Principles of Safe Autonomy: Lecture 12-13: Filtering and Robot Localization

Sayan Mitra

March 9, 2020

Reference: Probabilistic Robotics by Sebastian Thrun, Wolfram Burgard, and Dieter Fox

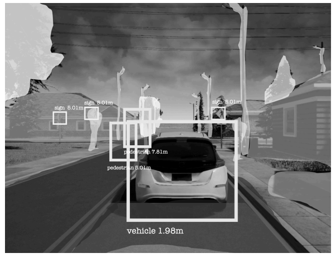
Slides: From the book's website

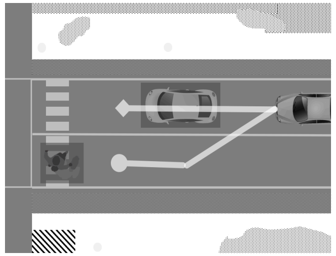
Announcements

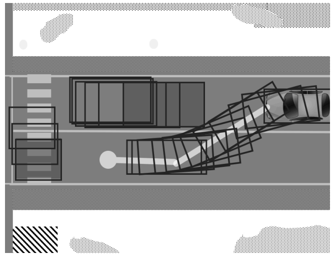
- No final exam
  - Unless Class Project has to be significantly downgraded because of coronavirus and University closure
- New date for Midterm 2: Wed April 15th
- MP4 + HW3 will be release this week
- Classes may go online after spring break
  - Install zoom application
  - Stay healthy and stay tuned

GEM platform

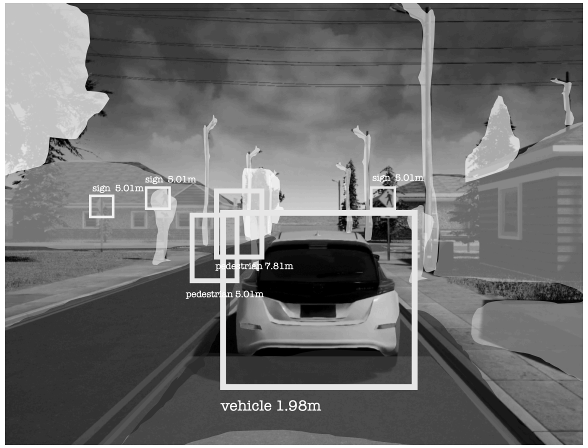
Autonomy pipeline







Sensing	Perception	Decisions and planning	Control
Physics-based models of camera, LIDAR, RADAR, GPS, etc.	Programs for object detection, lane tracking, scene understanding, etc.	Programs and multi-agent models of pedestrians, cars, etc.	Dynamical models of engine, powertrain, steering, tires, etc.



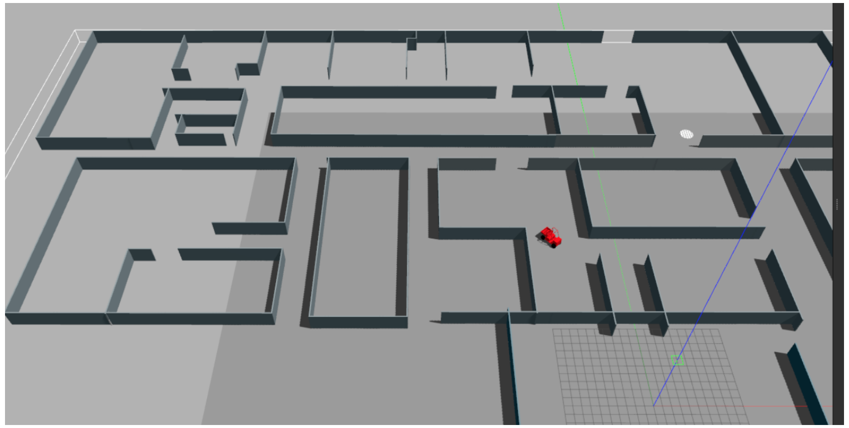
Perception

Programs for object detection, lane tracking, scene understanding, etc.

Outline

- Introduction: Localization problem, taxonomy
- Discrete Bayes Filter
- Histogram filter
  - Grid localization
- Particle filter
  - Monte Carlo localization
- Conclusions

Localization problem (MP4)

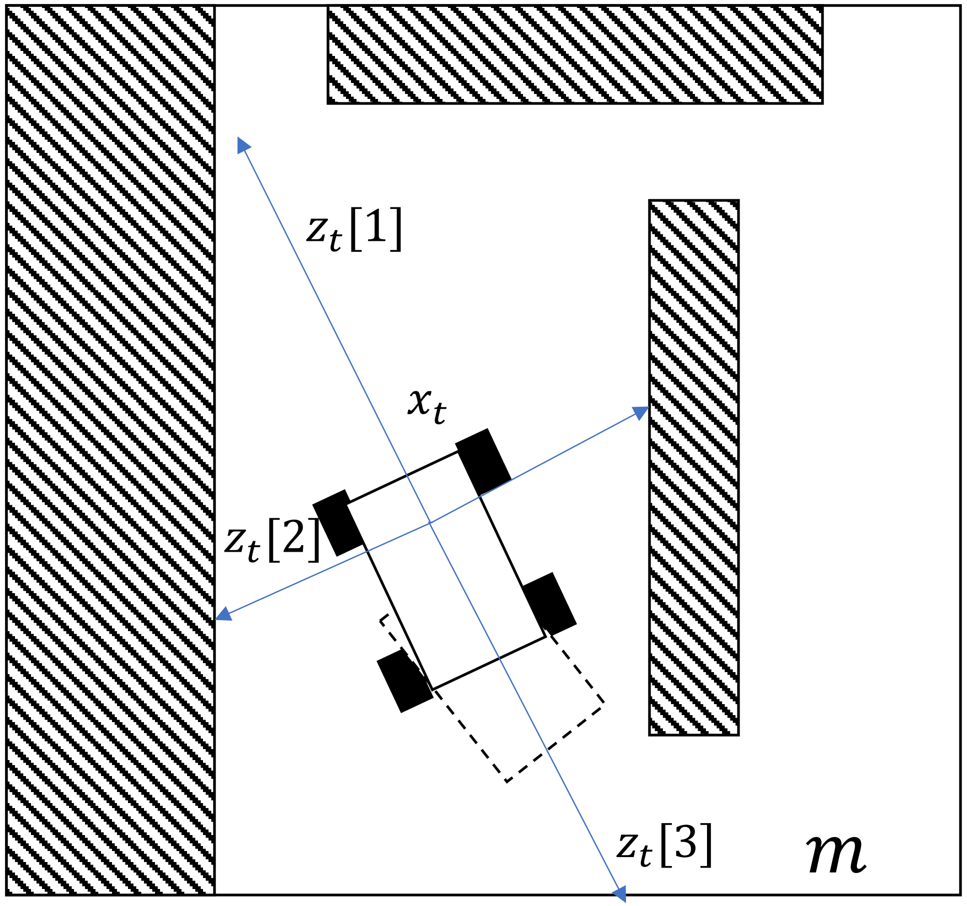


- Determine the pose of the robot relative to the *given map* of the environment
  - Pose: position, velocity, attitude, angles
  - Also known as position or state estimation problem

- First: why localize?
- How does your robot know its position in ECEB?
- "Localization is the biggest hack in autonomous cars" --- people drive without localization

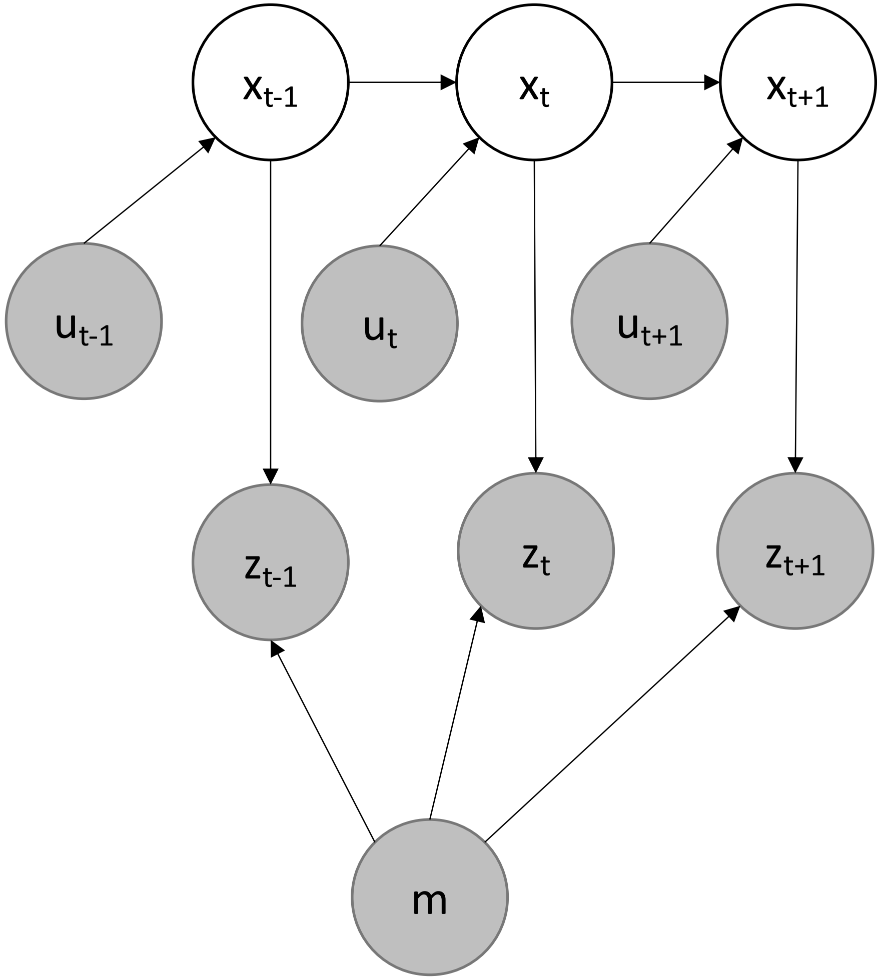
Setup

- System evolution: $x_{t+1} = f(x_t, u_t)$
  - x_t: unknown state of the system at time t
  - u_t: known control input at time t
  - f: known dynamic function, possibly stochastic
- Measurement: $z_t = g(x_t, m)$
  - z_t: known measurement of state x_t at time t
  - m: unknown underlying map
  - g: known measurement function



This is not exactly the measurement model of MP4

Localization as coordinate transformation



Shaded known:
map (m), control inputs (u), measurements(z). White nodes to be determined (x)

maps (m) are described in global coordinates. Localization = establish *coord transf.* between m and robot's local coordinates

Transformation used for objects of interest (obstacles, pedestrians) for decision, planning and control

Localization taxonomy

Global vs Local

- Local: assumes initial pose is known, has to only account for the uncertainty coming from robot motion (*position tracking problem)*
- **Global**: initial pose unknown; harder and subsumes position tracking
- Kidnapped robot problem: during operation the robot can get teleported to a new unknown location (models failures)

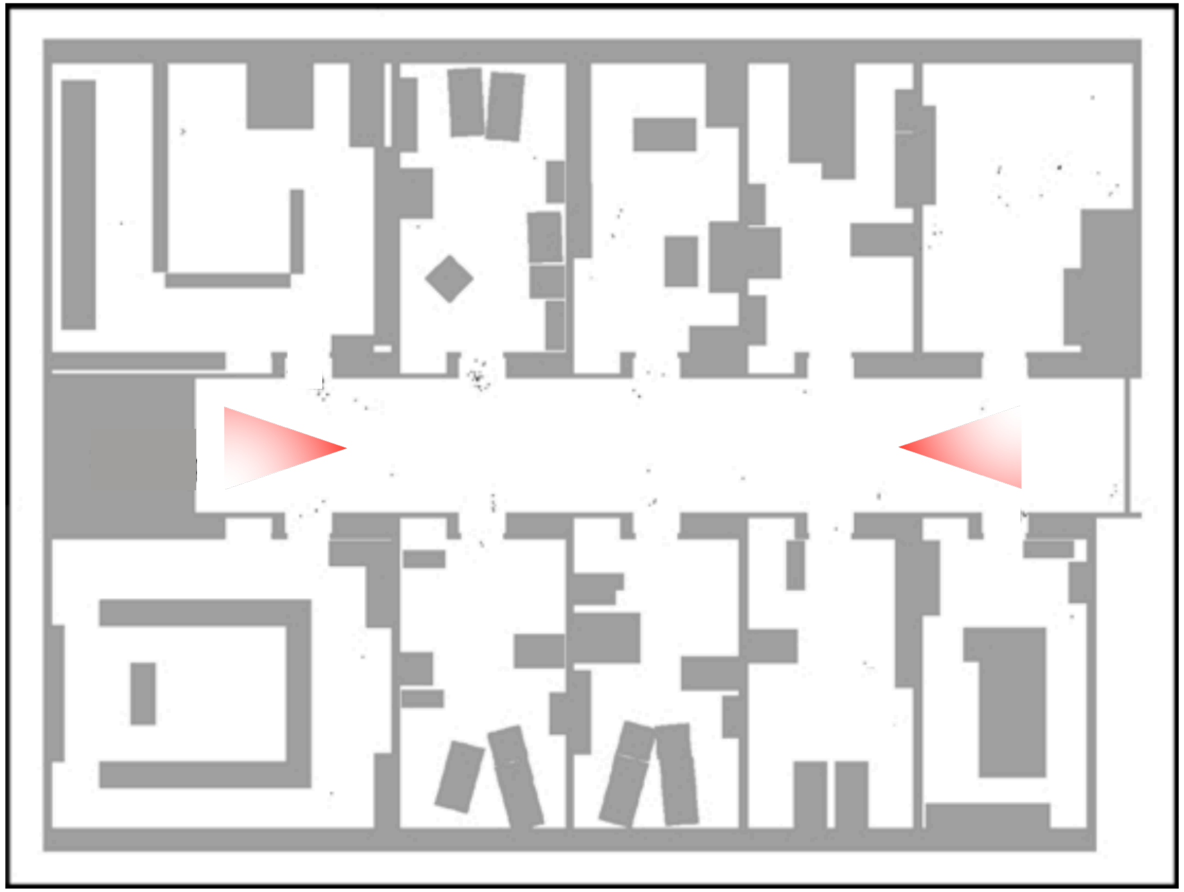
Static vs Dynamic Environments

Single vs Multi-robot localization

Passive vs Active Approaches

- **Passive**: localization module only observes and is controlled by other means; motion not designed to help localization (Filtering problem)
- Active: controls robot to improve localization

Ambiguity in global localization arising from locally symmetric environment



Discrete Bayes Filter Algorithm

- System evolution: $x_{t+1} = f(x_t, u_t)$
  - x_t: state of the system at time t
  - u_t: control input at time t
- Measurement: $z_t = g(x_t, m)$
  - z_t: measurement of state x_t at time t
  - m: unknown underlying map

Setup, notations

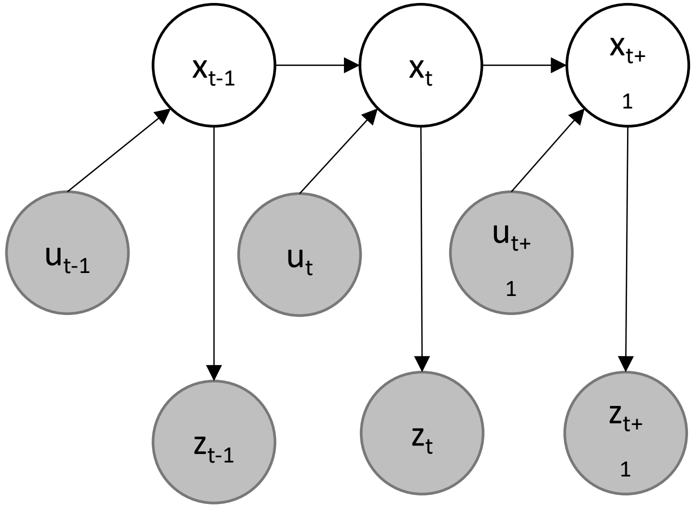
- Discrete time model
- $x_{t_1:t_2} = x_{t_1}, x_{t_1+1}, x_{t_1+2}, \ldots, x_{t_2}$ sequence of robot states $t_1$to t_2
- Robot takes one measurement at a time
  - $z_{t_1:t_2} = z_{t_1}, \ldots, z_{t_2}$ sequence of all measurements from $t_1$to t_2
- Control also exercised at discrete steps
  - $u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2}$ sequence control inputs

State evolution and measurement models

Evolution of state and measurements governed by probabilistic laws

$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t})$ describes motion/state evolution model

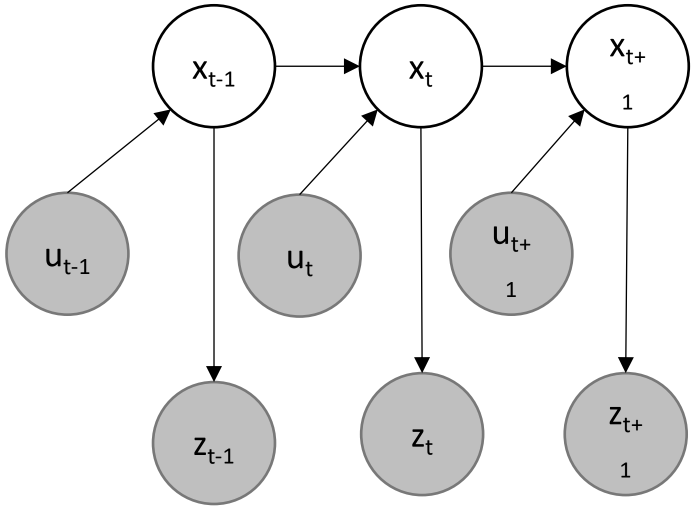
- If state is complete, sufficient summary of the history then
  - $p(x_t \mid x_{0:t-1}, z_{0:t-1}, u_{0:t-1}) = p(x_t \mid x_{t-1}, u_t)$ state transition prob.
  - $p(x' \mid x, u)$ if transition probabilities are time invariant



Measurement model

Measurement process $p(z_t \mid x_{0:t}, z_{1:t-1}, u_{0:t-1})$

- Again, if state is complete
- $p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t)$
- $p(z_t \mid x_t)$: measurement probability
- $p(z \mid x)$: time invariant measurement probability



Beliefs

Belief: Robot's knowledge about the state of the environment

True state is unknowable / measurable typically, so, robot must infer state from data and we have to distinguish this inferred/estimated state from the actual state x_t

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

Posterior distribution over state at time t given all past measurements and control

Prediction: $\overline{bel}(x_t) = p(x_t | z_{1:t-1}, u_{1:t})$

Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is called correction or measurement update

Recursive Bayes Filter

Algorithm Bayes_filter($bel(x_{t-1}), u_t, z_t$)

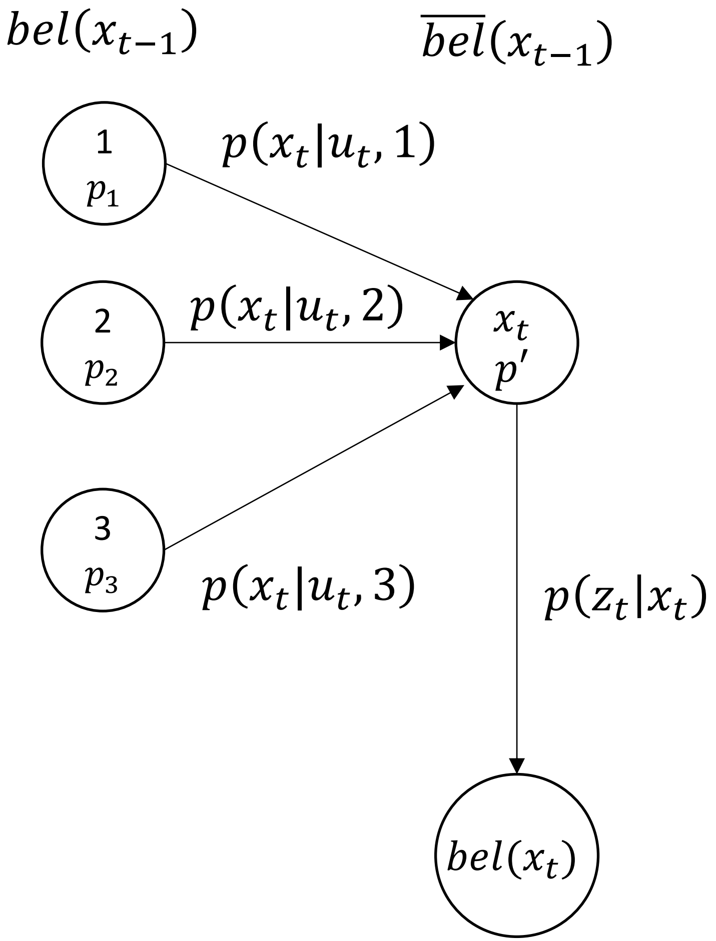
for all x_t do:

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

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

end for

return $bel(x_t)$



Histogram Filter or Discrete Bayes Filter

Finitely many states $x_i, x_k, etc.$ Random state vector X_t

$p_{k,t}$: belief at time t for state x_k; discrete probability distribution

Algorithm Discrete_Bayes_filter($\{p_{k,t-1}\}, u_t, z_t$**):**

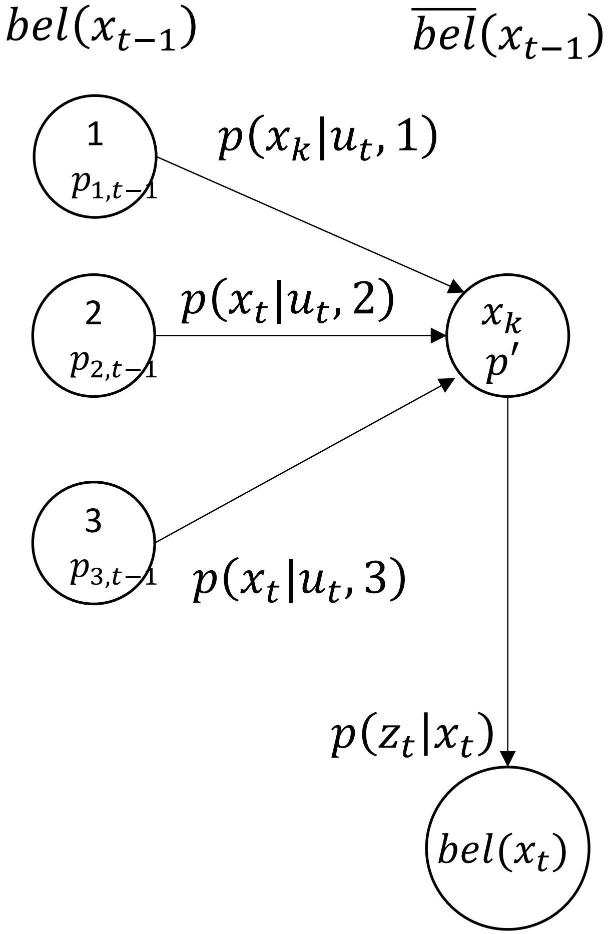
for all k do:

$$\bar{p}_{k,t} = \sum_i p(X_t = x_k | u_{t,} X_{t-1} = x_i) p_{i,t-1}$$

$$p_{k,t} = \eta\, p(z_t \,| X_t = x_k) \bar{p}_{k,t}$$

end for

return $\{p_{k,t}\}$



Grid Localization

- Solves global localization in some cases kidnapped robot problem
- Can process raw sensor data
  - No need for feature extraction
- Non-parametric
  - In particular, not bound to unimodal distributions (unlike Extended Kalman Filter)

