Particle Filtering

D.A. Forsyth

Non-linear dynamics are a problem

Many natural dynamic models are non-linear. There are two sources of problems. Firstly, in models where the dynamics have the form

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i); \Sigma_{d_i})$$

(where $\boldsymbol{f}$ is a non-linear function), both $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ and $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ tend not to be normal. As Section 1.1.1 will show, even quite innocuous looking nonlinearities can lead to very strange distributions indeed. Secondly, $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ may not be Gaussian either. This phenomenon is quite common in vision; it leads to difficulties that are still neither well understood nor easily dealt with (Section 1.1.2).

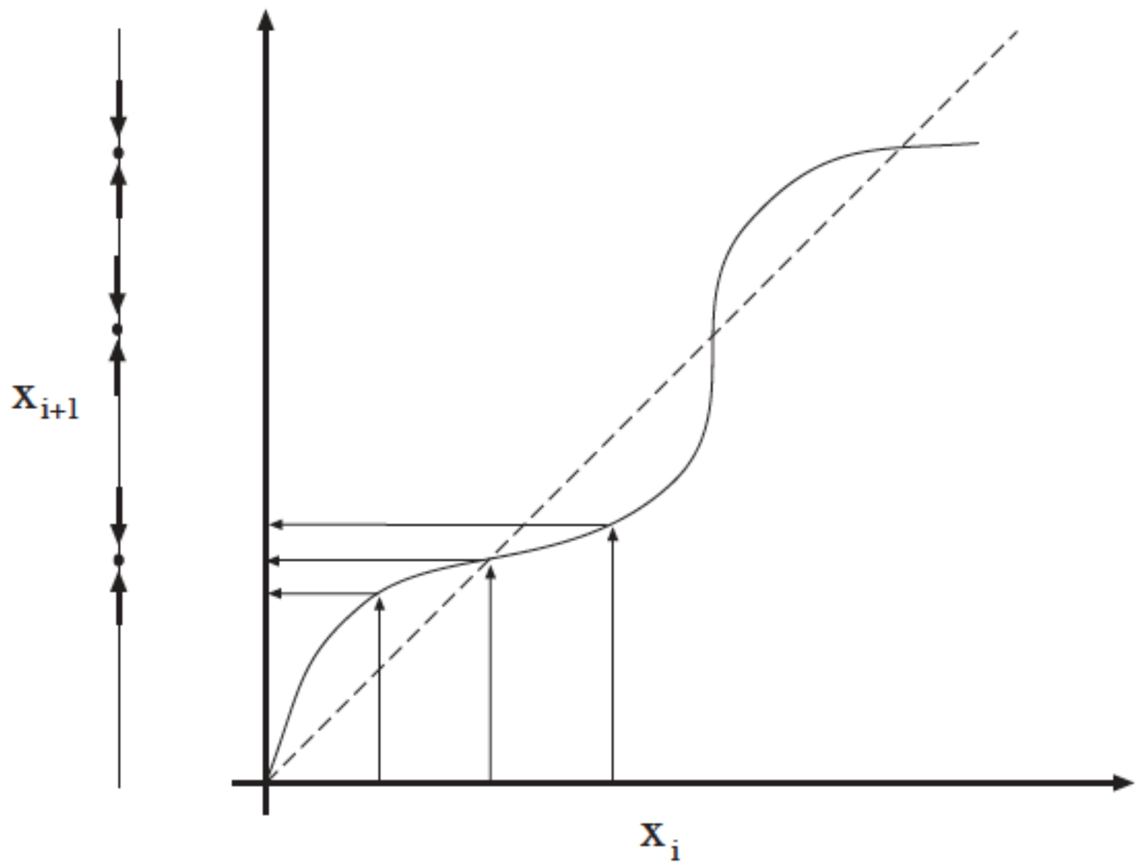


FIGURE 1.1: The non-linear dynamics $x_{i+1} = x_i + 0.1 \sin x_i$ cause points n the range $((2k)\pi, (2k+2)\pi)$ move towards $(2k+1)\pi$. As the figure on the left illustrates, this is because $x_i + 0.1 \sin x_i$ is slightly smaller than x_i for x_i in the range $((2k+1)\pi, (2k+2)\pi)$ and is slightly larger than x_i for x_i in the range $((2k)\pi, (2k+1)\pi)$. In fact, the nonlinearity of this function looks small — it is hardly visible in a scaled plot. However, as Figure 1.2 shows, its effects are very significant.

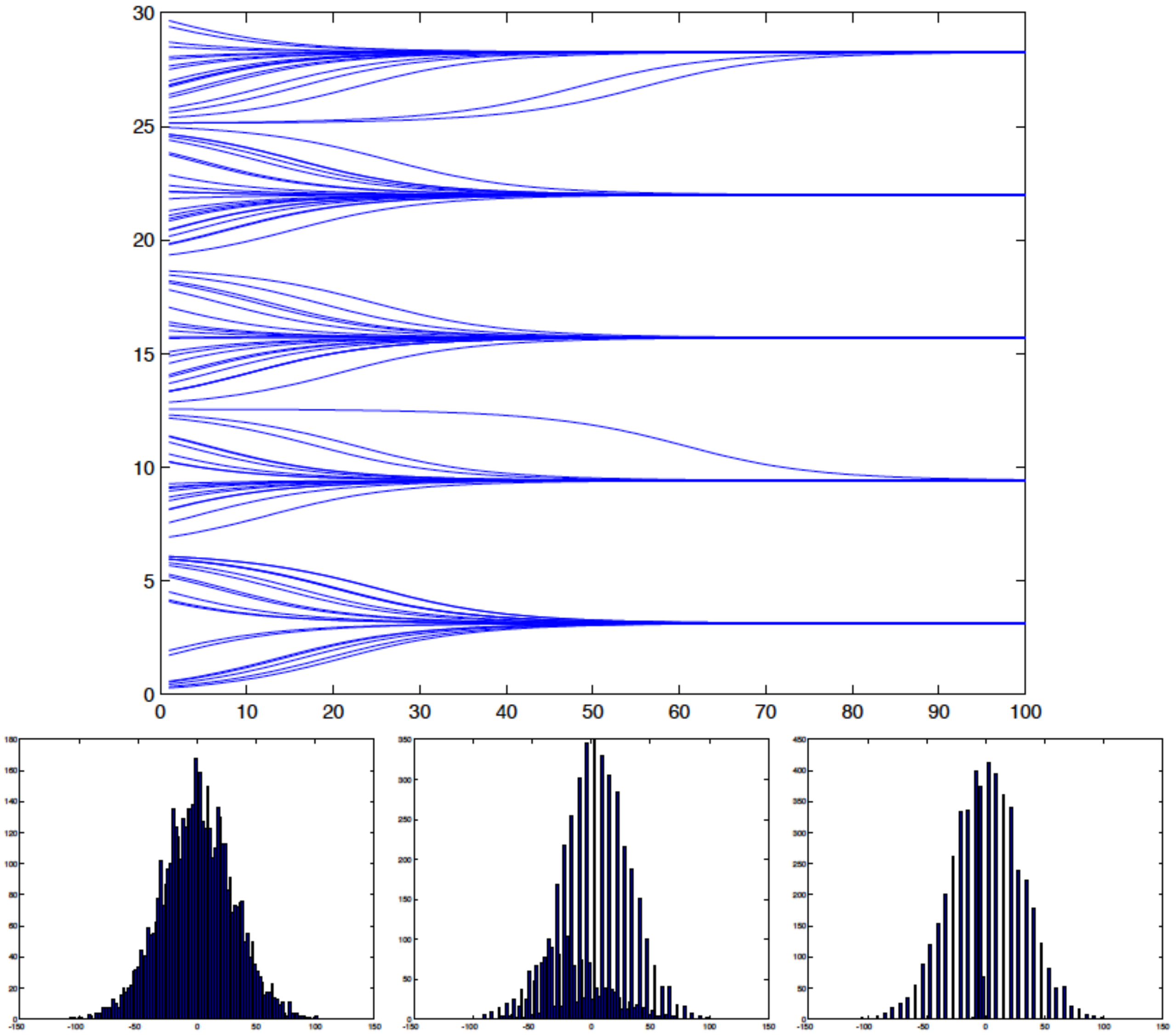


FIGURE 1.2: On the **top**, we have plotted the time evolution of the state of a set of 100 points, for 100 steps of the process $x_{i+1} = x_i + 0.1 * \sin x_i$. Notice that the points all contract rather quickly to $(2k+1)\pi$, and stay there. We have joined up the tracks of the points to make it clear how the state changes. On the **bottom left** we show a histogram of the start states of the points we used; this is an approximation to $P(x_0)$. The histogram on the **bottom center** shows a histogram of the point positions after 20 iterations; this is an approximation to $P(x_{20})$. The histogram on the **bottom right** shows a histogram of the point positions after 70 iterations; this is an approximation to $P(x_{70})$. Notice that there are many important peaks to this histogram — it might be very unwise to model $P(x_i)$ as a Gaussian.

Bad likelihoods

- In some problems, the likelihood has multiple peaks
  - different states produce about the same image
  - traditionally, 3D tracking of human body joint positions
- Even with linear dynamics, posterior has multiple peaks
- Issue:
  - the largest peak may not be the right state
    - and it could collapse in the future
  - you need to keep track of “many” peaks
    - and the number could grow
- Strategy
  - randomized search

Representing a probability distribution

- Parameters
  - gaussian +linear case: Kalman filter is just an easy maintenance process
    - there are a few others, but they're not important in practice

- Samples
  - Weak law of large numbers gives:

  - If

$$\mathbf{x}_i \sim P(\mathbf{x})$$

  - Then

$$\frac{1}{N}\sum h(\mathbf{x}_i) \approx \int h(\mathbf{x})P(\mathbf{x})d\mathbf{x}$$

It is better to weight samples

- The estimate is a random variable
  - mean is right
  - variance doesn’t depend on dimension
    - but can be very large, and depends on h
- Strategy:
  - weight the samples

Monte Carlo Integration using Importance Sampling Assume that we have a collection of N points $\boldsymbol{u}^i$, and a collection of weights w^i. These points are independent samples drawn from a probability distribution $S(\boldsymbol{U})$ — we call this the **sampling distribution**; notice that we have broken with our usual convention of writing any probability distribution with a P. We assume that $S(\boldsymbol{U})$ has a probability density function $s(\boldsymbol{U})$.

The weights have the form $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$ for some function f. Now it is a fact that

$$\begin{aligned} \mathrm{E}\left[\frac{1}{N}\sum_i g(\boldsymbol{u}^i)w^i\right] &= \int g(\boldsymbol{U})\frac{f(\boldsymbol{U})}{s(\boldsymbol{U})}s(\boldsymbol{U})d\boldsymbol{U} \\ &= \int g(\boldsymbol{U})f(\boldsymbol{U})d\boldsymbol{U} \end{aligned}$$

where the expectation is taken over the distribution on the collection of N independent samples from $S(\boldsymbol{U})$ (you can prove this fact using the weak law of large numbers). The variance of this estimate goes down as $1/N$, and is independent of the dimension of $\boldsymbol{U}$.

Representing Distributions using Weighted Samples If we think about a distribution as a device for computing expectations — which are integrals — we can obtain a representation of a distribution from the integration method described above. This representation will consist of a set of weighted points. Assume that f is non-negative, and $\int f(\boldsymbol{U})d\boldsymbol{U}$ exists and is finite. Then

$$\frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

is a probability density function representing the distribution of interest. We shall write this probability density function as $p_f(\boldsymbol{X})$.

Now we have a collection of N points $\boldsymbol{u}^i \sim S(\boldsymbol{U})$, and a collection of weights $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$. Using this notation, we have that

$$\begin{aligned} \mathrm{E}\left[\frac{1}{N}\sum_i w^i\right] &= \int 1\frac{f(\boldsymbol{U})}{s(\boldsymbol{U})}s(\boldsymbol{U})d\boldsymbol{U} \\ &= \int f(\boldsymbol{U})d\boldsymbol{U} \end{aligned}$$

Now this means that

$$\begin{aligned}
\mathrm{E}_{p_f}[g] &= \int g(\boldsymbol{U}) p_f(\boldsymbol{U}) d\boldsymbol{U} \\
&= \frac{\int g(\boldsymbol{U}) f(\boldsymbol{U}) d\boldsymbol{U}}{\int f(\boldsymbol{U}) d\boldsymbol{U}} \\
&= \mathrm{E}\left[\frac{\sum_i g(\boldsymbol{u}_i) w_i}{\sum_i w_i}\right] \\
&\approx \frac{\sum_i g(\boldsymbol{u}_i) w_i}{\sum_i w_i}
\end{aligned}$$

(where we have cancelled some N's). This means that we can *in principle* represent a probability distribution by a set of weighted samples (Algorithm 1). There are some significant practical issues here, however. Before we explore these, we will discuss how to perform various computations with sampled representations. We have already shown how to compute an expectation (above, and Algorithm 2). There are two other important activities for tracking: marginalisation, and turning a representation of a prior into a representation of a posterior.

