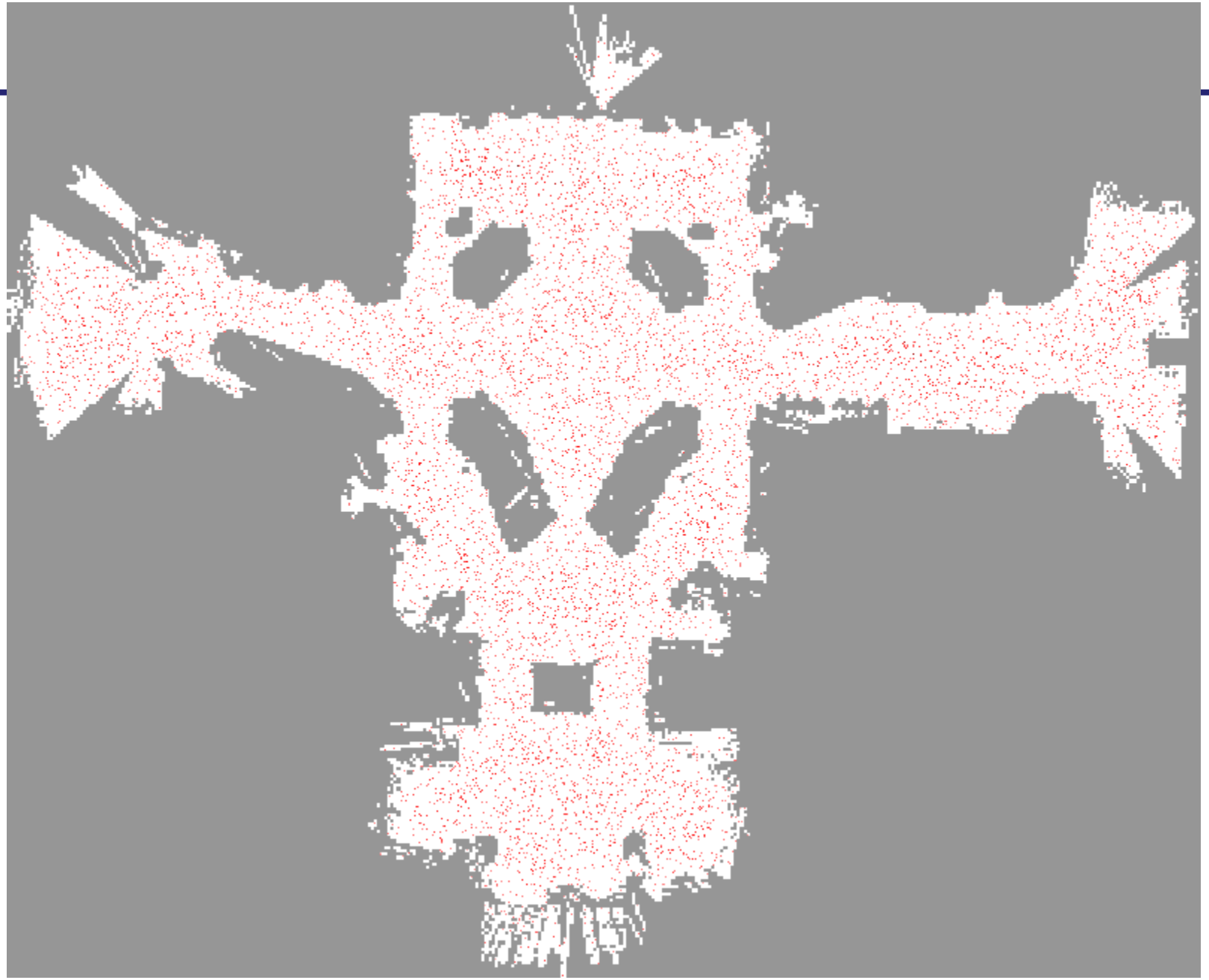
$$\propto p(z_t \mid x_t)$$

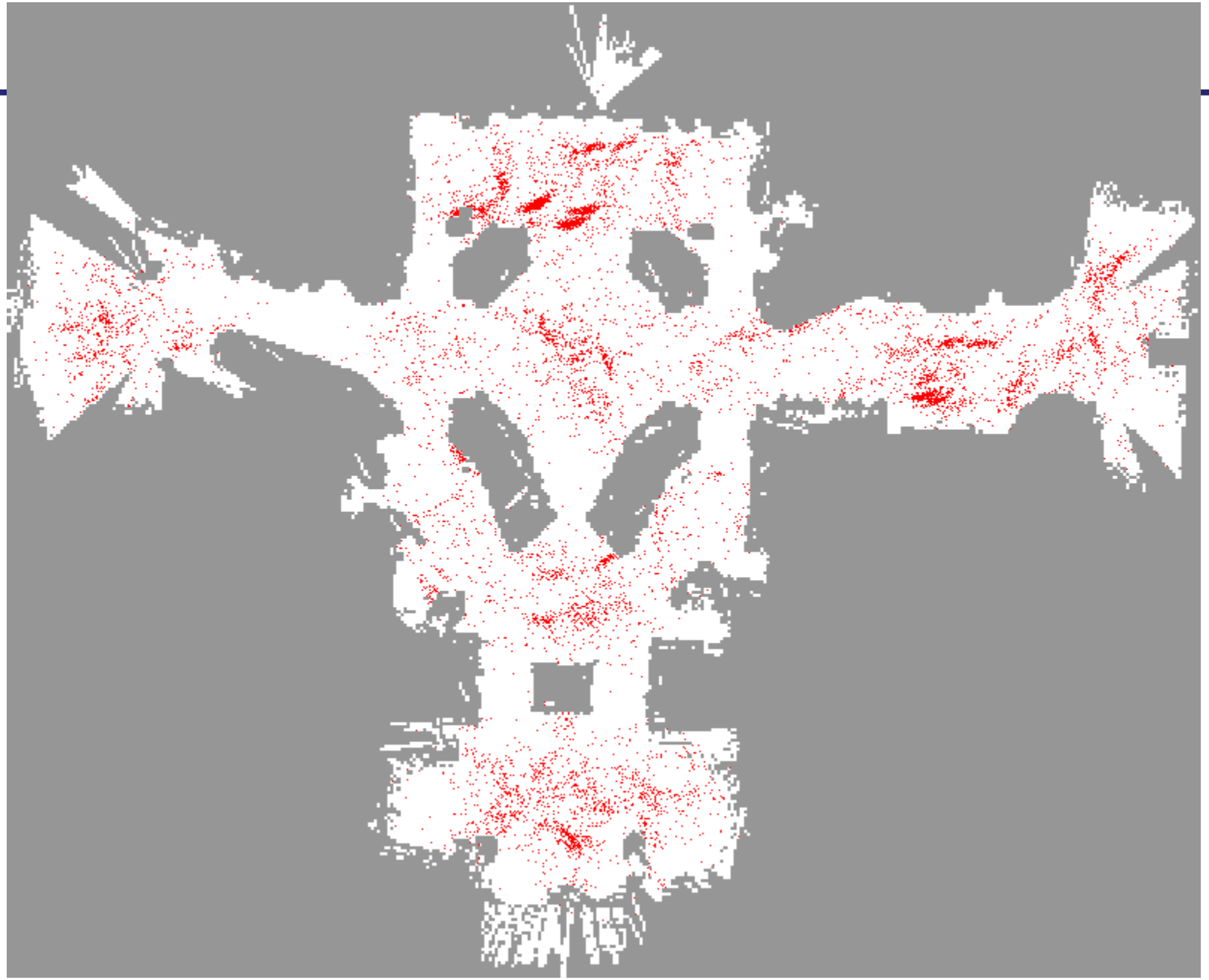
Particle Filter Algorithm

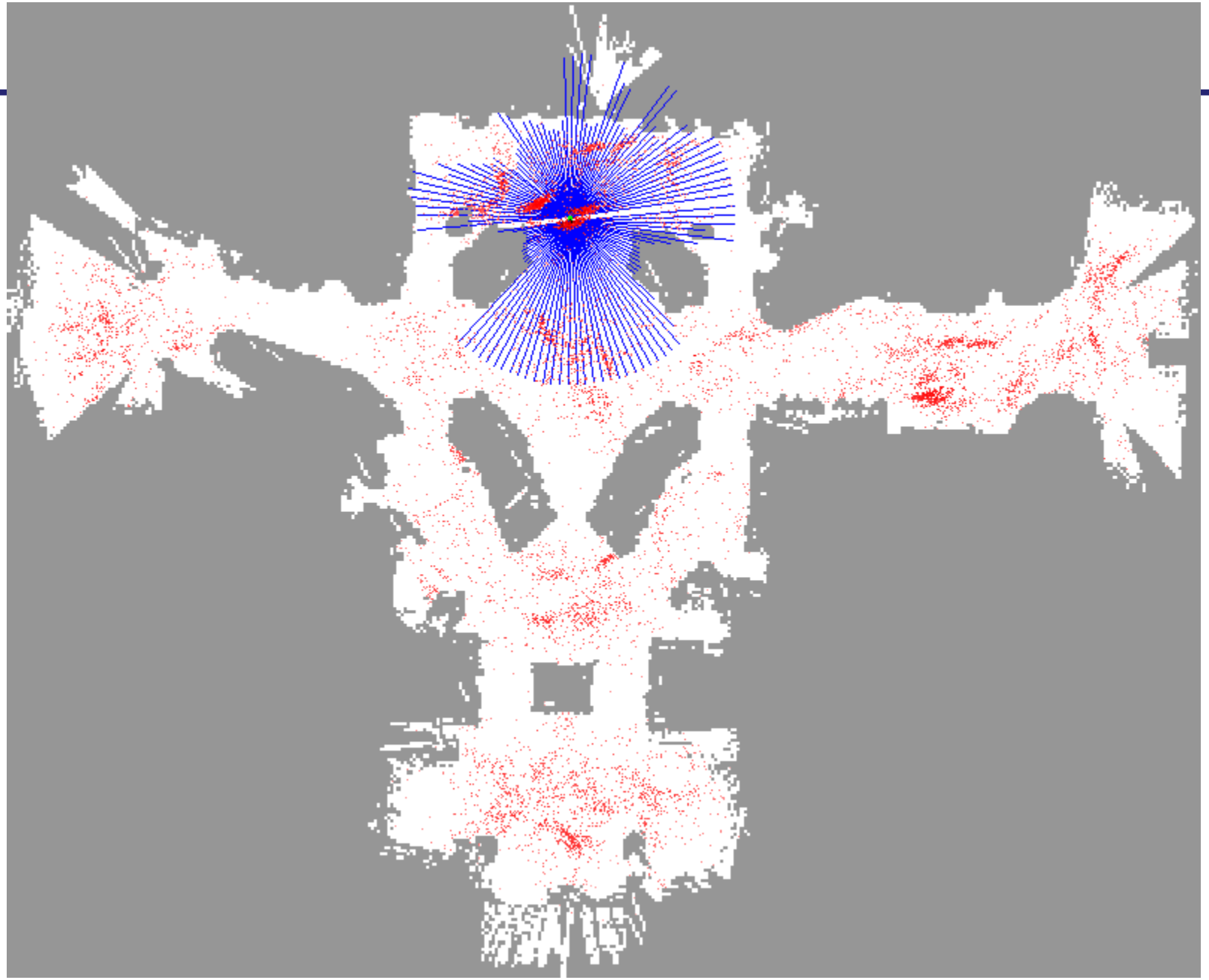
1. Algorithm **particle_filter**(S_{t-1}, u_{t-1}, z_t):
2. $S_t = \emptyset, \eta = 0$
3. **For** $i = 1 \ldots n$
4. Sample $\bar{x}_t^i$ from $p(\bar{x}_t | x_{t-1}, u_{t-1})$ using x_{t-1}^i and u_{t-1} ***Dynamics***
5. $w_t^i = p\left(z_t \middle| \bar{x}_t^i\right)$ ***Compute importance weight***
6. $\eta = \eta + w_t^i$ ***Update normalization factor***
7. $S_t = S_t \cup \{< \bar{x}_t^i, w_t^i >\}$ ***Insert***
8. **For** $i = 1 \ldots n$
9. $w_t^i = w_t^i / \eta$ ***Normalize weights***
10. **For** $i = 1 \ldots n$ ***Generate new samples***
11. Sample x_t^i from the discrete distribution given by $\bar{x}_t^i, w_t^i$

