ME8135 — Assignment 3 Solution

Student: Arash Basirat Tabrizi
Submitted to: Dr. Sajad Saeedi
Due: June 9, 2023

Introduction:

In the previous assignment, we used KF and EKF techniques to pass PDFs through nonlinearities by linearizing the nonlinear models about an operating point and then passing the PDFs through the linearized models analytically. In this section of the report, we aim to briefly outline and summarize the common techniques for passing PDFs through nonlinearities. The techniques are as follows:

- **Monte Carlo Method**: In this approach we draw a large number of samples from the input density, transform each one of these samples through the nonlinearity exactly, and then build the output density from the transformed samples. Although this method can be **terribly inefficient and computationally expensive**, it works with any PDF (not just Gaussian). In addition, we do not need to know the mathematical form of the nonlinear function. This method can be made **highly accurate**. We investigated this method in *A1 Question 2(e)*.

- **Linearization**: The most popular method of transforming a Gaussian PDF through a nonlinearity is linearization, which we investigated in *A2*. This approach is **highly inaccurate** and requires **complex calculations and approximations** (e.g. computing Jacobians in closed form). However, if the nonlinearity is locally invertible, this method is actually **reversible**. Meaning that we can recover the input PDF exactly by passing the output PDF through the inverse of the nonlinearity.

- **Sigmapoint Transformation**: This technique is the compromise between the Monte Carlo and linearization methods when the input density is roughly a Gaussian PDF. It is **more accurate than linearization**, but for a comparable **computational cost** to linearization.

The key takeaway is that the Monte Carlo approach is the most accurate method, but the computational cost is prohibitive in most practical situations.

Particle Filter Algorithm:

We aim to use our knowledge about the different methods of passing PDFs through nonlinearities to make some useful improvements to the KF and the EKF. Thus, we will discuss the Particle Filter (PF), which makes use of the Monte Carlo method.

PF is an alternative nonparametric implementation of the Bayes filter and approximate the posterior by a finite number of parameters. The key idea is to represent the posterior belief $bel(x_t)$ by a set of random state samples drawn from this posterior. The samples of a posterior distribution are called *particles*:

$$\chi_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{1}$$

Each particle $x_t^{[M]}$ (with $1 \leq m \leq M$) is a concrete instantiation of the state at time t. Here M denotes the number of particles in the particle set χ_t.

In the most basic variant of the PF algorithm, the input of the algorithm is the particle set χ_{t-1}, along with the most recent control input u_t, and the most recent measurement z_t.

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

Figure 1: The Particle Filter algorithm bases on sample importance resampling (from *Probabilistic Robotics* textbook.

In Figure 1, $\bar{\chi}$ is the temporary particle set that represents the predicted belief $\bar{bel}(x_t)$. Additionally in the algorithm, $w_t^{[M]}$ is the so-called *importance factor* calculated for each particle $x_t^{[M]}$ (line 5).

Lines 8 through 11 implement the real "trick" of the PF algorithm in what is known as *resampling*. Essentially, resampling transforms a particle set of M particles into another particle set of the same size. By incorporating the importance weights into the resampling process, the distribution of the particles change. Before the resampling step, the particles were distributed according to $\bar{bel}(x_t)$, after the resampling they are distributed (approximately) according to the posterior $bel(x_t)$ (equation 3).

$$\text{Particle Weight} = w_t^{[M]} = p(z_t \mid x_t^{[M]}) \tag{2}$$

$$\text{Posterior Belief} = bel(x_t) = \eta w_t^{[M]} \bar{bel}(x_t) \tag{3}$$

The importance factors computed by equation 2 represent the non-normalized probability mass of each particle. In equation 3, η is a normalization constant. In our PF implementation, we use Madow's resampling technique by creating bins based on the particle weights:

$$\beta_m = \frac{\sum_{n=1}^{m} w_m}{\sum_{l=1}^{M} w_l} \tag{4}$$

In equation 4, m is the unique particle index. We have $\beta_M = 1$ and select a random number, ρ, sampled from a uniform density on $[0, 1)$. The selection falls in bin β_m while the chosen sample corresponds to index m from the prior density. This procedure goes through M iterations and a new density is drawn after each iteration to make the particles converge from their initial scattered state. We will visualize this procedure in the simulation animations presenetd in later parts of this report.

