State Estimation - Assignment 3

Parham Jafary

June 12, 2023

Abstract

In this assignment, our objective is to perform position estimation using a Particle Filter (PF) for a linear and a non-linear system respectively. Our system is a simplified robot and we have measurement models instead of sensors. Randomness (noise) was added both to the robot's motion model (representing the actuator noise) and to the measurement model (representing the sensor noise). The code is written in Python language and the PyGame library is used for simulation and visualization purposes.

1 Linear System

1.1 Question

In this question we are given the 2D robot model below and we are asked to simulate it and perform state estimation using a Particle Filter.

$$\begin{gathered}\dot{x} = \frac{r}{2}(u_r + u_l) + w_x \ , \ \dot{y} = \frac{r}{2}(u_r + u_l) + w_y \\ w_x = N(0, 0.1) \ , \ w_y = N(0, 0.15) \ , \ r = 0.1\end{gathered} \tag{1}$$

the speed of the wheel is constant and equal to $0.1m/s$ and our initial values for x, y, and covariance matrix are:

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad , \quad P_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2}$$

and our measurement is given by:

$$\begin{gathered} z = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} r_x \\ r_y \end{bmatrix} \\ r_x = N(0, 0.05) \\ r_y = N(0, 0.075) \end{gathered} \tag{3}$$

1.2 Solution

Particle filtering, also known as sequential Monte Carlo (SMC) methods, is a technique used for state estimation or inference in dynamic systems. It is particularly useful when the system's evolution and measurements are nonlinear and/or non-Gaussian in nature. The overall picture for this question is that we have a robot model, a sensor model, Gaussian noise for the actuators, and Gaussian noise for the sensor and the objective is to create a PF to estimate the position of the robot. In order to do so, we need to form the position equations and the PF algorithm, the required steps are described below. The first step is the motion model; we have the $\dot{x}, \dot{y}$ therefore we need to integrate to calculate the x, y. Since we have constant speed the integration can be done by multiplying the $\dot{x}, \dot{y}$ by δt . Using Eq. 1, we can write the state of our robot in matrix form as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\delta t/2 & r\delta t/2 \\ r\delta t/2 & r\delta t/2 \end{bmatrix} \begin{bmatrix} u_r \\ u_l \end{bmatrix} + \begin{bmatrix} w_x \delta t \\ w_y \delta t \end{bmatrix} \tag{4}$$

Now that we have x, y we need to discretize the Eq. 4 to make it suitable for coding purposes. We also need to add the previous position to the equation.

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + \begin{bmatrix} r\delta t/2 & r\delta t/2 \\ r\delta t/2 & r\delta t/2 \end{bmatrix} \begin{bmatrix} u_r \\ u_l \end{bmatrix} + \begin{bmatrix} w_x^t \delta t \\ w_y^t \delta t \end{bmatrix} \tag{5}$$

Since the speed of the wheels is constant, we can solve for out control inputs and find them to be $u_r = u_l = 1$. To plot the true position, we can omit the noise in the Eq. 5

The next step is the sensor or the so called measurement model. Discretizing Eq. 3 gives our measurment model and since the measurement does not rely on the the previous data, the equation stays exactly the same but with a t index.

The PF algorithm as suggested in the Probabilistic Robotics reference is:

$$\begin{aligned}
&\bar{X}_t = X_t = 0 \\
&for\ m\ in\ M\ do: \\
&\quad sample\ x_t^{[m]} \approx p(x_t | u_t, x_{t-1}^{[m]}) \\
&\quad w_t^{[m]} = p(z_t | x_t^{[m]}) \\
&\quad \bar{X}_t = \bar{X}_t + < x_t^{[m]}, w_t^{[m]} > \\
&\\
&for\ m\ in\ M\ do: \\
&\quad draw\ i\ with\ probability\ \propto\ w_t^{[i]} \\
&\quad add\ x_t^i\ to\ X_t \\
&\\
&return\ X_t
\end{aligned} \tag{6}$$

based on the values given in the question we have:

$$\begin{aligned}
A_t &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad , \quad B_t = \begin{bmatrix} r\delta t/2 & r\delta t/2 \\ r\delta t/2 & r\delta t/2 \end{bmatrix} \quad , \quad R_t = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.15 \end{bmatrix} \\
C_t &= \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad , \quad Q_t = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.75 \end{bmatrix}
\end{aligned} \tag{7}$$