Grid localization

Algorithm Grid_localization ($\{p_{k,t-1}\}, u_t, z_t, m$)

for all k do:

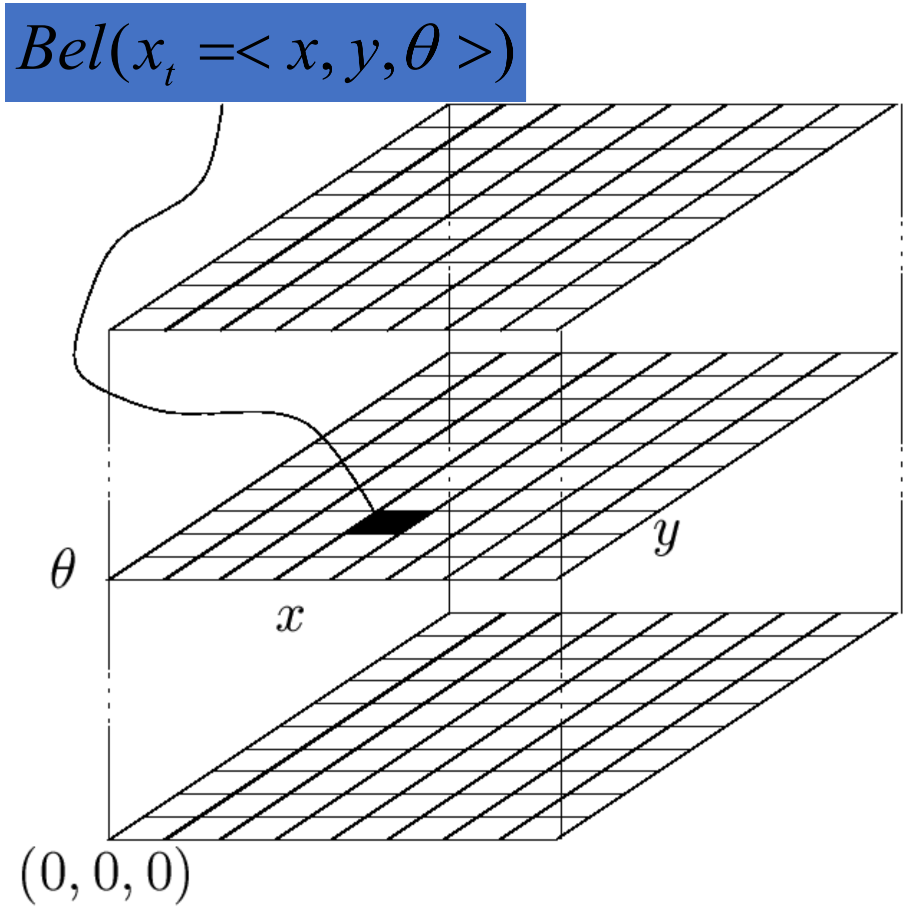
$$\bar{p}_{k,t} = \sum_i p_{i,t-1}\, \boldsymbol{motion\_model}(mean(x_k), u_t, mean(x_i))$$

$$p_{k,t} = \eta\, \bar{p}_{k,t} \boldsymbol{measurement\_model}(z_t, mean(x_k), m)$$

end for

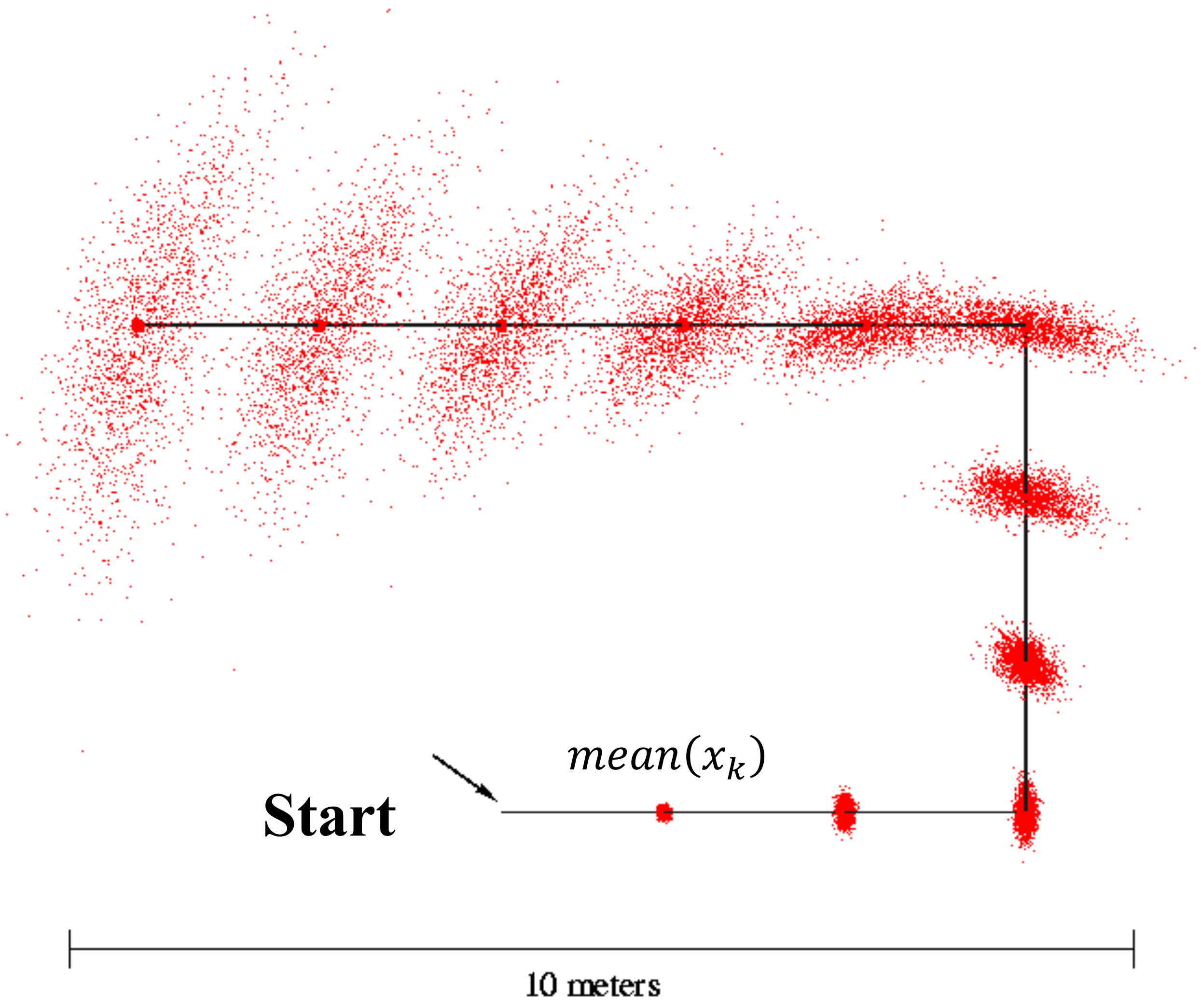
return $bel(x_t)$

Piecewise Constant Representation

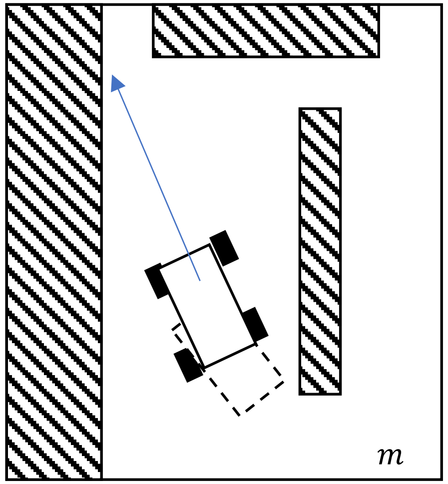


Fixing an input u_t we can compute the new belief

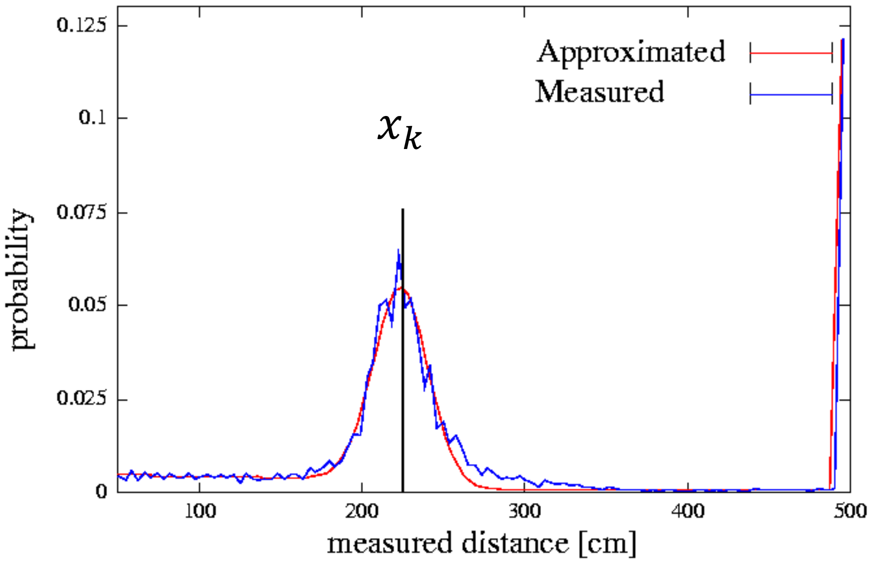
Motion Model without measurements



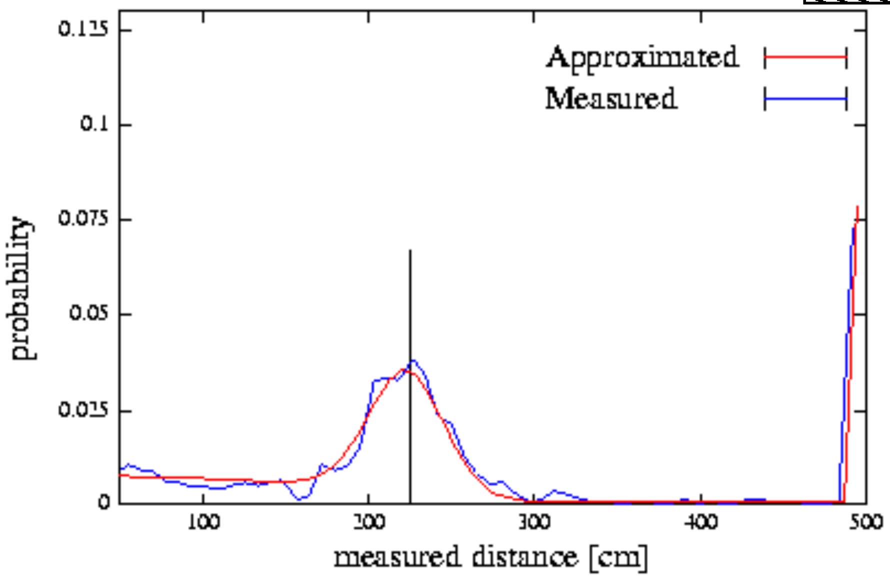
Proximity Sensor Model



$$p(z_t \mid X_t = x_k)$$

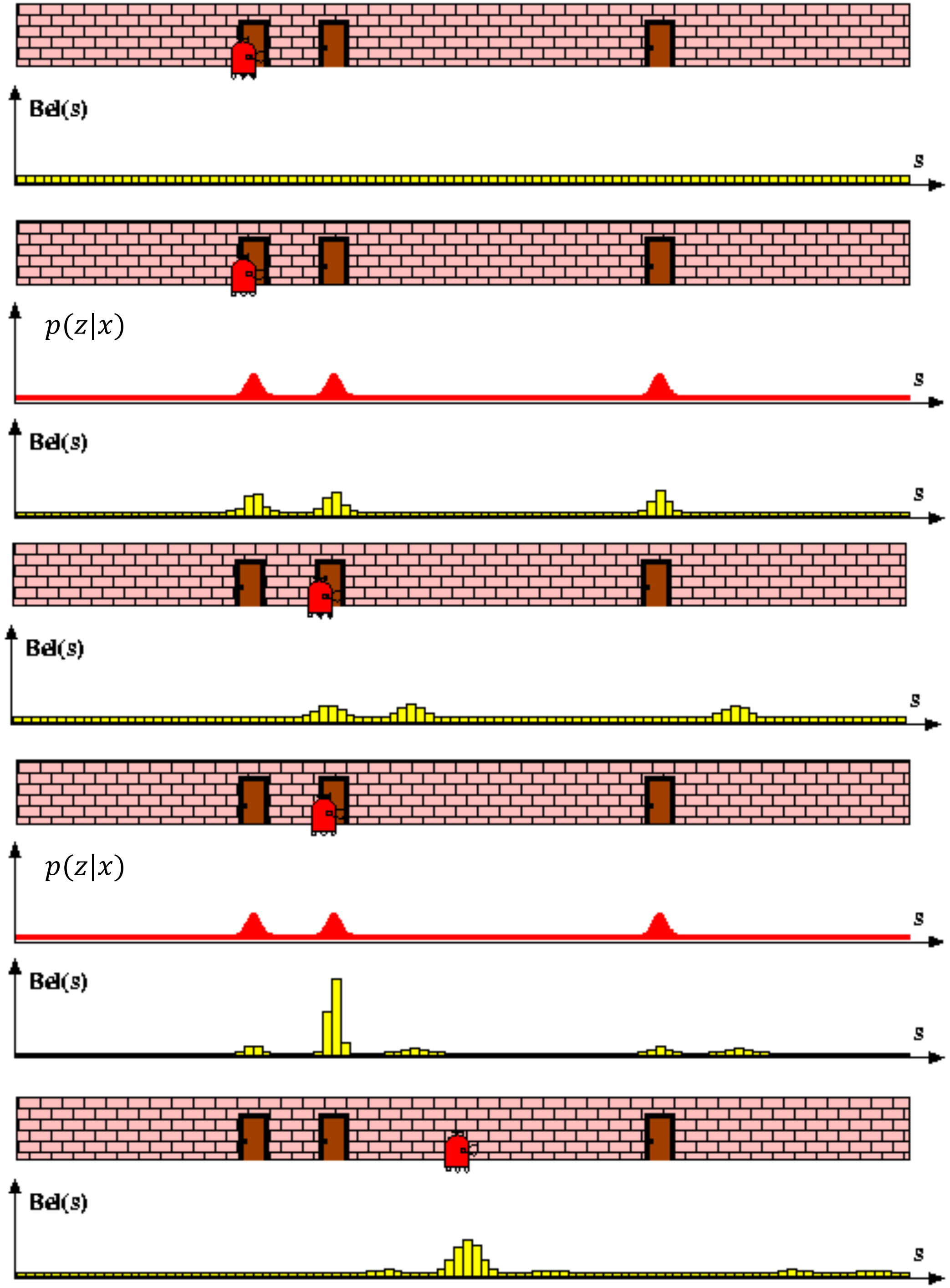


Laser sensor

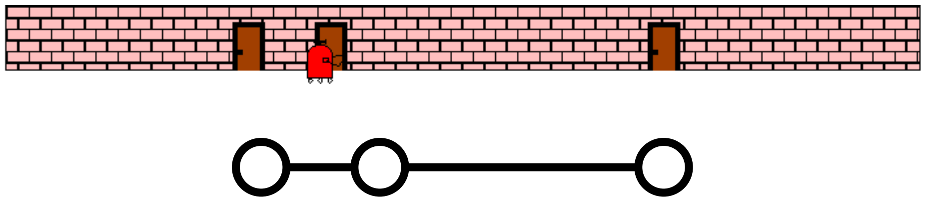


Sonar sensor

Grid localization, $bel(x_t)$represented by a histogram over grid

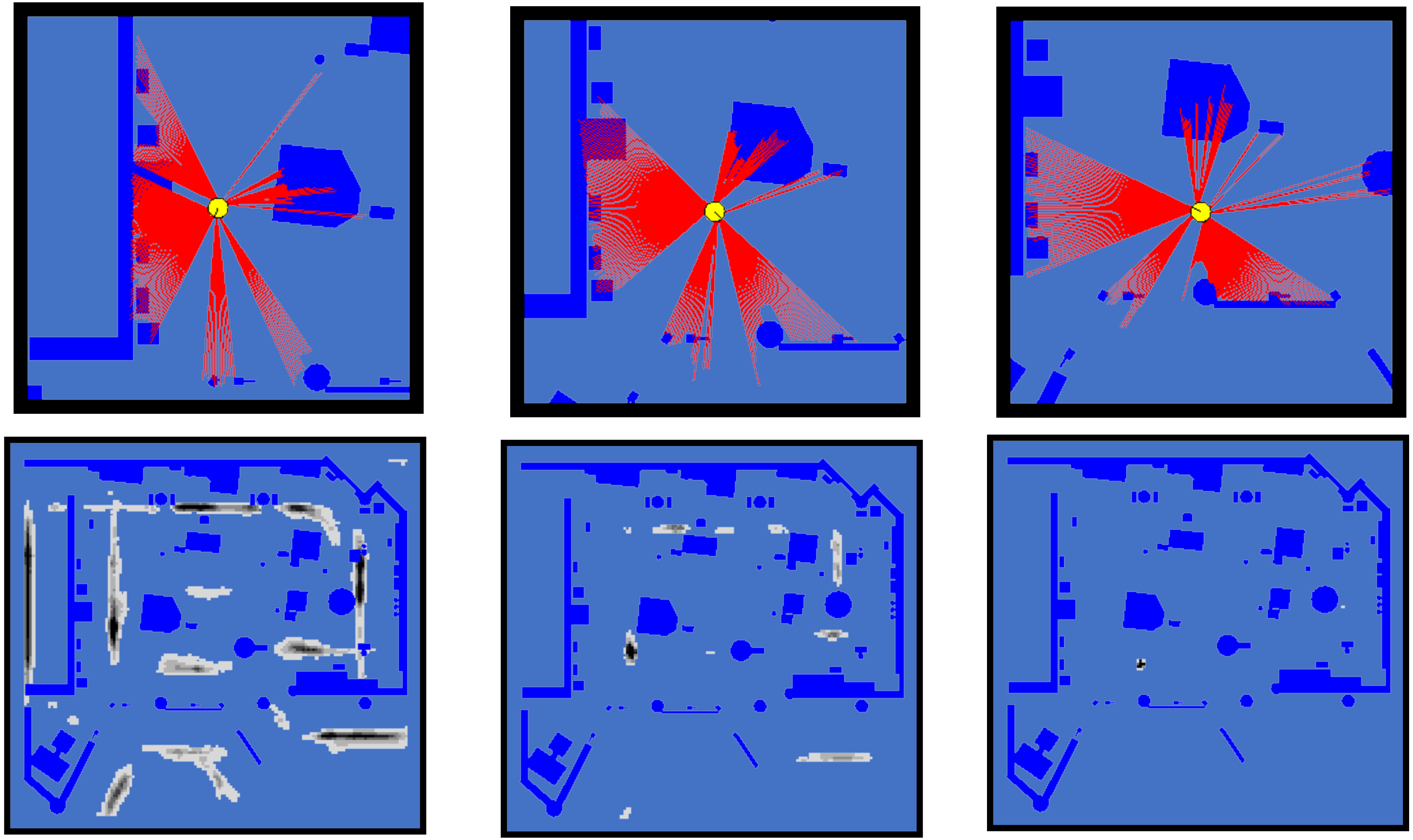


Summary

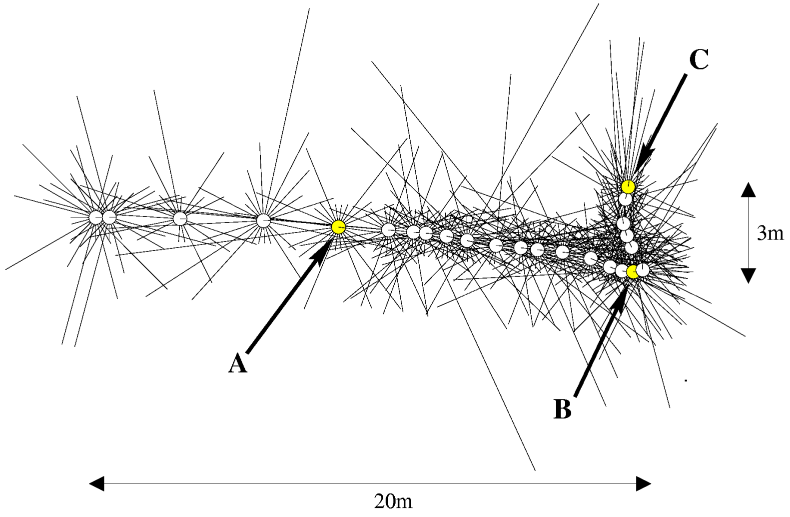


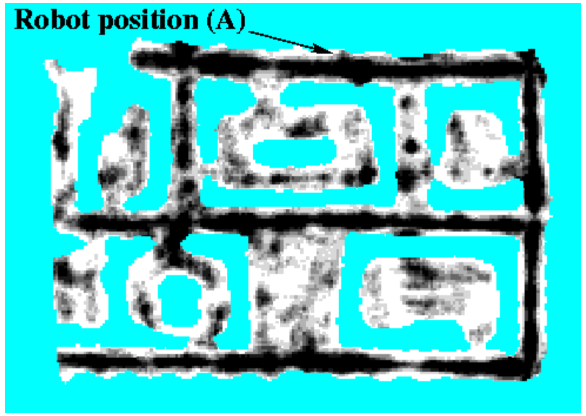
- Key variable: Grid resolution
- Two approaches
  - Topological: break-up pose space into regions of significance (landmarks)
  - Metric: fine-grained uniform partitioning; more accurate at the expense of higher computation costs
- Important to compensate for coarseness of resolution
  - Evaluating measurement/motion based on the center of the region may not be enough. *If motion is updated every 1s, robot moves at 10 cm/s, and the grid resolution is 1m, then naïve implementation will not have any state transition!*
- Computation
  - Motion model update for a 3D grid required a 6D operation, measurement update 3D
  - With fine-grained models, the algorithm cannot be run in real-time
  - Some calculations can be cached (ray-casting results)

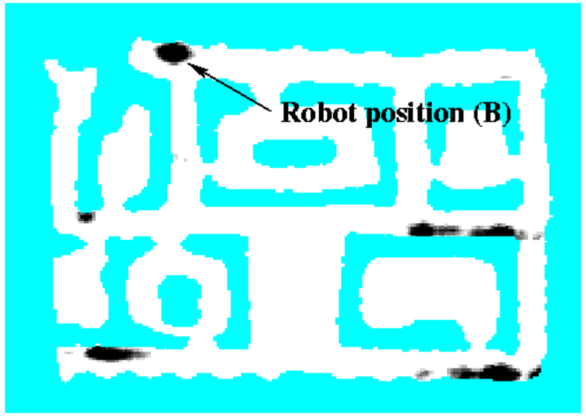
Grid-based Localization

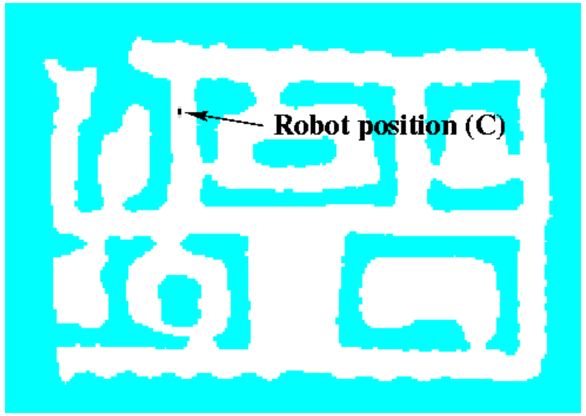


Sonars and Occupancy Grid Map









Monte Carlo Localization

- Represents beliefs by particles

Particle Filters

- Represent belief by finite number of parameters (just like histogram filter)
- But, they differ in how the parameters (particles) are generated and populate the state space
- Key idea: represent belief $bel(x_t)$ by a random set of state samples
- Advantages
  - The representation is approximate and nonparametric and therefore can represent a broader set of distributions than e.g., Gaussian
  - Can handle nonlinear tranformations
- Related ideas: Monte Carlo filter, Survival of the fittest, Condensation, Bootstrap filter, Filtering: [Rubin, 88], [Gordon et al., 93], [Kitagawa 96], Dynamic Bayesian Networks: [Kanazawa et al., 95]d

Particle filtering algorithm

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

Algorithm Particle_filter(X_{t-1}, u_t, z_t):
$\bar{X}_{t-1} = X_t = \emptyset$

for all m in [M] do:

sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$

$w_t^{[m]} = p\left(z_t \middle| x_t^{[m]}\right)$

$\bar{X}_t = \bar{X}_t + \langle\, x_t^{[m]}, w_t^{[m]} \rangle$

end for

for all m in [M] do:

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

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

end for

return X_t

ideally, $x_t^{[m]}$ is selected with probability prop. to $p(x_t \mid z_{1:t}, u_{1:t})$

$\bar{X}_{t-1}$ is the temporary particle set

// sampling from state transition dist.

