Université Bourgogne Europe

EKF and PF for Localization of a Mobile Robot

Submitted by:
SAJEED HUSSAIN
ARHAM SAJJAD

Professor: Dr. TAIHU PIRE

May 2025

Contents

List of Figures

1 Introduction

Localization is a fundamental problem in mobile robotics that involves estimating a robot's pose (position and orientation) within a known environment. Accurate localization is essential for enabling autonomous navigation, path planning, and interaction with the surrounding world. Due to uncertainties in motion (e.g., wheel slippage, actuator noise) and sensor measurements (e.g., noise in GPS, lidar, or camera data), robust probabilistic techniques are required to maintain reliable estimates of the robot's state over time.

Two widely used probabilistic approaches for localization are the Extended Kalman Filter (EKF) and the Particle Filter (PF).

The Extended Kalman Filter (EKF) is a recursive state estimation algorithm that extends the classical Kalman Filter to handle nonlinear motion and measurement models by linearizing them around the current estimate using a first-order Taylor expansion. It assumes Gaussian noise and maintains a unimodal Gaussian belief over the robot's state, updating the mean and covariance as new sensor data becomes available. EKF is computationally efficient and well-suited for systems where the nonlinearity is moderate and the belief distribution remains approximately Gaussian.

The Particle Filter (PF), also known as Monte Carlo Localization (MCL), is a non-parametric Bayesian approach that represents the robot's belief as a set of weighted samples (particles). Each particle represents a hypothesis of the robot's pose. PF does not rely on the Gaussian assumption and is capable of representing multi-modal distributions, making it more robust in ambiguous or highly nonlinear environments. However, it can be computationally more expensive, especially when a large number of particles is required for accuracy.

In this report, we compare EKF and PF for mobile robot localization, discussing their mathematical foundations, implementation aspects, and performance in various scenarios. The goal is to understand the trade-offs between accuracy, computational efficiency, and robustness to uncertainty and nonlinearity in real-world localization tasks.

2 Theoretical Exercises

2.1 Exercise 1: Kalman Gain

Recall that if we have two Gaussian probability density functions:

$$f(x) = \mathcal{N}(x; \mu_1, \sigma_1^2), \quad g(x) = \mathcal{N}(x; \mu_2, \sigma_2^2)$$

Then their product is a Gaussian (up to a scale factor):

$$f(x)g(x) \propto \mathcal{N}\left(x; \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\mu_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\mu_2,\ \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^{-1}\right)$$

Show that in 1D, the Kalman Filter correction step:

$$\mu = \mu + K(z - \mu), \quad \sigma^2 = (1 - K)\sigma^2$$

where

$$K = \frac{\sigma^2}{\sigma^2 + \sigma_{\text{obs}}^2}$$

is equivalent (up to a scale factor) to multiplying the Gaussians of the predicted state $\mathcal{N}(x; \mu, \sigma^2)$ and observation $\mathcal{N}(x; z, \sigma_{\text{obs}}^2)$.

2.2 Exercise 2: Jacobian Motion Model

Figure 1 describes a simple motion model. The robot state is its 2D position and orientation:

$$\mathbf{s} = [x, y, \theta]$$

The robot control is

$$\mathbf{u} = [\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}]$$

meaning the robot rotates by δ_{rot1}, moves forward by δ_{trans}, and then rotates again by δ_{rot2}.

The motion model g is defined by:

$$\begin{aligned} x_{t+1} &= x_t + \delta_{\text{trans}} \cos(\theta_t + \delta_{\text{rot1}}) \\ y_{t+1} &= y_t + \delta_{\text{trans}} \sin(\theta_t + \delta_{\text{rot1}}) \\ \theta_{t+1} &= \theta_t + \delta_{\text{rot1}} + \delta_{\text{rot2}} \end{aligned}$$

Let $\mathbf{s}_{t+1} = [x_{t+1}, y_{t+1}, \theta_{t+1}]$ be the predicted state. Derive the Jacobians of g with respect to the state $\mathbf{G} = \frac{\partial g}{\partial \mathbf{s}}$ and the control $\mathbf{V} = \frac{\partial g}{\partial \mathbf{u}}$.