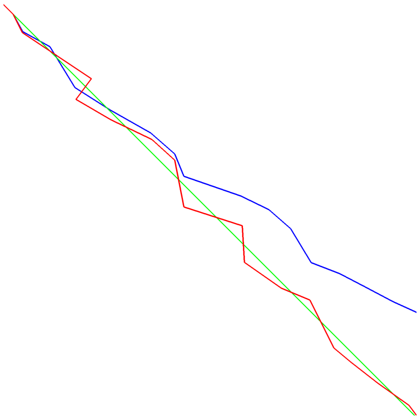
For particle filter algorithm, the first step is to generate random particles first. I used random normal distribution with Q_t as its covariance for generating particles. then we need to associate a probability distribution to the particles so we can take the most likely ones. I used a Gaussian PDF for the particles. Then we need to associate weights to the particles to determine the ones with higher possibility. Finally, we need to re-sample the particles that have higher weights.
Simulating the system with the above value yields the following figures for the predicted trajectory, measured position, predicted position, and calculated covariance (the video clip is uploaded at GitHub).

In Fig. 1-a, we can see the predicted, true, and measured trajectories, in blue, green, and red colors respectively. In sub-figure a) we can see the measured x, y positions and in sub-figure c) we can see the predicted x, y positions.

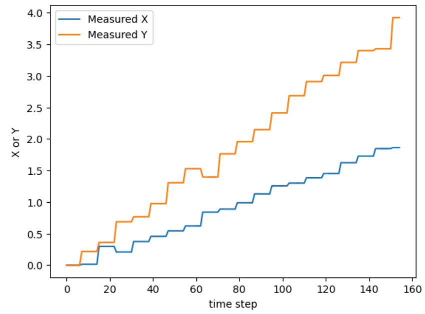
2 Nonlinear System

2.1 Question

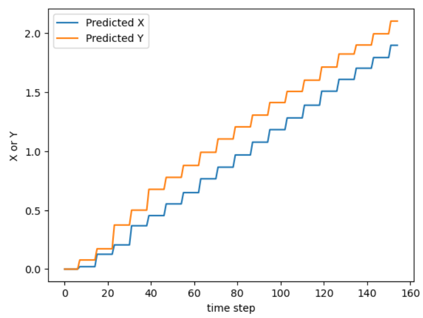
Repeat the previous assignment, this time with a classic motion model and range observations made from a landmark located at M = [10,10]. L is the distance between the wheel, known as wheelbase,



(a) Predicted, Measured, and True trajectories



(b) Measured Position



(c) Predicted Position

Figure 1: Results from Particle Filter

and is 0.3m. Program the robot such that it loops around point M

$$\begin{aligned}
&\dot{x} = \frac{r}{2}(u_r + u_l)\cos\theta + w_x \ , \ \dot{y} = \frac{r}{2}(u_r + u_l)\sin\theta + w_y \ , \ \dot{\theta} = \frac{r}{L}(u_r - u_L) \\
&u_\omega = \frac{1}{2}(u_r + u_l) \ , \ u_\psi = (u_r - u_l) \\
&\dot{x} = r u_\omega \cos\theta + w_\omega \ , \ \dot{y} = r u_\omega \sin\theta + w_\omega \ , \ \dot{\theta} = \frac{r}{L} u_\psi + w_\psi \\
&w_\psi = N(0, 0.01) \ , \ w_\omega = N(0, 0.1) \ , \ r = 0.1
\end{aligned} \tag{8}$$

(a) Compute the PF with the linear measurement model in the previous assignment.
(b) Compute the PF with range/bearing measurements of point M. Assume range noise is N(0,0.1) and bearing noise is N(0,0.01). Range is in meters, and bearing is in radians. Visualize the measurements as well.

