Implementation and Analysis of KF, EKF, and PF for 2D Robot Localization in Simulation

Student: ABDEL-WAHAB, Yussef **PK:** 52009226
Peer Reviewer: Muster MSc, Lucas

***Abstract*—Accurate localization is essential for autonomous mobile robots operating in uncertain and dynamic environments. This paper presents the implementation and comparative evaluation of three probabilistic localization algorithms, the Kalman Filter (KF), Extended Kalman Filter (EKF), and Particle Filter (PF), in a ROS 2-based simulation using the TurtleBot3 robot and Gazebo. Each filter was implemented as a standalone node and integrated into a shared framework to ensure fair benchmarking under identical conditions.**

Experiments were conducted across three test scenarios: standard waypoint navigation, sensor blind spots, and the kidnapped robot problem. Results were visualized using overlaid trajectories on a static map and evaluated using root mean square (RMS) error against ground truth. The KF was found to be inaccurate due to its linear assumptions, while the EKF performed well initially but exhibited drift over time. The PF, despite slightly higher computational cost, showed robust performance in all scenarios and demonstrated superior resilience to uncertainty and global localization failures.

These findings support the PF as the most reliable method for 2D robot localization in nonlinear and partially observable environments. The developed framework offers a reproducible basis for further research in probabilistic state estimation.

I. Introduction

Accurate localization is a fundamental requirement for mobile robots operating in dynamic or unknown environments. Without precise knowledge of its position and orientation, a robot cannot reliably plan paths, avoid obstacles, or interact with its surroundings. However, localization is inherently uncertain due to noisy sensors, wheel slippage, imperfect models, and environmental ambiguity [1]–[3].

To address this, probabilistic state estimation techniques are commonly used to infer the robot's pose from noisy sensor and control data. Among the most established probabilistic methods are the Kalman Filter (KF), the Extended Kalman Filter (EKF), and the Particle Filter (PF). These filters differ in their assumptions about system dynamics, noise characteristics, and computational complexity. The KF is optimal for linear systems with Gaussian noise [2], while the EKF extends this framework to mildly nonlinear systems via local linearization [1], [3]. The PF takes a sampling-based approach, allowing it to handle highly nonlinear and non-Gaussian problems by maintaining a distribution over possible robot states [4], [5].

In this project, these three localization methods are implemented and evaluated in a 2D mobile robot simulation using the TurtleBot3 platform in ROS 2 and Gazebo. The primary goal is to compare their accuracy, robustness, and computational performance under controlled conditions. A common experimental framework was developed to ensure fair comparison, including shared navigation inputs, consistent map data, and a unified evaluation pipeline using ground truth from the simulation [6].

The remainder of this paper is structured as follows: Section II reviews the theoretical foundations and related work on KF, EKF, and PF. Section III describes the simulation setup, implementation details, and methodology. Section IV presents experimental results and comparisons. Finally, Section V concludes with a summary and directions for future work.

II. State of the Art

Probabilistic localization methods aim to estimate a robot's state, typically position and orientation, by fusing sensor data with motion models under uncertainty [7]. Three widely used approaches are the Kalman Filter (KF), the Extended Kalman Filter (EKF), and the Particle Filter (PF), each offering different trade-offs between accuracy, computational cost, and robustness to nonlinearity and noise [1], [3], [8].

A. Kalman Filter (KF)

The Kalman Filter is a recursive Bayesian estimator optimal for linear systems with Gaussian noise [2]. It models the system dynamics and sensor measurements using linear equations and propagates a Gaussian belief over the state space. Due to its efficiency and simplicity, KF has been widely applied in mobile robotics for tasks such as sensor fusion and pose tracking [9]. However, its performance degrades significantly in the presence of non-linear motion or observation models, which are common in practical robotics applications [8].