// calculates *importance factor* w_t or weight

// resampling or importance sampling; these are distributed according to $\eta\, p\left(z_t \middle| x_t^{[m]}\right) \overline{bel}(x_t)$

// survival of fittest: moves/adds particles to parts of the state space with higher probability

Importance Sampling

suppose we want to compute $E_f[I(x \in A)]$ but we can only sample from density g

$E_f[I(x \in A)]$

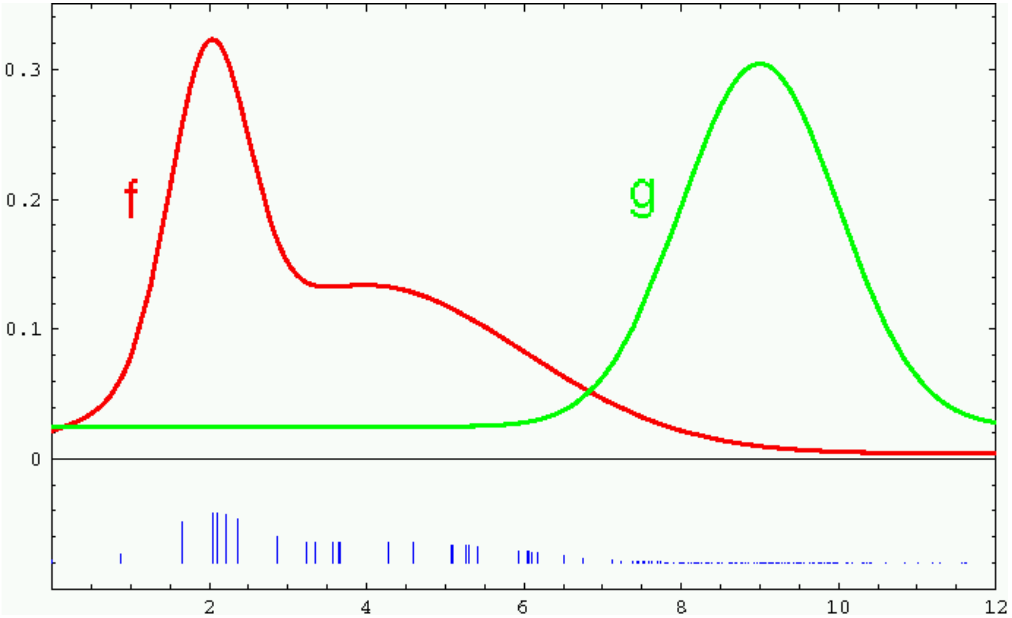
$= \int f(x) I(x \in A) dx$

$= \int \frac{f(x)}{g(x)} g(x) I(x \in A) dx$, provided $g(x) > 0$

$= \int w(x) g(x) I(x \in A) dx$

$= E_g[w(x) I(x \in A)]$

We need $f(x) > 0 \Rightarrow g(x) > 0$



Weight samples: $w = f/g$

Monte Carlo Localization (MCL)

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

Algorithm MCL(X_{t-1}, u_t, z_t,m):

$\bar{X}_{t-1} = X_t = \emptyset$

for all m in [M] do:

$$x_t^{[m]} = \boldsymbol{sample\_motion\_model}(u_t\ x_{t-1}^{[m]})$$

$$w_t^{[m]} = \boldsymbol{measurement\_model}(z_t, x_t^{[m],m})$$

$$\bar{X}_t = \bar{X}_t + \langle\, x_t^{[m]}, w_t^{[m]} \rangle$$

end for

for all m in [M] do:

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

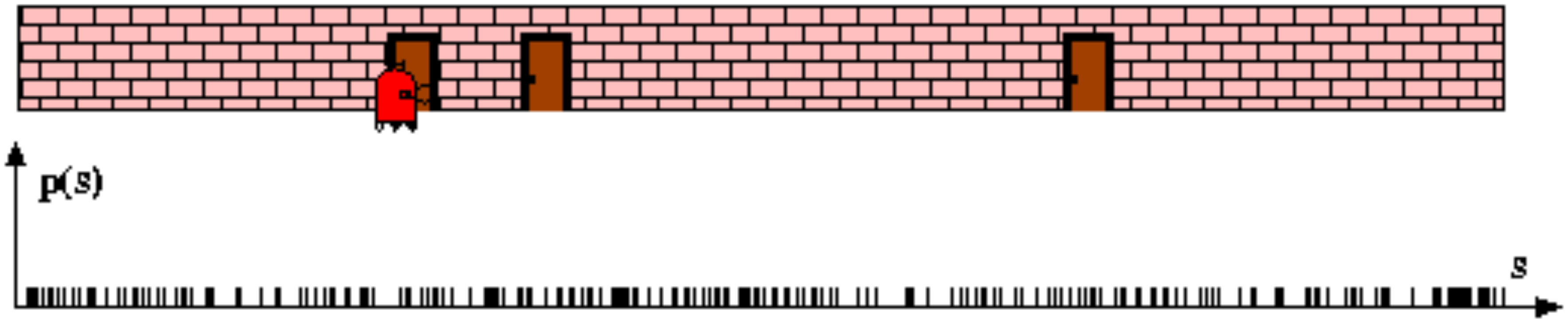
add $x_t^{[i]}$ $to\ X_t$

end for

return X_t

Plug in motion and measurement models in the particle filter

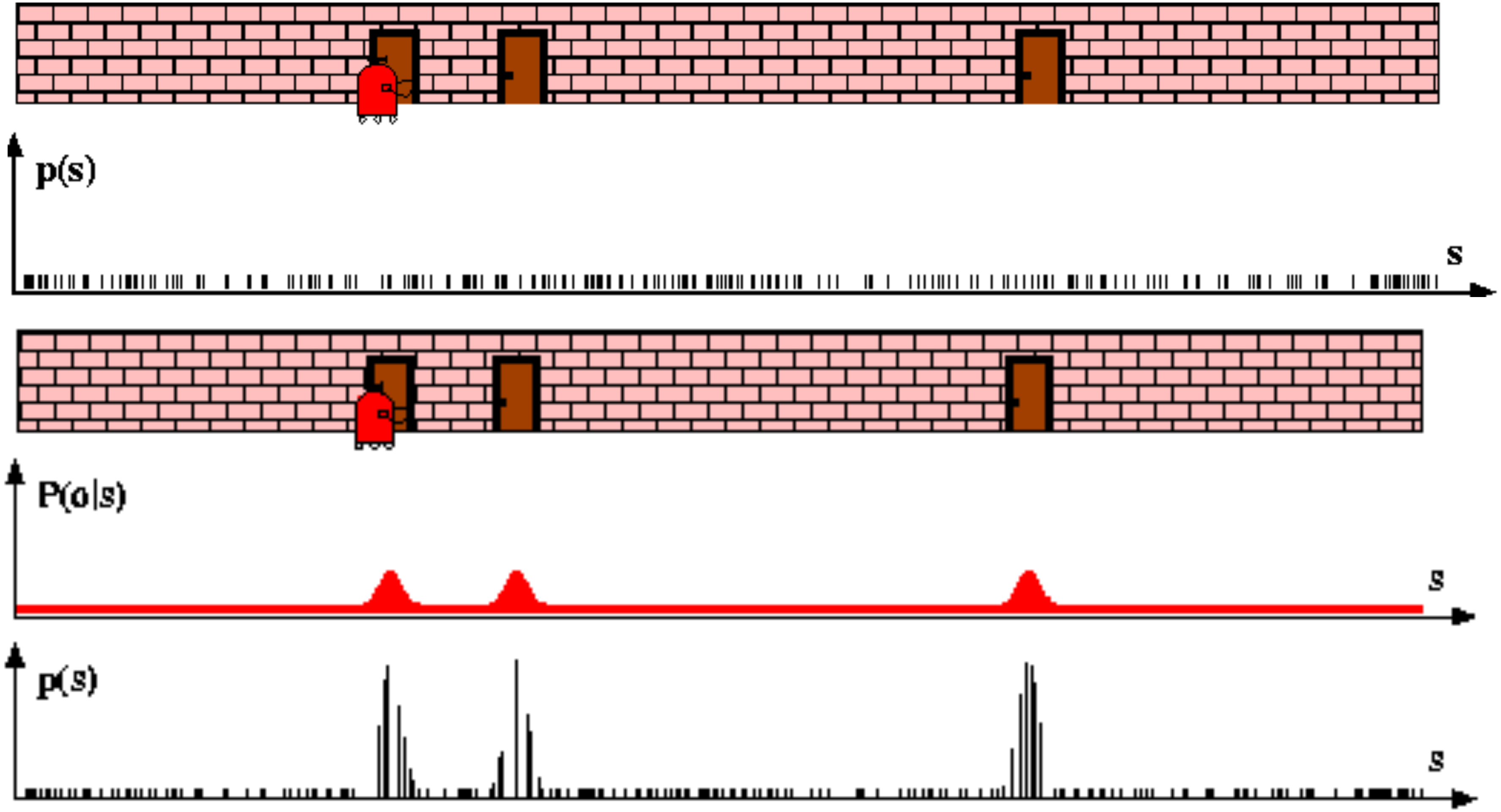
Particle Filters



Sensor Information: Importance Sampling

$$Bel(x) \leftarrow \alpha\, p(z \mid x)\, Bel^{-}(x)$$

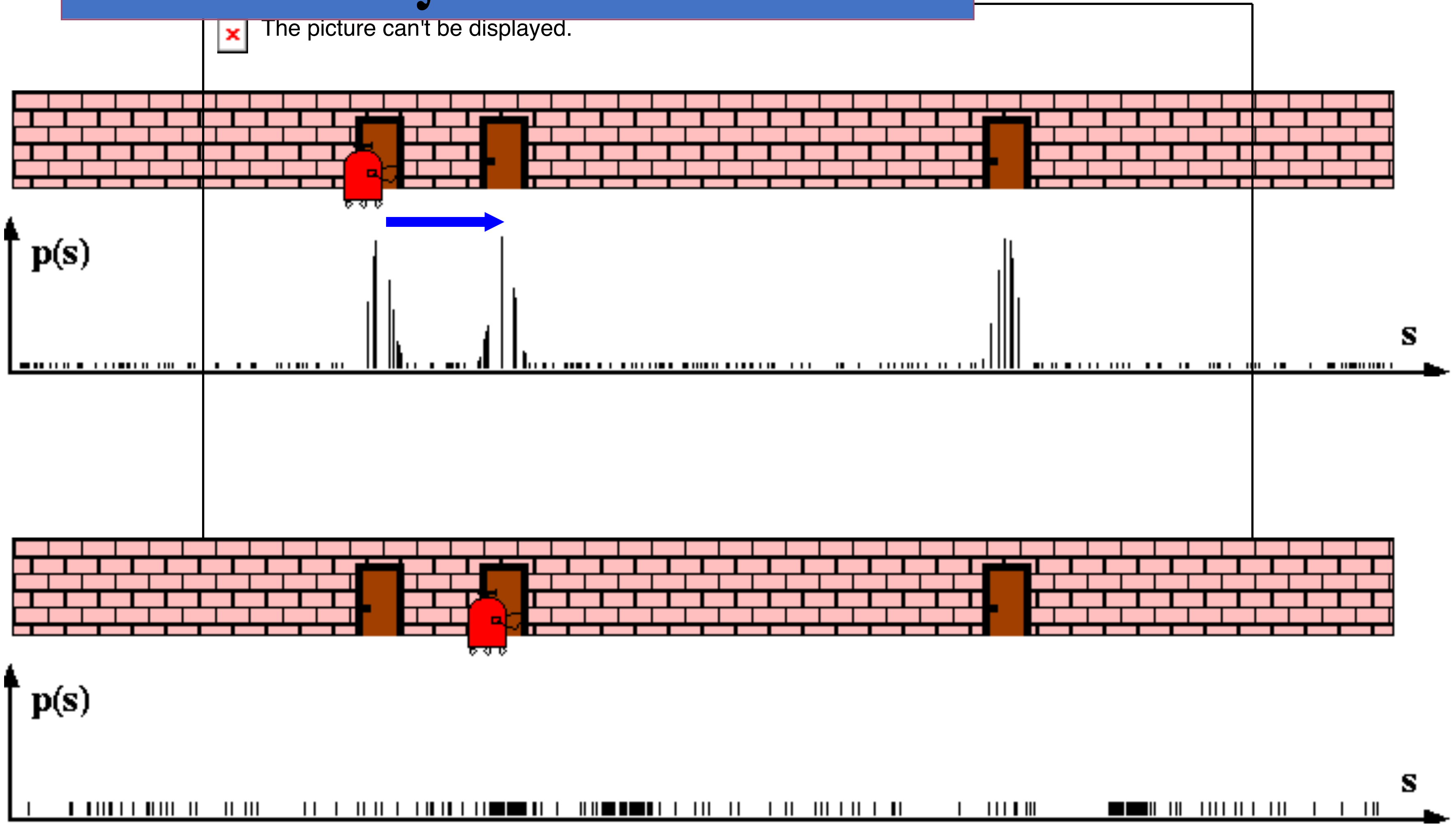
$$w \leftarrow \frac{\alpha\, p(z \mid x)\, Bel^{-}(x)}{Bel^{-}(x)} = \alpha\, p(z \mid x)$$



Robot Motion

$$Bel^-(x) \leftarrow \int p(x \mid u, x')\, Bel(x')\; \mathrm{d}x'$$

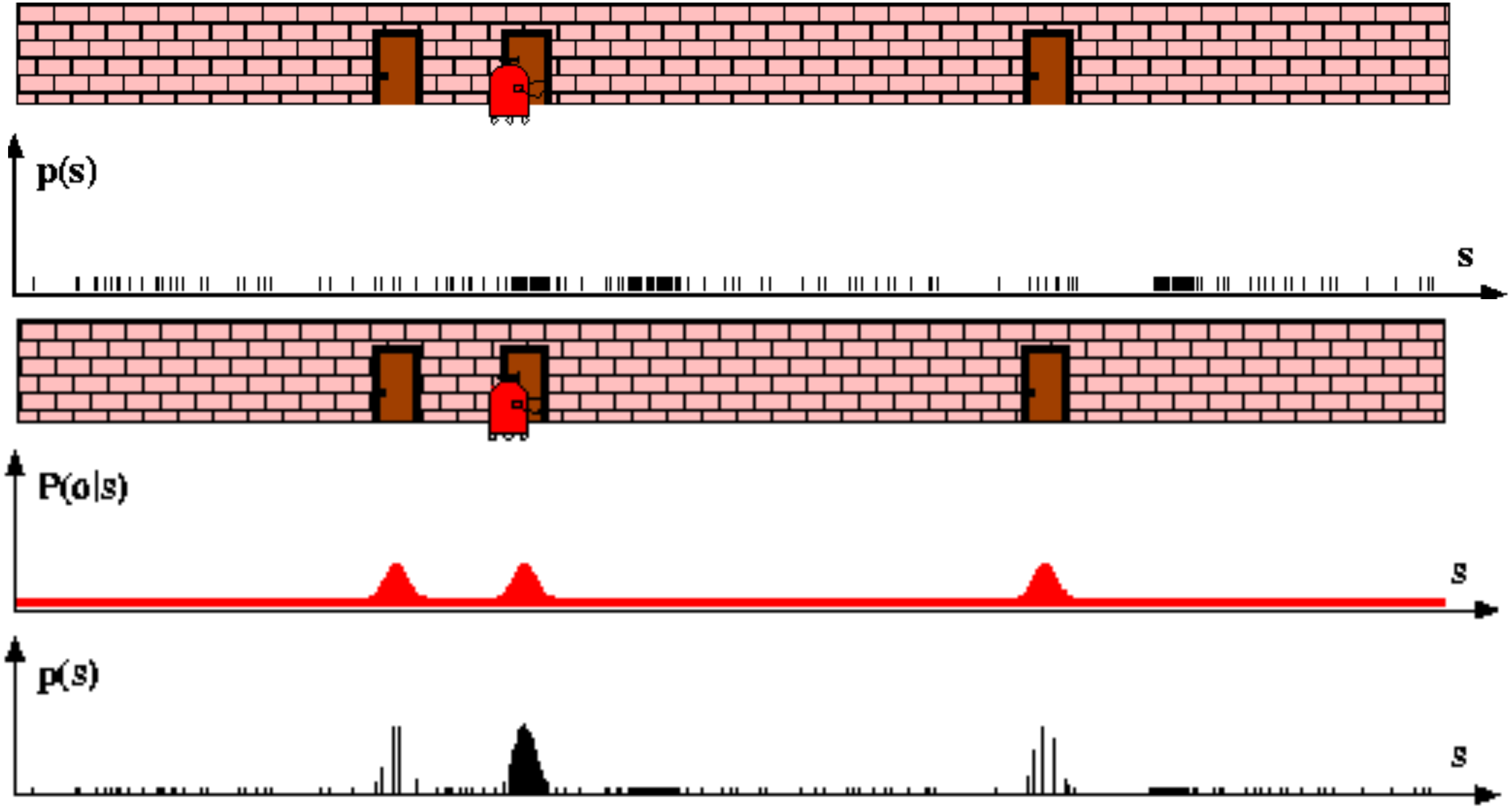
The picture can't be displayed.



Sensor Information: Importance Sampling

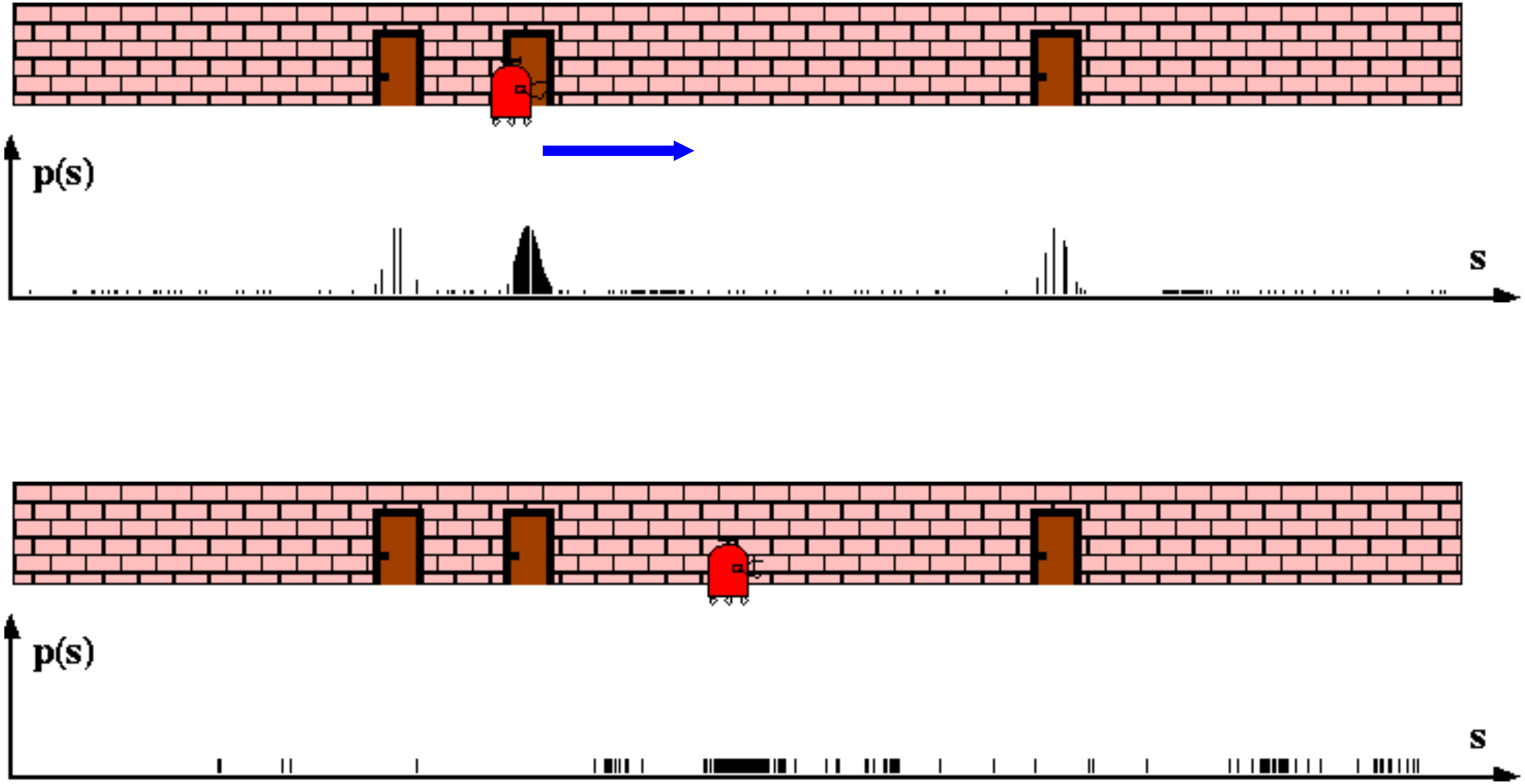
$$Bel(x) \leftarrow \alpha\ p(z\,|\,x)\ Bel^{-}(x)$$

