Lecture 04:
particle filters + Kalman filters

Katie DC

Jan. 30, 2020

Notes from Probabilistic Robotics Ch. 3,

Lukas Luft, and Wolfram Burgard

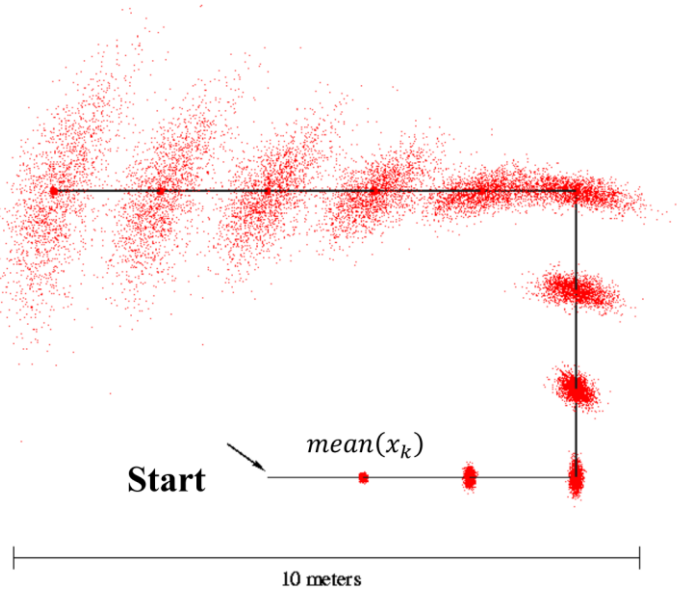
Admin

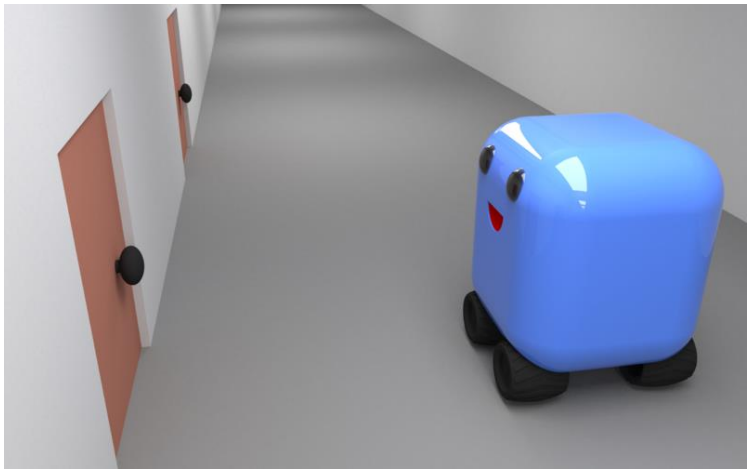
- HW1 is due tomorrow (Friday)
  - Also first homework party is tomorrow
- Next Tuesday will be a quick tutorial for HW3 and the associated lab
- Next Thursday is our first Guest Lecture

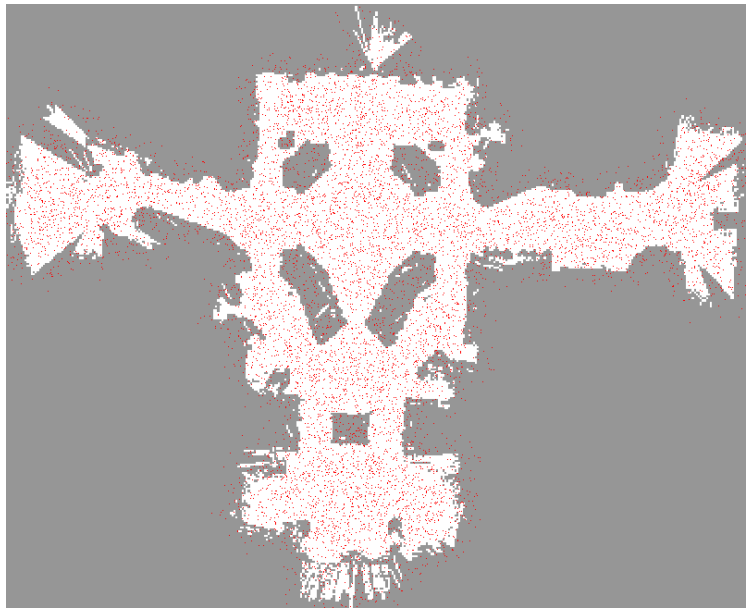
Who was Rudolf Kalman?

Image Credit: Wikipedia

- Kálmán was one of the most influential people on control theory today and is most known for his co-invention of the Kalman filter (or Kalman-Bucy Filter)
- The filtering approach was initially met with vast skepticism, so much so that he was forced to do the first publication of his results in mechanical engineering, rather than in electrical engineering or systems engineering
  - This worked out find as some of the first use cases was with NASA on the Apollo spacecraft
- *Kalman filters are inside every robot, commercial airplanes, uses in seismic data processing, nuclear power plant instrumentation, and demographic models, as well as applications in econometrics*


Start
$mean(x_k)$
10 meters





Particle Filters

- Particle filters are an implementation of recursive Bayesian filtering, where the posterior is represented by a set of weighted samples
- Instead of a precise probability distribution, represent belief $bel(x_t)$ by a set of particles, where each particle tracks its own state estimate
- Random sampling used in generation of particles
- Particles are periodically re-sampled, with probability weighted by likelihood of last generation of particles
- Nonparametric filter – can approximate complex probability distributions without explicitly computing the closed form solutions

Particle Filtering Algorithm // Monte Carlo Localization

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

- motion model guides the motion of particles
- $w_t^{[m]}$ is the importance factor or weight of each particle m, which is a function of the measurement model and belief
- Particles are resampled according to weight
- **Survival of the fittest:** moves/adds particles to part of space with higher probability

Particle Filtering Algorithm // Monte Carlo Localization

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

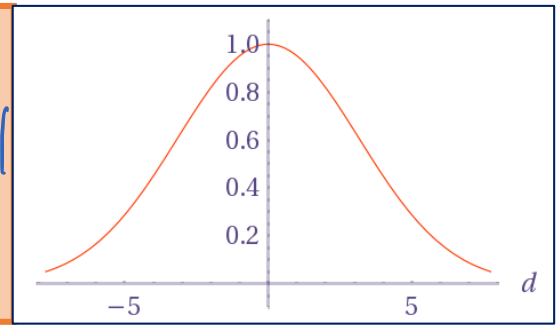
Algorithm MCL(X_{t-1}, u_t, z_t,m):
$\bar{X}_{t-1} = X_t = \emptyset$
for all m in [M] do:
$x_t^{[m]} = \boldsymbol{sample\_motion\_model}(u_t\ x_{t-1}^{[m]})$
$w_t^{[m]} = \boldsymbol{measurement\_model}(z_t, x_t^{[m],m})$
$\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
end for
for all m in [M] do:
draw i *with probability* $\propto w_t^{[i]}$
add $x_t^{[i]}$ *to* X_t
end for
return X_t

Step 1: Initialize particles uniformly distribute over space and assign initial weight $x^{(m)}$

Step 2: Sample the motion model to propagate particles

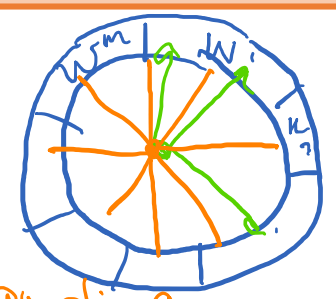
Step 3: Read measurement model and assign (unnormalized) weight:

$$w_t^{[m]} = \exp\left(\frac{-d^2}{2\sigma}\right) \quad \|z_t^R - z_t^m\|$$

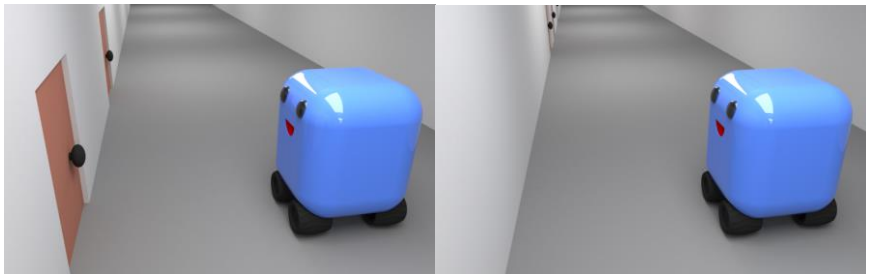


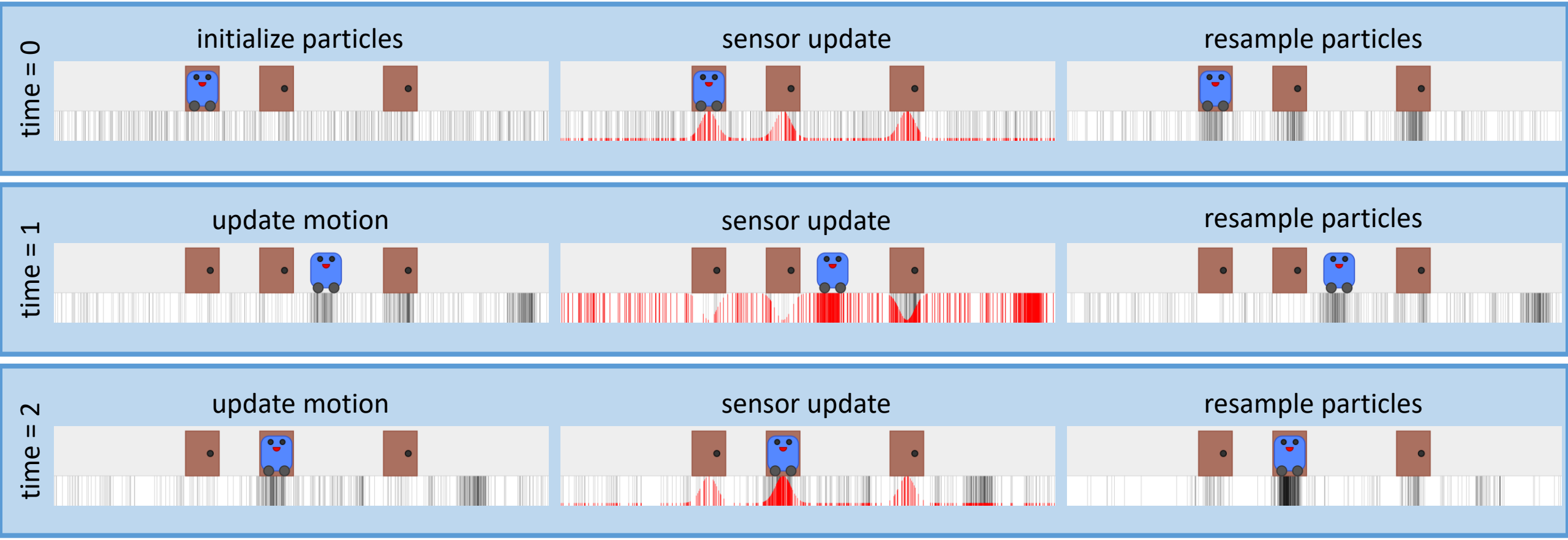
Step 4: Calculate your position update estimate by adding the particle positions scaled by weight
Note that weights must be normalized to sum to 1

