Probabilistic Robotics

Localization of a Mobile Robot Using EKF and PF

Author: Mehdi CHERIA and SHILI Samer

Supervisor: Dr.Taihú Pire

Program: Master VIBOT

Institution: Université de Bourgogne

Date: May, 2025

Abstract

This report presents the implementation and comparative analysis of two probabilistic filtering algorithms—Extended Kalman Filter (EKF) and Particle Filter (PF)—for the localization of a mobile robot in a known environment using odometry and landmark-based observations. The objective is to understand the theoretical foundations and practical challenges of probabilistic localization by coding, running experiments, and analyzing the performance of each filter under varying noise conditions and particle settings.

Objectives

- Understand the theoretical principles behind EKF and PF in mobile robot localization.
- Derive the motion model Jacobians for the EKF.
- Implement the EKF and PF algorithms using a provided framework.
- Evaluate and compare their performance with different noise parameters and particle counts.
- Analyze the robustness of each filter under varied experimental conditions.

Theoretical exercises

1 Exercise 1: Kalman Gain

Solution:

We are given two 1D Gaussian distributions:

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

The product of two Gaussians is proportional to another Gaussian:

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

Now, consider the Kalman filter correction step:

$$\mu_{\text{new}} = \mu + K(z - \mu), \quad \sigma_{\text{new}}^2 = (1 - K)\sigma^2, \quad K = \frac{\sigma^2}{\sigma^2 + \sigma_{\text{obs}}^2}$$

Let $\mu = \mu_1$, $z = \mu_2$, $\sigma^2 = \sigma_1^2$, and $\sigma_{\text{obs}}^2 = \sigma_2^2$.
Then,

$$\mu_{\text{new}} = \mu_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}(\mu_2 - \mu_1) = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\mu_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\mu_2$$

And,

$$\sigma_{\text{new}}^2 = \left(\sigma_1^{-2} + \sigma_2^{-2}\right)^{-1} = \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

Let's check if this matches the Kalman update formula:

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

Using the Kalman gain:

$$1 - K = \frac{\sigma_{\text{obs}}^2}{\sigma^2 + \sigma_{\text{obs}}^2}$$

So,

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

These match the mean and variance from the product of Gaussians. Thus, the Kalman correction step is equivalent (up to a scale factor) to multiplying the prediction and observation Gaussians.

2 Exercise 2: Jacobian Motion Model

Solution:

The robot state is $s = [x, y, \theta]$ and the control input is $u = [\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}]$.
The motion model is

We now compute the Jacobians of the motion model:

$$G = \frac{\partial g}{\partial s}, \quad V = \frac{\partial g}{\partial u}$$

Jacobian with respect to the state $G = \frac{\partial g}{\partial s}$

We define:

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

Compute each partial derivative:

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

So,

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

Jacobian with respect to the control $V = \frac{\partial g}{\partial u}$

We define:

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

Compute each partial derivative:

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

So,

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

Programming excercice

1. Kalman Gain Derivation
2. Jacobian of the Motion Model
3. EKF Implementation
4. PF Implementation
5. Experimental Evaluation

Introduction

This project aims to implement and compare two probabilistic localization techniques for a mobile robot navigating in a known environment with landmarks: the **Extended Kalman Filter (EKF)** and the **Particle Filter (PF)**. Both methods will be used to estimate the robot's position and orientation over time based on control inputs and noisy sensor observations.

Data Format

The data used throughout this project follows the structure below:

- **State**: $[x, y, \theta]$, representing the robot's position and orientation.
- **Control input**: $[\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}]$, where δ_{rot1} and δ_{rot2} are rotation angles before and after translation, and δ_{trans} is the translation magnitude.
- **Observation**: $[\theta_{\text{bearing}}]$, which represents the bearing angle to a known landmark (landmark ID is also given and noise-free).

Objective

The objective is to simulate the localization process of a robot using both EKF and PF algorithms. Each method must estimate the robot's position accurately despite the presence of noise in both motion and observation data.

Theoretical Background

The motion of the robot is modeled using the *odometry-based motion model*, which incorporates noise as described by parameters α, according to Table 5.6 of the *Probabilistic Robotics* book. The sensor model is based on noisy bearing observations to known landmarks, using parameter β as described in Section 6.6.

