ME8135 Assignment 3

Glenn Shimoda

November 2020

1 Introduction

1.1 Overview

The purpose of this assignment is to simulate a robot moving in a circle around a landmark using the Particle Filter algorithm. The same setup is used from HW2, where both a linear and nonlinear measurement model is used. The system was simulated using pygame in Python.

1.2 Monte Carlo Method

In section 4.2.3 of the text and HW2,we used nonlinear and/or non-Gaussian models and approximately linearized them using the Jacobians about an operating point, which led to the Extended Kalman Filter. Although the EKF algorithm is effective, it is only valid when the distribution of the models and/or noise are approximately Gaussian(such as only slightly skewed) [1].

Instead of making the models approximately linear about an operating point, there are multiple methods to pass the probability density functions through nonlinearities. Barfoot presents the 3 most common approaches:

1. **Monte Carlo Method(brute force)** - this method is straightforward but the most computationally expensive. It simply samples every single point from an unknown density and passes them directly through the nonlinearity to build the resulting density with enough samples. This method is obviously the most computationally expensive but with faster processors, it is more feasible to use. It can work with any distribution and doesn't have to make assumptions, making it the most accurate method and is usually bench mark to evaluate other methods.
2. **Linearization(as in the EKF)** - this is the same method used in section 4.2.3 and HW2. It is simple and works well with approximately Gaussian distribution, but can quickly get skewed results with different distributions and needs a closed-form or approximate computation of the Jacobians.

3. **Sigmapoint(or unscented) transformation** - a method to sample from the unknown input density and then calculate new parameters which can approximate a new distribution. Example 4.1 on page 112 of the textbook shows an example demonstrating how sigmapoint transformation is more accurate than linearization at approximately the same computational cost. Section 4.2.9 explains the sigmapoint Kalman filter.

1.3 Particle Filter

The Particle Filer is an state estimation method that uses the Monte Carlo sampling method. Its main advantages/disadvantages are characterized by the Monte Carlo method itself, in that it's very accurate but the most computationally expensive, so parameters, such as the number of samples, is determined to get the best accuracy/timing trade-off.

The basic PF algorithm can be explained in the following steps:

1. Draw M samples from the joint density comprising the prior and the motion noise:

$$\begin{bmatrix} \hat{\mathbf{x}}_{k-1,m} \\ \mathbf{w}_{k,m} \end{bmatrix} \leftarrow p(\mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{v}_{1:k-1}, \mathbf{y}_{1:k-1}) p(\mathbf{w}_k) \tag{1}$$

   Where m is the particle index, $m \in [0, M]$ and k is the time index.

2. Generate a prediction of the posterior PDF by using the latest input. This is done by passing each prior/particle noise sample through the nonlinearity:

$$\check{\mathbf{x}}_{k,m} = f(\hat{\mathbf{x}}_{k-1,m}, \mathbf{v}_k, \mathbf{w}_{k,m}) \tag{2}$$

   These new predicted particles together approximate the density:

$$p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{v}_{1:k}, \mathbf{y}_{1:k-1}) \tag{3}$$

3. Correct the pdf by incorporating the measurements. First, assign a weight for each particle m from 1 to M:

$$w_{k,m} = \frac{p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{v}_{1:k}, \mathbf{y}_{1:k})}{p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{v}_{1:k}, \mathbf{y}_{1:k-1})} = \eta p(\mathbf{y}_k \mid \check{\mathbf{x}}_{k,m}) \tag{4}$$

   In the textbook, the right side of equation 4 with a normalized probability is used, expanded on from equation 4.68 using a noise-less measurement model as approximation. Here, we will directly evaluate the weight as the fraction of the desired posterior and predicted posterior, which is the first equality in equation 4.