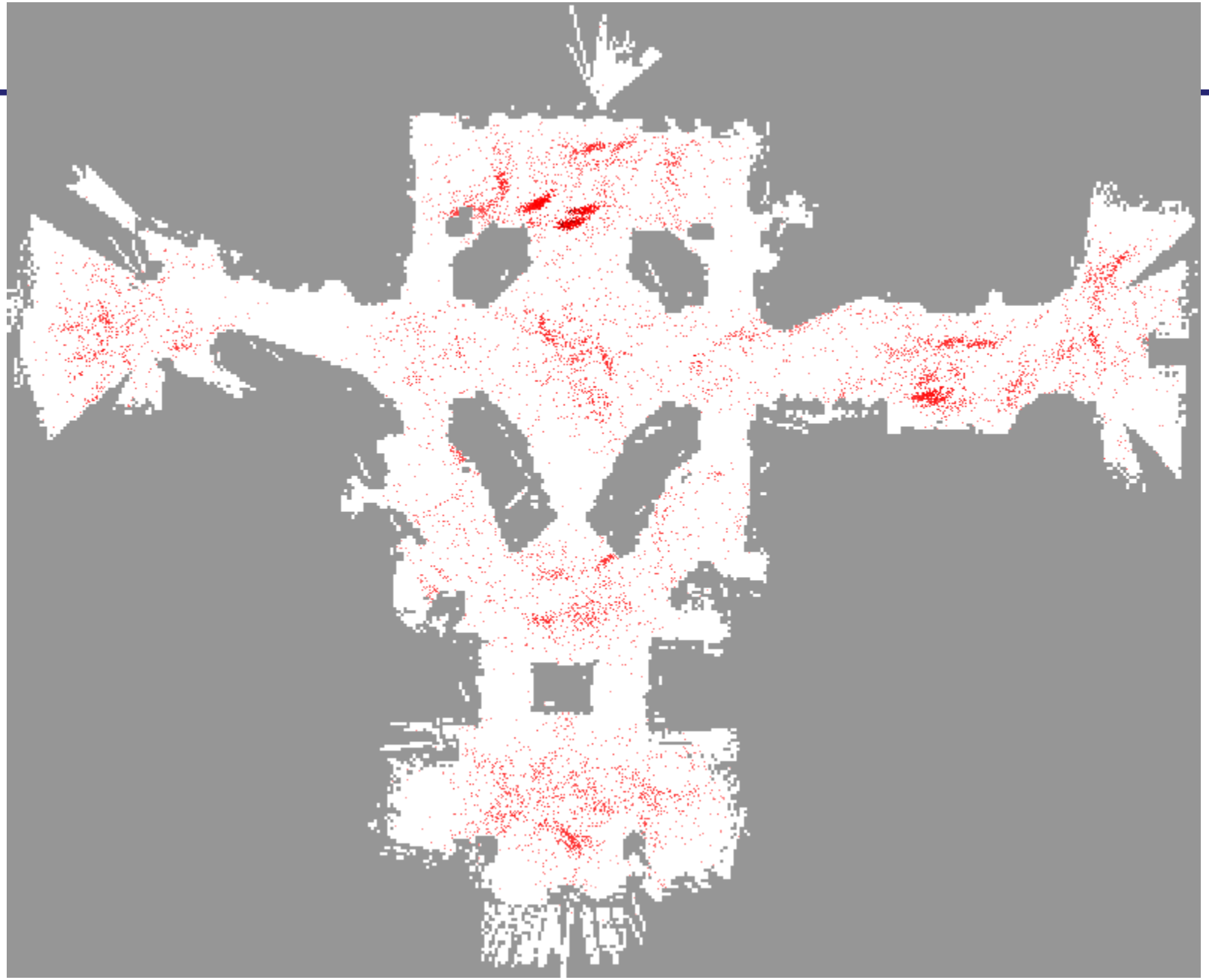


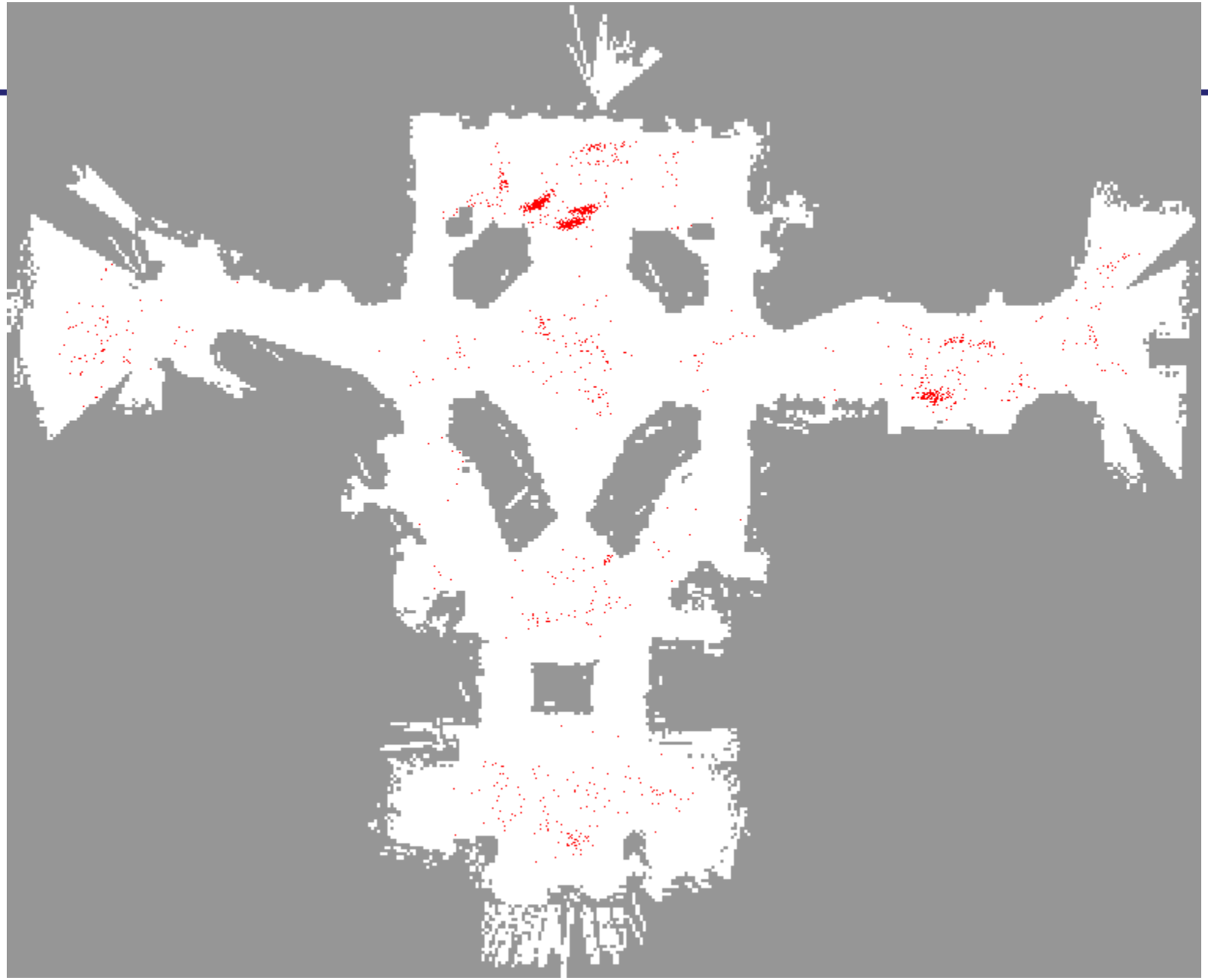


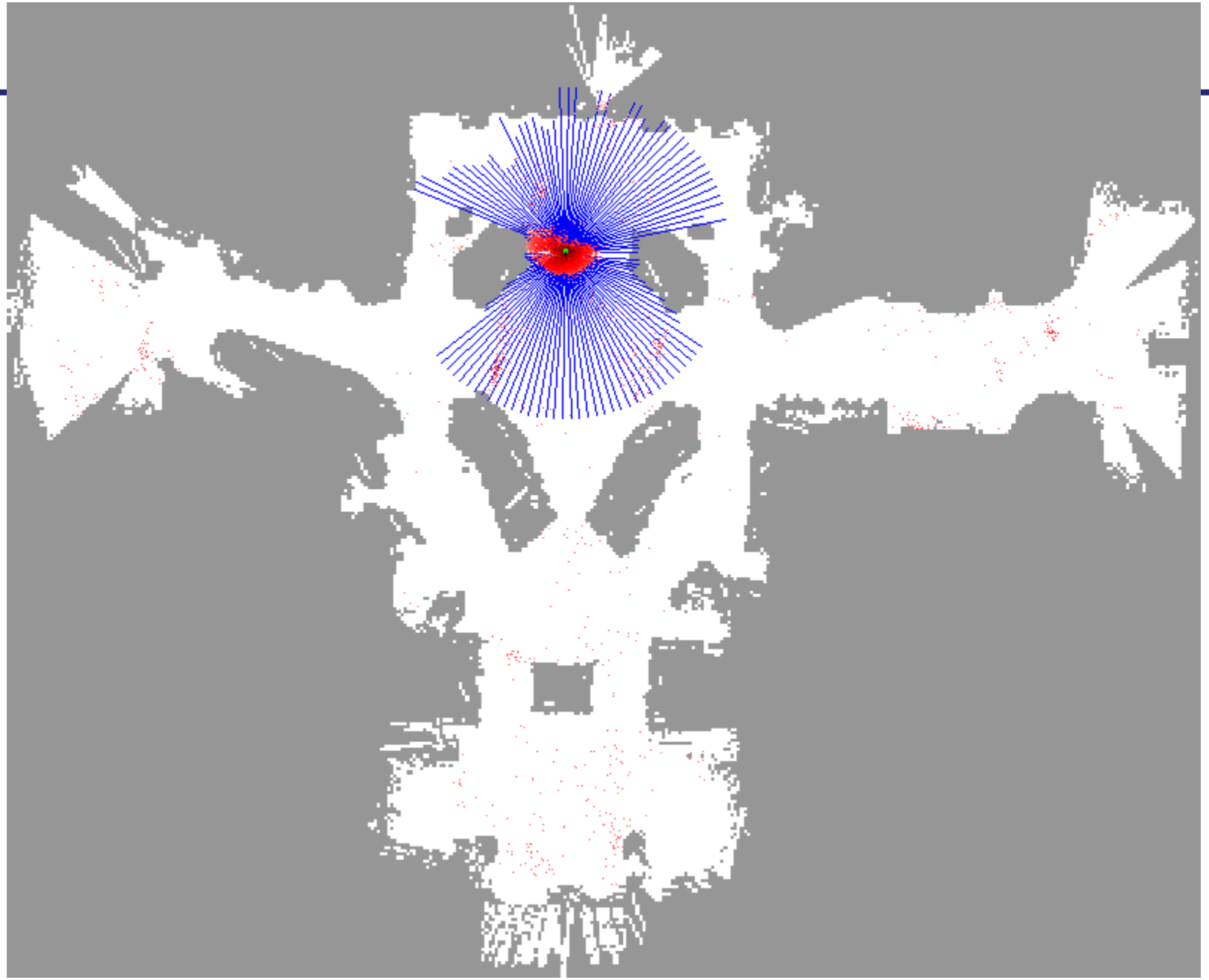


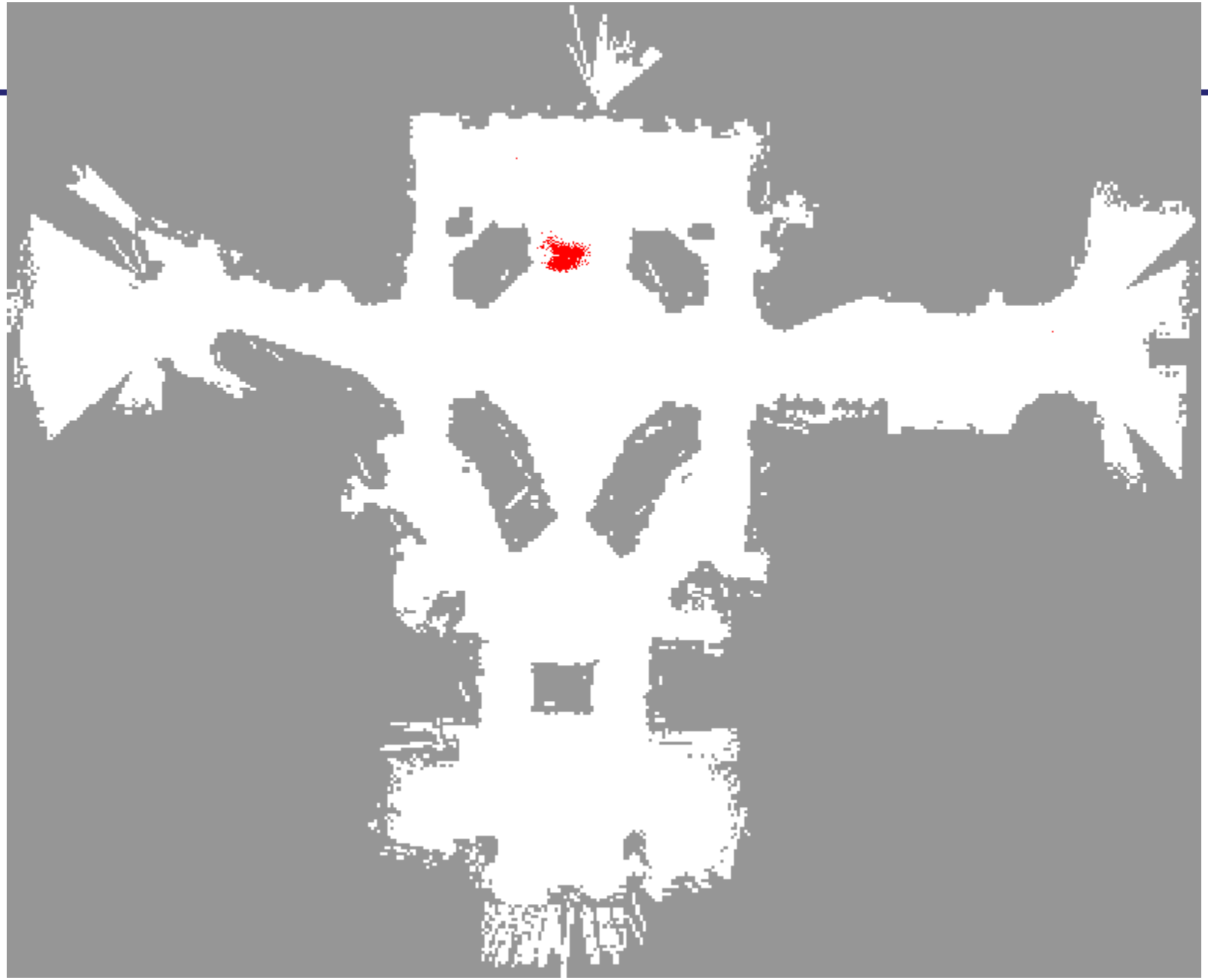


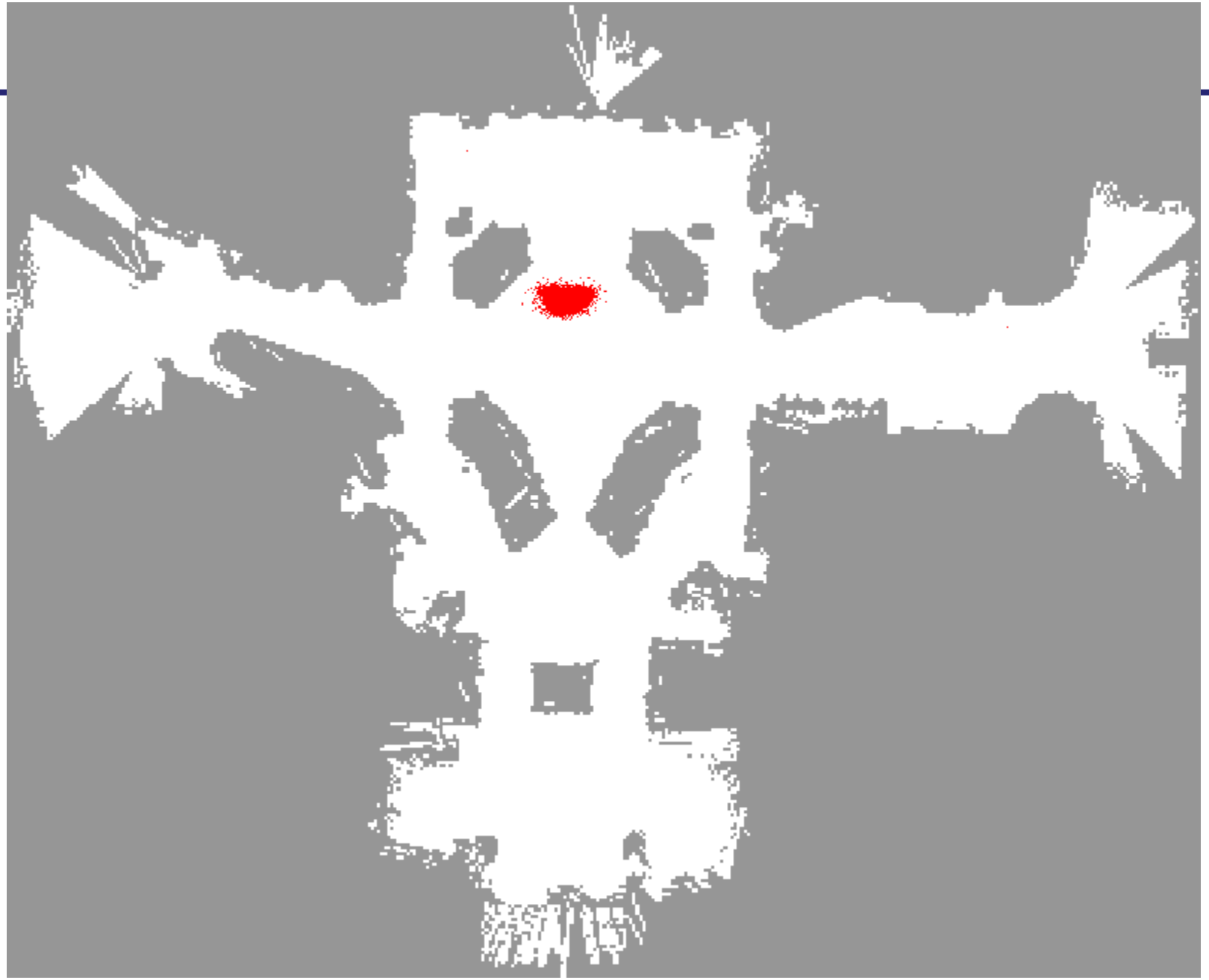


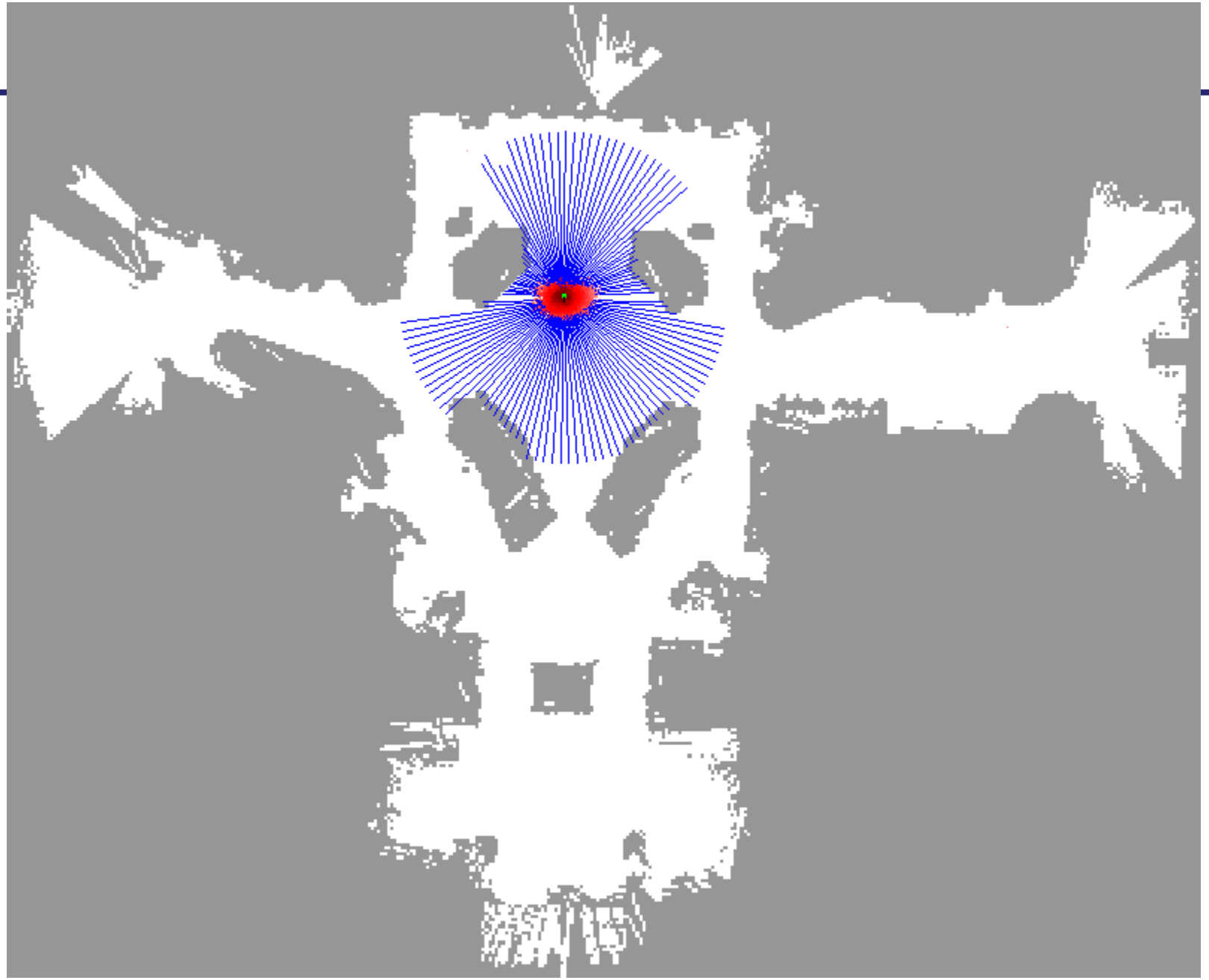


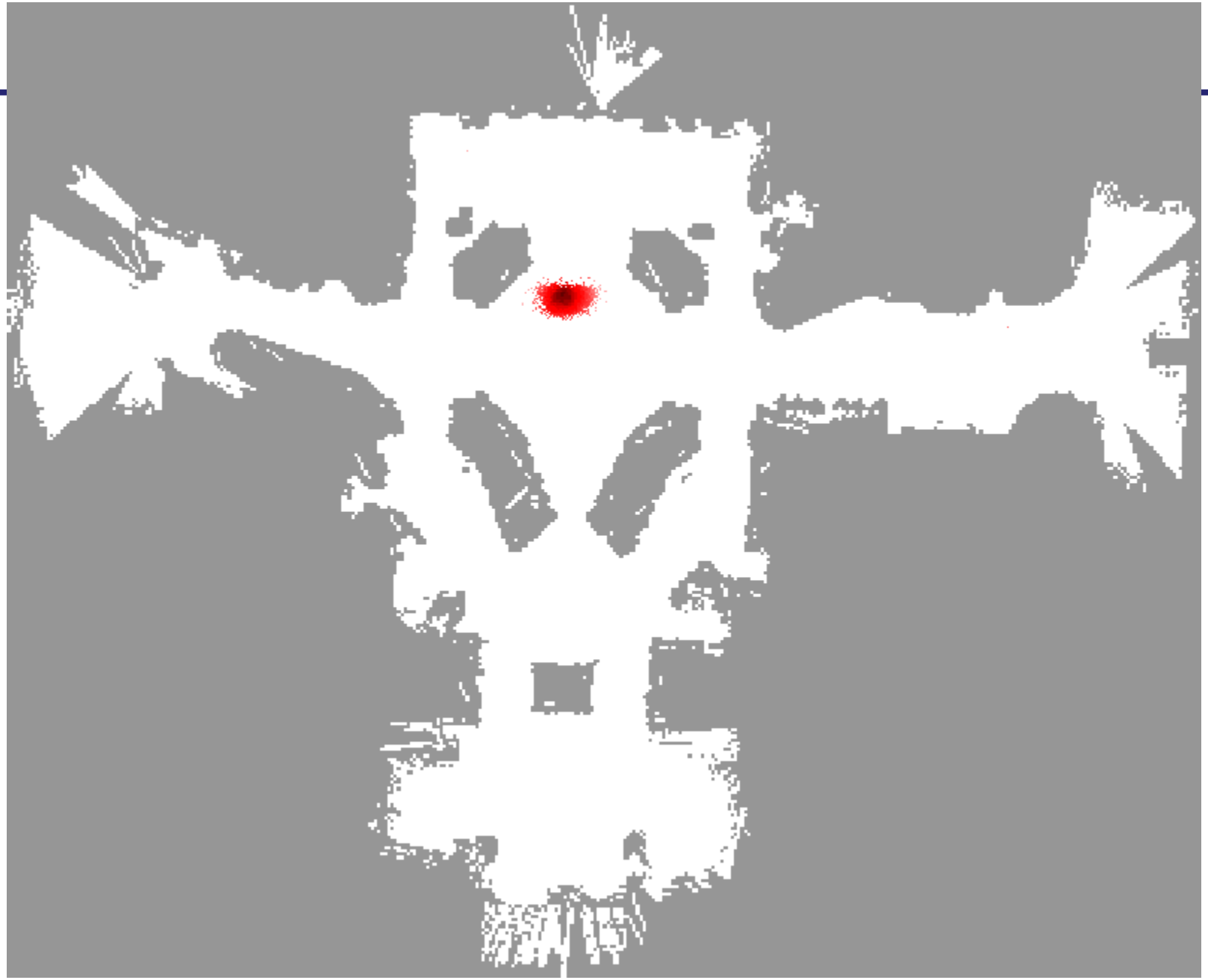


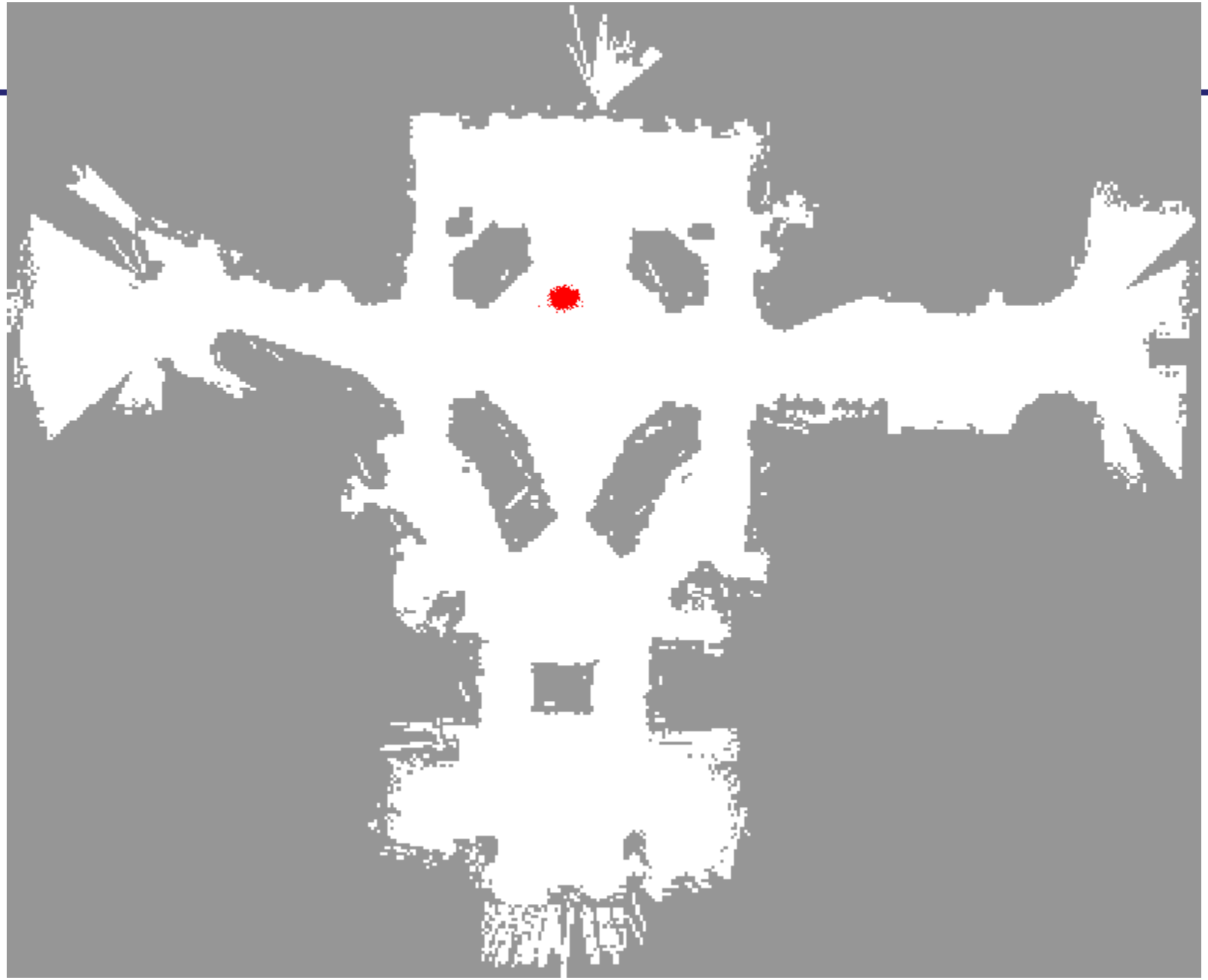


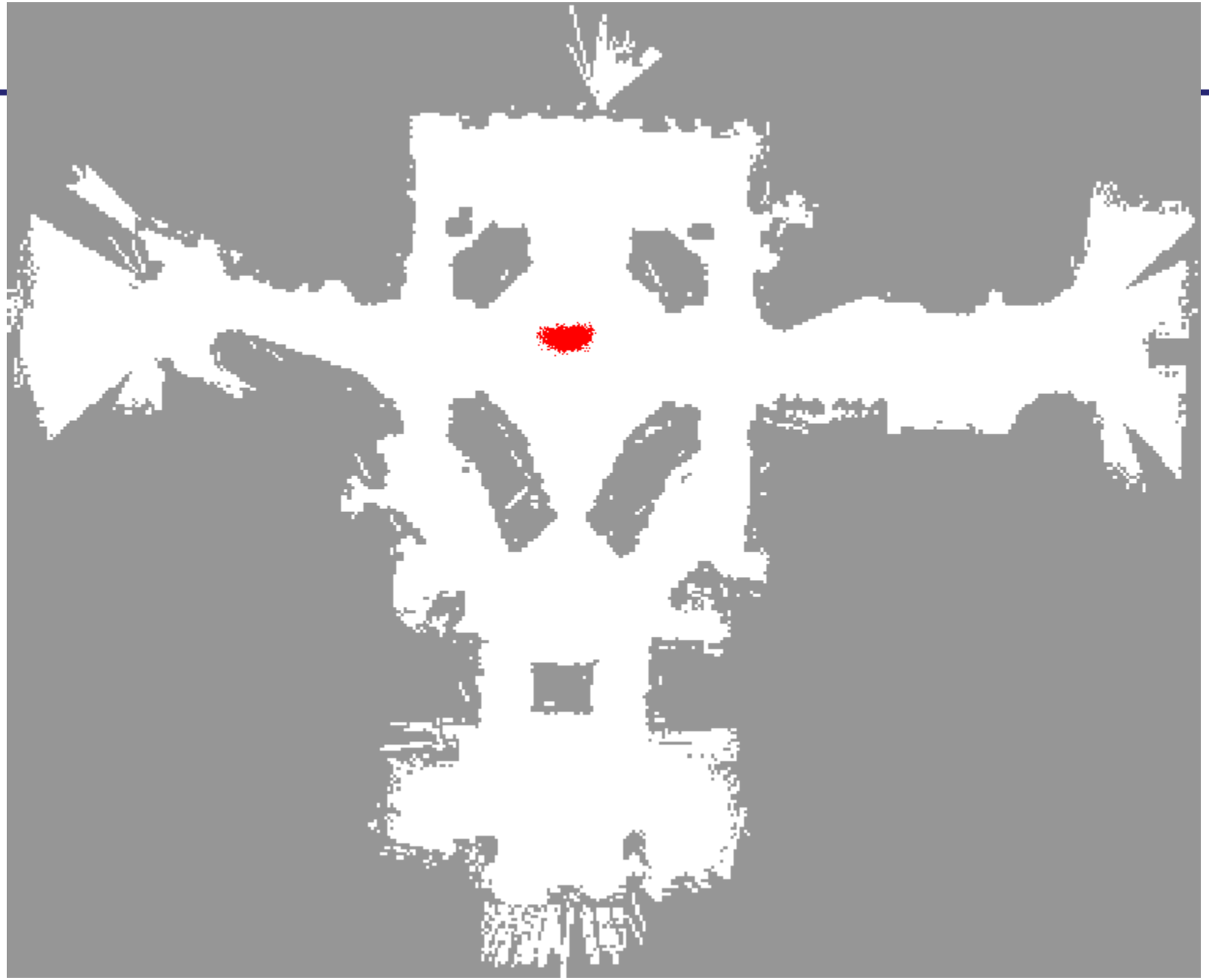


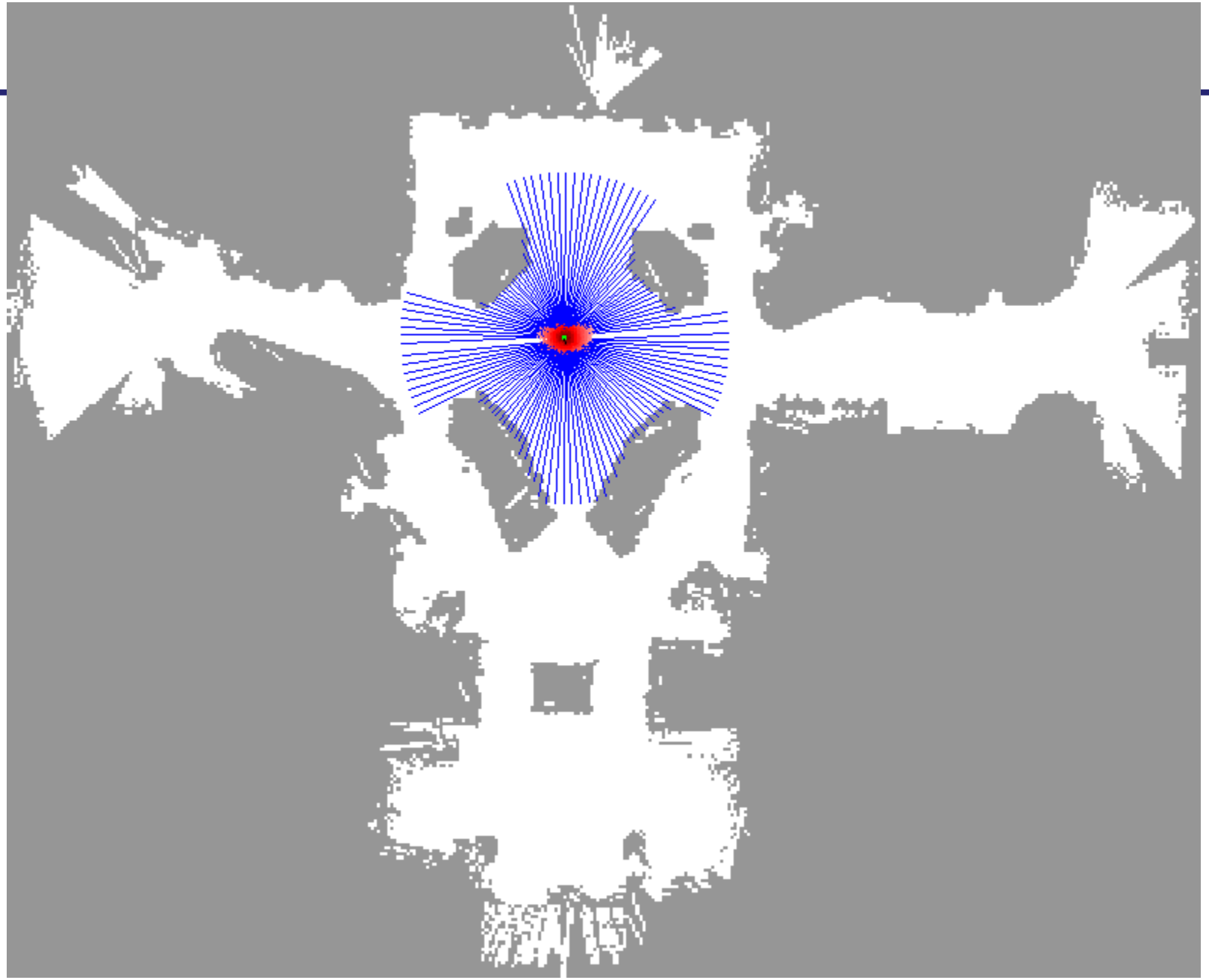


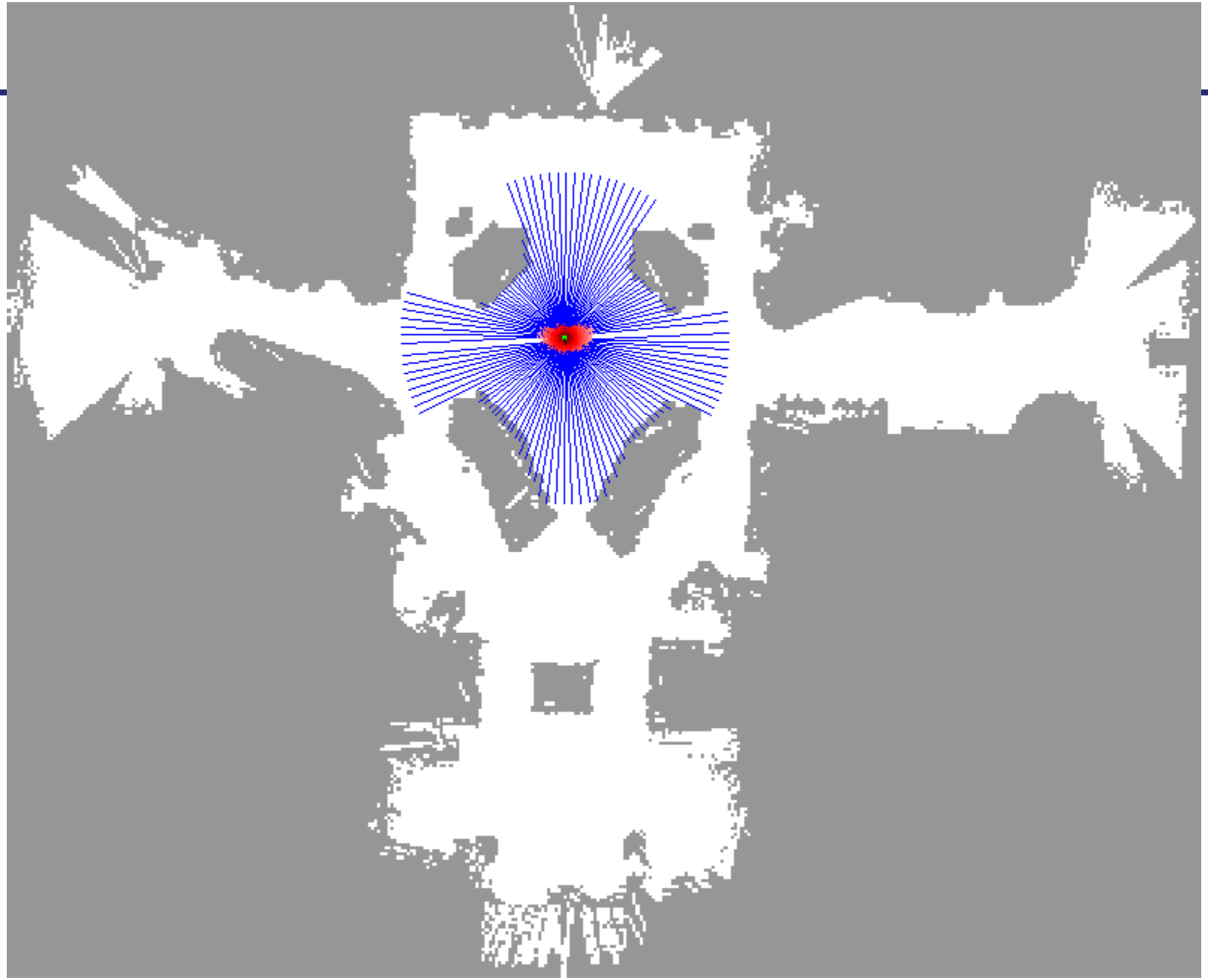




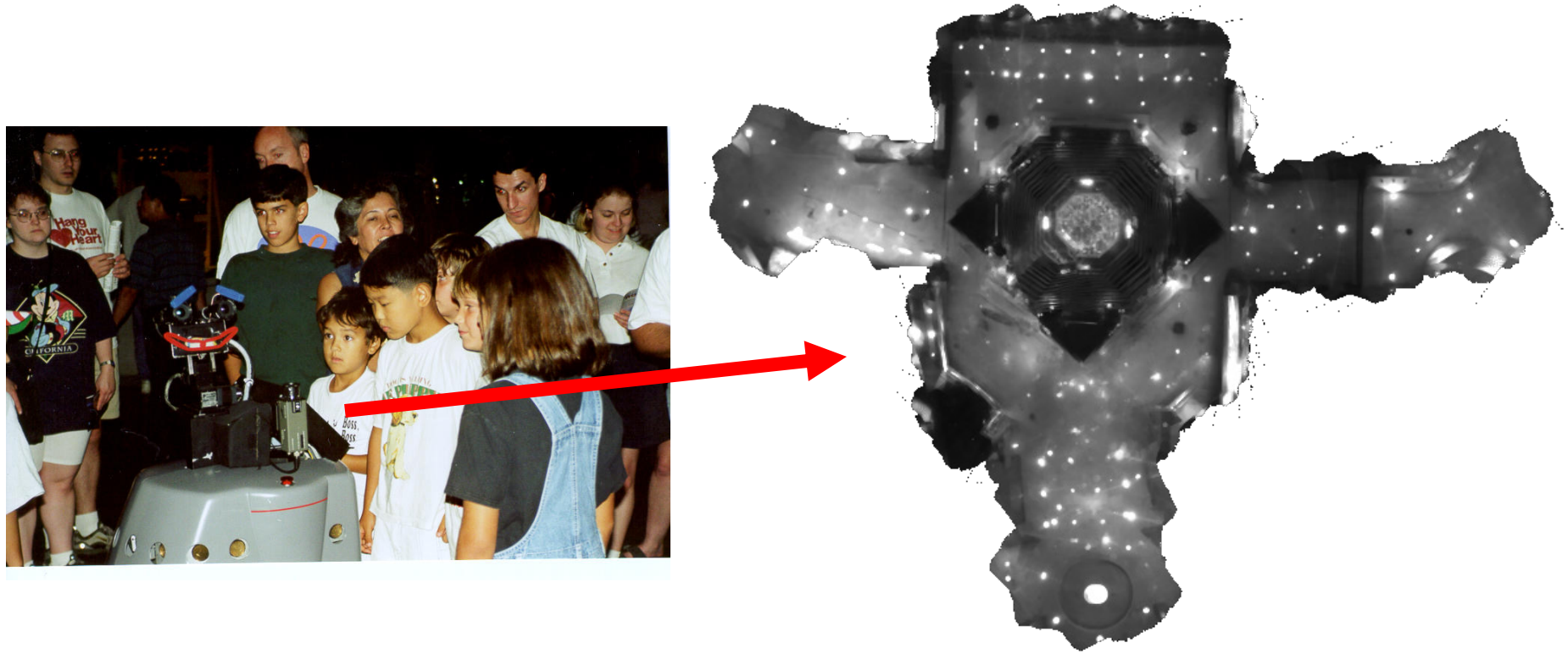






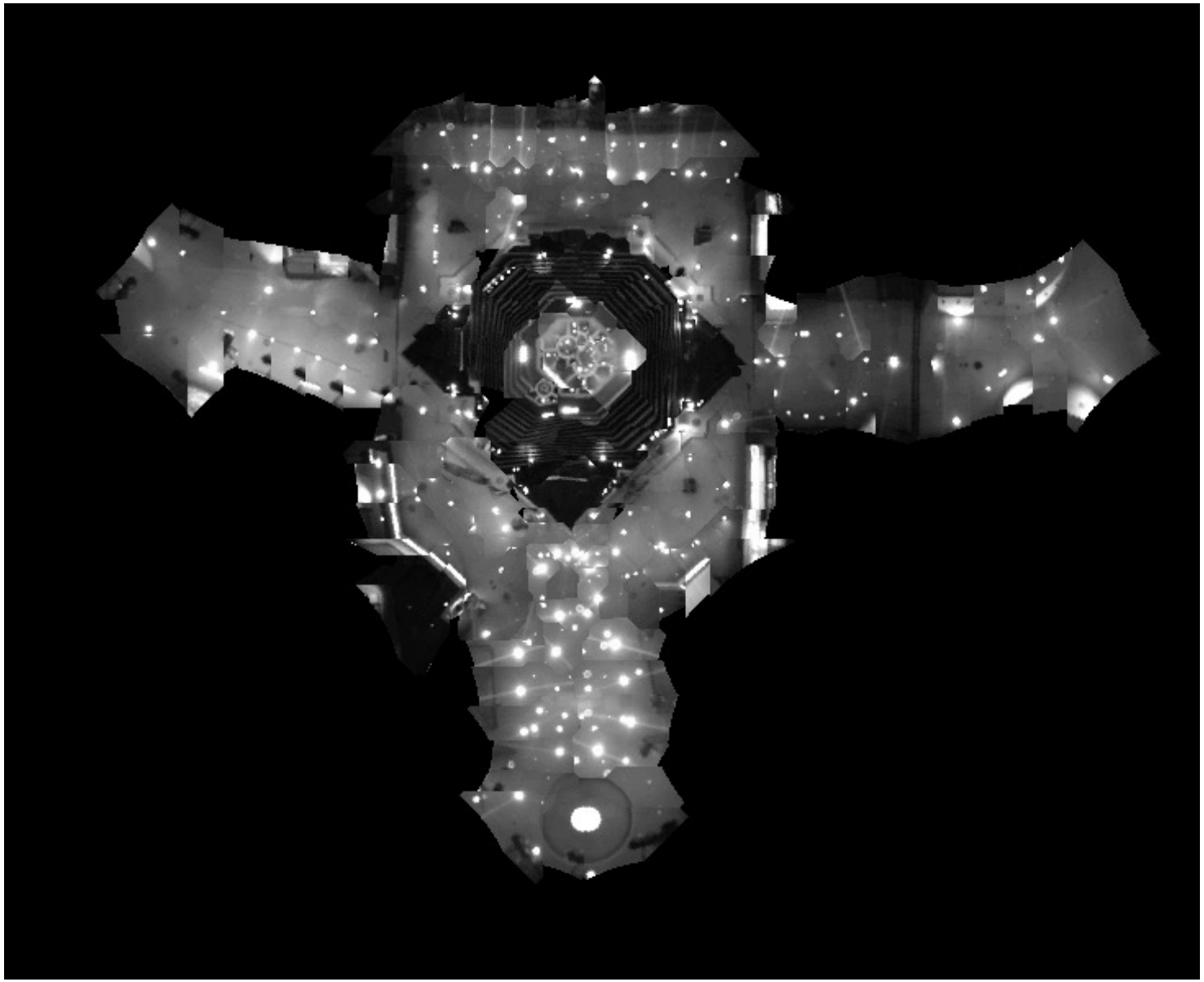


Using Ceiling Maps for Localization

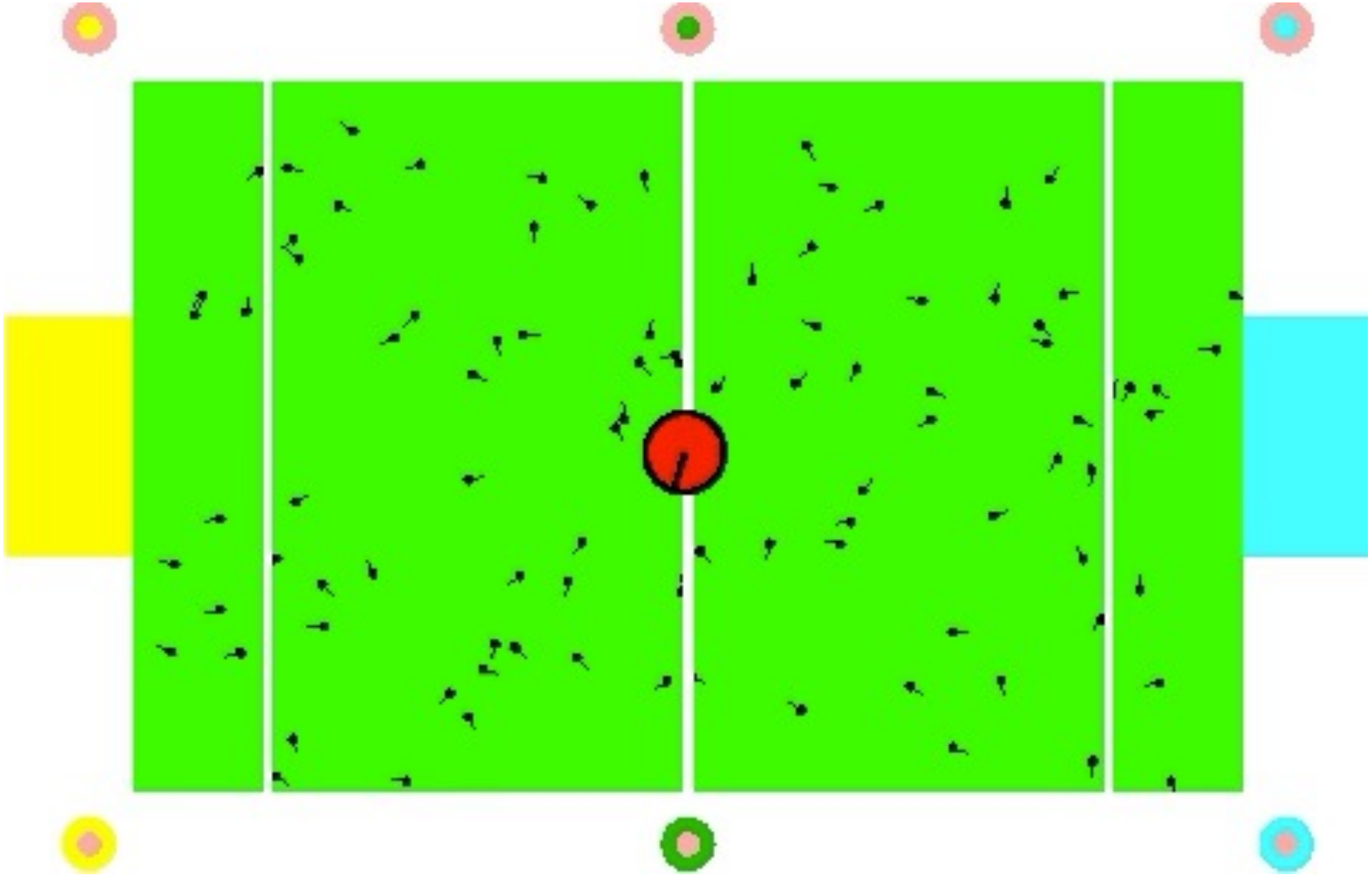


[Dellaert et al. 99]

Global Localization Using Vision



Localization for AIBO robots



When might the particle filter fail?

Why might this not work?

- Finitely many samples → introduces bias
- Variance of resampling operation → drops diversity
- Divergence of proposal and target distributions → degenerate importance weights
- Particle deprivation → belief collapse

Read Dieter Fox's papers!

Lecture Outline

Particle Filters

Motion Models

Sensor Models

Kalman Filtering

$$x_{t+1} = Ax_t + Bu_t + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$
$$z_{t+1} = Cx_{t+1} + \delta_t$$
$$\delta_t \sim \mathcal{N}(0, R)$$

Initial Prior

$p(x_0)$

Dynamics/Prediction
(given some u)

Estimate $\overline{Bel}(x_t)$

$$p(x_{t+1}|z_{0:t}, u_{0:t+1}) \sim \mathcal{N}(A\mu_{t|0:t} + Bu_t, A\Sigma_{t|0:t}A^T + Q_t)$$

Measurement/Correction
(given some z)

Estimate $Bel(x_t)$