Represent a probability distribution

$$p_f(\boldsymbol{X}) = \frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

by a set of N weighted samples

$$\{(\boldsymbol{u}^i, w^i)\}$$

where $\boldsymbol{u}^i \sim s(\boldsymbol{u})$ and $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$.

Algorithm 1: Obtaining a sampled representation of a probability distribution

We have a representation of a probability distribution

$$p_f(\boldsymbol{X}) = \frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

by a set of weighted samples

$$\{(\boldsymbol{u}^i, w^i)\}$$

where $\boldsymbol{u}^i \sim s(\boldsymbol{u})$ and $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$. Then:

$$\int g(\boldsymbol{U})p_f(\boldsymbol{U})d\boldsymbol{U} \approx \frac{\sum_{i=1}^{N} g(\boldsymbol{u}^i)w^i}{\sum_{i=1}^{N} w^i}$$

Algorithm 2: Computing an expectation using a set of samples

Marginalizing a Sampled Representation

An attraction of sampled representations is that some computations are particularly easy. Marginalisation is a good and useful example. Assume we have a sampled representation of $p_f(\boldsymbol{U}) = p_f((\boldsymbol{M}, \boldsymbol{N}))$. We write $\boldsymbol{U}$ as two components $(\boldsymbol{M}, \boldsymbol{N})$ so that we can marginalise with respect to one of them.

Now assume that the sampled representation consists of a set of samples which we can write as

$$\{((\boldsymbol{m}^i, \boldsymbol{n}^i), w^i)\}$$

In this representation, $(\boldsymbol{m}^i, \boldsymbol{n}^i) \sim s(\boldsymbol{M}, \boldsymbol{N})$ and $w^i = f((\boldsymbol{m}^i, \boldsymbol{n}^i))/s((\boldsymbol{m}^i, \boldsymbol{n}^i))$.

We want a representation of the marginal $p_f(\boldsymbol{M}) = \int p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N}$. We will use this marginal to estimate integrals, so we can derive the representation by thinking about integrals. In particular

$$\begin{aligned}
\int g(\boldsymbol{M}) p_f(\boldsymbol{M}) d\boldsymbol{M} &= \int g(\boldsymbol{M}) \int p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N} d\boldsymbol{M} \\
&= \int \int g(\boldsymbol{M}) p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N} d\boldsymbol{M} \\
&\approx \frac{\sum_{i=1}^{N} g(\boldsymbol{m}^i) w^i}{\sum_{i=1}^{N} w^i}
\end{aligned}$$

meaning that we can represent the marginal by dropping the $\boldsymbol{n}^i$ components of the sample (or ignoring them, which may be more efficient!).

Assume we have a sampled representation of a distribution

$$p_f(\boldsymbol{M}, \boldsymbol{N})$$

given by

$$\{((\boldsymbol{m}^i, \boldsymbol{n}^i), w^i)\}$$

Then

$$\{(\boldsymbol{m}^i, w^i)\}$$

is a representation of the marginal,

$$\int p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N}$$

Algorithm 3: Computing a representation of a marginal distribution

Transforming a Sampled Representation of a Prior into a Sampled Representation of a Posterior

Appropriate manipulation of the weights of a sampled distribution yields representations of other distributions. A particularly interesting case is representing a posterior, given some measurement. Recall that

$$\begin{aligned} p(\boldsymbol{U}|\boldsymbol{V}=v_0) &= \frac{p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})}{\int p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})d\boldsymbol{U}} \\ &= \frac{1}{K}p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U}) \end{aligned}$$

where $\boldsymbol{v}_0$ is some measured value taken by the random variable $\boldsymbol{V}$.

Assume we have a sampled representation of $p(\boldsymbol{U})$, given by $\{(\boldsymbol{u}^i, w^i)\}$. We can evaluate K fairly easily:

$$\begin{aligned} K &= \int p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})d\boldsymbol{U} \\ &= \mathrm{E}\left[\frac{\sum_{i=1}^N p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{u}^i)w^i}{\sum_{i=1}^N w^i}\right] \\ &\approx \frac{\sum_{i=1}^N p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{u}^i)w^i}{\sum_{i=1}^N w^i} \end{aligned}$$

Now let us consider the posterior.

$$\begin{aligned}\int g(\boldsymbol{U})p(\boldsymbol{U}|\boldsymbol{V}=v_0)d\boldsymbol{U} &= \frac{1}{K}\int g(\boldsymbol{U})p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{U})p(\boldsymbol{U})d\boldsymbol{U} \\ &\approx \frac{1}{K}\frac{\sum_{i=1}^{N} g(\boldsymbol{u}^i)p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{u}^i)w^i}{\sum_{i=1}^{N} w^i} \\ &\approx \frac{\sum_{i=1}^{N} g(\boldsymbol{u}^i)p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{u}^i)w^i}{\sum_{i=1}^{N} p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{u}^i)w^i}\end{aligned}$$

(where we substituted the approximate expression for K in the last step). This means that, if we take $\{(\boldsymbol{u}^i, w^i)\}$ and replace the weights with

$$w'^i = p(\boldsymbol{V}=\boldsymbol{v}_0|\boldsymbol{u}^i)w^i$$

the result $\{(\boldsymbol{u}^i, w'^i)\}$ is a representation of the posterior.

Assume we have a representation of $p(\boldsymbol{U})$ as

$$\{(\boldsymbol{u}^i, w^i)\}$$

Assume we have an observation $\boldsymbol{V} = \boldsymbol{v}_0$, and a likelihood model $p(\boldsymbol{V}|\boldsymbol{U})$. The posterior, $p(\boldsymbol{U}|\boldsymbol{V} = \boldsymbol{v}_0)$ is represented by

$$\{(\boldsymbol{u}^i, w'^i)\}$$

where

$$w'^i = p(\boldsymbol{V} = \boldsymbol{v}_0|\boldsymbol{u}^i)w^i$$

Algorithm 4: Transforming a sampled representation of a prior into a sampled representation of a posterior.

1.2.2 The Simplest Particle Filter

Assume that we have a sampled representation of $P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$, and we need to obtain a representation of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$. We will follow the usual two steps of prediction and correction.

We can regard each sample as a possible state for the process at step $\boldsymbol{X}_{i-1}$. We are going to obtain our representation by firstly representing

$$P(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

and then marginalising out $\boldsymbol{X}_{i-1}$ (which we know how to do). The result is the prior for the next state, and, since we know how to get posteriors from priors, we will obtain $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$.

Prediction Now

$$p(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) = p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1})p(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

Write our representation of $p(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as

$$\{(\boldsymbol{u}^k_{i-1}, w^k_{i-1})\}$$

(the superscripts index the samples for a given step i, and the subscript gives the step).

Now for any given sample $\boldsymbol{u}^k_{i-1}$, we can obtain samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}^k_{i-1})$ fairly easily. This is because our dynamic model is

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{x}_{i-1}) + \xi_i$$

where $\xi_i \sim N(0, \Sigma_{m_i})$. Thus, for any given sample $\boldsymbol{u}^k_{i-1}$, we can generate samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}^k_{i-1})$ as

$$\{(f(\boldsymbol{u}^k_{i-1}) + \xi^l_i, 1)\}$$

where $\xi^l_i \sim N(0, \Sigma_{m_i})$. The index l indicates that we might generate several such samples for each $\boldsymbol{u}^k_{i-1}$.

We can now represent $p(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as

$$\{((f(\boldsymbol{u}^k_{i-1}) + \xi^l_i, \boldsymbol{u}^k_{i-1}), w^k_{i-1})\}$$

(notice that there are *two* free indexes here, k and l; by this we mean that, for each sample indexed by k, there might be several different elements of the set, indexed by l).

Because we can marginalise by dropping elements, the representation of

$$P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

is given by

$$\{(f(\boldsymbol{u}_{i-1}^k) + \xi_i^l, w_{i-1}^k)\}$$

(we walk through a proof in the exercises). We will reindex this collection of samples — which may have more than N elements — and rewrite it as

$$\left\{(\boldsymbol{u}_i^{k,-}, w_i^{k,-})\right\}$$

assuming that there are M elements. Just as in our discussion of Kalman filters, the superscript '$-$' indicates that this our representation of the i'th state before a measurement has arrived. The superscript k gives the individual sample.

Correction Correction is simple: we need to take the prediction, which acts as a prior, and turn it into a posterior. We do this by choosing an appropriate weight for each sample, following Algorithm 4. The weight is

$$p(\boldsymbol{Y}_i = \boldsymbol{y}_i | \boldsymbol{X}_i = \boldsymbol{s}_i^{k,-}) w_i^{k,-}$$

(you should confirm this by comparing with Algorithm 4). and our representation of the posterior is

$$\left\{ (\boldsymbol{s}_i^{k,-}, p(\boldsymbol{Y}_i = \boldsymbol{y}_i | \boldsymbol{X}_i = \boldsymbol{s}_i^{k,-}) w_i^{k,-}) \right\}$$

The Tracking Algorithm In principle, we now have most of a tracking algorithm — the only missing step is to explain where the samples of $p(\boldsymbol{X}_0)$ came from. The easiest thing to do here is to start with a diffuse prior of a special form that is easily sampled — a Gaussian with large covariance might do it — and give each of these samples a weight of 1. It is a good idea to implement this tracking algorithm to see how it works (exercises!); you will notice that it works poorly even on the simplest problems (Figure 1.3 compares estimates from this algorithm to exact expectations computed with a Kalman filter). The algorithm gives bad estimates because most samples represent no more than wasted computation. In jargon, the samples are called **particles**.