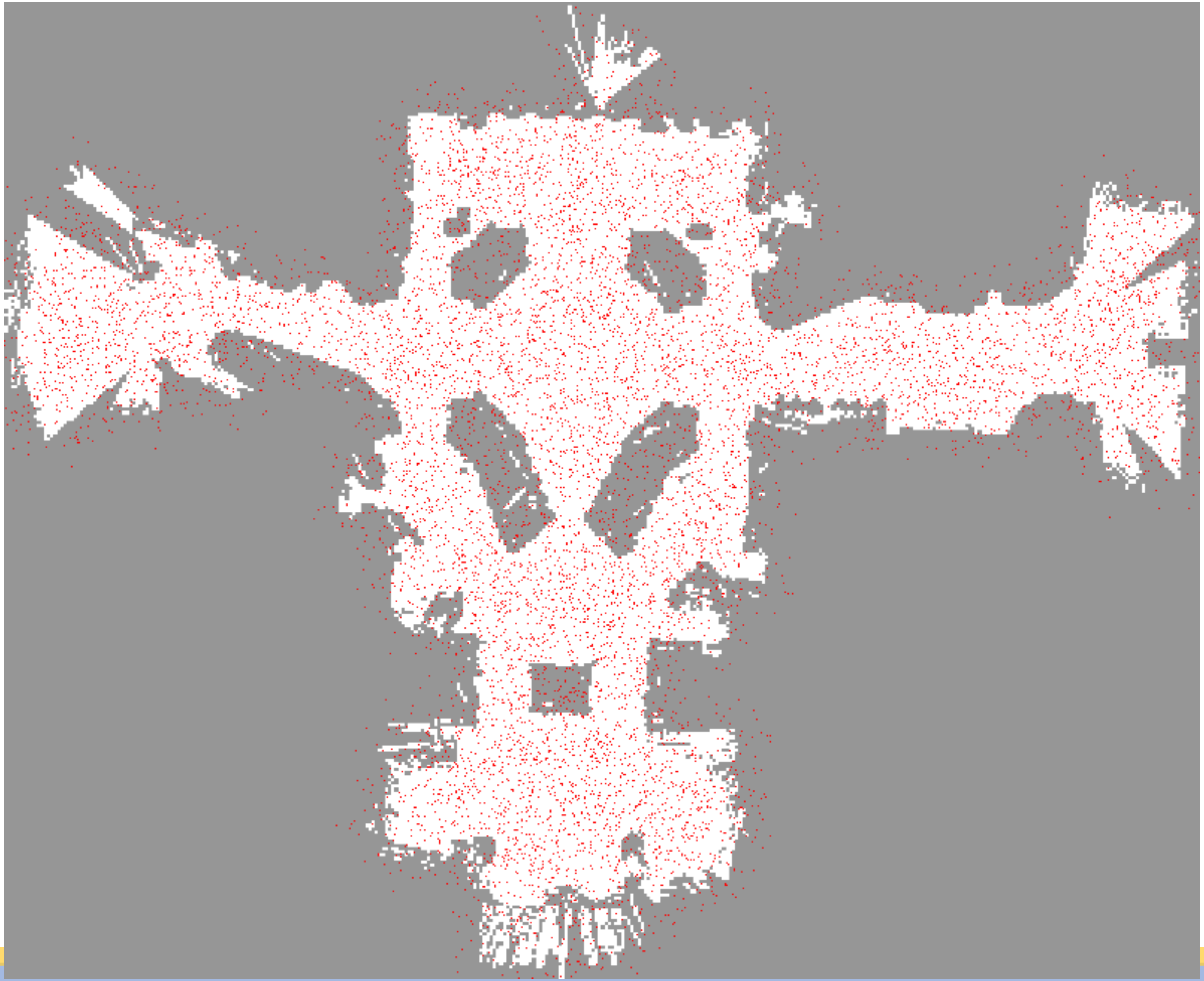
$$w \leftarrow \frac{\alpha\ p(z\,|\,x)\ Bel^{-}(x)}{Bel^{-}(x)} = \alpha\ p(z\,|\,x)$$

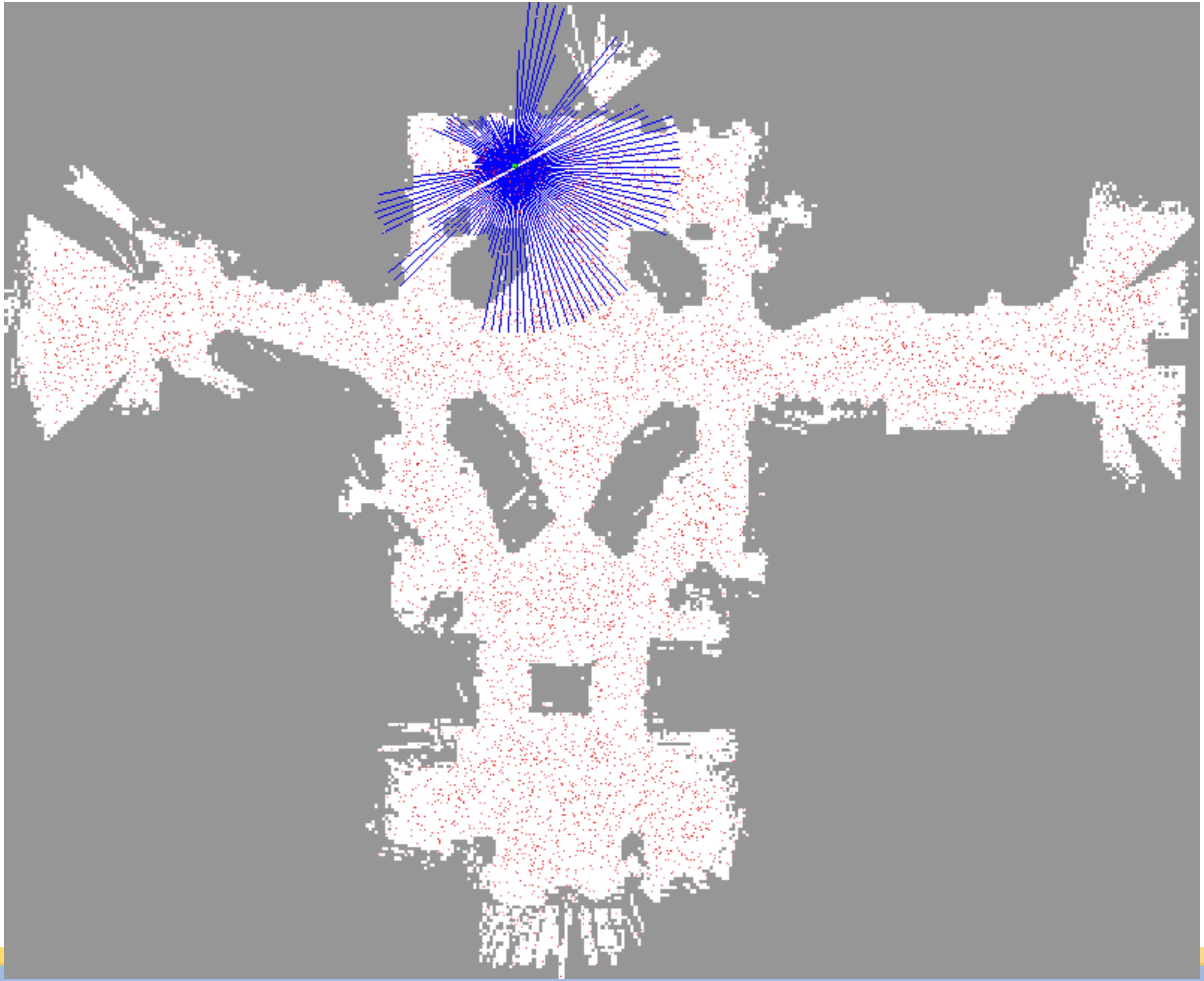


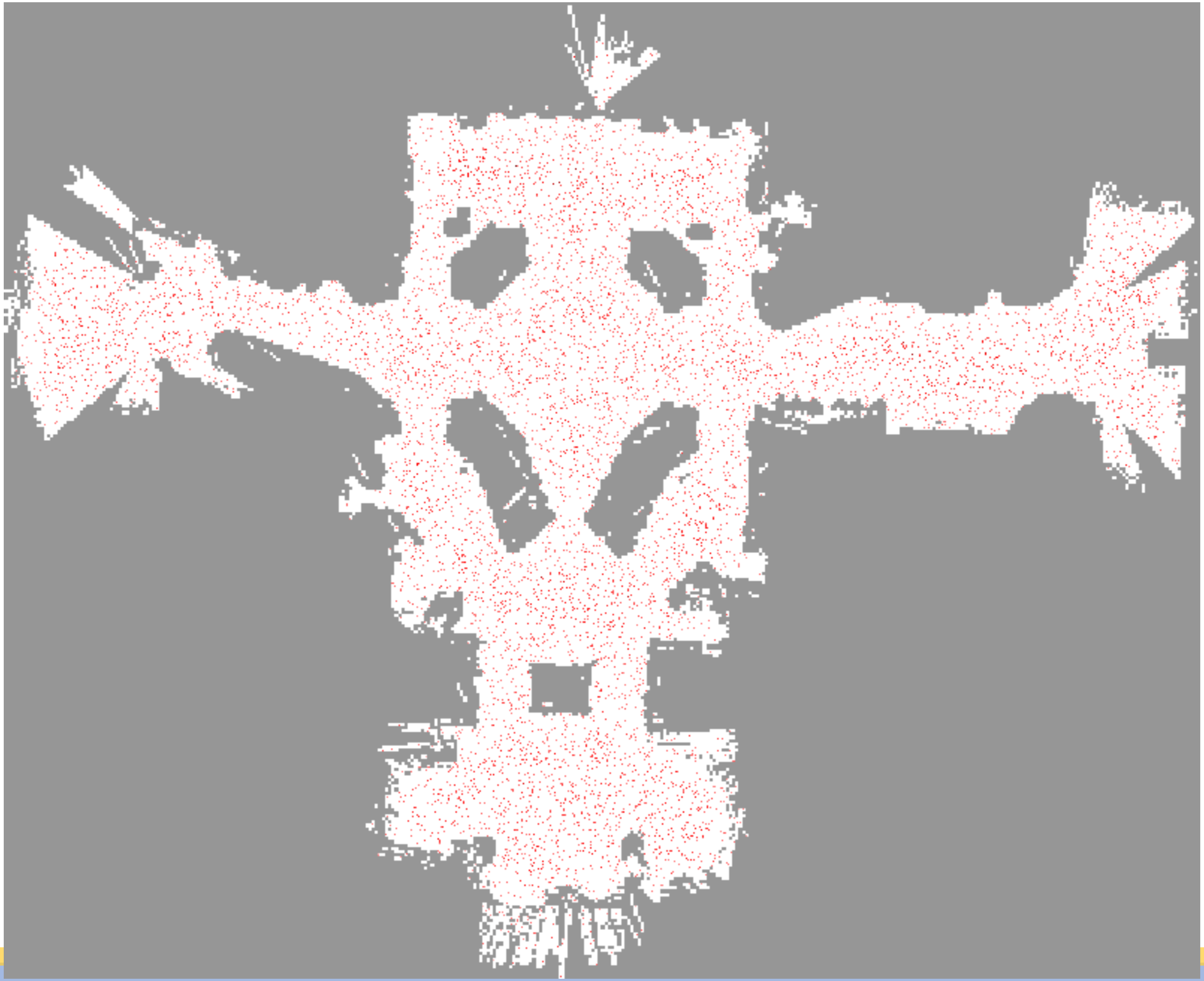
Robot Motion

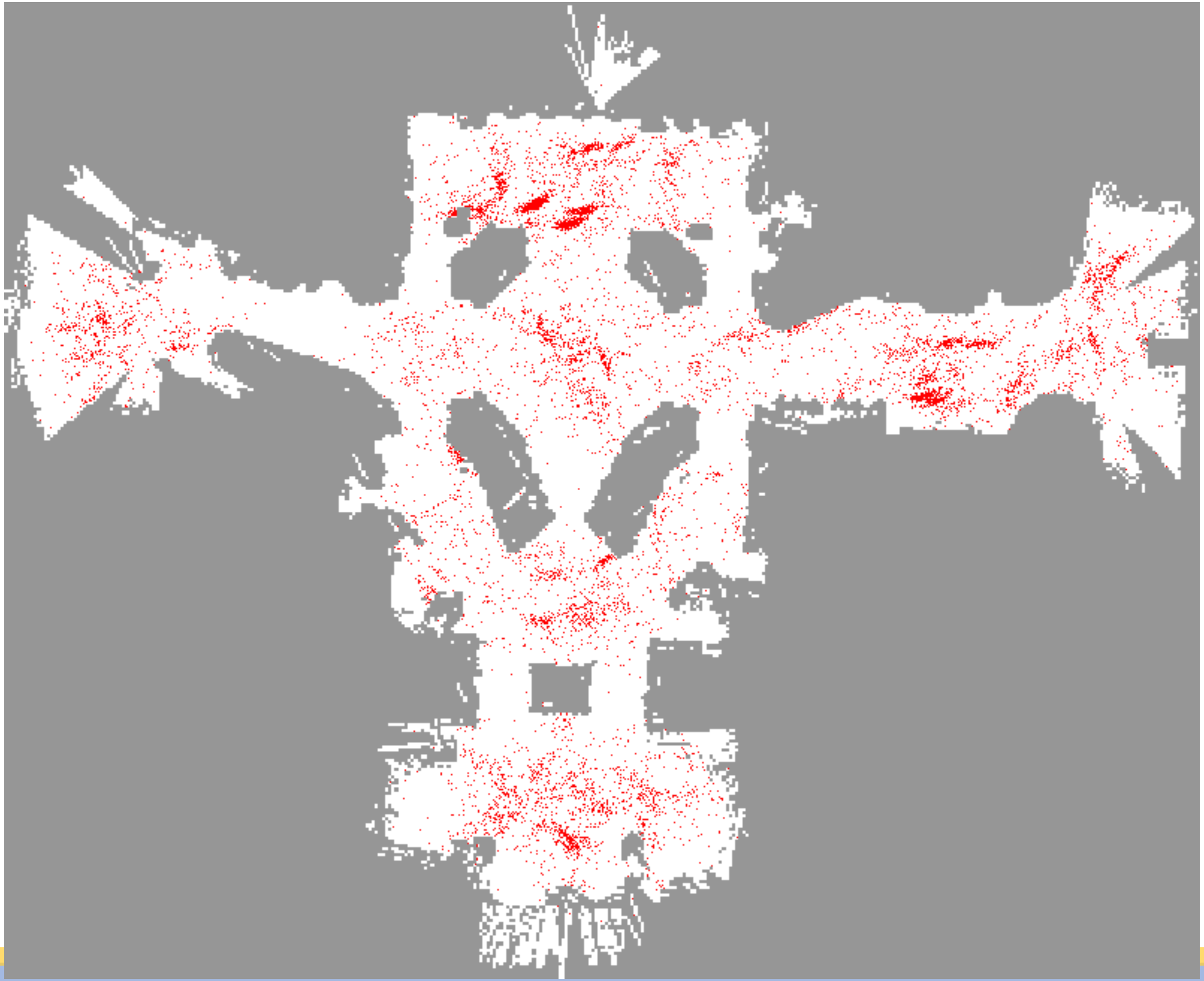
$$Bel^{-}(x) \quad \leftarrow \quad \int p(x \mid u, x')\, Bel(x')\ \mathrm{d}x'$$

