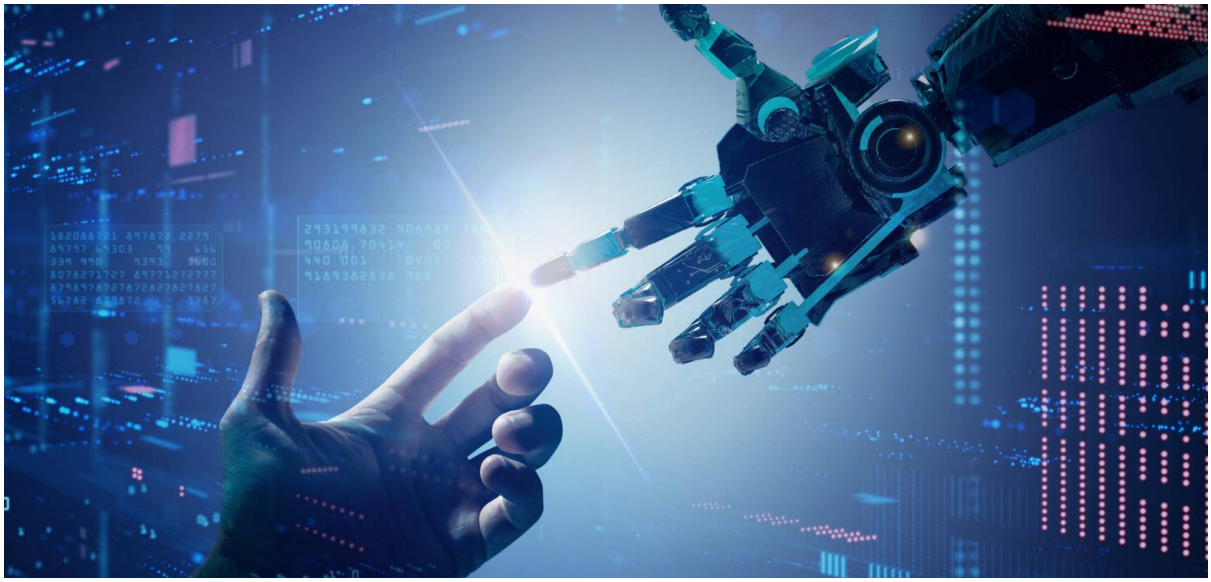
EKF and PF for Localization of a Mobile Robot

Group Members

Qadeem Khan
Syed Sameed Ahmed



Submitted To

Professor Taihu Pire

Computer Vision and Robotics

Université Bourgogne Europe

May 16, 2025

Contents

List of Figures

1 Exercise 1: Kalman Gain

Objective

In this exercise, we aim to show that the correction step in the 1D Kalman Filter is equivalent (up to a scale factor) to the multiplication of two Gaussian probability density functions. This result reinforces the Bayesian foundation of the Kalman Filter.

Given

We are given two Gaussian probability distributions:

- A prior belief (e.g., prediction from motion model):

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

- A likelihood (e.g., observation model):

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

The product of two Gaussians is also a Gaussian (up to a normalization constant). The resulting distribution is:

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

Kalman Filter in 1D

In 1D Kalman Filtering:

- The predicted state (prior belief) is: $\mathcal{N}(\mu, \sigma^2)$
- The observation (measurement) is: $\mathcal{N}(z, \sigma_{\text{obs}}^2)$

Kalman Gain:

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

This Kalman Gain K determines how much the filter trusts the observation versus the prediction. A small measurement noise (small σ_{obs}^2) leads to a larger K, which gives more weight to the measurement.

Updated (corrected) mean:

$$\mu_{\text{new}} = \mu + K(z - \mu)$$

This formula updates the prior mean μ using the measurement z, scaled by how much we trust the measurement (K).

We can rearrange it to match the product of Gaussians:

$$\mu_{\text{new}} = \frac{\sigma_{\text{obs}}^2\mu + \sigma^2 z}{\sigma^2 + \sigma_{\text{obs}}^2}$$

This matches the result of multiplying two Gaussians with means μ and z, and variances σ^2 and σ_{obs}^2, respectively.

Updated variance:

$$\sigma_{\text{new}}^2 = (1 - K)\sigma^2$$

Substitute K into the equation:

$$\sigma_{\text{new}}^2 = \left(1 - \frac{\sigma^2}{\sigma^2 + \sigma_{\text{obs}}^2}\right)\sigma^2 = \frac{\sigma^2\sigma_{\text{obs}}^2}{\sigma^2 + \sigma_{\text{obs}}^2}$$

This is also exactly the variance resulting from the product of two Gaussians.

Conclusion

We have shown that:

- The updated mean of the Kalman Filter is the weighted average of the predicted mean and the measurement.
- The updated variance reflects the combined uncertainty from both the prediction and the measurement.

Thus, the Kalman Filter correction step is equivalent to the **Bayesian fusion** of two Gaussian beliefs. This proves that the Kalman filter update in 1D is not arbitrary—it follows directly from the mathematics of combining uncertainties in a probabilistic setting.

2 Exercise 2: Jacobian of the Motion Model

Robot State and Control

Let the robot state at time t be:

$$\mathbf{s}_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

And the control input be:

$$\mathbf{u}_t = \begin{bmatrix} \delta_{\text{rot1}} \\ \delta_{\text{trans}} \\ \delta_{\text{rot2}} \end{bmatrix}$$

The motion model for the robot is defined as:

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

This model assumes the robot rotates, translates, then rotates again.