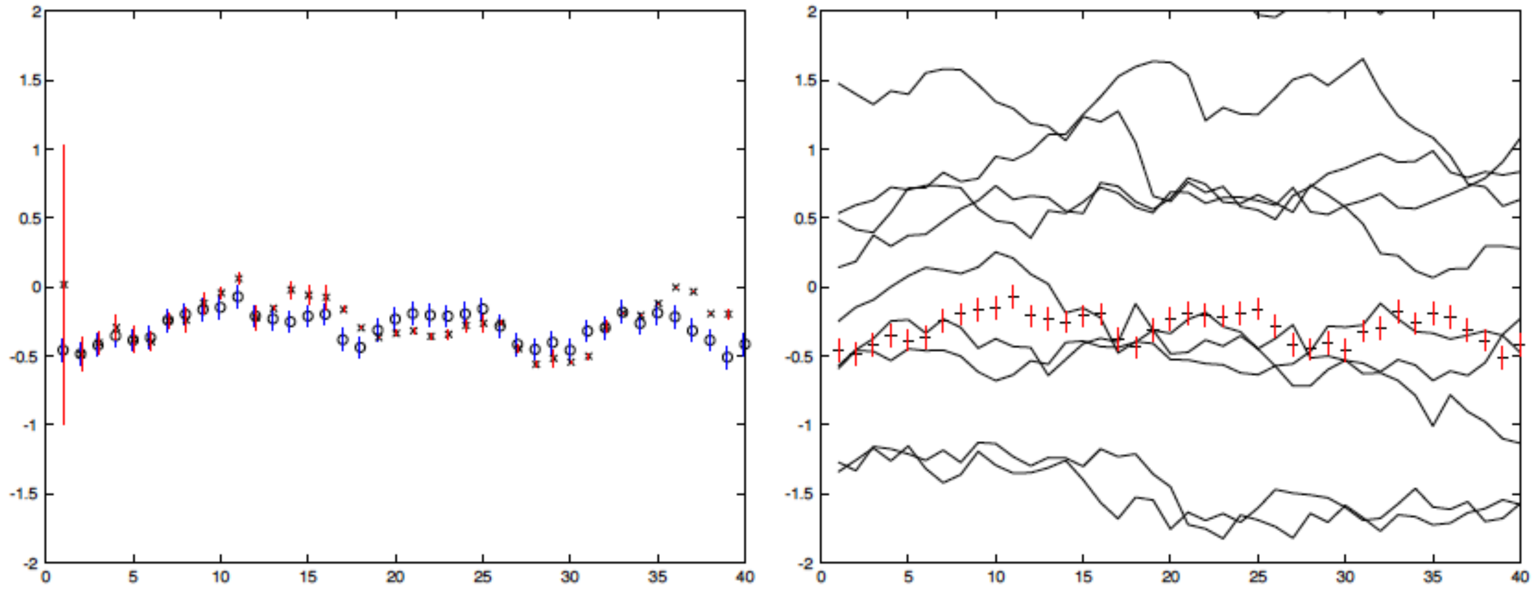
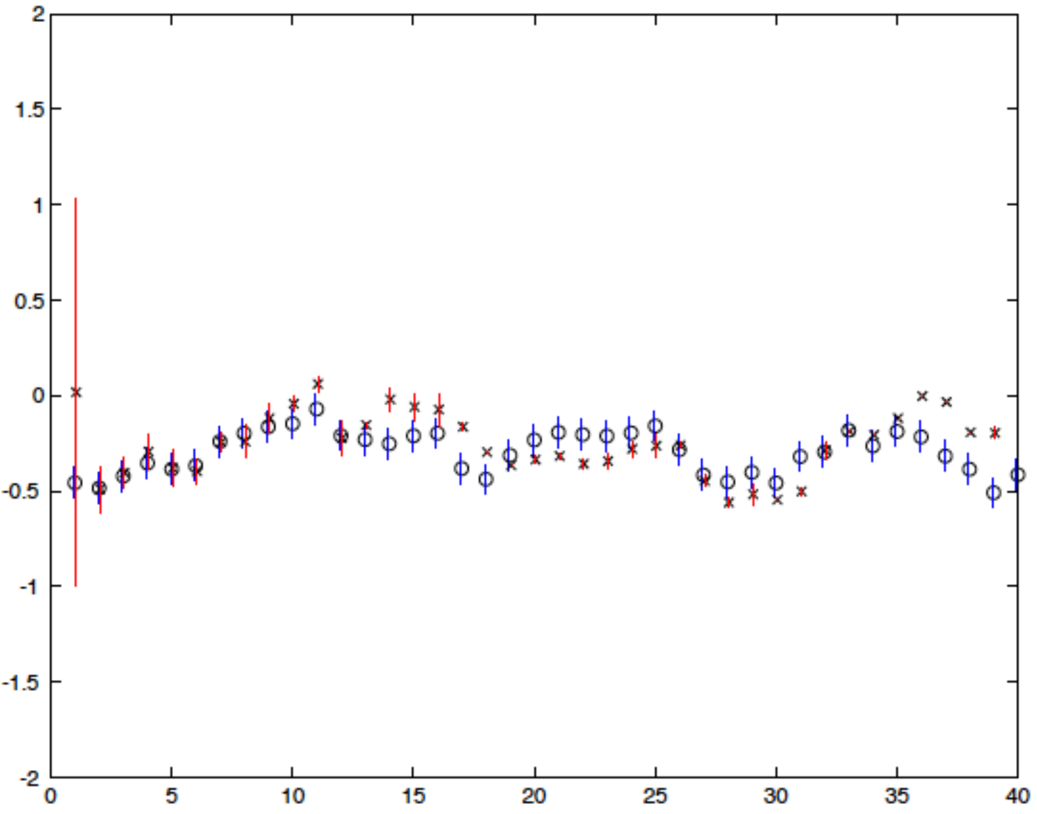
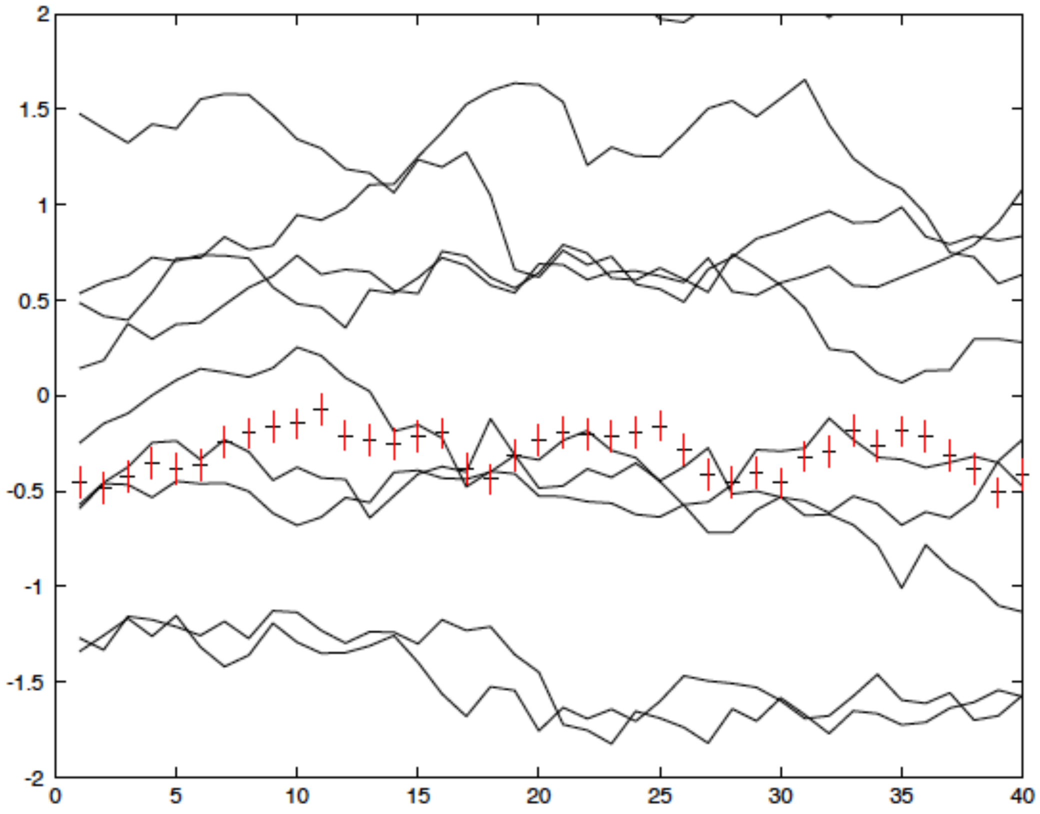


FIGURE 1.3: The simple particle filter behaves very poorly, as a result of a phenomenon called *sample impoverishment*, which is rather like quantisation error. In this example, we have a point on the line drifting on the line (i.e., $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise. In this case, we can get an exact representation of the posterior using a Kalman filter. In the figure on the **left**, we compare a representation obtained exactly using a Kalman filter with one computed from simple particle filtering. We show the mean of the posterior as a point with a *one* standard deviation bar (previously we used three standard deviations, but that would make these figures difficult to interpret). The mean obtained using a Kalman filter is given as an x; the mean obtained using a particle filter is given as an o; we have offset the standard deviation bars from one another so as to make the phenomenon clear. Notice that the mean is poor, but the standard deviation estimate is awful, and gets worse as the tracking proceeds. In particular, the standard deviation estimate woefully underestimates the standard deviation — this could mislead a user into thinking the tracker was working and producing good estimates, when in fact it is hopelessly confused. The figure on the **right** indicates what is going wrong; we plot the tracks of ten particles, randomly selected from the 100 used. Note that relatively few particles ever lie within one standard deviation of the mean of the posterior; in turn, this means that our representation of $P(x_{i+1}|y_0, \ldots, y_0)$ will tend to consist of many particles with very low weight, and only one with a high weight. This means that the density is represented very poorly, and the error propagates.


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
35
40


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
35
40

Resampling the Prior At each step i, we have a representation of

$$P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

via weighted samples. This representation consists of N (possibly distinct) samples, each with an associated weight. Now in a sampled representation, the frequency with which samples appear can be traded off against the weight with which they appear. For example, assume we have a sampled representation of $P(\boldsymbol{U})$ consisting of N pairs $(\boldsymbol{s}_k, w_k)$. Form a new set of samples consisting of a union of N_k copies of $(\boldsymbol{s}_k, 1)$, for each k. If

$$\frac{N_k}{\sum_k N_k} = w_k$$

this new set of samples is also a representation of $P(\boldsymbol{U})$ (you should check this).

Furthermore, if we take a sampled representation of $P(\boldsymbol{U})$ using N samples, and draw N' elements from this set with replacement, uniformly and at random, the result will be a representation of $P(\boldsymbol{U})$, too (you should check this, too). This suggests that we could (a) expand the sample set and then (b) subsample it to get a new representation of $P(\boldsymbol{U})$. This representation will tend to contain multiple copies of samples that appeared with high weights in the original representation.

This procedure is equivalent to the rather simpler process of making N draws with replacement from the original set of samples, using the weights w_i as the probability of drawing a sample. Each sample in the new set would have weight 1; the new set would predominantly contain samples that appeared in the old set with large weights. This process of resampling might occur at every frame, or only when the variance of the weights is too high.

Initialization: Represent $P(\boldsymbol{X}_0)$ by a set of N samples

$$\left\{(\boldsymbol{s}_0^{k,-}, w_0^{k,-})\right\}$$

where

$$\boldsymbol{s}_0^{k,-} \sim P_s(\boldsymbol{S}) \text{ and } w_0^{k,-} = P(\boldsymbol{s}_0^{k,-})/P_s(\boldsymbol{S} = \boldsymbol{s}_0^{k,-})$$

Ideally, $P(\boldsymbol{X}_0)$ has a simple form and $\boldsymbol{s}_0^{k,-} \sim P(\boldsymbol{X}_0)$ and $w_0^{k,-} = 1$.

Prediction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_{i-1})$ by

$$\left\{(\boldsymbol{s}_i^{k,-}, w_i^{k,-})\right\}$$

where

$$\boldsymbol{s}_i^{k,-} = f(\boldsymbol{s}_{i-1}^{k,+}) + \xi_i^k \text{ and } w_i^{k,-} = w_{i-1}^{k,+} \text{ and } \xi_i^k \sim N(0, \Sigma_{d_i})$$

Correction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_i)$ by

$$\left\{(\boldsymbol{s}_i^{k,+}, w_i^{k,+})\right\}$$

where

$$\boldsymbol{s}_i^{k,+} = \boldsymbol{s}_i^{k,-} \text{ and } w_i^{k,+} = P(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-})w_i^{k,-}$$

Resampling: Normalise the weights so that $\sum_i w_i^{k,+} = 1$ and compute the variance of the normalised weights. If this variance exceeds some threshold, then construct a new set of samples by drawing, with replacement, N samples from the old set, using the weights as the probability that a sample will be drawn. The weight of each sample is now $1/N$.

Algorithm 5: A practical particle filter resamples the posterior.

