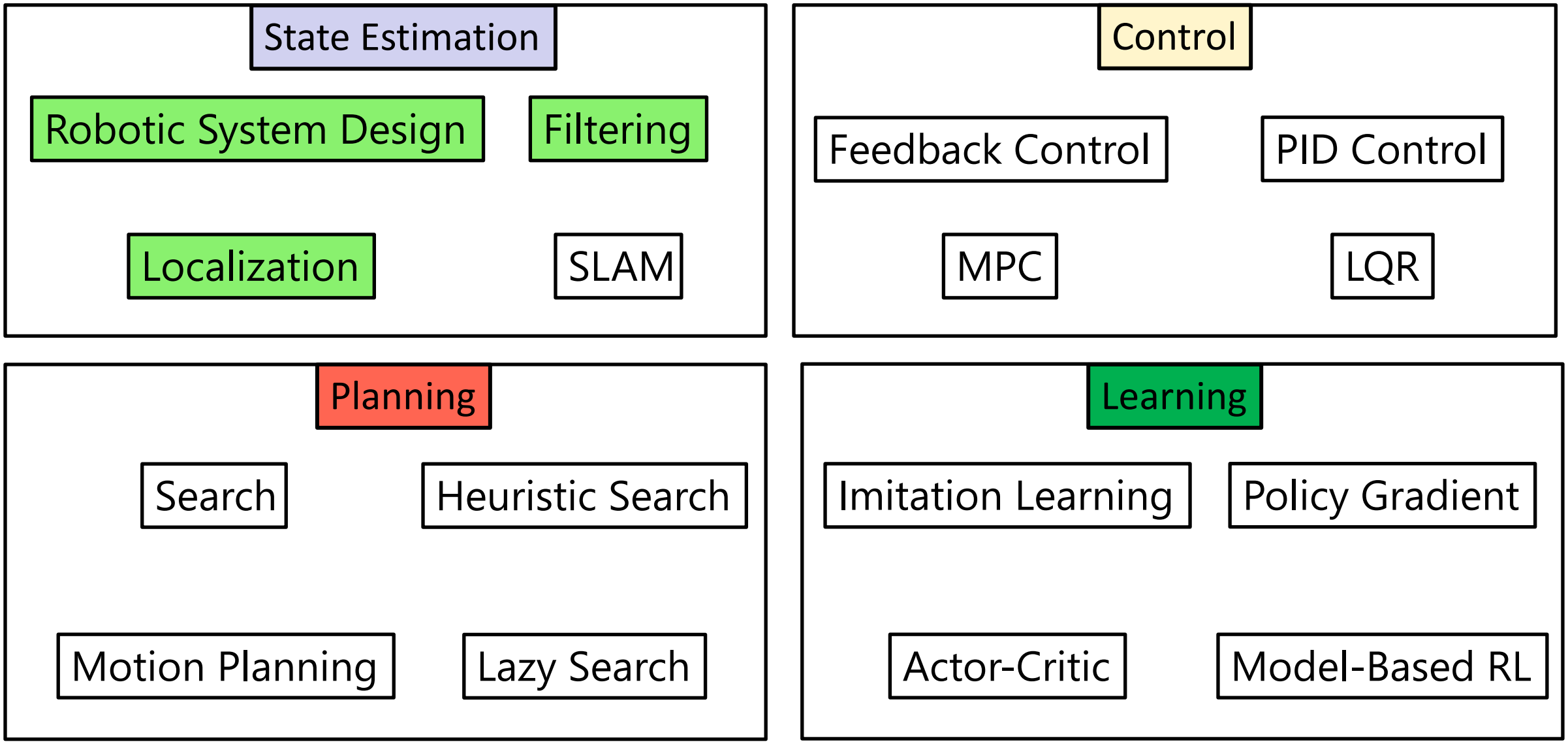
Autonomous Robotics
Winter 2024

Abhishek Gupta

TAs: Karthikeya Vemuri, Arnav Thareja

Marius Memmel, Yunchu Zhang

Class Outline



Logistics

- HW 2 released
- Extra lecture/OH after class to clear up Kalman Filters/Particle Filters
- Feedback welcome on how we can make the class better!

Lecture Outline

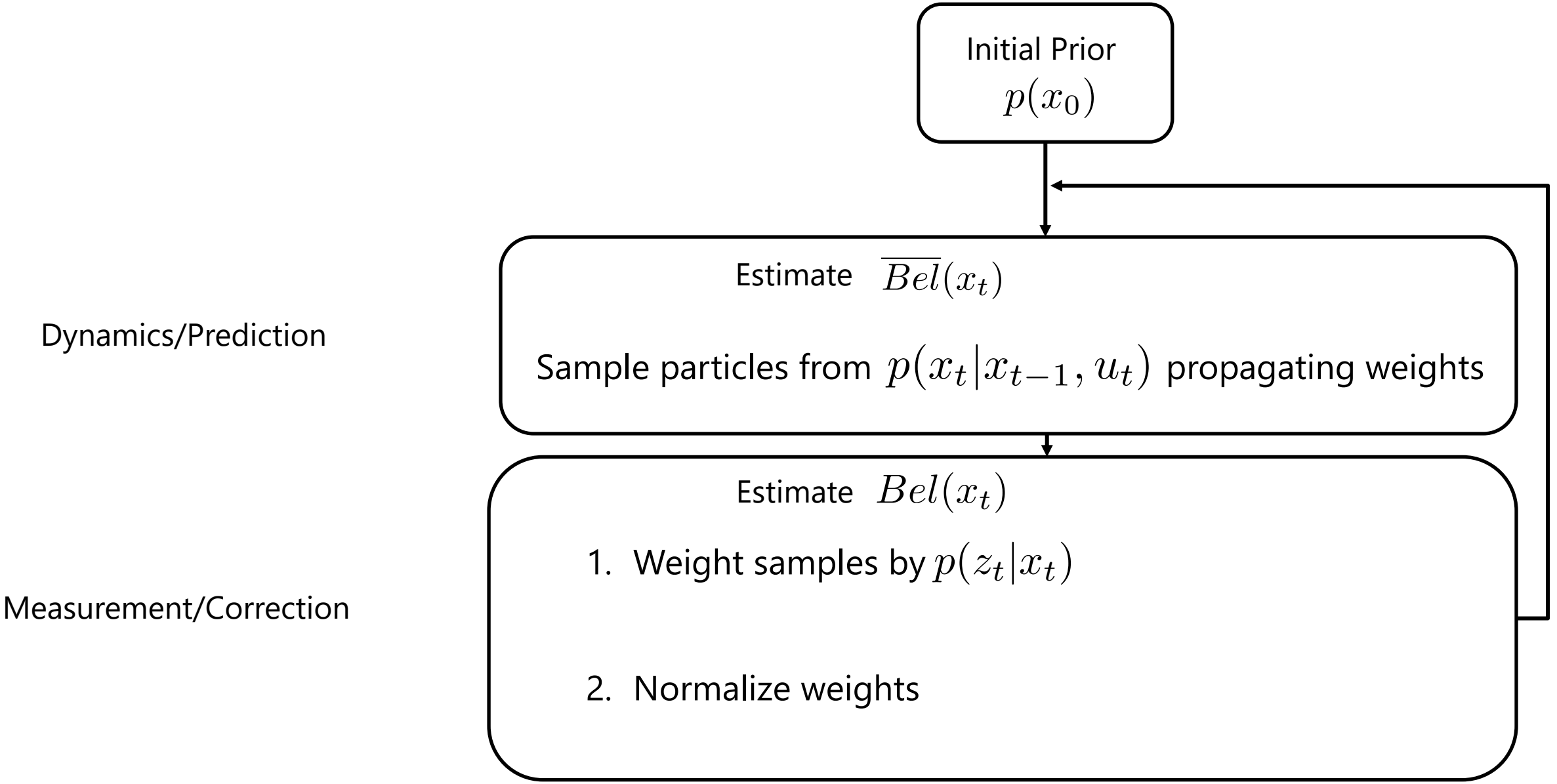
(Whiteboard) Recap of Basic Particle Filtering

Particle Filter w/ Resampling

Practical Considerations

EKF + Applications

Overall Particle Filter algorithm – v1



Lecture Outline

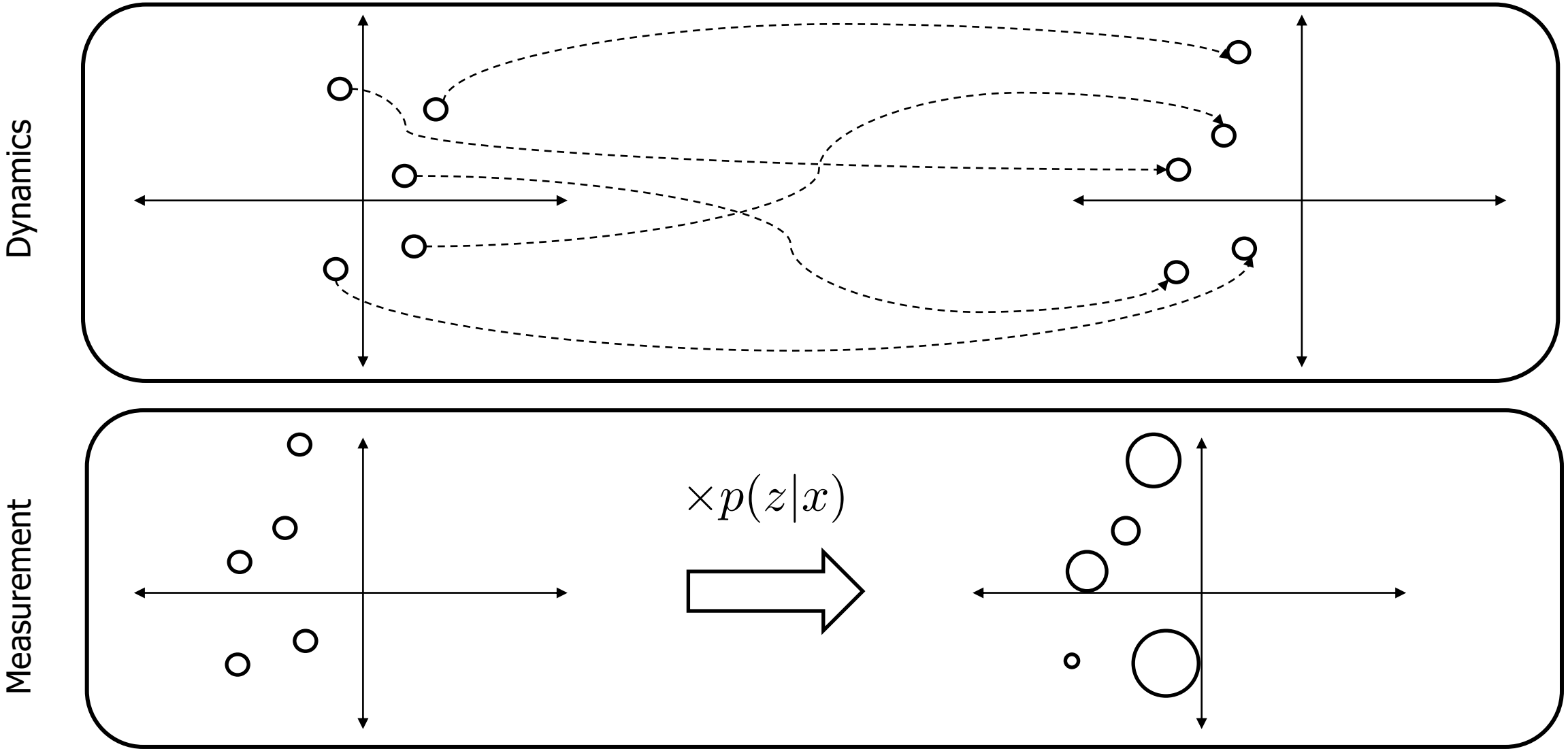
(Whiteboard) Recap of Basic Particle Filtering