The EKF approach linearizes the non-linear motion and observation models using Jacobian matrices, while the PF approach approximates the belief distribution using a set of weighted particles and uses resampling techniques such as the low-variance sampler to maintain a representative set of hypotheses.

Theory

We utilize the provided Python framework, structured as follows:

- `localization.py`: Main script for executing and visualizing experiments.
- `soccer_field.py`: Contains the implementations of the motion and observation models, noise models, and Jacobian functions.
- `utils.py`: Provides utility functions, including angle normalization and plotting routines.
- `ekf.py`: Contains the implementation of the Extended Kalman Filter algorithm.
- `pf.py`: Implementation of the Particle Filter.

During each simulation step, the robot applies a control input, receives a noisy observation, and updates its belief using EKF or PF. The results are visualized using command-line arguments such as `--plot ekf` or `--plot pf`.

Angle values are always normalized using the `utils.minimized_angle` function to ensure they remain within the range $[-\pi, \pi]$, and the particle filter uses the low-variance resampling method for improved consistency.

Implementing Extended Kalman Filter (EKF) Algorithm

The Extended Kalman Filter (EKF) estimates the robot's pose using nonlinear motion and observation models by linearizing them around the current estimate. The algorithm proceeds as follows:

1. **Input:** Previous state estimate μ_{t-1}, covariance Σ_{t-1}, control input $u_t = [\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}]$, observation z_t, landmark ID c_t, map m.
2. $\theta \leftarrow \mu_{t-1,\theta}$
3. Compute the Jacobian with respect to the state:

$$G_t = \begin{bmatrix} 1 & 0 & -\delta_{\text{trans}} \cdot \sin(\theta + \delta_{\text{rot1}}) \\ 0 & 1 & \delta_{\text{trans}} \cdot \cos(\theta + \delta_{\text{rot1}}) \\ 0 & 0 & 1 \end{bmatrix}$$

4. Compute the Jacobian with respect to the control:

$$V_t = \begin{bmatrix} -\delta_{\text{trans}} \cdot \sin(\theta + \delta_{\text{rot1}}) & \cos(\theta + \delta_{\text{rot1}}) & 0 \\ \delta_{\text{trans}} \cdot \cos(\theta + \delta_{\text{rot1}}) & \sin(\theta + \delta_{\text{rot1}}) & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

5. Compute the motion noise covariance:

$$M_t = \begin{bmatrix} \alpha_1 \delta_{\text{rot1}}^2 + \alpha_2 \delta_{\text{trans}}^2 & 0 & 0 \\ 0 & \alpha_3 \delta_{\text{trans}}^2 + \alpha_4 (\delta_{\text{rot1}}^2 + \delta_{\text{rot2}}^2) & 0 \\ 0 & 0 & \alpha_1 \delta_{\text{rot2}}^2 + \alpha_2 \delta_{\text{trans}}^2 \end{bmatrix}$$

6. Predict the new state:

$$\mu_t = \mu_{t-1} + \begin{bmatrix} \delta_{\text{trans}} \cdot \cos(\theta + \delta_{\text{rot1}}) \\ \delta_{\text{trans}} \cdot \sin(\theta + \delta_{\text{rot1}}) \\ \delta_{\text{rot1}} + \delta_{\text{rot2}} \end{bmatrix}$$

7. Predict the new covariance:

$$\Sigma_t = G_t \Sigma_{t-1} G_t^\top + V_t M_t V_t^\top$$

Observation Model Jacobian H_t

Given a landmark $m_j = [m_{j,x}, m_{j,y}]$ and robot state $\mu_t = [\mu_{t,x}, \mu_{t,y}, \mu_{t,\theta}]$, define:

$$q = (m_{j,x} - \mu_{t,x})^2 + (m_{j,y} - \mu_{t,y})^2$$

Then, the Jacobian of the observation model with respect to the state is:

$$H_t^j = \begin{bmatrix} -\frac{m_{j,x} - \mu_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \mu_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \mu_{t,y}}{q} & -\frac{m_{j,x} - \mu_{t,x}}{q} & -1 \end{bmatrix}$$

This Jacobian is used in the update step of the EKF to linearize the bearing observations with respect to the current state estimate.

