Contents

1 Monte Carlo Inference

Motivation

With nonlinear dynamics and/or measurement models exact inference is computationally intractable. One way out is to perform approximate inference using sampling. One of the best known such algorithms are particle filters, including Monte Carlo Localization.

1.1 Simulating from a Continuous Bayes Net

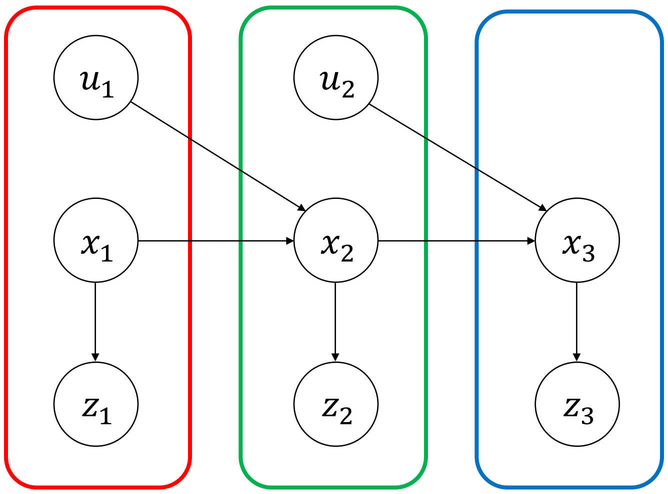

Figure 1: Ancestral sampling in a continuous Bayes net.

Sampling from a continuous Bayes net is done using ancestral sampling, as before: we topologically sort the DAG, and then sample the conditional densities in topological sort order. For the dynamic Bayes net from Figure **??**, the topological sort trivially follows the temporal ordering, and we sample first within the red, then green, and then the blue time-slice, as shown in Figure 1.

In each slice, the story is pretty much the same. For example, in the second (green) slice we will

1. Sample the state x_2 from the continuous motion model $p(x_2|x_1, u_1)$, by evaluating $g(x_1, u_1)$, and adding Gaussian noise with mean $\mu = 0$ and covariance Q.
2. Sample the measurement z_2 from the continuous measurement model $p(z_2|x_2)$, and adding zero-mean Gaussian noise with measurement covariance R.
3. Sample a control u_2 that we will use in the next time-slice.

The only exception is the first slice, where the first state x_1 is sampled from the prior $p(x_1)$.