Particle Filter w/ Resampling

Practical Considerations

EKF + Applications

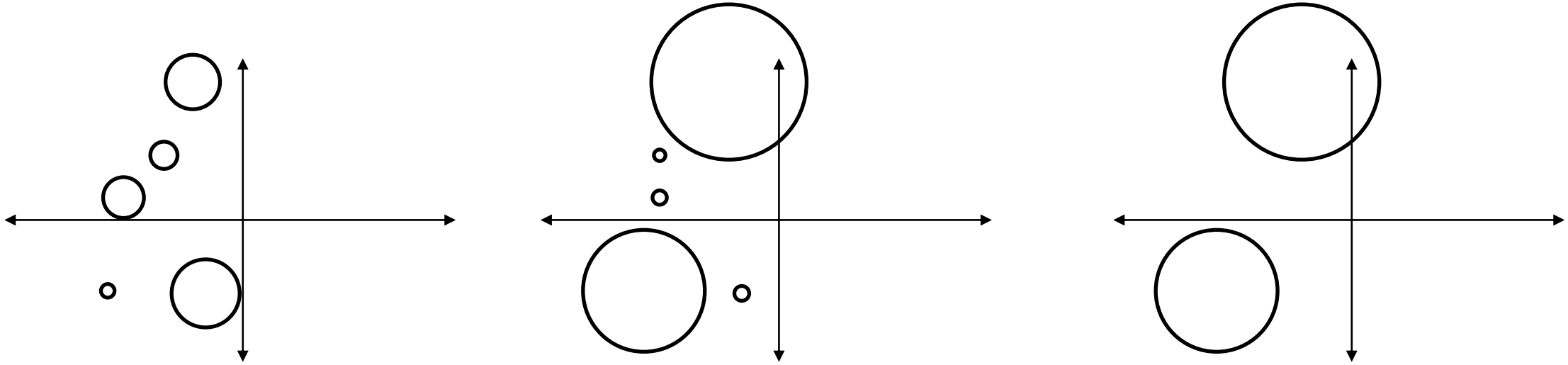
What happens across multiple steps?



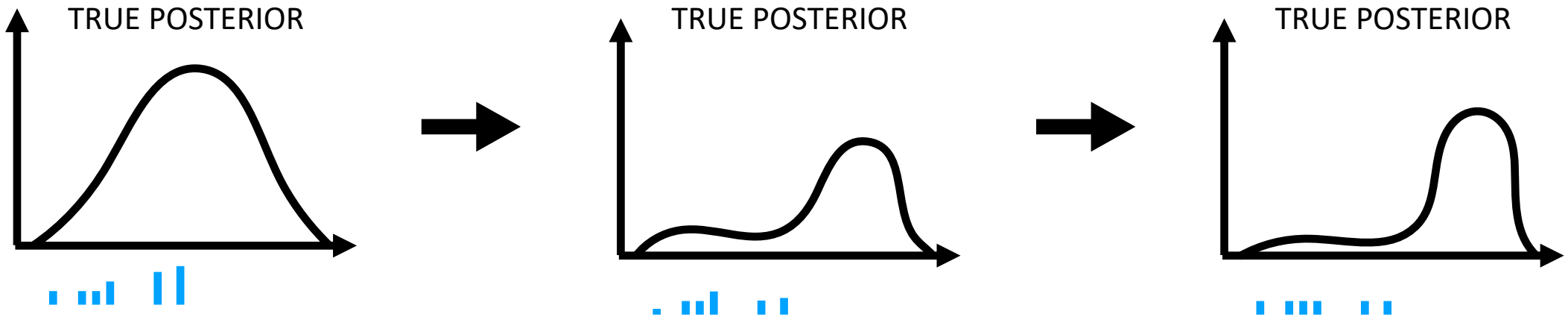
Importance weights get multiplied at each step

Why might this be bad?

Importance weights get multiplied at each step



1. May blow up and get numerically unstable over many steps
2. Particles stay stuck in unlikely regions



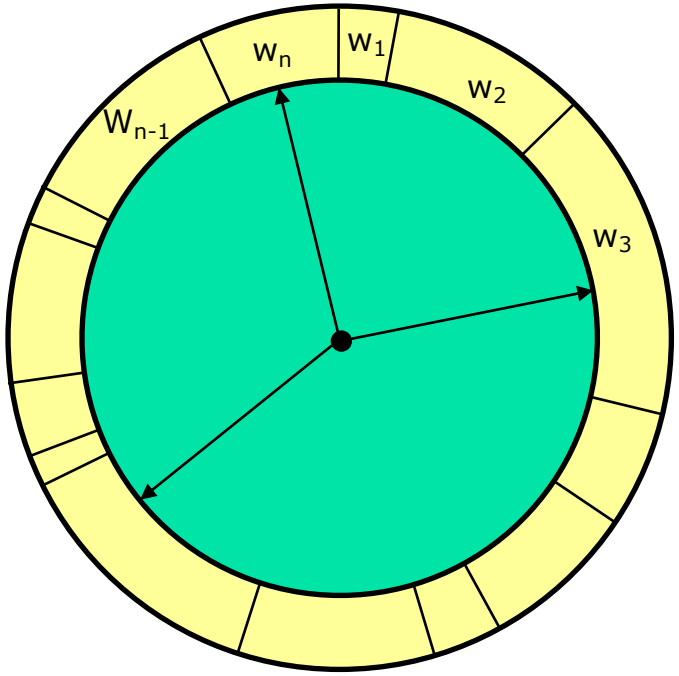
Resampling

- **Given**: Set *S* of weighted samples (from measurement step) with weights w_i

- **Wanted** : unweighted random sample, where the probability of drawing x_i is given by w_i.

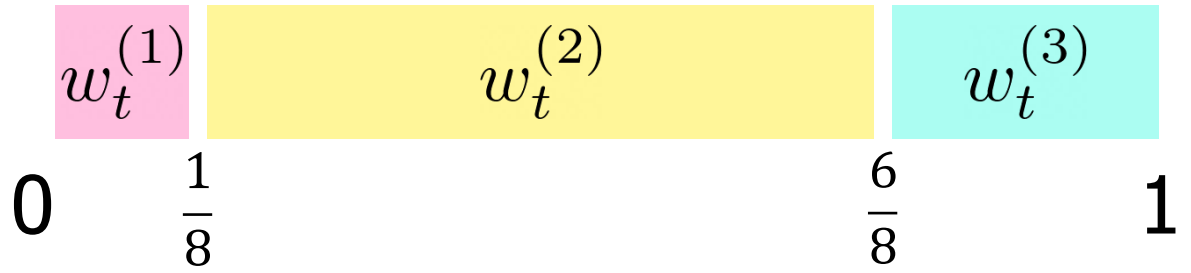
- Typically done *n* times with replacement to generate new sample set *S'*.

Resampling



Here are your random numbers:

0.97
0.26
0.72



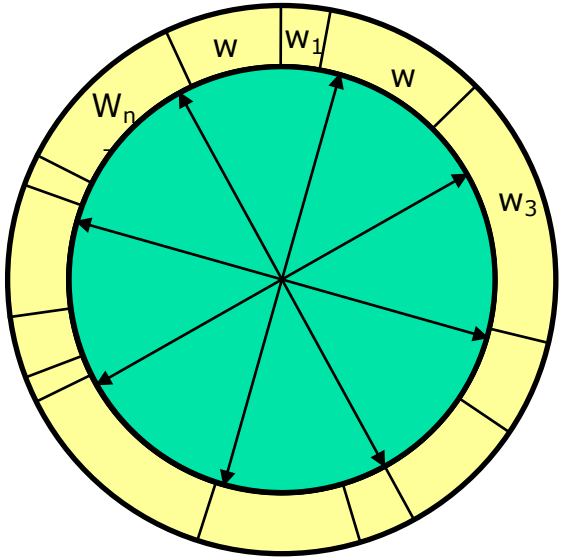
- Spin a roulette wheel
- Space according to weights
- Pick samples based on where it lands