4. Resample: Use Maddow's method to re-sample by creating bins based on the weights:

$$\beta_m = \frac{\sum_{n=1}^{m} w_n}{\sum_{l=1}^{M} w_l} \tag{5}$$

Where the bins define the M boundaries on the interval [0, 1]: $0 \leq \beta_1 \leq \beta_2 \leq ... \leq \beta_{M-1} \leq 1$, and $\beta_M = 1$. Given these bins, select a random number ρ, sampled from a uniform density on [0, 1). This will fall in a bin β_m, and we choose the sample from the original prior corresponding to weight index m. Repeat this procedure M times by adding 1/M to ρ every time, and a new density that is more dense is drawn. This new sample is then carried forward onto the next iteration, and the particles quickly converge from scattered initially to dense as the robot uses its measurements and dead reckoning to pinpoint out of all the initial random guesses, which state it is most likely to be in.

2 Problem

2.1 Setup

The problem setup is the same as homework 2with the same robot parameters and a landmark L is placed at the coordinates (10, 10). They are restated here:

$$\dot{\theta}_k = \frac{r}{L} u_\psi + w_\psi \tag{6}$$

$$\theta_k = T\dot{\theta}_k + \theta_{k-1} \tag{7}$$

$$\dot{\mathbf{X}}_k = \begin{bmatrix} \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} ru_\omega cos(\theta_k) + w_\omega \\ ru_\omega sin(\theta_k) + w_\omega \end{bmatrix} \tag{8}$$

$$\mathbf{X}_k = T\dot{\mathbf{X}}_{k-1} + \mathbf{X}_{k-1} = f(x_{k-1}, v_K, w_k) \tag{9}$$

As for the measurement, they can be the linear model in HW1 or the non-linear version in HW2:

$$\phi_k(\theta_k, n_\phi) = \frac{\pi}{2} + \theta_k + n_\phi \tag{10}$$

$$d(\mathbf{X}_k, n_d) = d(x_k, y_k, n_d) = \sqrt{(x_k - 10)^2 + (y_k - 10)^2} + n_d \tag{11}$$

$$\mathbf{z}_k = \begin{bmatrix} 10 + dcos(\phi_k) \\ 10 + dsin(\phi_k) \end{bmatrix} = g(\mathbf{X}_k, n_k) \tag{12}$$

2.2 Algorithm

Since no approximation is used and the Monte Carlo sampling method is simply brute force, the math and algorithm in the Particle Filter is fairly simple to describe.

1. **Initialize the State, Variables and Guesses.** - the initial state $\mathbf{X}_0$ is (6,10), which is 4 meters to the left of the landmark, while being at the same y coordinate. 100 particles(M = 100) are initiated around the (6, 10) coordinates with the mean being the nominal (x, y) values, with standard deviation of 3 for both. This creates an initial scatter of filters that is a reasonable distribution of initial guesses, but it can be any distribution. The initial angle $\theta_0 = \frac{\pi}{2}$ which indicates the robot is facing upwards in the positive y direction. Set the constant control variables as $u_r = 4.75, u_l = 5.25$. These were developed from the zero noise model to circle around the landmark with a distance around 4m at all times.

2. **Update the motion model and create the prior distribution** - directly using equations 6-9. Since each particle is passed through this state, no approximation is needed and the nonlinear equation 8 can be used to bring the theta from its Gaussian distribution to the new distribution. Create the approximate distribution of the prior given by equation 3, by building histograms with M/10 bins. Since 100 particles are used, 10 bins are used. Then a list corresponding to the number of occurrences in each bin is used as the probability of a given $\mathbf{X}_k$ being inside the distribution.

3. **Create the measurement distribution** - using the 100 values from step 2, pass each of them through equation 12 and use the same steps to build a measurement distribution out of the 100 particles. A list of probability for each bin is also created here.

4. **Calculate the weights** - calculate the weights $w_{k,m}$ using equation 4 for each $\mathbf{X}_k$ calculated in equation 2. The numerator of the equation shows the probability of the selected value being in the distribution created in step 4 and uses the list of probability of bins calculated there. Since the $\mathbf{X}_k$ values here are different from the values used to create the distribution in step 3, some values may not lie inside the distribution, so their probability will be zero. The denominator of the equation is the probability of $\mathbf{X}_k$ being in the distribution built in step 2. Since that distribution was built with the $\mathbf{X}_k$ values, there is no possibility of $\mathbf{X}_k$ not being in the distribution(which would result in a zero in the denominator and an invalid value).