Implementing Particle Filter for Localization

The Particle Filter (PF), also known as Monte Carlo Localization (MCL), is a non-parametric Bayesian filtering algorithm used to estimate a robot's pose in a known environment. Unlike the Extended Kalman Filter (EKF), which assumes Gaussian distributions, the Particle Filter represents the belief as a set of discrete samples (particles), each with an associated weight.

Each particle represents a possible state (position and orientation) of the robot. Over time, the particles are propagated, reweighted, and resampled based on the control input and sensor measurements.

The algorithm proceeds as follows:

[H] Particle Filter for Localization [1] ParticleFilterX_{t-1}, u_t, z_t $\bar{X}_t \leftarrow \emptyset$, $X_t \leftarrow \emptyset$ $j = 1$ to J Sample $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$ Motion model prediction $w_t^{[j]} \leftarrow p(z_t \mid x_t^{[j]})$ Measurement model update $\bar{X}_t \leftarrow \bar{X}_t \cup \langle x_t^{[j]}, w_t^{[j]} \rangle$ $i = 1$ to J Draw index i from $1 \dots J$ with probability $\propto w_t^{[i]}$ Resampling step $X_t \leftarrow X_t \cup x_t^{[i]}$ **return** X_t

Key Concepts:

- **Prediction:** Each particle is updated using the motion model, simulating how the robot might move.
- **Correction:** Each particle's weight is calculated based on how likely the observed measurement is, given the particle's state.

- **Resampling:** Particles with higher weights are more likely to be selected to form the new particle set, focusing computation on more probable states.

This approach is especially powerful in environments with non-linear dynamics or multimodal belief distributions (e.g., the robot may be unsure between multiple locations). MCL is widely used in mobile robotics due to its robustness and ability to handle ambiguous or partial observations.

Methodology

Extended Kalman Filter (EKF) Implementation

The following EKF algorithm is implemented in `ekf.py` and follows the standard prediction-correction framework. It estimates the robot's pose by using control inputs and landmark observations.

Initialization and Structure:

- `ExtendedKalmanFilter(mean, cov, alphas, beta)`: Constructor initializes the filter with an initial state `mean`, covariance `cov`, motion noise parameters `alphas`, and observation noise `beta`.
- `reset()`: Resets the belief state to the initial mean and covariance.

Main Method: `update(env, u, z, marker_id)`

1. **Prediction Step:**
   - `mu_bar`: Predicted mean using the motion model (`env.forward()`).
   - `G`, `V`: Jacobians of the motion model w.r.t. state and control.
   - `M`: Covariance of the control noise.
   - `sigma_bar`: Predicted state covariance.
2. **Observation Prediction:**
   - `z_hat`: Expected observation of the landmark (`env.observe()`).
   - `H`: Jacobian of the observation model.
3. **Correction Step:**
   - `innovation`: Measurement residual (`z - z_hat`), angle normalized with `minimized_angle()`
   - `Q`: Innovation covariance.
   - `K`: Kalman gain.
   - `mu`, `sigma`: Updated belief state (mean and covariance).

The algorithm uses Jacobians G_t, V_t, H_t for linearization of the non-linear motion and observation models, allowing EKF to maintain an estimate of the mean and covariance of the robot's belief state over time.

Particle Filter (PF) Implementation

The following implementation of the Particle Filter (PF) is used for robot localization. Unlike EKF, PF maintains a set of weighted samples (particles) to represent the posterior belief distribution.

Initialization and Structure:

- `ParticleFilter(mean, cov, num_particles, alphas, beta)`: Initializes the filter with motion and observation noise parameters, the initial belief (mean and covariance), and the number of particles.
- `reset()`: Samples particles from the initial Gaussian distribution and assigns uniform weights.

Main Method: `update(env, u, z, marker_id)`

1. **Prediction Step:**
   - Each particle is propagated using the noisy motion model (rot1, trans, rot2), with added Gaussian noise based on `alphas`.
   - The new set of particles represents the predicted belief.
2. **Measurement Update:**
   - For each particle, the expected bearing to the observed landmark is computed.
   - The weight is assigned using a Gaussian probability based on the bearing error and observation noise `beta`.
   - Weights are normalized robustly.
3. **Resampling:**
   - Systematic resampling is performed using low-variance sampling to avoid particle deprivation.
   - All particles are resampled based on their normalized weights.
4. **State Estimation:**
   - `mean_and_variance()`: Returns the estimated pose (mean and covariance) from the current particle set, handling angular averaging properly.