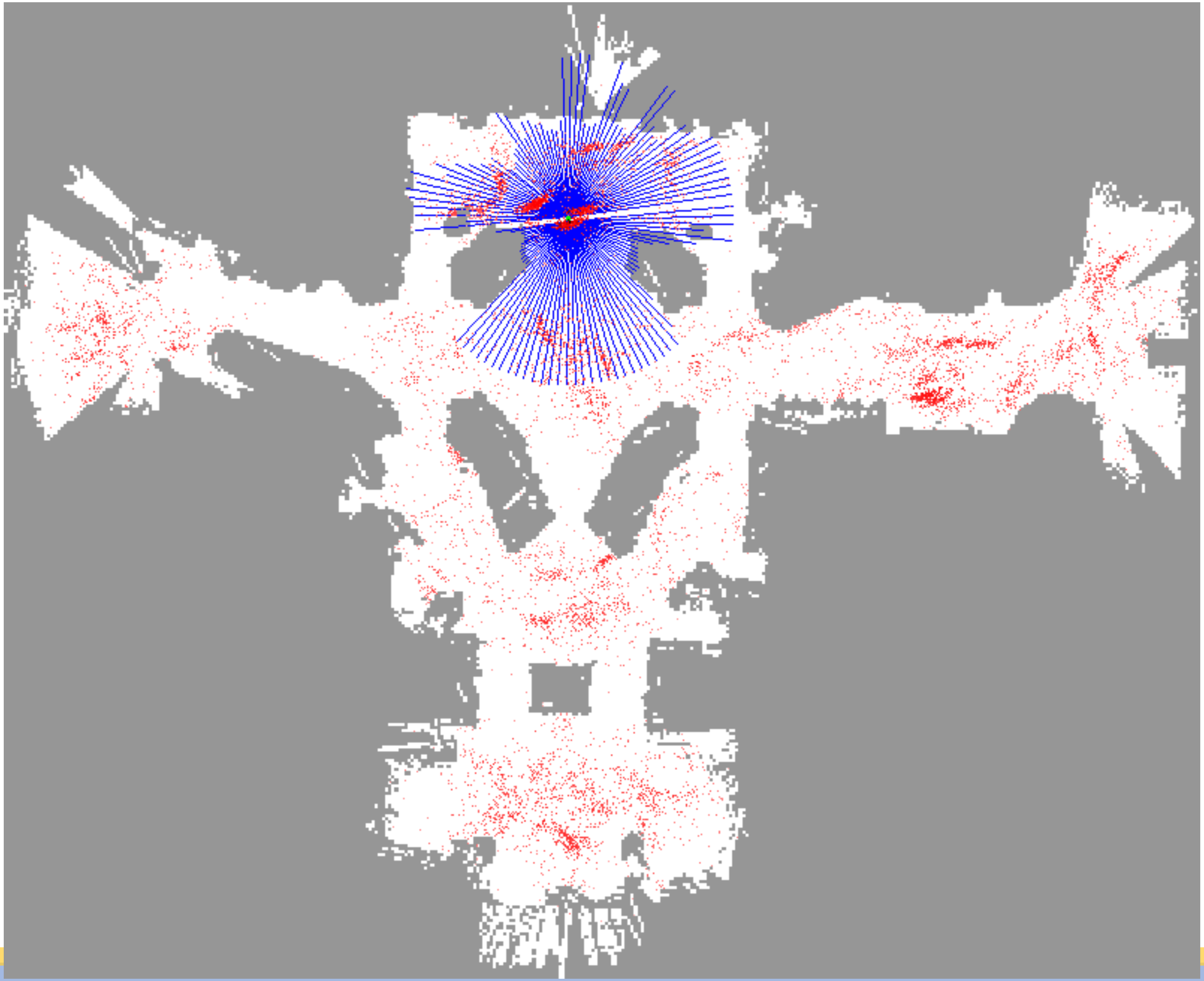


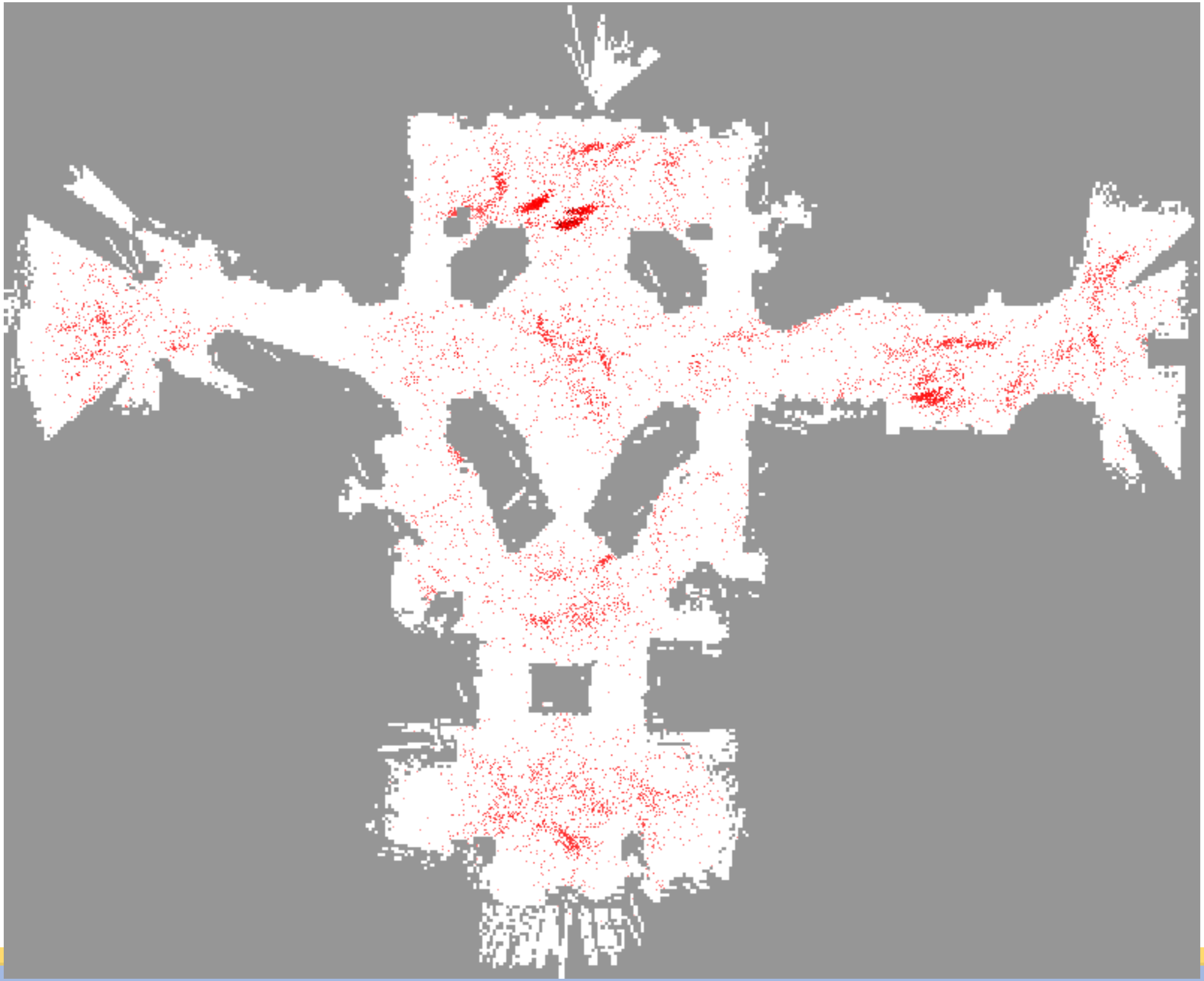


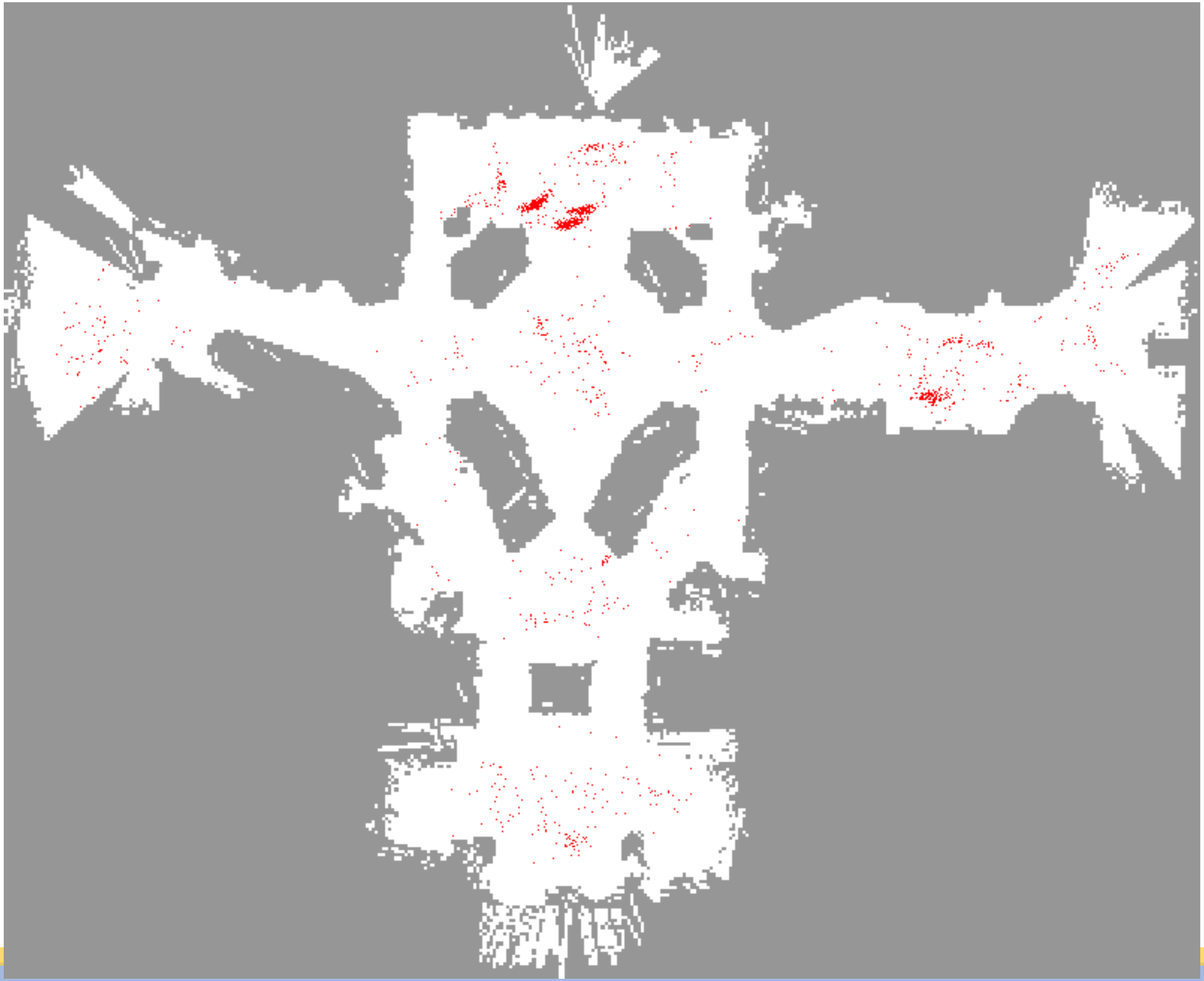


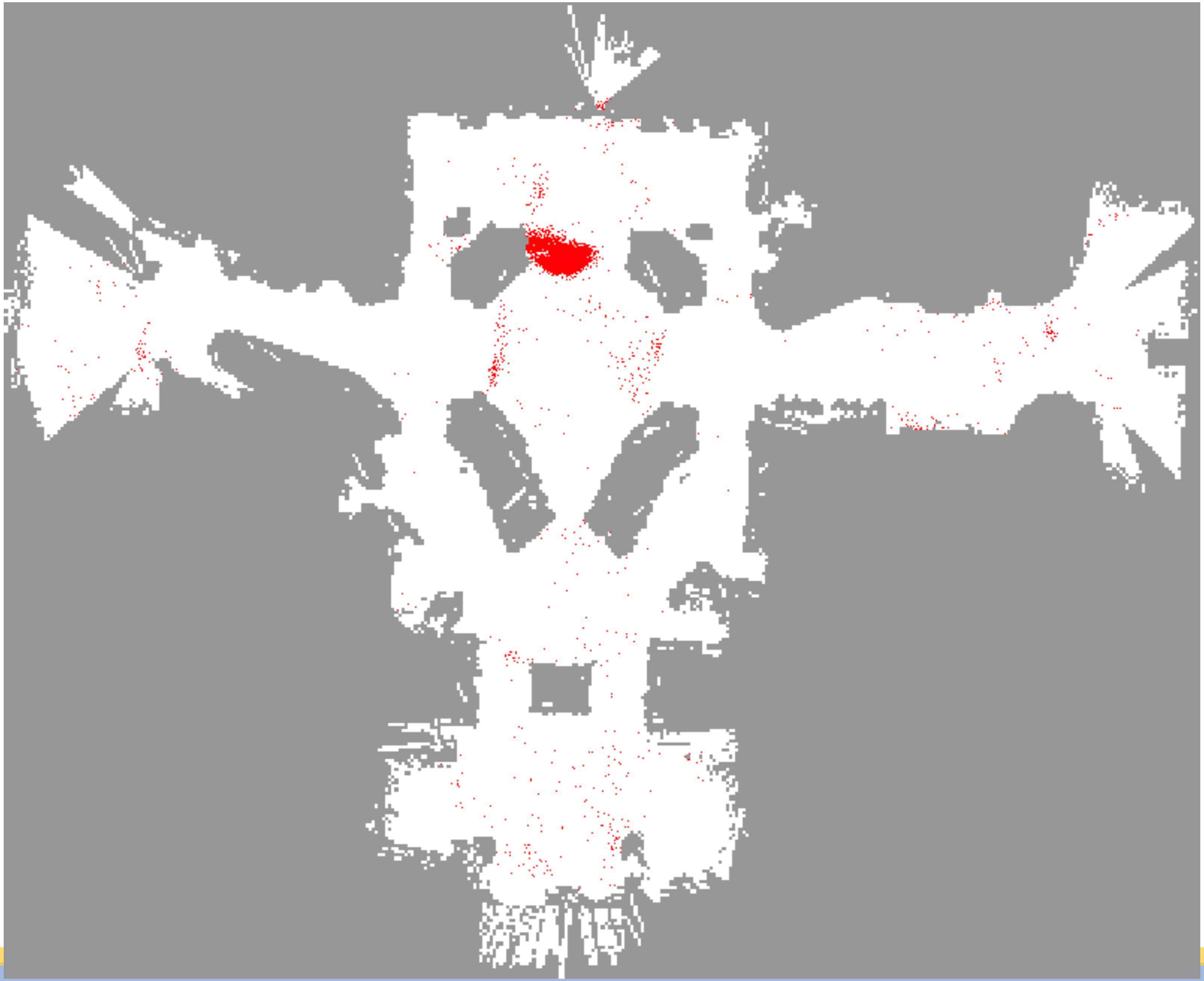


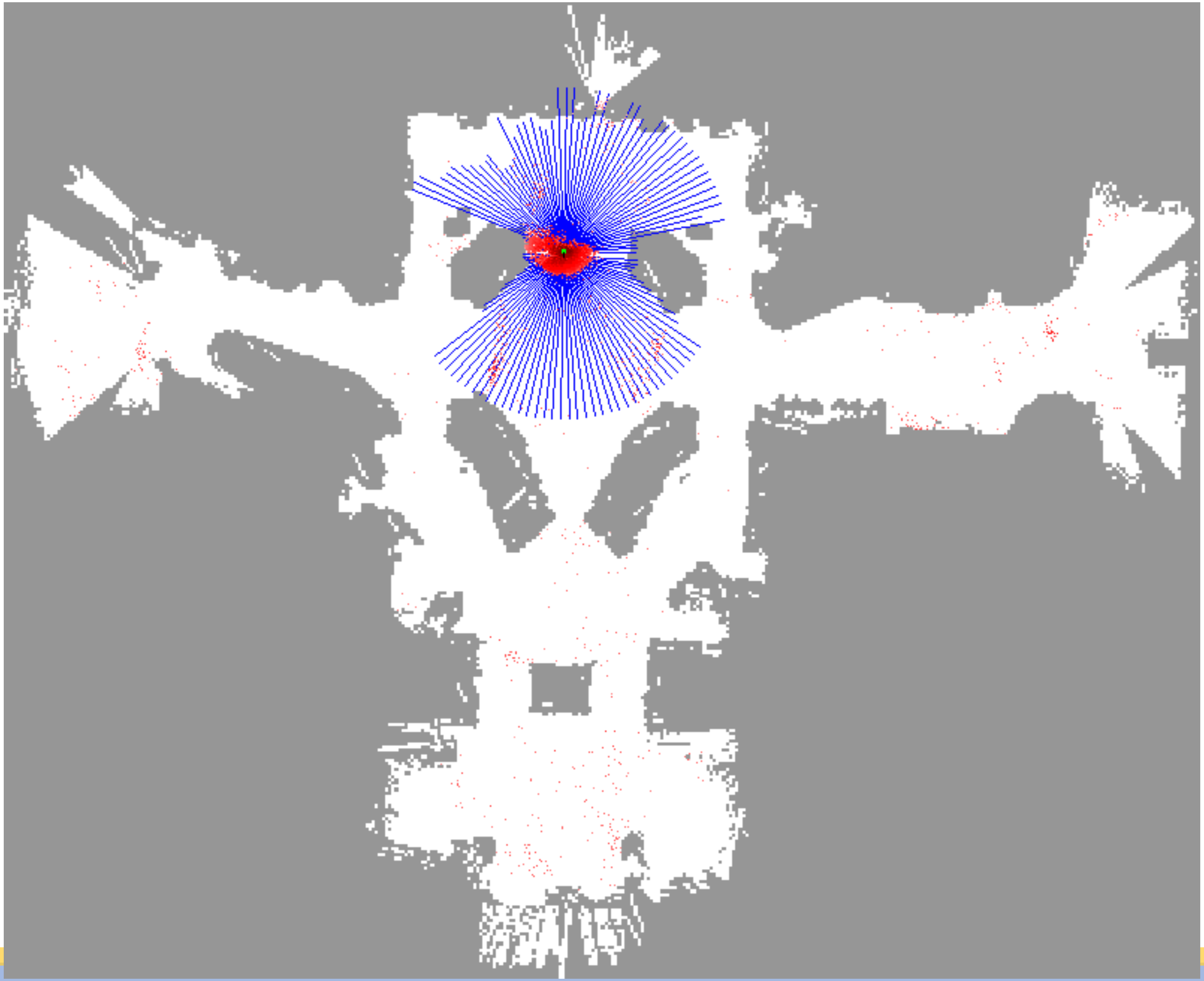


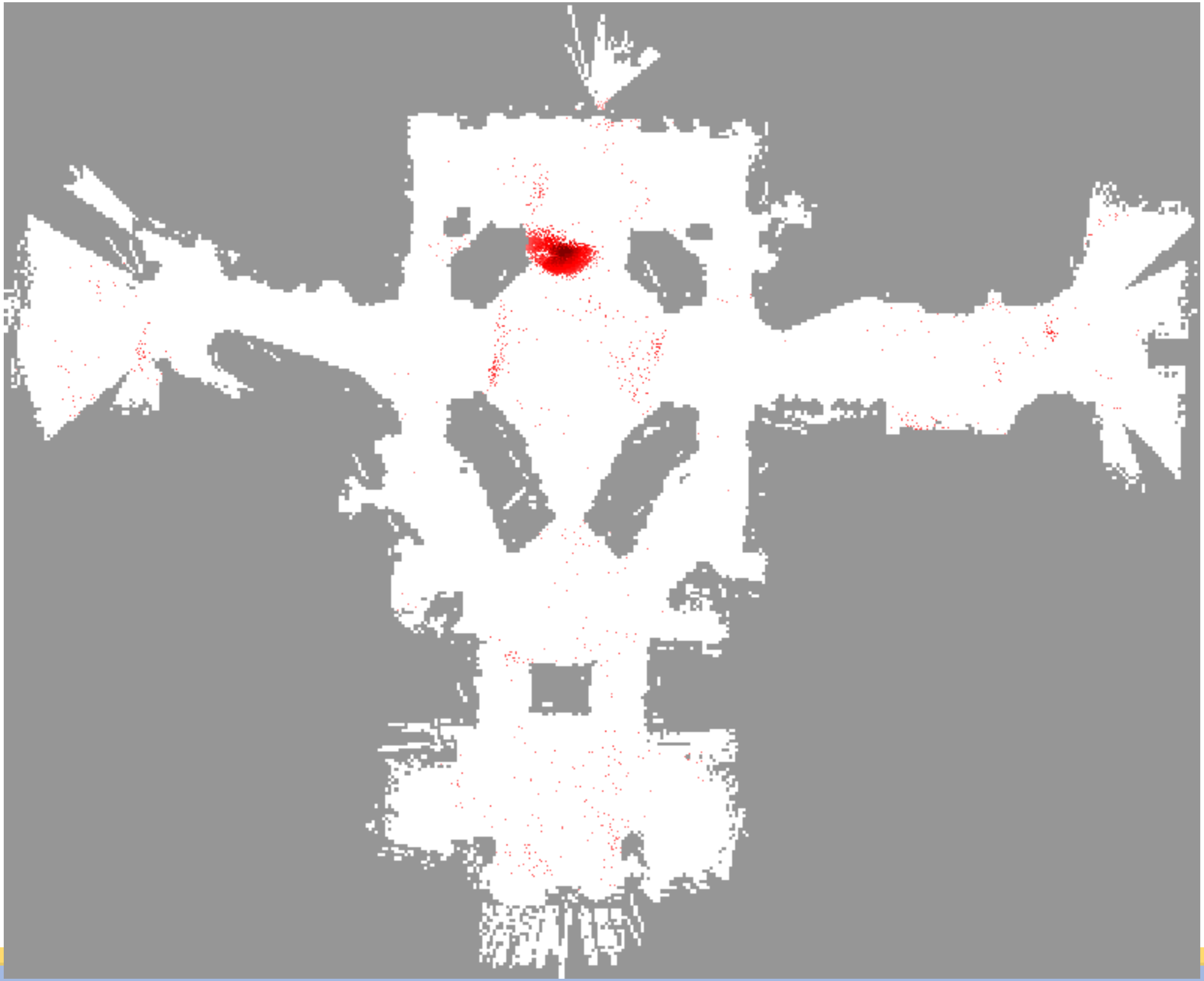


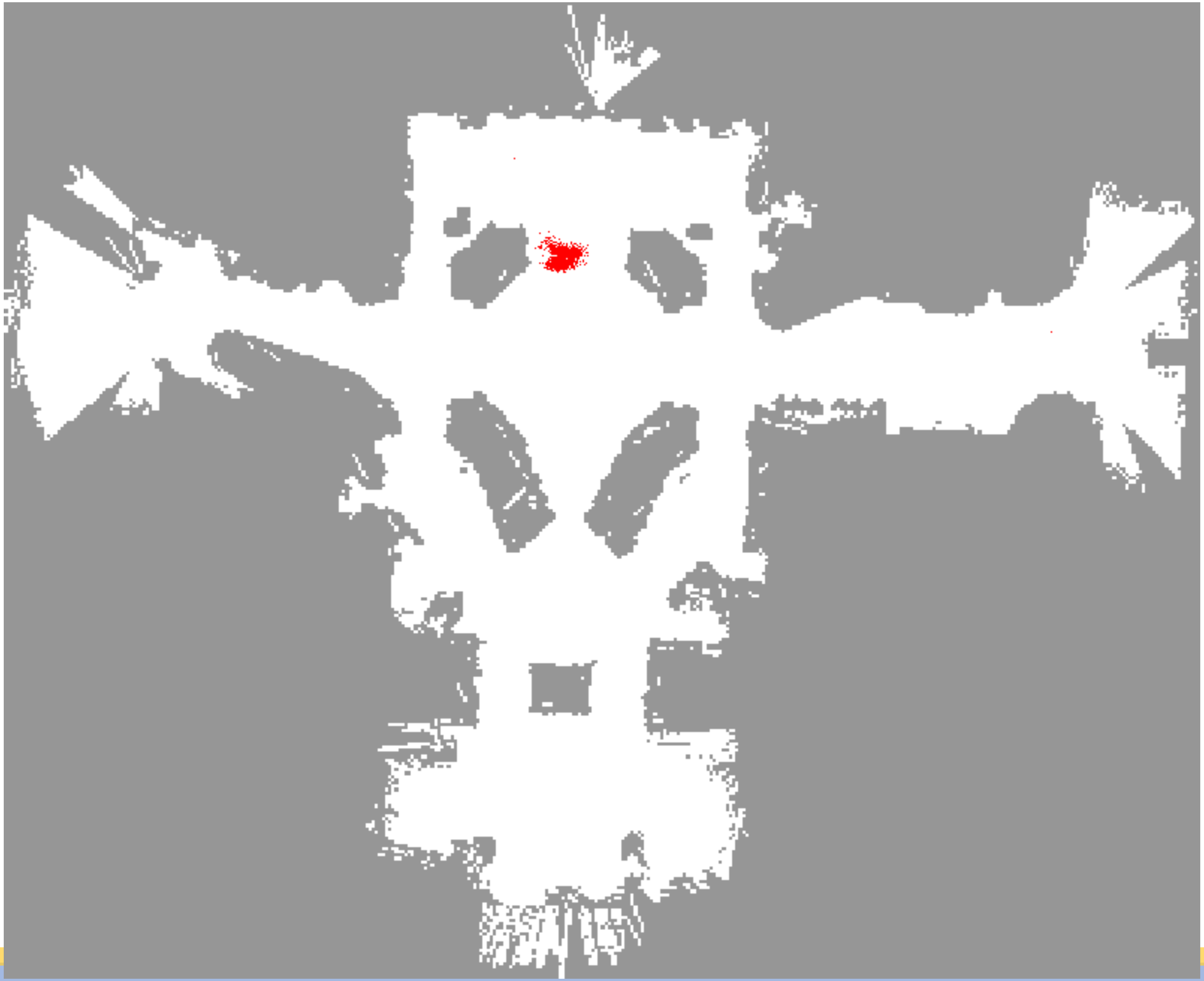