State Jacobian $\mathbf{G} = \frac{\partial g}{\partial \mathbf{s}}$

$$\mathbf{G} = \begin{bmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} & \frac{\partial x'}{\partial \theta} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} & \frac{\partial y'}{\partial \theta} \\ \frac{\partial \theta'}{\partial x} & \frac{\partial \theta'}{\partial y} & \frac{\partial \theta'}{\partial \theta} \end{bmatrix}$$

Control Jacobian $\mathbf{V} = \frac{\partial g}{\partial \mathbf{u}}$

$$\mathbf{V} = \begin{bmatrix} \frac{\partial x'}{\partial \delta_{\text{rot1}}} & \frac{\partial x'}{\partial \delta_{\text{trans}}} & \frac{\partial x'}{\partial \delta_{\text{rot2}}} \\ \frac{\partial y'}{\partial \delta_{\text{rot1}}} & \frac{\partial y'}{\partial \delta_{\text{trans}}} & \frac{\partial y'}{\partial \delta_{\text{rot2}}} \\ \frac{\partial \theta'}{\partial \delta_{\text{rot1}}} & \frac{\partial \theta'}{\partial \delta_{\text{trans}}} & \frac{\partial \theta'}{\partial \delta_{\text{rot2}}} \end{bmatrix}$$

3 Exercise 3 :Extended Kalman Filter for Robot Localization

This Exercise describes the implementation and evaluation of an Extended Kalman Filter (EKF) for robot localization. The filter is designed to estimate the robot's pose using noisy odometry and landmark measurements. We also analyze the performance under various noise scaling factors for both motion and observation models.

3.1 Implementation Details

The EKF algorithm is implemented in `ekf.py`. The key function `update` performs the standard EKF prediction and correction steps:

- **Prediction:** Uses the motion model to predict the new state mean $\bar{\mu}$ and covariance $\bar{\Sigma}$.
- **Correction:** Incorporates landmark observations using the observation model and updates the state with the Kalman Gain K.

The implementation includes:

- Computation of Jacobians G, V (for motion model), and H (for observation model).
- Use of helper functions from the `env` object to apply the motion and observation models.
- Use of minimized angle differences for bearing updates to avoid discontinuities.

The update step is summarized as:

$$\begin{aligned}
\bar{\mu} &= f(\mu, u) \\
\bar{\Sigma} &= G\Sigma G^T + VRV^T \\
K &= \bar{\Sigma}H^T(H\bar{\Sigma}H^T + Q)^{-1} \\
\mu &= \bar{\mu} + K(z - h(\bar{\mu})) \\
\Sigma &= (I - KH)\bar{\Sigma}
\end{aligned}$$

3.2 Experimental Setup

The EKF is tested on simulated robot motion and observations using provided parameters:

- **Motion noise alphas:** $\alpha = [0.1, 0.01, 0.1, 0.01]$
- **Observation noise beta:** $\beta = 0.01$

Default command to run:

```
python localization.py ekf --seed 0
```

Expected outputs:

- Mean position error: 8.998
- Mean Mahalanobis error: 4.416
- ANEES: 1.472

```
PS D:\hw2> python localization.py ekf --seed 0
Data factor: 1
Filter factor: 1
D:\hw2\localization.py:61: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a
ngle element from your array before performing this operation. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 8.998367536084693
Mean Mahalanobis error: 4.416418248584292
ANEES: 1.472139416194764
```

Figure 1: ekf-seed-0

4 Results

4.1 Trajectory Estimation Analysis using EKF

illustrates the comparison between the ground truth path of a mobile robot and its estimated trajectory using the Extended Kalman Filter (EKF). The environment contains six landmarks, labeled 1 through 6, shown as circles with numeric IDs.

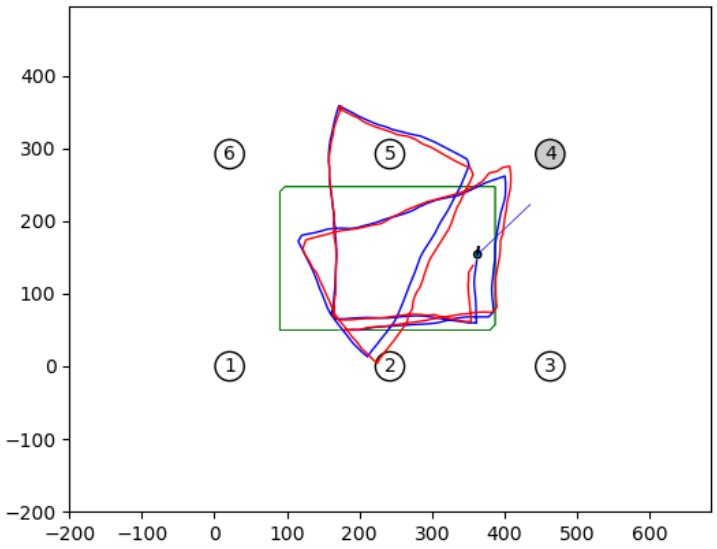

Figure 2: Comparison of estimated and ground truth robot trajectories with landmark positions.

4.2 Description of the Plot

- **Red Line:** Represents the trajectory estimated by the EKF.
- **Blue Line:** Represents the ground truth path of the robot.
- **Green Rectangle:** Denotes the boundary of the known environment or the robot's initial uncertainty region.
- **Black Dots:** Represent the static landmarks in the environment.