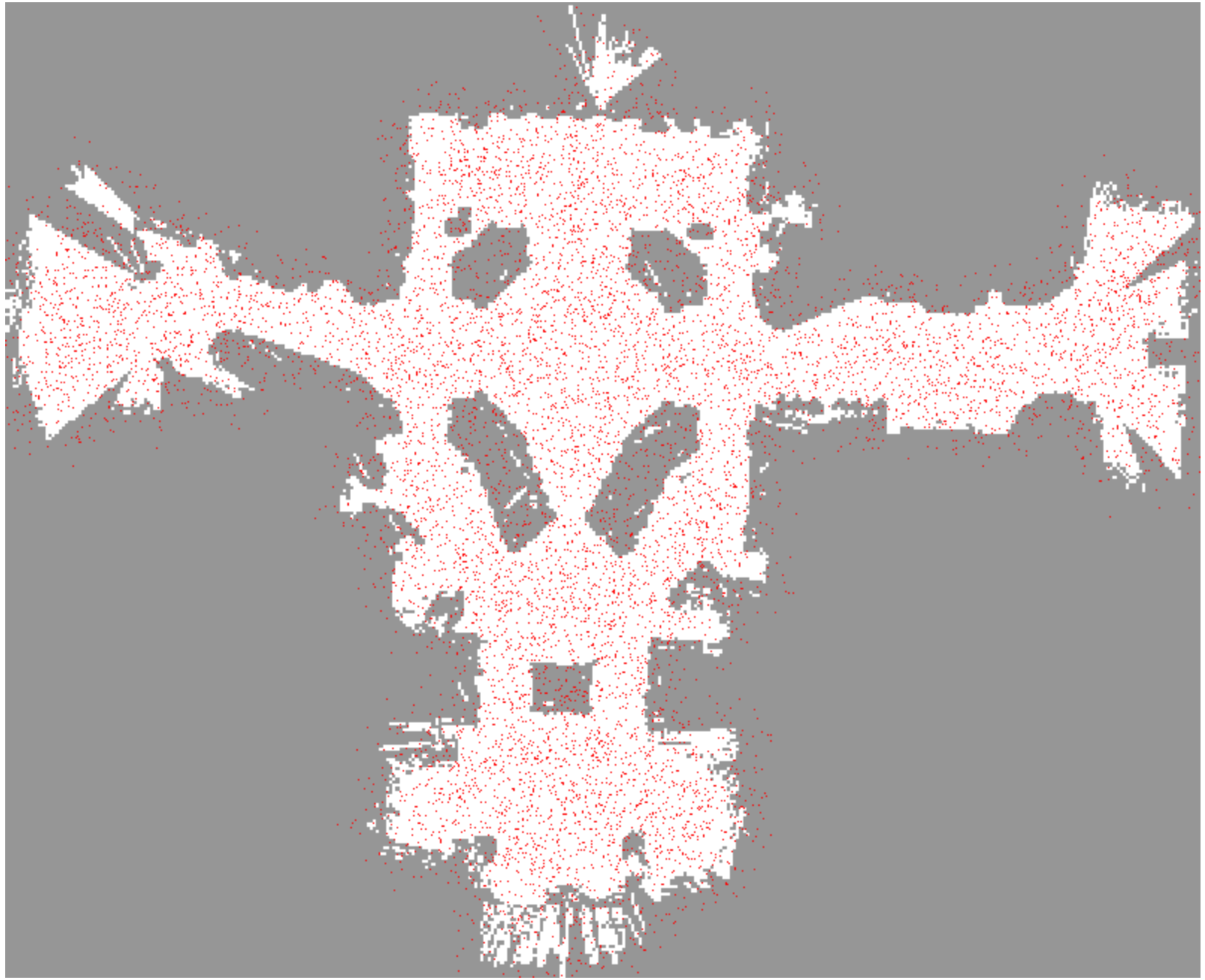
Step 5: Choose which particles to *resample* in the next iteration by replacing less likely particles with more likely ones

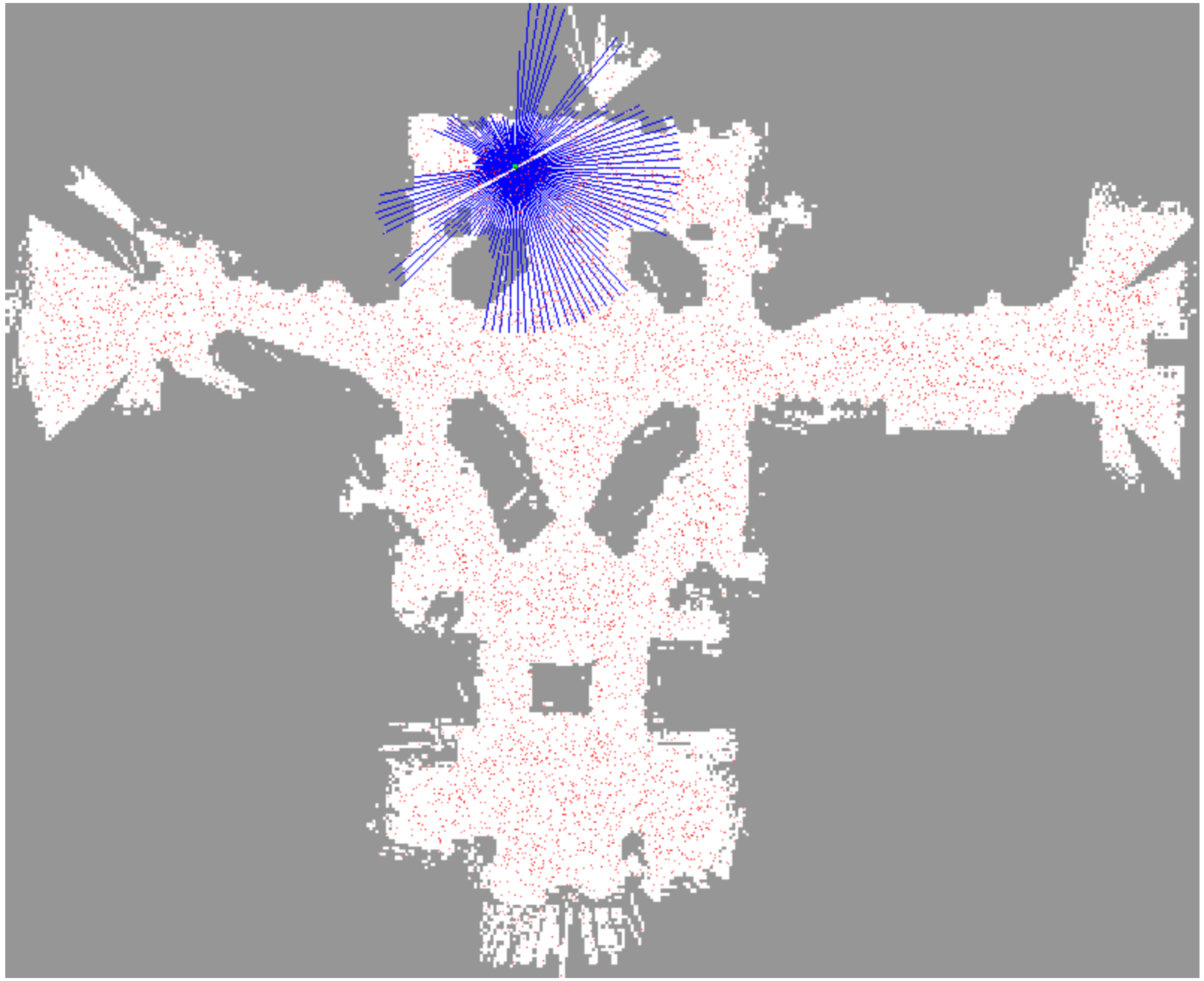


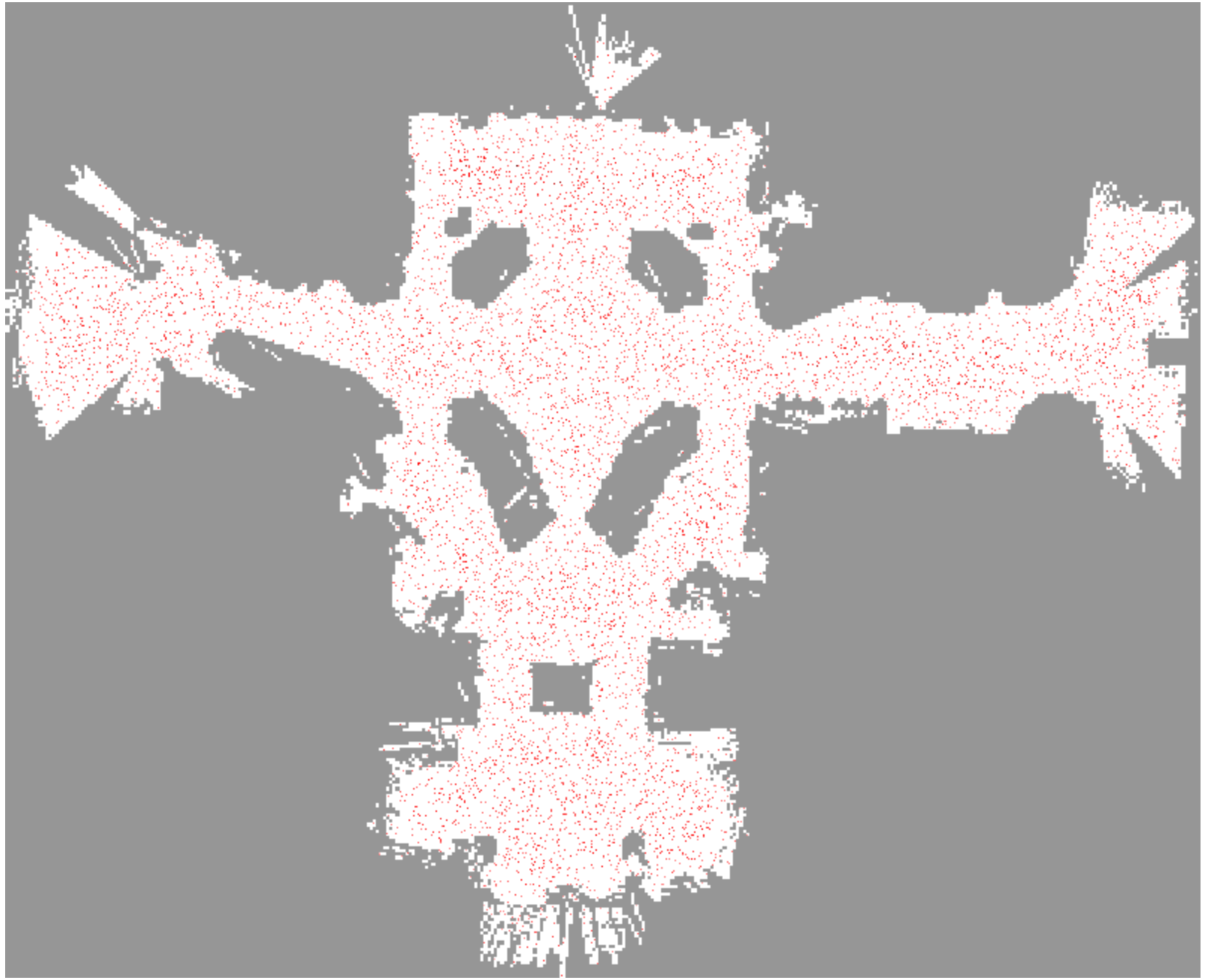
Particle Filter Localization - Illustration

