Robot Localization Using Particle Filtering

Chaitanya R. Maniar, Pushyami Shandilya, Sampada Upasani

February 14, 2017

1 Introduction

The uncertainty in a robotic system is aptly modelled by using probabilistic techniques. Probability theory along with calculus explicitly explains the uncertainty. It makes use of the distribution over a space of possible hypotheses to come to an approximate estimate of the required states, which encapsulates the ambiguity and belief associated with it. Localization is the problem of estimating a robot's coordinates in an external reference frame from sensor data, using a map of the environment. The robot is placed somewhere in the environment and has to localize itself from scratch.

2 Methodology

The two main type of filters that we can use are the parametric (Gaussian techniques) and non-parametric filters. As opposed to the parametric filters, the non-parametric filters do not rely on a fixed functional form of the posterior. The posterior is instead estimated using a finite number of values in the state space. The quality of the approximation depends on the number of parameters used to represent the posterior. As the number of parameters goes to infinity, nonparametric techniques tend to converge uniformly to the correct posterior (under specific smoothness assumptions).

A particle filter does not make strong assumptions on the posterior density. In particular, it well-suited for to represent complex multimodal beliefs, which is the reason why this method is being used for the problem at hand. The particle filter is an alternative nonparametric implementation of the Bayes filter. The key idea of the particle filter is to represent the posterior $bel\left(x_t\right)$ by a set of random state samples drawn from this posterior. The samples of a posterior distribution are called particles and are denoted by $X_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}$ where M is the number of particles. Each particle $x_t^{[m]}, \left(1 \leq m \leq M\right)$ is a constant instantiation of the state time t, that is, a hypothesis as to what the true world state may be at time t.

The main components of the particle filtering algorithm are detailed in the next section. An outline to that procedure is as follows (in no particular order):

i) Distribution Sampling: The motion model $p\left(x_t|u_t, x_{t-1}\right)$ is sampled, instead of computing the posterior for arbitrary $p\left(x_t|u_t, x_{t-1}\right)$. The odometry are considered here to sample from, i.e., their values are weighted against zero-mean gaussians that are formed using the random function. These sampled values are then used to compute the current position and heading using the values at $t-1$.

ii) Reading in data from map and log files (odometry and laser range data): The known data from the world are obtained using the two sensors- odometer and laser range finder, as well as the knowledge of the map. The measurements obtained from them have been put into a log data and map file respectively, which are read in for our computation. The two files used to read the raw data and convert it into usable form for further steps are *logParser* and *mapParser*.

iii) Motion Model: We use the odometry motion model here, instead of the velocity motion model as it has some advantages as opposed to the former model. The motion model basically describes how the states (x, y, θ) are changing over time and their dependencies with the other variables.

iv) Measurement Model: The measurement model here encapsulates the way the sensors read in the appropriate information, and how the sensors are modeled in order to compensate/account for the measurement noise. The noise, as we find can arise due to a lot of different reasons, and considering the possibility of it makes our model more robust to such conditions.

v) Re-sampling: Low Variance sampling is incorporated in this implementation of the particle filter. The particles are given weights according to the algorithm detailed in the next section. The particles need to be re-weighted each time in order to gradually move toward the best possible solution to the localization problem.

3 Implementation

3.1 Sampling Distribution

If particle filters are used for localization, we have an algorithm for sampling from $p\left(x_t|u_t, x_{t-1}\right)$. The particle filter requires a sample of the posterior state probability rather than a closed-form expression for computing the same. The sample motion model odometry algorithm that we have used in our code accepts initial pose x_{t-1} and an odometry reading u_t as input, and outputs a random x_t distributed according to $p\left(x_t|u_t, x_{t-1}\right)$. The pose x_t is randomly guessed.

3.2 Motion Model

Velocity motion model has some shortcomings, and hence we've used the odometry motion model instead, because the former uses the robot's velocity to compute posteriors over poses.