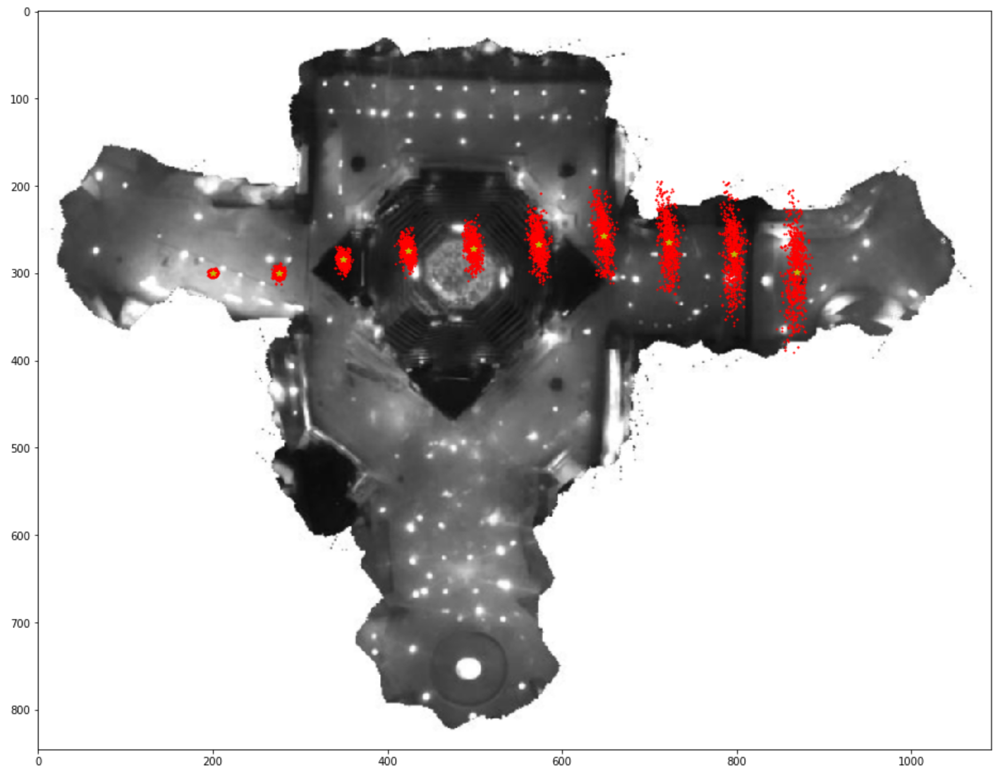

Figure 2: Sampling from the motion-model only, giving rise to so-called "banana-shaped" densities.

To illustrate these concepts we use an example where we use a large appearance-based map to localize a robot in a large known environment. The basic idea, which was detailed in a 1999 CVPR paper, was to generate an image mosaic of the ceiling of the environment in which the robot needs to operate, and use that during operation to localize the robot. An example of such a **ceiling mosaic** from our testbed-application is shown in Figure 2. The figure shows a large portion (40 m. deep by 60 m. wide) of the first floor ceiling of the Museum for American History (MAH), one of the Smithsonians in Washington, DC. The mosaic was generated from 250 images in a mapping phase.

The figure shows the result of sampling from the motion-model only, giving rise to so-called "banana-shaped" densities, shown as red point clouds. In this example, the states are 2D poses $x = (x_x, x_y, x_\theta)$, and the control has a forward velocity component u_v and an angular velocity component u_θ. The motion model is then:

$$g(x, u) = (x_x + u_v \cos(x_\theta)\Delta t, x_y + u_v \sin(x_\theta)\Delta t, x_\theta + u_\theta \Delta t)$$

What is happening is mysterious at first, until you realize that the red dots in the figure only show the marginal of the robot position (x_x, x_y), not the full 3-dimensional density on the pose. The Gaussian additive noise can make the resulting densities decidedly non-Gaussian after several iterations. The reason is the noise on the heading x_θ: this leads to non-linear effects over time.

Note that the Figure does not yet show the result of incorporating the information from the sensor. That will be discussed below.

1.2 Sampling as an Approximate Representation

The solution to the robot localization problem is obtained by running a Bayes filter, alternating between prediction and measurement, as we saw before. Depending on how one chooses to represent the **filtering density** $p(x_t|Z^t)$, one obtains algorithms with vastly different properties.

If both the motion and the measurement model are linear with additive Gaussian noise, and the initial state is also specified as a Gaussian, then the filtering density $p(x_t|Z^t)$ will remain Gaussian at all times. This is the basis of the classical **Kalman filter**. Kalman-filter based techniques are particularly efficient, but in most robotics applications this basic assumption of Gaussian densities is violated. Indeed, for many sensors the likelihood functions are typically complex and multi-modal, as discussed above.

To overcome these disadvantages, different approaches have used increasingly richer schemes to represent uncertainty, moving away from the restricted Gaussian density assumption inherent in the Kalman filter. These different methods can be roughly distinguished by the type of discretization used for the representation of the state space. We have already discussed **Markov localization,** which discretizes the world and can be used in grid worlds and for localizing topological maps. However, in many applications we are interested in a more fine-grained position estimates, e.g., in environments with a simple topology but large open spaces, where accurate placement of the robot is needed. Discretizing the continuous world is powerful, but suffers from the disadvantages of computational overhead and a priori commitment to the size of the state space. For example, the resolution and hence the precision at which they can represent the state has to be fixed in advance.

Finally, one can represent the density by a set of **samples** that are randomly drawn from it. This is the representation we will use in this section. In sampling-based methods one represents the filtering density $p(x_t|Z^t)$ by a set of N random samples or particles $S^t = x_t^{(s)}|s = 1..N$ drawn from it. We are able to do this be-cause of the essential duality between the samples and the density from which they are generated. From the samples we can always approximately reconstruct the density, e.g. using a histogram or a kernel-based density estimation technique.

One nice property of samples as an approximate density is that we can easily compute expected values such as the mean, using a **Monte Carlo approximation**. The name Monte Carlo derives from the city in the south of France, famous for its casinos, where of course a lot of randomness is involved. For the mean, the Monte Carlo approximation is

$$\mu = \int xp(x)dx \approx \frac{1}{N}\sum_r x^{(r)},$$

with N the number of samples. In fact, we can approximate any expectation of any function of x:

$$E_p[f(x)] \approx \frac{1}{N}\sum_r f\left(x^{(r)}\right),$$

where f can be a scalar or multivariate function of the random variable x.

