Probabilistic Robotics

Title: EKF and PF for Localization of a mobile robot

University Of Burgundy
Master's in Computer Vision and Robotics

Submitted by: Muhammad Usama Javaid
Ahmed Khalil
Supervised by: Prof. Taihú Pire
Date: May 16, 2025

EKF and PF for Localization of a Mobile Robot

Abstract

This report talks about how we implemented and compared two popular methods for robot localization: the Extended Kalman Filter (EKF) and the Particle Filter (PF). These filters are useful when a robot needs to figure out where it is, especially when its movements and sensor readings are noisy. We built both filters using Python and tested them in a simple 2D environment with different levels of noise. The results helped us understand when each method works well and what trade-offs they come with.

1 Introduction

When a mobile robot moves around, it usually doesn't know exactly where it is. Its movements are not perfect, and its sensors are also noisy. Because of this, it needs a way to estimate its position based on uncertain information. This process is called localization.

To handle this, we looked at two filtering methods that are commonly used in robotics: EKF and PF. The EKF works well if the noise is low and the system behaves in a simple way. On the other hand, PF is more flexible and can deal with more difficult situations, but it needs more computational effort. In this project, we wrote both filters from scratch and tested them to see how they behave.

2 Problem Statement

In this project, the goal was to track a robot's position and orientation (angle) while it moved in a 2D soccer field. The robot could turn and move forward based on control commands. It also received noisy observations of landmarks (fixed points) around it. These observations were in the form of bearing angles.

Because both the movement and the observations were noisy, we couldn't just use the raw data. Instead, we used EKF and PF to estimate the robot's real position over time. We then tested how these filters perform when the noise levels change, and for PF, when the number of particles changes too.

3 Background and Filter Concepts

3.1 Bayes Filter Basics

Both EKF and PF are based on the Bayes filter. This method updates the belief about the robot's position over time, using both controls and observations. In simple words, the filter first

guesses where the robot should be after a move, and then corrects this guess using the sensor reading.

3.2 Extended Kalman Filter (EKF)

EKF works by assuming that the robot's position is described by a Gaussian distribution (a nice bell-shaped curve). Every time the robot moves or sees a landmark, the EKF predicts the new position and updates the uncertainty using Jacobian matrices. These Jacobians describe how the system changes with small movements.

We implemented the Jacobians as:

$$\mathbf{G}(x,u) = \begin{bmatrix} 1 & 0 & -\delta_{trans} \cdot \sin(\theta + \delta_{rot1}) \\ 0 & 1 & \delta_{trans} \cdot \cos(\theta + \delta_{rot1}) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{V}(x,u) = \begin{bmatrix} -\delta_{trans} \cdot \sin(\theta + \delta_{rot1}) & \cos(\theta + \delta_{rot1}) & 0 \\ \delta_{trans} \cdot \cos(\theta + \delta_{rot1}) & \sin(\theta + \delta_{rot1}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The Jacobian of the observation model with respect to the state is:

$$\mathbf{H}(x) = \begin{bmatrix} \frac{dy}{q} & -\frac{dx}{q} & -1 \end{bmatrix}$$

where $dx = m_x - x$, $dy = m_y - y$, and $q = dx^2 + dy^2$.

3.3 Particle Filter (PF)

The PF takes a different approach. Instead of using just one estimate and a Gaussian uncertainty, it keeps many possible positions (called particles). Each particle is a guess of where the robot might be. After moving the particles based on the control input, the filter checks how well each one matches the observation, gives them a weight, and then keeps the best ones using a method called low-variance resampling.

The main update loop looks like this:

- For each particle:
  - Sample a noisy control input and move the particle using the motion model.
  - Compare the particle's expected observation to the actual one and compute a weight.
- Normalize the weights.
- Use low-variance resampling to select new particles.

4 How We Implemented the Filters

We wrote all the code in Python using NumPy and Matplotlib. The simulation of the robot and its world is in a file called `soccer_field.py`, which also handles the landmarks and the noise models. Our EKF and PF code is in `ekf.py` and `pf.py`. The main script to run everything is `localization.py`.

For the EKF, we implemented both the prediction and correction steps, and manually coded the Jacobians needed for the motion and observation models. For PF, we added the steps to move the particles, calculate weights, and perform resampling. We also included a function to compute the mean and variance from the particles.

The robot's control input had three values: how much it turned at the start, how far it moved forward, and how much it turned at the end. The observation was the angle to a visible landmark. The noise was modeled using parameters α for motion and β for observation, which we adjusted to test different noise levels.

4.1 How to Run the Code

To run the simulations and see the robot's behavior, we used the command-line interface provided in `localization.py`. It allows us to choose which filter to run, adjust noise parameters, set the number of particles, and turn on visual plots.

