Statistical Techniques in Robotics (16-831, F14) Lecture#04 (Thursday September 4)

Importance Sampling and Particle Filters

Lecturer: Drew Bagnell *Scribe: Joe Bartels*[1]

1 Introduction

Last time, we discussed the Monte Carlo Method. In this lecture we will first discuss Importance Sampling and then Particle Filters. In particular, we will explain how they work, and the bad aspects of Particle Filters as well as fixes.

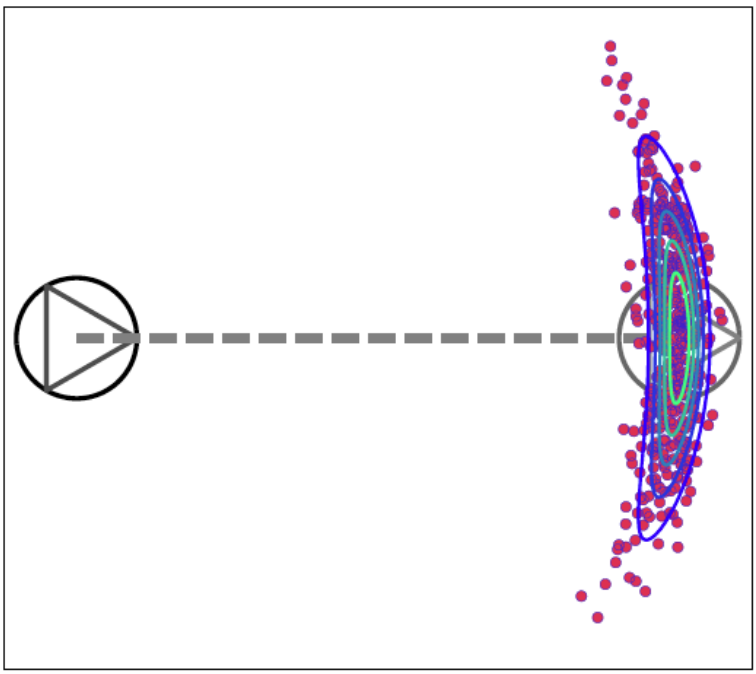


Figure 1: The motion model: Posterior distributions of the robot's pose upon executing the motion command illustrated by the dashed line. Isocontours are shown for the probability of the particle. This plot has been projected into 2-D. The original density is three-dimensional, taking the robot's heading direction, θ, into account.

2 Importance sampling (IS)

Although we are unable to sample from the required distribution, $p(x)$, we can use a trick that will allow us to sample from the known distribution, $q(x)$. The trick: Sample from the available distribution and reweight samples to fix it. Manipulating the expectation over $p(x)$ yields the following approximiation:

[1]Some content adapted from previous scribes: Nate Wood, David Fouhey, Mike Shomin, and M. Taylor

$$\underset{p(x)}{\mathrm{E}}[f(x)] = \sum p(x)f(x) \tag{1a}$$

$$= \sum p(x)f(x)\frac{q(x)}{q(x)} \tag{1b}$$

$$= \sum q(x)\frac{p(x)}{q(x)}f(x) \tag{1c}$$

$$= \underset{q(x)}{\mathrm{E}}[\frac{p(x)}{q(x)}f(x)] \tag{1d}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\frac{p(x_i)}{q(x_i)}f(x_i). \tag{1e}$$

$\frac{p(x)}{q(x)}$ is called the importance weight. When we are undersampling an area, it weights it stronger; likewise, oversampled areas are weighted more weakly.

Consider the following (somewhat contrived) pair of functions:

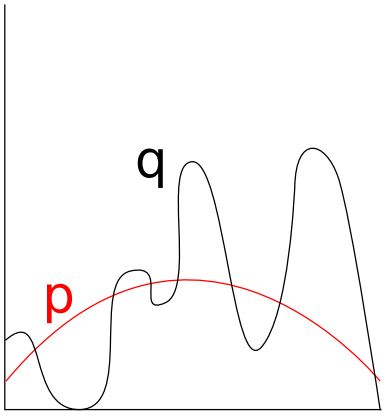


Figure 2: Comparison of distribution we wish to sample from, p, and distribution we can sample from, q.