Marginalization of sampled representations is very easy: if we have a set of samples

$$\left\{(x, y)^{(r)}\right\} \sim p(x, y)$$

then the sampling-based approximation of the marginal $p(y)$, for example, is obtained by just taking the y component of each sample:

$$\left\{y^{(r)}\right\} \sim p(y)$$

If we instantiate these ideas within the context of localization, the goal is then to recursively compute at each time- step t the set of samples S^t that is drawn from $p(x_t|Z^t)$. A particularly elegant algorithm to accomplish this is the particle filter, which is a Monte Carlo approximation of the Bayes filtering paradigm. Before we go there, however, we need to introduce one more piece of the puzzle in the next section, which is importance sampling.

Exercise

Prove that the Monte Carlo approximation of $y = Ax$ is equal to $A\hat{\mu}$, where $\hat{\mu}$ is the approximation of the mean.

1.3 Importance Sampling

Sampling from arbitrary densities is not easy in general. Gaussian densities, which are relatively easy to sample from, are the exception rather than the rule.

A simple method is **importance sampling**. Given any **target density** $p(x)$, we instead sample from a (hopefully easier) **proposal density** $q(x)$, and then produce *weighted samples* with **importance weights**

$$w^{(r)} = \frac{p(x^{(r)})}{q(x^{(r)})}$$

where $x^{(r)}$ is a sample (with index r) from $q(x)$. After sampling the required number, the importance weights can be normalized to sum up to 1.

Importance sampling can be used as a simple approximate implementation of Bayes rule. In this case, the target density is $p(x|z)$, and we can use the prior $p(x)$ as the proposal density. Hence, the weights can be computed as

$$w^{(r)} = \frac{p(x^{(r)}|z)}{p(x^{(r)})} \propto \frac{l(x^{(r)}; z)p(x^{(r)})}{p(x^{(r)})} = l(x^{(r)}; z)$$

where we used $p(x|z) \propto l(x; z)p(x)$, with $l(x; z)$ the likelihood of x given z. Hence, when implementing Bayes law by proposing from the prior, the weights are simply the measurement likelihoods $l(x^{(r)}; z)$.

1.4 Particle Filters and Monte Carlo Localization

Recall that localization in robotics can be done via the Bayes filter. For continuous variables, the (exact) **filtering density** associated with a Bayes filter is given by

$$p(x_t|Z^t) \propto l(x_t; z_t) \int_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|Z^{t-1}), \tag{1}$$

where $Z^t \triangleq \{z_1, z_2 \dots z_t\}$ is shorthand for all measurement up to and including time t. In other words, the filtering density$p(x_t|Z^t)$ at time t is recursively obtained from the filtering density $p(x_{t-1}|Z^{t-1})$ at time $t-1$, by first calculating the **predictive density**

$$p(x_t|Z^{t-1}) = \int_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|Z^{t-1}), \tag{2}$$

and then weighting with the likelihood function $l(x_t; z_t)$.

There are two ways to intuitively derive particle filtering, which we discuss one by one in the first two sections below.