$$p(x_{t+1}|z_{0:t+1}, u_{0:t})$$
$$= \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - C\mu_{t+1|0:t}), (I - K_{t+1}C)\Sigma_{t+1|0:t})$$

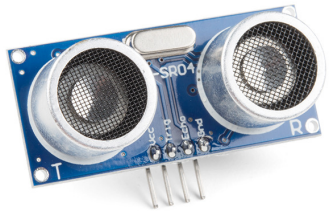
How do we instantiate this?

Linear Gaussian

$$x_{t+1} = Ax_t + Bu_t + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$
$$z_{t+1} = Cx_{t+1} + \delta_t$$
$$\delta_t \sim \mathcal{N}(0, R)$$

Non-linear

$$x_{t+1} = g(x_t, u_t) + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$
$$z_t = h(x_t) + \delta_t$$
$$\delta_t \sim \mathcal{N}(0, R)$$



How do we instantiate concretely?

Motion models and sensor models!

Types of Motion Models

Let's focus exclusively on mobile robot navigation

State of the system:

Global position $\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$

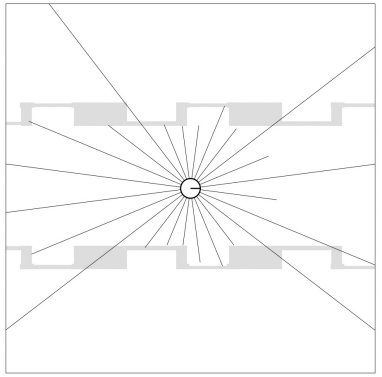
Actions:

Rotational and translational velocity

Translation / Rotation $\begin{pmatrix} \nu \\ \omega \end{pmatrix}$

+ more

Measurements



Sonar/ultrasound/LIDAR

More on this later

Velocity Based Model

Velocity Based Model

State of the system:

Global position $\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$

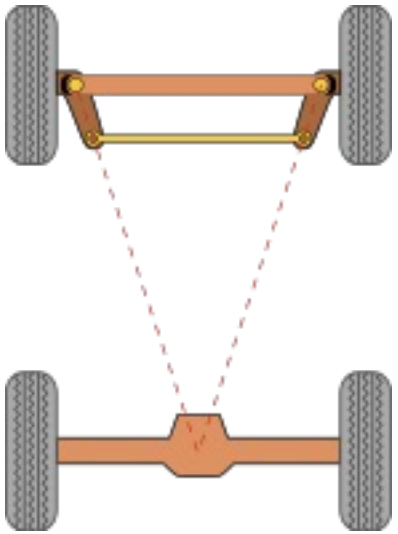
Actions:

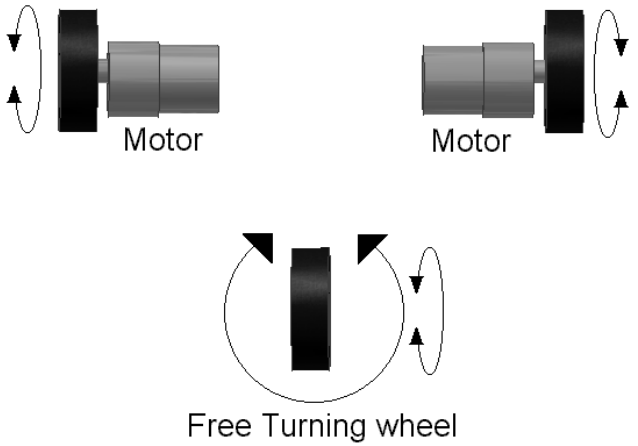
Rotational and translational velocity

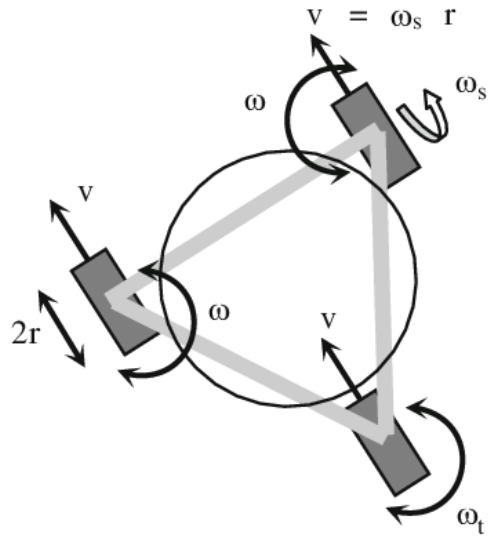
Translation ν

Rotation ω

Noise independently sampled per dimension

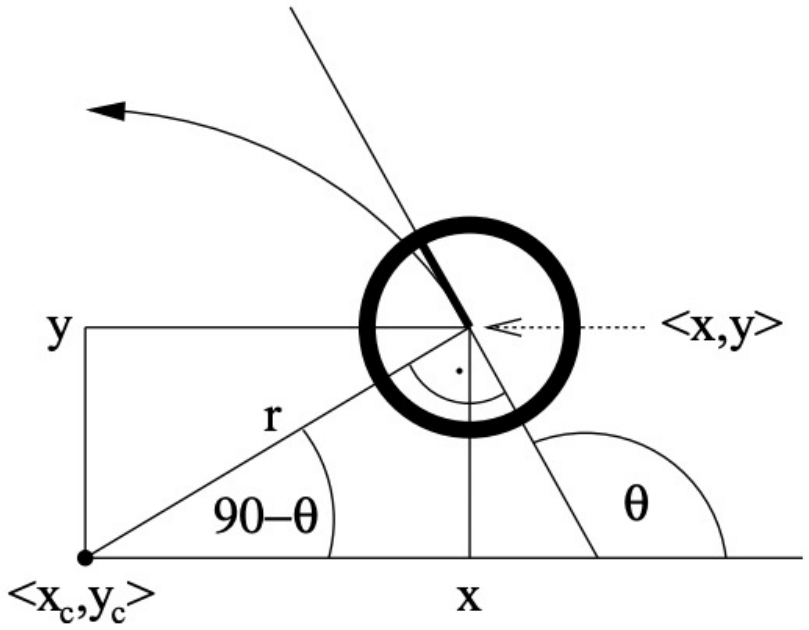






Velocity Based Sampling

Generate noise free motion and then add noise to it



Useful for particle filters