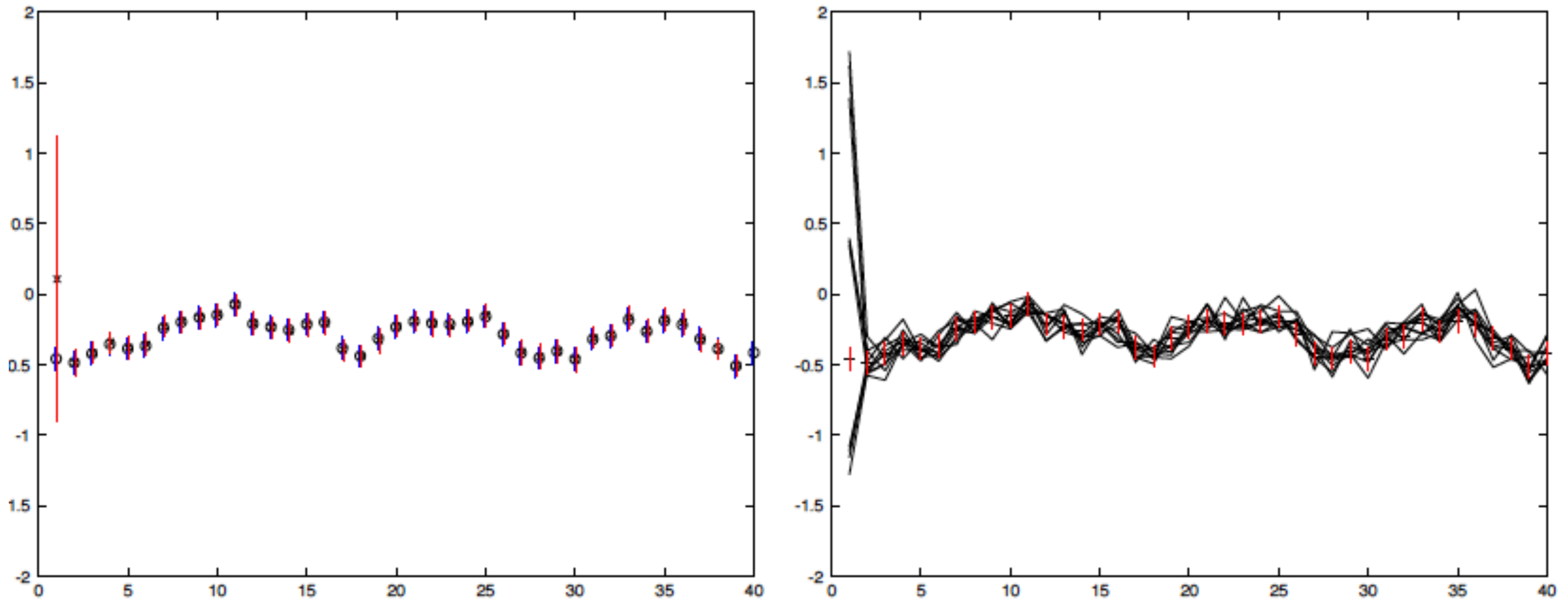
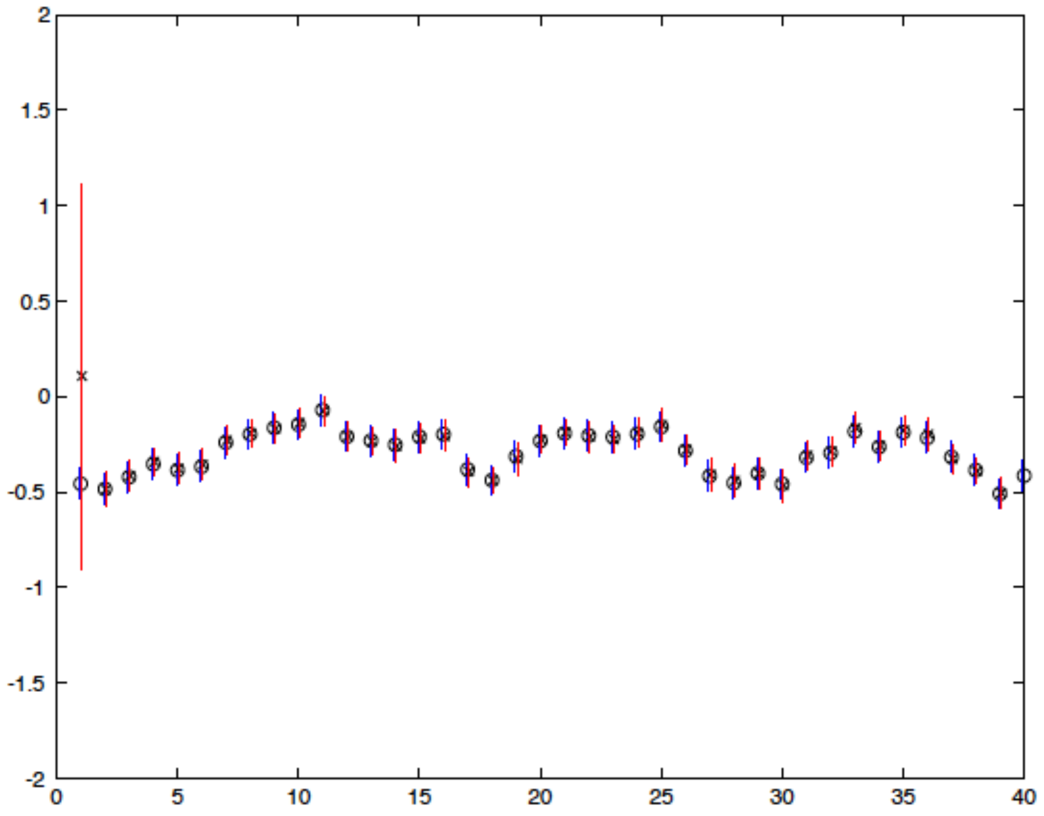
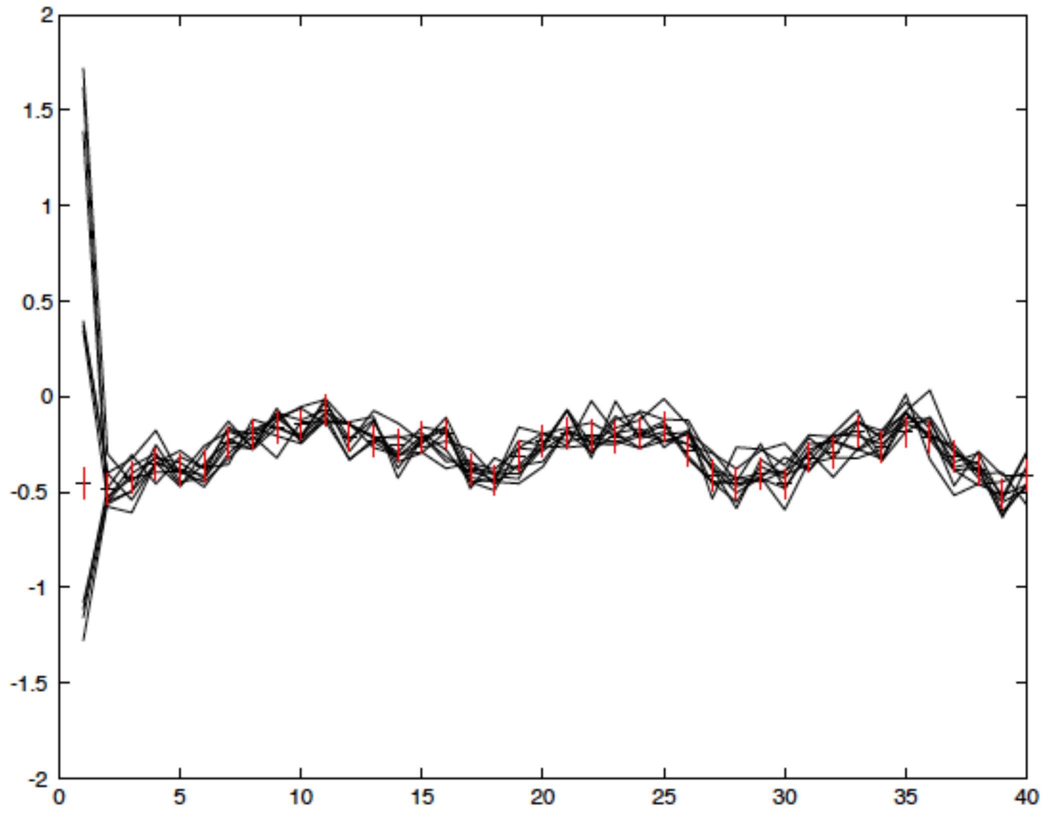
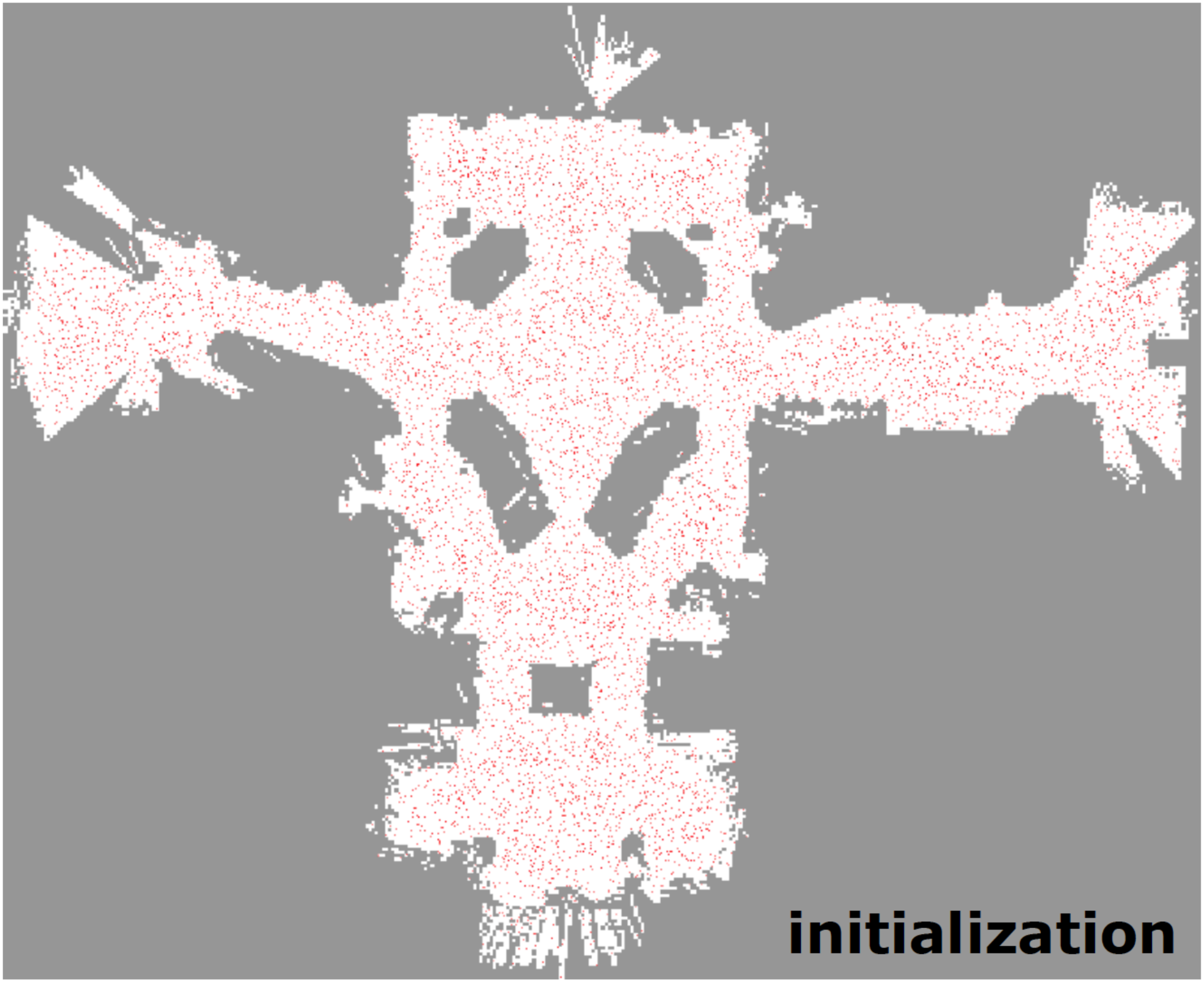
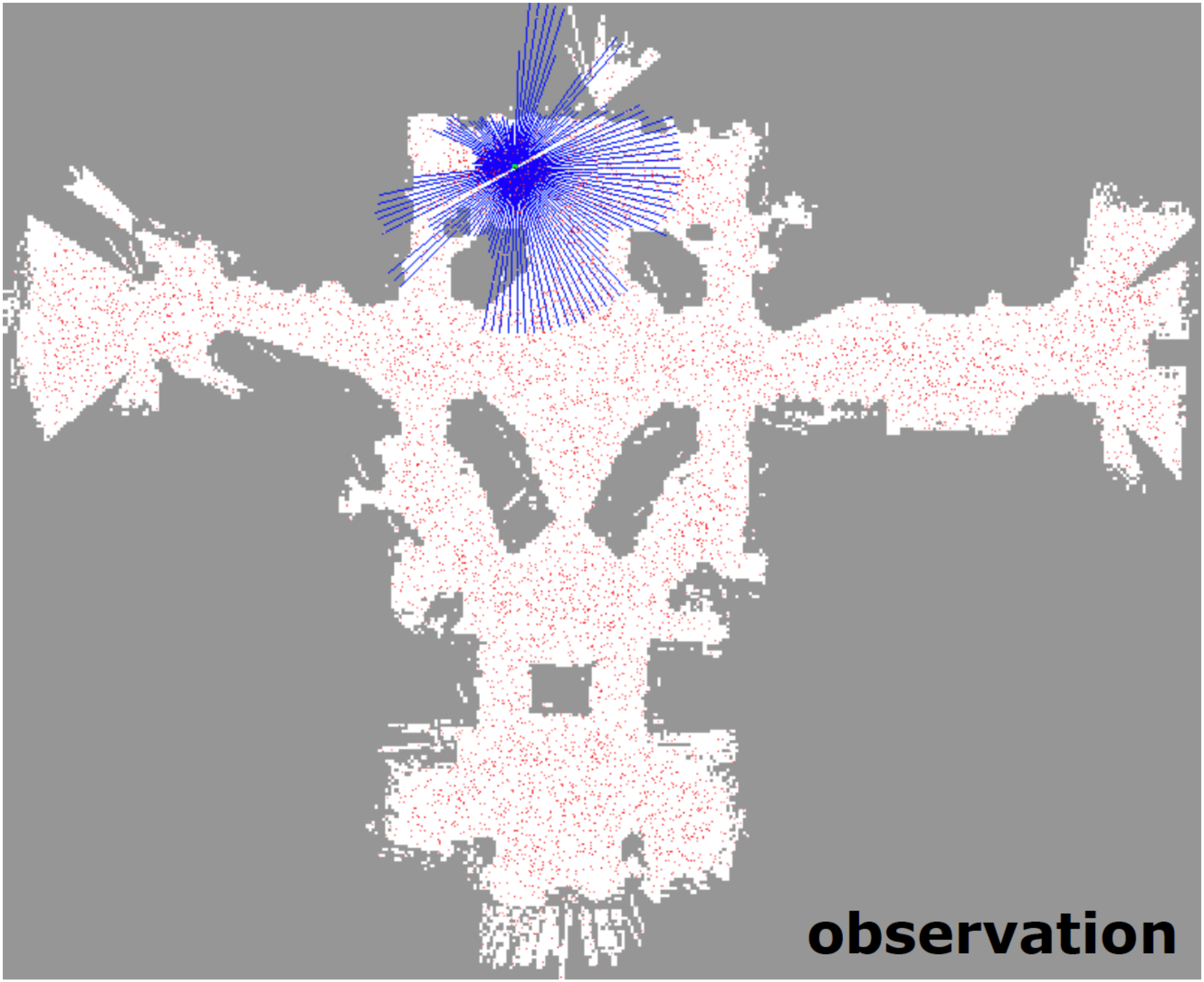
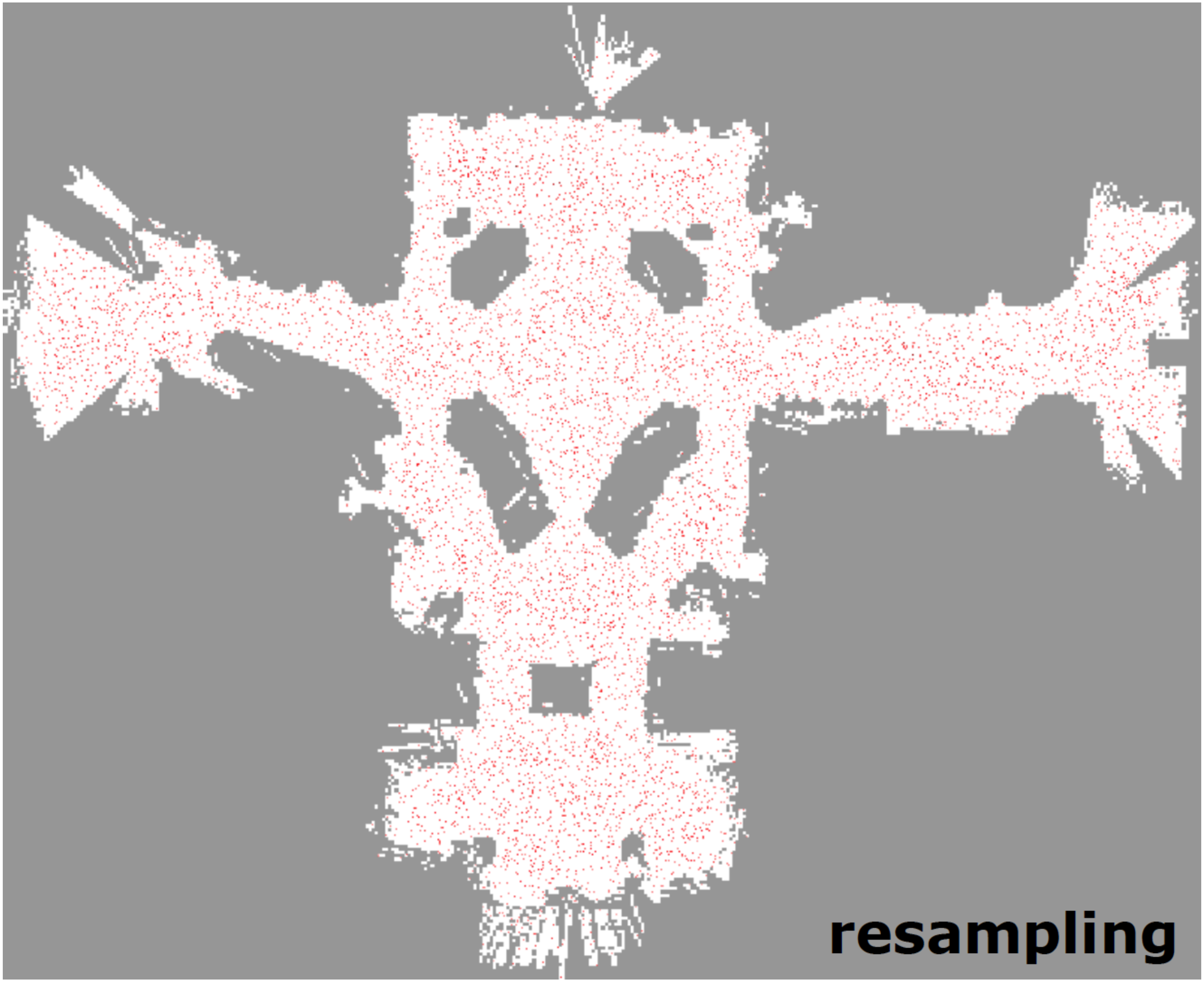