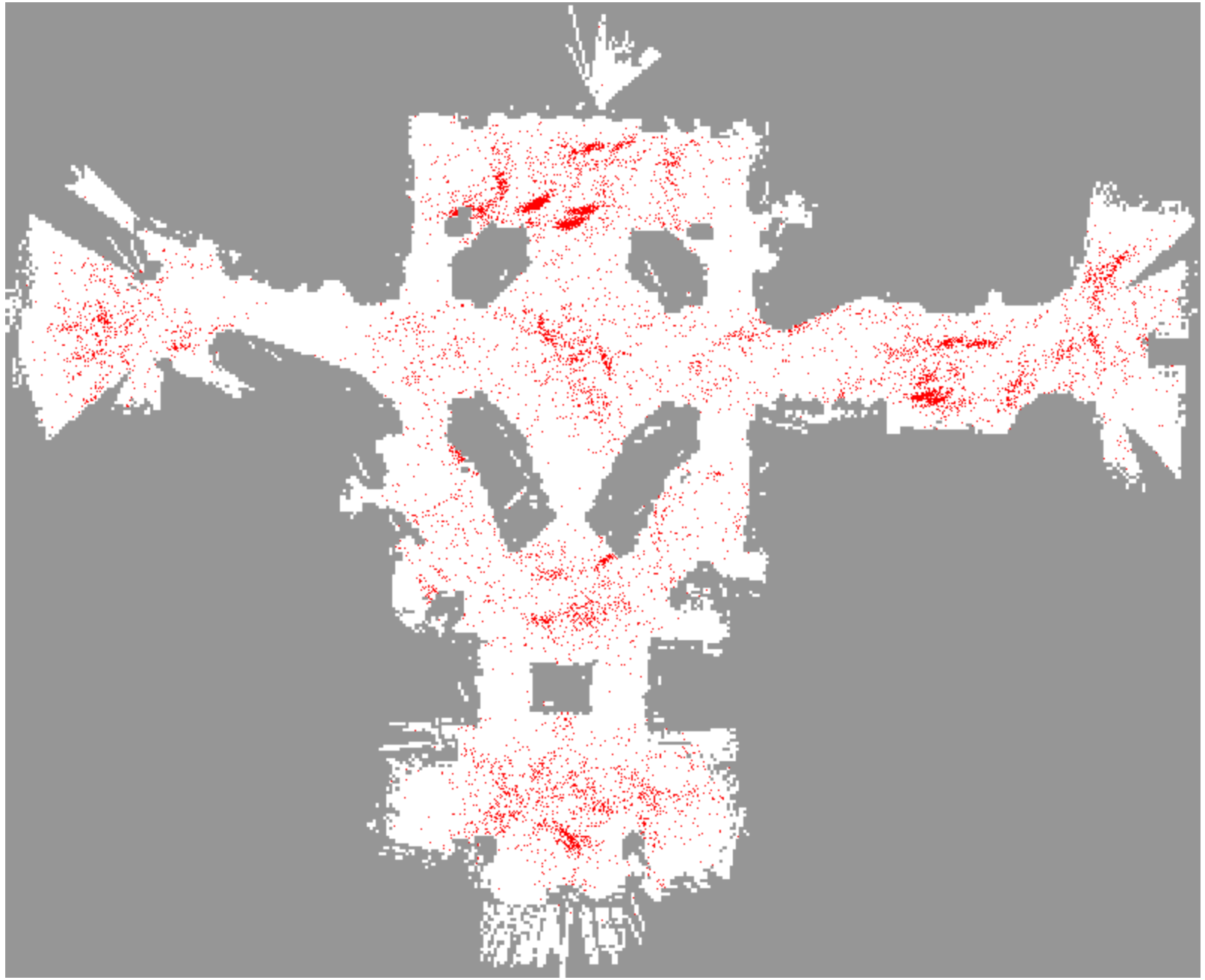


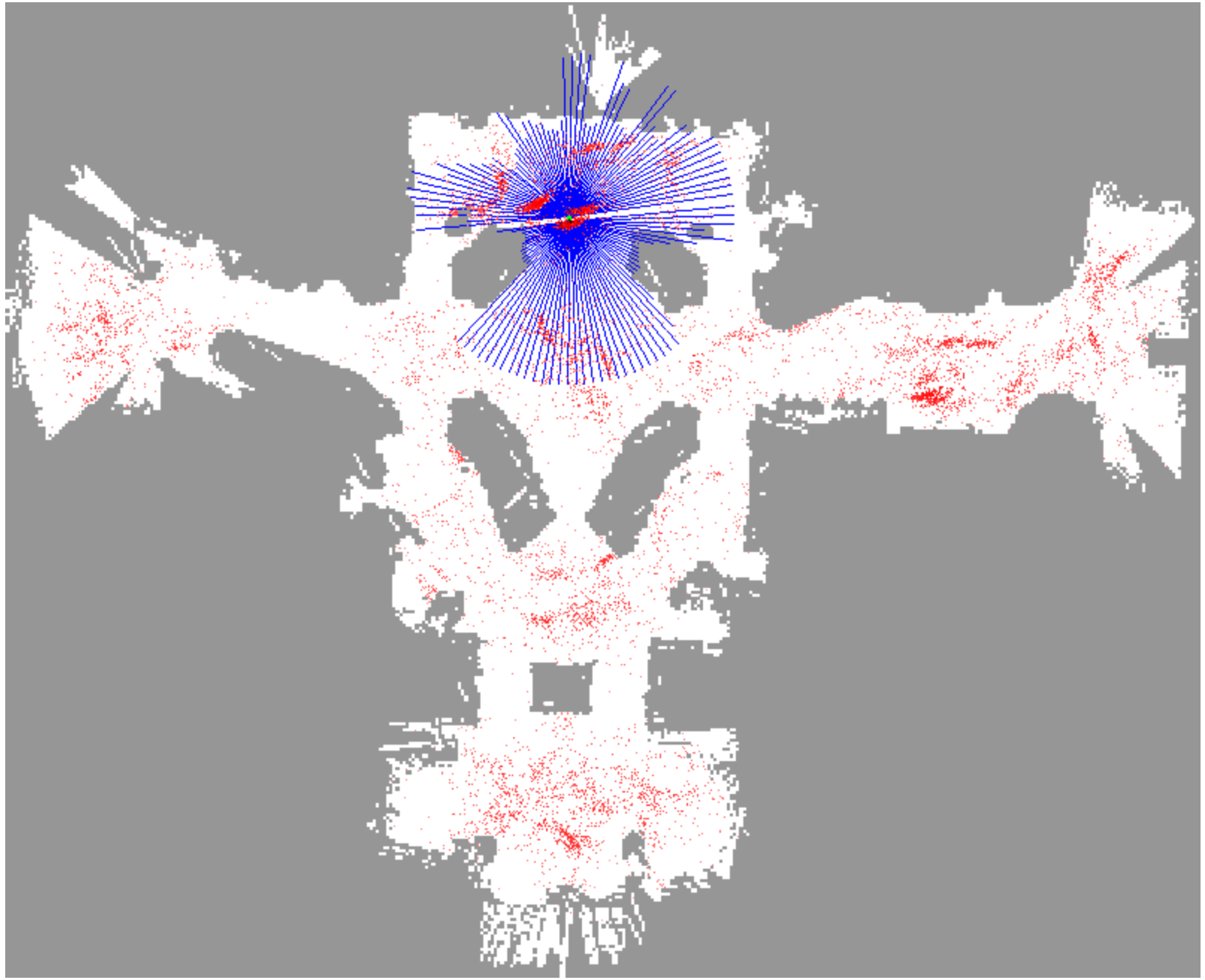


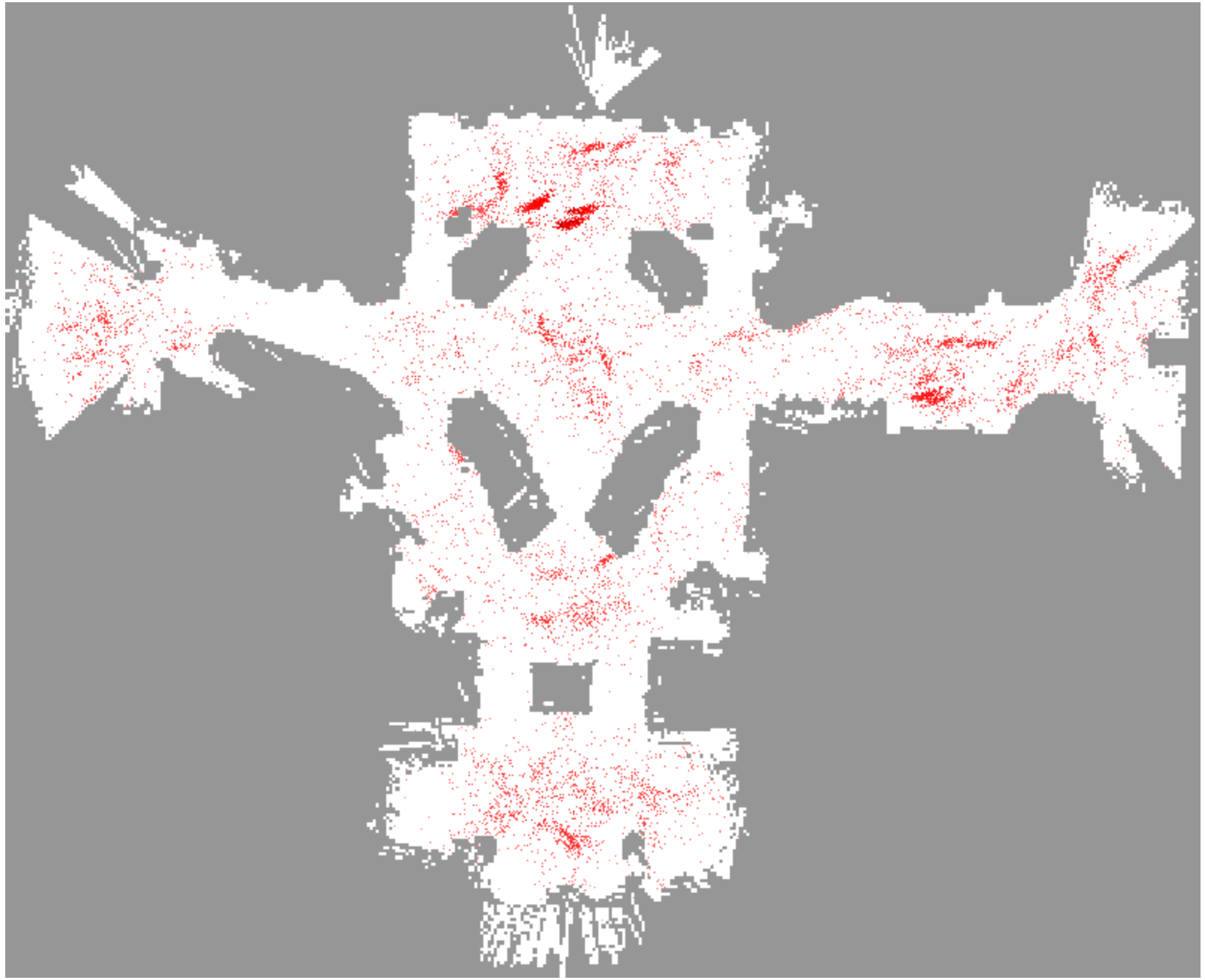


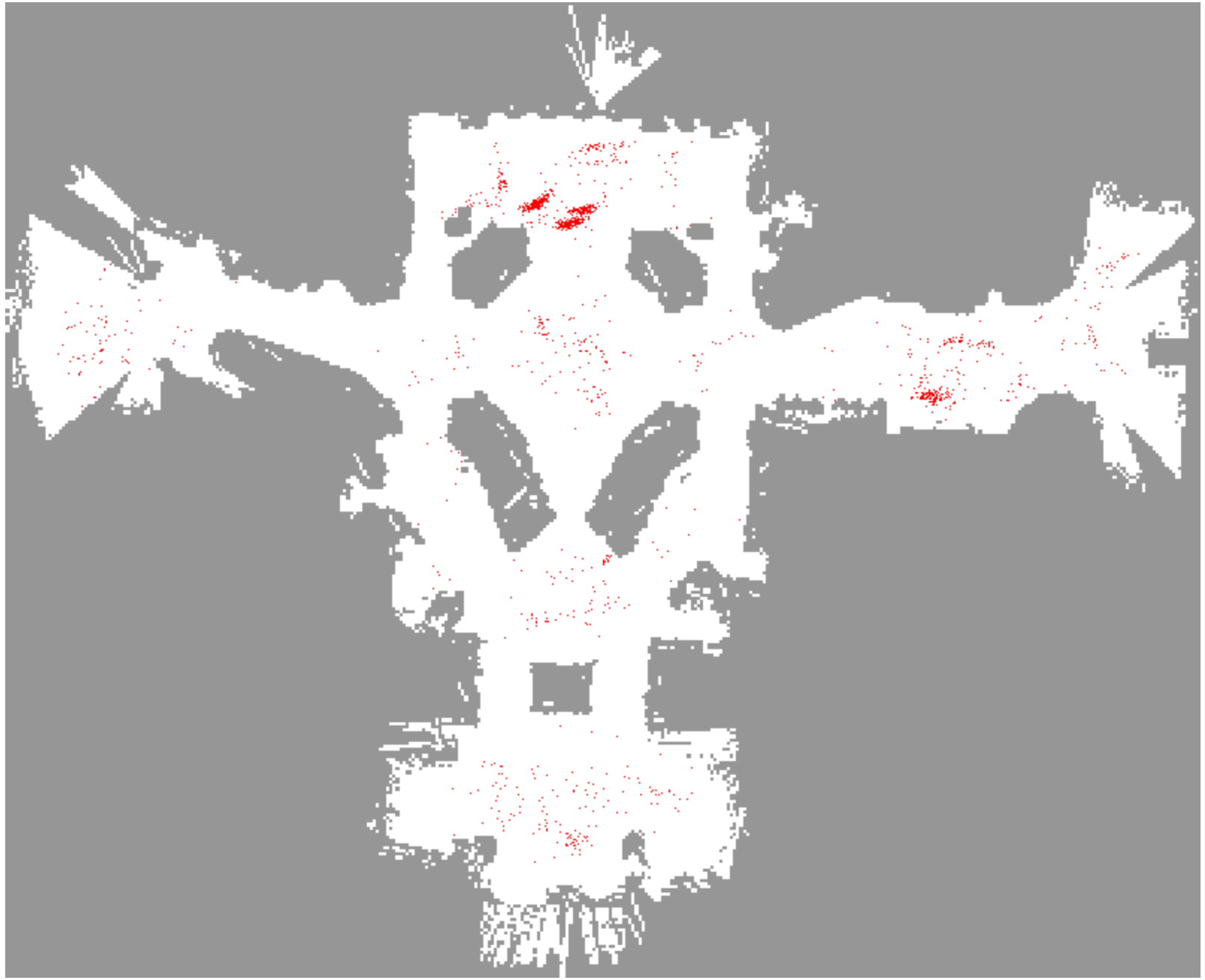


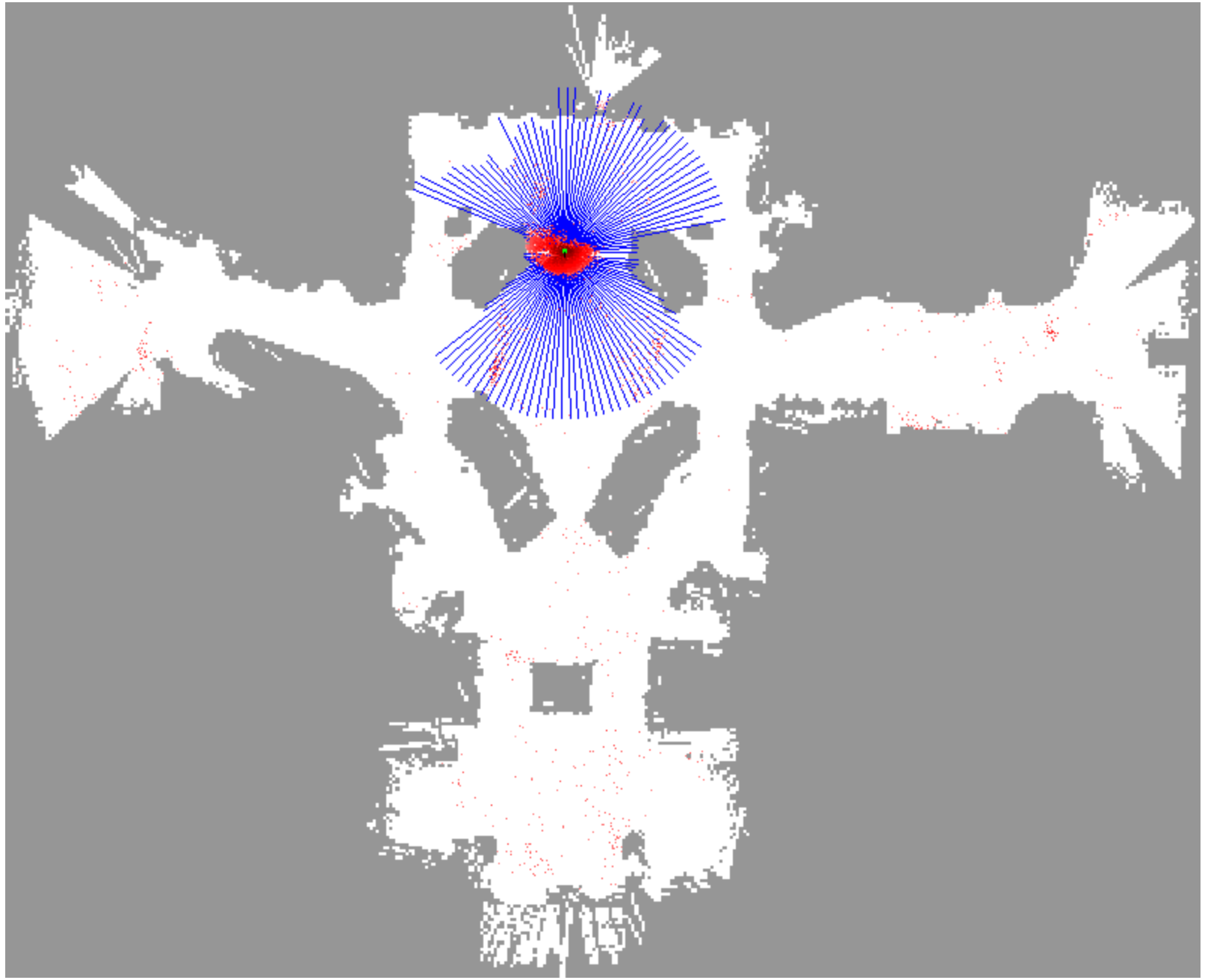


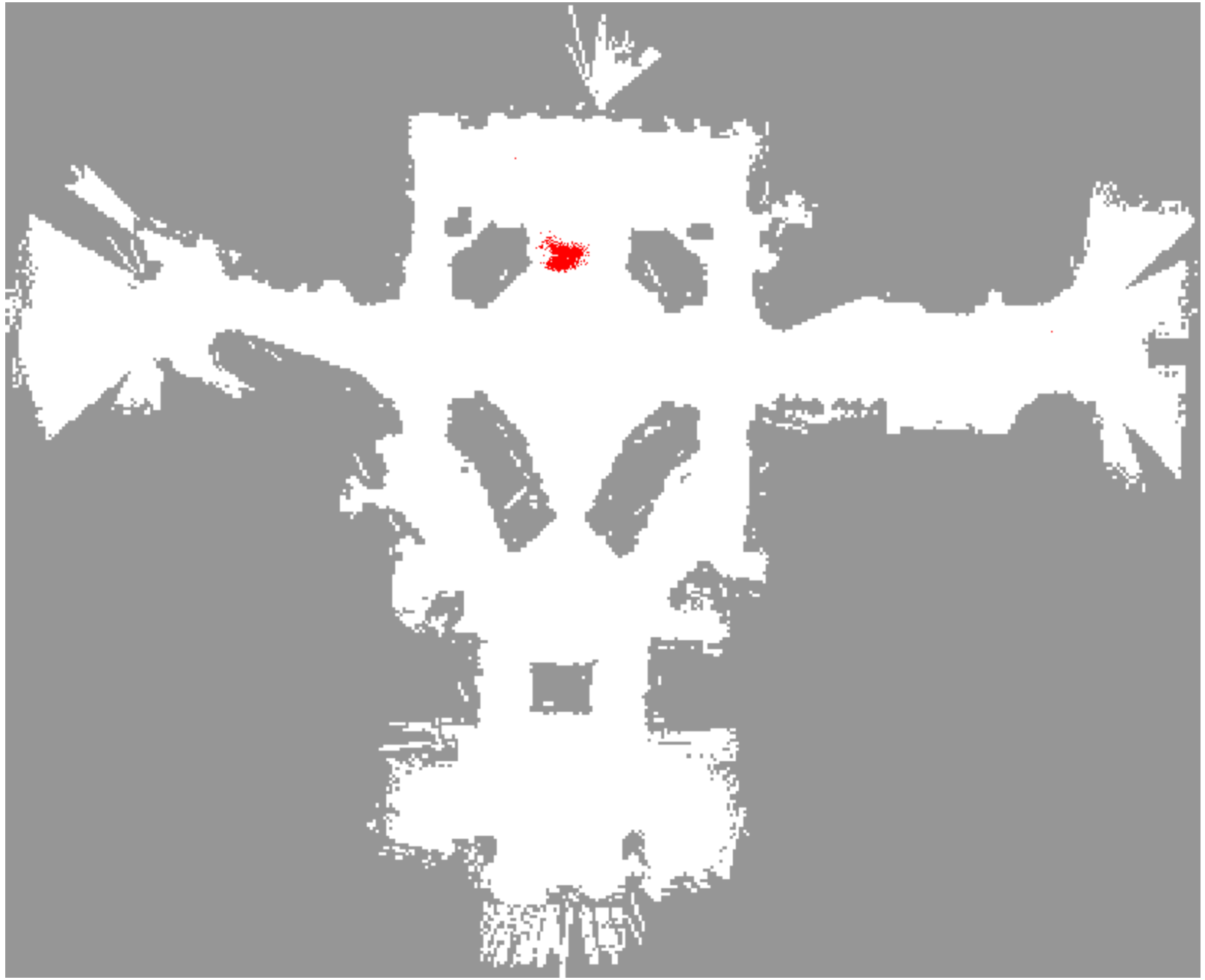


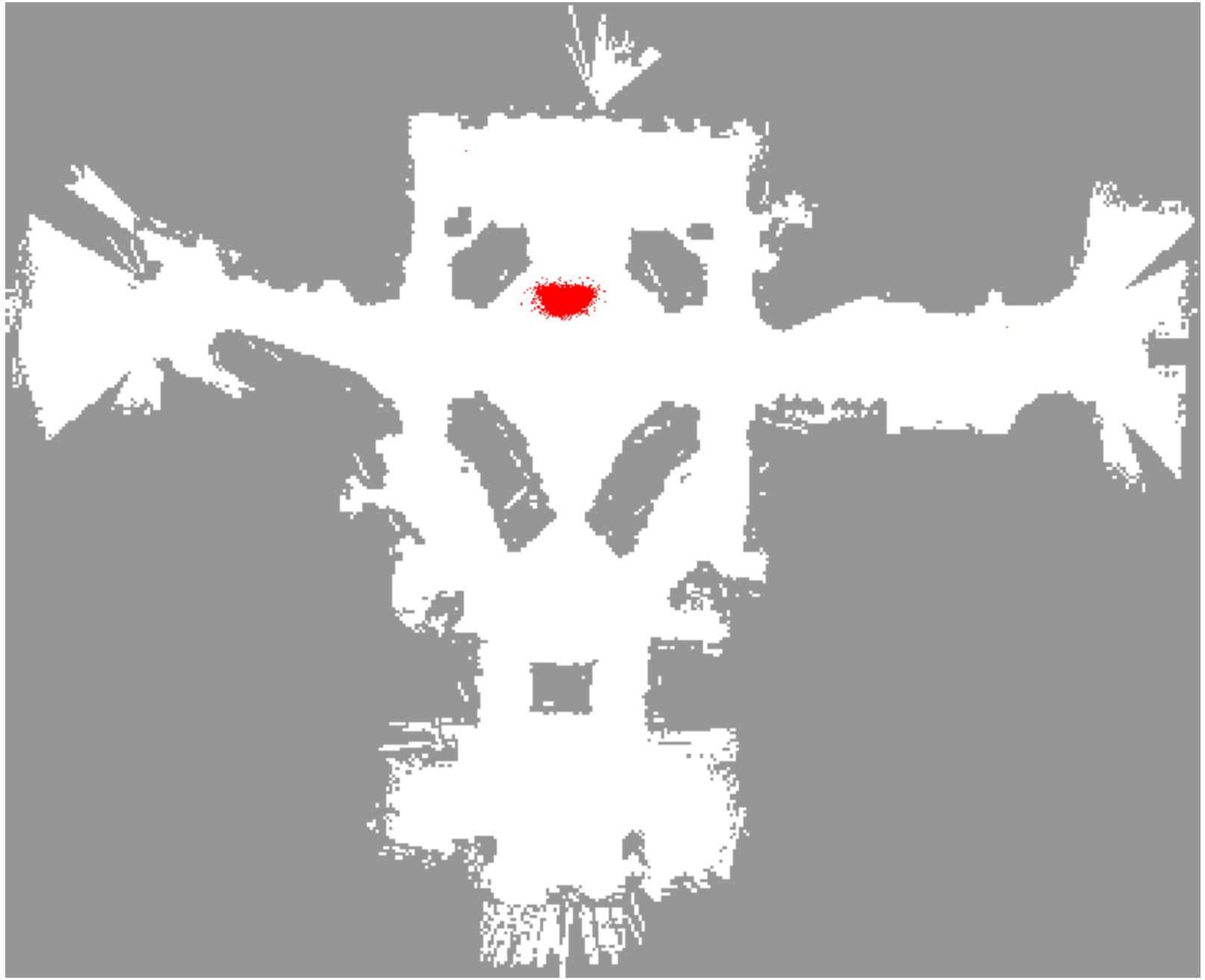


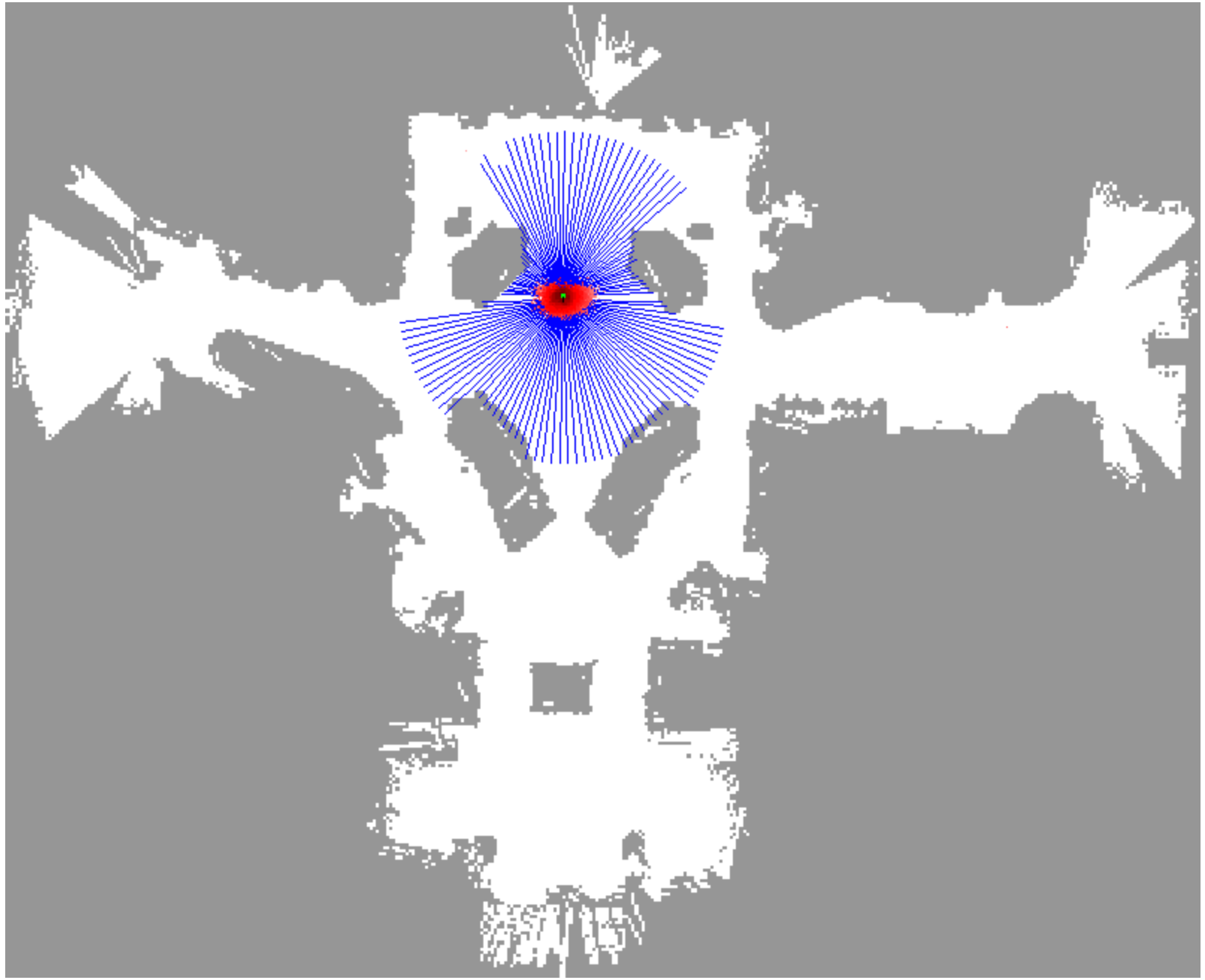