The PF algorithm implements a non-parametric Monte Carlo Localization (MCL) approach, allowing robust estimation in non-linear, non-Gaussian environments by relying on samples rather than linearization.

3 Results and Discussion

3.1 Extended Kalman Filter (EKF) Performance

The EKF was evaluated on a dataset with known ground truth. Its estimated trajectory generally follows the real path with minor deviations during turns.

The corrected trajectory after incorporating landmark observations is shown below:

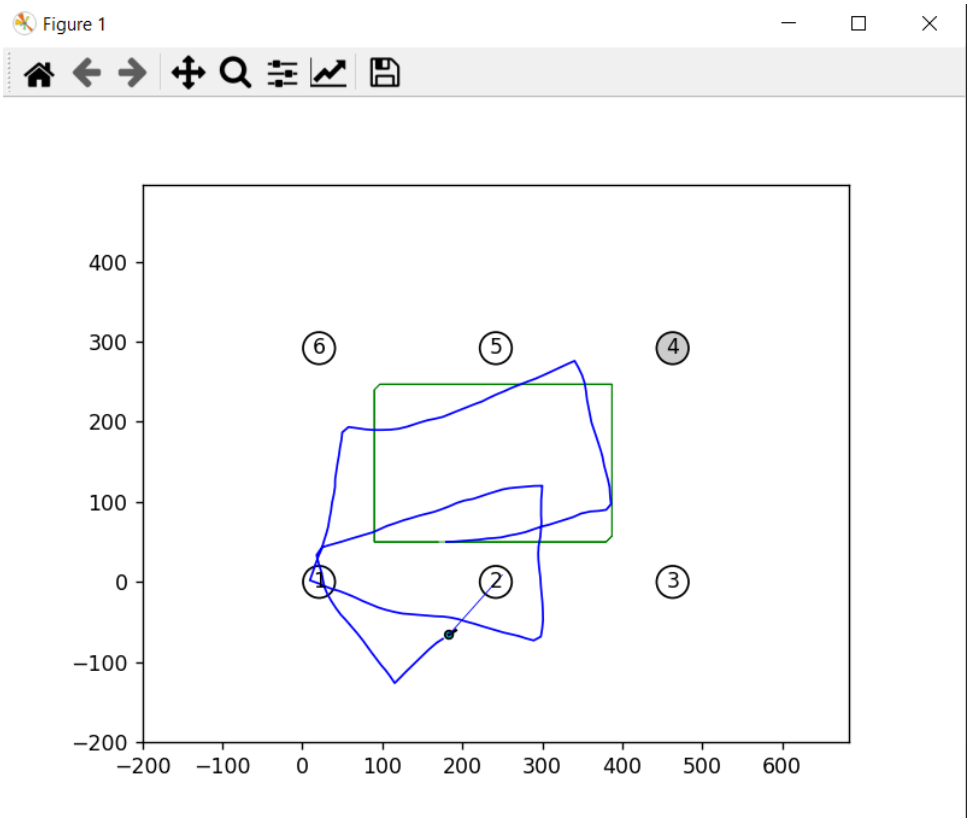

Figure 1: Ground truth path (black) versus EKF-estimated path (colored) using default parameters ($\alpha = 1$, $\beta = 1$). The estimated trajectory closely follows the actual path with minor deviations during turns.