Given ν, ω first compute the radius of motion to get x, y and then compute the heading change

$$x_c = x_t - r\sin\theta \qquad r = \frac{\nu}{\omega}$$
$$y_c = y_t + r\cos\theta$$

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta+\omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta+\omega\Delta t) \\ \theta+\omega\Delta t \end{pmatrix}$$
$$= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta+\omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta+\omega\Delta t) \\ \omega\Delta t \end{pmatrix}$$

Add noise to the velocities

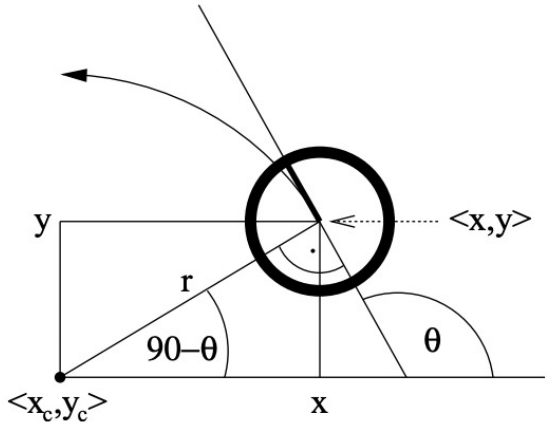
$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1|v|+\alpha_2|\omega|} \\ \varepsilon_{\alpha_3|v|+\alpha_4|\omega|} \end{pmatrix}$$

Replace ν, ω by $\hat{v}, \hat{w}$

Velocity Based Likelihood

$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1|v|+\alpha_2|\omega|} \\ \varepsilon_{\alpha_3|v|+\alpha_4|\omega|} \end{pmatrix}$$

Likelihood model depends on the choice of ε



Compute center of circle

Compute arc movement

Compute expected velocities

Calculate probs by score differences

Algorithm motion_model_velocity(x_t, u_t, x_{t-1})**:**

$$\mu = \frac{1}{2}\frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$$

$$x^* = \frac{x+x'}{2} + \mu(y-y')$$

$$y^* = \frac{y+y'}{2} + \mu(x'-x)$$

$$r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2}$$

$$\Delta\theta = \text{atan2}(y'-y^*, x'-x^*) - \text{atan2}(y-y^*, x-x^*)$$

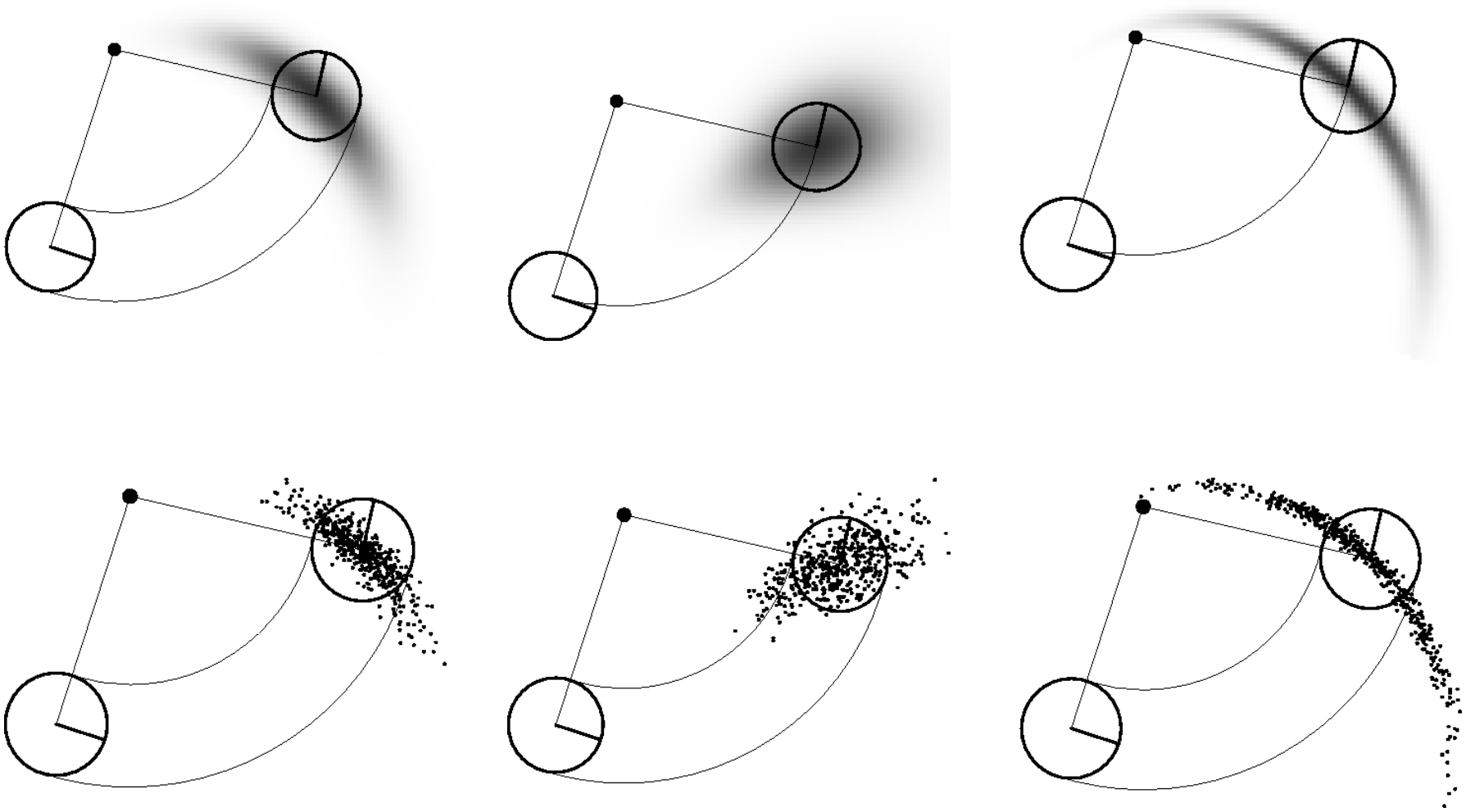
$$\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$$