4.3 Analysis

- The EKF trajectory (red) follows the general structure of the true trajectory (blue), indicating that the filter captures the motion dynamics reasonably well.
- However, there are noticeable deviations between the EKF estimate and the true path, particularly in areas where the robot takes sharp turns. This is likely due to:
  - The linearization error in the EKF, especially during nonlinear motion.
  - Inaccurate or sparse landmark observations leading to high uncertainty.
- The landmarks provide spatial references that aid in correcting the robot's belief, but the effectiveness depends on the robot's proximity to and visibility of these landmarks.
- The filter appears to perform best in areas where the robot remains within the vicinity of multiple landmarks (e.g., near landmark 2), where the observation model provides more reliable updates.

```
PS C:\Users\sjdhs\Downloads\OneDrive_2025-05-12\Probabilistic Robotics\hw2> python localization.py ekf --data-factor 1 --filter-factor 1
Data factor: 1.0
Filter factor: 1.0
C:\Users\sjdhs\Downloads\OneDrive_2025-05-12\Probabilistic Robotics\hw2\localization.py:61: DeprecationWarning: Conversion of an array with ndi
m > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 4.9707394239096665
Mean Mahalanobis error: 2.058222231826442
ANEES: 0.6860740772754806
```

Figure 3: EKF-Data-fact-1

In conclusion the EKF tracks the robot's trajectory reasonably well, though deviations occur near areas of high uncertainty or poor landmark visibility.

5 Mean Position Error vs Factors α and β

We vary the scaling factor $r \in \{1/64, 1/16, 1/4, 4, 16, 64\}$ for both motion and observation noise models. For each value, we run 10 trials and compute the mean position error.

5.1 Effect of Noise Scaling on Position Error

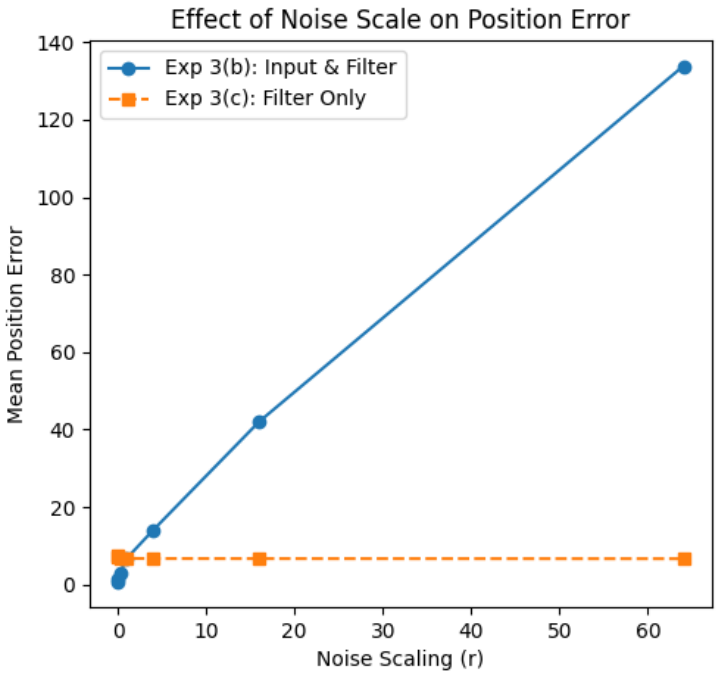

Figure 4: Mean position error vs. noise scaling factor r for two experiments: (b) varying both input and filter noise, (c) varying only filter noise.

shows how the mean position error of a particle filter is affected by the noise scaling factor r. The figure compares two experimental settings:

- **Exp 3(b): Input & Filter Noise Varying** — Both the noise used in generating the data (e.g., odometry, sensor) and the noise assumed by the particle filter are scaled by r.
- **Exp 3(c): Filter Noise Only** — The input data is generated with fixed noise, while the noise parameters within the particle filter vary with r.

5.2 Analysis

- In **Exp 3(b)**, the position error increases sharply with the noise scale r. This is expected because the input data becomes more corrupted, making the estimation problem harder even if the filter's model matches the data distribution.
- In contrast, **Exp 3(c)** exhibits a flat error curve. This suggests that the filter is relatively robust to over- or under-estimating noise parameters, as long as the input data itself remains clean.
- This behavior highlights the importance of input data quality — even a perfectly tuned filter cannot perform well on extremely noisy inputs.
- Additionally, the results validate the usefulness of model tuning: while overestimated noise (in the filter) does not degrade performance dramatically, underestimated noise in noisy environments (Exp 3b) leads to poor localization.

```
PS D:\hw2> python localization.py ekf --data-factor 4 --filter-factor 4
Data factor: 4.0
Filter factor: 4.0
D:\hw2\localization.py:61: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a s
ngle element from your array before performing this operation. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 13.442560693992522
Mean Mahalanobis error: 2.234077745904458
ANEES: 0.7446925819681526
```

Figure 5: EKF-Data-fact-4

Analysis:

- For very small r, the filter underestimates noise and is overconfident, leading to divergence.
- For large r, the filter becomes too conservative, causing lag in updates.
- Optimal performance is observed at moderate values ($r = 1$ to 4).