Alternatively, one might want to use the odometry measurements as the basis for calculating the robot's motion over time. Odometry is obtained by integrating wheel encoders information, estimation of which is available in periodic time intervals. The rotational and translational odometry readings are calculated from the poses x_t and x_{t-1}. Using the inverse motion model, the error distribution over a zero-mean gaussian is calculated. All the angular differences lie in $[-\pi, \pi]$. If this is not followed, the outcome of the rotational error also has to be correspondingly truncated but this often leads to divergence. Finally we assume independence between the different error sources.

3.3 Measurement Model

The measurement is modelled taking into account the different ways in which noise can be included in the readings. They are as follows:

i) Correct range with local measurement noise: In an ideal world, a range finder would always measure the correct range to the nearest object in its measurement field. Let us use z_t^{k*} to denote the "true" range of the object measured by z_t^k. In location-based maps, the range z_t^{k*} can be determined using ray casting. However, even if the sensor correctly measures the range to the nearest object, the value it returns is subject to error. This error arises from the limited resolution of range sensors, atmospheric effect on the measurement signal, and so on. This noise is usually modeled by a narrow Gaussian with mean z_t^{k*} and standard deviation σ_{hit}. We will denote the Gaussian by p_{hit}. In practice, the values measured by the range sensor are limited to the interval $[0; z_{max}]$, where z_{max} denotes the maximum sensor range. Thus, the measurement probability is given by $p_{hit}\left(z_t^k|x_t, m\right) = \eta N(z_t^k; z_t^{k*}, \sigma_{hit}^2)$ if $0 \leq z_t^k \leq z_max$ where z_t^{k*} is calculated from x_t and m via ray tracing, and $N(z_t^k; z_t^{k*}, \sigma_{hit}^2)$ denotes the univariate normal distribution with mean z_t^{k*} and variance σ_{hit}

The variance σ_{hit} is an intrinsic noise parameter of the measurement model. Below we will discuss strategies for setting this parameter.

ii) Unexpected objects. Environments of mobile robots are dynamic, whereas maps m are static. As a result, objects not contained in the map can cause range finders to produce surprisingly short ranges—at least when compared to the map. A typical example of moving objects are people that share the operational space of the robot. One way to deal with such objects is to treat them as part of the state vector and estimate their location; another, much simpler approach, is to treat them as sensor noise. Treated as sensor noise, unmodeled objects have the property that they cause ranges to be shorter than z_t^{k*}, not longer. More generally, the likelihood of sensing unexpected objects decreases with range. To see, imagine there are two people that independently and with the same, fixed likelihood show up in the perceptual field of a proximity sensor. One person's range is z_1, and the second person's range is z_2. Let us further assume that $z_1 < z_2$, without loss of generality. Then we are more likely to measure z_1 than z_2. Whenever the first person is present, our sensor measures z_1. However, for it to measure z_2, the second person must be present and the first must be absent. Mathematically, the probability of range measurements in such situations is described by an exponential distribution. The parameter of this distribution, λ_{short}, is an intrinsic parameter of the measurement model. .

3.4 Resampling

As mentioned in the previous section, the resampling is done in order to weight the particles after every iteration in order to get as close as possible to the optimal solution. Intuitively, this means that given our knowledge of the previous state and output corresponding to it, each time we give the particles new weights so that iterations converge. There are two ways in which the resampling can be done: Roulette wheel which implements the binary search algorithm, which has a time complexity of $O(nlogn)$; the other is Stochastic universal sampling which implements the low variance sampling and has a time complexity of $O(n)$. In the low variance sampling, when we have M particles, we choose a random number and select those particles which correspond to $u = r + (m-1)\,M^{-1}$.

In our program, we have used the low variance sampling since it has advantages as compared to binary search, and they are that: more systematic coverage of space of samples; if all the samples have same weights, no samples are lost; lower computational complexity.