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

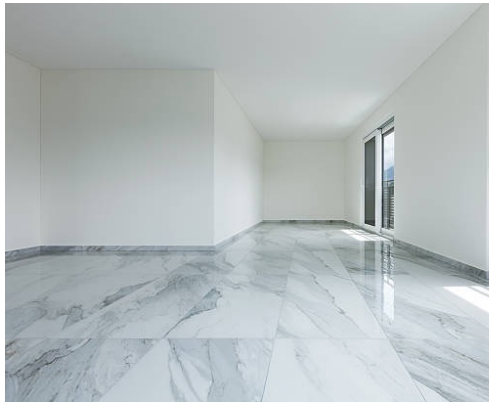
$$\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{\omega}$$

return $\textbf{prob}(v-\hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot \textbf{prob}(\omega-\hat{\omega}, \alpha_3|v| + \alpha_4|\omega|) \cdot \textbf{prob}(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

Examples (velocity based)



Why might velocity not be enough?



Open loop velocity measurement might be quite noisy in varying environments

→ What if we had more information??

Odometry Based Model

What is odometry?

- Instead of just measuring likelihoods with velocity, we can actually use more information
- Integrating wheel encoders (degrees rotated) with known wheel radius can give us estimates of position (still noisy)
  - Pros: More accurate than velocity
  - Cons: Only available after motion

Odometry Based Model: Reparameterization

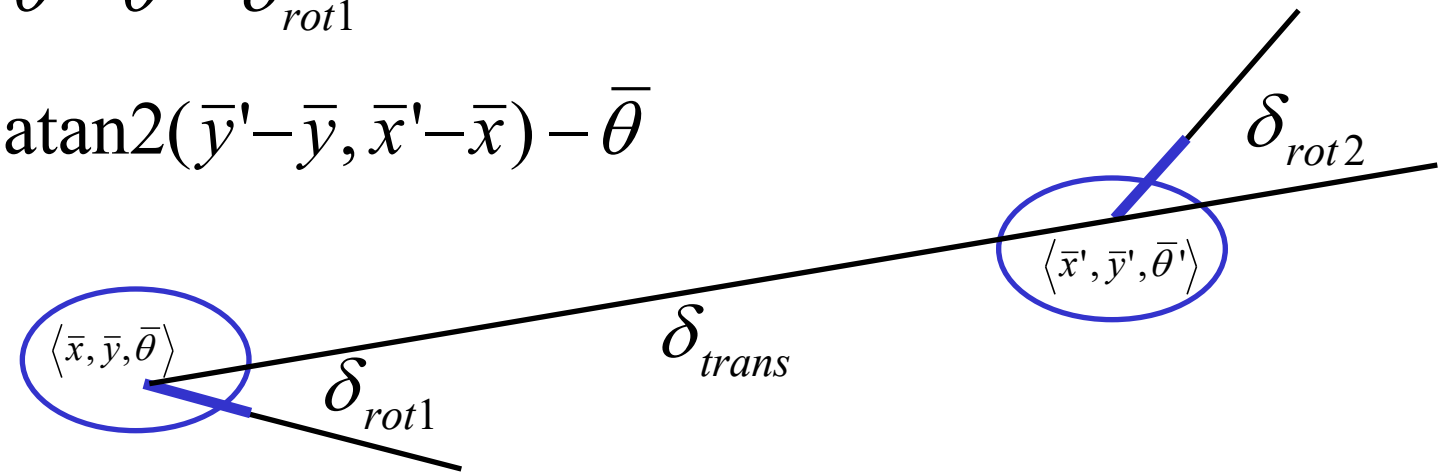
- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$ - this is input action u
- Reparametrize odometry information reparameterized $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$

.

$$\delta_{trans} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2}$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$

$$\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$

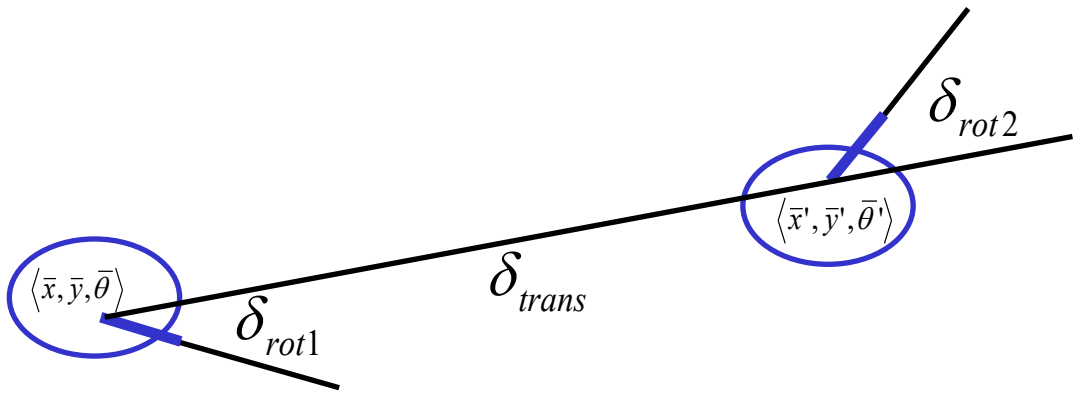


Odometry Based Model: Sampling

Goal: sample x_{t+1} from x_t with action $u = (\bar{x}, \bar{x'})$

1. Reparametrize u from $(\bar{x}, \bar{x'})$ to $(\delta_{\text{rot1}}, \delta_{\text{rot2}}, \delta_{\text{trans}})$
2. Add noise to $(\delta_{\text{rot1}}, \delta_{\text{rot2}}, \delta_{\text{trans}})$ to get $(\hat{\delta}_{\text{rot1}}, \hat{\delta}_{\text{rot2}}, \hat{\delta}_{\text{trans}})$
3. Compute next state x_{t+1} from $(\hat{\delta}_{\text{rot1}}, \hat{\delta}_{\text{rot2}}, \hat{\delta}_{\text{trans}})$

Key idea: odometry gives you change in angles, this is noisy and gives next state



Odometry Based Model: Sampling

Reparameterization

Noise addition

Computing moved state

Algorithm sample_motion_model_odometry(u_t, x_{t-1})**:**

$$\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$

$$\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$$

$$\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$$

$$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta_{\text{rot1}} + \alpha_2 \delta_{\text{trans}})$$

$$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\ \delta_{\text{trans}} + \alpha_4 (\delta_{\text{rot1}} + \delta_{\text{rot2}}))$$

$$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta_{\text{rot2}} + \alpha_2 \delta_{\text{trans}})$$

$$x' = x + \hat{\delta}_{\text{trans}}\ \cos(\theta + \hat{\delta}_{\text{rot1}})$$

$$y' = y + \hat{\delta}_{\text{trans}}\ \sin(\theta + \hat{\delta}_{\text{rot1}})$$

$$\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$$

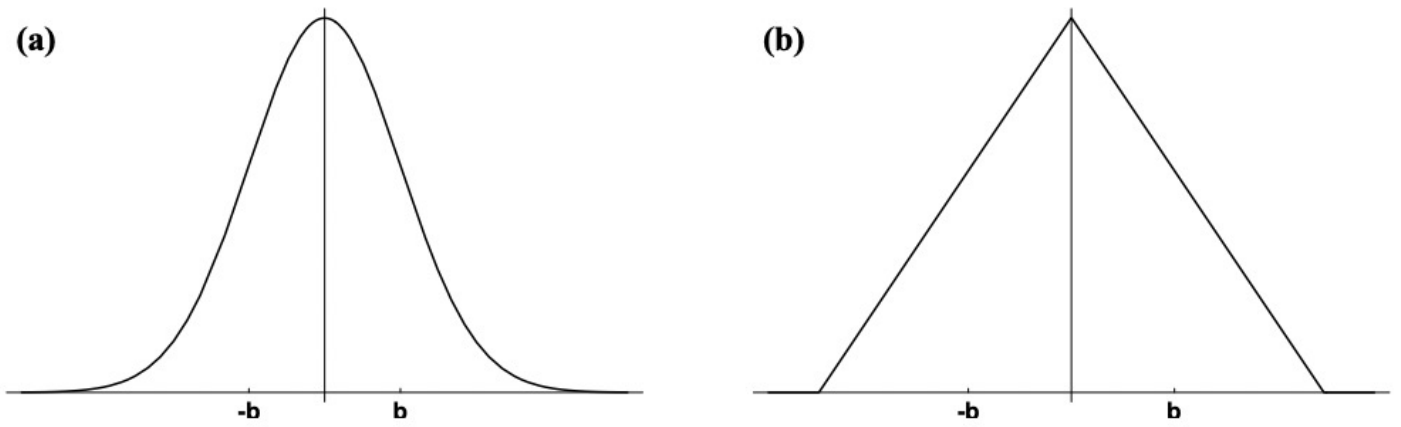
return $x_t = (x', y', \theta')^T$

Odometry Based Model: Likelihood

Goal: find likelihood x_{t+1} from x_t with action $u = (\bar{x}, \bar{x}')$

Key idea: Find $(\delta_{\text{rot1}}, \delta_{\text{rot2}}, \delta_{\text{trans}})$ from (x_t, x_{t+1}) and $(\bar{x}, \bar{x}')$, compare under noise model

Often called an inverse motion model



Odometry Based Model: Likelihood

Reparameterized odometry

Reparameterized actual motion

Computed probabilities

Joint likelihood

Algorithm motion_model_odometry(x_t, u_t, x_{t-1})**:**

$$\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$

$$\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$$

$$\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$$

$$\hat{\delta}_{\text{rot1}} = \text{atan2}(y' - y, x' - x) - \theta$$

$$\hat{\delta}_{\text{trans}} = \sqrt{(x - x')^2 + (y - y')^2}$$

$$\hat{\delta}_{\text{rot2}} = \theta' - \theta - \hat{\delta}_{\text{rot1}}$$

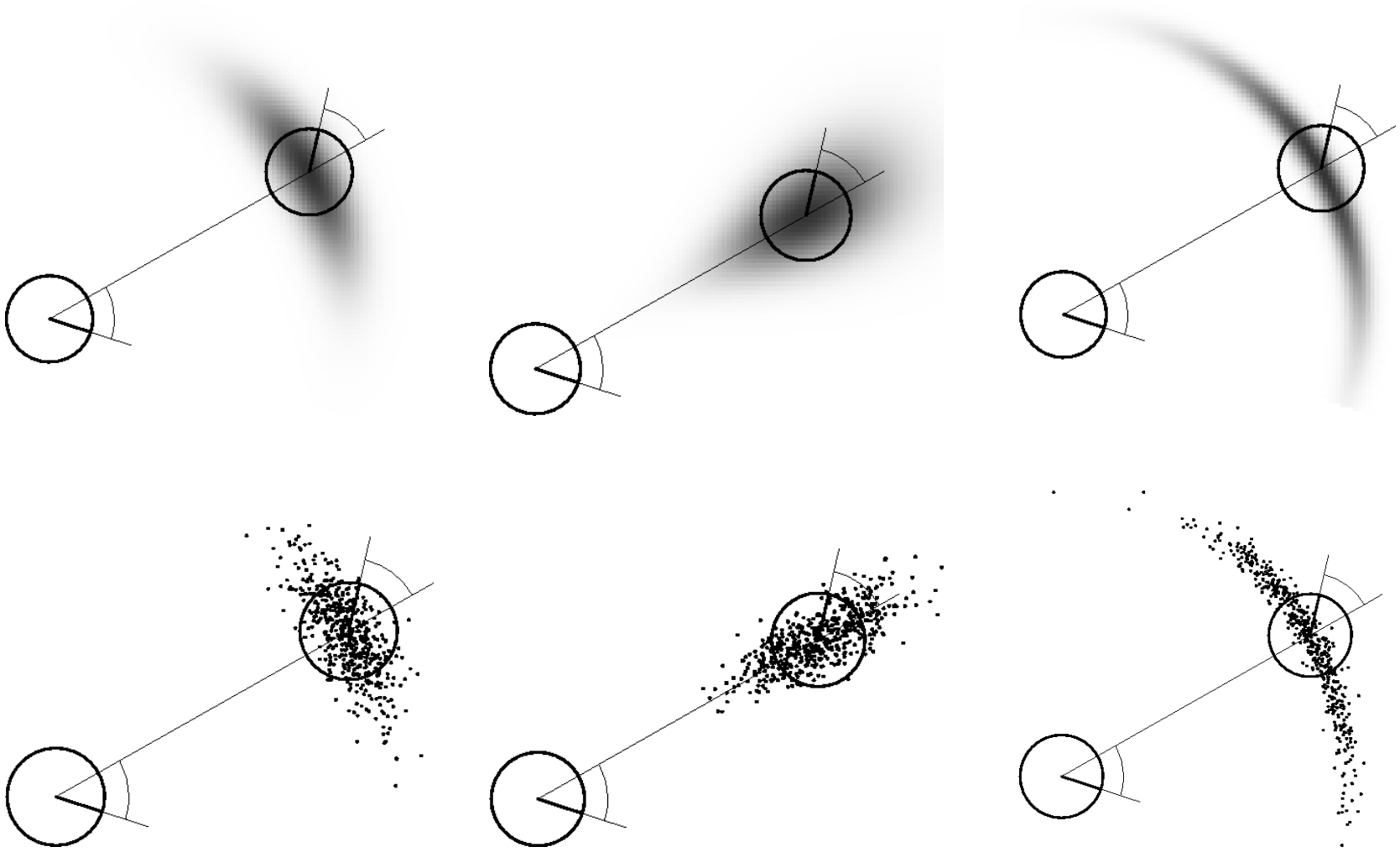
$$p_1 = \textbf{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \hat{\delta}_{\text{rot1}} + \alpha_2 \hat{\delta}_{\text{trans}})$$

$$p_2 = \textbf{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \, \hat{\delta}_{\text{trans}} + \alpha_4 (\hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}))$$

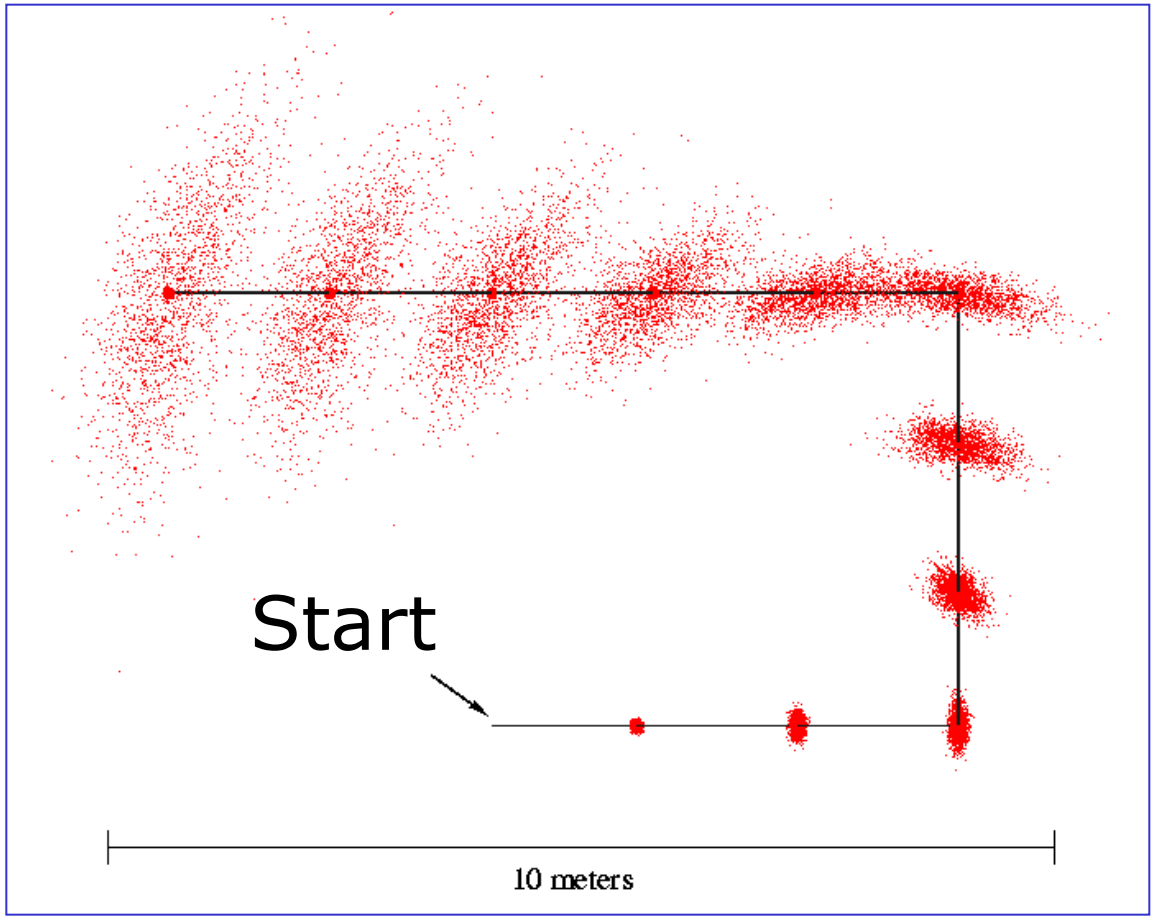
$$p_3 = \textbf{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \hat{\delta}_{\text{rot2}} + \alpha_2 \hat{\delta}_{\text{trans}})$$

return $p_1 \cdot p_2 \cdot p_3$

Examples (odometry based)