1.4.1 Resampling-based MCL

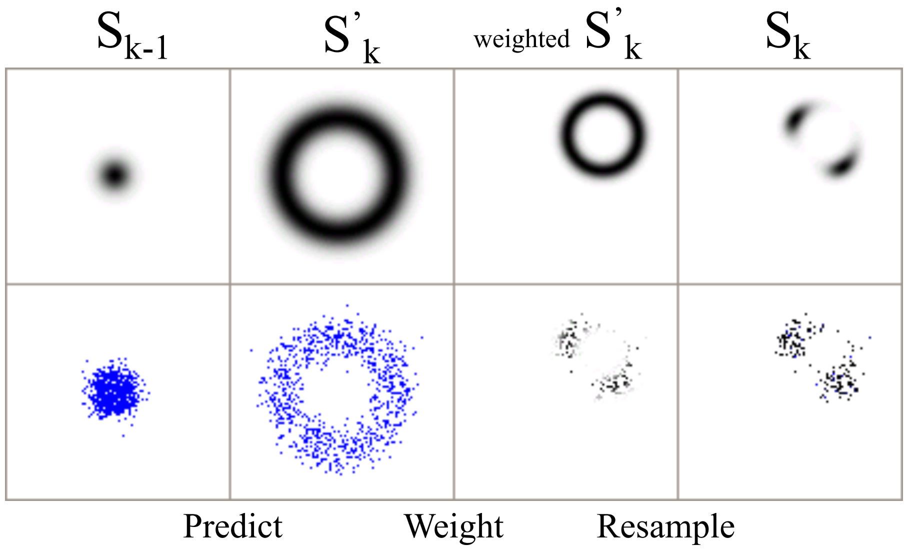

Figure 3: Schematic explanation of resampling-based MCL, see text.

Let us assume that we have a set of samples on X_{t-1}, i.e., $\{x_{t-1}^{(s)}\} \sim p(x_{t-1}|Z^{t-1})$. We create a tiny Bayes net, $x_{t-1} \to x_t$, and extend the sample set as in ancestral sampling, obtaining $\left\{(x_{t-1}, x_t)^{(s)}\right\} \sim p(x_{t-1}, x_t|Z^{t-1})$. We then marginalize to $\{x_t^{(s)}\} \sim p(x_t|Z^{t-1})$, weight by the likelihoods $l(x_t^{(s)}; z_t)$ to obtain $\{x_t^{(s)}, w_t^{(s)}\} \sim p(X_t|Z^t)$. Finally, resample to get back to an unweighted set of particles for the next step. In summary, given the approximate posterior at the previous time step $\{x_{t-1}^{(s)}\} \sim p(x_{t-1}|Z^{t-1})$, we

1. extend to include the next state to obtain $\left\{(x_{t-1}, x_t)^{(s)}\right\} \sim p(x_{t-1}, x_t|Z^{t-1})$;
2. marginalize to obtain the approximate predictive density $\{x_t^{(s)}\} \sim p(x_t|Z^{t-1})$;
3. weight by the likelihoods $w_t^{(s)} = l(x_t^{(s)}; z_t)$ to obtain $\{x_t^{(s)}, w_t^{(s)}\} \sim p(x_t|Z^t)$;
4. resample to get back an unweighted approximation $\{x_t^{(r)}\} \sim p(x_t|Z^t)$.

One iteration of the algorithm is illustrated in 3. Each panel in the top row shows the exact density, whereas the panel below shows the particle-based representation of that density. In panel A, we start out with a cloud of samples S_{k-1} representing our uncertainty about the robot position. In the example, the robot is fairly localized, but its orientation is unknown. Panel B shows what happens to our belief state when we are told the robot has moved exactly one meter since the last time-step: we now know the robot to be somewhere on a circle of 1 meter radius around the previous location. Panel C shows what happens when we observe a landmark, half a meter away, somewhere in the top-right corner: the top panel shows the likelihood $p(z_k|x_k)$, and the bottom panel illustrates how each sample is weighted according to this likelihood. Finally, panel D shows the effect of resampling from this weighted set, and this forms the starting point for the next iteration.

1.4.2 Mixture-density MCL

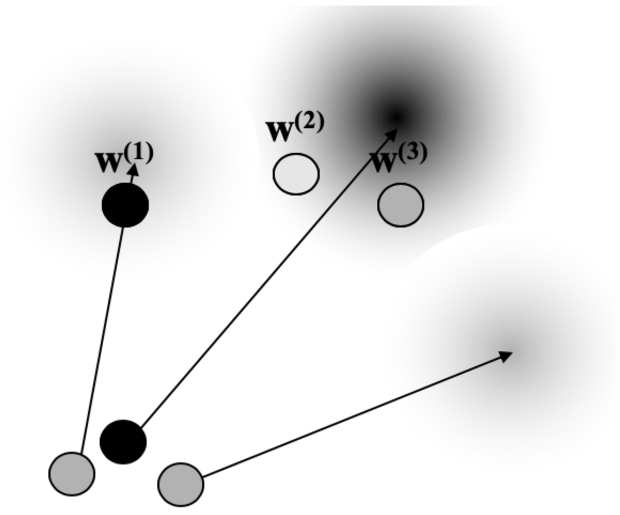

Figure 4: Sampling from the predictive density $p(x_t|Z^{t-1})$, approximated by a mixture density $\sum_r w_{t-1}^{(r)} p(x_t|x_{t-1}^{(r)})$ with 3 components. The middle component happens to be picked twice. After that, the three new samples each get a new likelihood weight $w^{(s)}$, as shown.