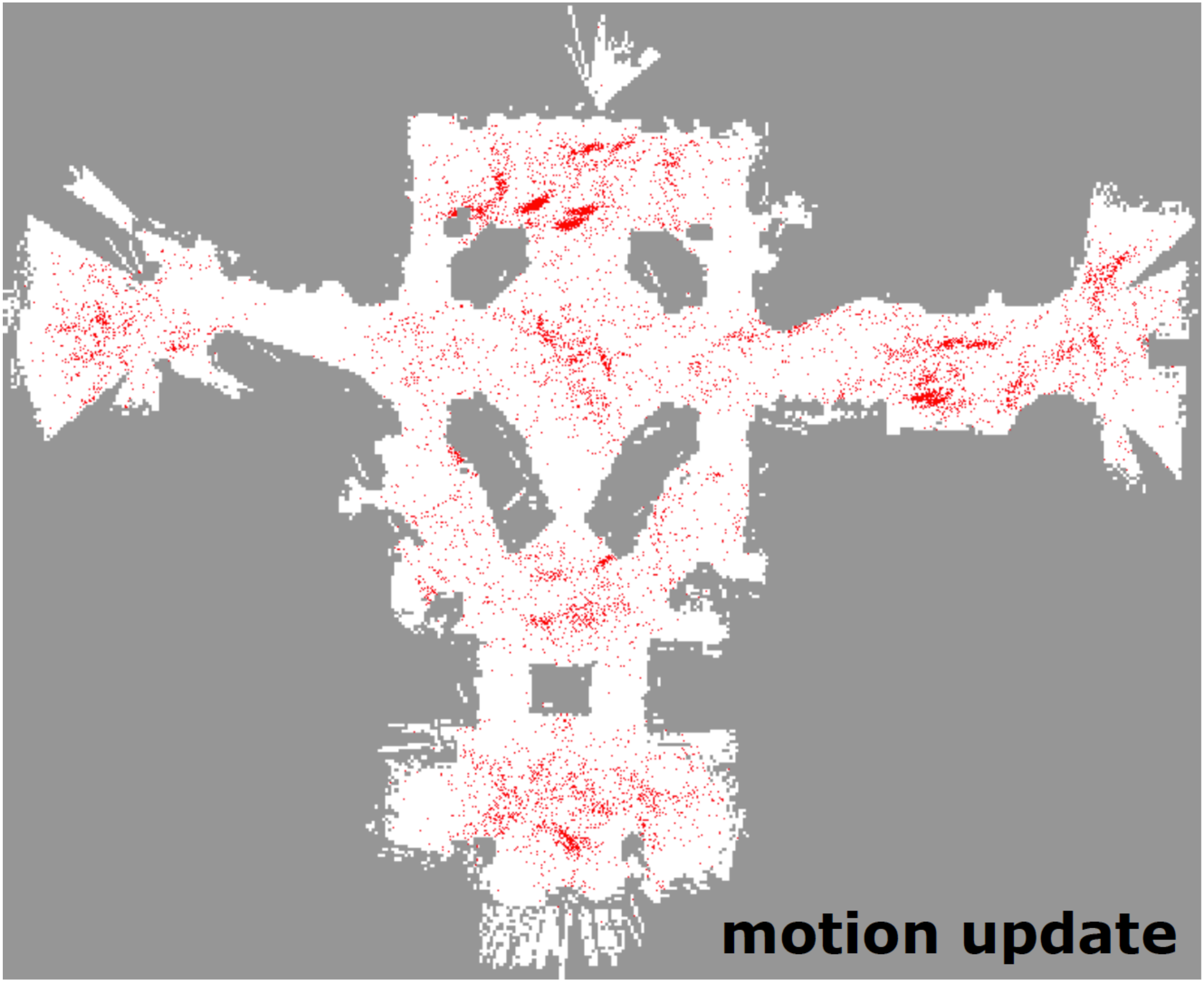
FIGURE 1.4: Resampling hugely improves the behavior of a particle filter. We now show a resampled particle filter tracking a point drifting on the line (i.e., $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise, and are the same as for Figure 1.3. In the figure on the **left**, we compare an exact representation obtained using a Kalman filter with one computed from simple particle filtering. We show the mean of the posterior as a point with a one standard deviation bar. The mean obtained using a Kalman filter is given as an 'x'; the mean obtained using a particle filter is given as an 'o'; we have offset the standard deviation bars from one another so as to make the phenomenon clear. Notice that estimates of both mean and standard deviation obtained from the particle filter compare well with the exact values obtained from the Kalman filter. The figure on the **right** indicates where this improvement came from; we plot the tracks of ten particles, randomly selected from the 100 used. Because we are now resampling the particles according to their weights, particles that tend to reflect the state rather well usually reappear in the resampled set. This means that many particles lie within one standard deviation of the mean of the posterior, and so the weights on the particles tend to have much smaller variance, meaning the representation is more efficient.