Sample-based Motion



HW 1EKF Prediction Step Pseudocode: Odometry Style

1. def EKF_prediction($\mu_{t|0:t}, \Sigma_{t|0:t}, u_t, z_{t+1}$):
2. Linearize dynamics:

Set noise to 0 during the gradient computation

$$G = \begin{pmatrix} \frac{\partial x_{t+1}}{\partial x_t} & \frac{\partial x_{t+1}}{\partial y_t} & \frac{\partial x_{t+1}}{\partial \theta_t} \\ \frac{\partial y_{t+1}}{\partial x_t} & \frac{\partial y_{t+1}}{\partial y_t} & \frac{\partial y_{t+1}}{\partial \theta_t} \\ \frac{\partial \theta_{t+1}}{\partial x_t} & \frac{\partial \theta_{t+1}}{\partial y_t} & \frac{\partial \theta_{t+1}}{\partial \theta_t} \end{pmatrix} \quad V = \begin{pmatrix} \frac{\partial x_{t+1}}{\partial \delta_{rot1}} & \frac{\partial x_{t+1}}{\partial \delta_{trans}} & \frac{\partial x_{t+1}}{\partial \delta_{rot2}} \\ \frac{\partial y_{t+1}}{\partial \delta_{rot1}} & \frac{\partial y_{t+1}}{\partial \delta_{trans}} & \frac{\partial y_{t+1}}{\partial \delta_{rot2}} \\ \frac{\partial \theta_{t+1}}{\partial \delta_{rot1}} & \frac{\partial \theta_{t+1}}{\partial \delta_{trans}} & \frac{\partial \theta_{t+1}}{\partial \delta_{rot2}} \end{pmatrix}$$

3. Prediction:

$$\mu_{t+1|0:t} = g(\mu_{t|0:t}, u_t)$$
$$\Sigma_{t+1|0:t} = G\Sigma_{t|0:t}G^T + VMV^T$$

1. Return $\mu_{t+1|0:t}$ $\Sigma_{t+1|0:t}$

State – (x, y, θ)

Action - $\delta_{rot1}, \delta_{trans}, \delta_{rot2}$

$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$
$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3 |\delta_{trans}| + \alpha_4 |\delta_{rot1} + \delta_{rot2}|}$$
$$\hat{\delta}_{rot2} = \delta_{rot2} + \varepsilon_{\alpha_1 |\delta_{rot2}| + \alpha_2 |\delta_{trans}|}$$

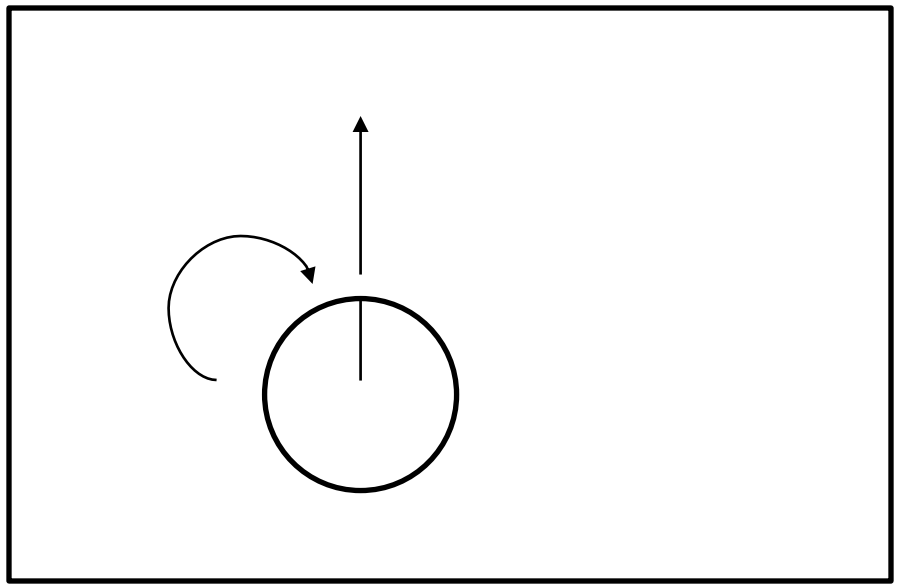
$$\begin{bmatrix} \epsilon_{\delta_{rot1}} \\ \epsilon_{\delta_{trans}} \\ \epsilon_{\delta_{rot2}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \underbrace{\begin{bmatrix} Q_{\delta_{rot1}} & 0 & 0 \\ 0 & Q_{\delta_{trans}} & 0 \\ 0 & 0 & Q_{\delta_{rot2}} \end{bmatrix}}_{M} \right)$$

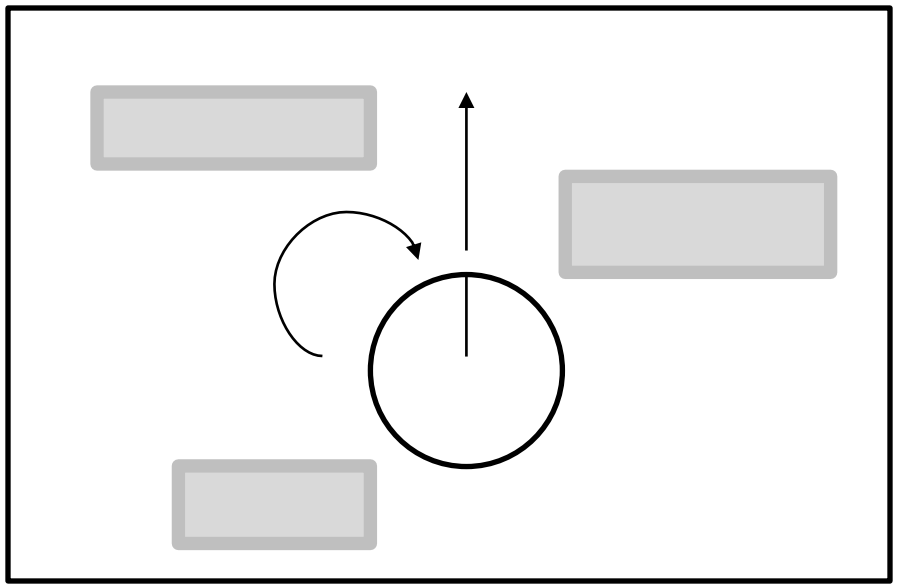
$$x' = x + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}})$$
$$y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$$
$$\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$$

Integrating Maps into Motion Models

From free space motion models to maps

Free space motion models do not account for obstacles in a **known** map

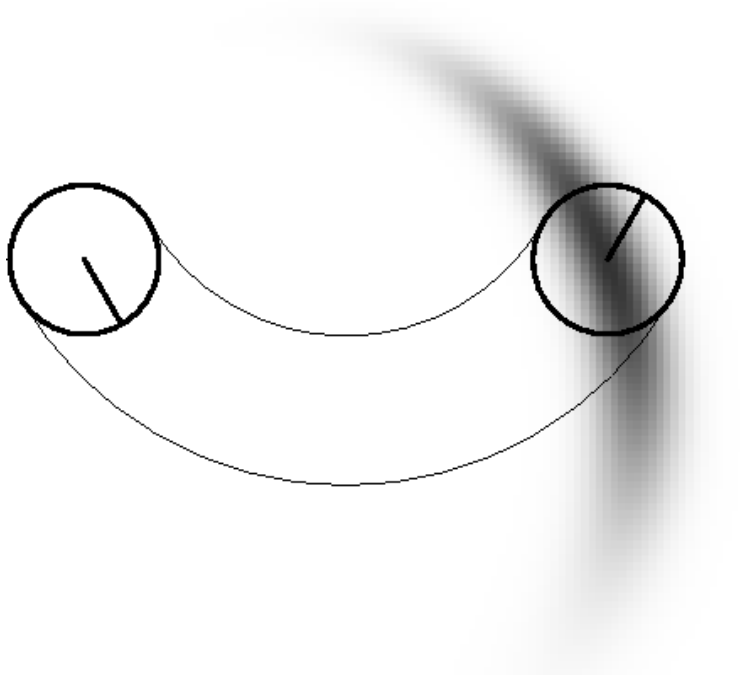




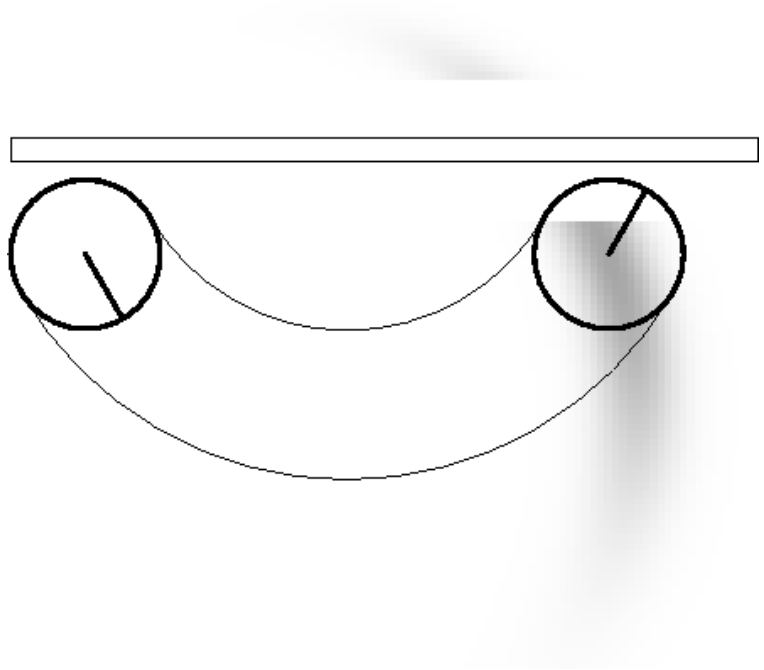
$$P(x \mid u, x', m) \approx P(x \mid m)\, P(x \mid u, x')$$

Zero-out positions that are not possible in the map

Motion Model with Map

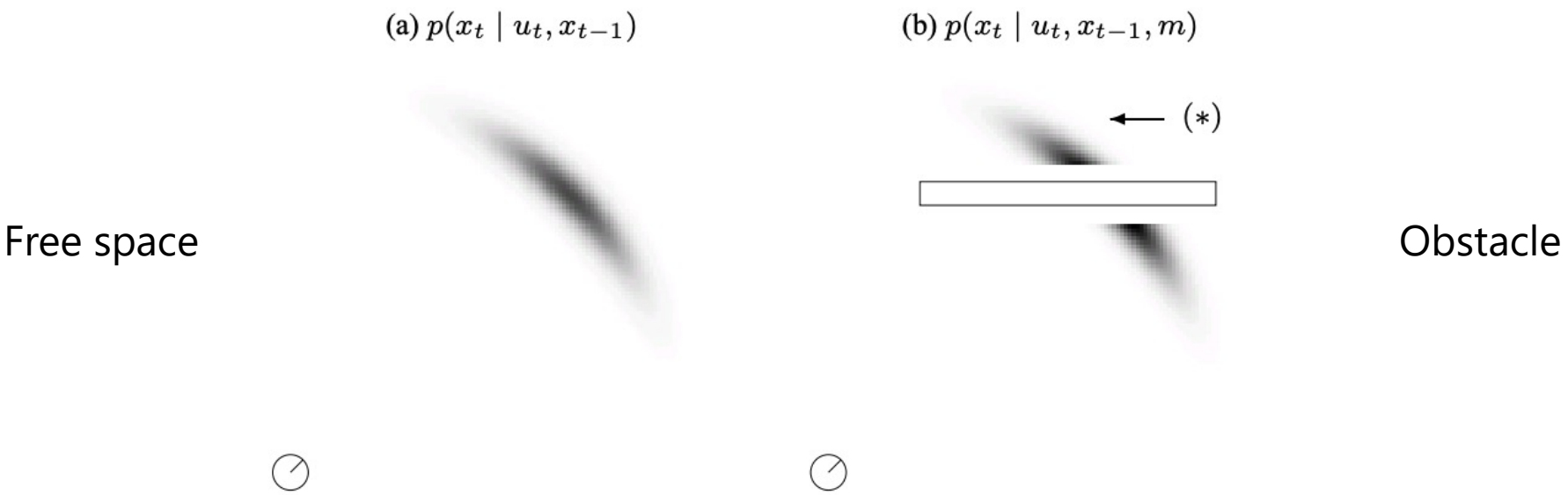


$P(x \mid u, x')$



$P(x \mid u, x', m) \approx P(x \mid m)\ P(x \mid u, x')$

Failure Case



Don’t account for motion through walls → deal with by increasing frequency

Lecture Outline

Particle Filters

Motion Models

Sensor Models

Sensor Models for Bayesian Filtering

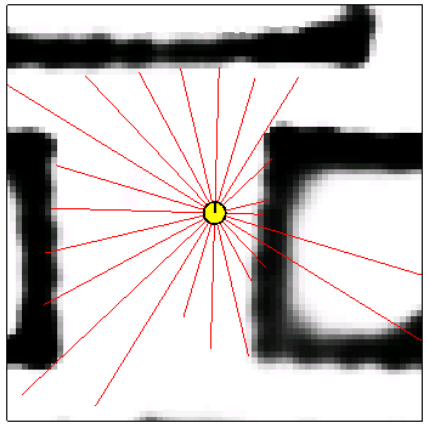
$$Bel(x_t) = P(x_t|u_{0:t-1}, z_{0:t})$$
$$= \eta\, p(z_t|x_t) \int P(x_t|u_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

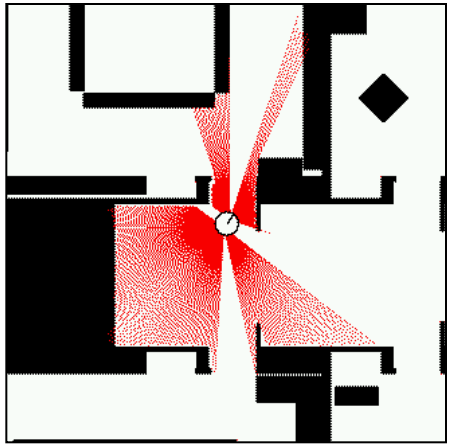
Let's try and specify what this is

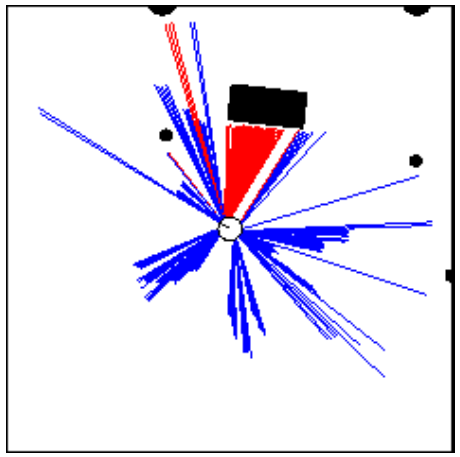
Sensors for Mobile Robots

- Contact sensors: Bumpers, touch sensors
- Internal sensors
  - Accelerometers (spring-mounted masses)
  - Gyroscopes (spinning mass, laser light)
  - Compasses, inclinometers (earth magnetic field, gravity)
  - Encoders, torque
- Proximity sensors
  - Sonar (time of flight)
  - Radar (phase and frequency)
  - Laser range-finders (triangulation, tof, phase)
  - Infrared (intensity)
- Visual sensors: Cameras, depth cameras
- Satellite-style sensors: GPS, MoCap

Proximity Sensors







- The central task is to determine *P(z|x)*, i.e. the probability of a measurement *z* given that the robot is at position *x*.
- **Question**: Where do the probabilities come from?
- **Approach**: Let's try to explain a measurement.

Beam-based Sensor Model

Beam-based Sensor Model

- Scan z consists of K measurements.

$$z = \{z_1, z_2, \ldots, z_K\}$$

Beam-based Sensor Model

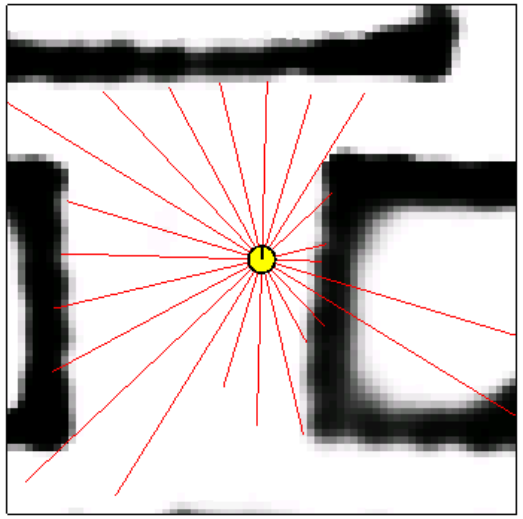
- Scan *z* consists of *K* measurements.