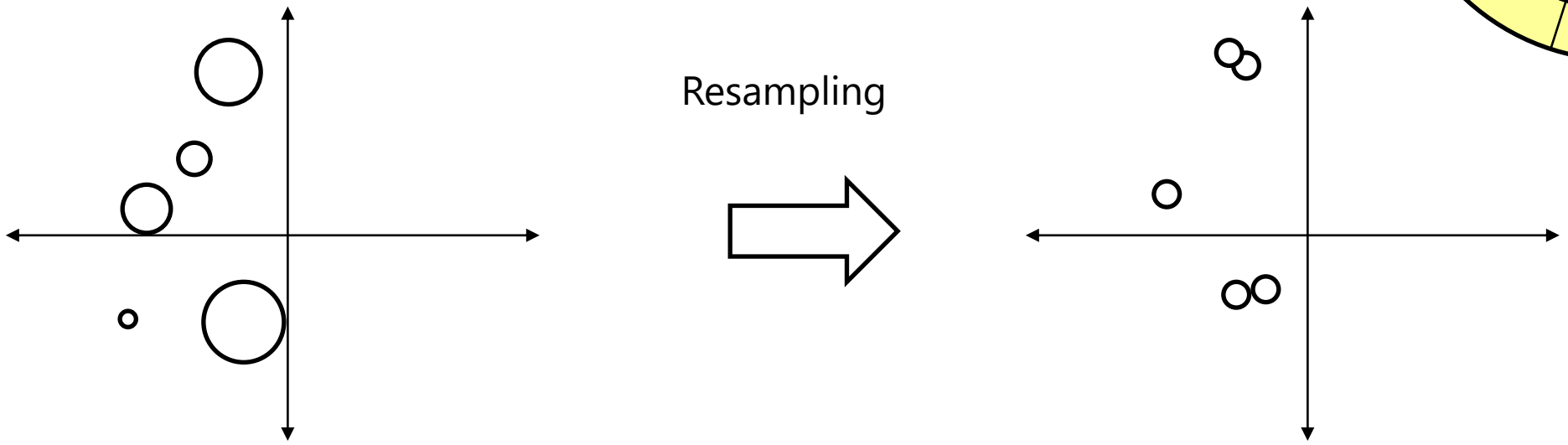
Resampling in a particle filter

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

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

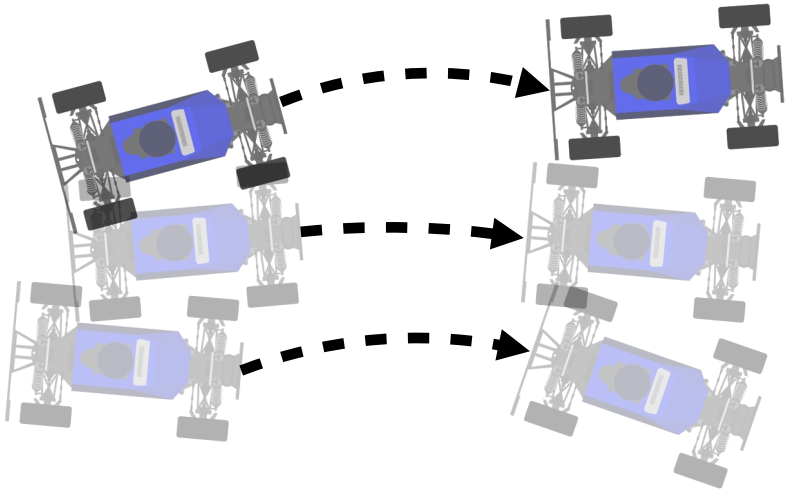
$$w_i = \frac{P(z_t|x_t^i)}{\sum_j P(z_t|x_t^j)}$$

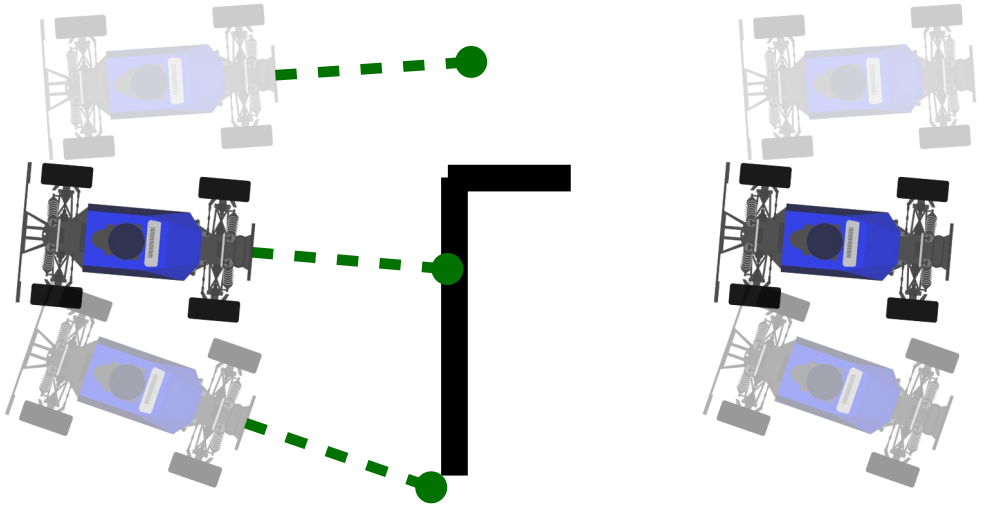




Resample particles from weighted distribution to give unweighted set of particles

Original: Normalized Importance Sampling





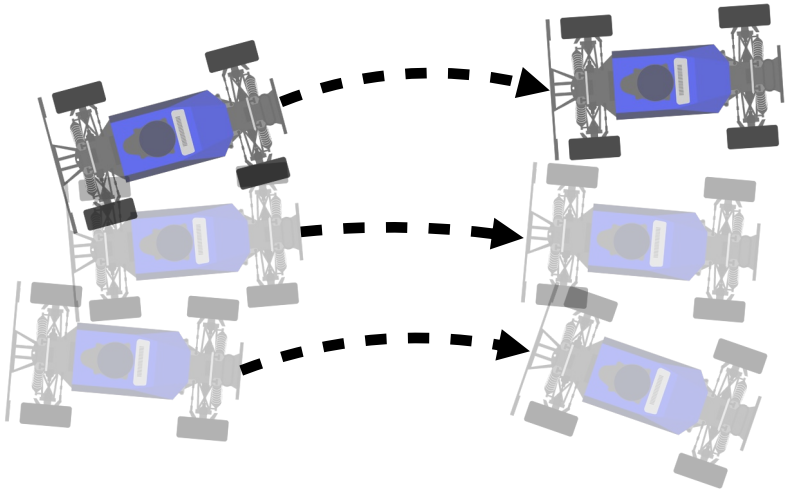
0.125

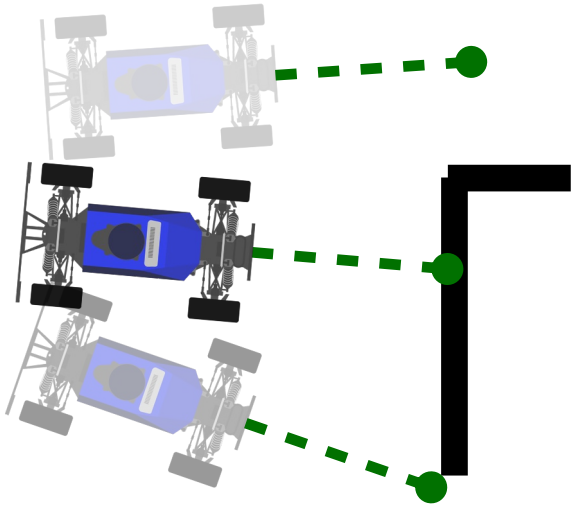
0.625

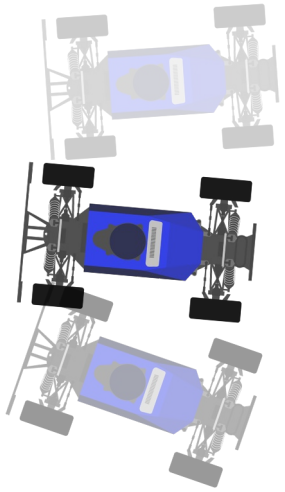
0.250

$$Bel(x_t) = \left\{ \begin{array}{cccc} \bar{x}_t^{(1)} & \bar{x}_t^{(2)} & \cdots & \bar{x}_t^{(M)} \\ w_t^{(1)} & w_t^{(2)} & \cdots & w_t^{(M)} \end{array} \right\}$$

New: Normalized Importance Sampling with Resampling



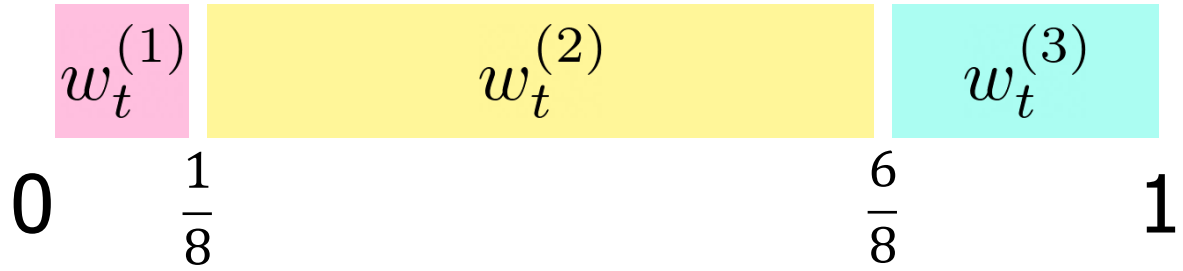




0.125

0.625

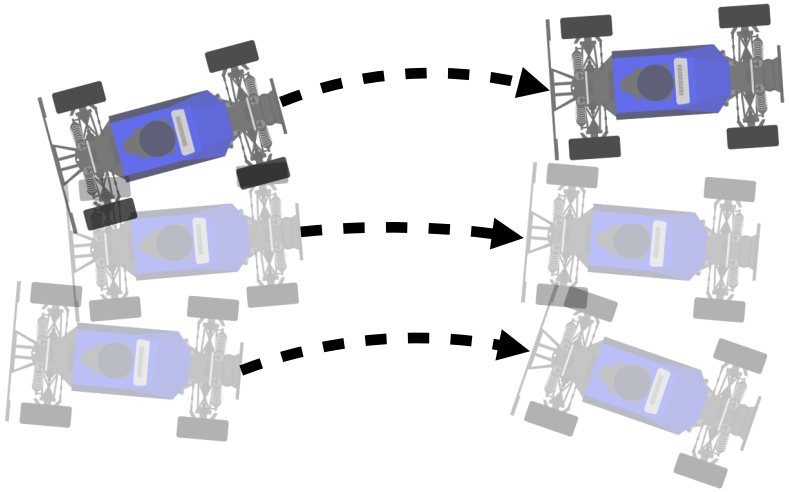
0.250

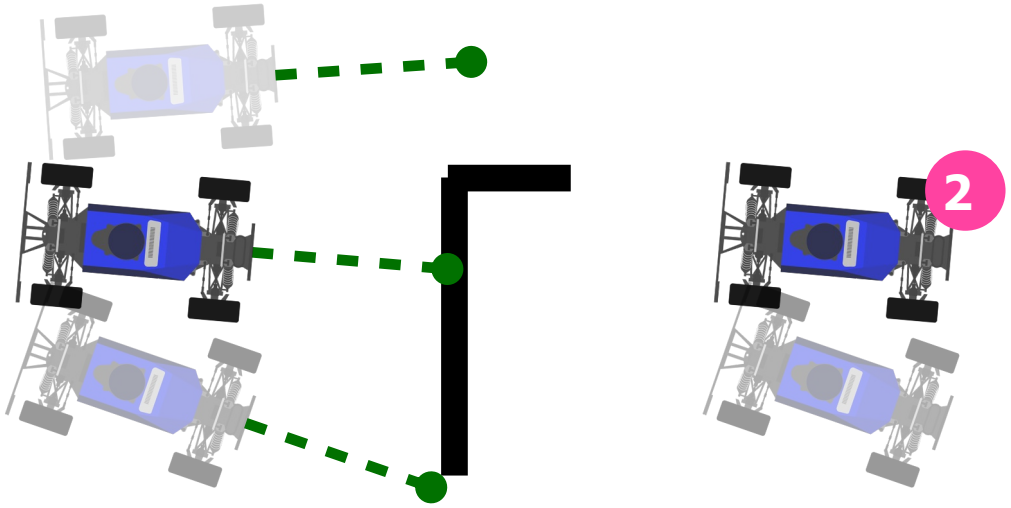


Here are your random numbers:

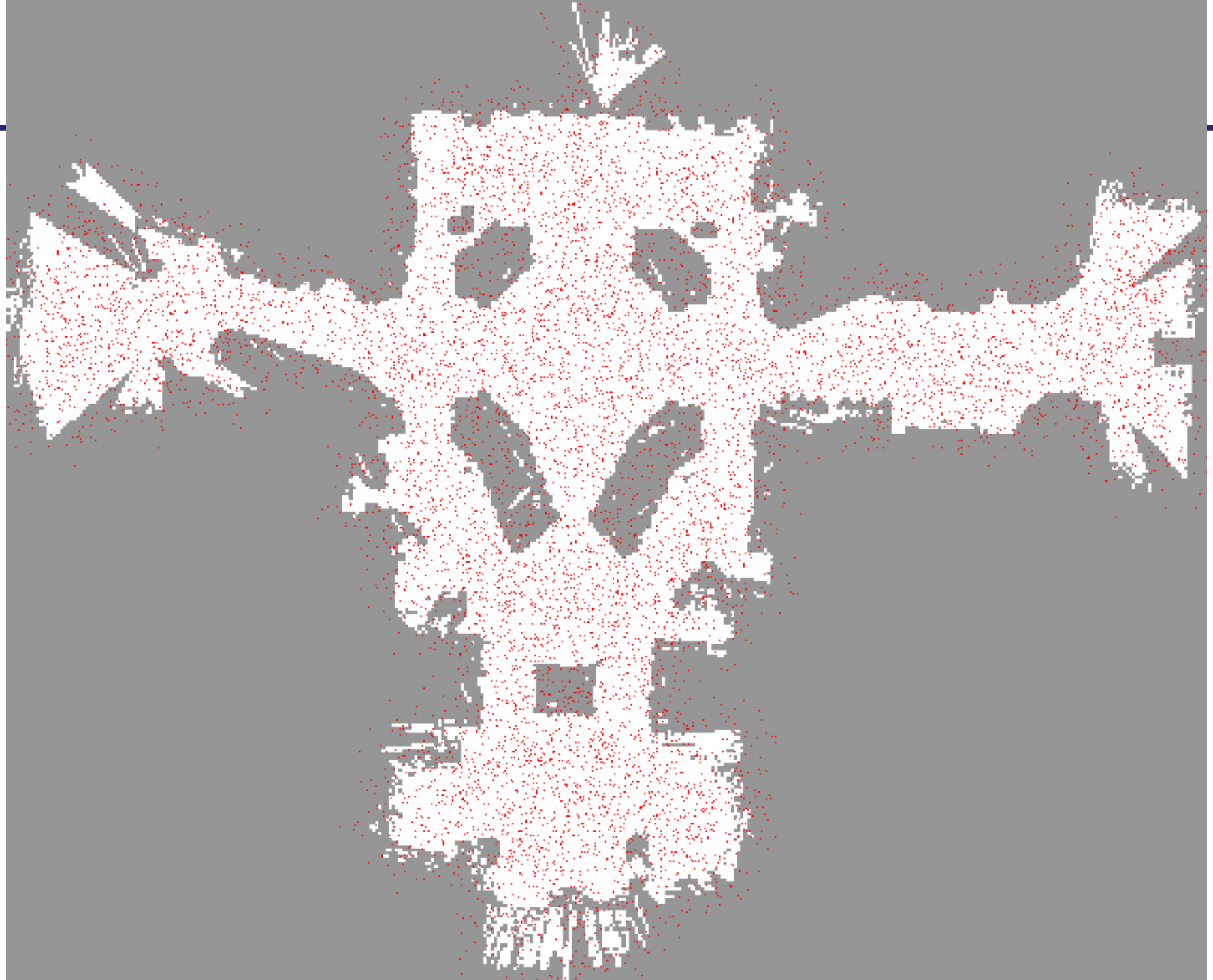
0.97
0.26
0.72

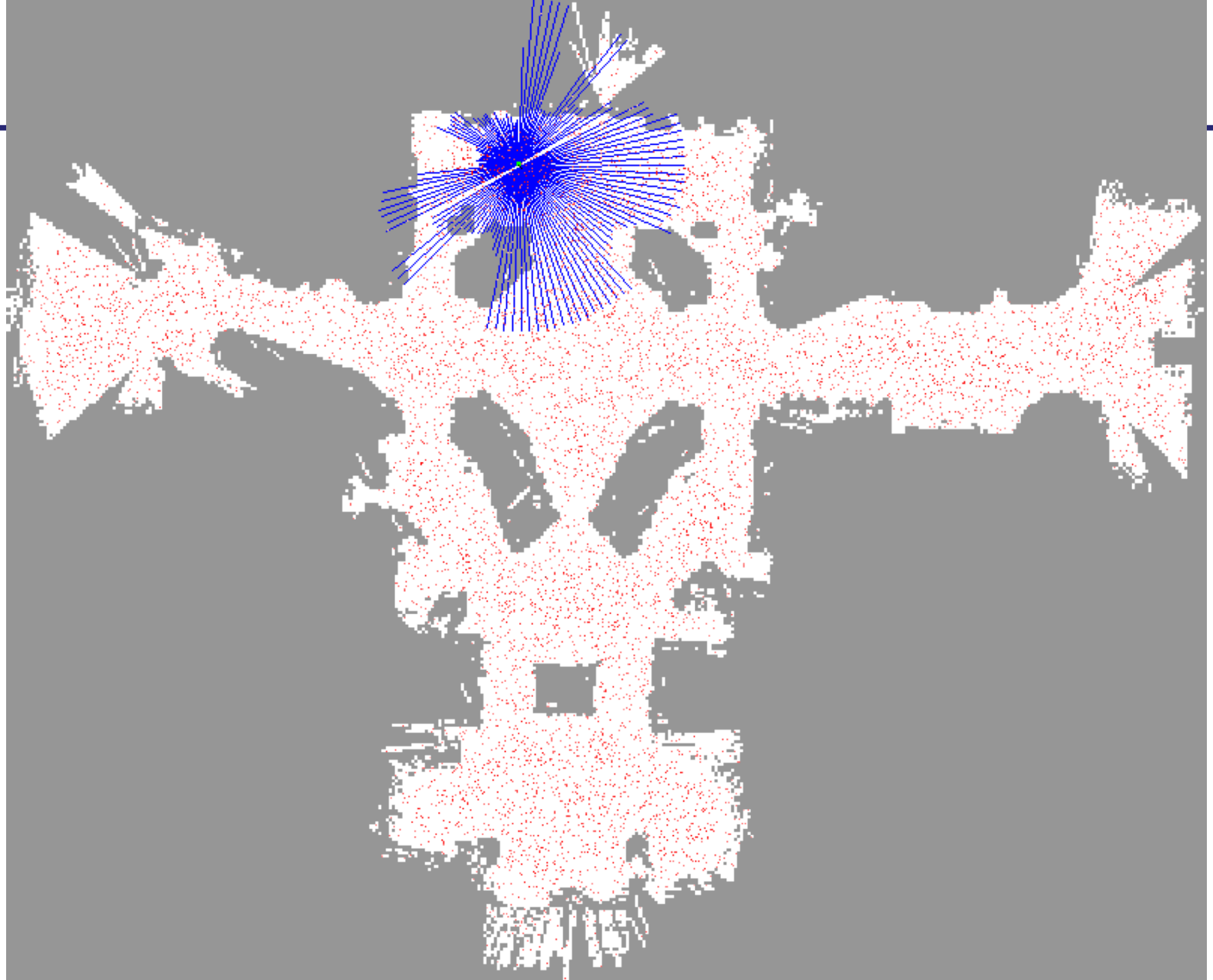
New: Normalized Importance Sampling with Resampling