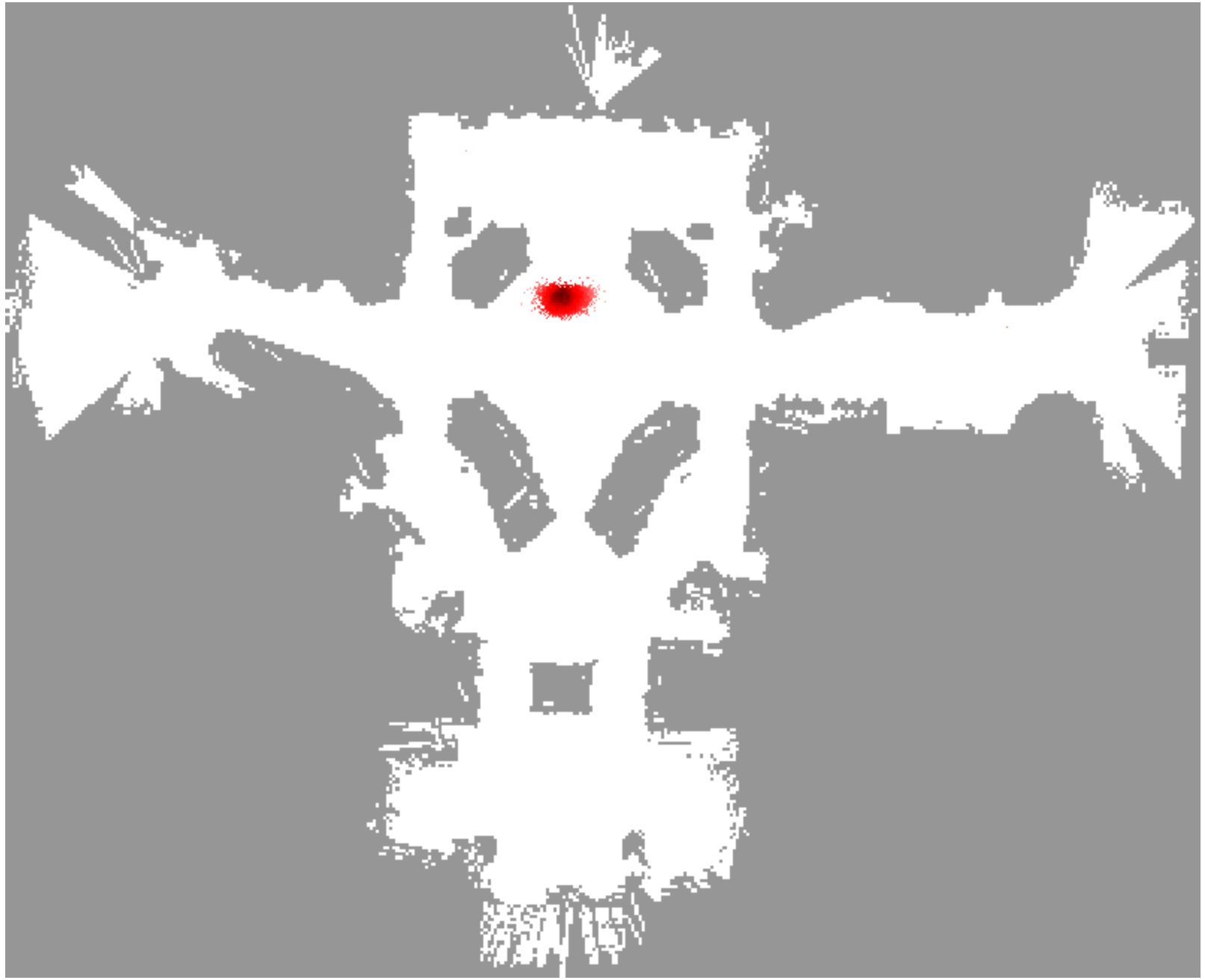


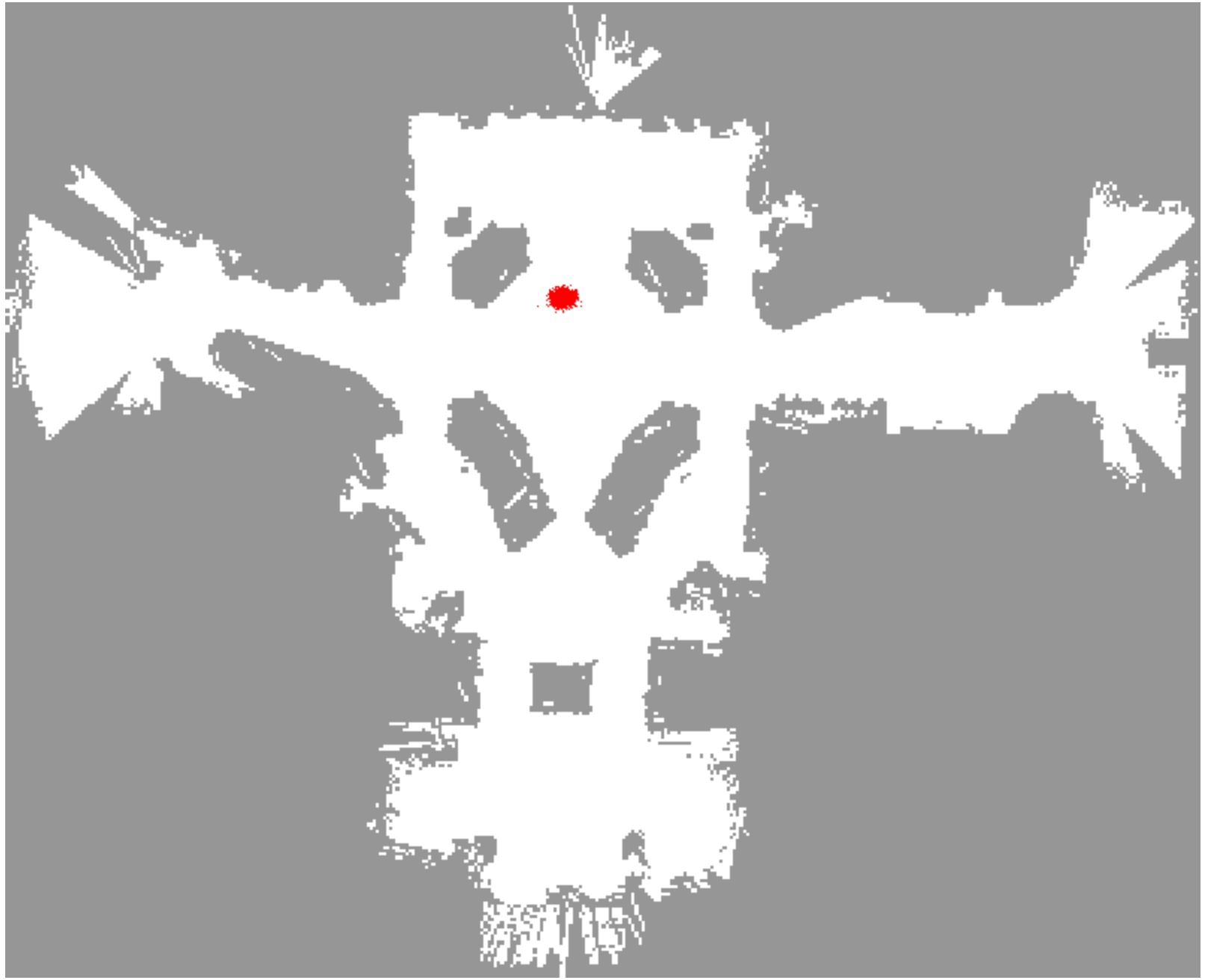


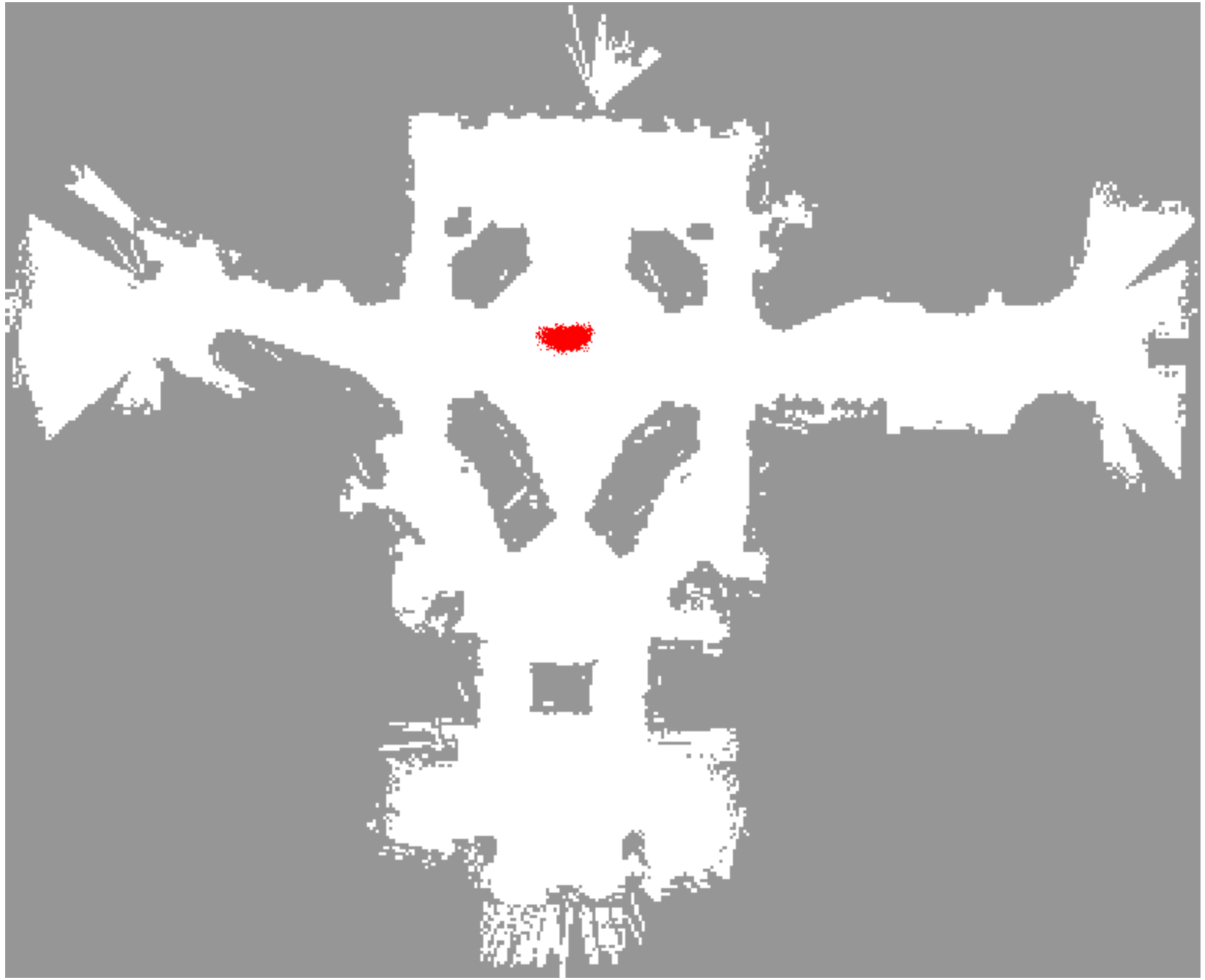


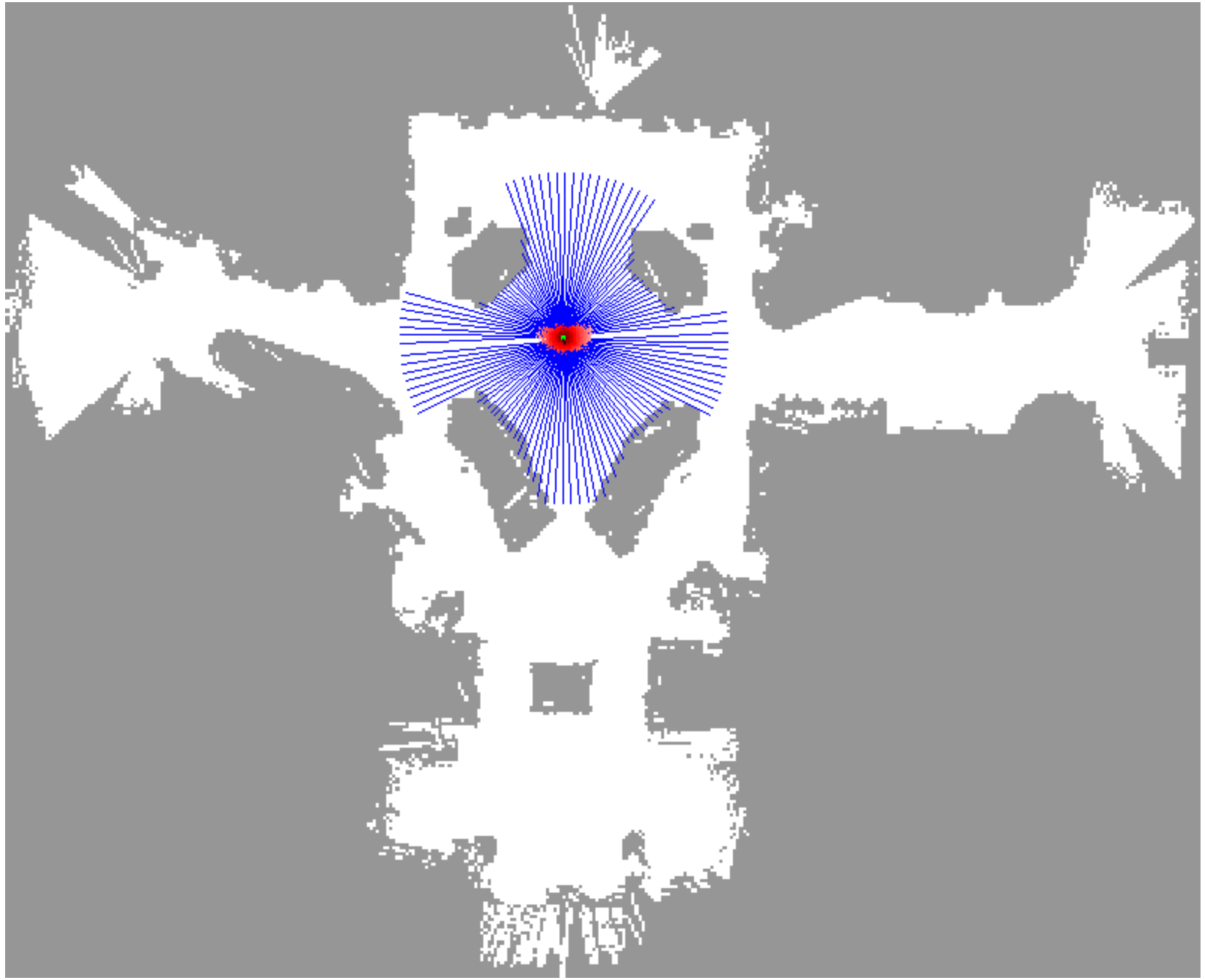


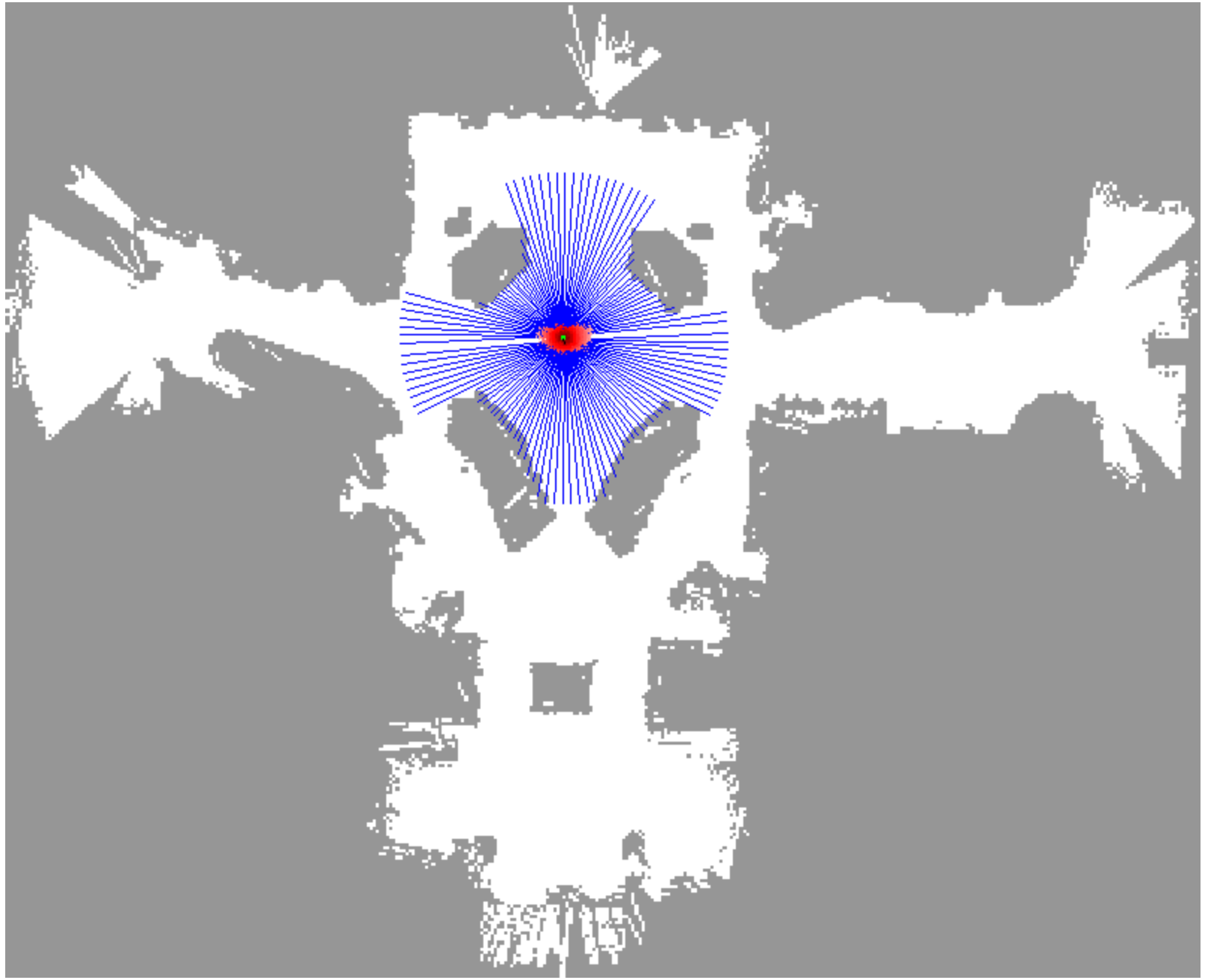


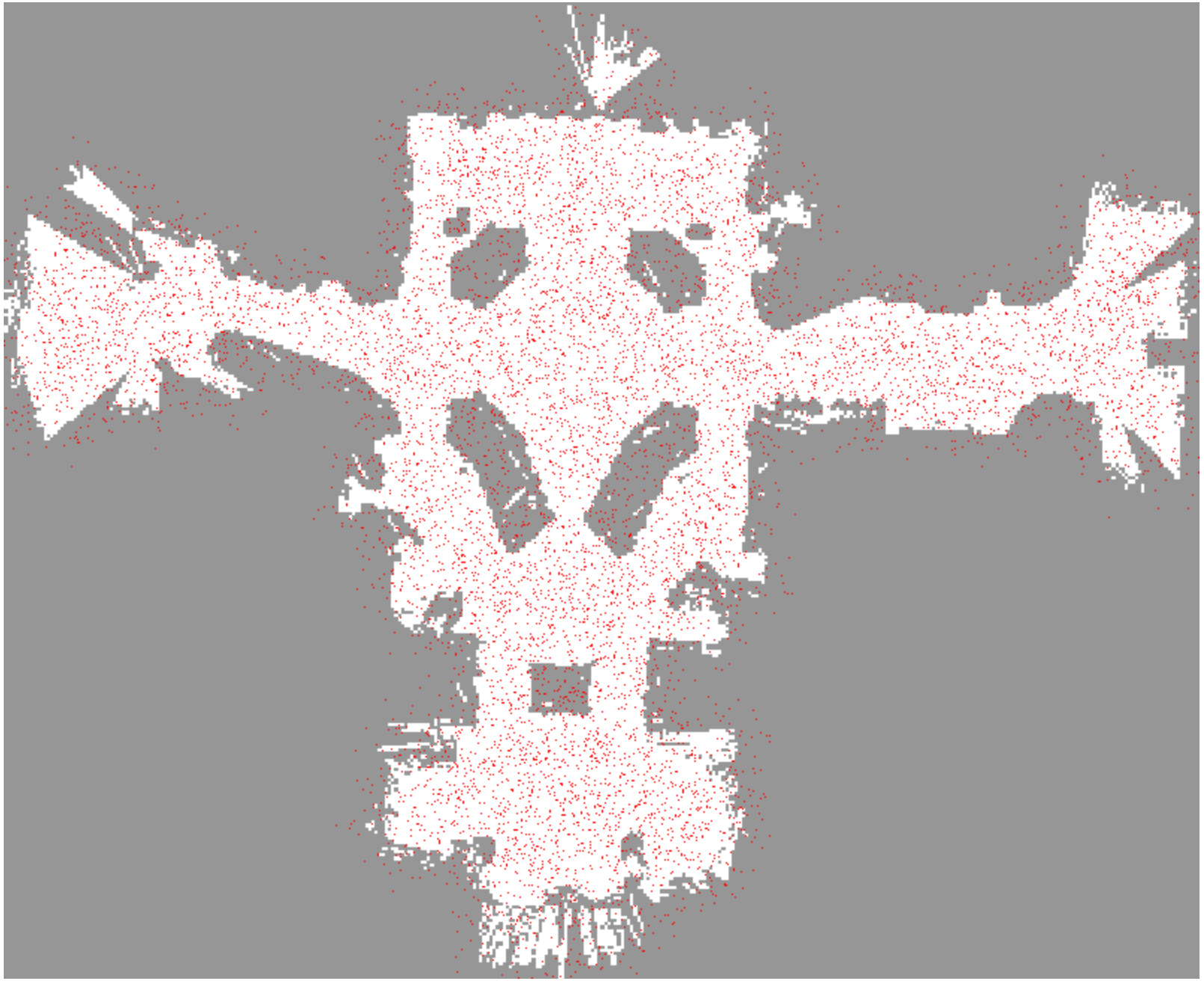




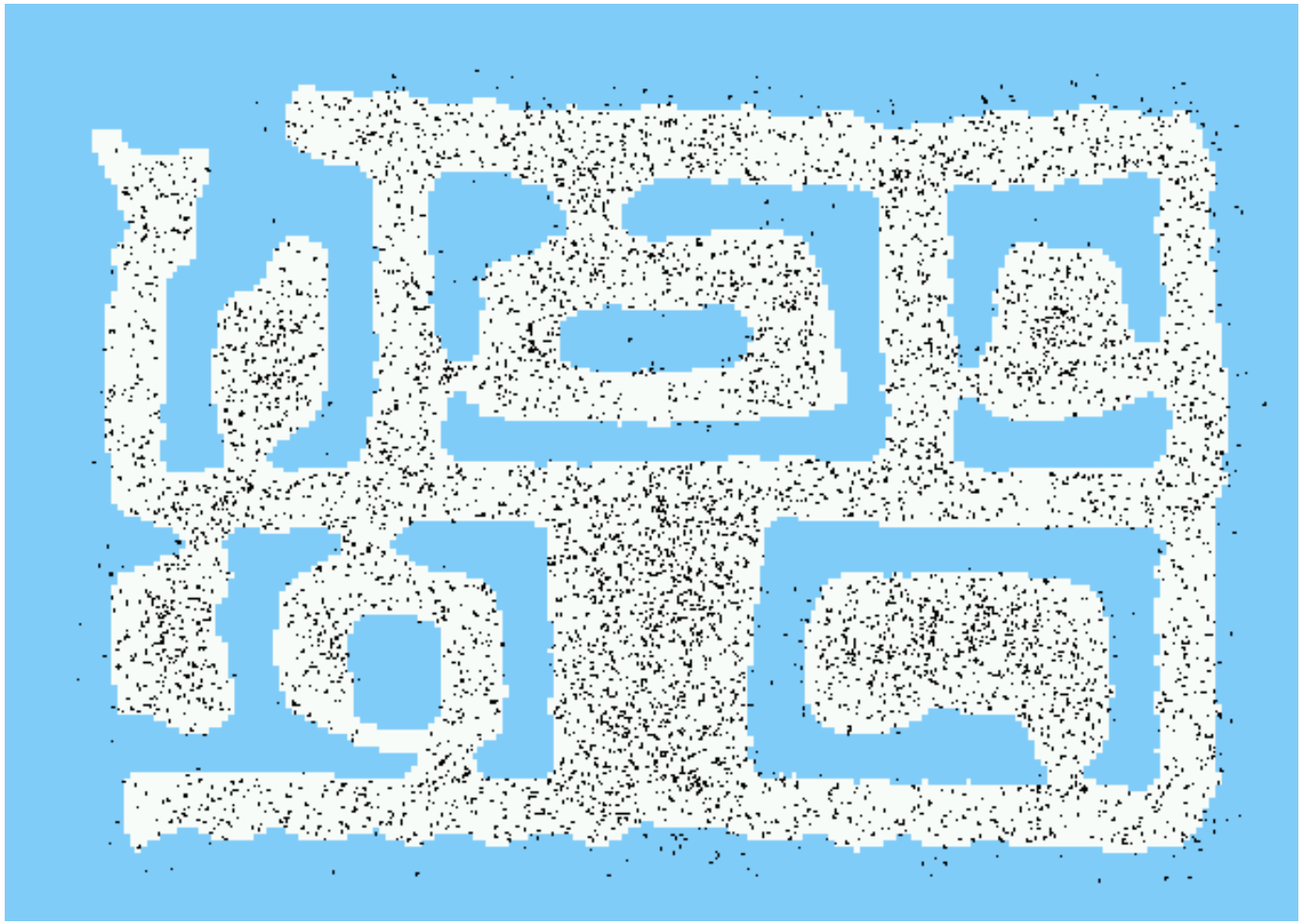