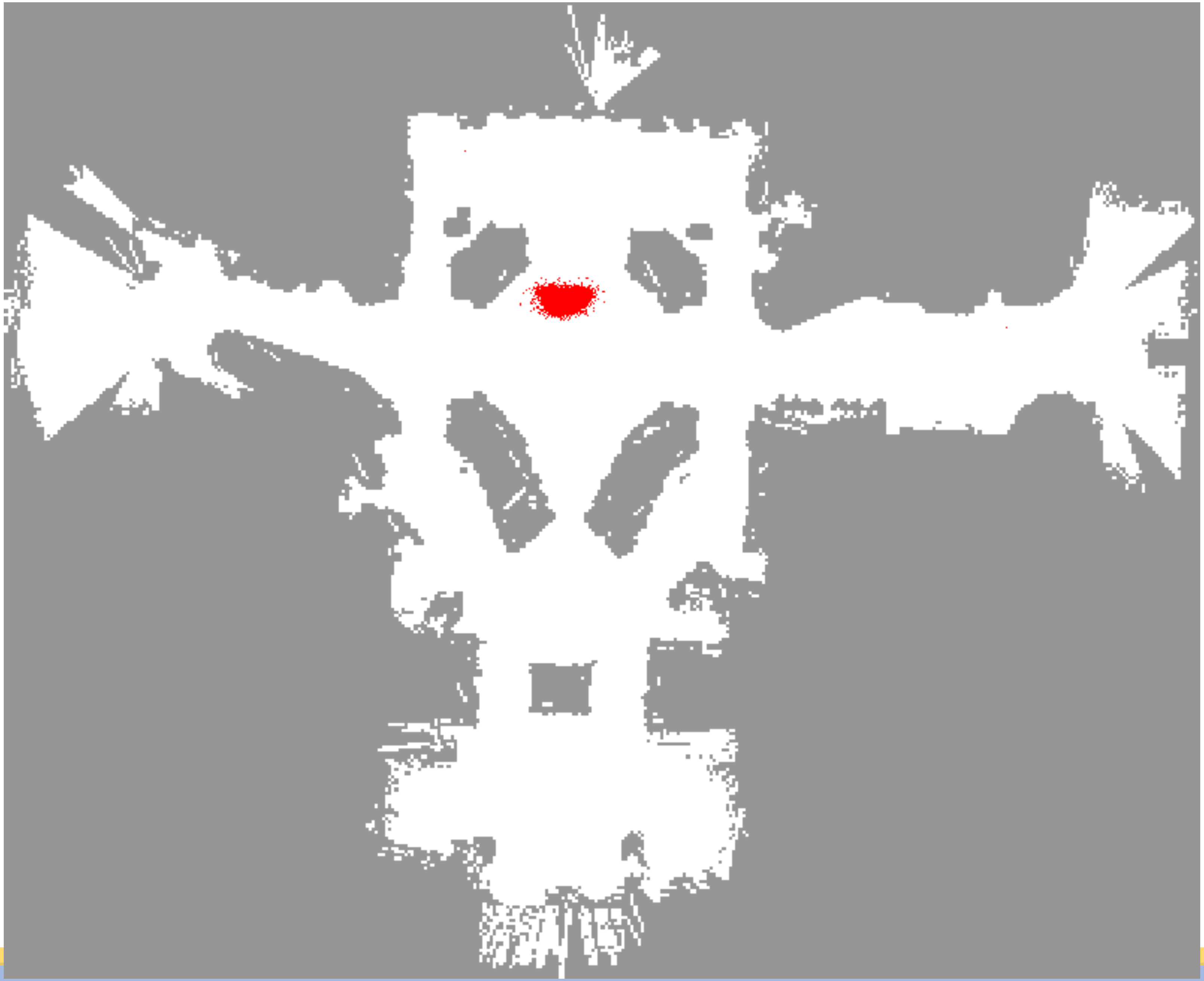


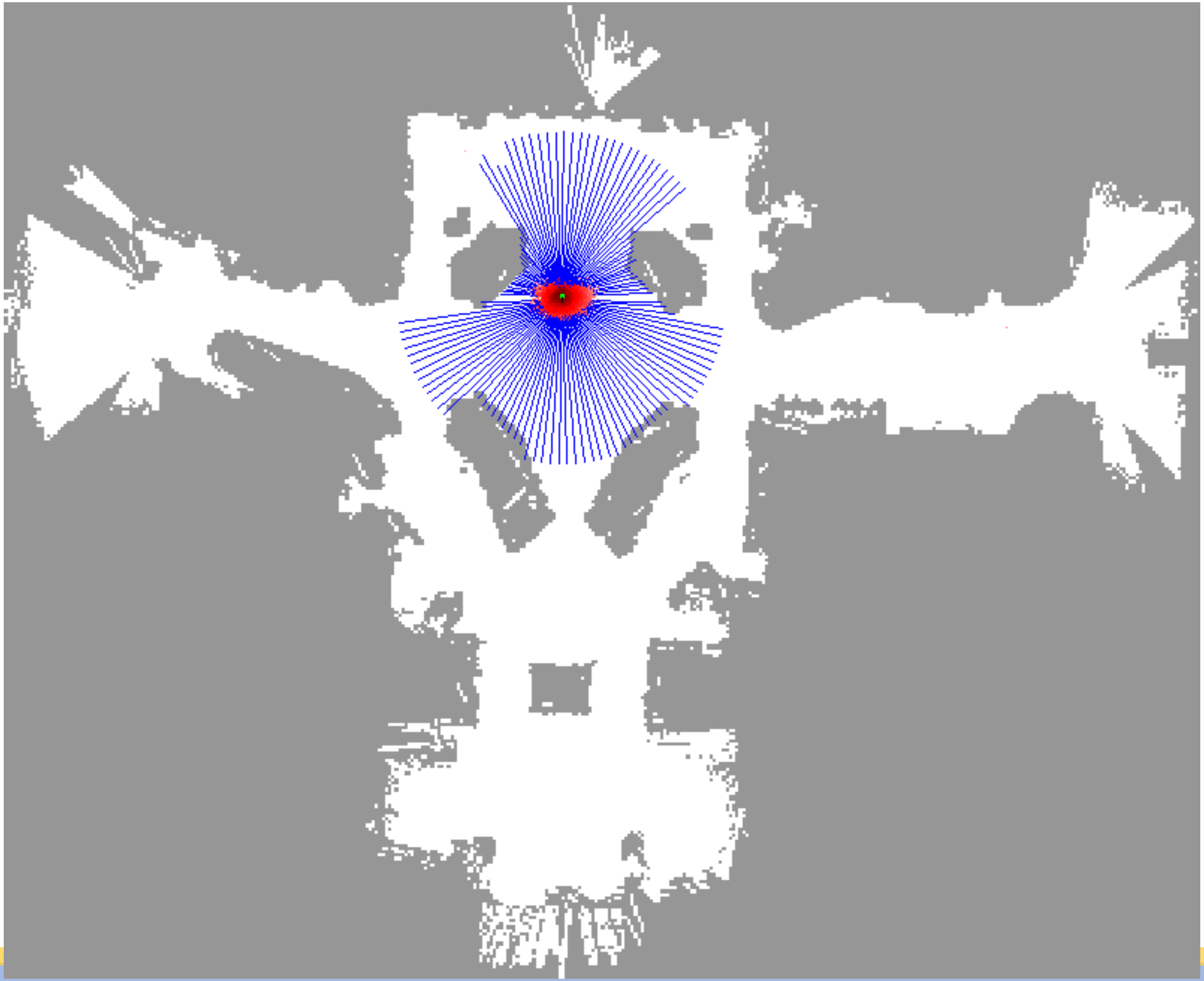


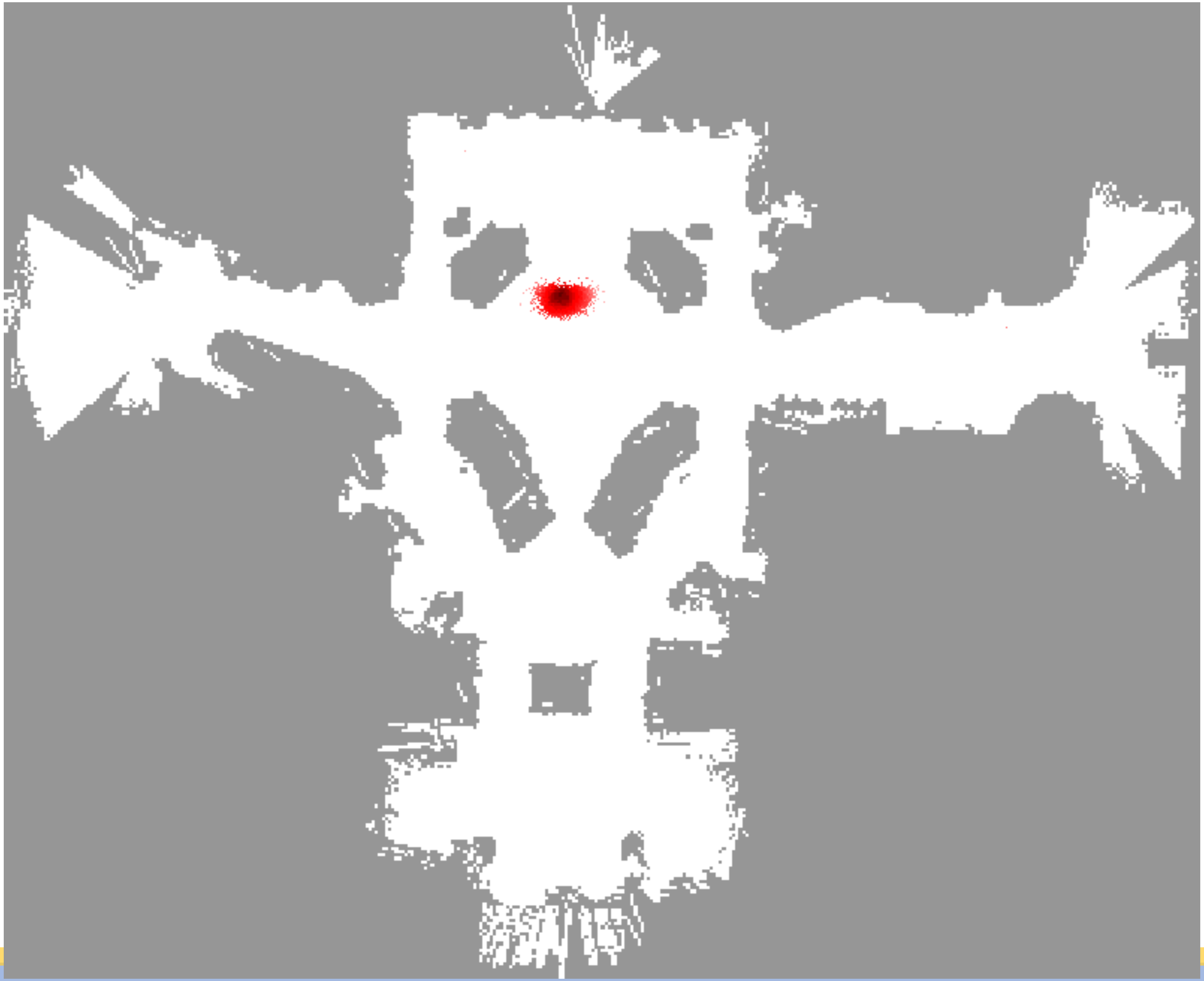


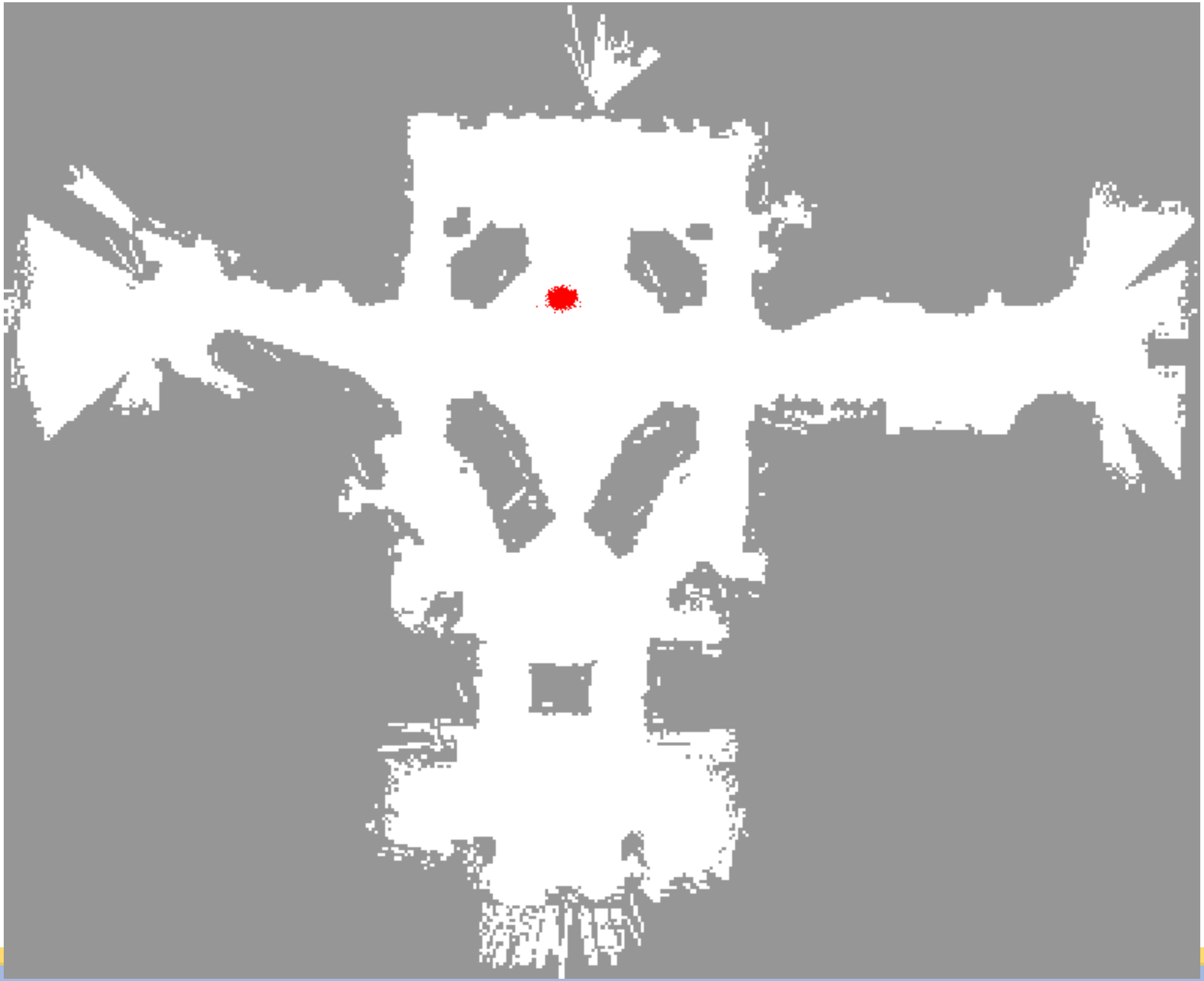


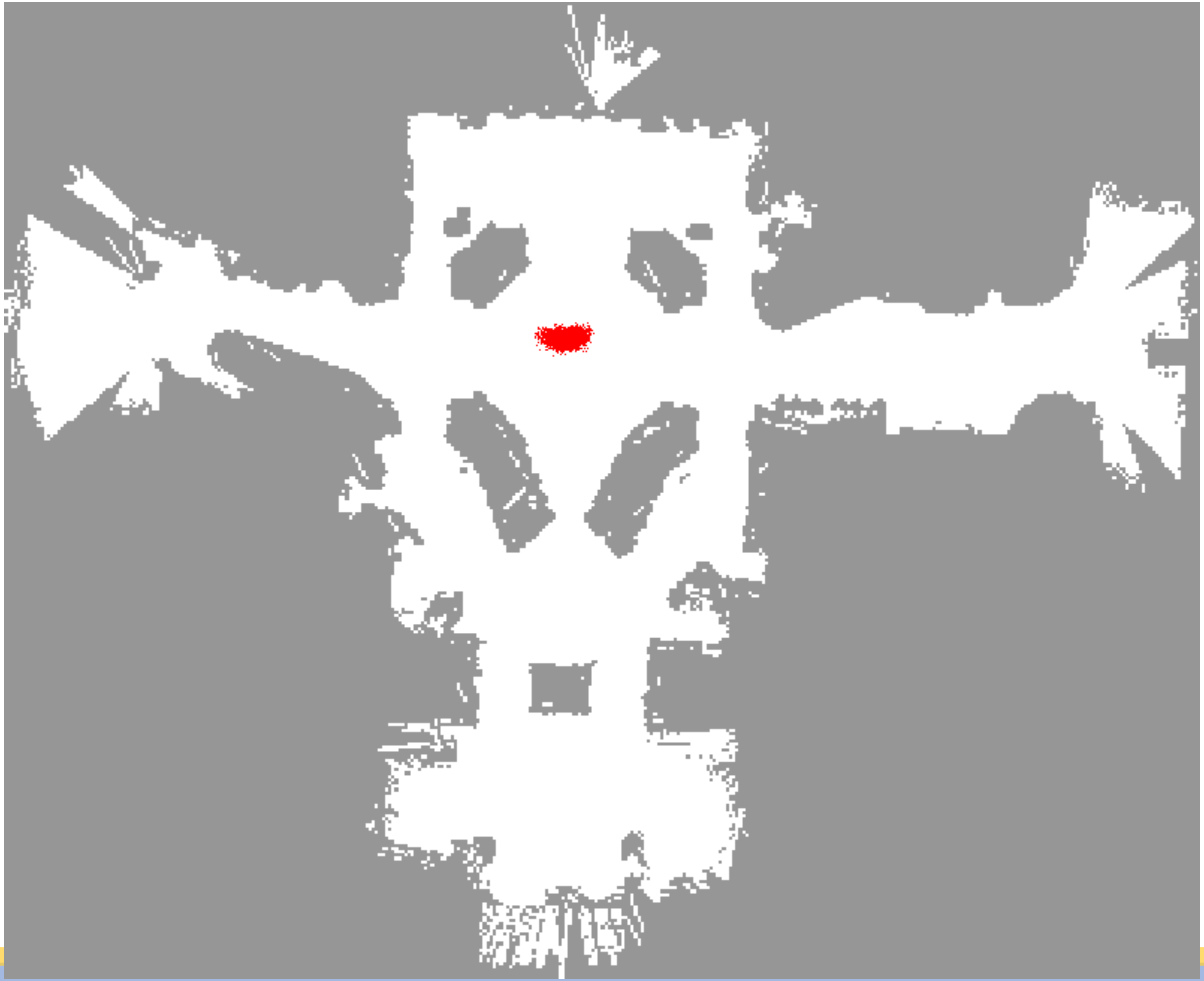


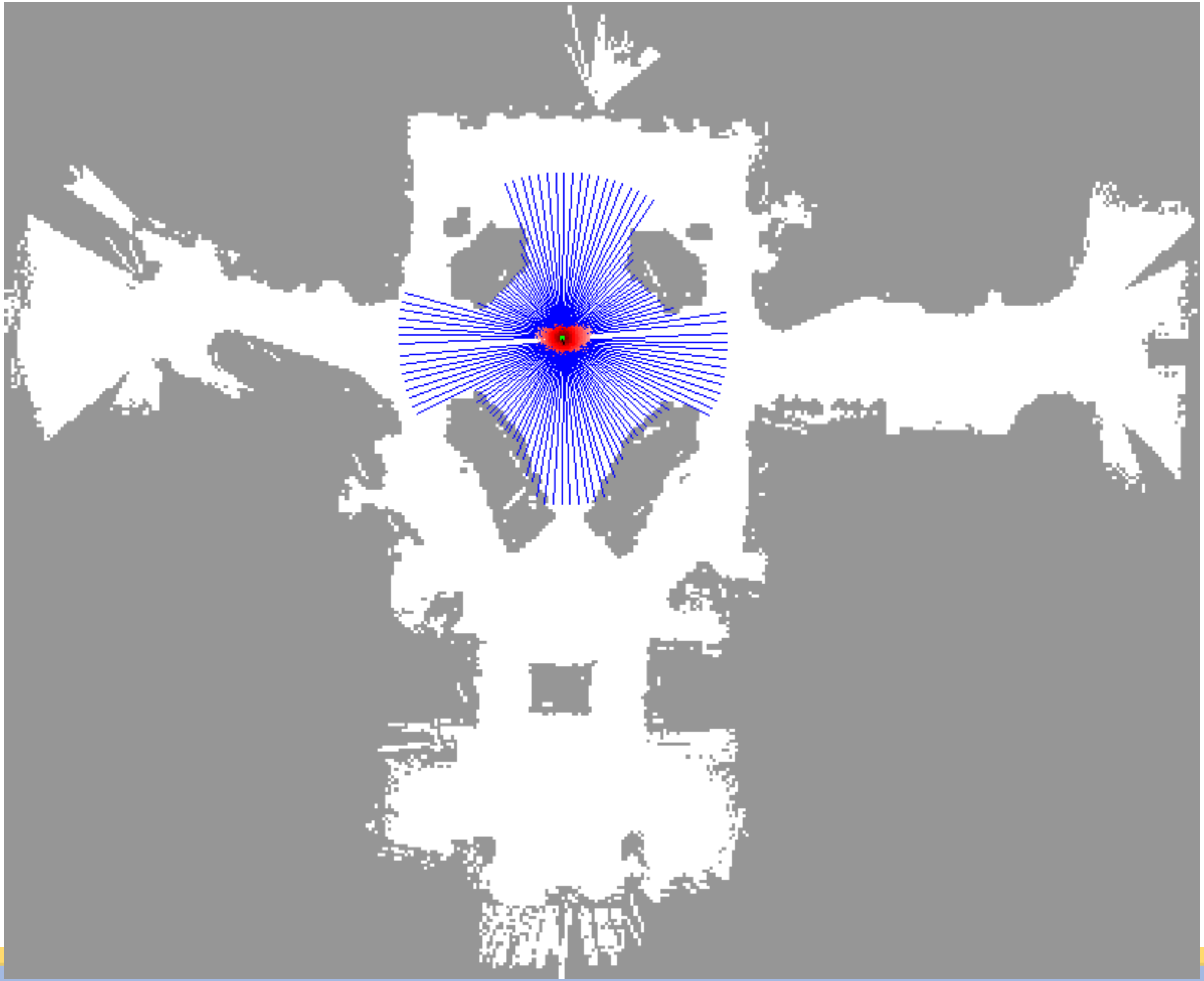


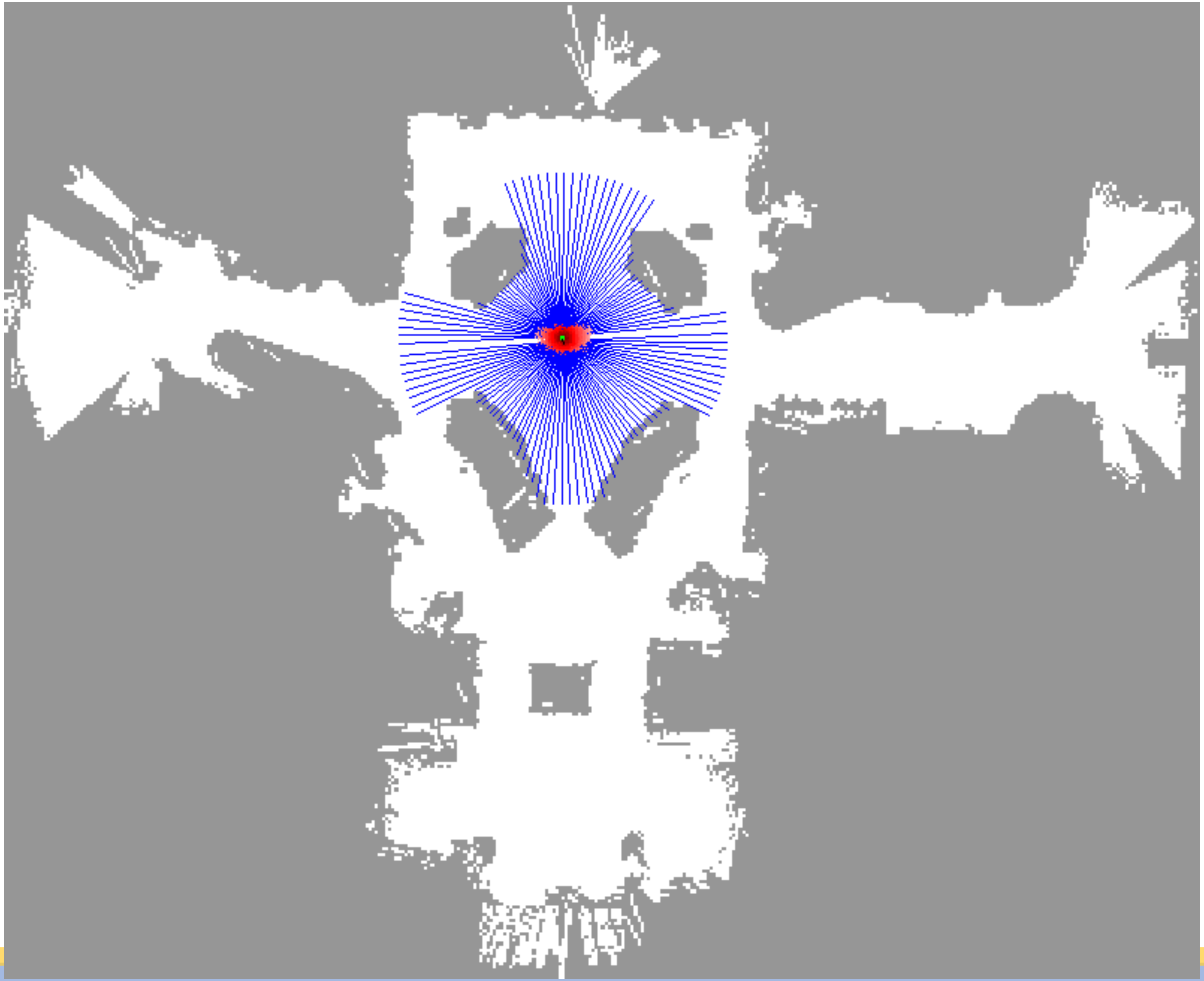