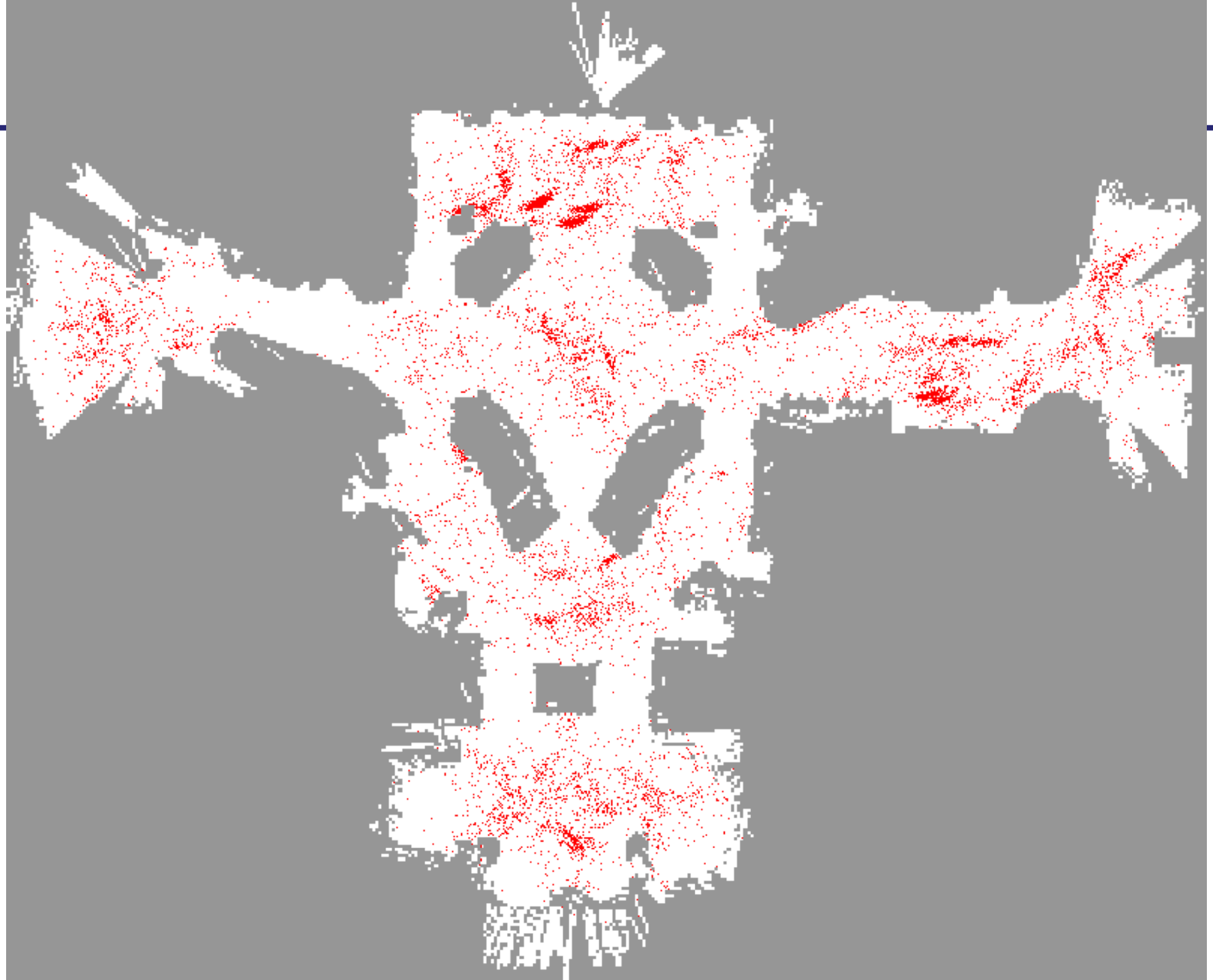


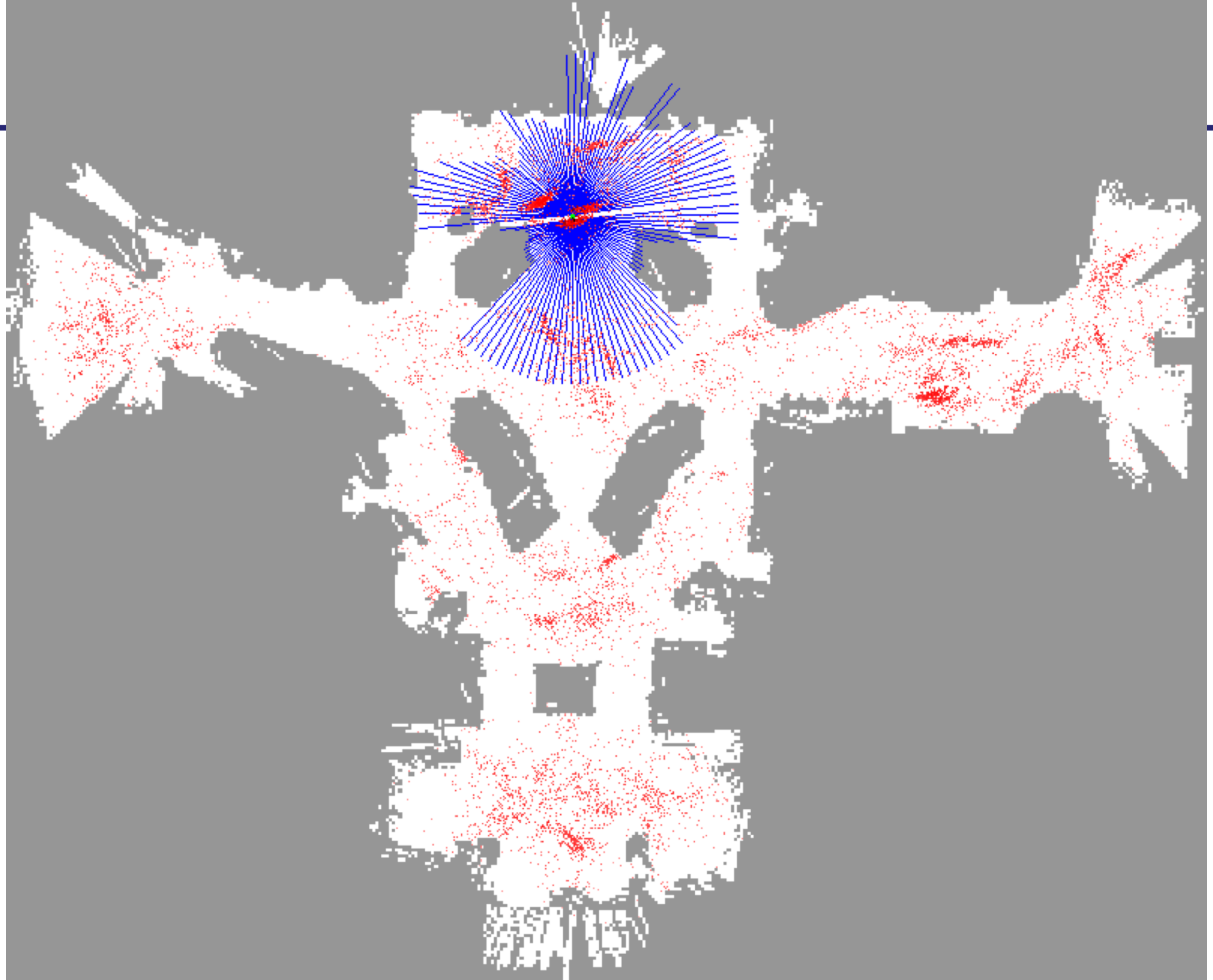


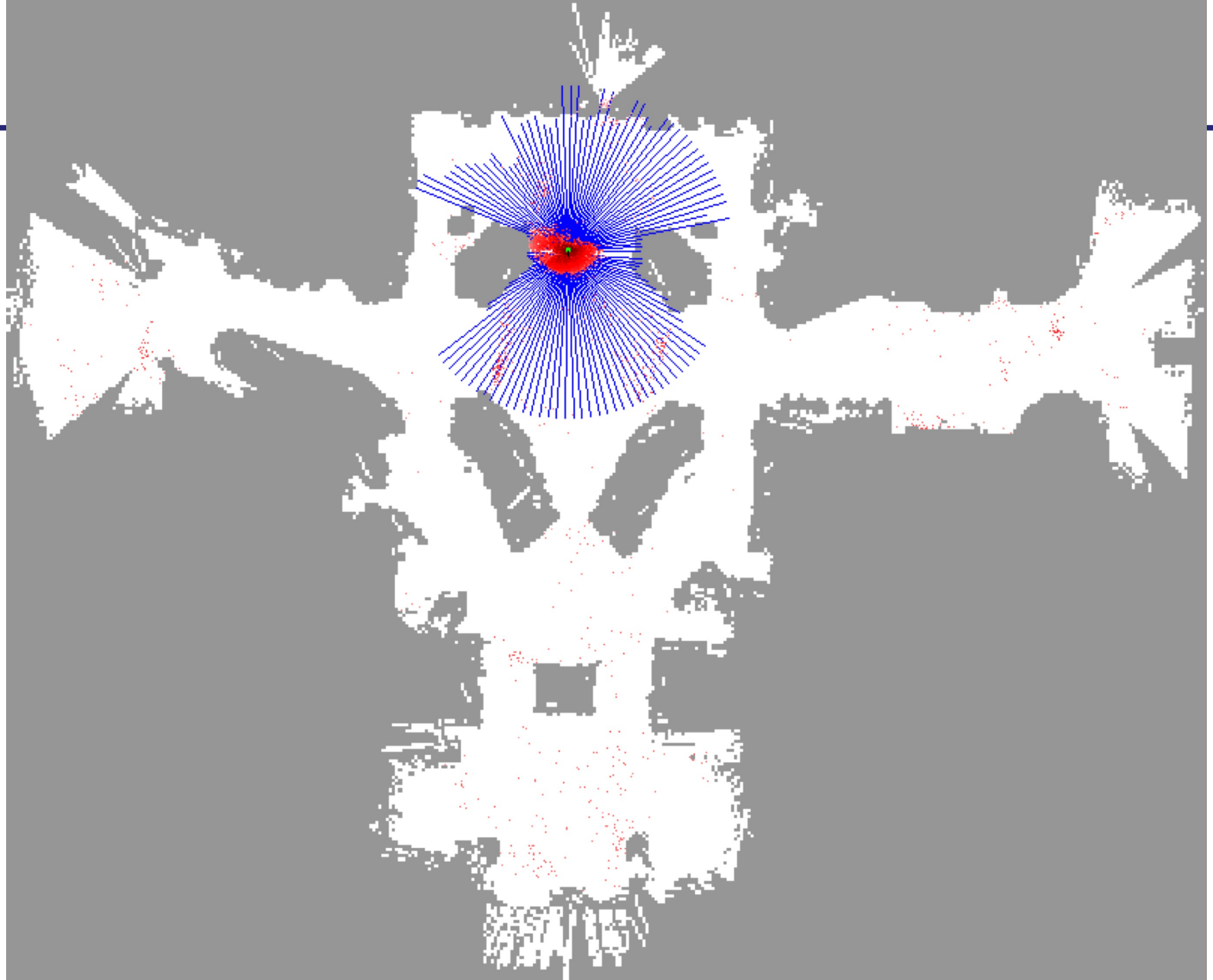
$$x_t^{(i)} \sim w_t^{(i)}, \; Bel(x_t) = \left\{ \begin{array}{ccc} x_t^{(1)} & \cdots & x_t^{(M)} \\ \frac{1}{M} & \cdots & \frac{1}{M} \end{array} \right\}$$