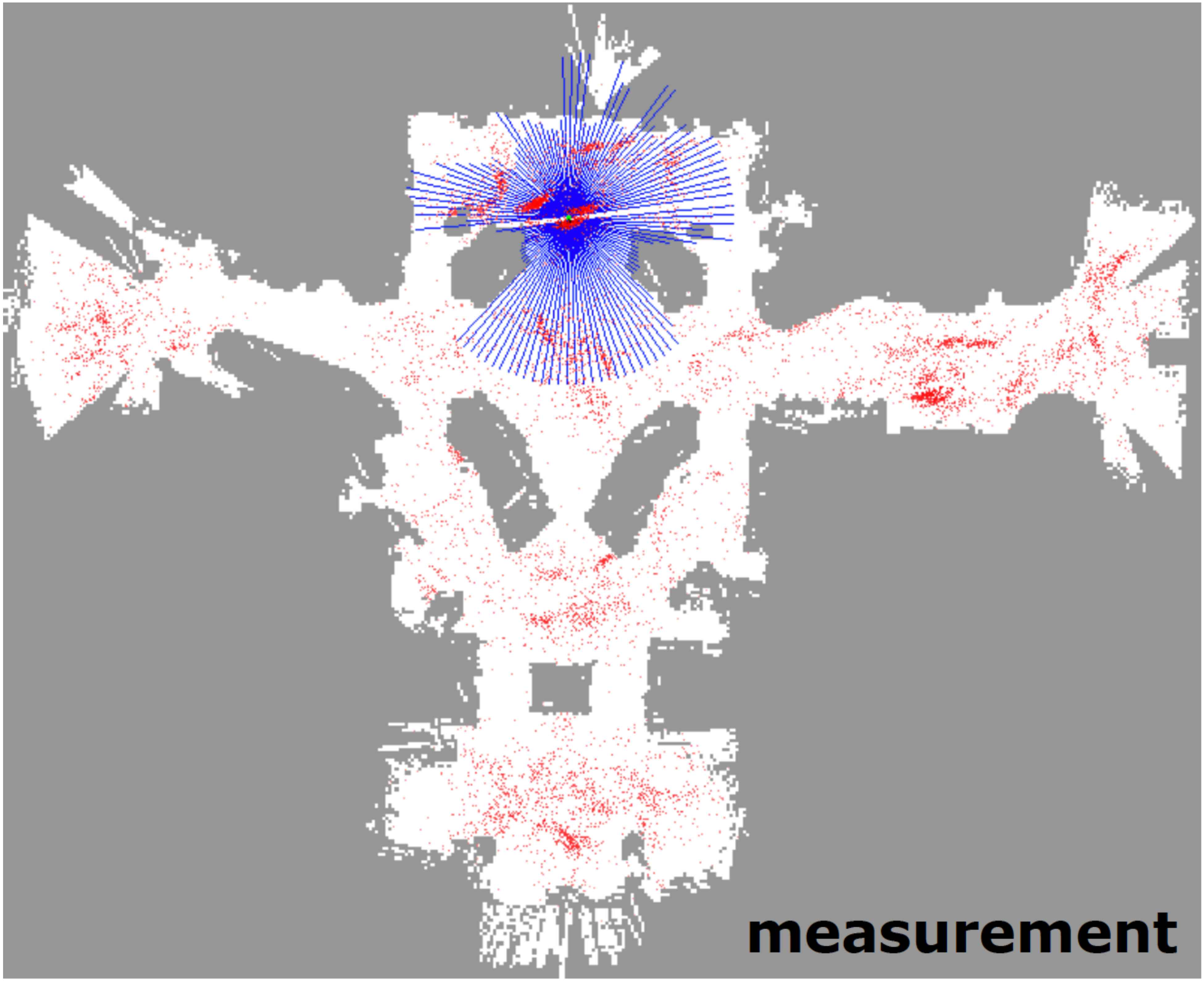

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
35
40

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
35
40

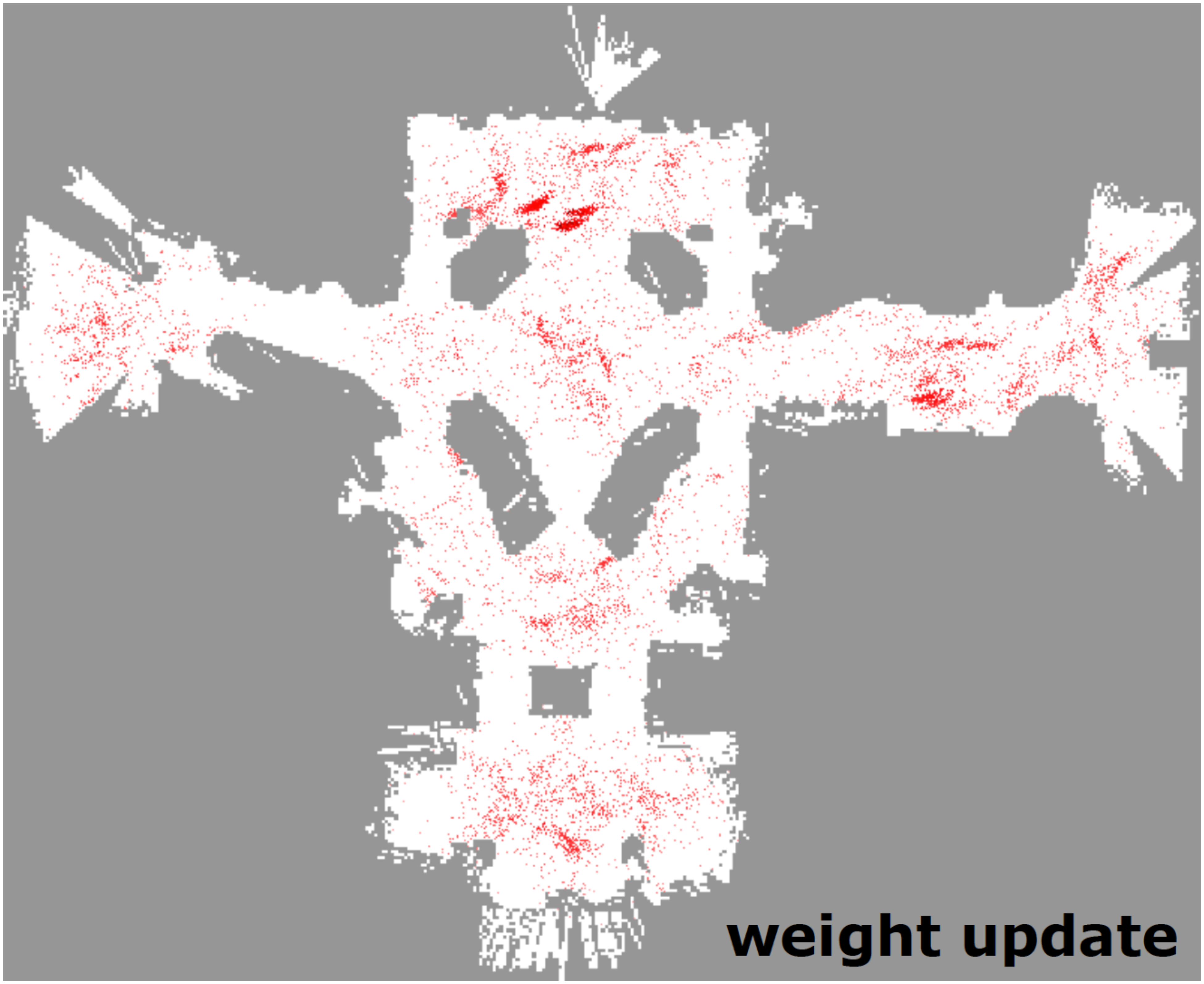

initialization

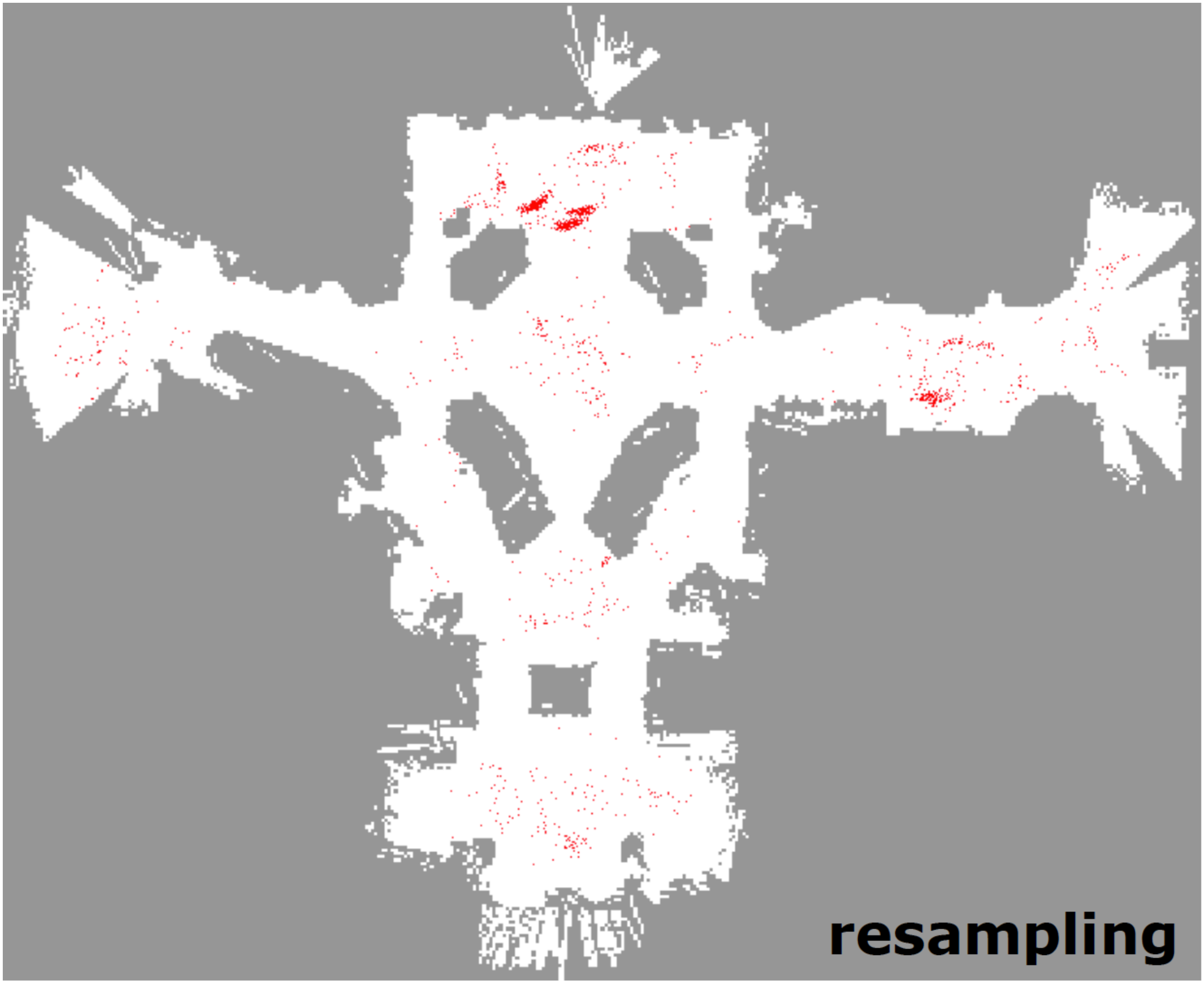

observation

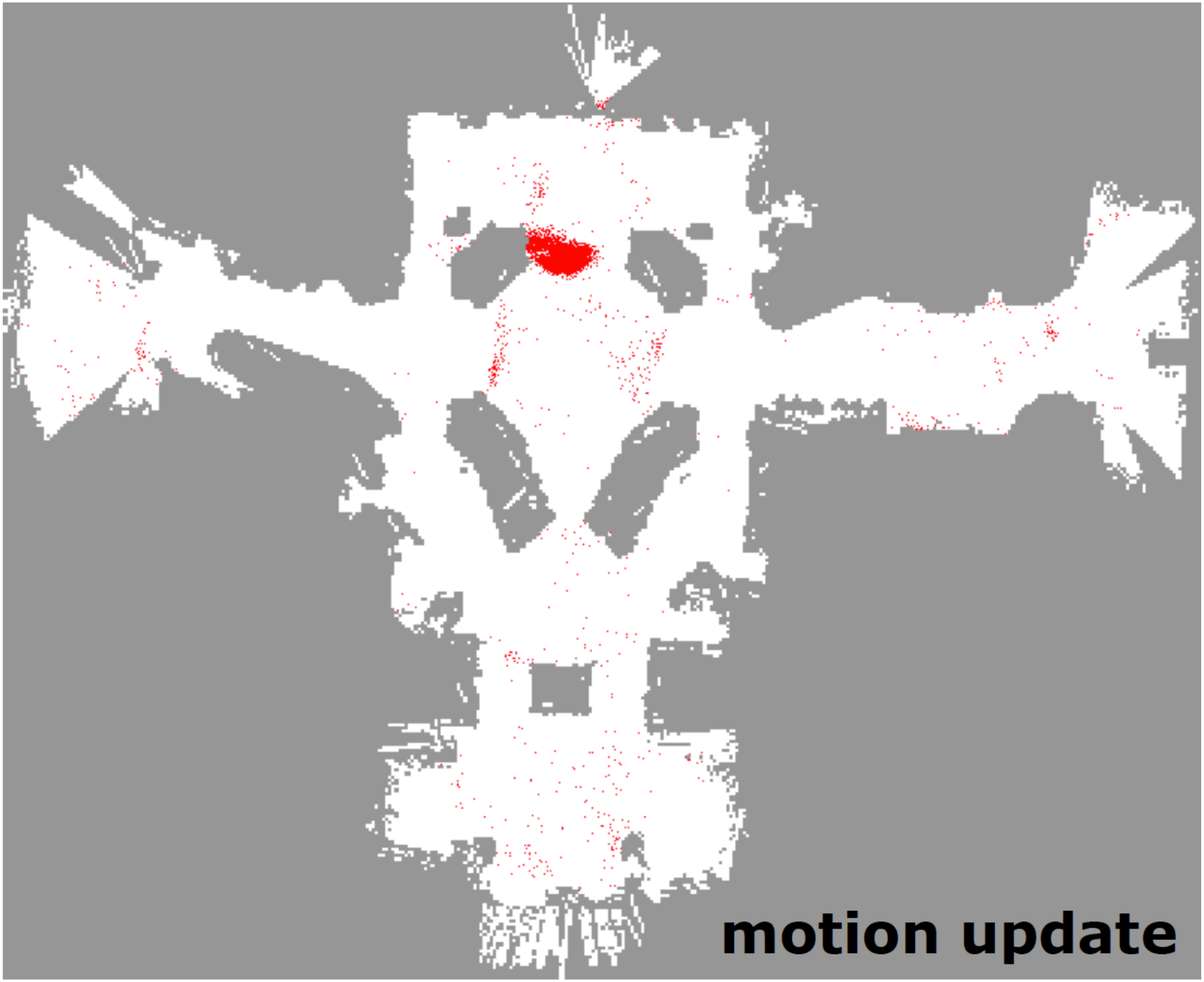

resampling

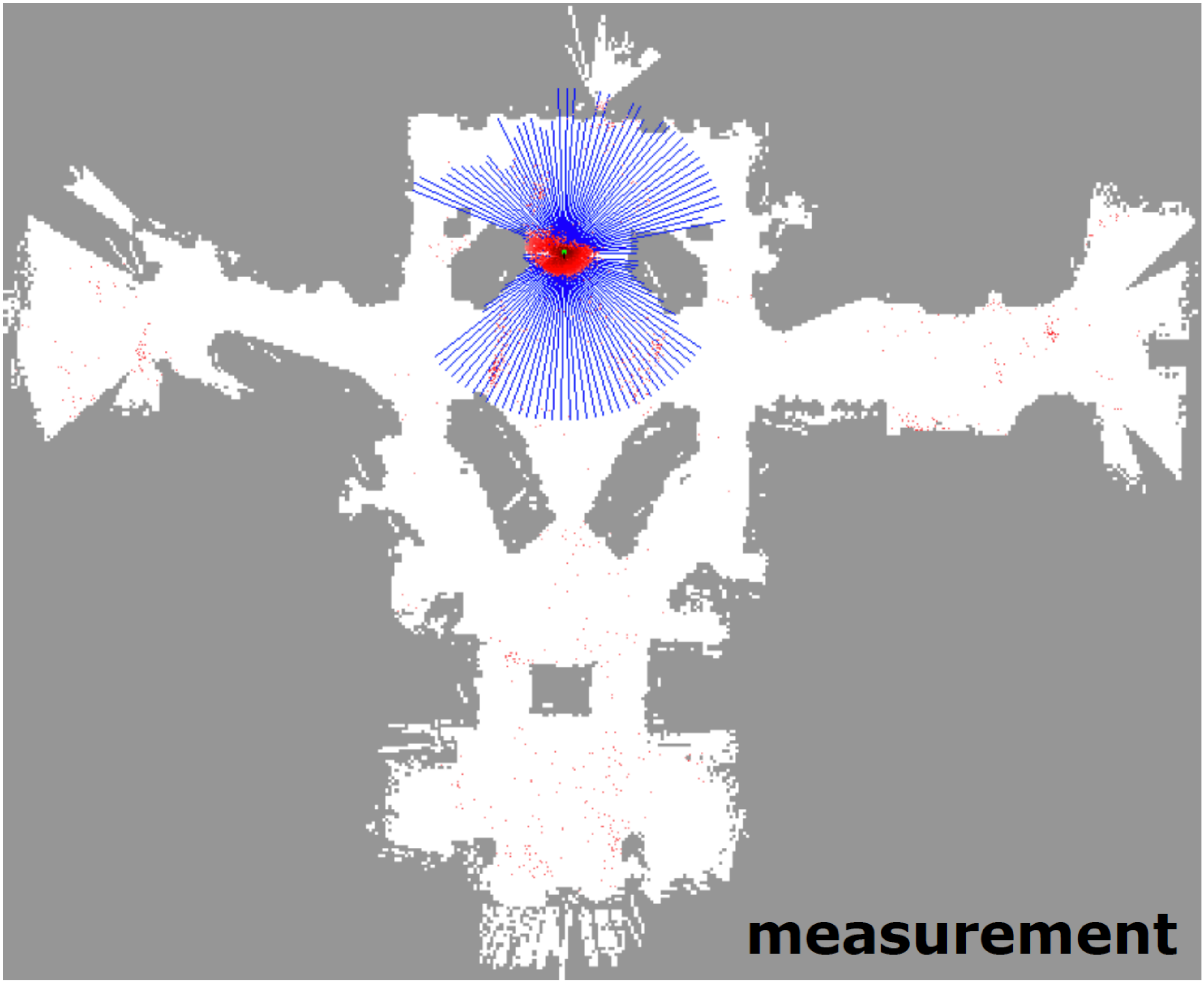

motion update

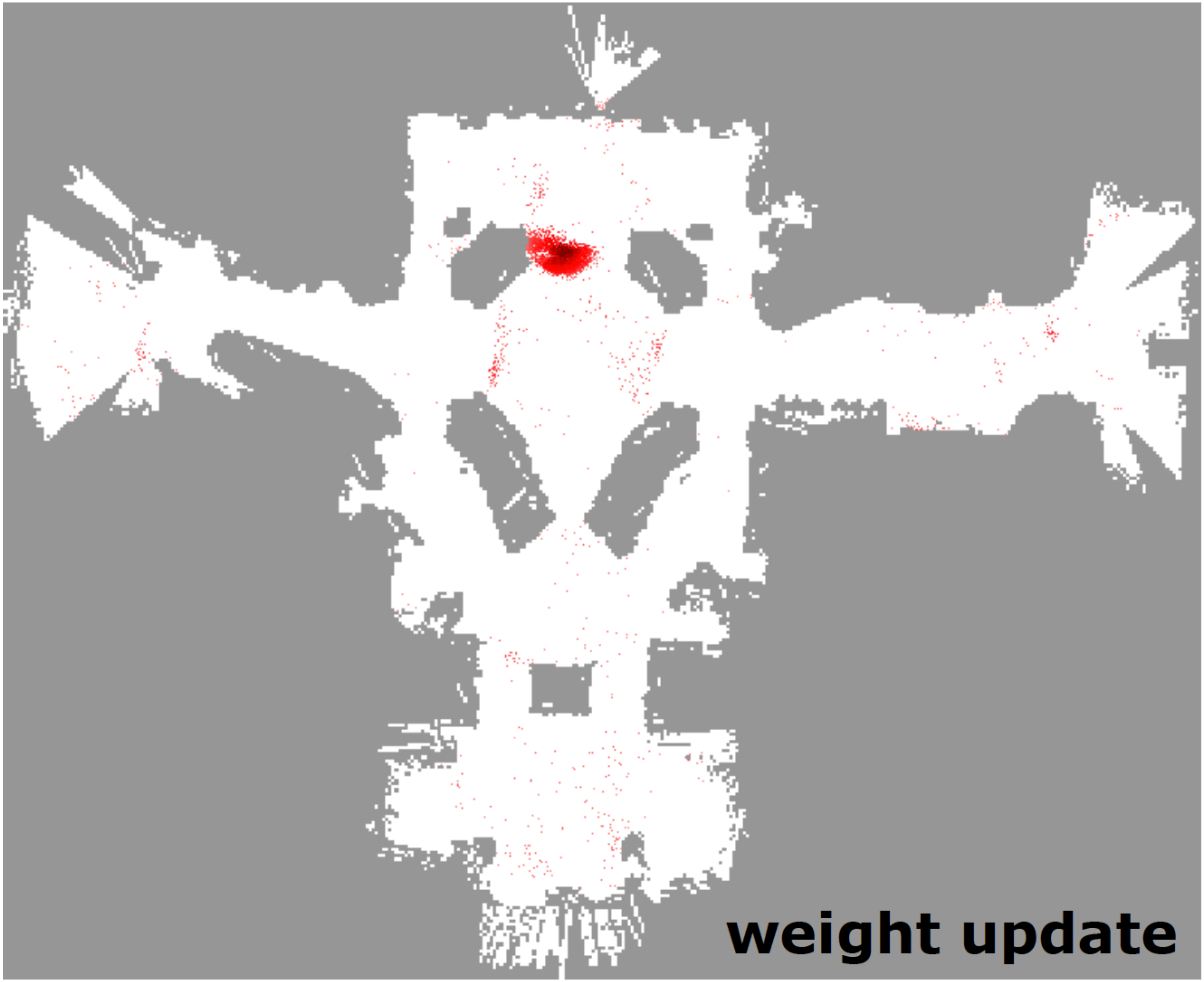

measurement

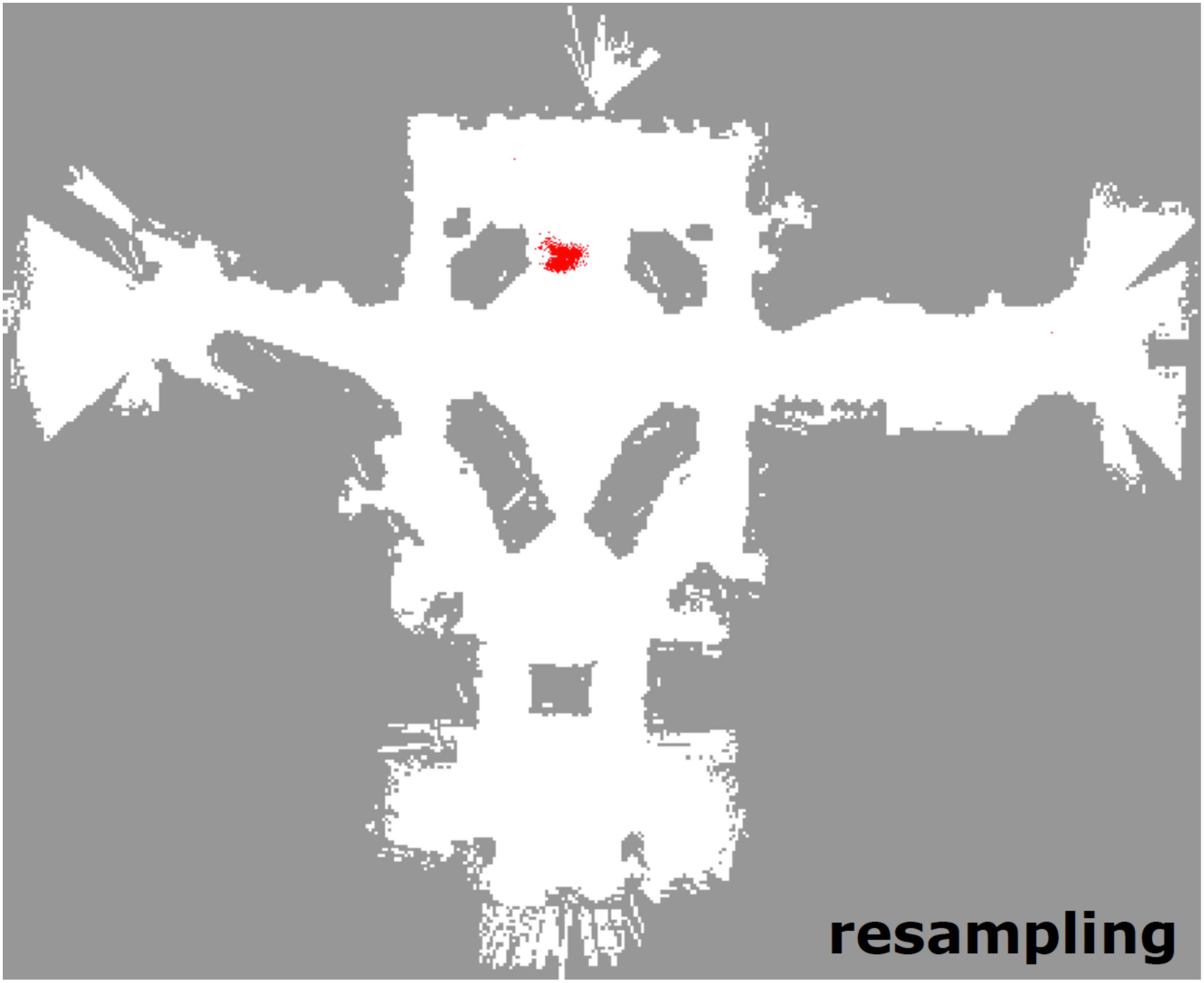

weight update

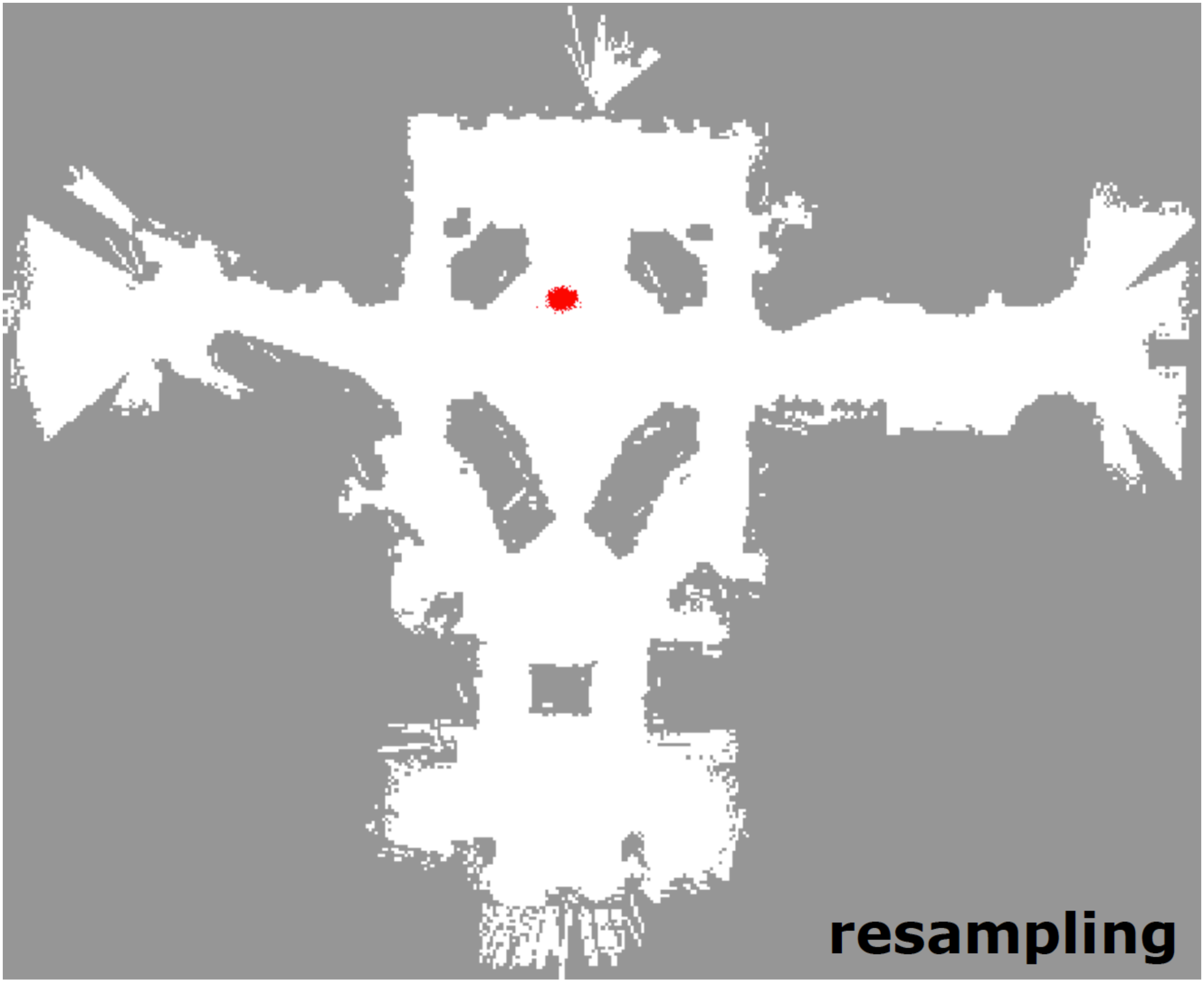

resampling


motion update


measurement


weight update


resampling

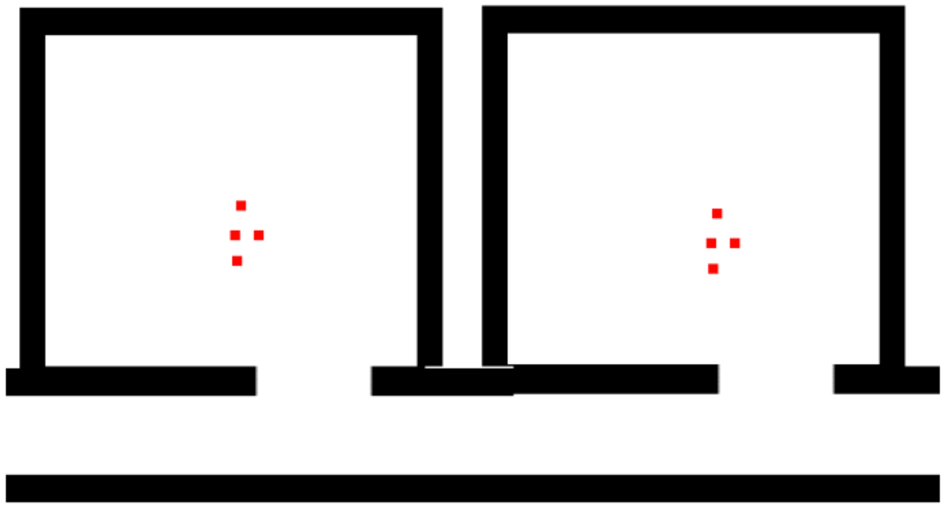

resampling

Features

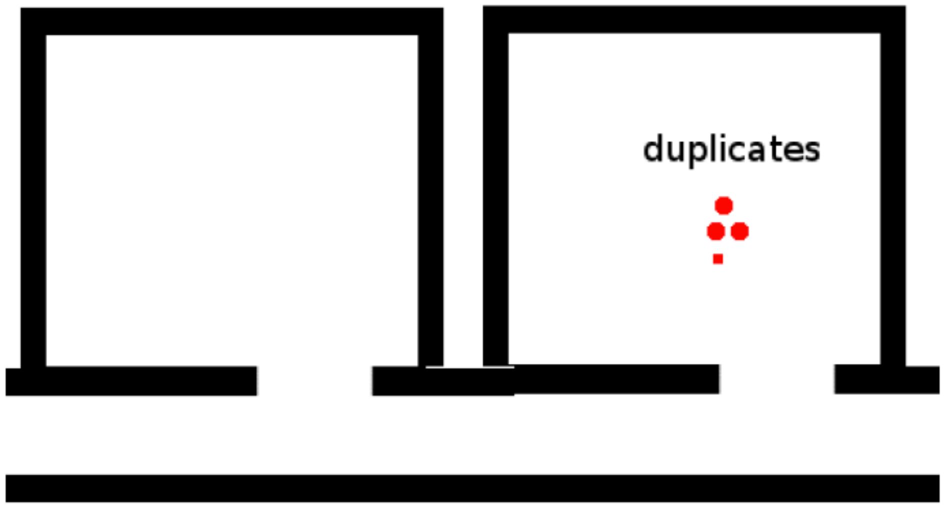
- Can (in principle) produce any expectation of posterior
- Works for any observation model, any motion model
- Embarrassingly parallel