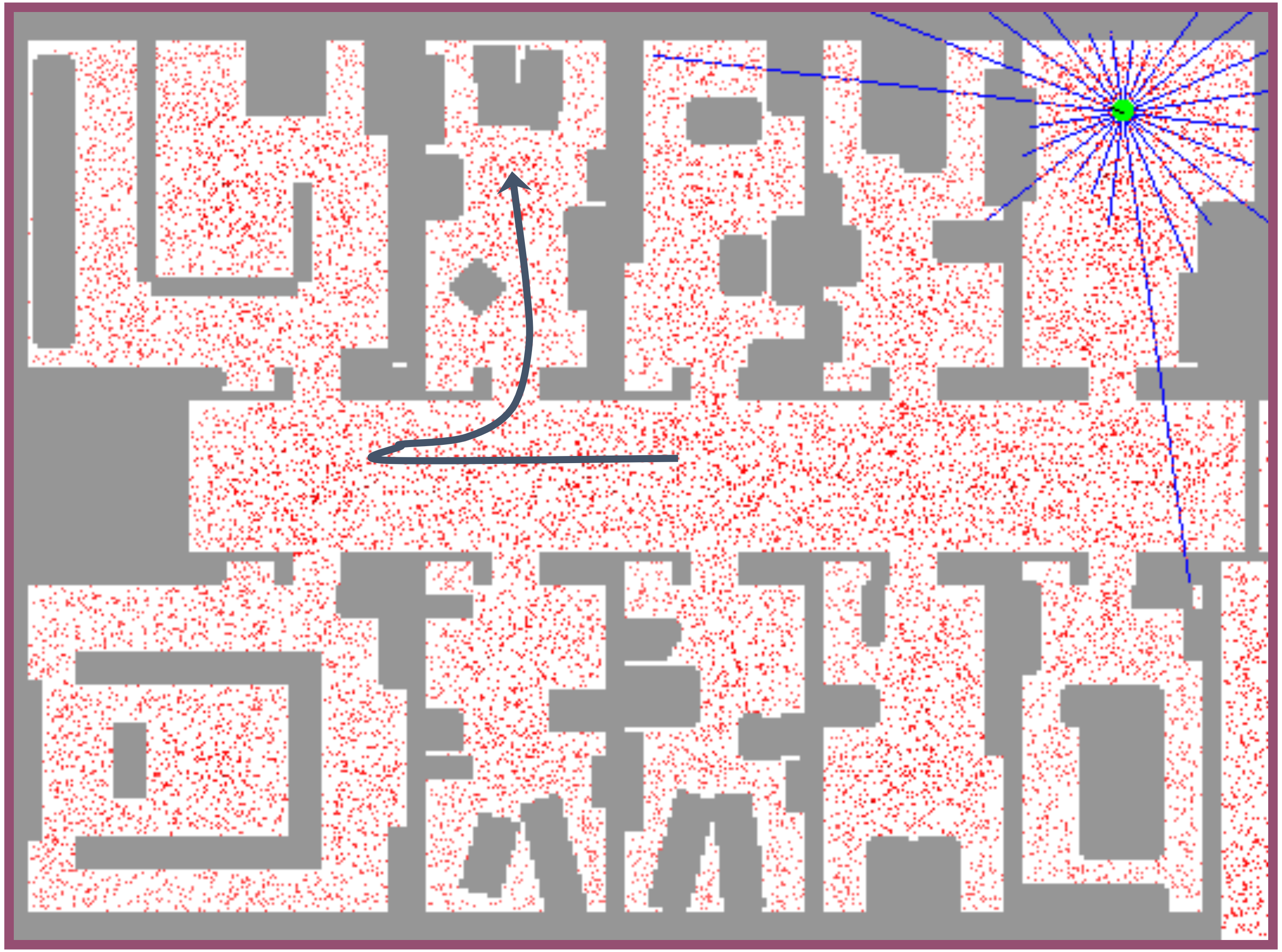




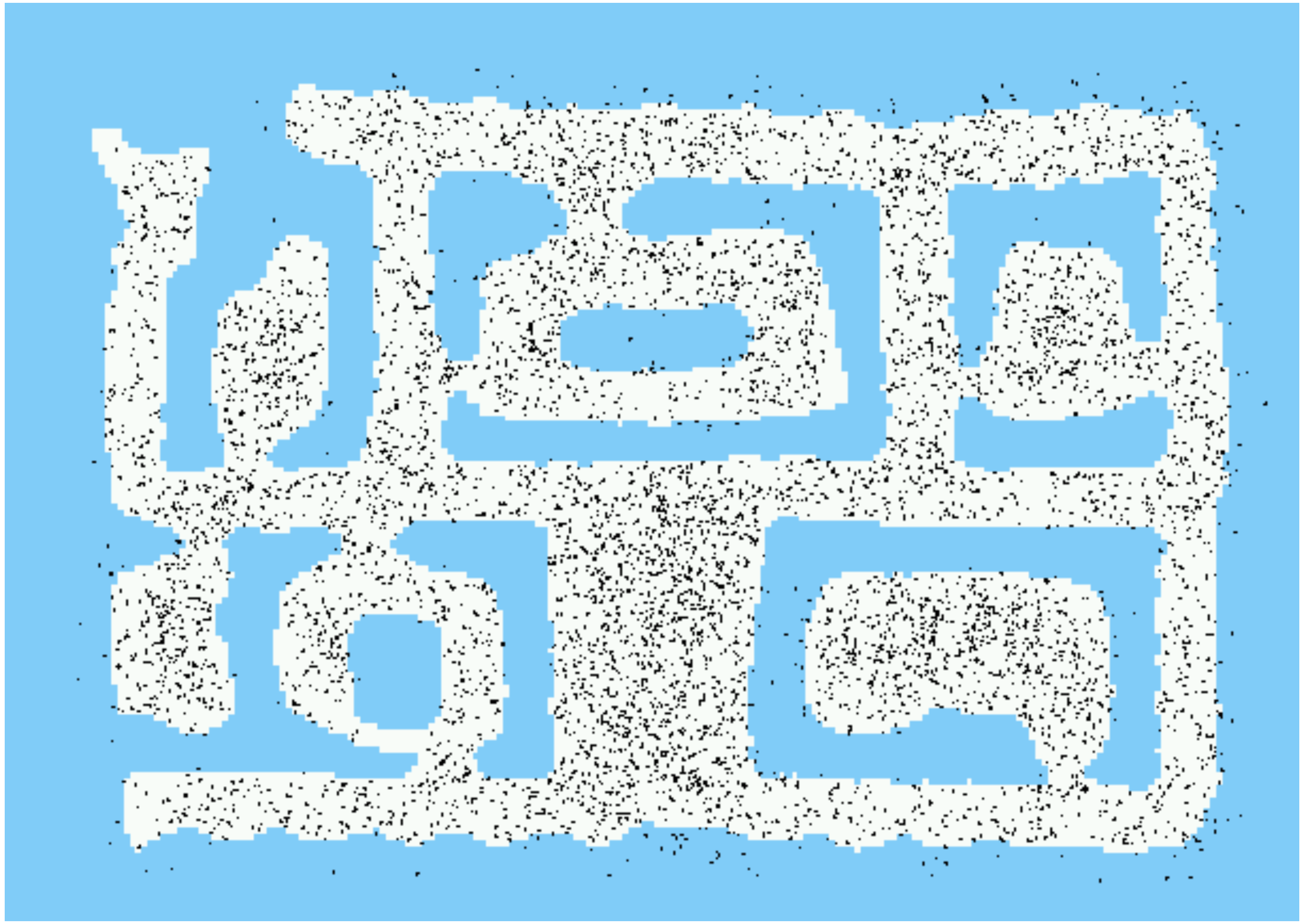




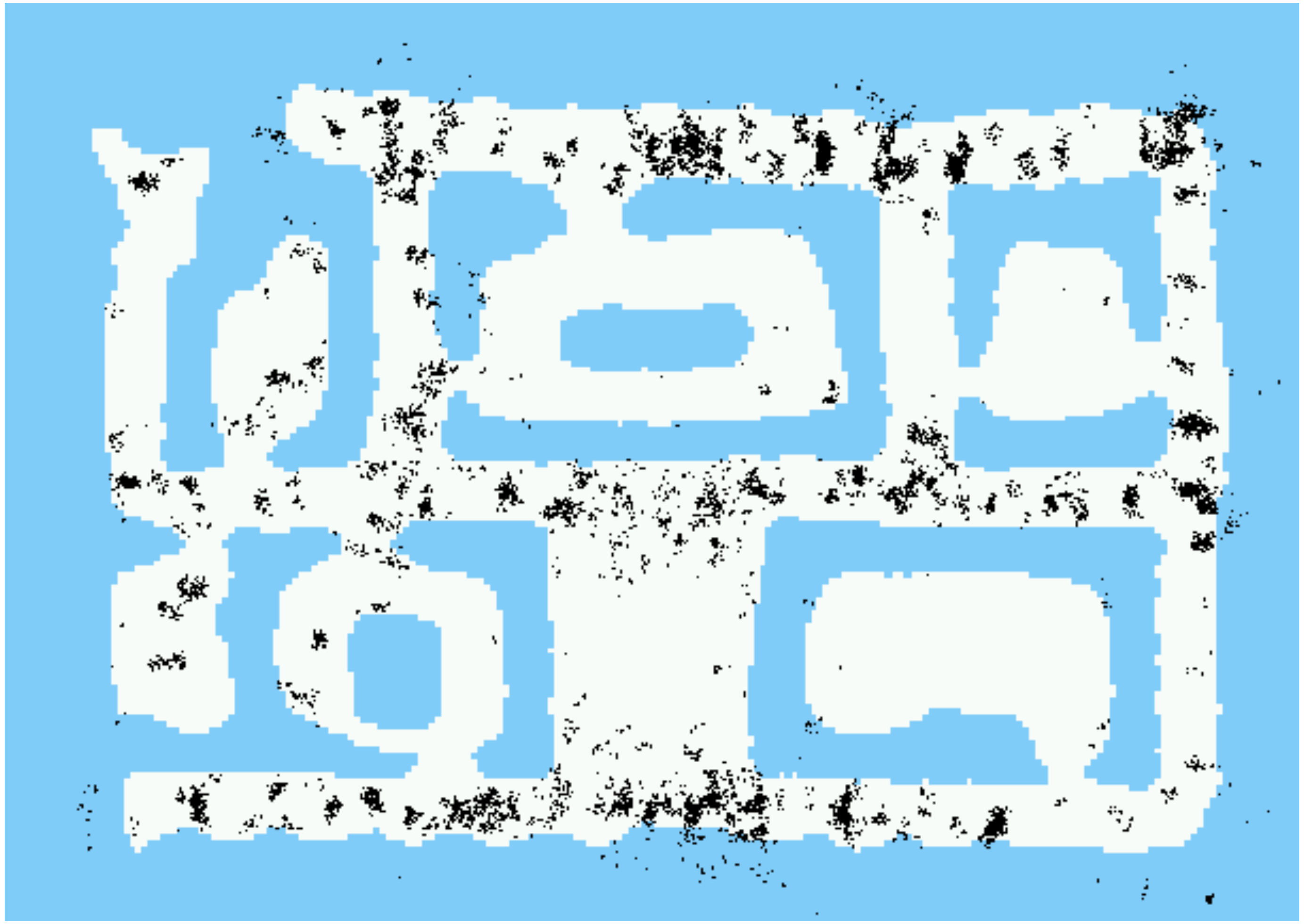
Sample-based Localization (sonar)



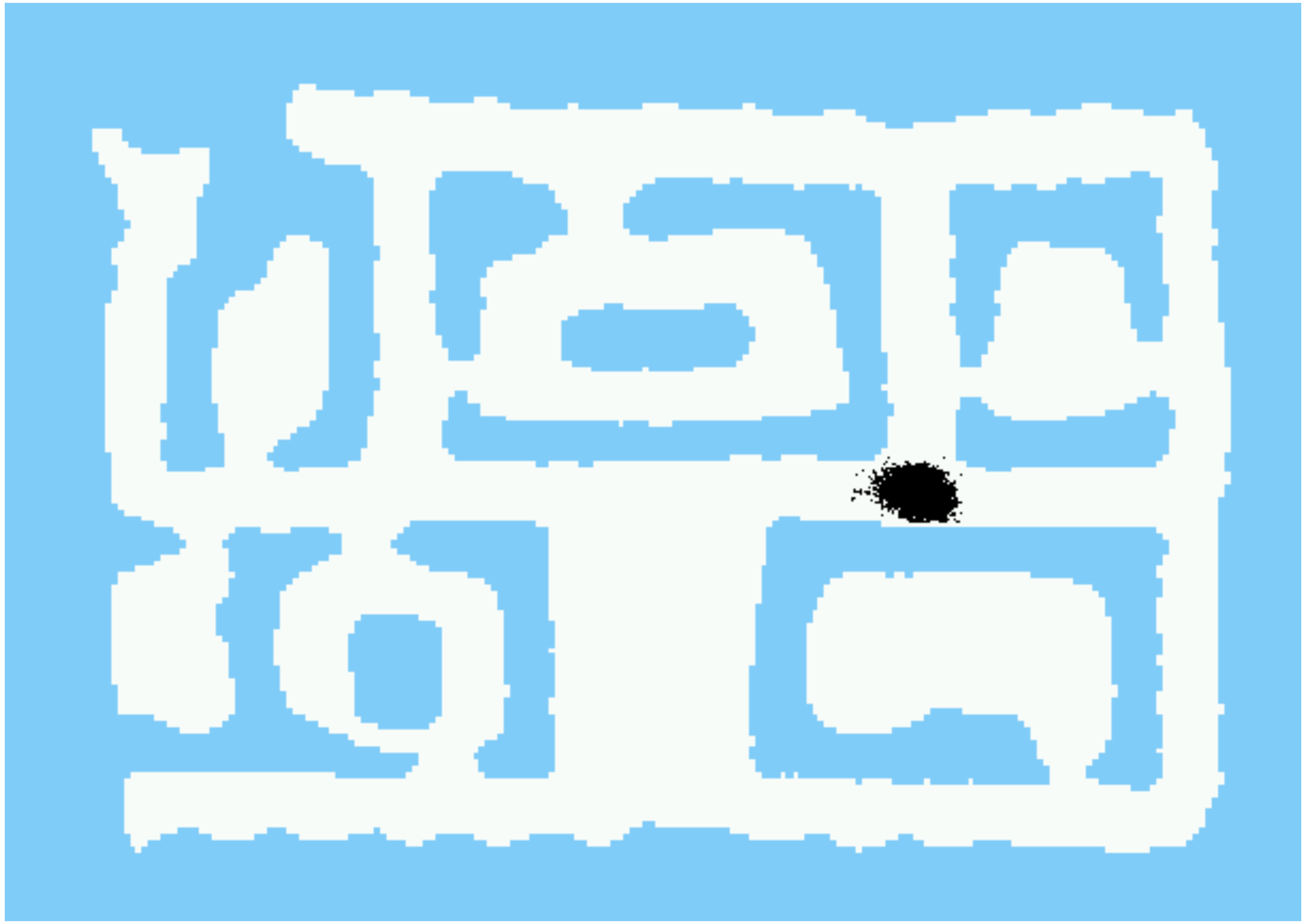
Initial Distribution



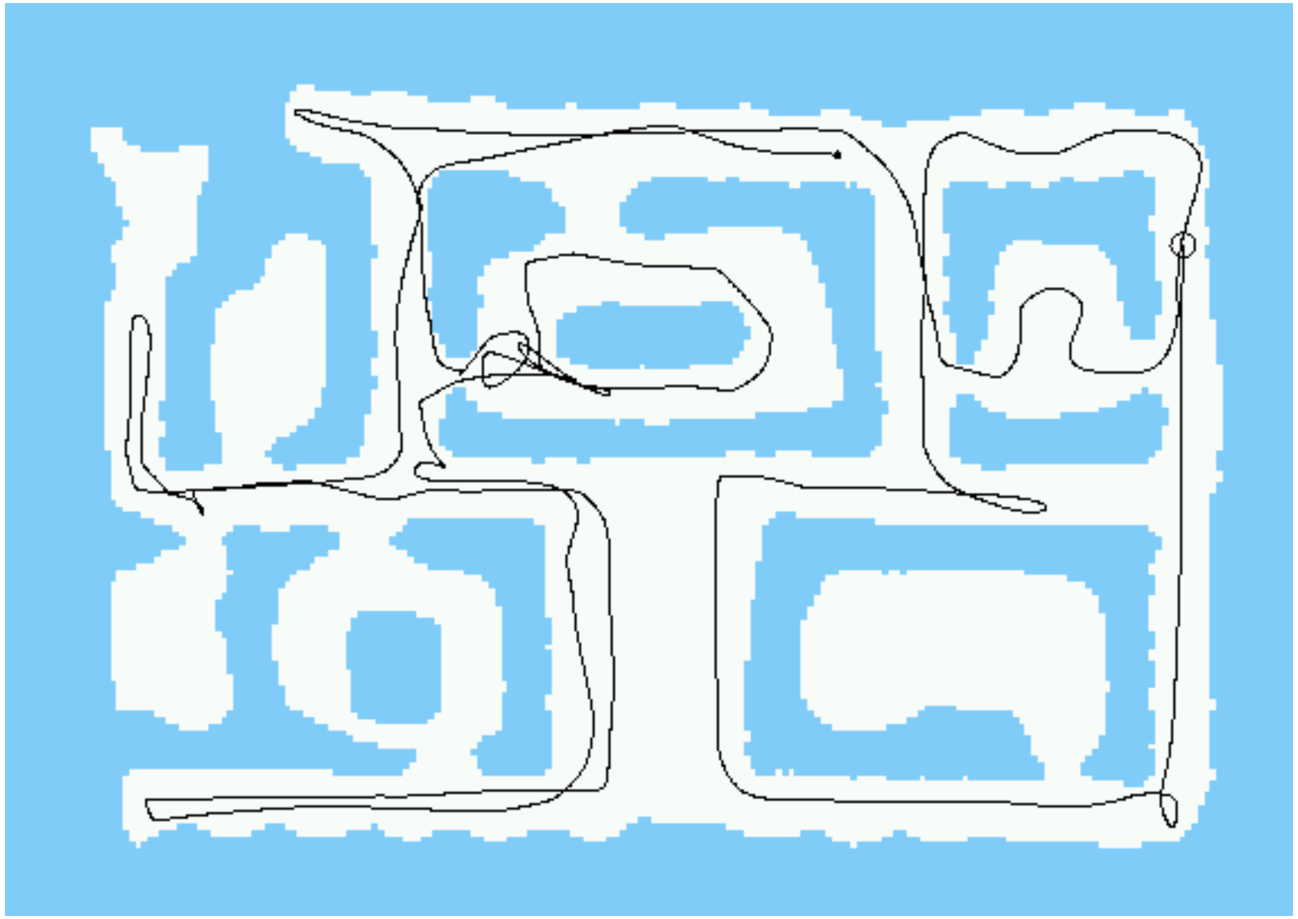
After Incorporating Ten Ultrasound Scans