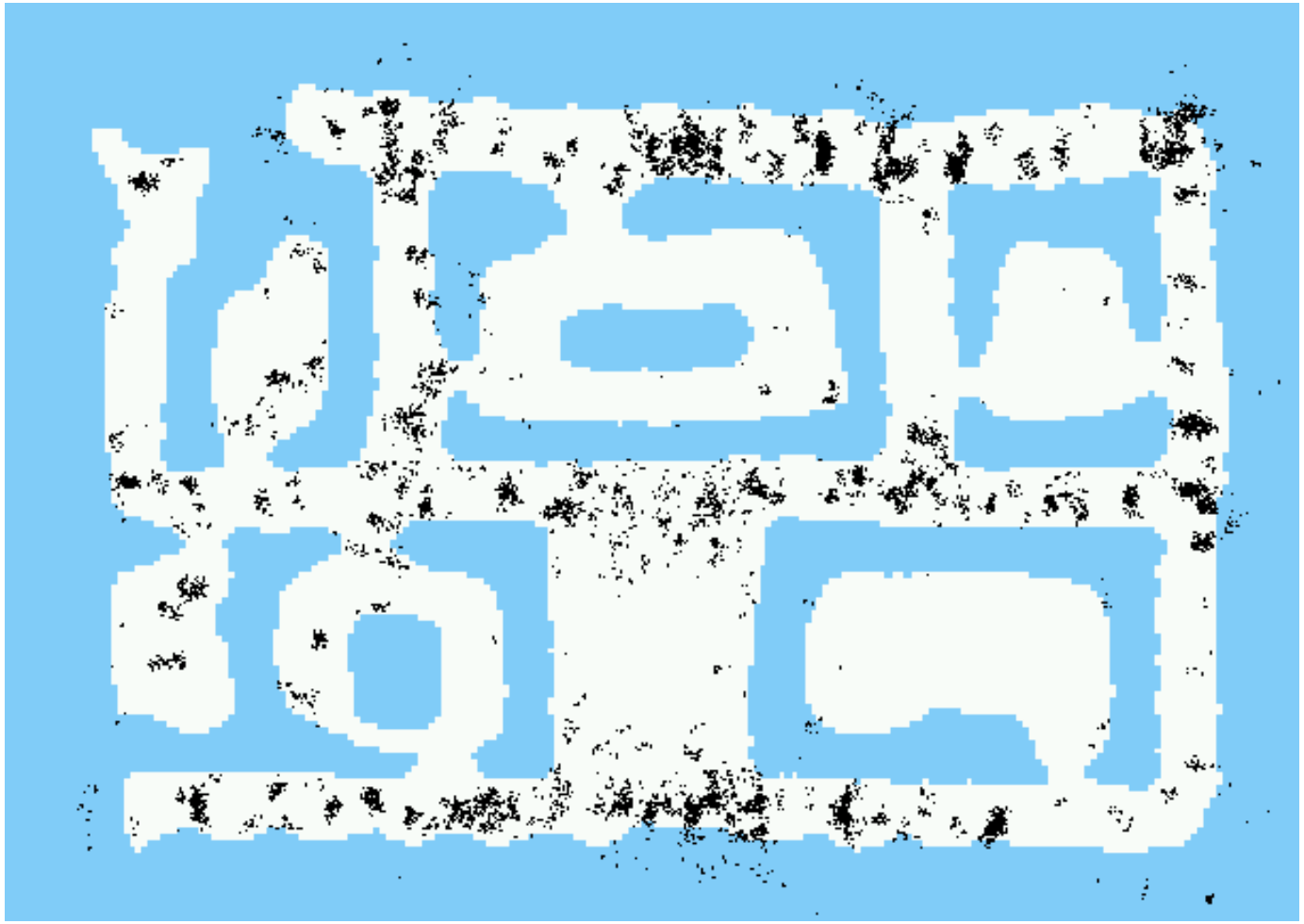




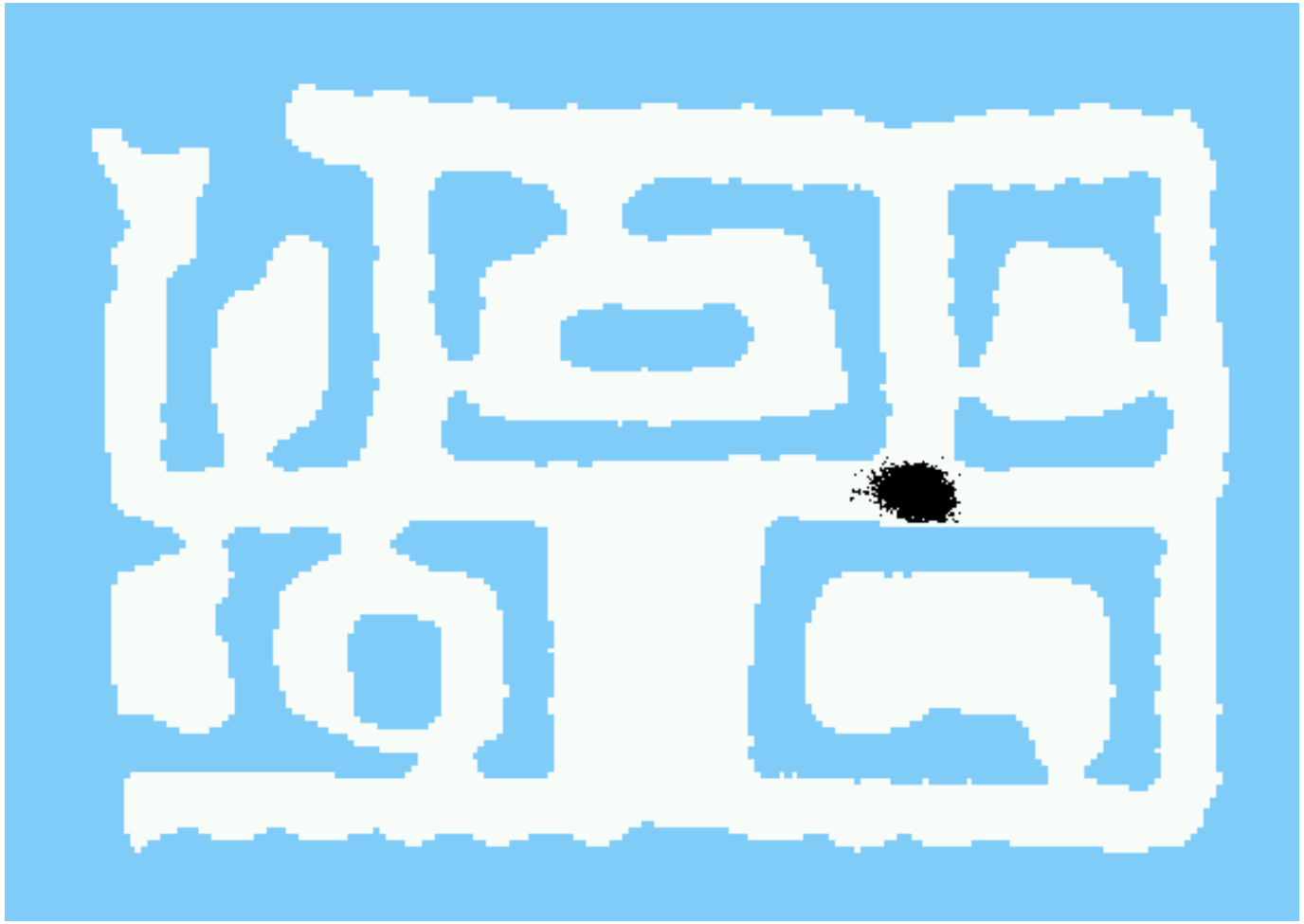
Initial Distribution



After Incorporating Ten Ultrasound Scans



After Incorporating 65 Ultrasound Scans



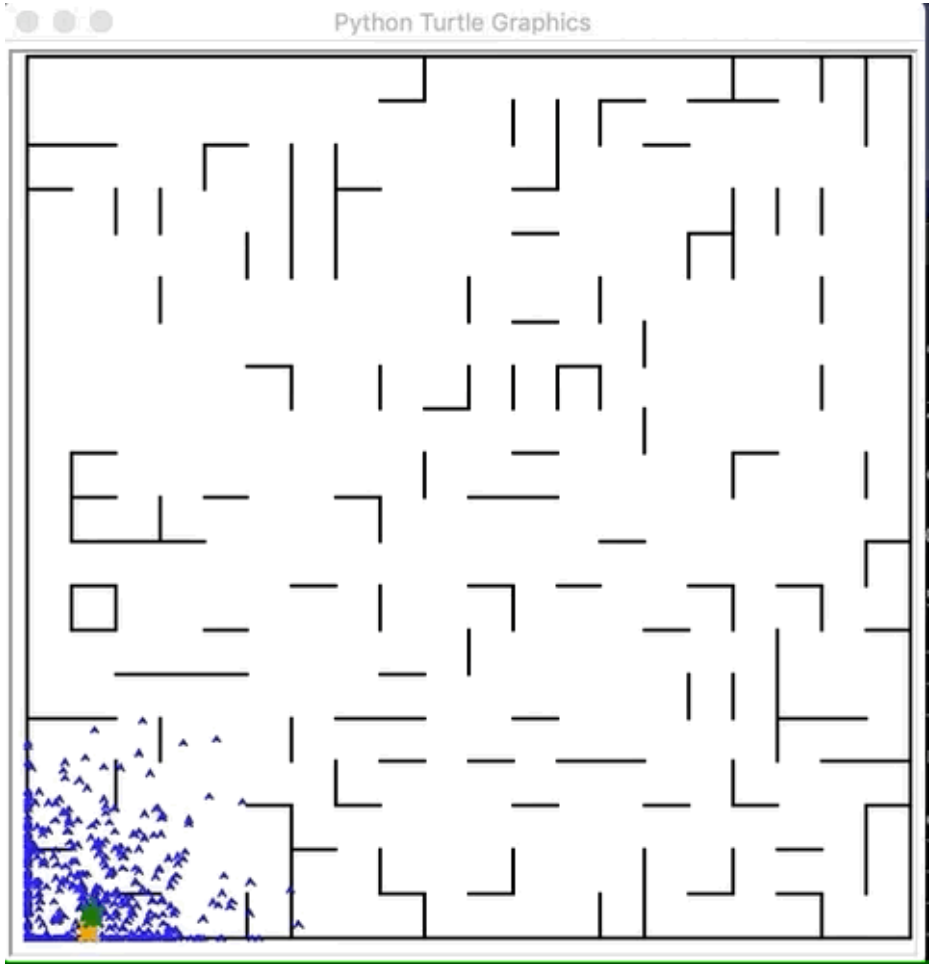
Part 1 Summary

- Particle filters are an implementation of recursive Bayesian filtering, where the posterior is represented by a set of weighted samples
- The particles are propagated according to the motion model and 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
- Limitations:
  - Some errors are particularly hard to deal with (e,g., the kidnapped robot problem)
  - The success of your filter is highly reliant on the number of particles → PF can be very memory intensive

Homework 3 and Lab 3:
Particle Filtering Algorithm // Monte Carlo Localization

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



Instead of a homework 3, you'll be given a skeleton to implement your own particle filter. This will be checked by your lab TA and used in Lab 3.