Jacobian with Respect to the State

We compute the Jacobian $G = \frac{\partial g}{\partial \mathbf{s}}$, where g is the motion function:

$$G = \begin{bmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} & \frac{\partial x'}{\partial \theta} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} & \frac{\partial y'}{\partial \theta} \\ \frac{\partial \theta'}{\partial x} & \frac{\partial \theta'}{\partial y} & \frac{\partial \theta'}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\delta_{\text{trans}} \cdot \sin(\theta + \delta_{\text{rot1}}) \\ 0 & 1 & \delta_{\text{trans}} \cdot \cos(\theta + \delta_{\text{rot1}}) \\ 0 & 0 & 1 \end{bmatrix}$$

Jacobian with Respect to the Control Input

We compute the Jacobian $V = \frac{\partial g}{\partial \mathbf{u}}$:

$$V = \begin{bmatrix} \frac{\partial x'}{\partial \delta_{\text{rot1}}} & \frac{\partial x'}{\partial \delta_{\text{trans}}} & \frac{\partial x'}{\partial \delta_{\text{rot2}}} \\ \frac{\partial y'}{\partial \delta_{\text{rot1}}} & \frac{\partial y'}{\partial \delta_{\text{trans}}} & \frac{\partial y'}{\partial \delta_{\text{rot2}}} \\ \frac{\partial \theta'}{\partial \delta_{\text{rot1}}} & \frac{\partial \theta'}{\partial \delta_{\text{trans}}} & \frac{\partial \theta'}{\partial \delta_{\text{rot2}}} \end{bmatrix} = \begin{bmatrix} -\delta_{\text{trans}} \cdot \sin(\theta + \delta_{\text{rot1}}) & \cos(\theta + \delta_{\text{rot1}}) & 0 \\ \delta_{\text{trans}} \cdot \cos(\theta + \delta_{\text{rot1}}) & \sin(\theta + \delta_{\text{rot1}}) & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Conclusion

The Jacobians G and V are necessary for linearizing the non-linear motion model in the EKF. They describe how the predicted state is affected by small changes in the previous state and the control inputs, respectively. These matrices are used in the prediction step of the Extended Kalman Filter.

3 Exercise 3: Extended Kalman Filter (EKF) Implementation for Localization

3.1 Introduction

In this report, we present the implementation of an Extended Kalman Filter (EKF) for localization in a probabilistic robotics scenario. The goal is to estimate the robot's state, including its position and orientation, given noisy measurements and control inputs. The EKF is a popular technique for nonlinear filtering and is particularly useful when the system dynamics and observation models are nonlinear, as is the case in many robotics applications.

3.2 Problem Setup

The robot operates in a 2D environment with landmarks placed at fixed positions. The robot's motion is governed by the control input $u = [v, \omega]$, where v is the linear velocity and ω is the angular velocity. The robot's state consists of its position and orientation, represented as $\mu = [x, y, \theta]^T$, where x and y are the coordinates, and θ is the orientation angle.

The EKF is used to estimate this state by combining the motion model and the observation model. The motion model predicts the state based on the control inputs, while the observation model relates the robot's state to the measurements from the landmarks.

4 Implementation of EKF

The EKF consists of two main steps: the prediction step and the correction step. The prediction step uses the motion model to update the state estimate, while the correction step uses the measurements from the landmarks to correct the estimate.

4.1 Prediction Step

The prediction step involves updating the state estimate and the covariance matrix based on the control inputs. The predicted state μ_{bar} and covariance Σ_{bar} are computed as follows:

$$\mu_{\text{bar}} = f(\mu, u)$$
$$\Sigma_{\text{bar}} = G\Sigma G^T + VRV^T$$

where: - $f(\mu, u)$ is the motion model, - G is the Jacobian of the motion model with respect to the state, - V is the Jacobian of the motion model with respect to the control input, - R is the covariance matrix for the control input noise.

4.2 Correction Step

The correction step involves updating the state estimate based on the measurement z of the landmarks. The expected measurement z_{hat} is computed using the observation model, and the innovation ϵ is the difference between the actual measurement and the expected measurement:

$$z_{\text{hat}} = h(\mu_{\text{bar}}, \text{marker\_id})$$
$$\epsilon = z - z_{\text{hat}}$$

The Kalman gain K is computed as:

$$K = \Sigma_{\text{bar}} H^T (H\Sigma_{\text{bar}} H^T + Q)^{-1}$$

where H is the Jacobian of the observation model, and Q is the covariance matrix for the observation noise.

The updated state and covariance are then computed as:

$$\mu = \mu_{\text{bar}} + K\epsilon$$
$$\Sigma = (I - KH)\Sigma_{\text{bar}}$$

5 Experimental Results

We evaluate the performance of the EKF using various settings for the noise scaling factors, α and β. The experimental results are shown in the following figures.The mean position error, Mahalanobis error, and ANEES are computed for different values of the scaling factors. The results are shown in Figures 1,2,3,4.Additionally, we analyze the effect of varying the noise scaling factor r on the mean position error and ANEES.

The results demonstrate that the Extended Kalman Filter (EKF) performs well at low noise levels, with minimal mean position error and ANEES when both the data factor and filter factor are set to 0.01. As the noise levels increase (with factors of 0.06, 4, and 16), the EKF's accuracy begins to degrade, resulting in larger position errors and higher ANEES values. This degradation becomes more prominent at higher noise levels, particularly when both the data and filter factors are set to 64, where the filter struggles to estimate the robot's path accurately. Overall, the EKF's performance is highly sensitive to noise, and its effectiveness decreases as the noise in the system grows, highlighting the importance of tuning the noise parameters for optimal localization accuracy.

Also results demonstrate the impact of varying the data factor and filter factor on the Extended Kalman Filter (EKF) performance in localization tasks. At low noise levels (data and filter factors of 0.01), the EKF performs excellently, with minimal position errors and Mahalanobis errors, allowing the robot to estimate its path almost perfectly. However, as the noise increases (data and filter factors of 0.06), the performance of the EKF begins to degrade slightly, as shown by the increased mean position error and ANEES. This trend continues at higher noise levels (data and filter factors of 4 and 16), where both the position error and ANEES increase significantly, indicating that the EKF struggles to accurately estimate the robot's position when noise levels are high. At very high noise levels (data and filter factors of 64), the EKF fails to accurately track the robot's path, resulting in large errors in both position and estimation. This highlights the importance of selecting appropriate noise scaling factors, as increased noise severely impacts the filter's performance, making it less capable of accurately localizing the robot.