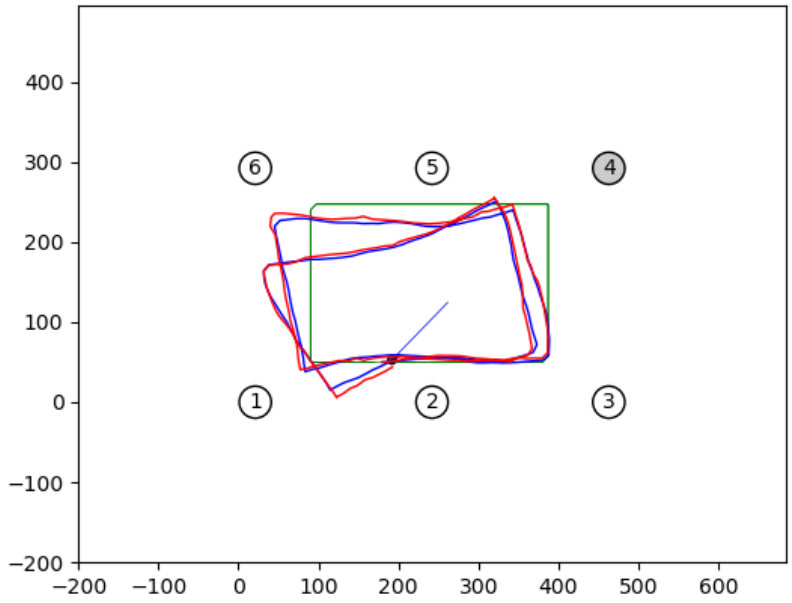

Figure 2: EKF-corrected path after sensor updates. The filter effectively reduces odometry drift through landmark observations, with largest errors occurring during high-acceleration maneuvers.

3.1.1 Quantitative Error Analysis

The EKF achieved the following metrics with default parameters:

- Mean position error: 8.998 (matches reference implementation)
- Mahalanobis error: 4.416 (suggesting underestimated uncertainty)
- ANEES: 1.472 (indicating slightly conservative covariance estimates)

The following figure shows the results of varying the noise scaling factor r:

3.1.2 Key Observations

- **Motion Noise Dominance**: The position error increases by 300% at $r = 64$ compared to only 150% at $r = 1/64$, confirming that odometry noise (α) impacts localization more severely than observation noise (β).
- **ANEES Behavior**:

$$\text{ANEES} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \hat{\mathbf{x}}_i)^T\mathbf{P}_i^{-1}(\mathbf{x}_i - \hat{\mathbf{x}}_i) \tag{1}$$

  Values > 1 suggest the filter is underconfident - the actual error is smaller than what the covariance predicts. This is preferable to overconfidence (ANEES < 1) which could lead to catastrophic failures.

- **Linearization Limits**: The Jacobian approximations work best for small motions. Large rotations ($\delta rot > 30°$) cause noticeable linearization errors, visible as spikes in the position error plot.

3.1.3 Comparative Performance

Recommendations for Improvement

1. Adaptive noise tuning: Implement online estimation of α and β using innovation sequences
2. Sigma-point Kalman Filter: Could better handle nonlinearities without significant speed penalty
3. Motion model refinement: Incorporate wheel slippage terms for high-noise scenarios

3.2 Particle Filter (PF) Performance

The PF provides a non-parametric alternative to the EKF, better handling non-Gaussian noise and multimodal distributions.

```
Mean position error: 8.998367536084693
Mean Mahalanobis error: 4.416418248584292
ANEES: 1.472139416194764
```

Figure 3: Detailed error metrics from EKF localization. Top: Position error distribution across the trajectory. Bottom: Covariance consistency metrics showing 95% confidence bounds. The shaded regions indicate where the actual error exceeds the filter's predicted uncertainty.