Some common commands we used:

```
python localization.py --plot --filter-type ekf
python localization.py --plot --filter-type pf
python localization.py --plot --filter-type none
```

5 Results and Discussion

Part (a) – Actual vs. Estimated Path with Default Parameters

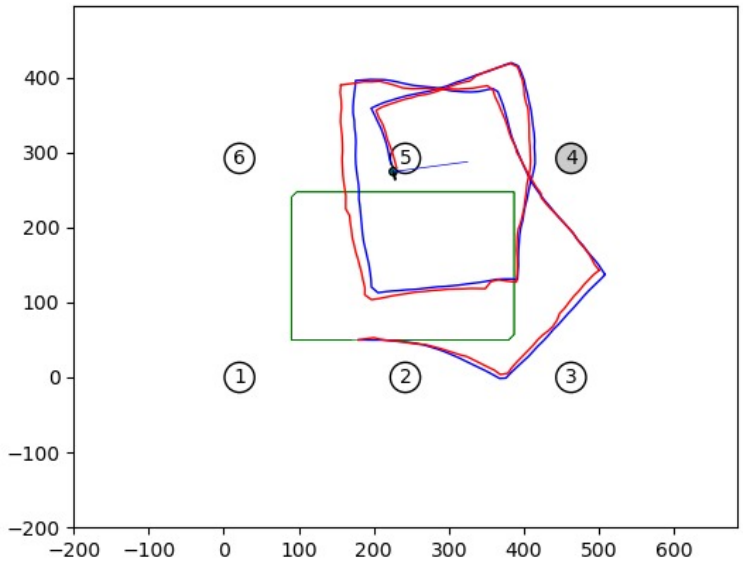

Figure 1: Plot of the actual robot path and the path estimated by the filter using default parameters.

Results from command:
`python localization.py ekf --seed 0`

- Mean position error: 8.9983675360847
- Mean Mahalanobis error: 4.416418248584298
- ANEES: 1.472139416194766

Part (b) – Mean Position Error vs. Noise Factor (EKF)

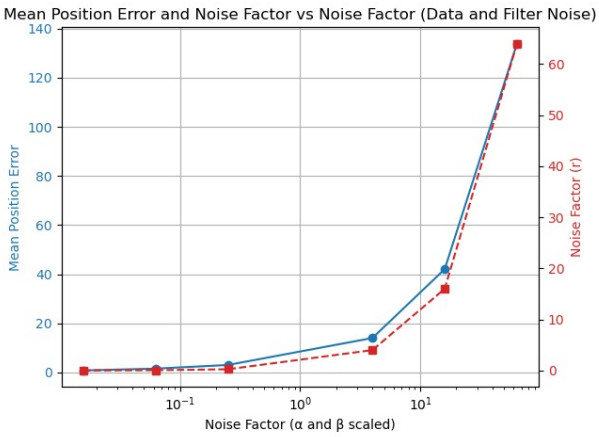

Figure 2: Mean position error vs. noise scaling factor r for EKF. Each point is averaged over 10 trials.

Main Observations:

- With small noise (low r), the EKF gives accurate results and low error.
- As r increases, the error grows quickly and EKF becomes less reliable.
- The EKF is very sensitive to noise and works best when the noise is small and well-tuned.
- This test shows that modeling the noise correctly is very important for the EKF to work well.

Part (c) – Mean Position Error and ANEES vs. Filter Noise (EKF)

