#Junseo Kim

Assignment 3

Summer 2023

1 Particle Filter (Linear model)

Use pyGame, or any other similar libraries, to simulate a simplified 2D robot and perform state estimation using a Kalman Filter. Motion Model:

$$\dot{x} = \frac{r}{2}(u_r + u_l) + w_x \quad , \quad \dot{y} = \frac{r}{2}(u_r + u_l) + w_y$$

r = 0.1 m, is the radius of the wheel, u_r and u_l are control signals applied to the right and left wheels. $w_x = N(0, 0.1)$ and $w_y = N(0, 0.15)$

Simulate the system such that the robot is driven 1 m to the right. Assume the speed of each wheel is fixed and is 0.1 m/s

Use these initial values

$$x_0 = 0 \ , \ y_0 = 0 \ , \ P_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \text{ (initial covariance matrix)}$$

and assume the motion model is computed 8 times a second. Assume every second a measurement is given:

$$z = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} r_x & 0 \\ 0 & r_y \end{bmatrix}$$

where $r_x = N(0, 0.05)$ $r_x = N(0, 0.075)$

Initialization

For the particle filter algorithms, particles each representing a possible state are set in the algorithm. Each particle is then assigned with initial weight.

$$\begin{bmatrix} \hat{\mathbf{x}}_{k-1,m} \\ \mathbf{w}_{k,m} \end{bmatrix} \leftarrow p\left(\mathbf{x}_{k-1} | \check{\mathbf{x}}_0, \mathbf{v}_{1:k-1}, \mathbf{y}_{1:k-1}\right) p(\mathbf{w}_k)$$ [2]

Prediction

Particles move based on the control input and the motion model as given in the requirement. The motion model will include noise to account for the uncertainties in the system.

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

Correction

Each particle is then updated based on how they predicted well compared to the actual measurement. This is calculated with the likelihood (Gaussian). Particles with better prediction will get a higher weight.

$$w_{k,m} = \frac{p\left(\check{\mathbf{x}}_{k,m} | \check{\mathbf{x}}_0, \mathbf{v}_{1:k}, \mathbf{y}_{1:k}\right)}{p\left(\check{\mathbf{x}}_{k,m} | \check{\mathbf{x}}_0, \mathbf{v}_{1:k}, \mathbf{y}_{1:k-1}\right)} = \eta\, p\left(\mathbf{y}_k | \check{\mathbf{x}}_{k,m}\right) \quad [2]$$

Resampling

Once the weights are updated, particles are resampled. This is done for each particle proportional to its weight which also means that particles with higher weights are more likely to be resampled.

$$\hat{\mathbf{x}}_{k,m} \overset{\text{resample}}{\longleftarrow} \{\check{\mathbf{x}}_{k,m}, w_{k,m}\} \quad [2]$$

Estimation

The new state of the system is then estimated with the average of the weighted particles.

Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
- $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
- for $m = 1$ to M do
  - sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
  - $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
  - $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
- endfor
- for $m = 1$ to M do
  - draw i with probability $\propto w_t^{[i]}$
  - add $x_t^{[i]}$ to $\mathcal{X}_t$
- endfor
- return $\mathcal{X}_t$

Figure 1: Particle Filter Algorithm [1]

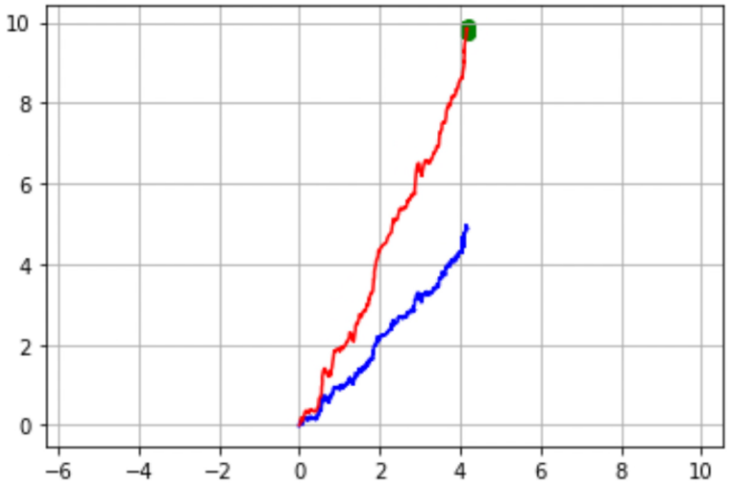

Figure 2: Particle Filter with the linear measurement model

In Figure 2, the red line represents the estimated states with particles and the blue line represents the true state of the system.