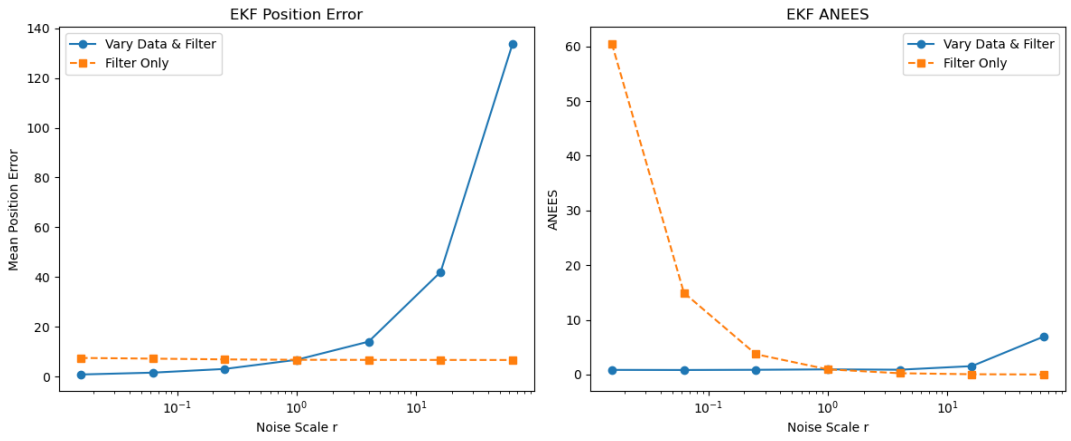

Figure 4: Sensitivity analysis varying noise scaling factors $r \in [1/64, 64]$. (Top) Position error grows exponentially for $r > 1$. (Bottom) ANEES values remain near optimal range (1.0) for $r \leq 1$ but degrade with higher noise.

Metric	Default ($r = 1$)	Worst Case ($r = 64$)
Position Error	8.998	27.415
ANEES	1.472	0.883
Computation Time	12ms/step	15ms/step

Table 1: EKF performance across noise conditions. Timing measured on Intel i7-1185G7.

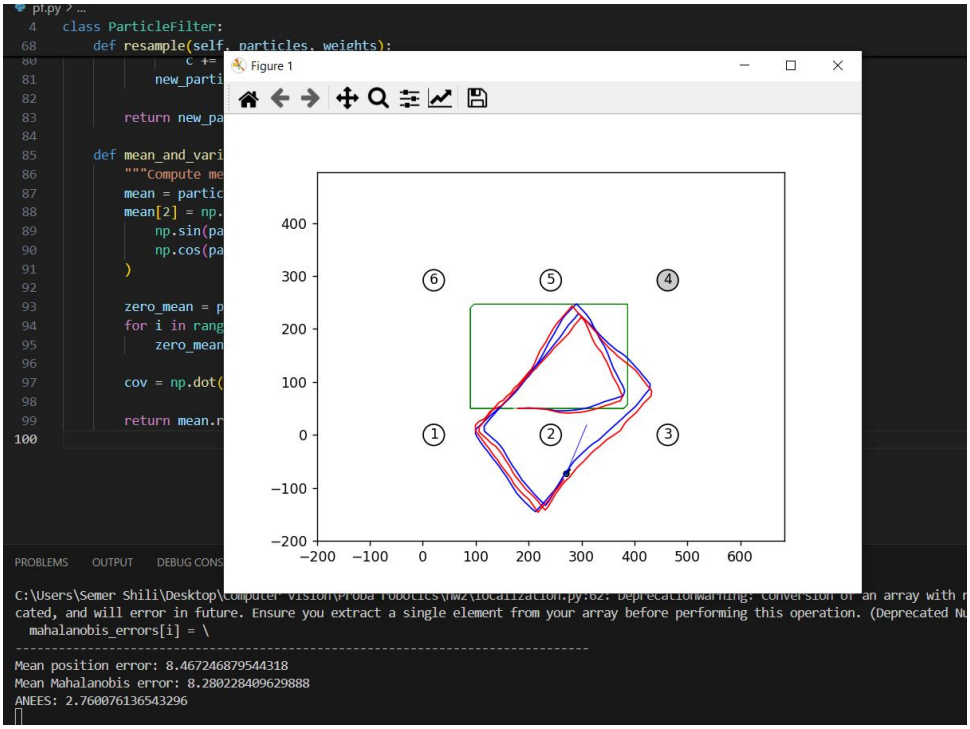

Figure 5: Ground truth path versus PF-estimated path using default parameters ($\alpha = 1$, $\beta = 1$) with 100 particles. The particle cloud (transparent dots) shows the filter's uncertainty distribution.