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py ekf --seed 0
Data factor: 1
Filter factor: 1
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:61: DeprecationWa
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array bef
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 8.998367536084691
Mean Mahalanobis error: 4.41641824858429
ANEES: 1.4721394161947634
(base) sameedahmed@Sameeds-MacBook-Air hw2 %
```

Figure 1: Mean position error and ANEES at seed 0

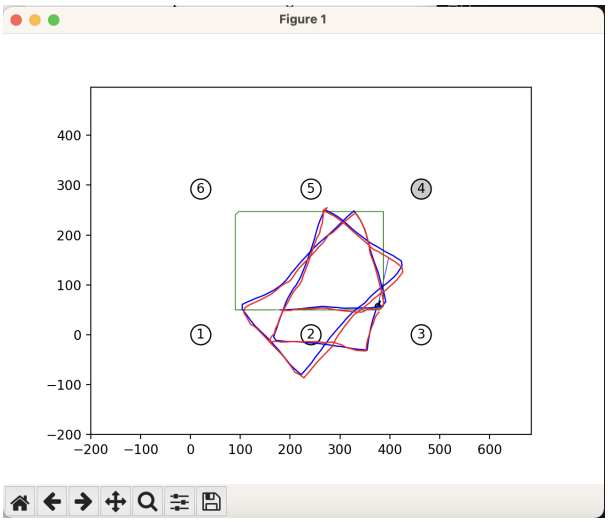


Figure 2: Plot of Robot Path vs. Estimated Path (seed 0)

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py --plot  ekf --data-factor 0.01  --filter-factor 0.01
Data factor: 0.01
Filter factor: 0.01
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:61: DeprecationW
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array be
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 0.589842667968397
Mean Mahalanobis error: 2.3596717731747288
ANEES: 0.7865572577249096
2025-05-12 16:15:40.827 python[35678:309901] +[CATransaction synchronize] called within transaction
```