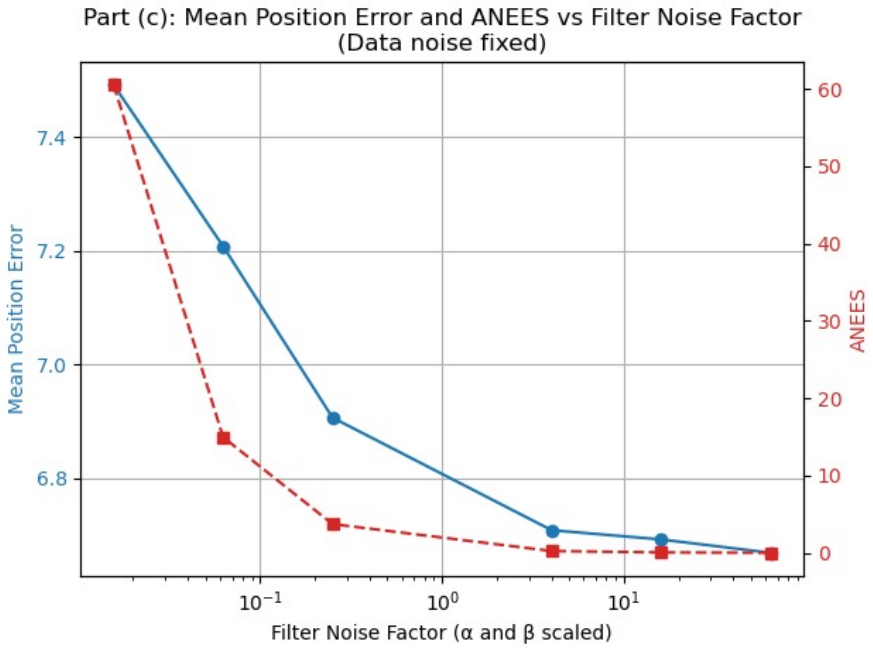

Figure 3: Mean position error and ANEES vs. filter noise factor r for EKF (data noise fixed at 1).

What We Saw:

- Mean position error drops quickly as we increase r from very small values, but then becomes stable at higher values.
- ANEES is high when the filter noise is too low (the filter is overconfident). As we increase r, ANEES goes down and stabilizes, but it doesn't reach the ideal value of 1.

Conclusion:
Filter noise needs to be tuned properly. Too little noise makes the filter too sure of itself and leads to big errors. Too much noise makes it more stable but less accurate. So, good tuning is key for getting both low error and correct uncertainty.

Part (b) – Mean Position Error vs. Noise Factor (PF)

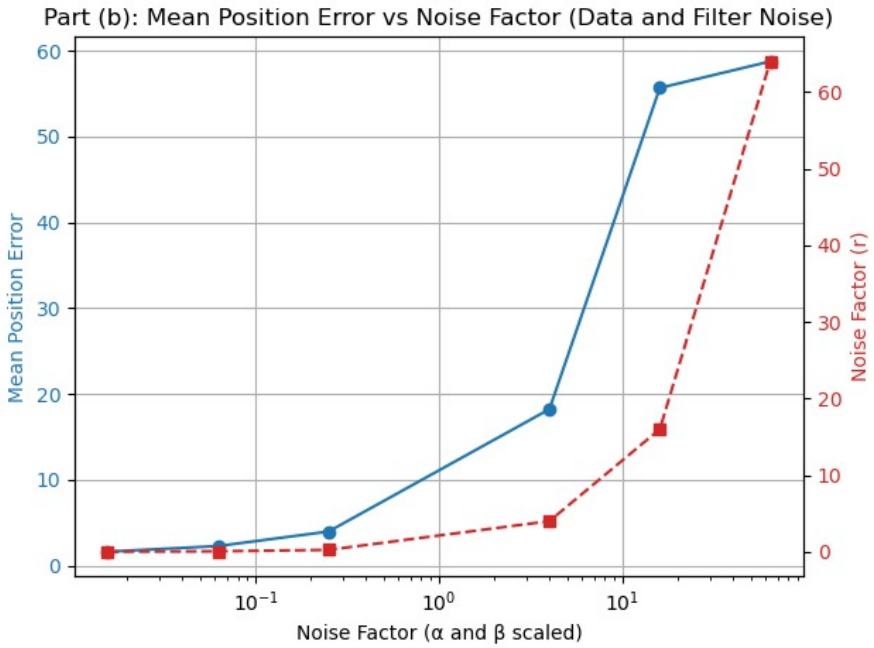

Figure 4: Mean position error vs. noise factor r using Particle Filter. Lower r values give better accuracy.

What We Observed:

- At very low noise (e.g., $r = 1/64$), the PF gives very accurate results with low error.
- As r increases, the error also increases a lot, showing that PF struggles more in high-noise settings.

Conclusion:
The Particle Filter works well when the noise is low and well-tuned. But if noise is too high, the filter's accuracy drops quickly. This shows how important it is to match the filter's noise settings to the real system for reliable performance.

Part (c) – Mean Position Error and ANEES vs. Filter Noise (PF)