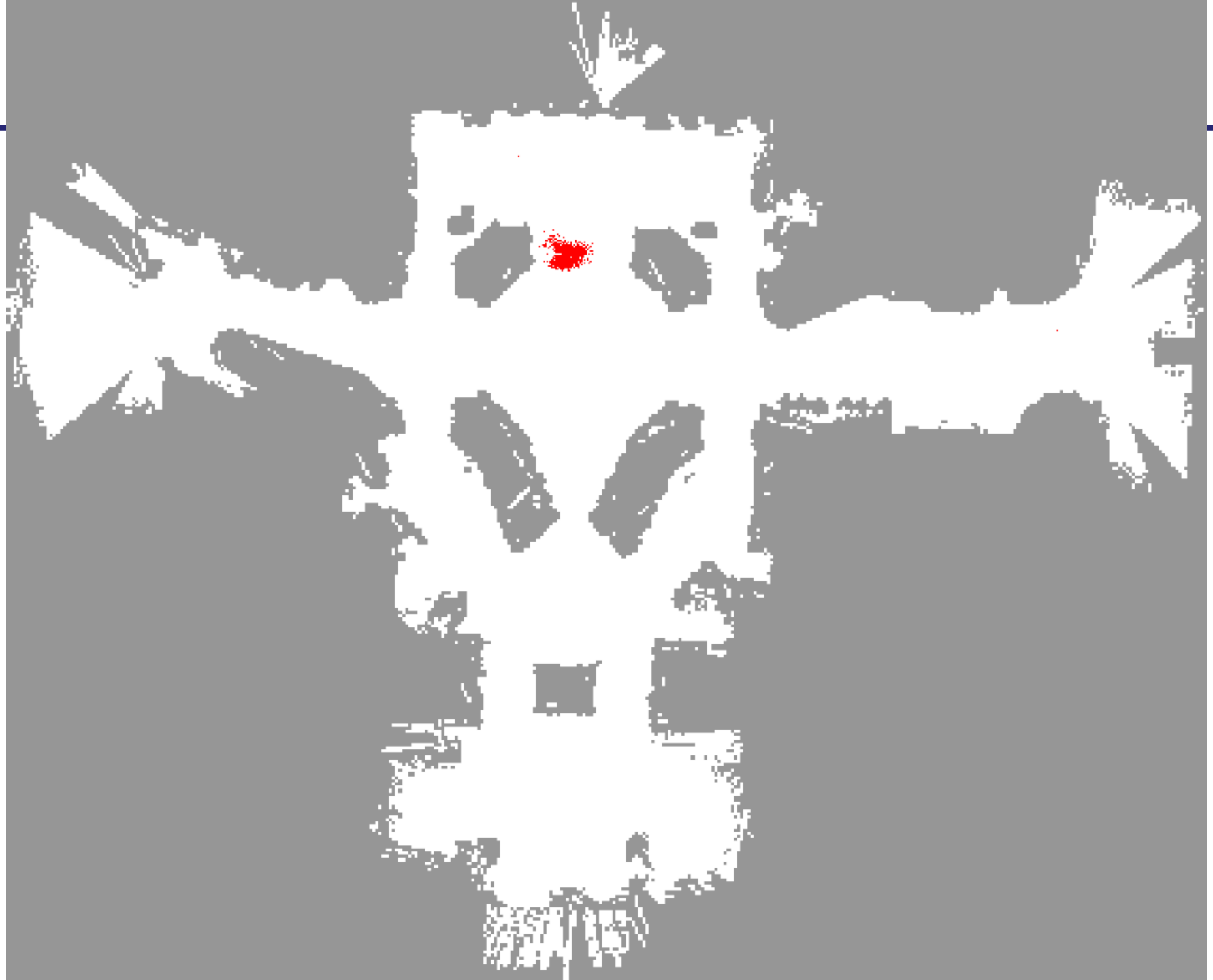




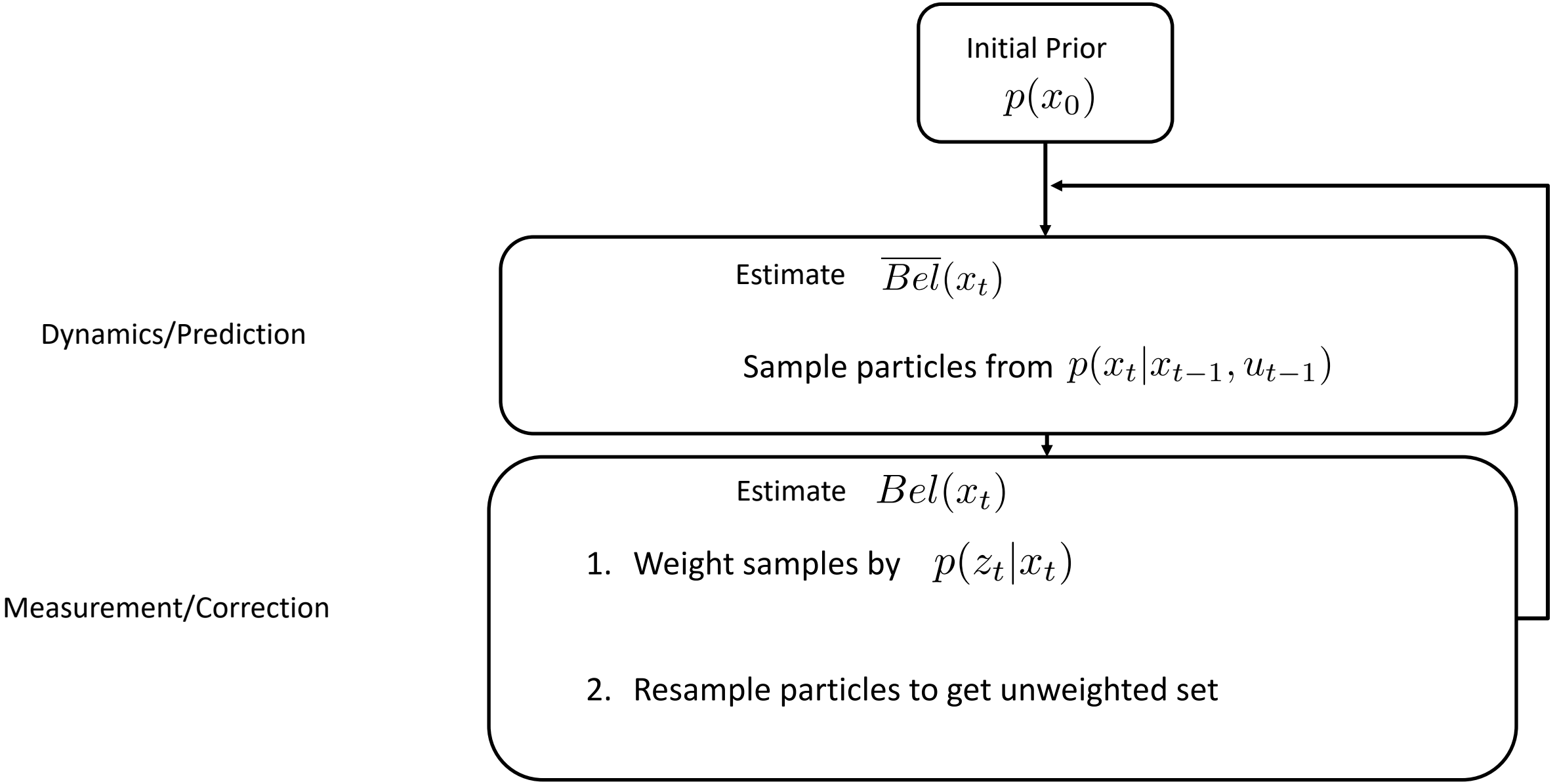








Overall Particle Filter algorithm – v2



Lecture Outline

(Whiteboard) Recap of Basic Particle Filtering

Particle Filter w/ Resampling

Practical Considerations

EKF + Applications

Problem 1: Two Room Challenge

Particles begin equally distributed, no motion or observation

All particles migrate to one room!

Reason: Resampling Increases Variance

50% prob. of resampling particle from Room 1 vs Room 2
31% prob. of preserving 50-50 particle split

All particles migrate to one room!

Idea 1: Judicious Resampling

Key idea: resample less often! (e.g., if the robot is stopped, don't resample). Too often may lose particle diversity, infrequently may waste particles

Common approach: don't resample if weights have low variance

Can be implemented in several ways: don't resample when…

…all weights are equal

…weights have high entropy

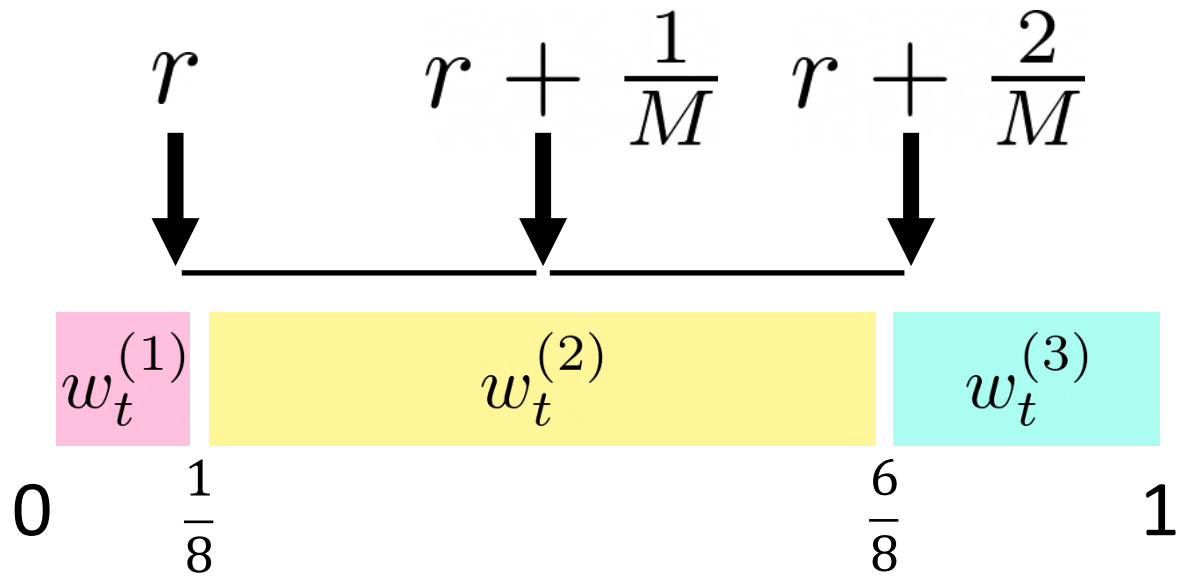
…ratio of max to min weights is low

Idea 2: Low-Variance Resampling

Sample one random number $r \sim \left[0, \frac{1}{M}\right]$

Covers space of samples more systematically (and more efficiently)

If all samples have same importance weight, won't lose particle diversity



Other Practical Concerns

How many particles is enough?

Typically need more particles at the beginning (to cover possible states)

KLD Sampling (Fox, 2001) adaptively increases number of particles when state uncertainty is high, reduces when state uncertainty is low

Particle filtering with overconfident sensor models

Squash sensor model prob. with power of 1/m (Lecture 3)

Sample from better proposal distribution than motion model

Manifold Particle Filter (Koval et al., 2017) for contact sensors

Particle starvation: no particles near current state

MuSHR Localization Project

Implement kinematic car motion model

Implement different factors of single-beam sensor model

Combine motion and sensor model with the Particle Filter algorithm

Lecture Outline

(Whiteboard) Recap of Basic Particle Filtering

Particle Filter w/ Resampling

Practical Considerations

EKF + Applications

Ok, but maintaining particles is kind of a pain.

Can I reuse Kalman Filter math for non-linear systems?

Nonlinear Dynamic Systems

- Most realistic robotic problems involve nonlinear functions

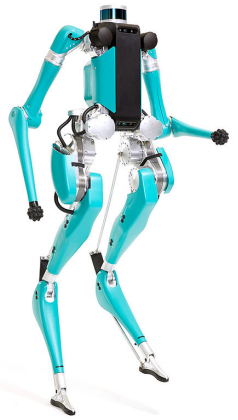
$$x_{t+1} = g(x_t, u_t) + \epsilon_t$$

$$z_t = h(x_t) + \delta_t$$

$$\epsilon_t \sim \mathcal{N}(0, Q)$$

$$\delta_t \sim \mathcal{N}(0, R)$$

Non-linear system



Additive Gaussian noise

More reasonable assumption than linear Gaussian. More on non-Gaussian systems next time

How do we deal with non-linearity?