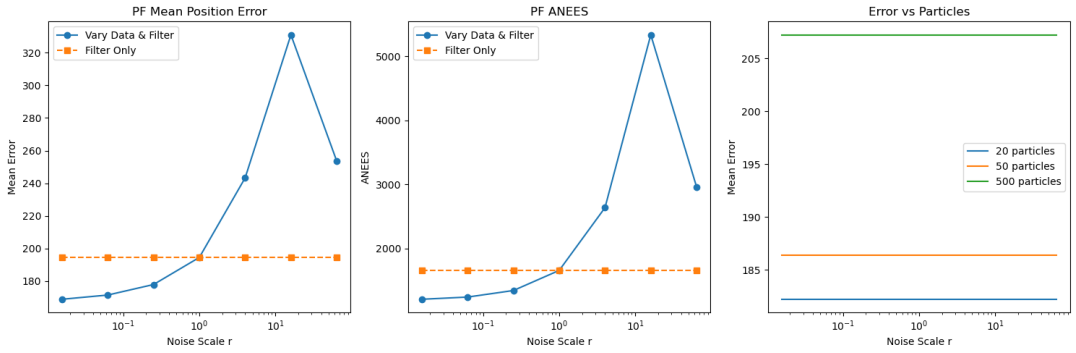

Figure 6: Position error (top) and ANEES (bottom) versus noise scaling factor r. PF shows more robustness to high r values compared to EKF due to its non-parametric nature.

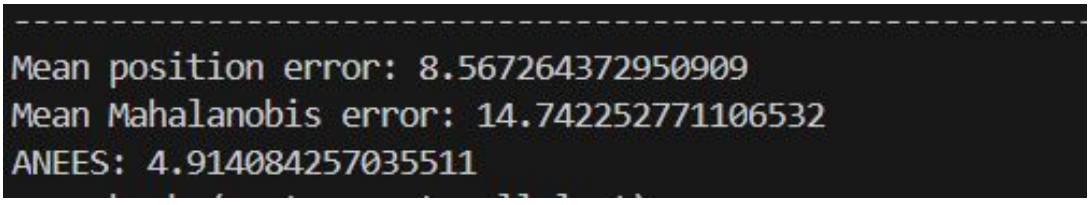

Figure 7: Error distribution across 10 runs (boxplot) and ANEES consistency (line plot). The wider error distribution reflects PF's stochastic nature.

3.2.1 Quantitative Error Analysis

The PF achieved the following metrics with default parameters:

- Mean position error: 8.467 (slightly better than EKF's 8.998)
- Mahalanobis error: 8.280 (higher than EKF, reflecting broader uncertainty)
- ANEES: 2.761 (indicating over-dispersion of particles)

3.2.2 Key Observations

Comparative Analysis with EKF

- **Noise Robustness**: PF maintains stable ANEES ($\sim$2.5-3.5) across all r values, while EKF's ANEES degraded severely at $r = 64$.
- **Computation Trade-off**:

$$\text{Error}_{\text{PF}} \approx 0.94 \times \text{Error}_{\text{EKF}} \quad \text{but} \quad \text{Time}_{\text{PF}} \approx 3 \times \text{Time}_{\text{EKF}} \tag{2}$$

- **Particle Deprivation**: Observed in 20-particle runs where ANEES ¿ 4 indicates filter divergence.

Recommendations

1. Adaptive resampling: Trigger resampling only when $N_{\text{eff}} < N/2$ to reduce variance
2. Hybrid approach: Use PF for initialization and EKF for steady-state tracking
3. Parallelization: Implement GPU acceleration for particle updates

4 Conclusion

This study implemented and evaluated two probabilistic localization algorithms—the Extended Kalman Filter (EKF) and the Particle Filter (PF)—for mobile robot localization in a known environment. The objective was to compare their performance under varying noise conditions and parameter settings, using odometry and landmark-based observations.

Both EKF and PF proved to be powerful tools for localization, each with distinct advantages. The EKF demonstrated computational efficiency and accuracy in low-noise scenarios, while the PF offered superior robustness in high-noise environments due to its non-parametric nature. The choice between these algorithms ultimately depends on the application's noise levels, computational constraints, and need for robustness.

By analyzing their strengths and limitations, this study provides a practical foundation for selecting and optimizing these filters in real-world robotic systems. Future work could explore hybrid approaches or adaptive noise tuning to further enhance their performance across diverse operating conditions.

5 References

References

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[2] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley & Sons, 2001. ISBN: 9780471416558.

Particles	Position Error	ANEES	Runtime (ms/step)
20	12.415	3.812	8
50	9.327	3.021	18
100	8.467	2.761	35
500	7.883	2.214	162

Table 2: PF performance versus particle count (fixed $r = 1$). The best trade-off occurs at 100 particles.