1. Problem Statement:

We wish to use PyGame to simulate a simplified 2D robot and perform state estimation using a Kalman Filter. The robot's Motion Model is defined as follows:

$$\dot{x} = \frac{r}{2}(u_r + u_l) + w_x, \dot{y} = \frac{r}{2}(u_r + u_l) + w_y \tag{5}$$

In equation 5, 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. Additionally, $w_x = N(0, 0.1)$ and $w_y = N(0, 0.15)$.

We wish to simulate the system such that the robot is driven 1 m to the right. We are to assume the speed of each wheel is fixed at $0.1\frac{m}{s}$. We use the following initial values:

$$x_0 = 0, y_0 = 0, P_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \text{ (initial covariance matrix)} \tag{6}$$

Also, we are to assume that the motion model is computed 8 times a second. We assume every second a measurement is given:

$$z = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} r_x \\ r_y \end{bmatrix} \tag{7}$$

Where $r_x = N(0, 0.05)$ and $r_y = N(0, 0.075)$.

Deliverables: We wish to plot and animate the trajectory along with covariance ellipse for all motion models and measurement updates. In the animation, we will show both ground-truth and PF estimate trajectories.

Solution Formulation:

The state transition model in matrix form remains the same as in the previous assignment and is restated here (see *GitHub Repo*/A2/SolA2_Arash.pdf for derivation):

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{r}{2}T & \frac{r}{2}T \\ \frac{r}{2}T & \frac{r}{2}T \end{bmatrix} \begin{bmatrix} u_{r,k} \\ u_{l,k} \end{bmatrix} + \begin{bmatrix} w_{x,k} \\ w_{y,k} \end{bmatrix} T \tag{8}$$

Simulation Results and Observations:

The simulation animation can be found at: *GitHub Repo*/A3/animation-1-new.gif

The animation has been slowed down for demonstration purposes.

From the simulation results we deduce that the PF algorithm generated a much more accurate estimation trajectory when compared to the KF for the same linear motion and measurement models (Figure 2). This behaviour was expected. Even though we set $M = 100$, the implemented technique is still more computationally expensive than the KF approach. Using more particles always helps, but comes with a computational cost. Analyzing the computational cost and comparing it with the cost of the KF implementation is outside of the scope of this assignment.

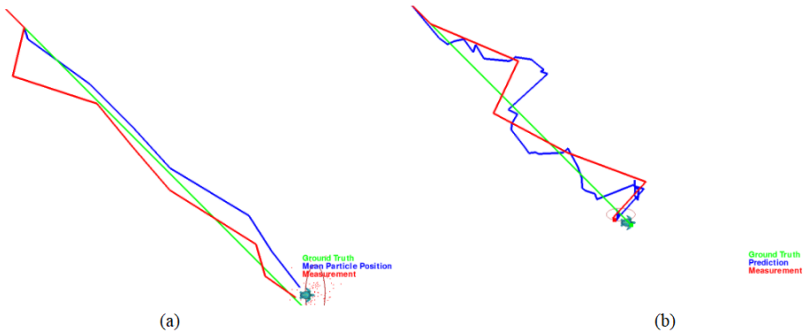

Figure 2: Simulation results for linear motion and measurement models: (a) PF implementation (b) KF implementation.

We observe that as the particles converge and become dense, the covariance ellipse shrinks.