- Differentiable non-linear functions can be expressed via their Taylor expansion

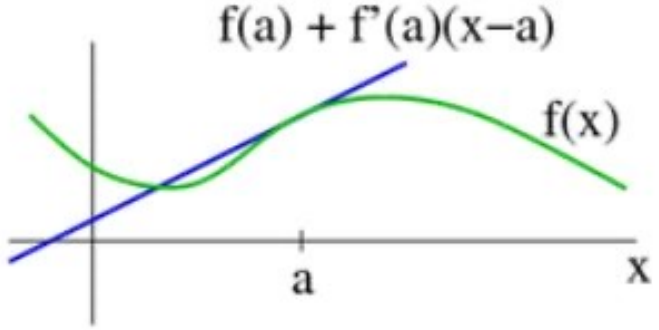
$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots,$$

$$f(x) \approx f(a) + \frac{f'(a)}{1!}(x-a)$$

Dropping higher order terms, when x-a is small enough

Linear function in x

Pretend that your function is linear in this neighborhood
→ Reapprox in a new neighborhood



EKF Linearization: First Order Taylor Series Expansion

- Idea behind EKF: Linearize the dynamics and measurement around current μ_t
- Dynamics Model (linearize around previous belief):

$$x_{t+1} = g(x_t, u_t) + \epsilon_t \approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

$$= g(\mu_t, u_t) + G(x_t - \mu_t) + \epsilon_t$$

- Measurement Model (linearize around post dynamics belief):

$$z_t = h(x_t) + \delta_t \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t \approx h(\bar{\mu}_t) + H(x_t - \bar{\mu}_t) + \delta_t$$

Now everything is linear → back to Kalman filtering!

Modified System under EKF Linearization

- Start by linearizing dynamics model under current belief
- Dynamics Model (linearize around previous belief):

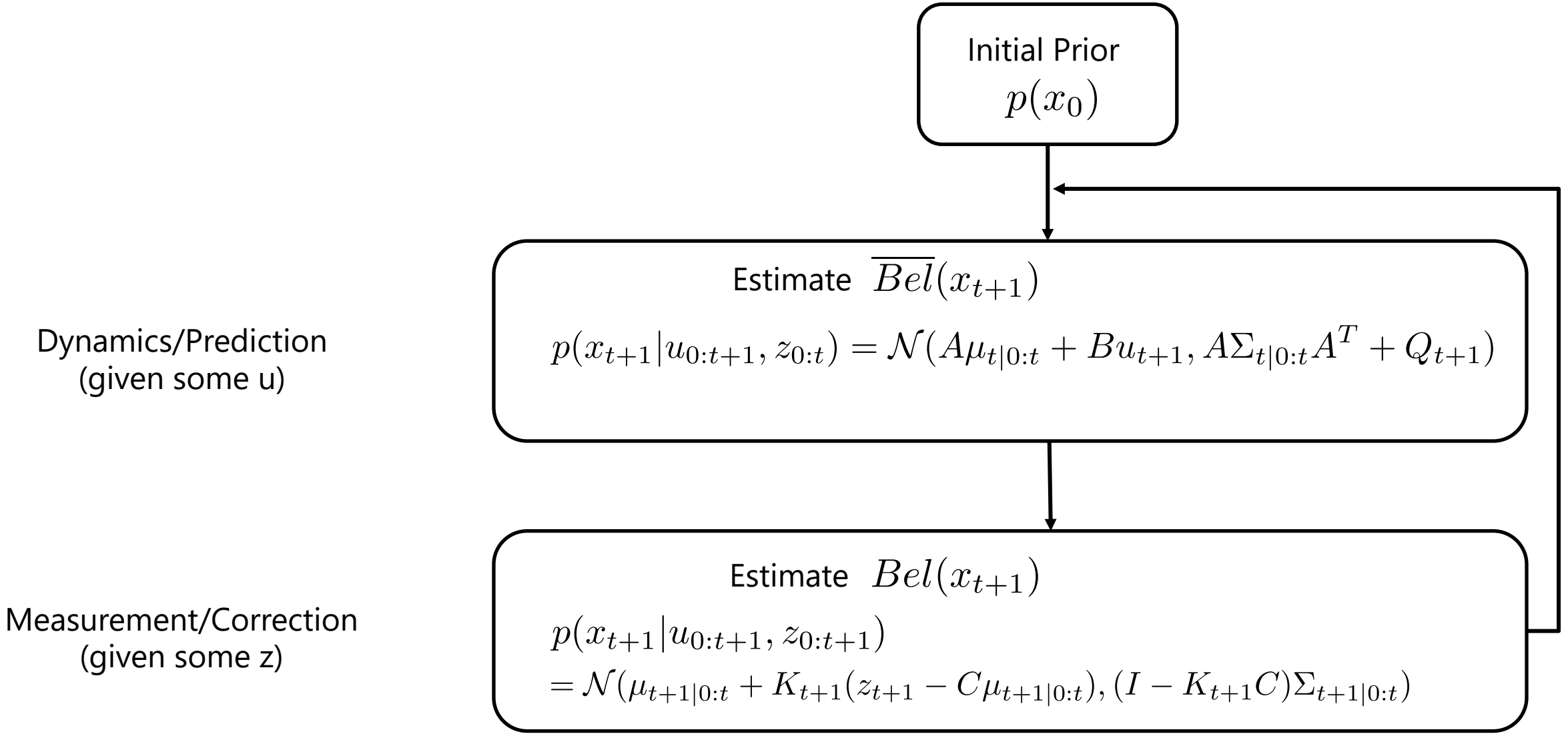
$$x_{t+1} = g(x_t, u_t) + \epsilon_t \qquad \approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

- Perform dynamics update
- Linearize measurement around post dynamics belief

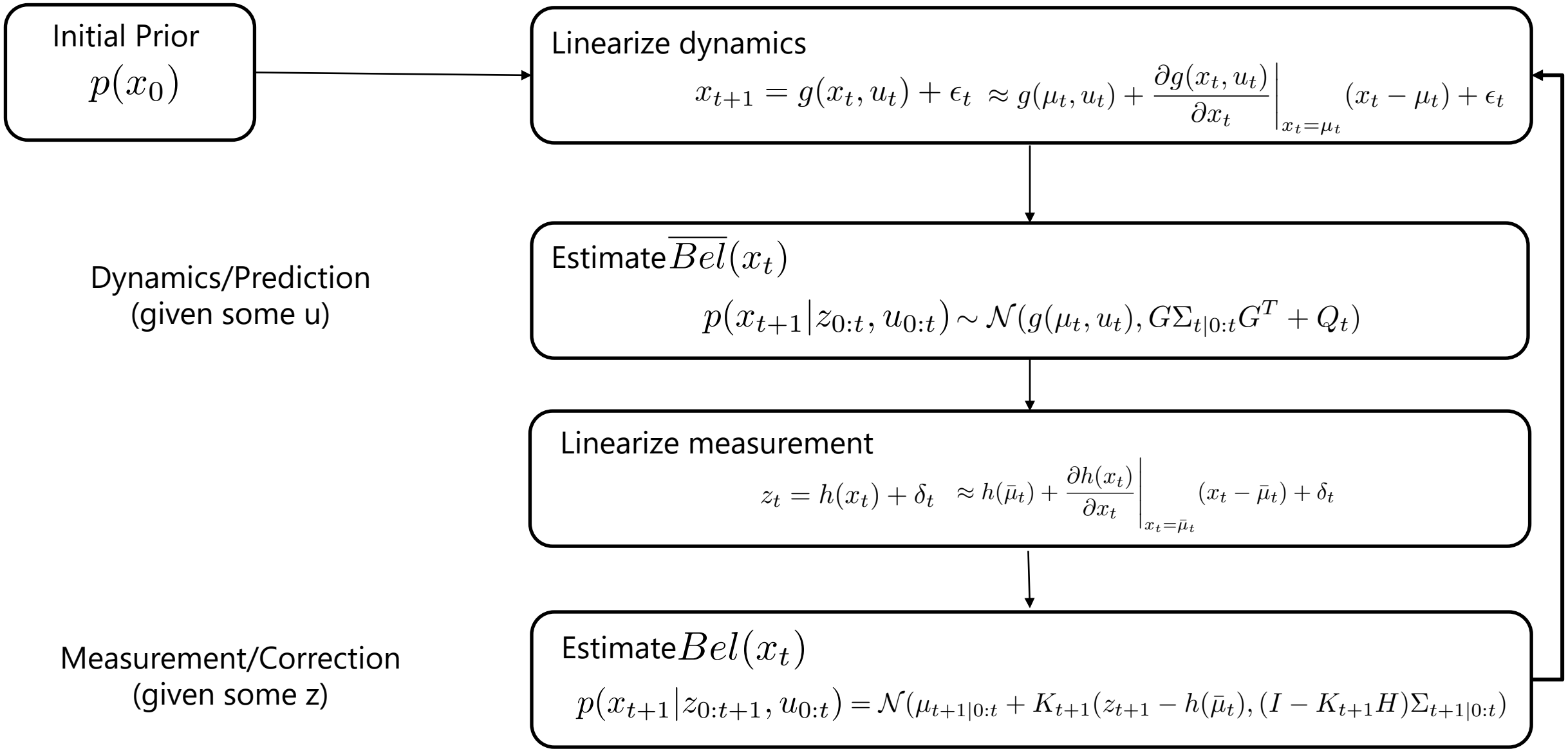
$$z_t = h(x_t) + \delta_t \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t \approx h(\bar{\mu}_t) + H(x_t - \bar{\mu}_t) + \delta_t$$

- Perform measurement update
- Repeat

Original Kalman Filter Algorithm

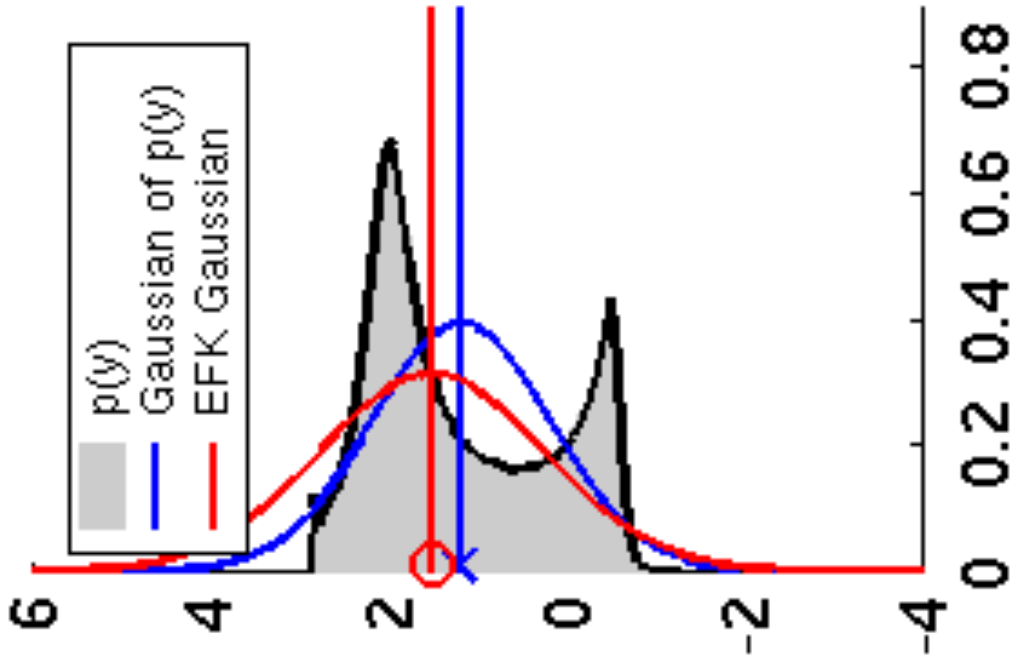


EKF Algorithm – linearize non-linear functions



Why might we still want to use particle filters?