Notice that there are places where p is nonzero and q is zero, which means we have regions to sample that we aren't sampling at all.

Our estimate of the expectation of p becomes:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{p(x_i)}{q(x_i)}f(x_i) \rightarrow_{N\rightarrow\infty} E_p[f(x)] \tag{2}$$

2.1 Example

Set q as the motion model. Note that in reality we never use this – we use a better approximation of $p(x_t|x_{t-1})$ because the motion model takes too many samples to be reliable.

$$p = \eta \; p(z_t|x_t)p(x_t|x_{t-1}) \tag{3}$$

$$X^i \sim p(x_t|x_{t-1}) \tag{4}$$

$$w^i = \frac{p}{q} = p(z_t|x_t) \tag{5}$$

$$\bar{X} = \mathrm{E}[X] = \frac{1}{N}\sum_{i=1}^{N} w^i X^i \tag{6}$$

But notice that we dropped the normalizer from bayes rule! (if we don't know Z)

3 Normalized IS

We can normalize estimate using:

$$\frac{1}{N}\sum w^i \rightarrow_{N\rightarrow\infty} \sum q(x) = \sum p(x_t|x_{t-1})p(z_t|x_t) = Z \tag{7}$$

$$\tilde{w}^i = \frac{w^i}{\sum w^i} \tag{8}$$

$$\bar{X} = \sum \tilde{w}^i X^i \tag{9}$$

Algorithm 1 NIS Filter Algorithm

for all i **do**
  sample x_0^i from $p(x_0)$
end for
for all t **do**
  for all i **do**
    sample x_t^i from $p(x_t|x_{t-1}^i)$
    $w_i \quad *= p(z_t|x_t^i)$
  end for
  $\tilde{w}_i = \frac{w_i}{\sum w_i}$
end for
$E[f(x)] = \sum_i \tilde{w}_i f(x_i)$

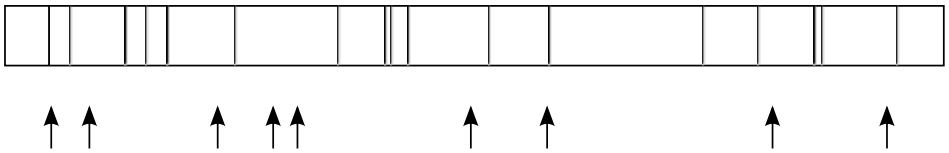


Figure 3: Resampling

4 SIR Filter (Particle filter, Condensation)

Unfortunately, in Importance Sampling, as time progresses, most particles become useless since they do not match the observations. The weights then go towards either 0 (most particles) or 1 (the few particles that match the observation). Essentially, we are using particles to keep track of parts of the state space that we know are not likely. The true path will be a sample with vanishing probability, so we have to throw out low probability samples and resample in high probability regions. This is known as a Particle or SIR filter.

4.1 Solution - Particle Filter (Sampling with Importance Resampling)

The solution is to do NIS, but to resample particles instead of changing the weights. This way, we do not waste particles on unlikely portions of the state space, but instead use our particles more intelligently.

Resampling proceeds as shown in Fig. 3: we select the particles with replacement according to their weight. Intuitively, we can think of this as lining the particles up on a dartbord that is 1 unit long, with width according to weight, and then throwing darts to select which ones to keep (possibly taking two copies of one particle). Having selected the particles, we set all their weights to $1/N$, since otherwise we are double counting the weights.