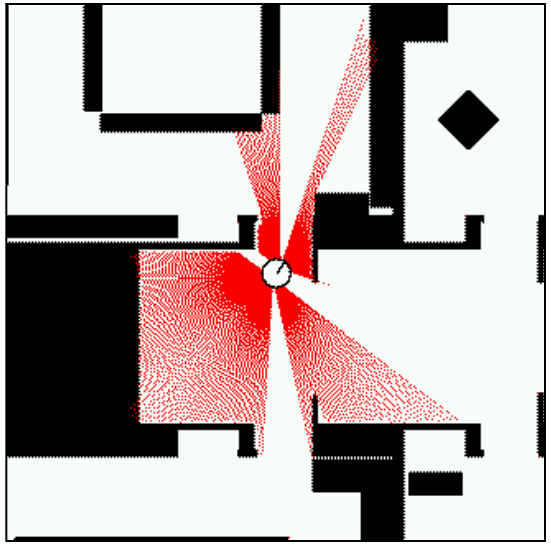
$$z = \{z_1, z_2, \ldots, z_K\}$$

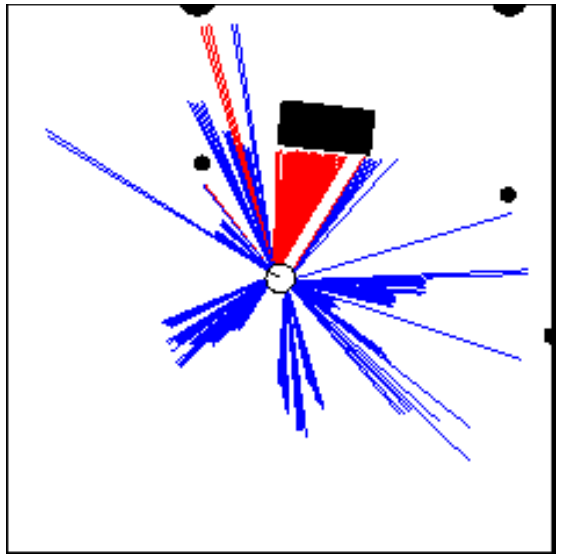
- Individual measurements are independent given the robot position and a map.

$$P(z \mid x, m) = \prod_{k=1}^{K} P(z_k \mid x, m)$$

Beam-based Sensor Model







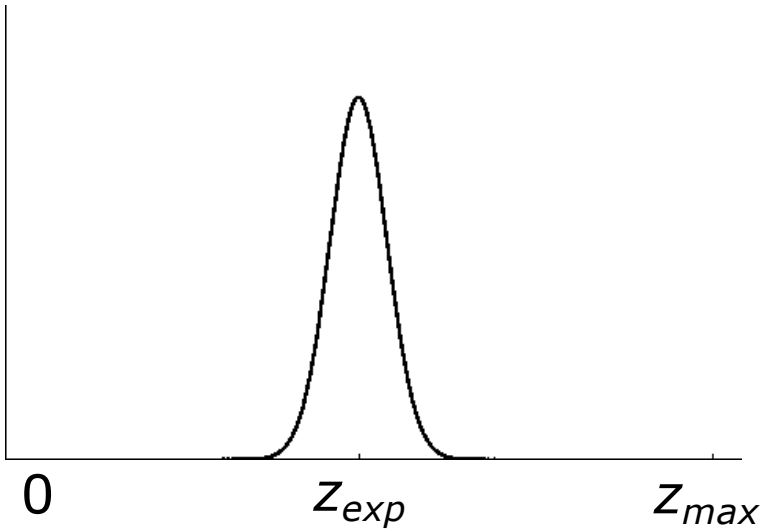
$$P(z \mid x, m) = \prod_{k=1}^{K} P(z_k \mid x, m)$$

Proximity Measurement

- Measurement can be caused by …
  - a known obstacle.
  - cross-talk.
  - an unexpected obstacle (people, furniture, …).
  - missing all obstacles (total reflection, glass, …).
- Noise is due to uncertainty …
  - in measuring distance to known obstacle.
  - in position of known obstacles.
  - in position of additional obstacles.
  - whether obstacle is missed.

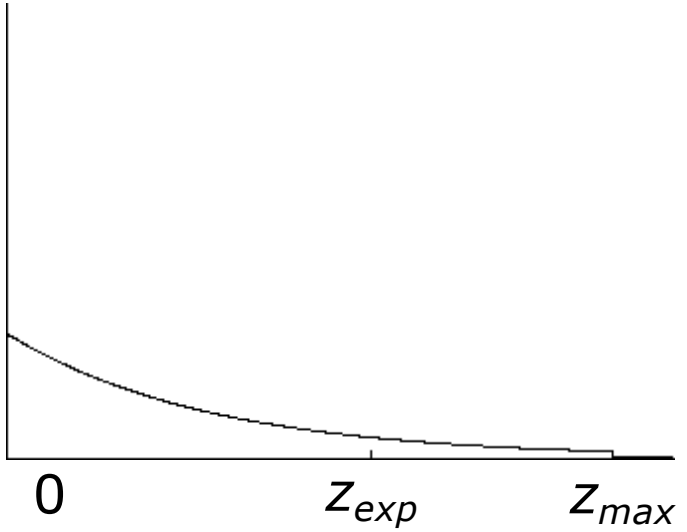
Beam-based Proximity Model

Measurement noise



$$P_{hit}(z \mid x,m) = \eta \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(z-z_{\exp})^2}{\sigma^2}}$$

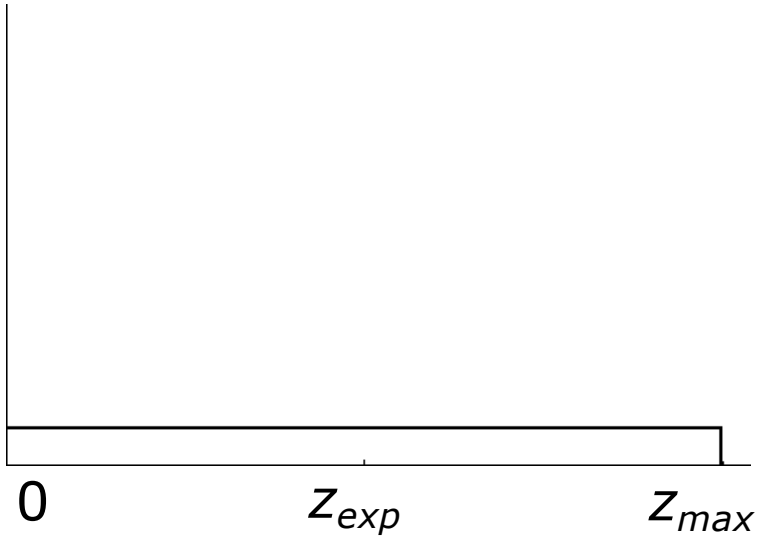
Unexpected obstacles



$$P_{\text{unexp}}(z \mid x,m) = \eta\, \lambda\, \mathrm{e}^{-\lambda z}$$

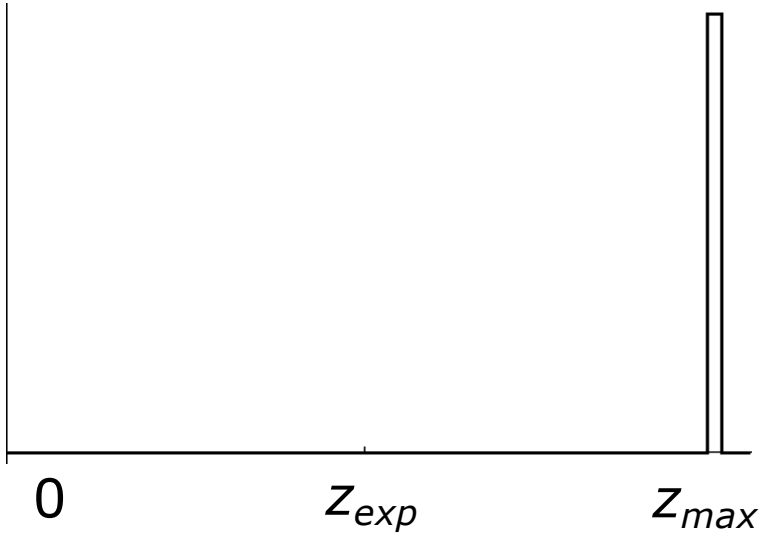
Beam-based Proximity Model

Random measurement



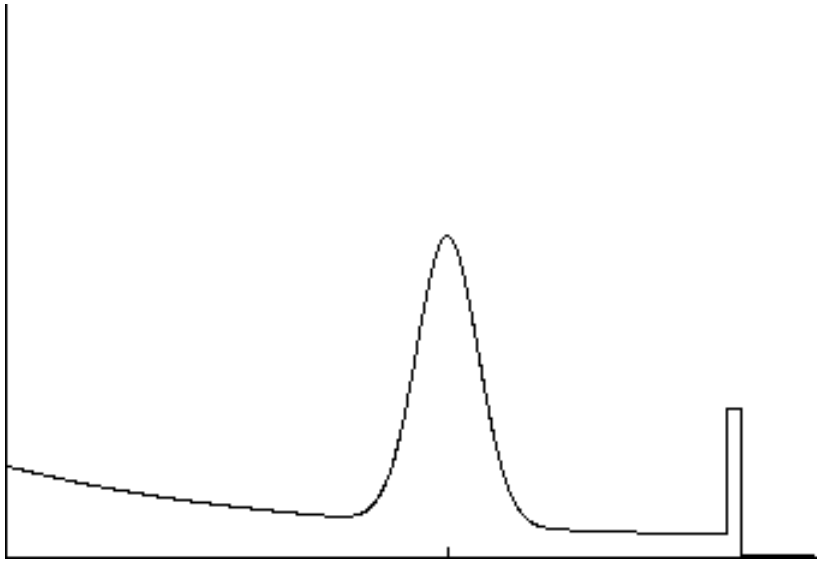
$$P_{rand}(z \mid x, m) = \eta \frac{1}{z_{\max}}$$

Max range



$$P_{\max}(z \mid x, m) = \eta \frac{1}{z_{small}}$$

Mixture Density



$$P(z \mid x,m) = \begin{pmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{unexp}} \\ \alpha_{\text{max}} \\ \alpha_{\text{rand}} \end{pmatrix}^{T} \cdot \begin{pmatrix} P_{\text{hit}}(z \mid x,m) \\ P_{\text{unexp}}(z \mid x,m) \\ P_{\text{max}}(z \mid x,m) \\ P_{\text{rand}}(z \mid x,m) \end{pmatrix}$$

How can we determine the model parameters?

→ More on this next lecture

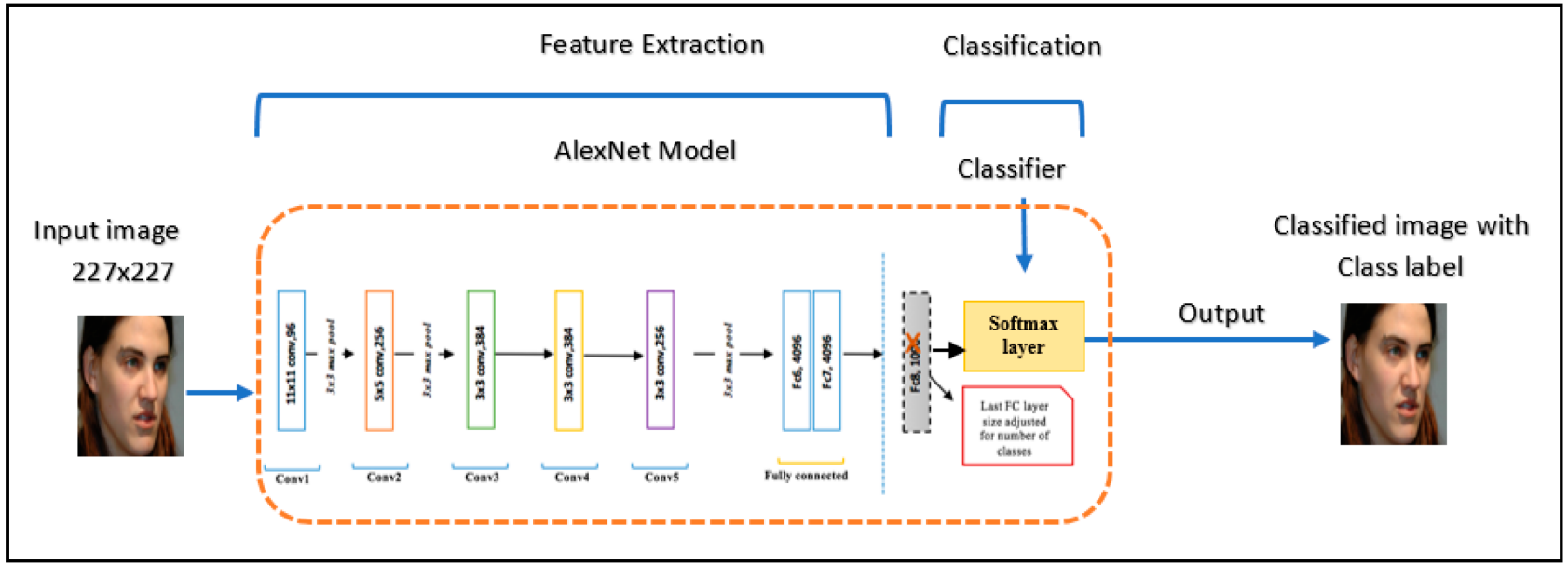
Summary Beam-based Model

- Assumes independence between beams.
  - Overconfident!
- Models physical causes for measurements.
  - Mixture of densities for these causes.
- Implementation
  - Learn parameters based on real data.
  - Different models can be learned for different angles at which the sensor beam hits the obstacle.
  - Determine expected distances by ray-tracing.
  - Expected distances can be pre-processed.

Landmark-based Sensor Model

When are raw measurement based models not enough?

- Scales unfavorably with dimensionality of the meas

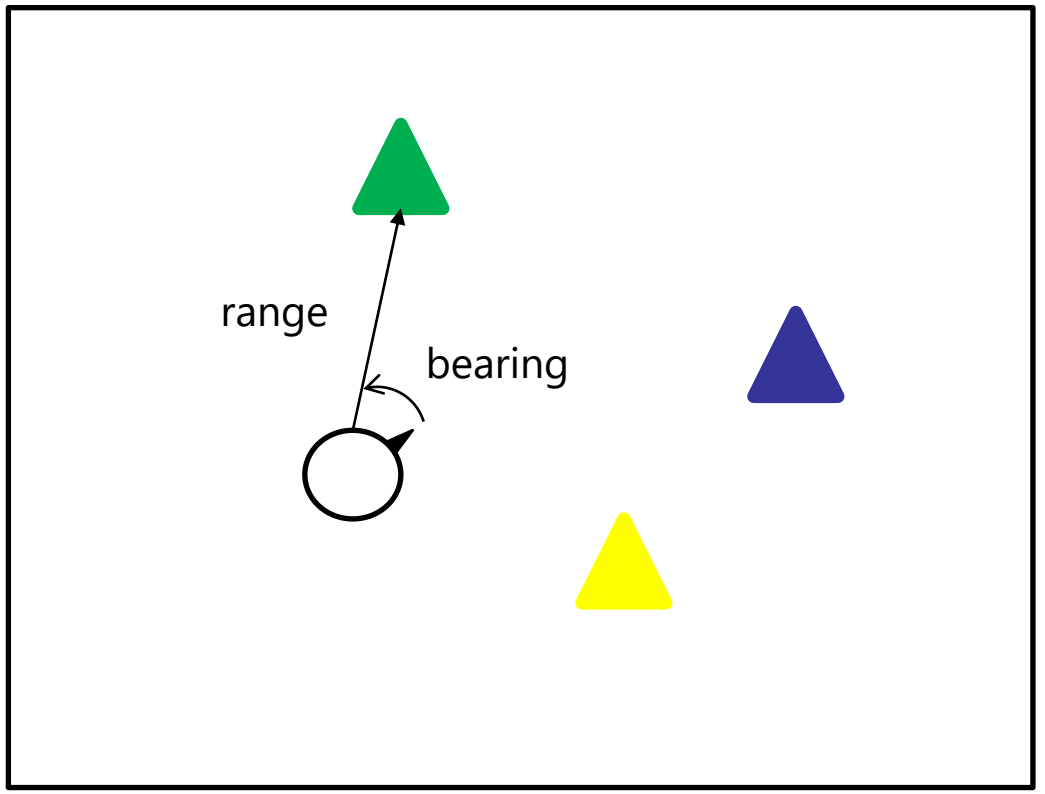


- Common strategy in machine learning - extract low dimensional features

Landmarks

- Active beacons (*e.g.* radio, GPS)
- Passive (*e.g.* visual, retro-reflective)
- Sensor provides
  - distance, or
  - bearing, or
  - distance and bearing.
  - signature

Distance and Bearing



Probabilistic Model

1. Algorithm **landmark_detection_model**(z,x,m):

$$z = \langle i, d, \alpha \rangle, x = \langle x, y, \theta \rangle$$

Compute expected range/bearing

2. $$\hat{d} = \sqrt{(m_x(i) - x)^2 + (m_y(i) - y)^2}$$

3. $$\hat{\alpha} = \text{atan2}(m_y(i) - y, m_x(i) - x) - \theta$$

Compute likelihood

4. $$p_{\text{det}} = \text{prob}(\hat{d} - d, \varepsilon_d) \cdot \text{prob}(\hat{\alpha} - \alpha, \varepsilon_\alpha)$$

5. Return $$z_{\text{det}} p_{\text{det}} + z_{\text{fp}} P_{\text{uniform}}(z \mid x, m)$$

HW 1EKF Correction Step Pseudocode: Landmark Style

1. def EKF_correction($\mu_{t+1|0:t}, \Sigma_{t+1|0:t}, u_t, z_{t+1}$):
2. Linearize measurement:

$$H = \begin{bmatrix} \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \theta} \end{bmatrix}$$

3. Correction:

$$K_{t+1} = \Sigma_{t+1|0:t} H^T (H \Sigma_{t+1} H^T + R)^{-1}$$
$$\mu_{t+1|0:t+1} = \mu_{t+1|0:t} + K_{t+1}(z_{t+1} - h(\mu_{t+1|0:t}))$$
$$\Sigma_{t+1|0:t+1} = (I - K_{t+1} H) \Sigma_{t+1|0:t}$$

4. Return $\mu_{t+1|0:t+1}, \Sigma_{t+1|0:t+1}$

State – (x, y, θ)

Measurement – ϕ

(assumes d is perfectly known)

$$\phi = \text{atan2}(l_y - y, l_x - x) - \theta + \delta$$

$$\delta \sim \mathcal{N}(0, \underbrace{\sigma_\delta^2}_{R})$$

Summary of Parametric Motion and Sensor Models

- Explicitly modeling uncertainty in motion and sensing is key to robustness.
- In many cases, good models can be found by the following approach:
  1. Determine parametric model of noise free motion or measurement.
  2. Analyze sources of noise.
  3. Add adequate noise to parameters (eventually mix in densities for noise).
  4. Learn (and verify) parameters by fitting model to data.
  5. Likelihood of measurement is given by “probabilistically comparing” the actual with the expected measurement.
- It is extremely important to be aware of the underlying assumptions!

Lecture Outline

Particle Filters

Motion Models

Sensor Models

But where do the actual noise values and A, B come from?