The second way to view a particle filter is by considering approximating the integral in the predictive density (2) with a sum approximation, i.e.,

$$p(x_t|Z^{t-1}) = \int_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|Z^{t-1}) \approx \sum_r w_{t-1}^{(r)} p(x_t|x_{t-1}^{(r)}) \tag{3}$$

Hence, in this view we start out with a weighted sample approximation for the previous filtering density$\{x_{t-1}^{(r)}, w_{t-1}^{(r)}\} \sim p(x_{t-1}|Z^{t-1})$, and then approximate predictive density with a mixture density as in Equation 3. We then sample from it to get $\{x_t^{(s)}\} \sim p(x_t|Z^{t-1})$, and then weight the samples using the likelihood: $\{x_t^{(s)}, w_t^{(s)}\} \sim P(x_t|Z^t)$, where $w_t^{(s)} = l(x_t^{(s)}; z_t)$. In summary, given the approximate posterior using weighted samples $\{x_{t-1}^{(r)}, w_{t-1}^{(r)}\} \sim p(x_{t-1}|Z^{t-1})$, we

1. approximate the predictive density as a mixture density $p(x_t|Z^{t-1}) \approx \sum_r w_{t-1}^{(r)} p(x_t|x_{t-1}^{(r)})$;

2. sample from it to get $\{x_t^{(s)}\} \sim p(x_t|Z^{t-1})$;

3. weight the samples using the likelihood: $\{x_t^{(s)}, w_t^{(s)}\} \sim P(x_t|Z^t)$, where $w_t^{(s)} = l(x_t^{(s)}; z_t)$.

The former derivation is a straight extension of importance sampling as in the previous section, but the resampling step is somewhat magical. The latter derivation shows that the resampling can be interpreted as approximating the predictive density for X_t. Of course, both derivations lead to the exact same algorithm.