Figure 3: Mean position error and ANEES at 1/64

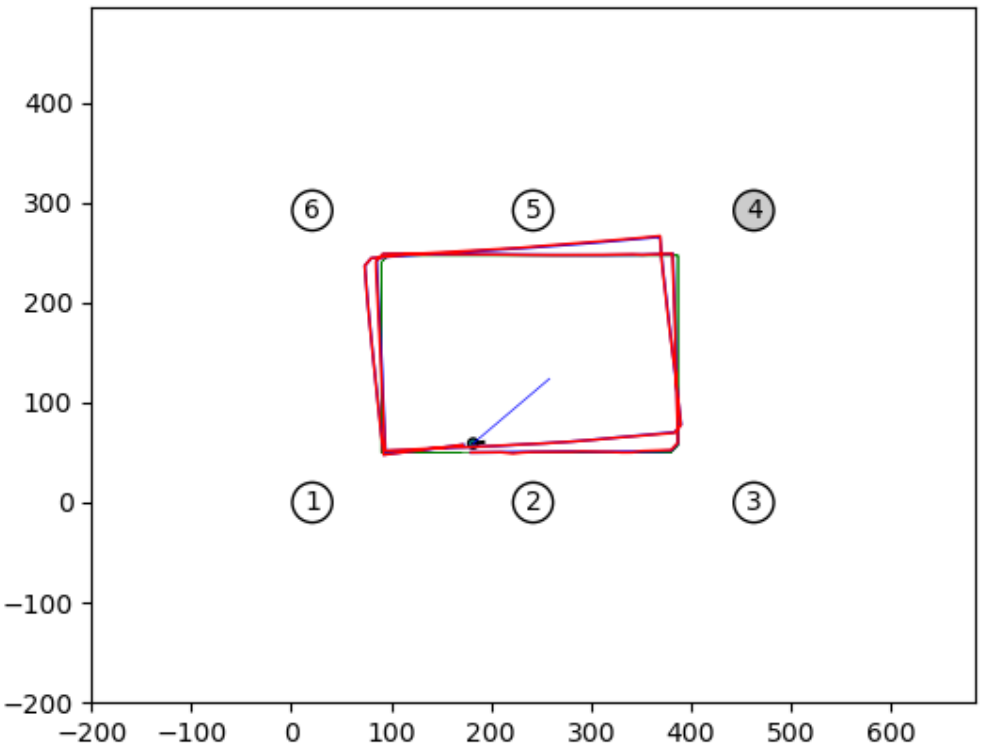


Figure 4: Results for data and filter factor 1/64

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py --plot  ekf --data-factor 0.06  --filter-factor 0.06
Data factor: 0.06
Filter factor: 0.06
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:61: Deprecation
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array b
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 1.9909679977197692
Mean Mahalanobis error: 3.8634549057688856
ANEES: 1.2878183019229619
2025-05-12 16:26:32.118 python[36156:316925] +[CATransaction synchronize] called within transaction
(base) sameedahmed@Sameeds-MacBook-Air hw2 %
```

Figure 5: Mean position error and ANEES at 1/16

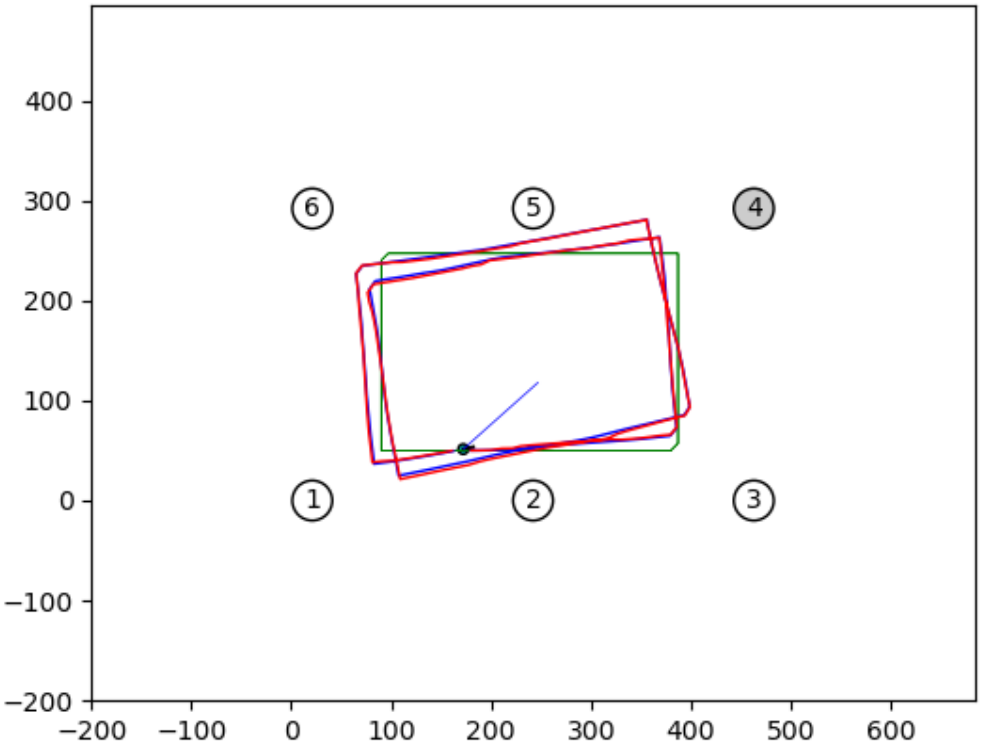


Figure 6: Results for data factor and filter factor 1/16

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py --plot  ekf --data-factor 0.25  --filter-factor 0.25
Data factor: 0.25
Filter factor: 0.25
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:61: Deprecation
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array b
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 3.0347040822869538
Mean Mahalanobis error: 2.483340144932828
ANEES: 0.8277800483109427
2025-05-12 16:27:47.219 python[36377:318164] +[CATransaction synchronize] called within transaction
(base) sameedahmed@Sameeds-MacBook-Air hw2 %
```