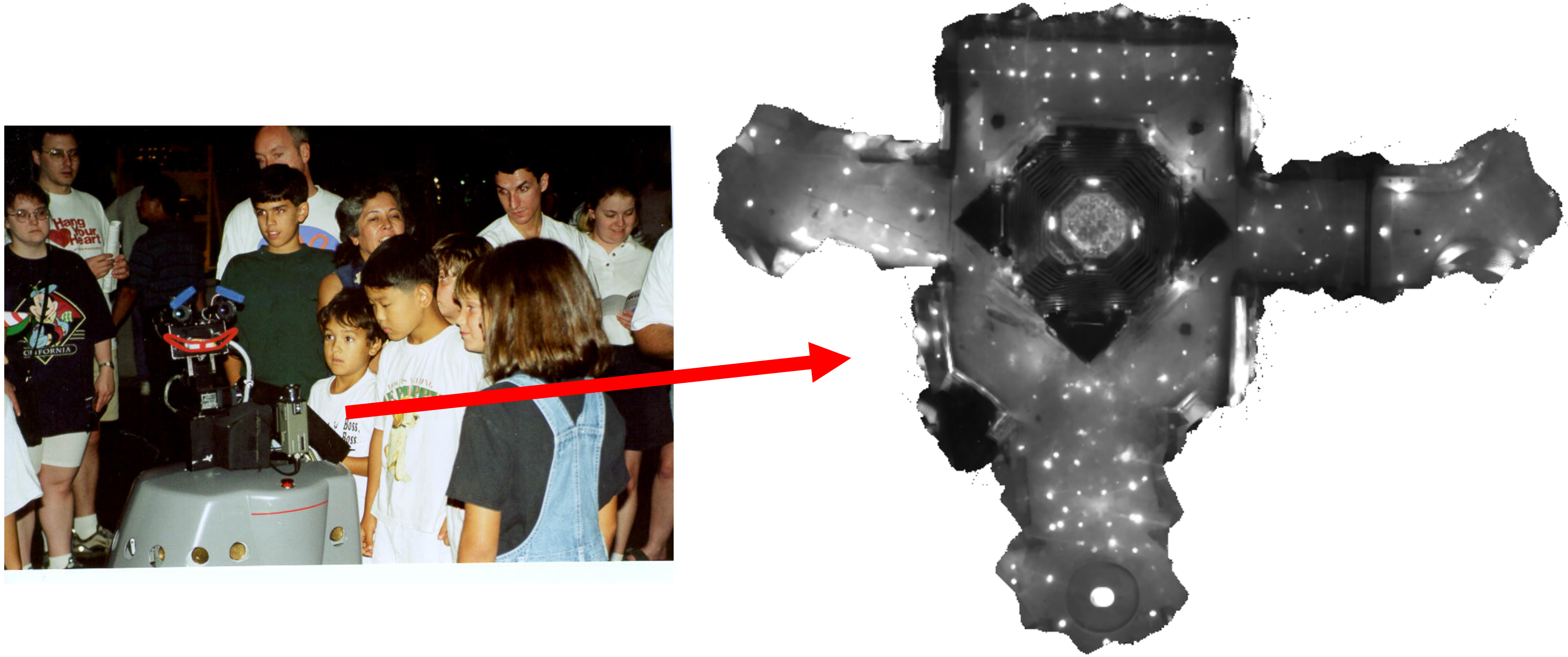
After Incorporating 65 Ultrasound Scans



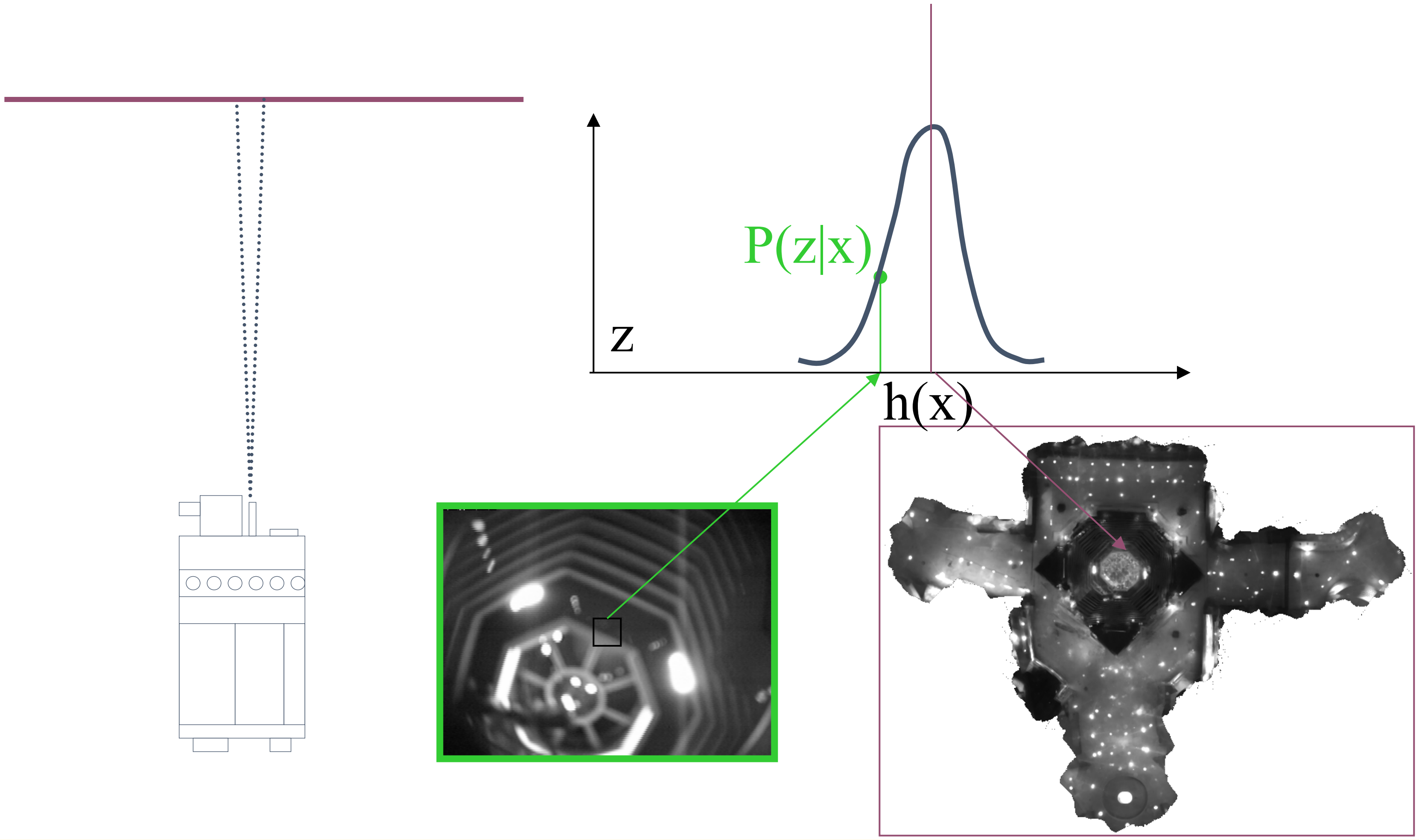
Estimated Path



Using Ceiling Maps for Localization

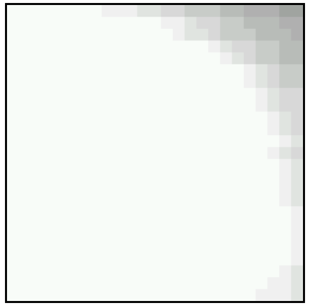


Vision-based Localization

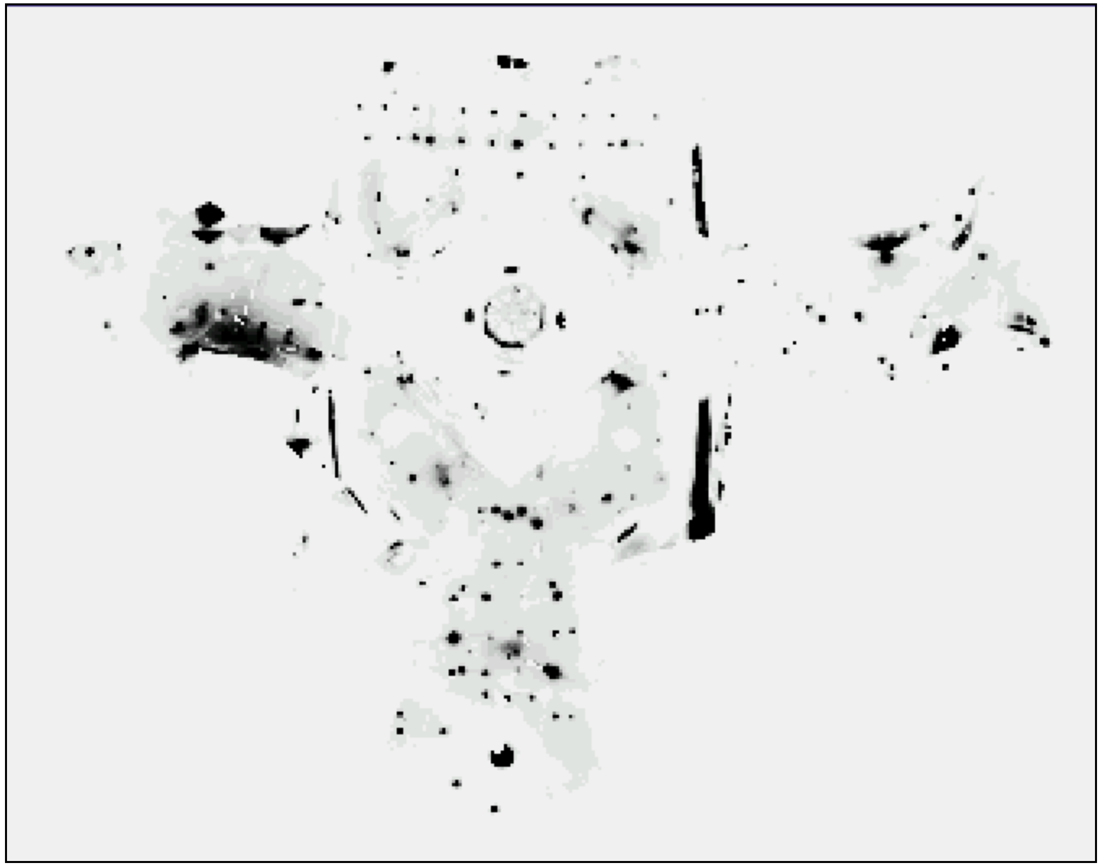


Under a Light

Measurement z:

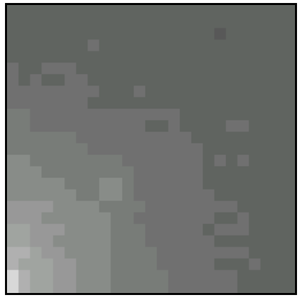


***P(z|x)*:**

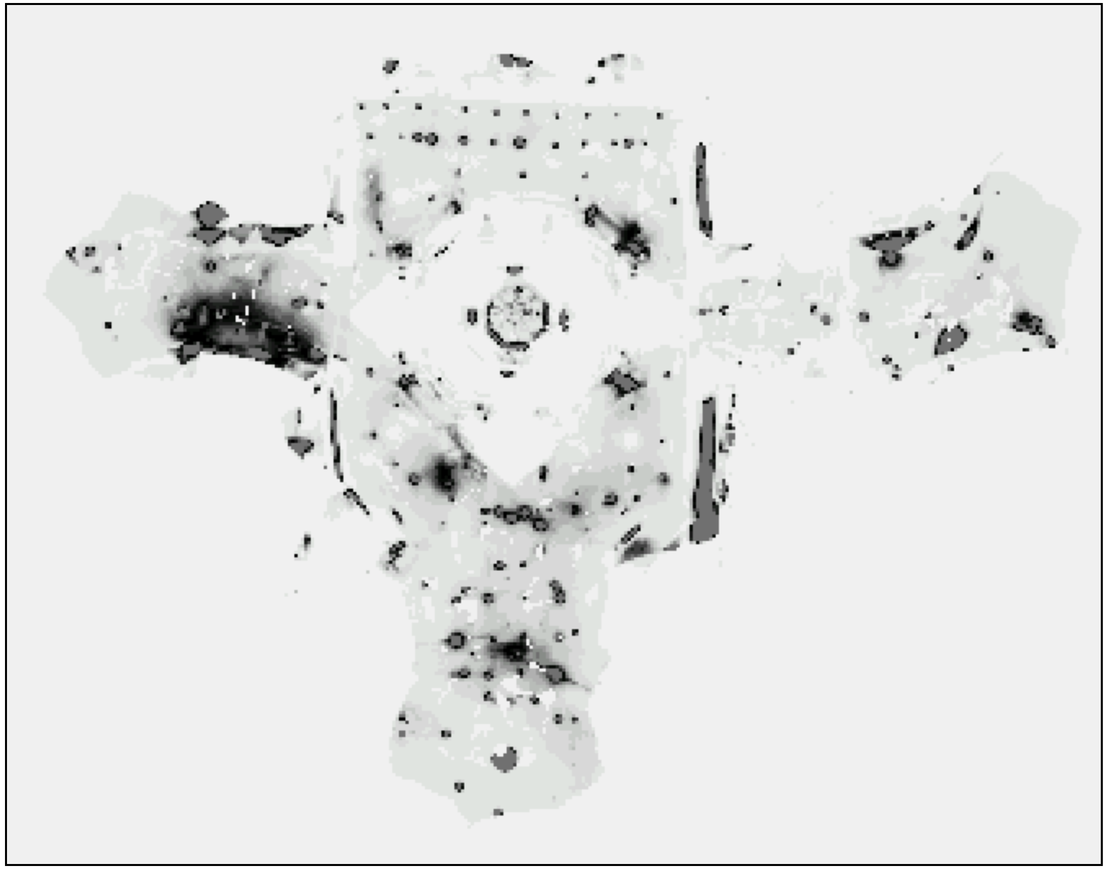


Next to a Light

Measurement z:



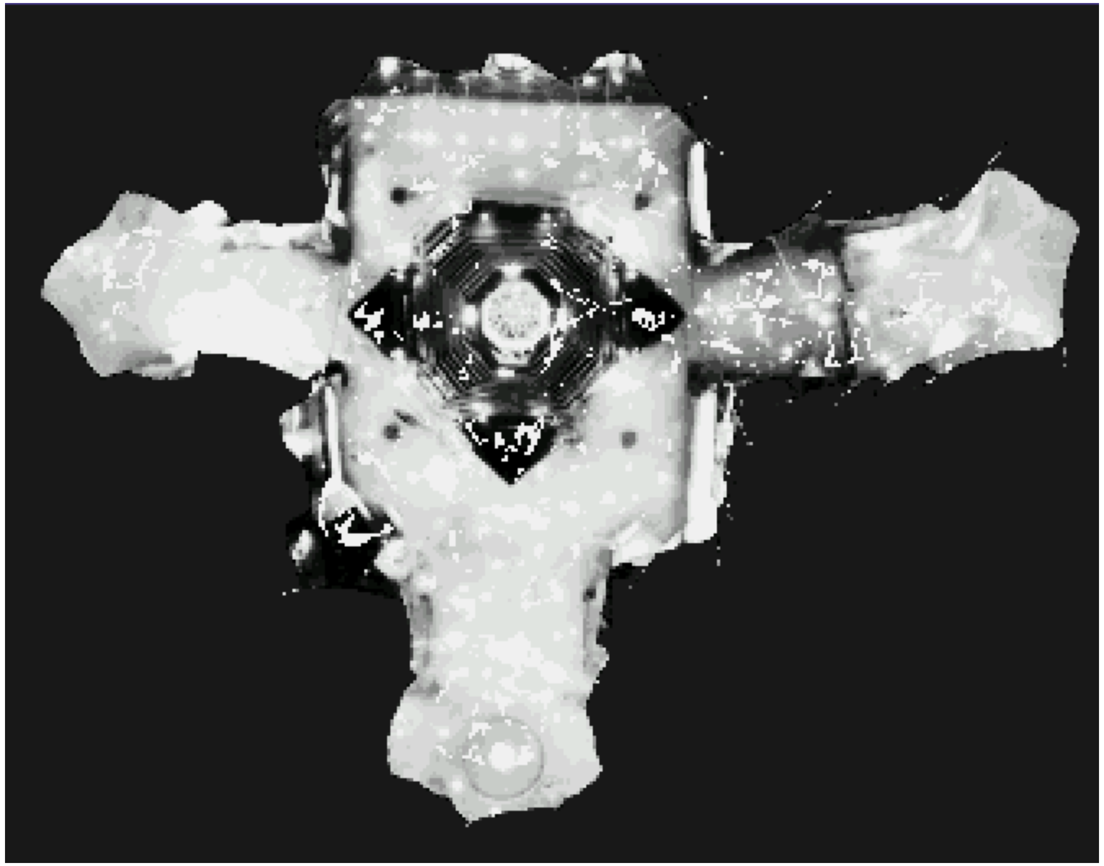
***P(z|x)*:**



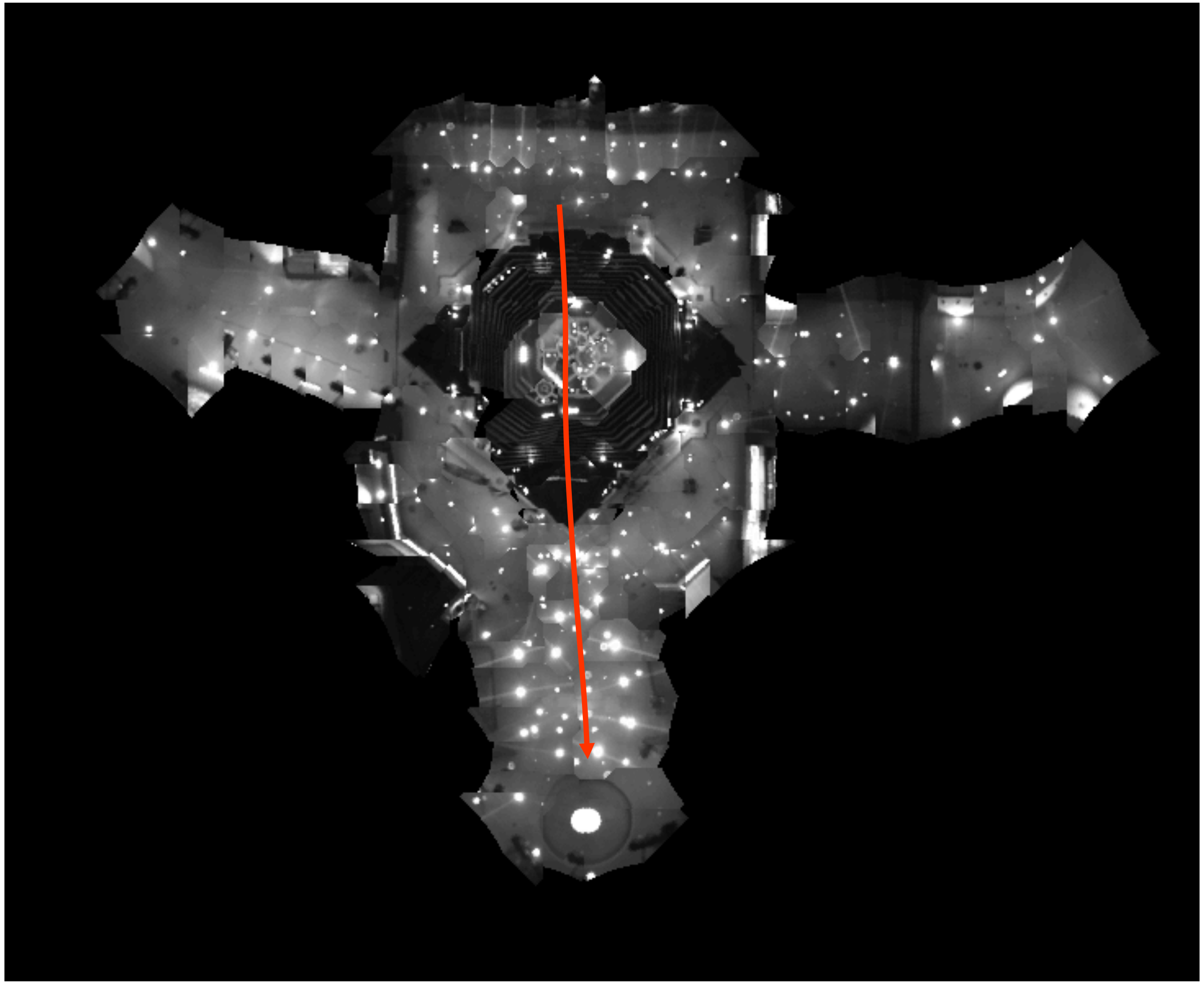
Elsewhere

Measurement z:

***P(z|x)*:**



Global Localization Using Vision



Limitations

- The approach described so far is able to
  - track the pose of a mobile robot and to
  - globally localize the robot.

- Can we deal with localization errors (i.e., the kidnapped robot problem)?
- How to handle localization errors/failures?
  - Particularly serious when the number of particles is small

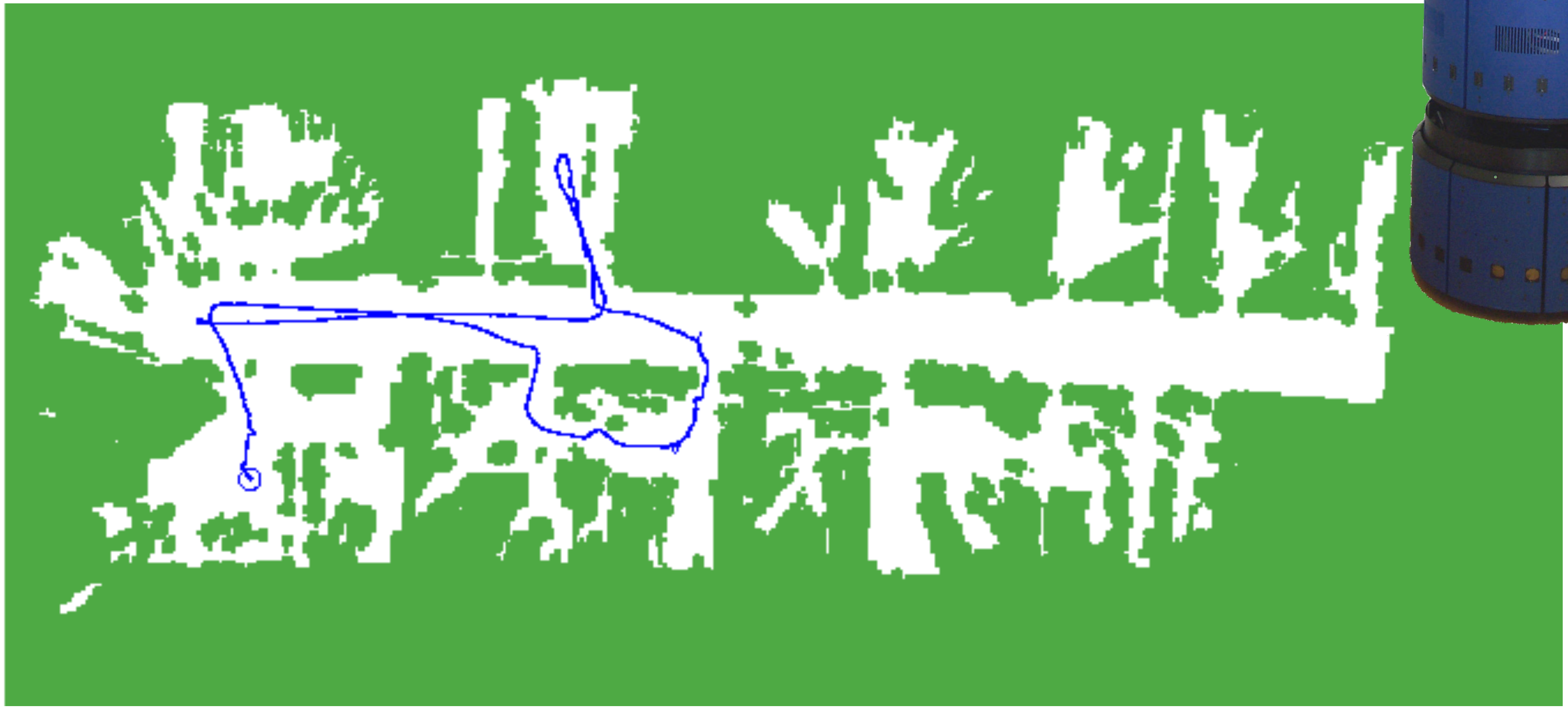
Approaches

- Randomly insert samples
  - Why?
  - The robot can be teleported at any point in time
- How many particles to add? With what distribution?
  - Add particles according to localization performance
  - Monitor the probability of sensor measurements $p(z_t|z_{1:t-1}, u_{1:t}, m)$
  - For particle filters: $p(z_t|z_{1:t-1}, u_{1:t}, m) \approx \frac{1}{M}\sum w_t^{[m]}$
- Insert random samples proportional to the average likelihood of the particles (the robot has been teleported with higher probability when the likelihood of its observations drops).

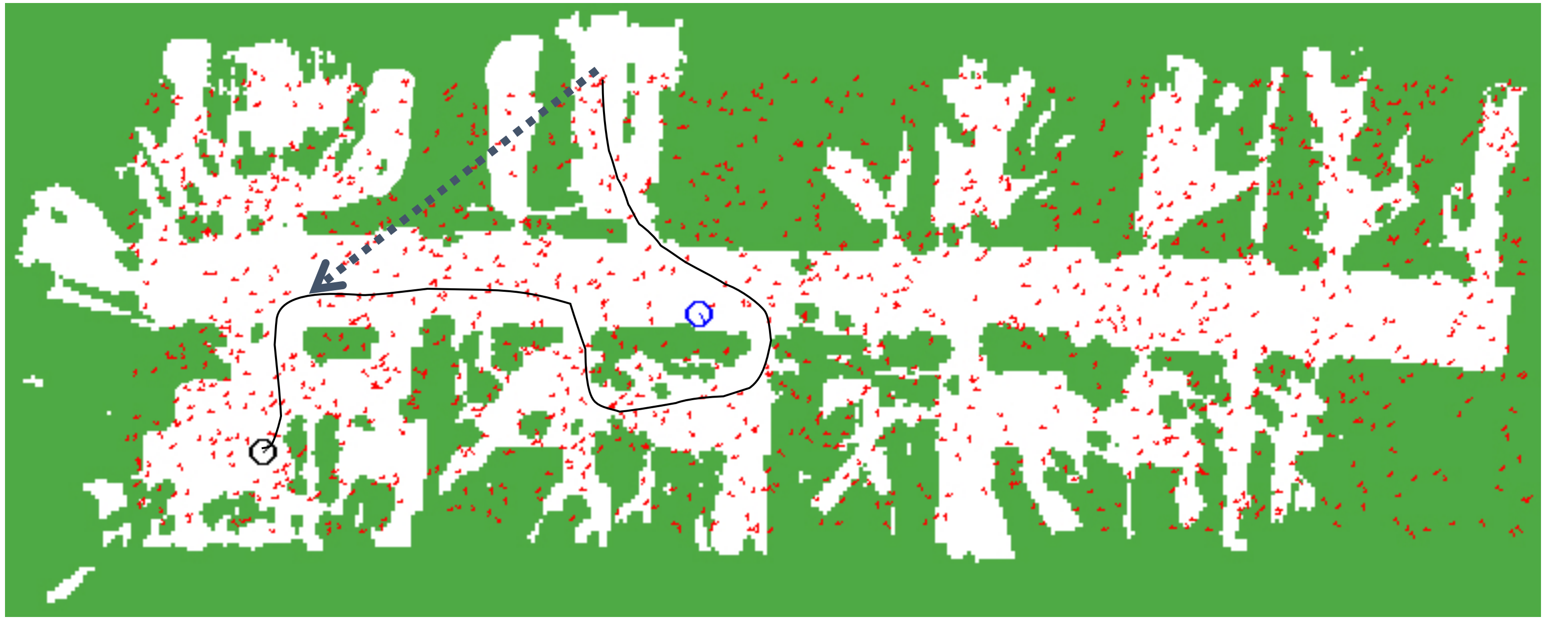
Random Samples
Vision-Based Localization

936 Images, 4MB, .6secs/image

Trajectory of the robot:



Kidnapping the Robot



Summary

- Particle filters are an implementation of recursive Bayesian filtering
- They represent the posterior by a set of weighted samples.
- In the context of localization, the particles are propagated according to the motion model.
- They are then weighted according to the likelihood of the observations.
- In a re-sampling step, new particles are drawn with a probability proportional to the likelihood of the observation.