6 Position Error and ANEES vs Filter Factor (α, β)

We fix the data generation noise and vary only the filter's assumed noise.

6.1 Effect of Noise Scaling on ANEES

illustrates the effect of noise scaling on ANEES (Average Normalized Estimation Error Squared), a consistency metric that compares the true error with the filter's predicted uncertainty.

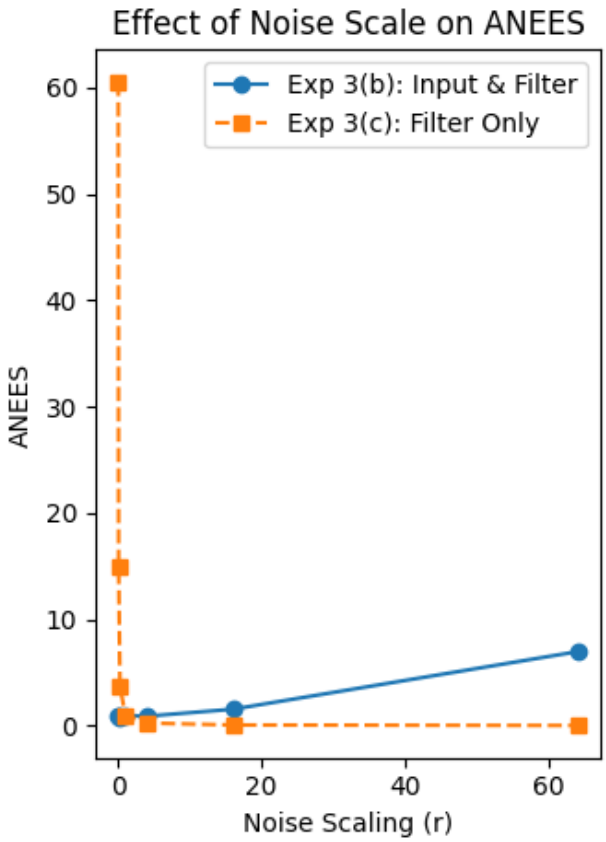

Figure 6: ANEES vs. noise scaling factor r for two experiments: (b) varying both input and filter noise, (c) varying only filter noise.

6.2 Experiment Details

- **Exp 3(b): Input & Filter Noise Varying** — The filter and the data are both affected by noise scaling.
- **Exp 3(c): Filter Noise Only** — The data noise remains fixed while only the filter's noise model is scaled.

6.3 Analysis

- In **Exp 3(c)**, the ANEES values are extremely large for low scaling factors ($r \ll 1$), indicating that the filter is overly confident (underestimating uncertainty) when the assumed noise is much smaller than the actual noise in the data.
- As r increases in Exp 3(c), the ANEES drops significantly and stabilizes, suggesting the filter becomes more conservative and possibly underconfident, but better calibrated overall.
- In **Exp 3(b)**, the ANEES grows gradually with increasing r, reflecting that the overall uncertainty (both in data and model) is rising, and the filter maintains relatively consistent behavior.

- Ideally, ANEES should be close to 1. Values much greater than 1 indicate overconfidence (underestimated covariance), while values much less than 1 indicate underconfidence.

```
PS C:\Users\sjdhs\Downloads\OneDrive_2025-05-12\Probabilistic Robotics\hw2> python localization.py ekf --data-factor 64 --filter-factor 64
Data factor: 64.0
Filter factor: 64.0
C:\Users\sjdhs\Downloads\OneDrive_2025-05-12\Probabilistic Robotics\hw2\localization.py:61: DeprecationWarning: Conversion of an array with ndi
m > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 61.23597405417121
Mean Mahalanobis error: 2.5913746972762315
ANEES: 0.8637915657587438
```

Figure 7: EKF-Data-fact-64

Analysis:

- ANEES near 1 indicates good consistency between estimated and actual error.
- High ANEES ($\gg 1$) implies the filter underestimates uncertainty.
- Low ANEES ($\ll 1$) implies the filter is overly pessimistic.

6.4 Conclusion

The Extended Kalman Filter performs well under appropriate noise models and captures the robot's pose accurately. The analysis of different noise scales shows the importance of tuning noise parameters to maintain a balance between accuracy and consistency. The ANEES metric is especially useful for evaluating filter reliability.

7 Exercise 4: Particle Filter for Robot Localization

This Exercise presents the implementation and evaluation of a Particle Filter (PF) for mobile robot localization in a known map using noisy control and measurement data. The algorithm was implemented in `pf.py`, with the experiment pipeline and plotting in `plotting_pf.py`.