- Work (in principle) for any dimension
- Easy to make

Problems: Loss of diversity

- Loss of diversity
  - repeated resampling can cause serious problems



(a) Two rooms with equal probability



(b) The same filter, after repeated resampling

Fixes

- Skip resampling
  - if (say) variance/entropy of weights hasn't changed much
- Use low variance resampling
  - next slide
- Add particles
  - easy; often helps;
  - push more particles than required through dynamics; weight+sample
- Inject uniform samples
  - yuck - only in desperation

Low variance resampling

- M = number of particles

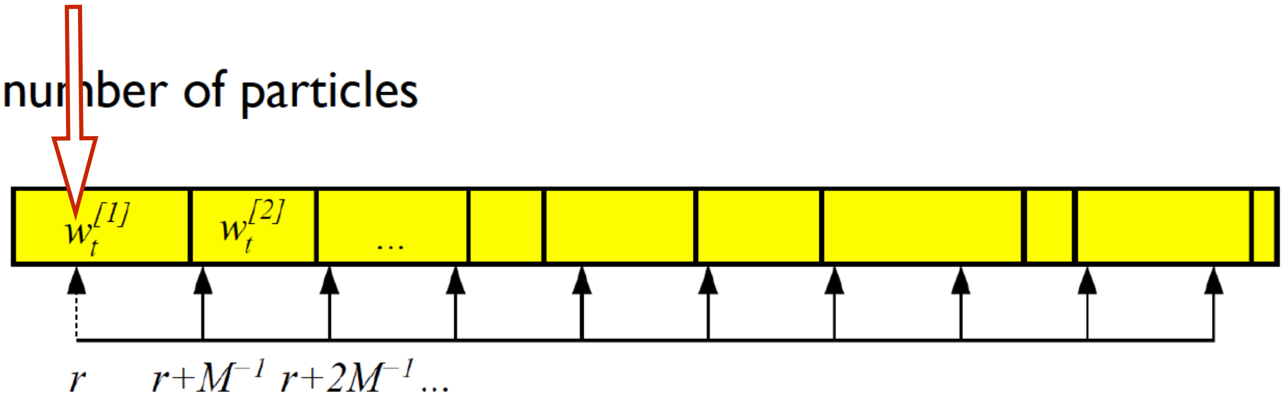
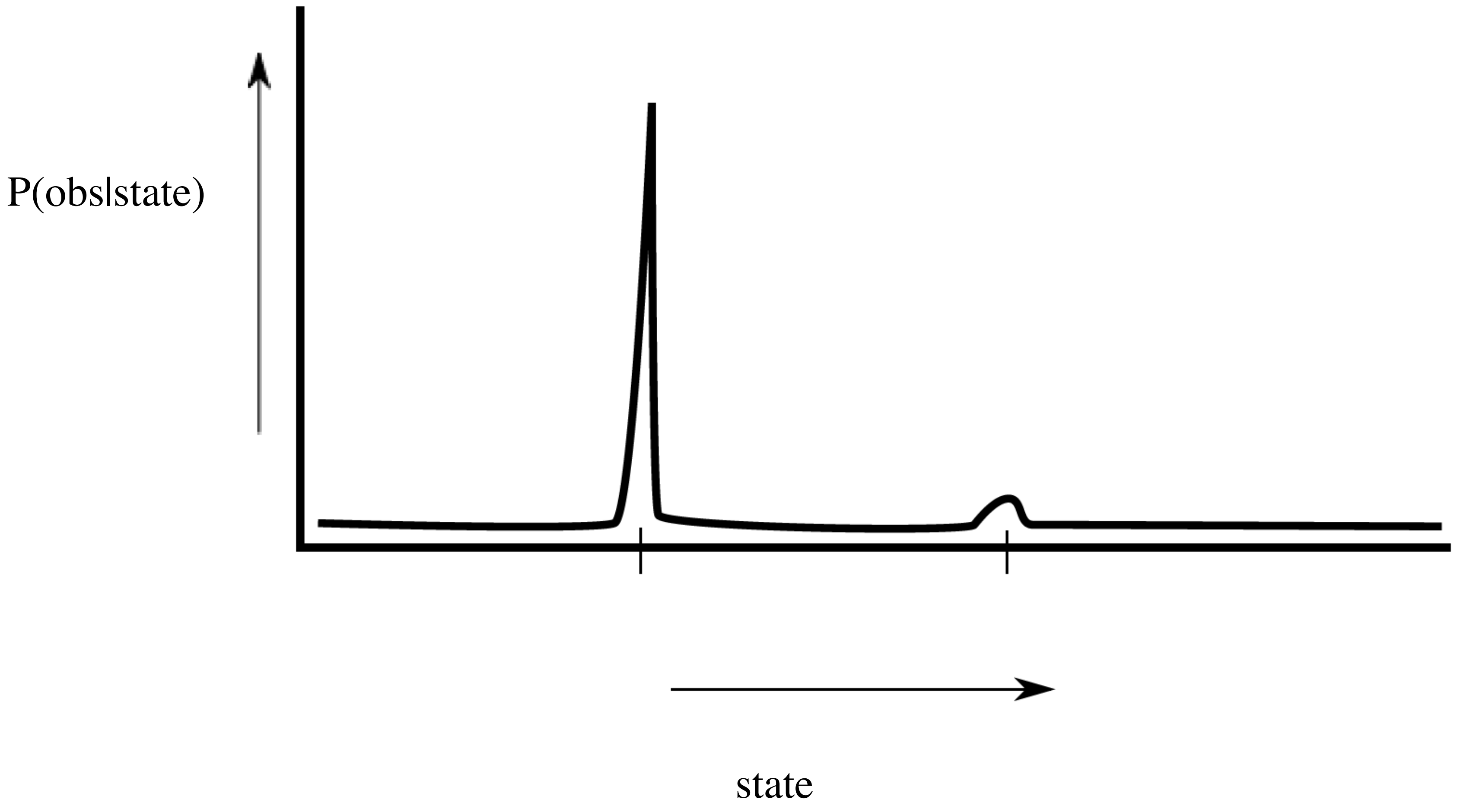


Figure 4.7 Principle of the low variance resampling procedure. We choose a random number r and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

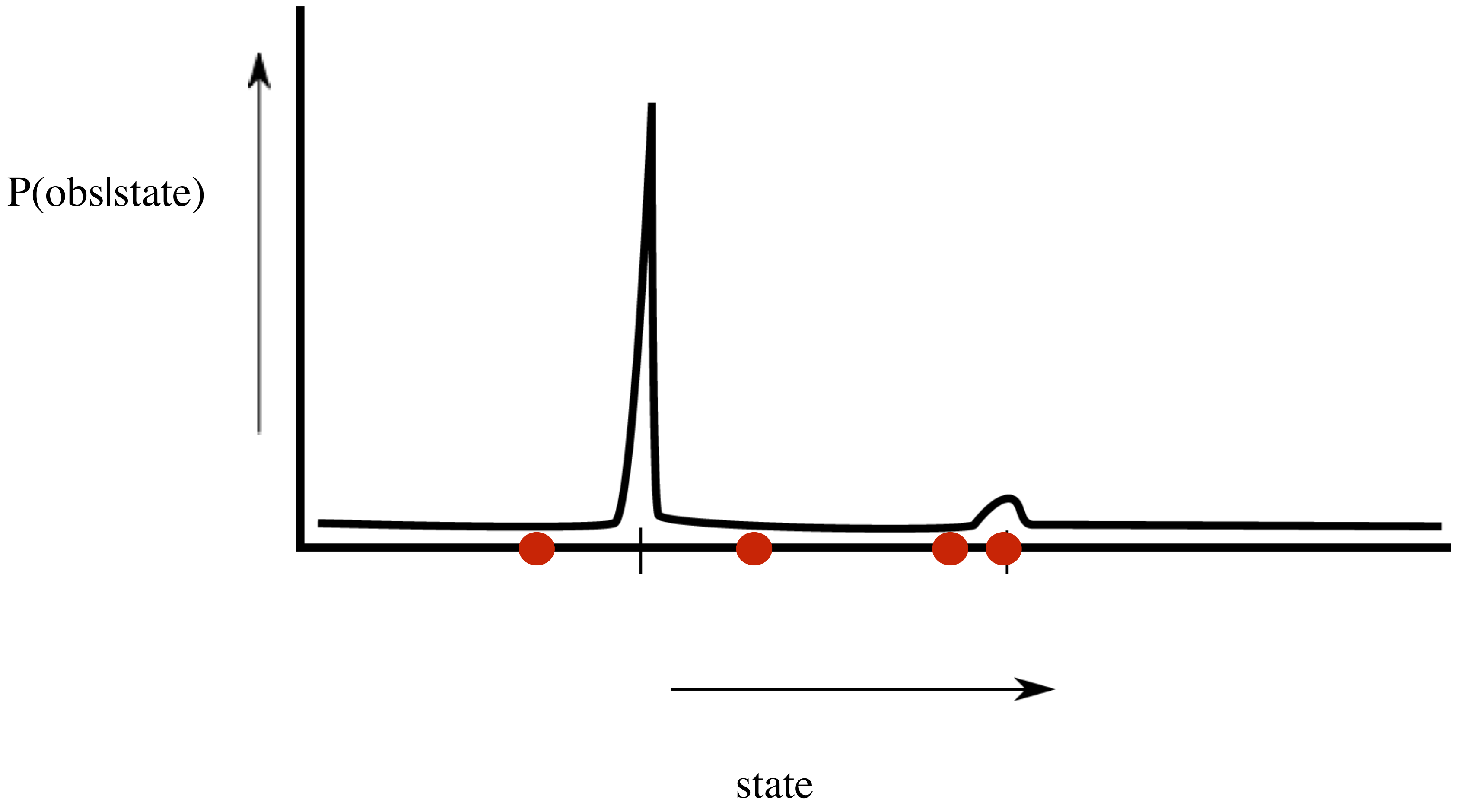
  - r ∈ [0, 1/M]

- Advantages:
  - More systematic coverage of space of samples
  - If all samples have same importance weight, no samples are lost
  - Lower computational complexity

Problem: Accurate observations (!?!)



Problem: Accurate observations (!?!)



What is going on?

- We are searching a sharply peaked function
  - using a proposal that is broad
  - this can’t work well
- Fix
  - use a different proposal process
    - particles don’t *HAVE* to come from dynamics
    - eg generate samples from observation
      - reweight with dynamics
      - can be hard to do
  - more particles
    - iffy
  - smooth observation model
    - aaargh!

Other nuisances

- Performance measurement can be hard
- Can be quite demanding computationally
  - if distributions are unimodal use Kalman filter (or EKF)
- Not deterministic