2.2 Solution

We are asked to make the robot to go around point M around. The procedure is the same as part 1 but the motion model is different, we start by discretizing, integrating the state variables, and writing the state equations in matrix format:

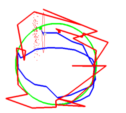
$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} r\cos\theta\delta t & 0 \\ r\sin\theta\delta t & 0 \\ 0 & \frac{r}{L}\delta t \end{bmatrix} \begin{bmatrix} u_\omega^t \\ u_\psi^t \end{bmatrix} + \begin{bmatrix} w_\omega^t \delta t \\ w_\omega^t \delta t \\ w_\psi^t \delta t \end{bmatrix} \tag{9}$$

We want to program the robot to stay on a circular trajectory around point $M = (10, 10)$, and the starting position is $x_0 = (0, 0)$ therefore, left and right wheels' control input should be adjusted to keep a distance of 10 from M.

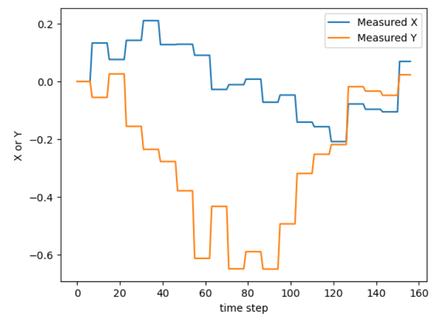
For part a, since the sensor's model is the same as question 1, the rest of the algorithm is the same but the matrices are different as shown in Eq. 10.

$$
\begin{aligned}
A_t &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad B_t = \begin{bmatrix} r\delta t \cos\theta & 0 \\ r\delta t \sin\theta & 0 \\ 0 & r\delta t/L \end{bmatrix} \quad , \quad R_t = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.01 \end{bmatrix} \\
C_t &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad Q_t = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.75 \end{bmatrix}
\end{aligned} \tag{10}
$$

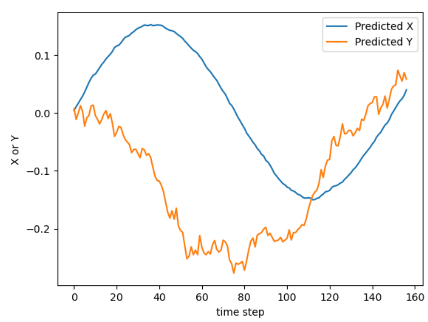
The results from the simulation are illustrated in Fig.2.



(a) Predicted, Measured, and True trajectories



(b) Measured Position



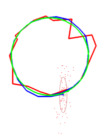
(c) Predicted Position

Figure 2: Results from Particle Filter with the linear measurement model

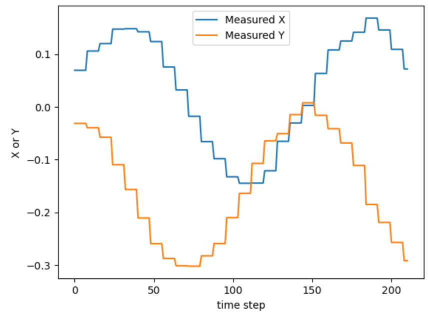
We can see that PF performs poorly, when trying to estimate a circular motion from linear measurements.
For part b, we need to convert the measured data to a range and bearing measurement, which is basically a polar representation of the robot's position with origin located at point M. Then, we need to update the control algorithm with the range and bearing measurement. The conversion formula are given in the equation below:

$$
\begin{aligned}
\rho &= \sqrt{(x - x_M)^2 + (y - y_M)^2} \\
\theta &= \arctan{(y - y_M)/(x - x_M)}
\end{aligned} \tag{11}
$$

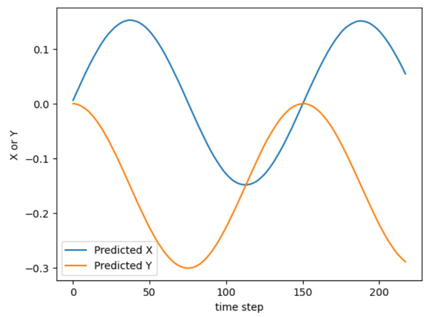
Two new functions were added to the code to convert the x, y measurement to polar coordinate system, the results are given below in Fig.3.



(a) Predicted, Measured, and True trajectories



(b) Measured Position



(c) Predicted Position

Figure 3: Results from Particle Filter with the Polar measurement model

We can observe that using a range-bearing measurement model has improved the accuracy of the Particle Filter's prediction and has enabled it to follow the circular path correctly.