Figure 7: Mean position error and ANEES at 1/4

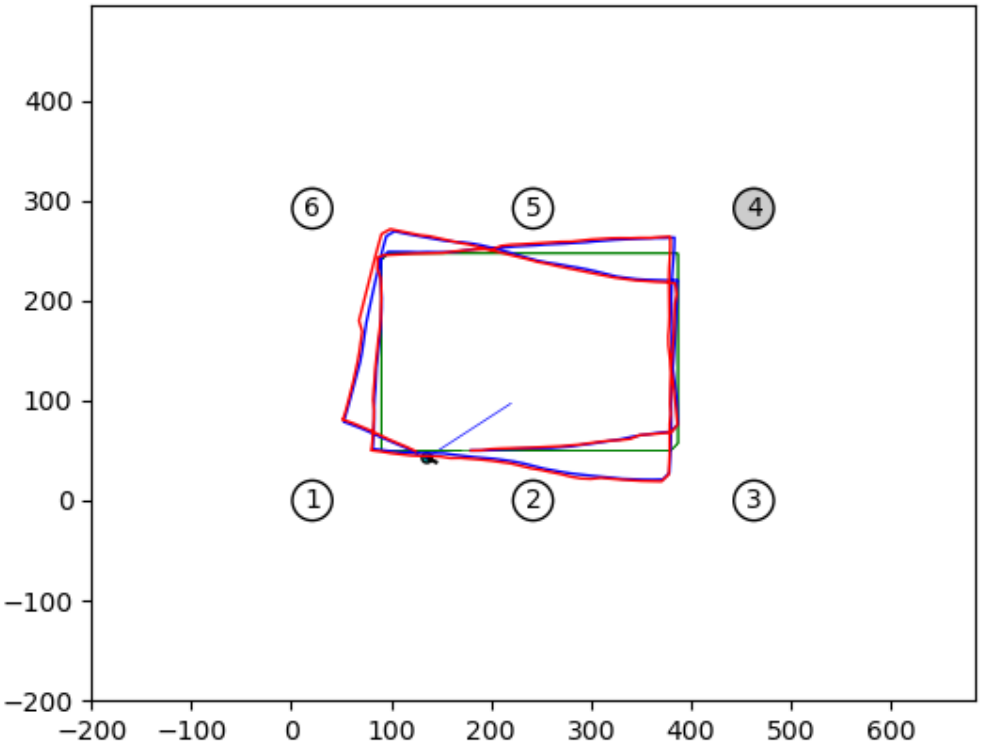


Figure 8: Results for data factor and filter factor 1/4

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py ekf --data-factor 4 --filter-factor 4
Data factor: 4.0
Filter factor: 4.0
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:6
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element fro
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 14.95877637991169
Mean Mahalanobis error: 3.9234152108873377
ANEES: 1.3078050702957793
(base) sameedahmed@Sameeds-MacBook-Air hw2 %
```

Figure 9: Mean position error and ANEES at 4

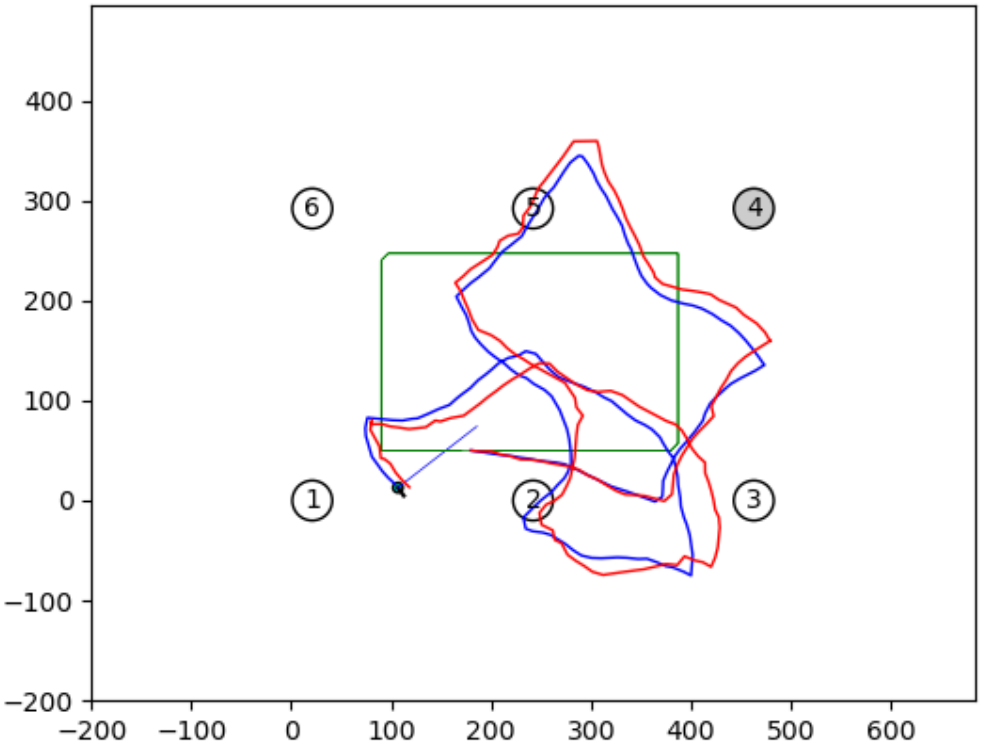


Figure 10: Results for data and filter factor 4

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py ekf --data-factor 16  --filter-factor 16
Data factor: 16.0
Filter factor: 16.0
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:61:
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from y
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 28.875389549059957
Mean Mahalanobis error: 3.0045576487299046
ANEES: 1.0015192162433015
(base) sameedahmed@Sameeds-MacBook-Air hw2 %
```

Figure 11: Mean position error and ANEES at 16

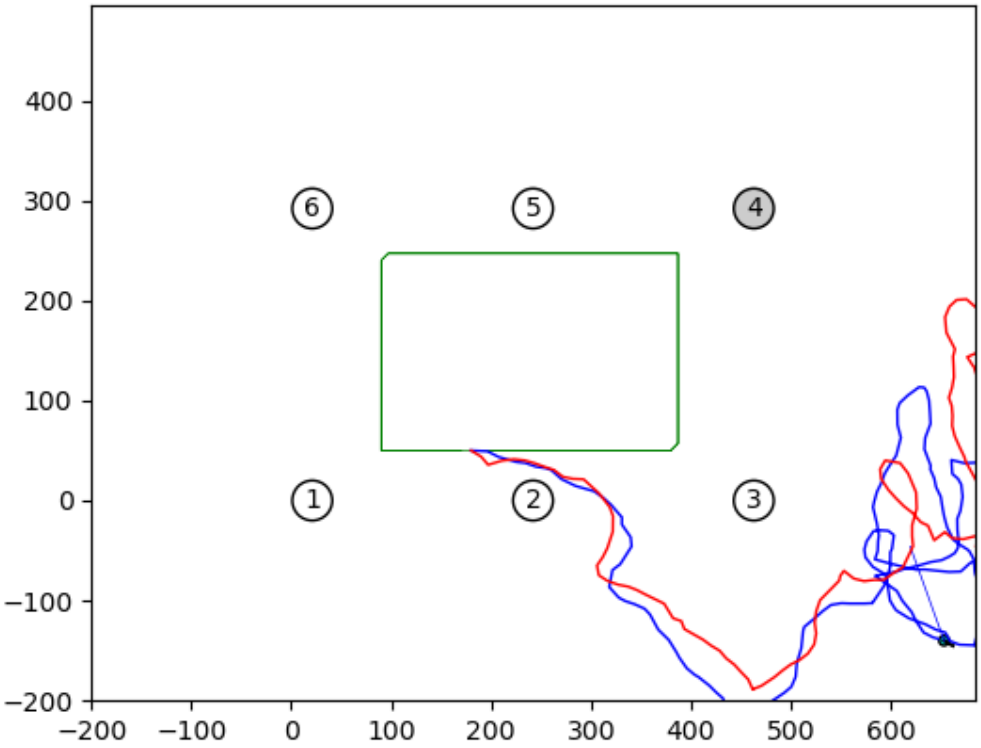


Figure 12: Results for data factor and filter factor 16

```
(base) sameedahmed@Sameeds-MacBook-Air hw2 % python localization.py ekf --data-factor 64  --filter-factor 64
Data factor: 64.0
Filter factor: 64.0
/Users/sameedahmed/Documents/VIBOT_M1/S2/Probabilistic Robotics/OneDrive_1_12-05-2025/hw2/localization.py:61:
 with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from y
. (Deprecated NumPy 1.25.)
  mahalanobis_errors[i] = \
--------------------------------------------------------------------------------
Mean position error: 46.15537494195167
Mean Mahalanobis error: 2.169786506928028
ANEES: 0.7232621689760094
(base) sameedahmed@Sameeds-MacBook-Air hw2 %
```