1.4.3 Implementation

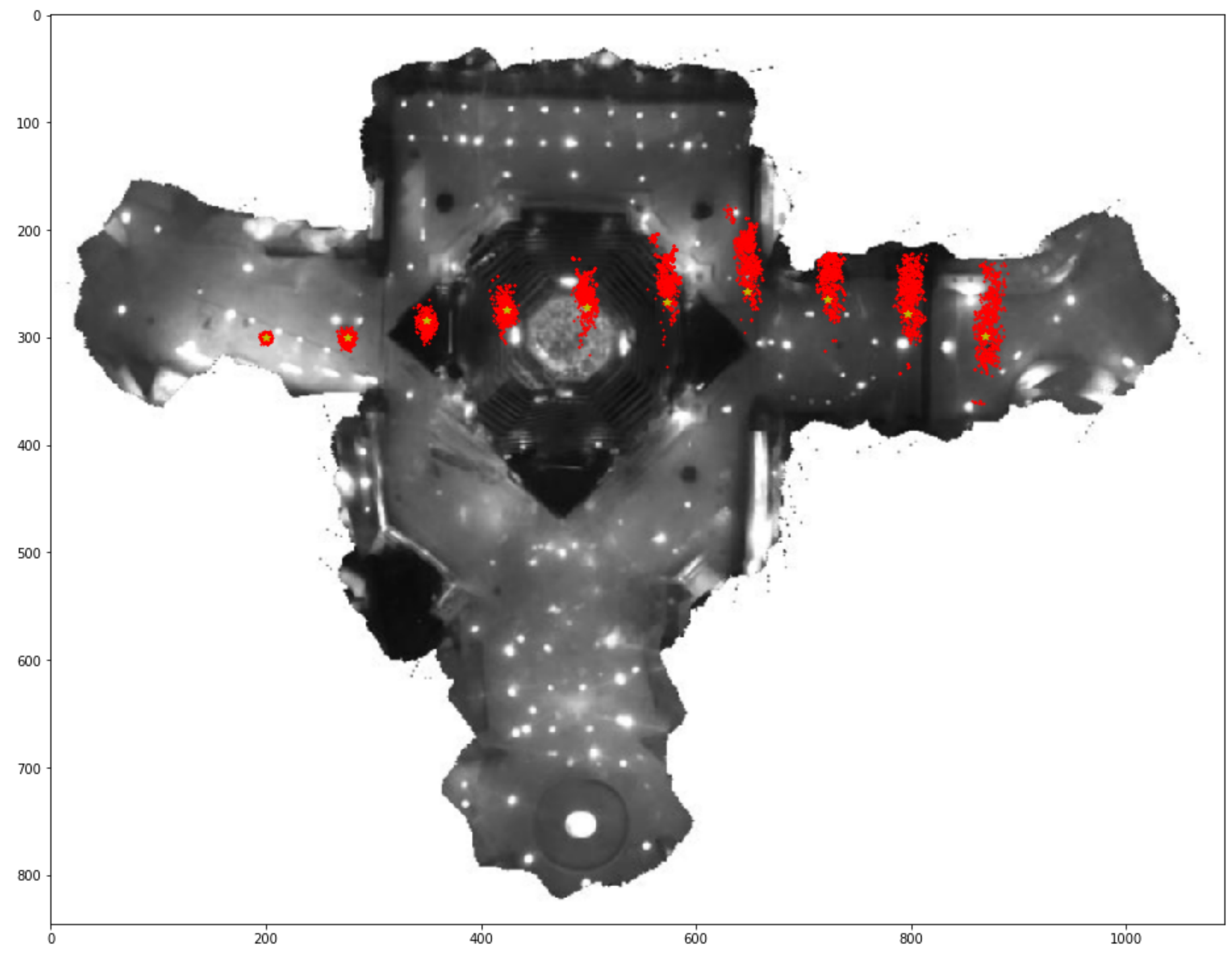

Figure 5: Monte Carlo localization resamples the particles based on weights derived from the sensors. In this case the sensor is simply picking up the grayscale value of the ceiling above the robot.

Particle filters are very easy to implement. In fact, the inner loop can be done in just three lines of python (provided you have numpy and scipy installed and imported):

```
I = numpy.random.choice(N,size=N,p=W)
X = numpy.array([np.random.multivariate_normal(g(X[i],u), cov) for i in I])
W = scipy.special.softmax([-(h(x)-z)**2 for x in X])
```

Above N is the number of samples, W are the weights for the samples X, and the variable *cov* is the motion model covariance Q. In the code above, the first two lines sample from the predictive mixture density $p(x_t|Z^{t-1}) \approx \sum_r w_{t-1}^{(r)} p(x_t|x_{t-1}^{(r)})$, and the last line uses "softmax" to weight each sample using the likelihood $l(x_t^{(s)}; z_t)$ and then normalize the weights.

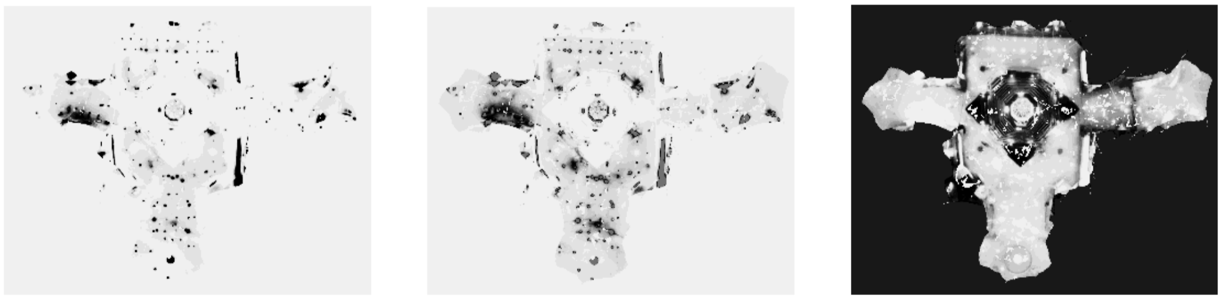
Figure 6: The likelihood functions $l(x|z)$ of being at a pose x given a brightness measurement z for three values of z. Left: high z indicates the robot is under a light. Middle: intermediate z. Right: low z: not under a light.

An example of this code in action is shown in Figure 5, for the ceiling localization example. In this case, the simulated sensor was simply the grayscale value at the simulated location of the robot. This is not a very informative sensor, but it does focus the distributions more on the correct location, indicated in yellow, as compared to the motion-model only particles from Figure (5). The likelihood functions $l(x|z)$ associated with this simple sensor model are in general complex and multimodal. To illustrate this, three different likelihood functions are shown in Figure (6), respectively when the robot is under a light, at the border of a light, and not under a light. It is clear that these densities do not resemble anything remotely like a Gaussian density. To see the full results with a real robot, operating in a real environment, please refer to the full 1999 paper.