- Case 1: Fully observed training
  - We have an oracle technique to observe x, z, u (motion capture) at training time.
  - Maximize likelihood
- Case 2: Partially observed training
  - We can only observe z, u
  - Expectation-Maximization

Case 1: Observed z, x, u – Maximize Likelihood

- Maximize log likelihood of the data (z, x, u) under the motion and sensor models

Linear Gaussian

$$x_{t+1} = Ax_t + Bu_t + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$
$$z_{t+1} = Cx_{t+1} + \delta_t$$
$$\delta_t \sim \mathcal{N}(0, R)$$

$$\max_{A,B,Q} \mathbb{E}_{(x,u,x')} \left[\hat{p}(x'|x,u)\right]$$
$$\hat{p}(.|x,u) = \mathcal{N}(Ax + Bu, Q)$$

$$\max_{C,R} \mathbb{E}_{(z,x)} \left[\hat{p}(z|x)\right]$$
$$\hat{p}(.|x) = \mathcal{N}(Cx, R)$$

Non-linear

$$x_{t+1} = g(x_t, u_t) + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$
$$z_t = h(x_t) + \delta_t$$
$$\delta_t \sim \mathcal{N}(0, R)$$

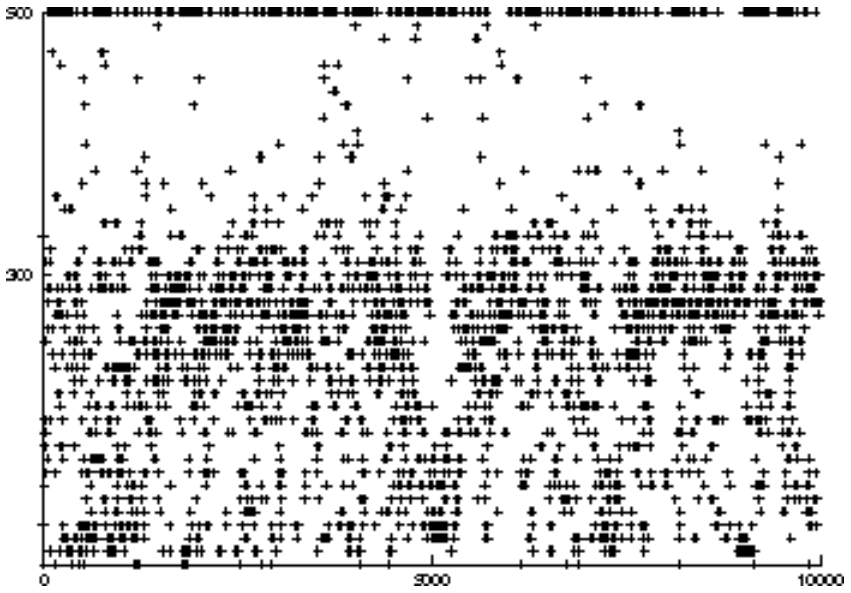
Solve with LS or SGD

$$\max_{\theta} \mathbb{E}_{(x,u,x')} \left[\hat{p}(x'|x,u)\right]$$
$$\hat{p}(.|x,u) = \mathcal{N}(g_\theta(x,u), Q_\theta)$$

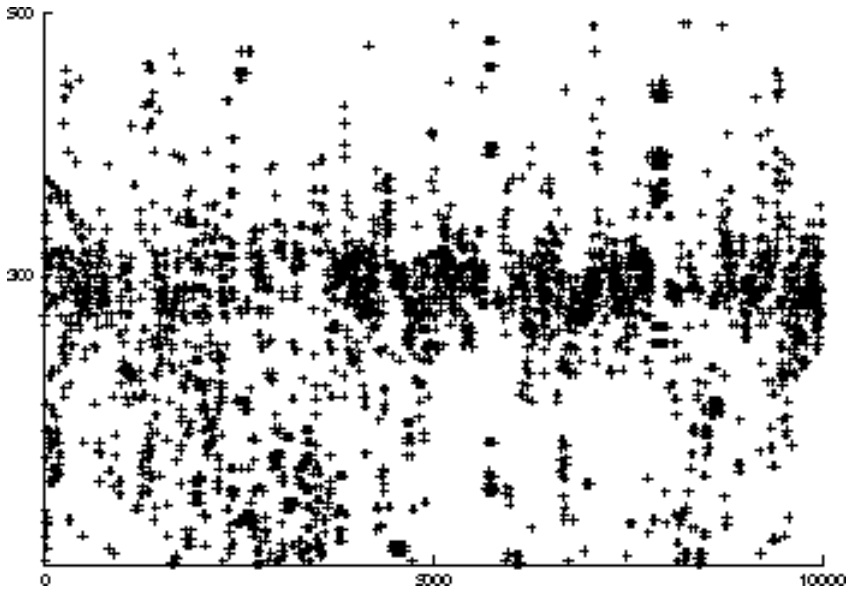
$$\max_{\phi} \mathbb{E}_{(z,x)} \left[\hat{p}(z|x)\right]$$
$$\hat{p}(.|x) = \mathcal{N}(h_\phi(x), R_\phi)$$

Raw Sensor Data

Measured distances for expected distance of 300 cm.

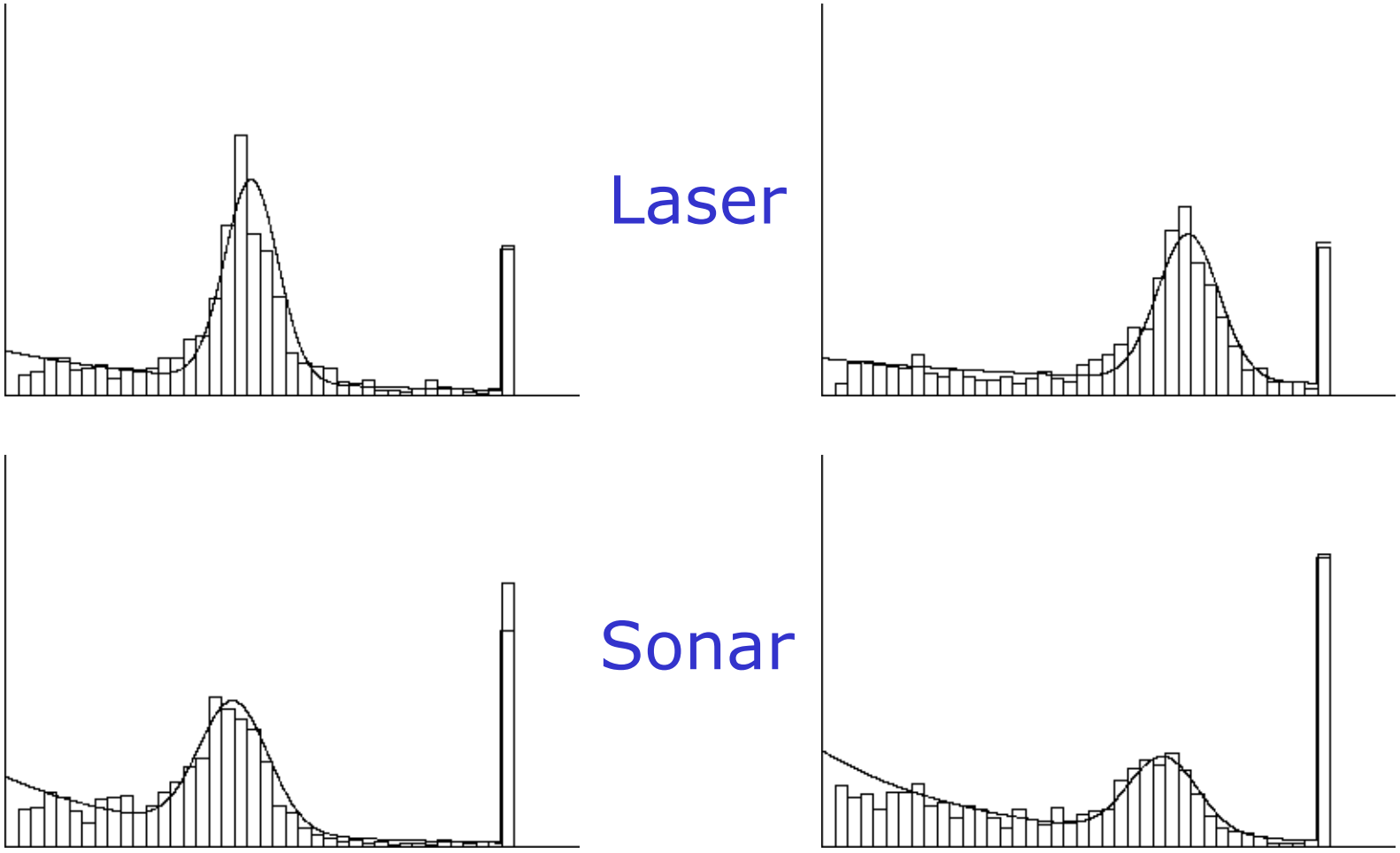


Sonar

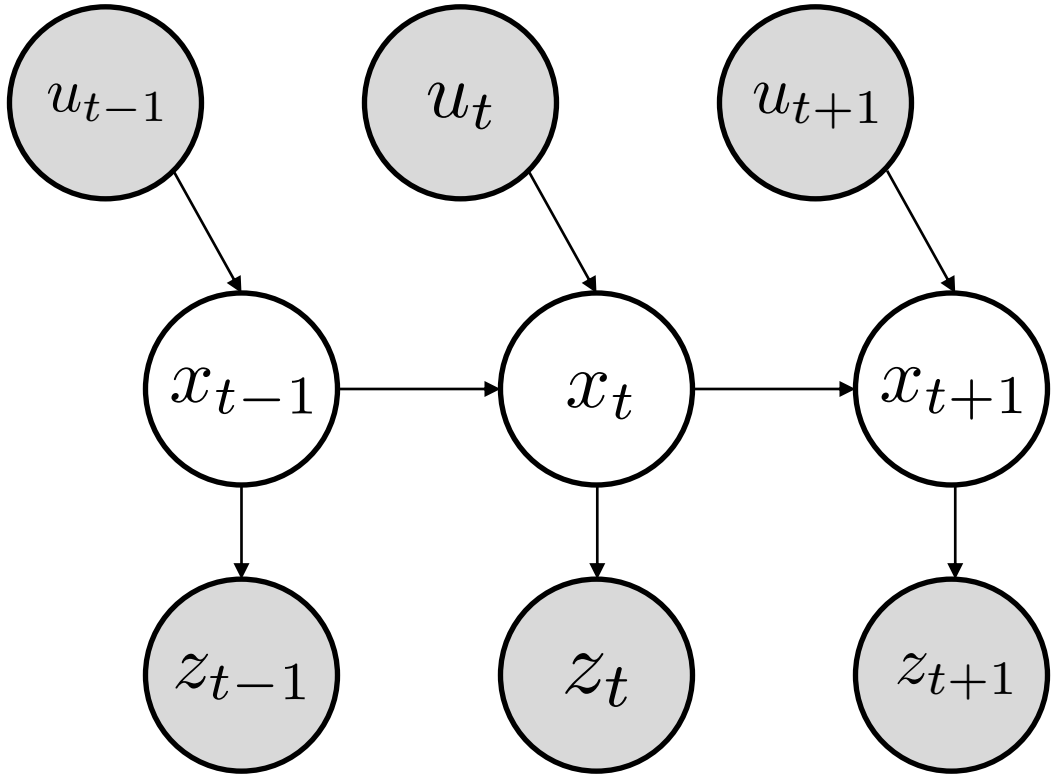


Laser

Approximation Results



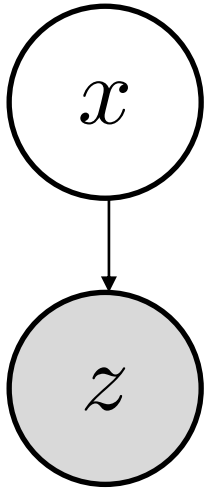
Why is estimating parameters generally not so easy?



X is actually not observed typically, only z, u

Latent variable inference problem

Parameter Estimation in Latent Variable Models



Hard problem to solve exactly

$$\max \log p(z)$$

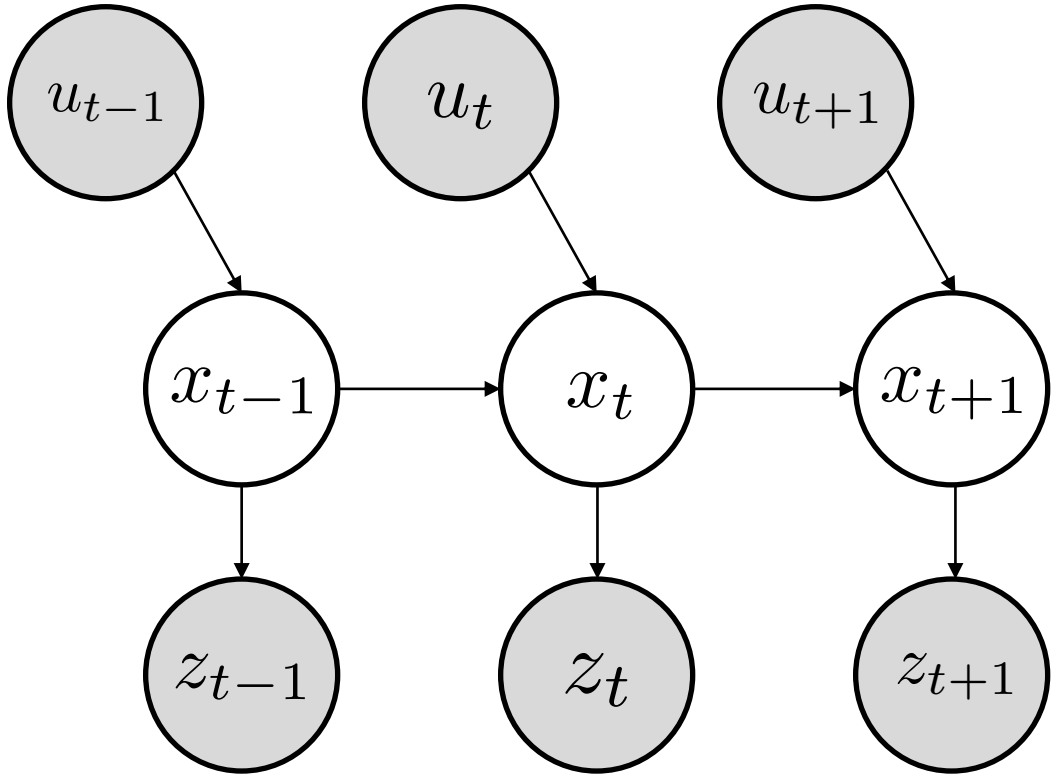
$$= \log \int p(z|x)p(x)dx$$

Intractable problem

Solve via iterative optimization – Expectation Maximization algorithm (EM)

→ Much more general than filtering/localization

EM Algorithm for Latent Variable Parameter Estimation



$$\max \log p(z)$$

Approximate with a 2 step process:

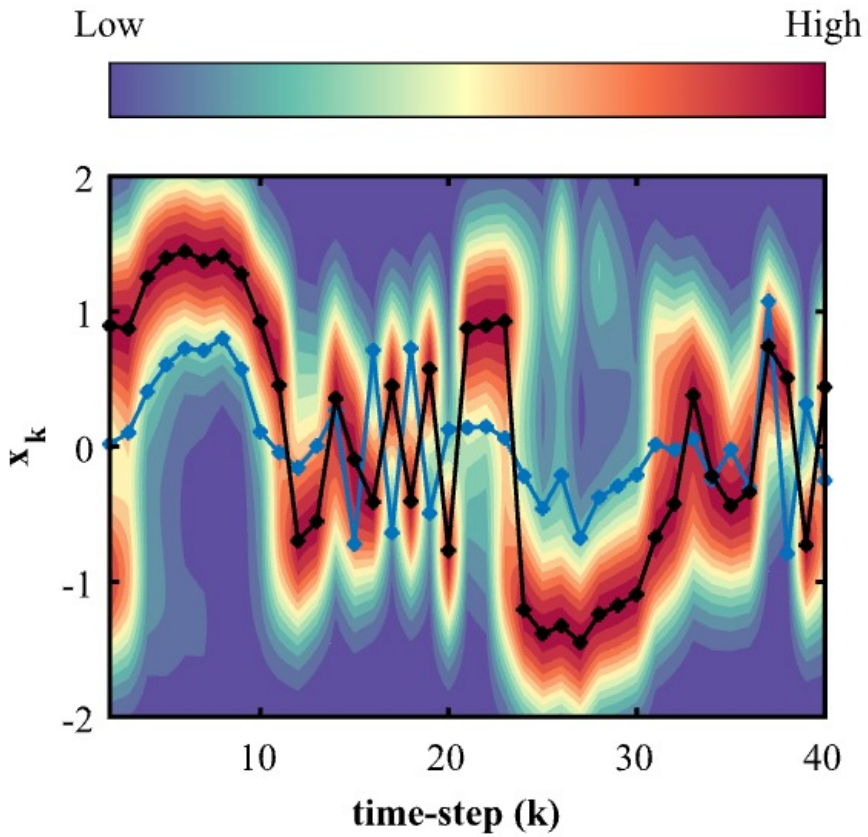
1. **E-step:** Fill in missing data x(i) according to what is most likely given the current model

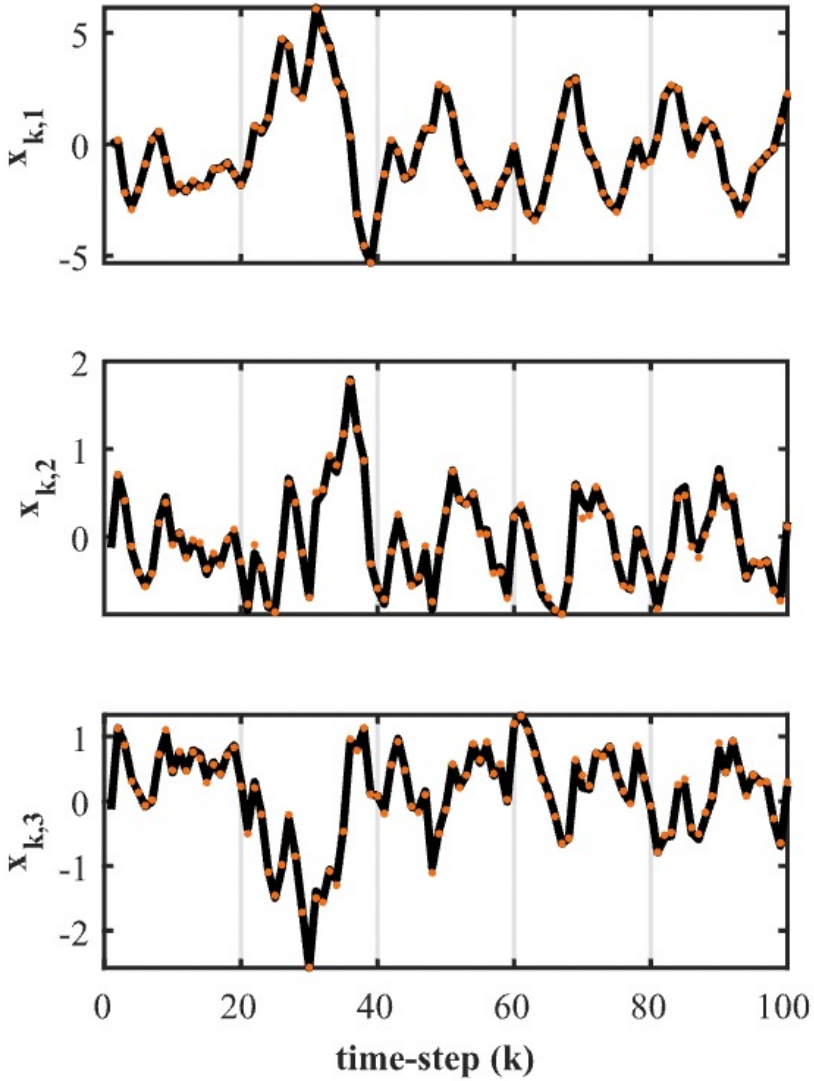
2. **M-step:** Run ML for completed data, which gives new model

Pretend we know the X

Do the best possible given inferred latents

EM Algorithms in Action for Estimating Motion/Sensor Models





Lecture Outline

Motion Models

Sensor Models

Localization with Motion/Sensor Models

Estimating Model Parameters