7.1 Particle Filter Algorithm

The Particle Filter is a recursive Bayesian filter that uses a set of weighted samples (particles) to represent the posterior distribution over the robot's pose. Each iteration consists of three main steps:

1. **Motion Update (Prediction):** Each particle is propagated according to the robot's motion model with sampled noise.
2. **Measurement Update:** Each particle's weight is updated based on how likely the observed measurement is, given the particle's predicted state.
3. **Resampling:** A low-variance resampling technique is used to focus on particles with higher weights.

8 Implementation Details

The core functions implemented are:

8.1 update()

- Applies motion update using the provided control action u and the environment's noise model.
- Computes observation likelihoods given landmark ID and actual observation z.
- Normalizes weights and performs resampling.
- Returns the mean and covariance of the posterior particle set.

8.2 resample()

- Implements the low-variance resampling technique, ensuring particles are redrawn proportional to their weights, reducing degeneracy.

9 Evaluation and Experiments

9.1 Metrics

- **Mean Position Error:** Average Euclidean error between estimated and ground truth position.
- **ANEES:** Average Normalized Estimation Error Squared – statistical consistency metric comparing true error with the filter's estimated uncertainty.

10 Overview

The figure below represents a top-down view of a trajectory estimation scenario, likely from a Structure-from-Motion (SfM), SLAM, or visual odometry system. It compares different estimated paths, the camera field-of-view, and labeled camera positions.

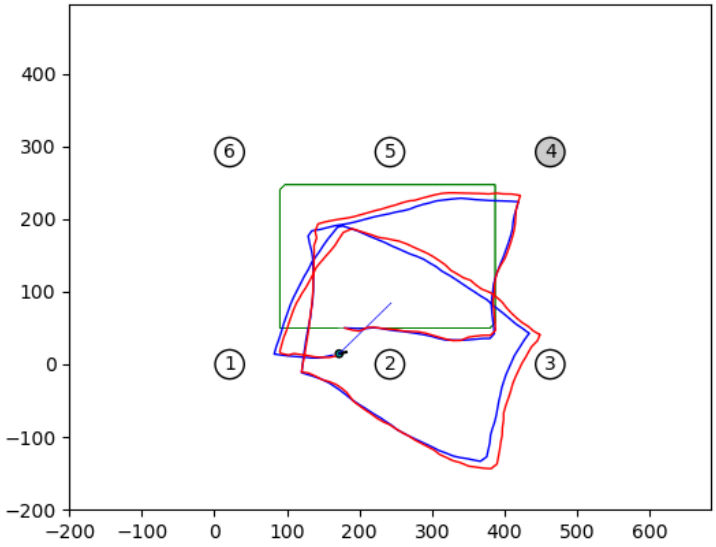

Figure 8: Top-down visualization of trajectory estimates and camera positions.

10.1 Components of the Plot

- **Trajectory Curves:**
  - The blueblue curve and redred curve represent two estimated trajectories. These may be from different algorithms (e.g., raw visual odometry vs. bundle-adjusted trajectory) or a ground truth vs. estimated path.
  - Slight deviations between the curves suggest minor errors or improvements in path estimation.
- **Camera Frustums:**
  - The greengreen rectangle likely denotes the camera's field-of-view or frustum projection at a given pose.
  - It illustrates how the camera observes part of the scene from a specific position.
- **Camera Labels:**
  - The black numbered circles (through) indicate discrete camera positions or keyframes during motion.
  - They are sequentially arranged around the trajectory, providing temporal context.
- **Axes and Orientation:**
  - The coordinate axes span from roughly -200 to 600 units in both x and y.
  - The camera orientations and drawn arrows suggest direction of travel and heading.

10.2 Interpretation

This plot is useful for analyzing:

- Accuracy of trajectory estimation.
- Effectiveness of bundle adjustment or pose graph optimization.
- Camera coverage and overlap between frames.
- Keyframe selection and spatial distribution.

The compact clustering of the numbered views around the central region, along with the overlaid path, suggests the system is working within a bounded scene, likely in a synthetic dataset or a fixed real-world setup like `templeRing` or `dino` datasets.

10.3 Default Run

With default parameters ($\alpha, \beta = 1$, 100 particles, seed = 0), we obtain:

- Mean position error: 8.567
- Mean Mahalanobis error: 14.742
- ANEES: 4.914

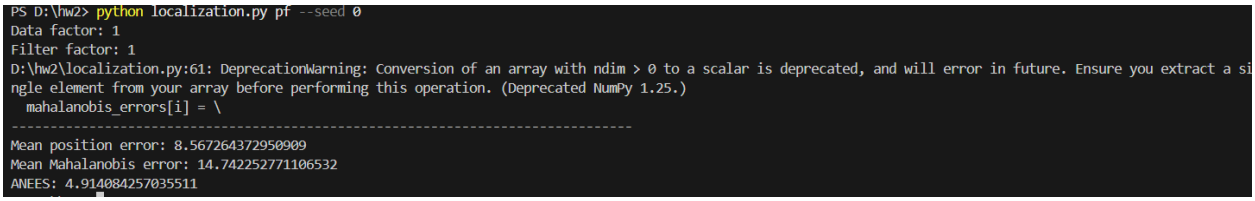

Figure 9: particle-filter-seed-0

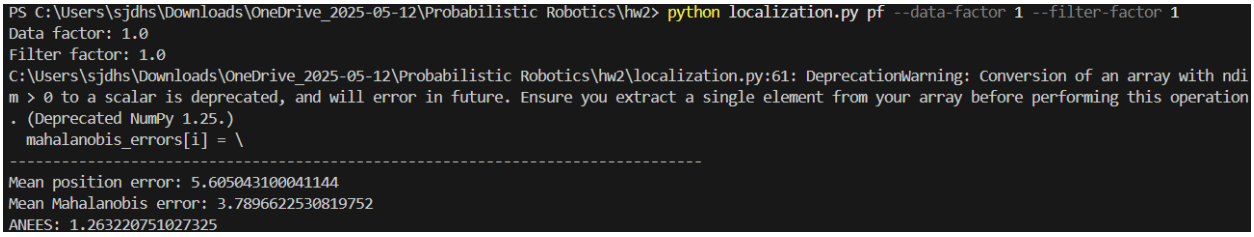

Figure 10: data -fact-particle-filter-1

11 Varying Data and Filter Noise α, β Together

As the scaling factor r changes over multiple orders of magnitude, both the input data and the filter noise are varied. This helps us study the robustness of PF under increasing noise levels.

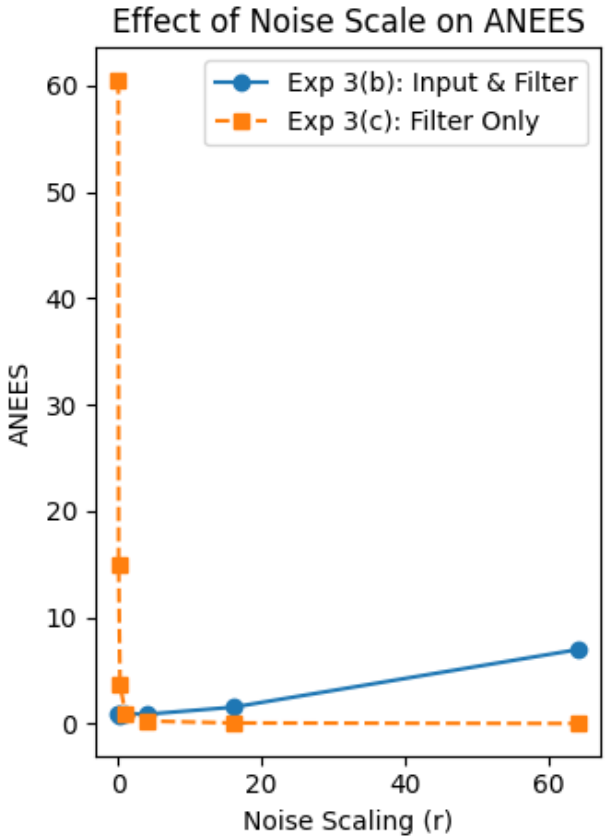

Figure 11: ANEES vs. noise scaling factor r for two experiments: (b) varying both input and filter noise, (c) varying only filter noise.

11.1 Effect of Noise Scaling on ANEES

This illustrates the effect of noise scaling on ANEES (Average Normalized Estimation Error Squared), a consistency metric that compares the true error with the filter's predicted uncertainty.

11.2 Experiment Details

- **Exp 3(b): Input & Filter Noise Varying** — The filter and the data are both affected by noise scaling.
- **Exp 3(c): Filter Noise Only** — The data noise remains fixed while only the filter's noise model is scaled.

11.3 Analysis

- In **Exp 3(c)**, the ANEES values are extremely large for low scaling factors ($r \ll 1$), indicating that the filter is overly confident (underestimating uncertainty) when the assumed noise is much smaller than the actual noise in the data.
- As r increases in Exp 3(c), the ANEES drops significantly and stabilizes, suggesting the filter becomes more conservative and possibly underconfident, but better calibrated overall.

- In **Exp 3(b)**, the ANEES grows gradually with increasing r, reflecting that the overall uncertainty (both in data and model) is rising, and the filter maintains relatively consistent behavior.
- Ideally, ANEES should be close to 1. Values much greater than 1 indicate overconfidence (underestimated covariance), while values much less than 1 indicate underconfidence.