Exercise

1. Show that both algorithms are the same.
2. Examine the python code in detail and try to fully understand it.

1.5 Connection with the Elimination Algorithm*

Factor graphs were not harmed in the making of this filter. But, you can interpret the weighted samples as a factor $f_1(x_{t-1})$ on x_{t-1}, then add a motion model to x_t and a likelihood factor on x_t.

A continuous version of eliminating x_{t-1}, if this was tractable, would form the product factor

$$\psi(x_{t-1}, x_t) = f_1(x_{t-1})p(x_t|x_{t-1})$$

and try to re-write it as $p(x_{t-1}|x_t)\tau(x_t)$. We saw in the discrete sum-product algorithm that this is done by marginalization. Doing this here, we obtain the new factor

$$\tau(x_t) = \sum f_1(x_{t-1})p(x_t|x_{t-1}),$$

which can be recognized as approximating the predictive density $p(x_t|Z^{t-1})$. In the elimination step we also usually calculate the conditional $p(x_{t-1}|x_t)$ but we do not need it here.

Then, eliminating x_t would form the product $\tau(x_t)l(x_t; z_t)$. This is hard if we would like to modify mixture components exactly, but easy if we first sample from them: we just re-weight then with the likelihood, and we are done. Hence, the factor graph interpretation closely aligns with the second derivation above.

1.6 Importance Sampling and Gibbs Sampling in Bayes Nets*

One of the downsides of Bayes filtering schemes is that we only get the posterior $p(x_t|Z^t)$ on the current state. But what if we are interested in the posterior $p(X^t|Z^t)$ on the entire trajectory? Below we discuss two schemes: importance sampling and Gibbs sampling.

1.6.1 Importance sampling

Importance sampling is easy. We once again partition the variables into unknown variables X and observed values Z. Rather than converting to a factor graph, we simply sample from the unknown variables in topological sort order, but whenever we encounter a variable $z \in Z$ we of course use the known value rather than sample. Then we give a weight to each sample $X^{(r)}$ equal to $w^{(r)} = p(Z|X) = \prod_z p(z|\Pi_z)$, and normalize the weights. Note that in calculating the weight we can omit any conditionals that do not involve unknown variables x.

As an example, consider again the dynamic Bayes net in Figure **??**. If the evidence is all actions and measurements, importance sampling comes down to sampling from the motion model Markov chain, as illustrated in Figure 2, and then weighting by $\prod p(z_t|x_t)$. We don't need to bother with the probabilities of the actions, as they do not depend on the states $\mathcal{X}$ in any way.

However, the variance associated with producing samples from the posterior makes it only useful for small examples. Indeed: we are multiplying many small numbers together, and if the posterior $p(x|z)$ is very different from the proposal distribution, many of the samples will be wasted. For a large number of variables, almost all samples will be thus wasted.

1.6.2 Gibbs Sampling

A theoretically better way to sample from a posterior is using **Gibbs sampling**. For this, we convert to a factor graph with the method we explained before. Then, we start with a random initial estimate for all variables, and then repeat the following simple procedure over and over: for all variables, sample from the conditional density $p(x|\mathcal{N}_x)$, where $\mathcal{N}_x$ is the set of all neighbors of x, and pretending these neighbors are *known*. We then replace the current value of x with the sampled value. The conditional density $p(x|\mathcal{N}_x)$ can be computed by eliminating x, i.e., forming the product factor. After visiting each variable in turn in each iteration, we produce a large sequence of samples for the set of unknowns X, one for each iteration of Gibbs sampling.

There are pros and cons to Gibbs sampling, as well. The biggest advantage is that it does not has much of a variance problem as importance sampling does. The disadvantages are three-fold: (a) it might not always be easy to calculate the densities needed, or sample from them; (b) the samples of X are correlated over time; (c) it could take a long time for this sample to converge to it stationary distribution $p(X|Z)$.

Summary

We briefly summarize what we learned in this section:

1. Simulating Continuous Bayes Nets
2. Sampling as Approximation
3. Importance Sampling
4. Particle Filters and Monte Carlo Localization
5. Monte Carlo Localization & the Elimination Algorithm