The most significant plot generated by our Python simulation script (A3/Q1.py) is the following:

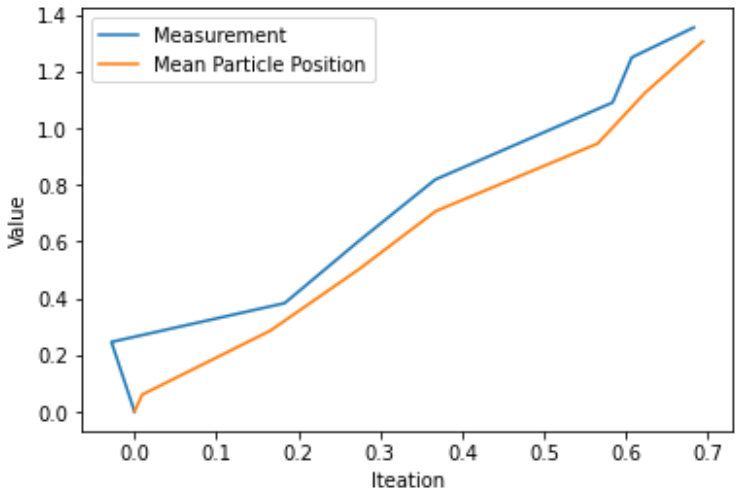

Figure 3: PF prediction compared with measurement model readings for Q1.

Figure 3 is the realization that the PF estimation can accurately track the measurement readings.

2. Problem Statement:

We wish to repeat the previous question, this time with a classic **nonlinear** motion model and range measurements made from a landmark located at M = [10,10]. L is the distance between the wheel, known as wheelbase, and is defined to be 0.3m. We must program the robot such that it loops around point M.

In *part(a)*, the motion model presented in equation 11 is used in combination with the linear measurement model previously presented in equation 7. In *part(b)*, the same nonlinear motion model is used in combination with the range/bearing measurements of point M. Where we 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.

$$\dot{x} = \frac{r}{2}\left(u_r + u_l\right) cos(\theta) + w_x, \dot{y} = \frac{r}{2}\left(u_r + u_l\right) sin(\theta) + w_y, \dot{\theta} = \frac{r}{L}\left(u_r - u_l\right) \tag{9}$$

$$u_w = \frac{1}{2}\left(u_r + u_l\right), u_\psi = \left(u_r - u_l\right) \tag{10}$$

Combining equations 9 and 10 gives:

$$\dot{x} = r u_w cos(\theta) + w_w, \dot{y} = r u_w sin(\theta) + w_w, \dot{\theta} = \frac{r}{L} u_\psi + w_\psi \tag{11}$$

Where $n_\psi = N(0, 0.01)$ and $n_d = N(0, 0.1)$.

Part (a) Solution Formulation:

The state transition model in matrix form remains the same as in the previous assignment:

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} r \cdot T \cdot \cos(\theta) & 0 \\ r \cdot T \cdot \sin(\theta) & 0 \\ 0 & T \cdot \frac{r}{L} \end{bmatrix} \begin{bmatrix} u_{r,k} \\ u_{l,k} \end{bmatrix} + w_n T \tag{12}$$

The measurement model remains the same as in the previous question (equation 7).

Part (b) Solution Formulation:

The state transition model remains the same as in the previous question (equation 12).

The range/bearing measurement model in matrix form remains the same as in the previous assignment:

$$\mathrm{z} = \begin{bmatrix} \sqrt{(p_x - x)^2 + (p_y - y)^2} \\ \arctan\left(\frac{p_y - y}{p_x - x}\right) - \theta \end{bmatrix} + \begin{bmatrix} n_d \\ n_\phi \end{bmatrix} \tag{13}$$

Simulation Results and Observations:

The simulation animations can be found at: *GitHub Repo*/A3/animation-2a.gif *and* animation-2b.gif

The animations have been slowed down for demonstration purposes.

For the simulation of part (a), we observe that the PF estimated trajectory does not follow the ground-truth trajectory as accurately as we would like it to. The estimated trajectory is observed to lag behind the ground-truth trajectory. This behaviour is influenced by the linear measurement model's parameters. We correct the measurement model in the simulation for part (b) to reduce the lag and observe significantly more accurate estimation trajectories.

Although we did not draw the measurement trajectories (in red) for the the simulation of part (a) for demonstration purposes, we produced the plot shown in Figure 4 to demonstrate that the PF estimated trajectory generally follows the trend of the measurement trajectories regardless.