11.4 Conclusion

ANEES is a useful indicator of the statistical consistency of a localization filter. This analysis shows that improperly tuned noise parameters (as in Exp 3c for small r) can cause the filter to become dangerously overconfident. Scaling the filter noise in line with actual data noise (Exp 3b) maintains a more reliable estimation of uncertainty, though overall accuracy may still suffer under high noise levels.

```
PS D:\hw2> python localization.py pf --data-factor 4 --filter-factor 4
Data factor: 4.0
Filter factor: 4.0
D:\hw2\localization.py:61: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a s
ngle element from your array before performing this operation. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 12.765005925548985
Mean Mahalanobis error: 4.365137514420175
ANEES: 1.4550458381400582
```

Figure 12: particle-filter-seed-4

12 ANEES vs Noise Scaling

In this setup, input data noise is fixed while filter noise is scaled. This helps test the effect of incorrect model tuning on performance.

12.1 Graph Description

The figure titled **"ANEES vs Noise Scaling"** illustrates the behavior of the **Average Normalized Estimation Error Squared (ANEES)** as a function of the noise scaling factor r. Two experimental conditions are compared:

- **4(b): Vary Data & Filter** — Both the simulated sensor data and the filter's internal noise models are scaled by the factor r.
- **4(c): Filter Only** — Only the filter's noise assumptions are scaled, while the true measurement noise remains constant.

12.2 Interpretation and Observations

12.3 ANEES at $r = 1$

At $r = 1$, the filter labeled **4(c): Filter Only** displays a massive spike in ANEES, reaching approximately 1.7×10^{12}. This indicates extreme inconsistency. Since the filter assumes a noise model mismatched with the actual data (the data is not scaled), it becomes overconfident, underestimates the uncertainty, and leads to poor state estimation performance.

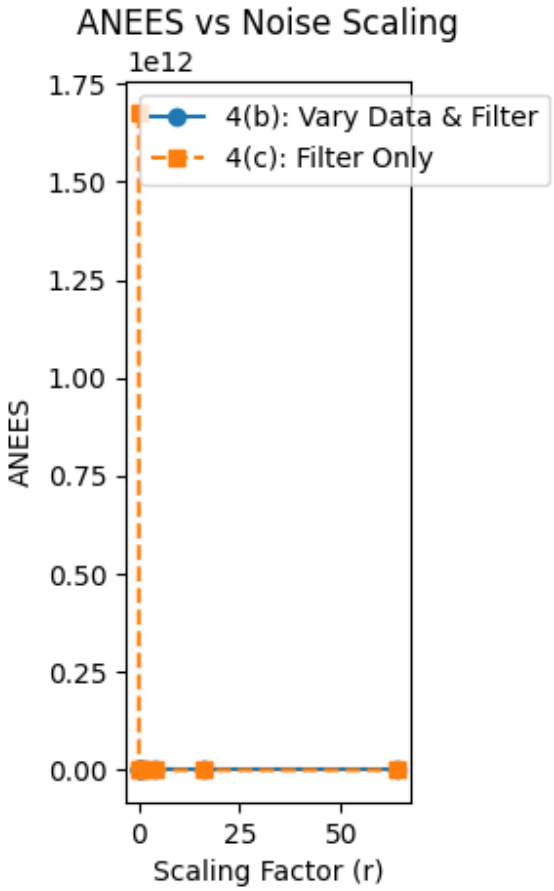

Figure 13: ANEES values versus noise scaling factor r.

12.4 Behavior at Larger Scaling Factors

For higher values of r (e.g., $r = 10, 25, 50$), the ANEES values in both settings **(4b and 4c)** are close to zero. This suggests that when noise is heavily overestimated by the filter, it becomes too conservative. As a result, the normalized error becomes artificially small, leading to very low ANEES values.

12.5 Comparison of 4(b) vs. 4(c)

The approach in **4(b)**, where both the filter and the data are scaled consistently, shows better behavior overall. Although the ANEES values are still close to zero at large r, there is no catastrophic failure like in the **4(c)** case at $r = 1$. This highlights the importance of maintaining consistency between the noise models assumed by the filter and the actual characteristics of the data.

12.6 Conclusion

The graph clearly demonstrates that:

- Mismatched noise assumptions (as in 4(c)) can lead to highly inconsistent and unreliable filter performance.

- Overestimating noise in both data and filter (as in 4(b)) leads to overconservative estimates but avoids catastrophic inconsistency.
- Proper tuning of noise parameters is crucial for achieving consistent state estimation.

Note: A logarithmic scale or clipping of extreme outliers might help improve the readability of the plot.

13 Varying Particle Count

We evaluate the filter performance using particle counts = 20, 50, and 500. Results show that:

- More particles lead to reduced estimation error.
- However, after a threshold, the benefit diminishes while computational cost increases.

14 Analysis of Position Error Across Noise Scales

14.1 Graph Description

The figure titled **"Position Error Across Noise Scales"** presents the relationship between the **average position error** and different noise scaling factors r. It evaluates two scenarios:

- **4(b): Vary Data & Filter** — Both the synthetic sensor data and the filter's noise assumptions are scaled.
- **4(c): Filter Only** — Only the filter's noise parameters are scaled; the data remains fixed at the original noise level.