Kalman Filters

Bayes Filter Reminder

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|\, u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$$

- Prediction

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

- Correction

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

Bayes Filter Reminder

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|\, u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$$

- Prediction

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

- Correction

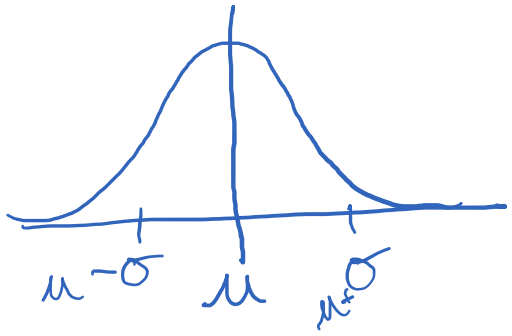
$$bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t)$$

What if we have a good model of our (continuous) system dynamics and we assume a Gaussian model for our uncertainty?

→ Kalman Filters!

Review of Gaussians

univariate: $p(x) \sim N(\mu, \sigma^2)$ | $p(x) = \frac{1}{2\pi\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$



multivariate: $p(x) \sim N(\mu, \Sigma)$ | $p(x) = \frac{1}{(2\pi)^{2d}|\Sigma|^{1/2}} e^{-1/2(x-\mu)^T\Sigma^{-1}(x-\mu)}$

$Y = AX + B$

$Y \sim N(A\mu + B, A\Sigma A^T)$

Linear Systems with Gaussian Noise

Suppose we have a system that is governed by a linear difference equation:

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

with measurement

$$z_t = C_t x_t + \delta_t$$

A_t	a $n \times n$ matrix that describes how the state evolves from $t-1$ to t without controls or noise
B_t	a $n \times m$ matrix that describes how the control u_t changes the state from $t-1$ to t
C_t	a $k \times n$ matrix that describes how to map the state x_t to an observation z_t
ϵ_t & δ_t	random variables representing the process and measurement noise, and are assumed to be independent and normally distributed with zero mean and covariance Q_t and R_t, respectively

What is a Kalman Filter?

Suppose we have a system that is governed by a linear difference equation:

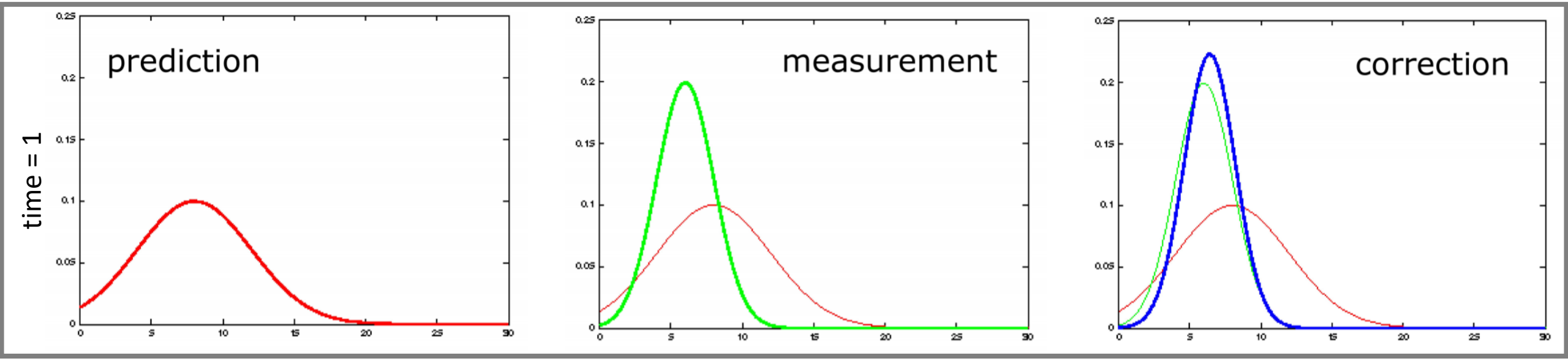
$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

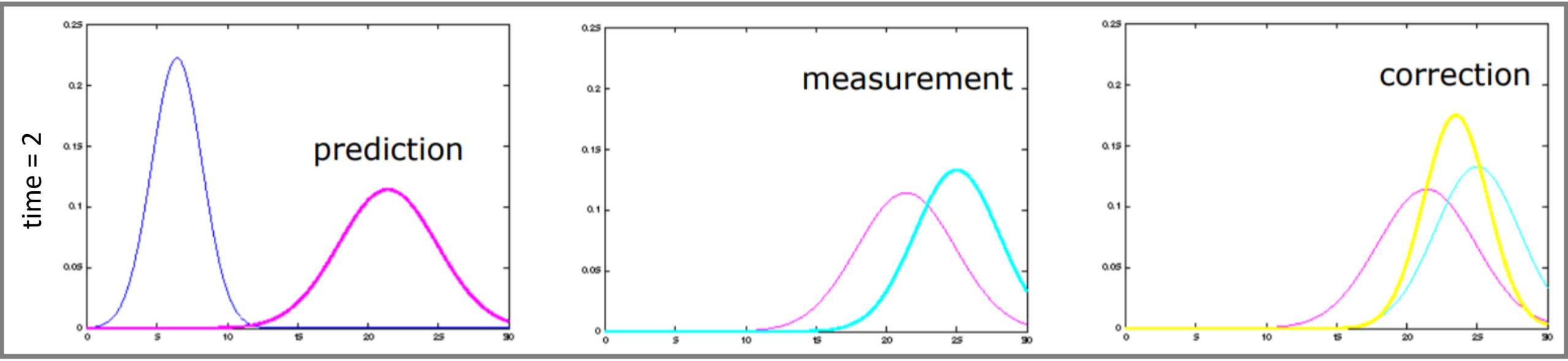
with measurement

$$z_t = C_t x_t + \delta_t$$

- Tracks the estimated state of the system by the mean and variance of its state variables -- minimum mean-square error estimator
- Computes the Kalman gain, which is used to weight the impact of new measurements on the system state estimate against the predicted value from the process model
- Note that we no longer have discrete states or measurements!

Kalman Filter Example





Linear Gaussian Systems: Dynamics

recall: $\varepsilon_t \sim N(0, Q_t)$

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

$$p(x_t \mid u_t, x_{t-1}) \leftarrow N(A_t x_{t-1} + B_t u_t, Q_t)$$

$$= N(\arg x_t; A_t x_{t-1} + B_t u_t, Q_t)$$

recall prediction step:

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) \, bel(x_{t-1}) \, d x_{t-1}$$

$$bel(x_{t-1}) \sim N(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})$$

... some manipulation ...

$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t \end{cases}$$

Linear Gaussian Systems: Observations

recall: $\delta_t \sim N(0, R_t)$

$$z_t = C_t x_t + \delta_t$$

$$p(z_t | x_t) \leftarrow N(z_t; C_t x_t, R_t)$$

recall correction step:

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

$$\overline{bel}(x_t) \rightarrow N(x_t; \bar{\mu}_t, \bar{\Sigma}_t)$$

$$\vdots$$

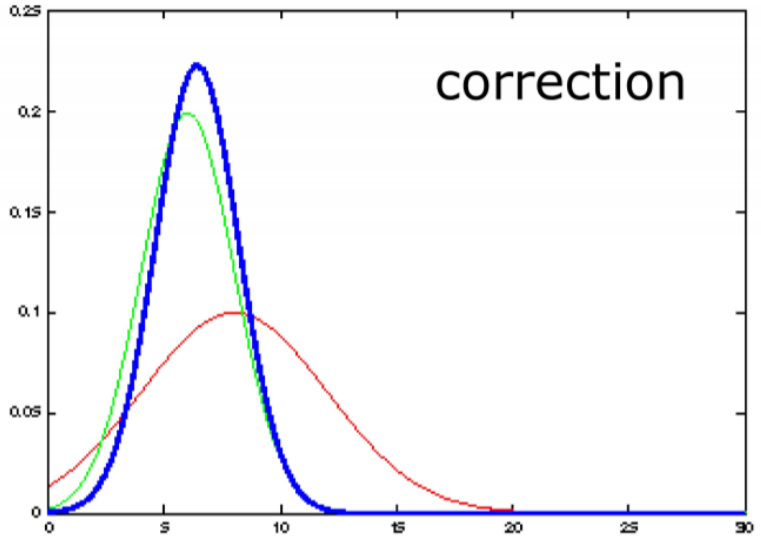
$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t)\bar{\Sigma}_t \end{cases}$$

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

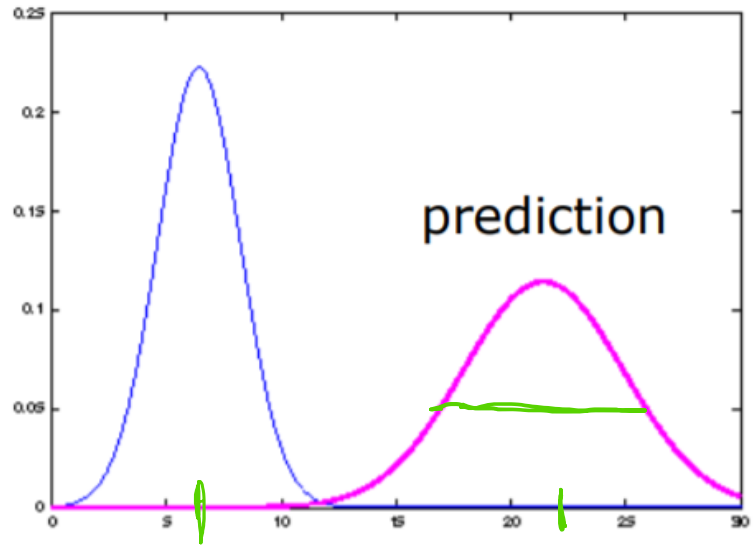
K_t: Kalman Gain

Kalman Filter Algorithm

1. Algorithm Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2. Prediction
   1. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
   2. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + Q_t$
3. Correction:
   1. $K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + R_t)^{-1}$
   2. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
   3. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
4. Return μ_t, Σ_t



Apply control action

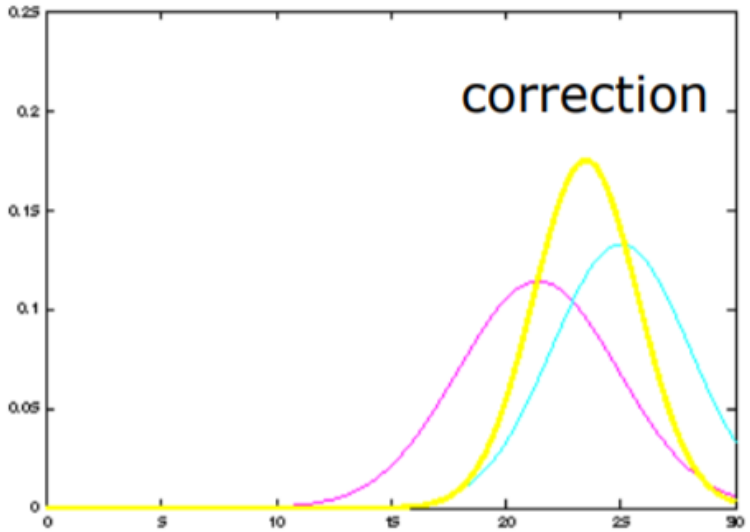


Correction:

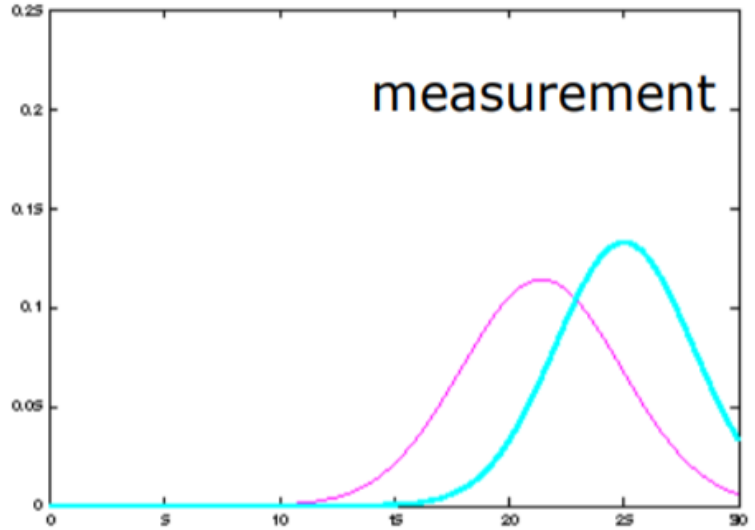
1. $K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + R_t)^{-1}$
2. $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$
3. $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$

Prediction

1. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^\top + Q_t$



Get sensor measurement



Part 2 Summary

- Kalman filters give you the **optimal estimate for linear Gaussian systems**
  - However, most systems aren't linear ☹
- Nonetheless, this filter (and its extensions) is highly efficient and widely used in practice

- For a nice 2D example, check out:
  - How a Kalman filter works, in pictures by Tim Babb