3.5 Parameter Tuning

In order to integrate all the above models into one main program, there are a lot of parameters that need to be tweaked. The performance of the algorithm depends on these values. The above listed models and the corresponding parameters that we have used in our implementation of the particle filter are as follows.

i) Motion Model: The parameter considered here is alpha, which is an array of four values. The value of alpha decides by how much the state variables undergo rotation and translation.

$$\alpha = [0.05, 0.05, 0.1, 0.1]$$

ii) Measurement Model: The intrinsic parameters associated with the measurement model have been tuned by trial-and-error and the best guess for the particular proble. Our model has the following parameters:

$$z_{Hit} = 0.67, z_{Short} = 0.13, z_{Rand} = 0.19, z_{Max} = 0.01, \sigma_{Hit} = 2,$$
$$\lambda_{Short} = 0.03, laserMax = 1000$$

The Parameters, z_{Hit}, z_{Short}, z_{Rand}, z_{Max} correspond to the probability distributions as explained in Section 3.3. z_{Hit} gets the most weight because we are certain that the most noise is concentrated around the correct value for the sensor reading, and hence it gets the most weight. It is a Gaussian distribution with zero mean and a variance of 2. Rest of the three parameters are weighted lesser because the noise associated with those is relatively lesser. We observe that the least weight is given to z_{Max} as the sensor is continually measuring and is assumed to be rather accurate. $LaserMax$ is the maximum allowable reading from the laser range finder.

iii) Particle Parameters: The algorithm performance is strongly related to the number of particles that are used in the belief calculations. We have used 100 particles, as this gives a better accuracy. The down-sampling rate is set at 10. The down-sampling rate is what is sent as an input to the ray casting function. It describes the number of values that we choose out of the whole space of readings to get the best result. A down-sample rate of 10, here means that out of the 180 readings (one reading for every degree in rotation), we are choosing readings at an interval of $(180/10) = 18$ degrees. The lesser the downsampling rate, the better the performance, but this is a trade-off between computation time and accuracy.

4 Results and Future Work

We have successfully implemented the particle filtering algorithm for robot localization. This algorithm localizes randomly generated particles to a particular location, which is the best estimate of the robot position on the map. During the course of implementing the particle filtering algorithm, we encountered a lot of minor bugs in the algorithm, mainly due to the errors in approximation and perhaps a few shortcomings in the individual algorithms. There are a couple of ways in which this implementation can be furthered and improved. First of which is trying to writing in better models for each of the modules- motion and measurement. In our algorithm, we have used the beam range finder model, whereas we can use the likelihood range finder model instead. Another improvement can be by making the algorithm in terms of choosing the number of particles as well as choosing appropriate parameters. In the case of choosing parameters, we can make the algorithm adaptively adjust and use only those many number of particles as required and no more- this reduces the time taken for computation as well as the memory space usage. The second method to make the system more robust by explicitly learning the intrinsic parameters for modeling the sensors. We have used the values for parameters only by trial and error, and not learning from the given data.

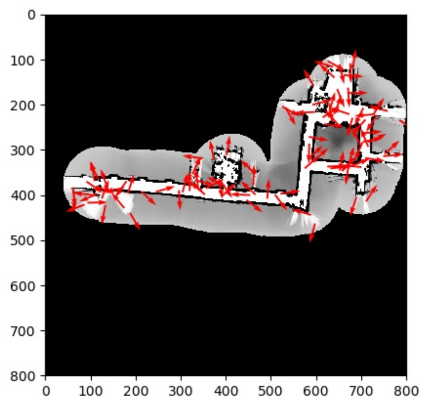

Figure 1: Initially generated particles

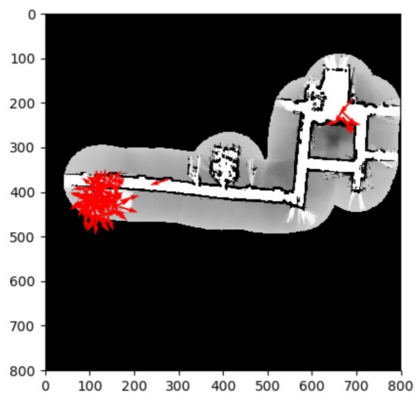

Figure 2: Particles after filtering