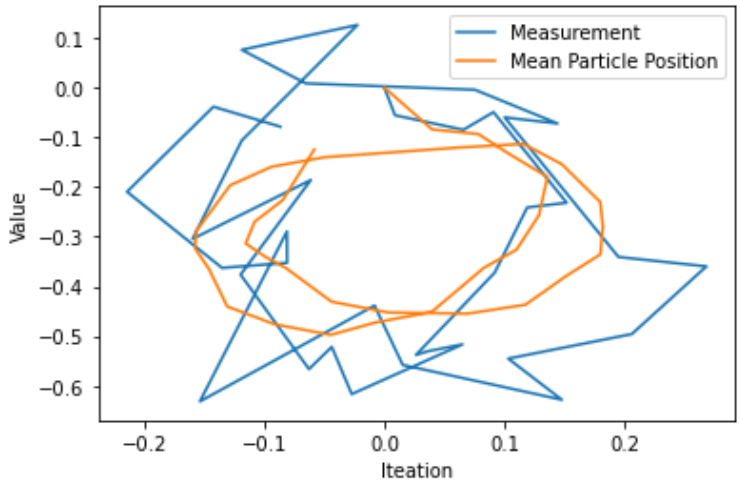

Figure 4: PF prediction compared with measurement model readings for Q2(a).

In case we wish to visualize the measurement trajectories during the live simulation, lines 327-329 and 348-350 in the Python simulation script (A3/Q2-a.py) should be uncommented.

For the simulation of part (b), Figure 5 captures the notion that the PF estimated trajectory encapsulates the nonlinear measurement model's trajectory. This is verified by the blue and red trajectories animated during the simulation. Due to the nonlinear nature of the measurement model in part (b), the measurement trajectories are better visualized in the simulation animation than in the following plot.

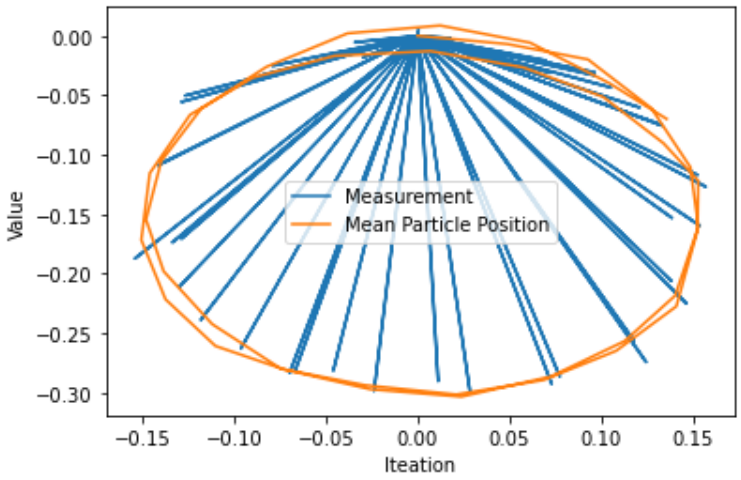

Figure 5: PF prediction compared with measurement model readings for Q2(b).

Summary:

It is apparent that the behaviour of the estimation in a PF depends on the specific characteristics of the motion and measurement models, as well as the number of particles used and their initial distribution. For Q1, the estimation gradually converges to the true position as the robot moves to the right. The particles spread out due to the noise in the motion and measurement models, but the filter updates their weights based on the likelihood of the measurements, leading to increased accuracy over time.

For Q2(a) the estimation encountered some challenges. The nonlinear motion model introduced complexities, and the particles struggled to represent the true state accurately. The estimation exhibited drifting or had difficulty capturing sudden changes in motion direction which introduced delay/lag between trajectories.

For Q2(b) the estimation overcame the drifting challenges however it exhibited increased uncertainty and particle divergence.

While the PF is well-suited for nonlinear and non-Gaussian problems, KF and EKF offer a simpler and more computationally efficient solution for linear and Gaussian problems.