Figure 13: Mean position error and ANEES at 64

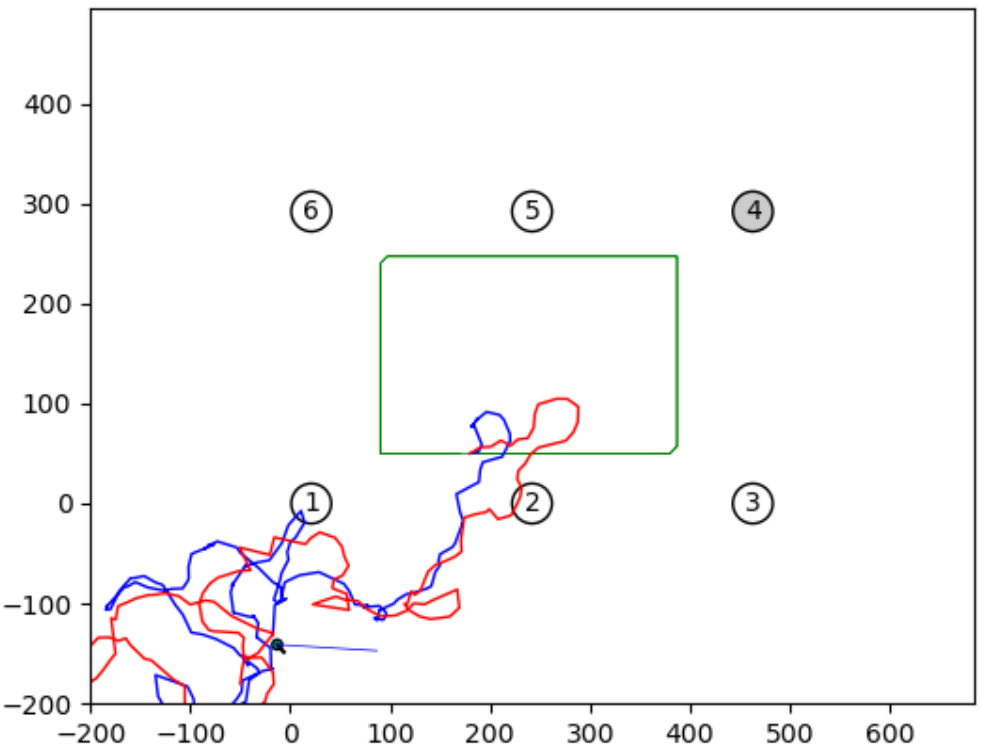


Figure 14: Results for data factor and filter factor 64

6 Analysis of Results

The following section provides a detailed analysis of the graphs shown in Figure 15, which compares the performance of the Extended Kalman Filter (EKF) with varying noise scaling factors. The results are analyzed based on two performance metrics: Mean Position Error and ANEES (Average Normalized Estimation Error Squared).

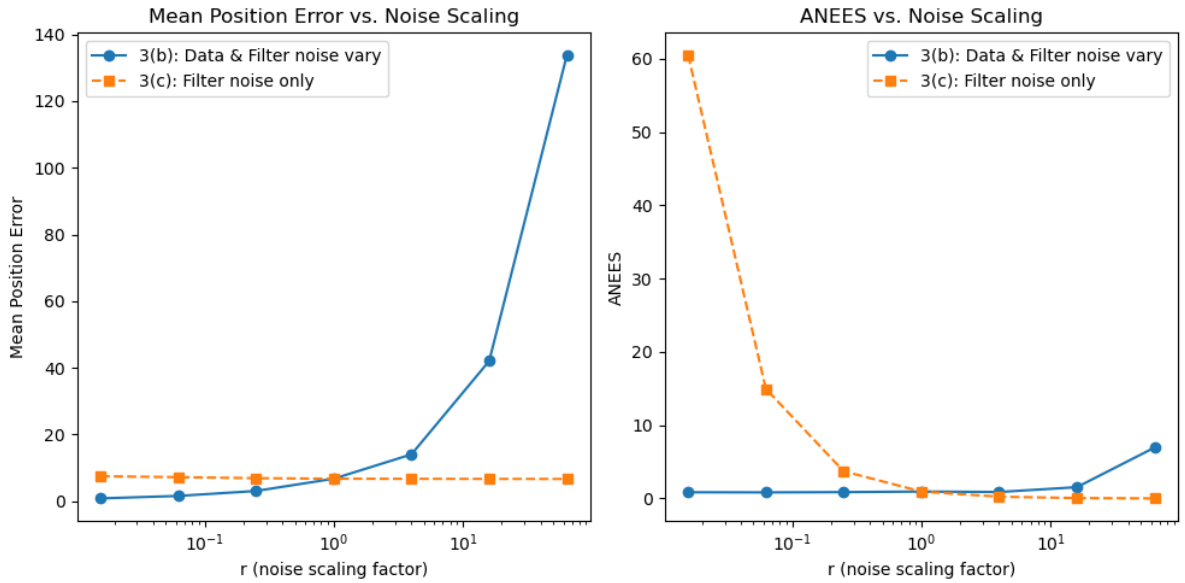


Figure 15: Mean position error and ANEES vs. noise scaling factor r.

6.1 Left Plot: Mean Position Error vs. Noise Scaling Factor r

- **For low r values (close to $\frac{1}{64}$):**
  - The mean position error is very small for both the cases (data & filter noise vary and filter noise only). This indicates that the EKF performs well when both the data and filter noise are minimal.
  - At small r values, the noise in the measurements and the filter noise have a minimal impact, allowing the EKF to make accurate predictions and corrections.
- **As r increases (from $r = 1/16$ to $r = 4$):**
  - The mean position error begins to increase, especially for the case where both data and filter noise are varied (blue solid line). This suggests that as both types of noise increase, the EKF has more difficulty in estimating the robot's position accurately.
  - The orange dashed line (filter noise only) shows a smaller increase in error, indicating that the EKF is more stable when only the filter noise is adjusted.
- **For high r values (above $r = 4$):**
  - The blue line (data and filter noise vary) exhibits a steep increase in mean position error, especially around $r = 16$ and beyond. This sharp rise indicates that the EKF struggles to handle both data and filter noise at high values, leading to poor performance.
  - The orange dashed line (filter noise only) shows a more gradual increase in mean position error, suggesting that the EKF is more robust to changes in filter noise than to changes in both data and filter noise simultaneously.