- **Non-linear functions**
- **Non-Gaussian functions** ⟵ EKFs still require Gaussian Distributions



Ok so what have we learned

Bayesian Filtering!

Key Idea: Apply Markov to get a recursive update!

Step 0. Start with the belief at time step t-1

$$bel(x_{t-1})$$

Step 1: Prediction - push belief through dynamics given action

$$\overline{bel}(x_t) = \sum P(x_t|u_t, x_{t-1})bel(x_{t-1})$$

Step 2: Correction - apply Bayes rule given measurement

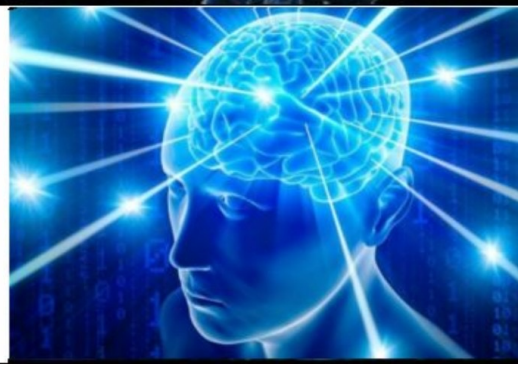
$$bel(x_t) = \eta P(z_t|x_t)\overline{bel}(x_t)$$

Motion and Measurement Model

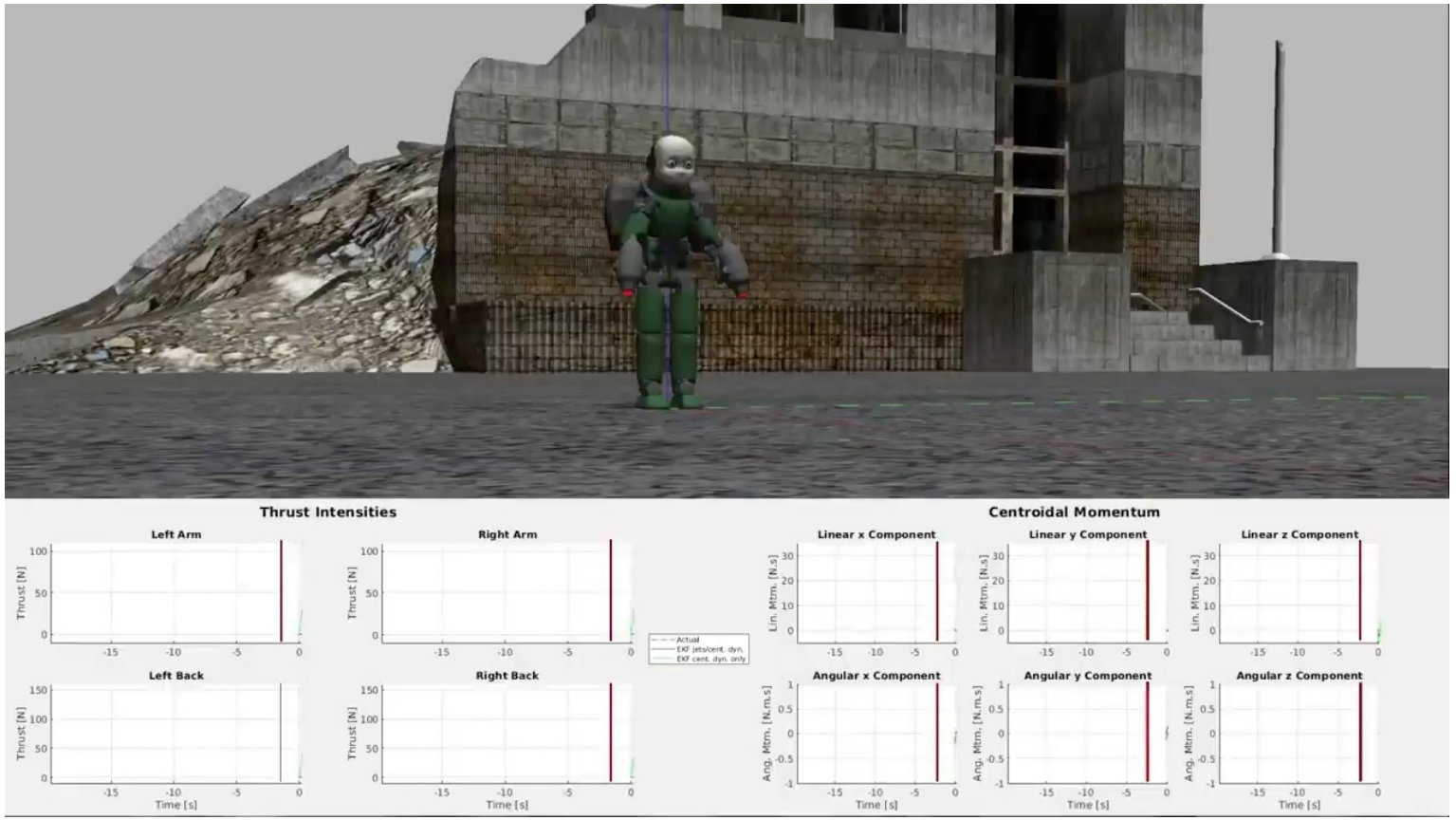
Linear Gaussian
– Kalman Filter

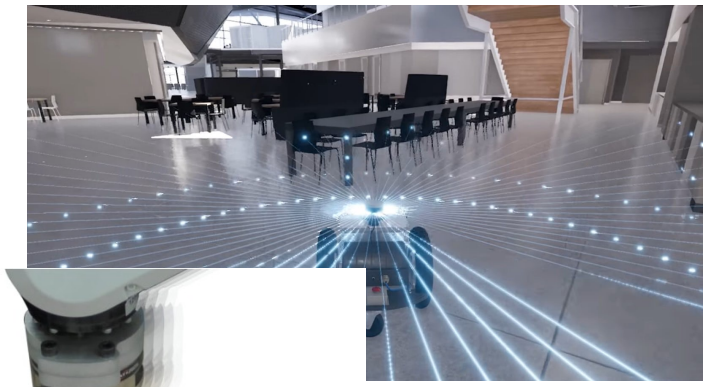
Nonlinear Gaussian
– Extended Kalman Filter

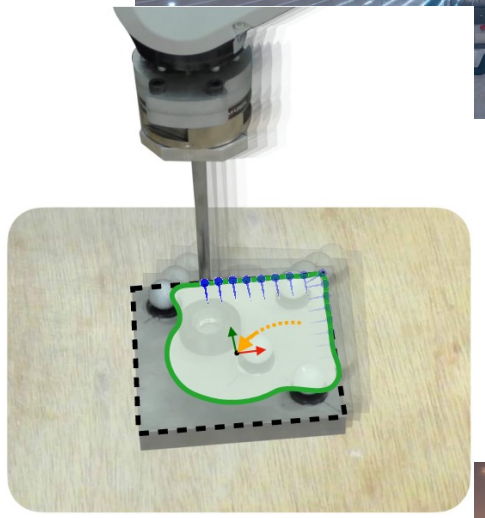
Nonlinear non-gaussian
– Particle Filter

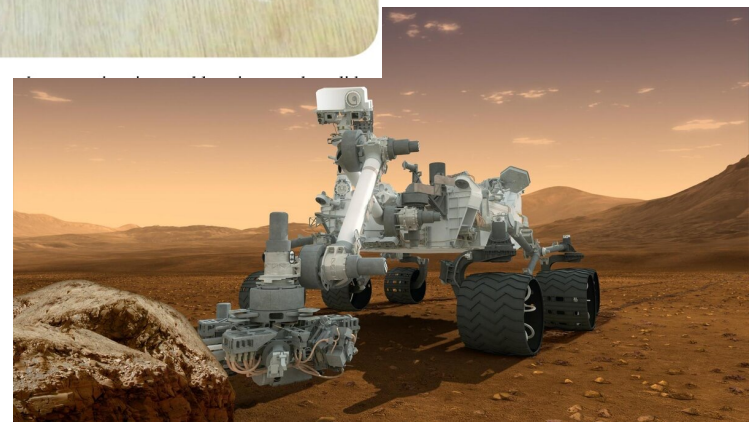


Why is this useful - Localization





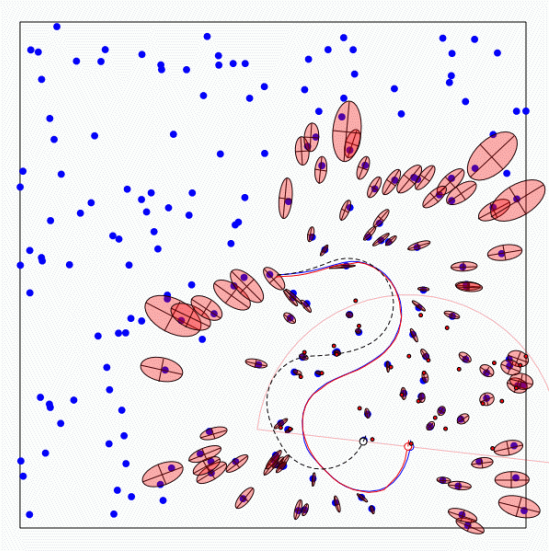




Why is this useful - Localization

Why is this useful - SLAM

- So far, the maps have been assumed to be known → often untrue → SLAM problem
- A robot is exploring an unknown, static environment.



Given:

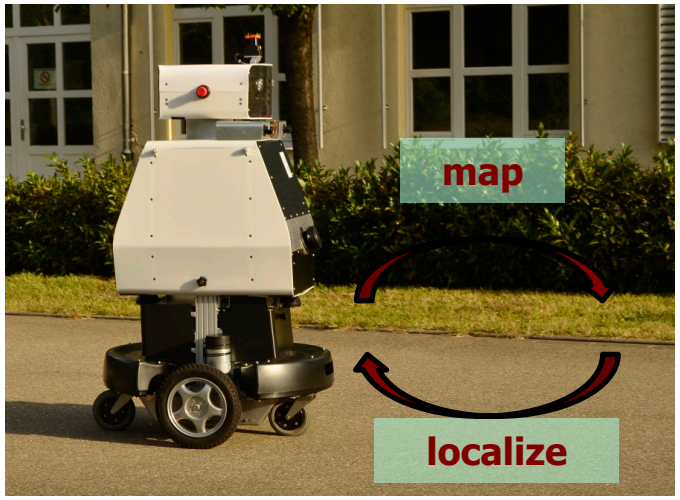
The robot's controls (u)

Observations of nearby features (z)

Estimate:

Map of features (x)

Path of the robot (x)



Why is this useful - SLAM

Class Outline