5. **Use Maddow's method to re-sample** - use the weights from step 4 to calculate the bins in equation 5. Then, initialize a random ρ value between 0 and 1, and choose the weight and $\mathbf{X}_k$ value associated with the bin that ρ falls on. Then increase ρ by 1/M and repeat this process 100 times to re-sample 100 $\hat{\mathbf{X}}_{k,m}$ values. The resulting distribution should be more concentrated on specific values.

6. **Repeat** - pass the distribution from step 5 onto step 1 in the next iteration and keep repeating as the robot traverses. The particles should converge over time as the robot moves more and gets more measurements, as it becomes more certain of its state.

3 Results

Figure 1 shows the initial particles, and 2 shows the particles after enough iterations to have them converge. The red square is the ground truth(i.e. the actual robot path with zero noise). As can be seen, the algorithm works well to quickly narrow the search for the most probable robot position. Many simplifications were used in this code, which can be expanded upon.

First, the particles were initially normally distributed, but as stated before, they can be any distribution and any type of guess. Second, in addition to the (x,y) coordinates, the angles were also sampled and passed on to the next iteration. Third, even though the true distribution can be created, computationally it is still in a discrete histogram form, where the probabilities have discrete jumps between bins. To get the most advantage out of the particle filter, more particles(possible in the thousands) will be needed with many bins to get a more smooth-looking distribution.

Additionally, tips pointed out by Barfoot can improve the performance of the filter. The first is to manually sample a portion of the initial particle from a uniform distribution which protects against outliers which can skew the final position that the particles converge to. Resampling can also be delayed by only moving weights forward to speed up computation and possibly increase the number of particles. Finally, a heuristic can be used to limit the number of particles to re-sample in step 5 if the cumulative weights exceeds a threshold. This way, the particles are limited to ones that we are confident have a high probability. This increases both accuracy and computation.

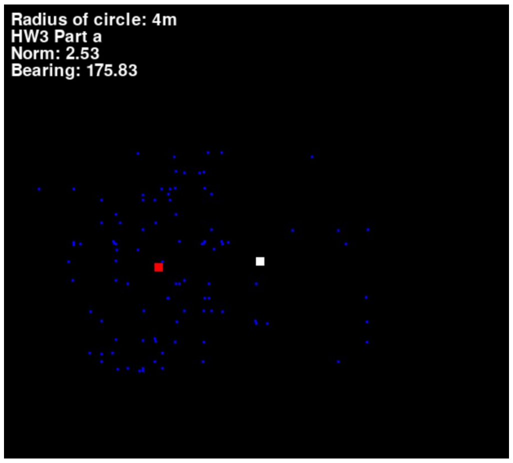

Figure 1: Initial Particle Filters

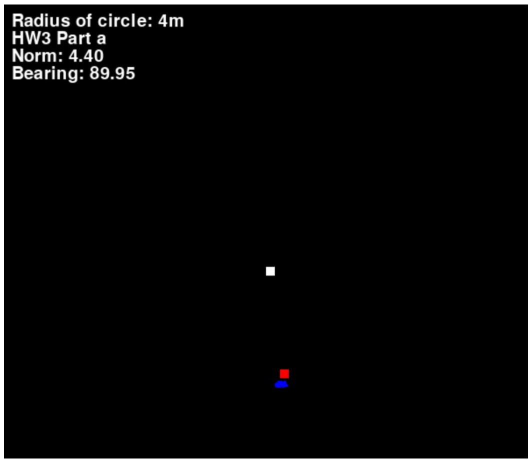

Figure 2: Converged Particle Filters

References

[1] T. D. Barfoot, *State Estimation for Robotics.* Cambridge University Press, 2020.