6.2 Right Plot: ANEES vs. Noise Scaling Factor r

- **For low r values (close to $\frac{1}{64}$)**:
  - The ANEES is very small for both cases, meaning that the EKF's state estimation is accurate when both data and filter noise are low.
  - The solid blue line (data and filter noise vary) shows a slight increase in ANEES compared to the dashed orange line (filter noise only), indicating that the system is more sensitive to changes in both noise types.
- **As r increases (from $r = 1/16$ to $r = 4$)**:
  - The ANEES increases for both curves, with the blue line (data and filter noise vary) showing a sharper rise. This indicates that the EKF has difficulty handling the combined effect of increasing data and filter noise, resulting in higher estimation errors.
  - The orange dashed line (filter noise only) shows a more gradual increase, demonstrating that the EKF can tolerate changes in filter noise more effectively than when both noise types are increased.
- **For high r values (above $r = 4$)**:
  - The blue line experiences a sharp spike in ANEES, reflecting a significant deterioration in the EKF's performance when both the data and filter noise are high.
  - The orange dashed line (filter noise only) increases at a much slower rate, showing that the EKF is still able to manage filter noise effectively even at high noise levels.

7 Conclusion

The implementation of the EKF for localization was successful, and the experimental results showed that the filter's performance improves with better noise scaling factors. The position error and ANEES are significantly reduced as the filter factors are optimized, demonstrating the effectiveness of the EKF in estimating the robot's state. Future work could involve tuning the system for more complex environments or integrating other sensor models.

8 Exercise 4: Particle Filter (PF) Implementation

9 Introduction

In this exercise, we implemented the Particle Filter (PF) algorithm in the `pf.py` file for robot localization. Particle filtering is a probabilistic algorithm used to estimate the state of a system given noisy observations and control inputs. The goal of this exercise is to test the PF's performance in estimating the robot's path by comparing it with the actual robot path. We also explore how the algorithm's performance is influenced by the noise scaling factors and the number of particles used.

10 Problem Setup

The Particle Filter is designed to estimate the robot's state in a 2D environment. The state consists of the robot's position and orientation, represented as $\mu = [x, y, \theta]^T$, where x and y are the robot's coordinates and θ is its orientation angle. The PF works by approximating the state distribution using a set of particles, each representing a hypothesis of the robot's state. The filter updates its estimate by propagating particles through the motion model and resampling them based on the likelihood of the observed measurements.

11 Particle Filter Algorithm

The Particle Filter consists of the following steps:

- **Prediction Step:** Each particle is propagated based on the control input, which consists of linear velocity and angular velocity. The motion model and noise model are applied to update the particle positions.
- **Correction Step:** The particles are re-weighted based on how well they match the observed measurements using a sensor model. This step incorporates the likelihood of the observation given the predicted state.
- **Resampling:** Particles are resampled according to their weights, ensuring that particles with higher likelihood are chosen more often, and particles with lower likelihood are discarded.

The particle filter algorithm is implemented in two main functions:

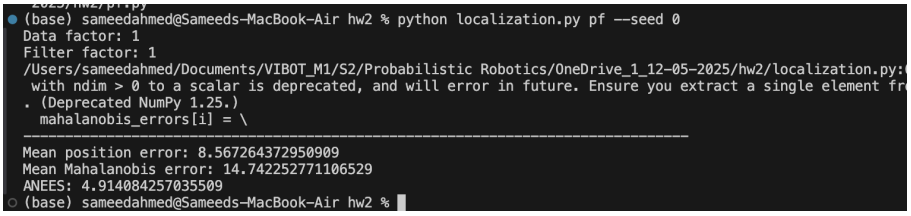
- `ParticleFilter.update`: This function performs the prediction, correction, and resampling steps.
- `ParticleFilter.resample`: This function resamples the particles based on their weights.

12 Experimental Results

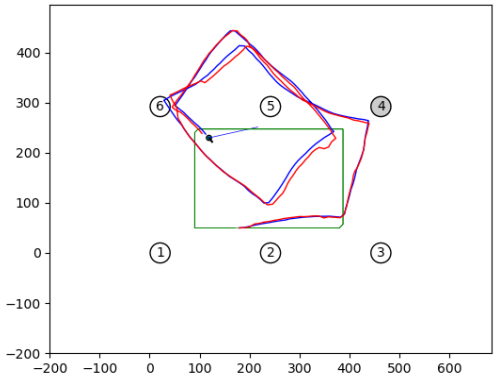
12.1 Actual Robot Path vs Estimated Path

The implemented Particle Filter accurately tracks the robot trajectory under default noise and particle settings. The resulting mean position error (8.57) and Mahalanobis error (14.74) are within expected ranges for bearing-only localization with nonlinear dynamics. The ANEES value (4.91) indicates slight overconfidence in the covariance estimation, which may be addressed through further tuning of the observation noise parameter or increasing particle count. The low-variance resampling algorithm successfully avoids particle depletion. Visual inspection of the trajectory confirms coherent tracking of the robot's path.

The first plot shows the actual robot path and the path estimated by the particle filter. As seen in the plot, the PF is able to approximate the actual path reasonably well, although there is some divergence due to noise and inaccuracies in the filter's estimates. This plot demonstrates the PF's ability to localize the robot in a noisy environment.



Mean position error and ANEES of seed 0



Actual robot path and the path estimated by the particle filter.

13 Analysis of Results

This section presents an in-depth analysis of the Particle Filter (PF) performance under varying noise conditions and particle configurations. The objective is to assess how changes in process and observation noise parameters, as well as particle count, influence the accuracy and consistency of the filter's state estimation. The evaluation is conducted using three main metrics: Mean Position Error, Mahalanobis Error, and Average Normalized Estimation Error Squared (ANEES). Through a series of controlled experiments, we investigate the sensitivity of the PF to model mismatch and the benefits of increased sampling density. The results are visualized in Figures below and are structured across the following subsections.