The true state is updated in the 'simulate_observation' function from the code and the noise is added to simulate the real-world conditions

The estimated state of the system is based on the observations and control inputs. The estimated state is updated in the 'particle_filter_localization' function from the code. The particles are moved based on the control inputs and then weighted based on how they predict the observations.

In the simulation, only 100 particles were used. Even though the particle filter is not the best solution to solve the linear measurement model, the accuracy of the particle filter will increase with the increase in the number of particles and decreasing the amount of noise.

For this linear measurement model, Gaussian distribution is used as the likelihood function in the particle filter.

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Where μ is the mean and b is the standard deviation.

In the resampling process, a large number of particles are drawn from this distribution. Due to the central limit theorem, random variables tend towards a Gaussian distribution regardless of the shape of the original distribution.

However, other distributions can be used for the likelihood function. For part 2 using the non-linear model, double exponential distribution (Laplace distribution) is used to deal with the Non-gaussian model.

Detailed Codes can be found at
https://github.com/JunseoKim19/State_estimation/blob/main/Particle_Filter_taska.py

Full simulation can be found at
https://youtu.be/3Md9g_VARHo

2 Particle Filter (Non-linear model)

Repeat the previous assignment, this time with a classic motion model and range observations made from a landmark located at M = [10,10]. L is the distance between the wheel, known as wheelbase, and is 0.3m.

$$\dot{x} = \frac{r}{2}(u_r + u_l)\cos(\theta) + w_x \quad , \quad \dot{y} = \frac{r}{2}(u_r + u_l)\sin(\theta) + w_y \quad , \quad \dot{\theta} = \frac{r}{L}(u_r - u_l).$$

Assume

$$u_\omega = \frac{1}{2}(u_r + u_l) \quad , \quad u_\psi = (u_r - u_l)$$

Then the equations become:

$$\dot{x} = r u_\omega \cos(\theta) + w_\omega \quad , \quad \dot{y} = r u_\omega \sin(\theta) + w_\omega, \quad \dot{\theta} = \frac{r}{L} u_\psi + w_\psi$$

$w_\psi = N(0, 0.01)$ and $w_\omega = N(0, 0.1)$. Program the robot such that it loops around point M.

The main difference from Part 1 is the system model, and control inputs because a non-linear model is applied to the system. For a non-linear model system, new control input: yaw rate is applied to the system.

The observation model will, therefore, involve the distance and angle measurement from the landmark to the robot's position and orientation.

In terms of the noise, as the non-linear model is applied, noise should be included in both velocity and yaw rate for accurate estimation.

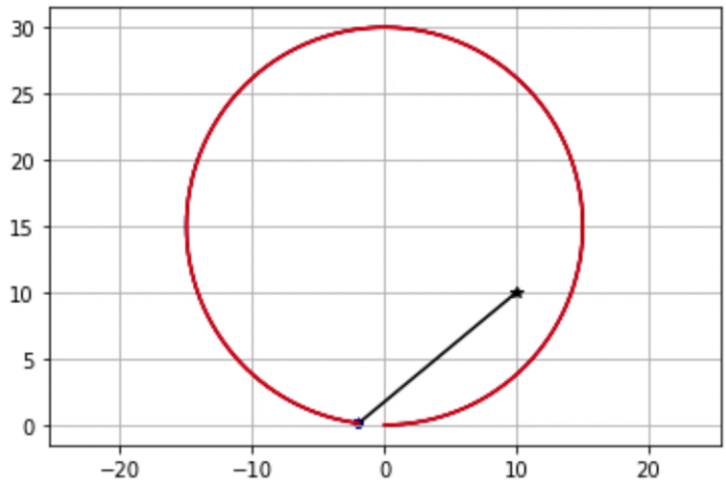

Figure 3: Particle Filter for the non-linear model

From Figure 3, the ground state (blue line) is hidden under the estimated state (red line) and is not visible through the simulation.
As for the likelihood, double exponential distribution (Laplace distribution) was used for the system

$$f(x|\mu, b) = \frac{1}{2b} exp\left(-\frac{|x-\mu|}{b}\right)$$

Where μ is the mean and b is the standard deviation.

Compared to the Gaussian distribution, the Laplace distribution will give more likelihood to larger deviations from the mean and will be more concentrated around the mean.

However, after running the simulation the difference between two likelihood functions were minimal.

The detailed process of the particle filter implementation is described in Part 1.

Detailed Codes can be found at
https://github.com/JunseoKim19/State_estimation/blob/main/Particle_Filter_taskb.py

Full simulation can be found at
https://youtu.be/S6kPBNh-YTk

Reference

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2010.

[2] T. D. Barfoot, *State Estimation for Robotics*. Cambridge, United Kingdom: Cambridge University Press, 2017.