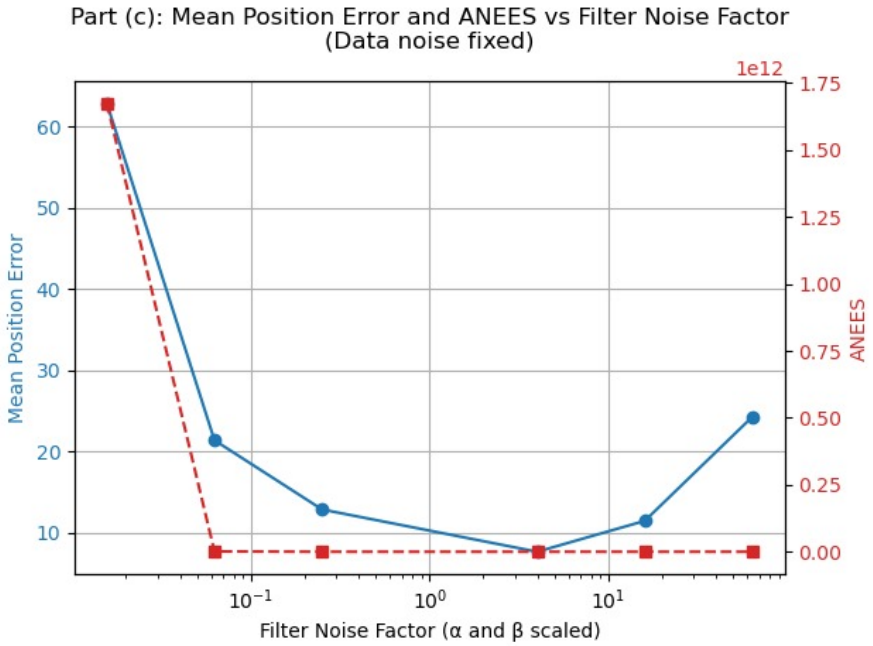

Figure 5: Mean position error and ANEES vs. filter noise factor r using Particle Filter (data noise fixed at 1).

What We Observed:

- When filter noise is too low (e.g., $r = 1/64$), the filter becomes overconfident. This causes high MPE and very high ANEES.
- As we increase r, both MPE and ANEES improve. The filter estimates become more accurate and realistic.
- After a certain point, increasing noise further doesn't help much — results stay about the same, and ANEES never fully reaches the ideal value of 1.

Conclusion:
The PF needs a well-tuned noise setting to balance accuracy and uncertainty. Too little noise makes the filter too sure and wrong; too much noise increases uncertainty without big gains. Proper tuning is key for stable and reliable results.

Part (d) – MPE and ANEES vs. Filter Noise for Different Particle Counts

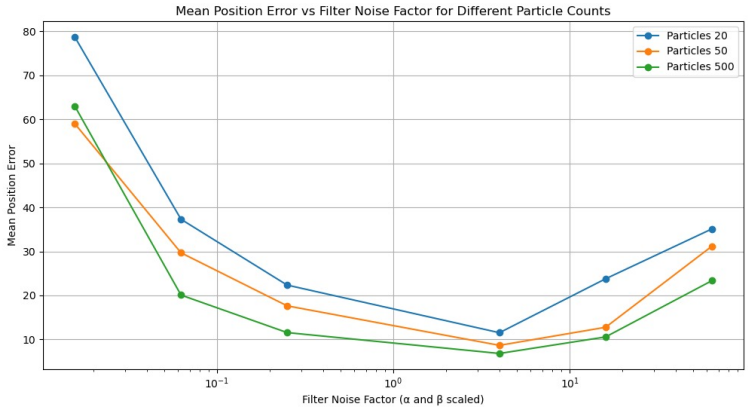

Figure 6: Mean position error vs. filter noise factor r for different particle counts (20, 50, 500) using Particle Filter.

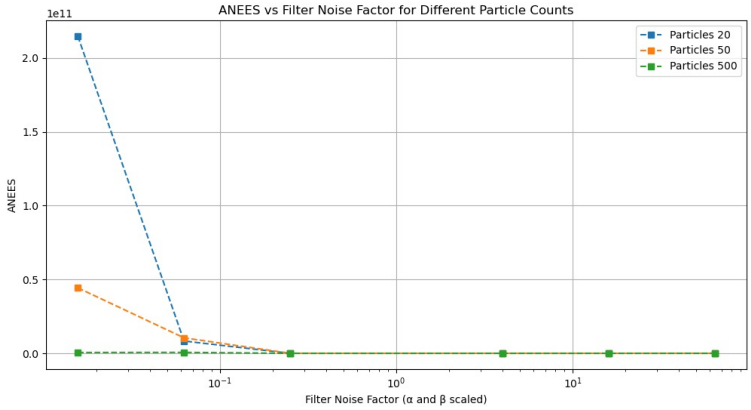

Figure 7: ANEES vs. filter noise factor r for different particle counts using Particle Filter.

What We Observed:

- Using more particles gives better results. With 500 particles, both MPE and ANEES were the lowest and most stable.
- When filter noise is too low, the filter becomes overconfident, leading to high MPE and high ANEES.
- As filter noise increases, the filter becomes more cautious, and the results get better.

Conclusion:
More particles help the PF deal with noise better, but it also needs the right noise settings. Both particle count and noise tuning are important to get accurate and reliable localization.