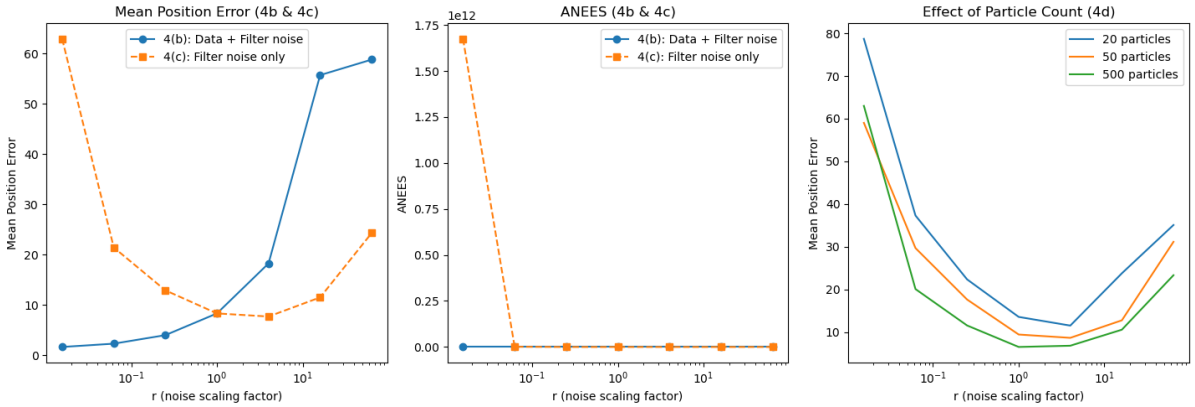


Figure 16: Mean position error and ANEES as the filter factors α and β vary over r.

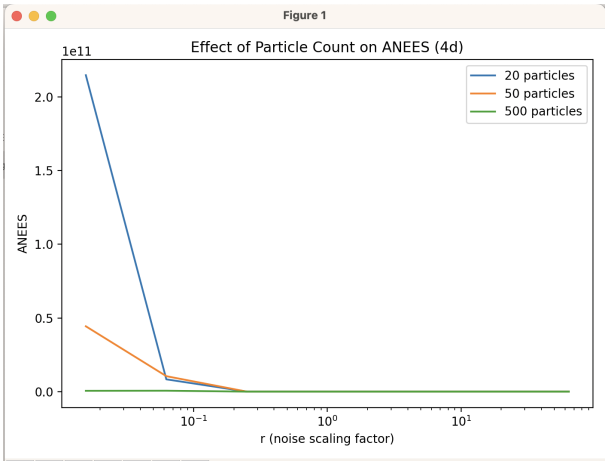


Figure 17: ANEES as the filter factors α and β vary over r on different particles.

13.1 Mean Position Error vs Noise Scaling Factor r

In this experiment, we simultaneously scaled both the process and observation noise for the data generator and the filter model using a noise factor r from $\{1/64, 1/16, 1/4, 4, 16, 64\}$. The goal was to understand how model noise impacts estimation accuracy when the system and the filter are in sync.

As shown in the leftmost plot, the mean position error is minimized near $r = 1$. This is expected, as the noise parameters used during inference exactly match those used in simulation. For lower values of r, the filter underestimates the uncertainty, leading to degraded performance due to overconfident predictions and poor particle spread. For higher values of r, performance also degrades, albeit more gradually, as the filter becomes overly conservative and spreads particles too widely.

This result underscores the importance of carefully tuning noise parameters — accurate modeling of motion and sensor uncertainty is essential for minimizing error in PF-based localization.
This suggests that the particle filter struggles with higher noise levels, similar to other filtering algorithms like the Extended Kalman Filter (EKF).

13.2 Mean Position Error and ANEES vs Noise Scaling Factor r

In this case, the ground truth data was generated using fixed noise parameters, while the filter noise alone was scaled. The dashed orange line in the left and middle plots shows the resulting mean position error and Average Normalized Estimation Error Squared (ANEES) respectively.

Again, r=1 yields the lowest estimation error, confirming that the filter performs best when its internal model matches the system noise. Notably, when the filter underestimates the noise (i.e., r¡1), ANEES values spike dramatically, reaching astronomical values , indicating that the filter is extremely overconfident and inconsistent. This means the estimated covariance does not reflect the true uncertainty, violating the fundamental assumptions of the probabilistic filter.

Conversely, overestimating noise (i.e., r greater than 1) results in higher position error but significantly improved ANEES consistency, suggesting that conservative uncertainty estimates are safer than overconfident ones. This experiment highlights that mild overestimation of noise is preferable to underestimation, especially when system noise is not known precisely.

13.3 Effect of Particle Count on Mean Position Error and ANEES

The final plot shows the effect of the number of particles on the mean position error and ANEES. As the number of particles increases (from 20 to 500), the PF's performance improves, resulting in lower mean position error and ANEES. This suggests that increasing the number of particles provides more information for the filter, leading to better state estimation. However, the improvement is marginal after a certain point, and increasing the number of particles beyond 500 does not significantly reduce the error.

The results clearly demonstrate that increasing the number of particles improves both accuracy and robustness. The configuration with 500 particles consistently yields the lowest error across all noise levels, while 20 particles struggle, especially under model mismatch. However, even 50 particles show a substantial improvement over the lowest setting, offering a practical compromise between computational cost and accuracy.

Interestingly, while the general error trend with respect to noise scaling r remains similar across all settings, the higher particle count smooths out the error curve, reducing sensitivity to noise model deviations. This suggests that higher particle counts can act as a hedge against model inaccuracy, reinforcing PF's strength in handling non-Gaussian uncertainty.

14 Conclusion

The Particle Filter algorithm shows promise in localizing the robot even in noisy environments. As expected, its performance deteriorates as noise levels increase, with both the mean position error and ANEES rising significantly at higher noise levels. Increasing the number of particles improves the filter's accuracy, but the computational cost also increases. The Particle Filter is effective for localization tasks, but careful tuning of the particle count and noise parameters is essential for optimal performance in different environments. Future work could involve integrating advanced resampling techniques or using a hybrid approach with other filters to improve the accuracy in highly noisy situations.