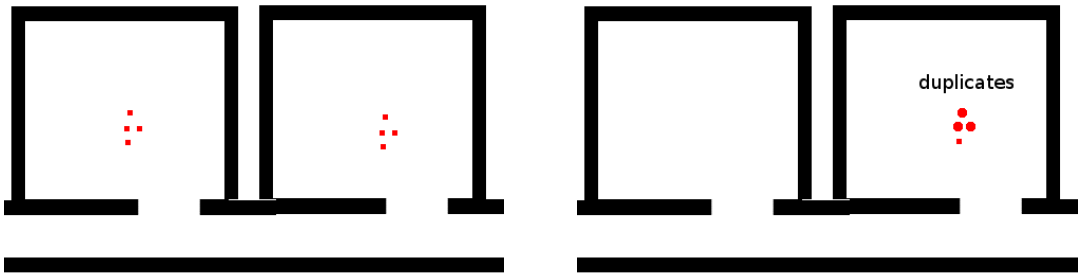
This approach has a variety of names: SIR, Particle Filter, CONDENSATION, etc., and can be applied to a variety of problems. All that a particle filter entails is sampling particles, reweighting them according to a belief, normalizing the weights, and then resampling. Thus, while this lecture primarily deals with localization, particle filters can be applied to a great deal of other problems.

5 The Bad (And Fixes)

In this section, we will discuss some shortcomings of particle filters, particularly in their naïve implementation, and a variety of fixes that are crucial for getting them to work in practice. Many of these fixes are very simple tweaks to the naïve implementation, and should be implemented in practice regardless of whether the associated bad behavior is observed.

5.1 Lack of Diversity

Suppose we start out with a distribution of particles as seen in Fig. 4(a): our filter has converged to two identical rooms, and is unsure which room a robot is in. Now, suppose we let the filter run for 100 timesteps, and check back. We observe Fig. 4(b). Our robot has not moved and is getting



(a) Two rooms with equal probability (b) The same filter, after repeated resampling

Figure 4: A problem with assuming the independence of the observations and using naïve sampling. Without gaining any new information, the particle filter converges on one of the rooms.

the same observations; however, over the course of 100 timesteps, the filter has become certain that it is in the right room.

What has happened? The filter is non-deterministic, and so when we pick the particles with equal weight, it is unlikely that we will equal numbers from each room. As time progresses, this selection process only becomes more imbalanced: with observations equally likely, if a state has very few particles in its vicinity, in expection, it will have very few particles after resampling. Further, once a state space loses its particles, there are no ways to regain them without motion happening.

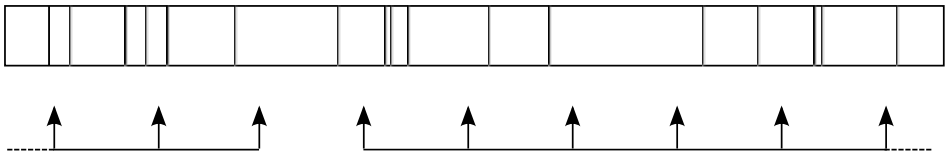


Figure 5: Low Variance Resampling; in practice, this approach is preferred over the naïve approach.

Fixes: There are a number of ways to prevent situations like this from happening in practice.

1. One way is to detect situations in which you do not have enough new information to resample, and only run the resampling when your observation model has sufficiently new observation data to process. One statistic that might be useful for evaluting this is looking at $\max(\{w_i\})/\min(\{w_i\})$, or looking at the entropy of the weights.

2. One easy fix is to run a more sophisticated resampling method, the low-variance resampler. In this approach, which is depicted in Fig. 5, a single number r is drawn, and then particles are selected at every $1/N$ units on the dartboard in our example. Effectively, the same sampler is used, but the random numbers generated are $r+i/N$ for all (positive and negative) i such that $r + i/N \in [0, 1]$. This has desirable properties: for instance, every particle with normalized weight over $1/N$ is guaranteed to be selected at least once. Due to these properties and its ease of implementation, it is always recommended that you implement this sampler rather than the naïve approach. Note that this is also known as the "Stochastic Universal Sampler", in the genetic algorithms domain.

3. One easy, but undesirable approach is to just inject uniform samples. This should be treated as a last resort as it is unprincipled.