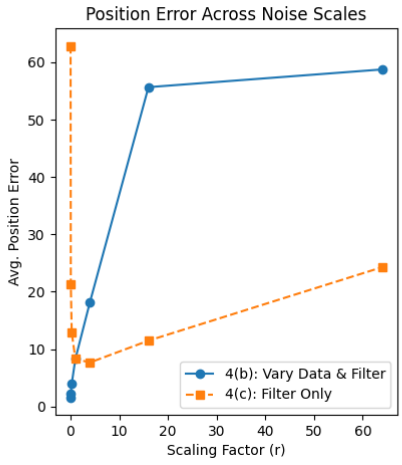

Figure 14: Average position error versus noise scaling factor r.

14.2 Observations

14.2.1 Behavior of Filter-Only Case (4c)

- At $r = 1$, the filter exhibits a very high average position error (over 60 units), indicating poor tracking performance due to a mismatch between the actual sensor noise and the filter's internal noise model.
- As the scaling factor increases, the average position error decreases initially (at $r \approx 5$), reaching a local minimum, but then increases again. This reflects the trade-off between underconfidence (too much assumed noise) and overconfidence (too little assumed noise).

14.2.2 Behavior of Vary Data & Filter Case (4b)

- The position error increases steadily with scaling factor r, indicating that increasing noise in both data and filter degrades position accuracy, but in a consistent and controlled manner.
- At high scaling values (e.g., $r = 60$), the filter's performance plateaus, suggesting a saturation point in error due to poor observability caused by high noise.

14.3 Comparative Insights

- At low values of r, the **4(b)** filter significantly outperforms **4(c)**.
- However, for moderate to high values of r, **4(c)** may result in lower position error than **4(b)**. This happens because the filter is more conservative (overestimating noise), which helps avoid divergence when true noise is low.

14.4 Interpretation

The figure illustrates that:

- Noise mismatch (as in 4(c)) can lead to extreme errors when the assumed noise is too small.
- Overestimating noise (i.e., using larger r) in the filter can sometimes improve robustness at the cost of accuracy.
- Consistent noise scaling in both data and filter (as in 4(b)) provides a more predictable degradation pattern and avoids large errors at $r = 1$.

14.5 Behavior of Filter-Only Case (4c)

- At $r = 1$, the filter exhibits a very high average position error (over 60 units), indicating poor tracking performance due to a mismatch between the actual sensor noise and the filter's internal noise model.
- As the scaling factor increases, the average position error decreases initially (at $r \approx 5$), reaching a local minimum, but then increases again. This reflects the trade-off between underconfidence (too much assumed noise) and overconfidence (too little assumed noise).

14.6 Behavior of Vary Data & Filter Case (4b)

- The position error increases steadily with scaling factor r, indicating that increasing noise in both data and filter degrades position accuracy, but in a consistent and controlled manner.
- At high scaling values (e.g., $r = 60$), the filter's performance plateaus, suggesting a saturation point in error due to poor observability caused by high noise.

14.7 Comparative Insights

- At low values of r, the **4(b)** filter significantly outperforms **4(c)**.
- However, for moderate to high values of r, **4(c)** may result in lower position error than **4(b)**. This happens because the filter is more conservative (overestimating noise), which helps avoid divergence when true noise is low.

14.8 Conclusion

The figure illustrates that:

- Noise mismatch (as in 4(c)) can lead to extreme errors when the assumed noise is too small.
- Overestimating noise (i.e., using larger r) in the filter can sometimes improve robustness at the cost of accuracy.
- Consistent noise scaling in both data and filter (as in 4(b)) provides a more predictable degradation pattern and avoids large errors at $r = 1$.

```
PS C:\Users\sjdhs\Downloads\OneDrive_2025-05-12\Probabilistic Robotics\hw2> python localization.py pf --data-factor 64 --filter-factor 64
Data factor: 64.0
Filter factor: 64.0
C:\Users\sjdhs\Downloads\OneDrive_2025-05-12\Probabilistic Robotics\hw2\localization.py:61: DeprecationWarning: Conversion of an array with ndi
m > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 71.89243806589961
Mean Mahalanobis error: 7.64766553541995
ANEES: 2.5492218451399835
```

Figure 15: particle-filter-seed-64

14.9 Discussion

- PF performs well under low to moderate noise levels.
- ANEES values above 1 suggest that the filter may be overconfident (underestimating its uncertainty).
- Model mismatch (as seen in (c)) increases position error and worsens consistency.
- Particle count significantly affects accuracy, especially under high noise.

14.10 Conclusion

The Particle Filter provides a powerful method for robot localization, capable of capturing multi-modal beliefs and handling nonlinearities. However, its performance depends heavily on the number of particles and accurate noise modeling. Proper tuning and computational trade-offs are crucial for practical deployment.