Algorithm 1 Kalman_filter($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t$), based on [1]

```
1: $\bar{\boldsymbol{\mu}}_t = A_t \boldsymbol{\mu}_{t-1} + B_t \mathbf{u}_t$
2: $\bar{\boldsymbol{\Sigma}}_t = A_t \boldsymbol{\Sigma}_{t-1} A_t^\top + R_t$
3: $K_t = \bar{\boldsymbol{\Sigma}}_t C_t^\top (C_t \bar{\boldsymbol{\Sigma}}_t C_t^\top + Q_t)^{-1}$
4: $\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + K_t(\mathbf{z}_t - C_t \bar{\boldsymbol{\mu}}_t)$
5: $\boldsymbol{\Sigma}_t = (I - K_t C_t) \bar{\boldsymbol{\Sigma}}_t$
6:
7: return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$ =0
```

Kalman Filter Notation and Dimensions

- $\mathbf{x}_t \in \mathbb{R}^n$: State vector at time t, e.g., $[x, y, \theta]^T$ for 2D localization.
- $\boldsymbol{\mu}_t \in \mathbb{R}^n$: Estimated mean (expected state) at time t.
- $\boldsymbol{\Sigma}_t \in \mathbb{R}^{n \times n}$: Covariance matrix representing uncertainty of the state estimate.

- $\mathbf{u}_t \in \mathbb{R}^m$: Control input vector at time t, e.g., $[v, \omega]^T$ for linear and angular velocity.
- $\mathbf{z}_t \in \mathbb{R}^k$: Measurement vector at time t, e.g., observed pose from sensors.
- $A_t \in \mathbb{R}^{n\times n}$: State transition matrix that models system dynamics.
- $B_t \in \mathbb{R}^{n\times m}$: Control input matrix mapping $\mathbf{u}_t$ to state space.
- $C_t \in \mathbb{R}^{k\times n}$: Observation matrix mapping states to measurements.
- $R_t \in \mathbb{R}^{n\times n}$: Covariance of process noise ε_t, modeling uncertainty in motion.
- $Q_t \in \mathbb{R}^{k\times k}$: Covariance of measurement noise δ_t, modeling sensor noise.
- $K_t \in \mathbb{R}^{n\times k}$: Kalman gain matrix that weights correction based on sensor reliability.
- $I \in \mathbb{R}^{n\times n}$: Identity matrix.
- $\bar{\boldsymbol{\mu}}_t, \bar{\boldsymbol{\Sigma}}_t$: Predicted mean and covariance before correction step.

B. Extended Kalman Filter (EKF)

The EKF extends the KF to non-linear systems by linearizing both the motion and observation models using a first-order Taylor expansion around the current state estimate [1], [3]. It remains efficient and compact, maintaining a Gaussian belief, but it assumes that the linearization is sufficiently accurate. EKF-based localization has been widely adopted in robotic systems due to its balance between computational efficiency and the ability to handle moderate nonlinearities. However, EKF is sensitive to large initial errors or strong non-linearities, which can lead to divergence [10].

Algorithm 2 Extended_Kalman_filter($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t$), based on [1]

```
1: $\bar{\boldsymbol{\mu}}_t = g(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})$
2: $\bar{\boldsymbol{\Sigma}}_t = G_t \boldsymbol{\Sigma}_{t-1} G_t^\top + R_t$
3: $K_t = \bar{\boldsymbol{\Sigma}}_t H_t^\top (H_t \bar{\boldsymbol{\Sigma}}_t H_t^\top + Q_t)^{-1}$
4: $\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + K_t(\mathbf{z}_t - h(\bar{\boldsymbol{\mu}}_t))$
5: $\boldsymbol{\Sigma}_t = (I - K_t H_t)\bar{\boldsymbol{\Sigma}}_t$
6:
7: return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$ =0
```

Differences from the Kalman Filter

The Extended Kalman Filter (EKF) generalizes the linear Kalman Filter by replacing the linear system models with nonlinear functions and their Jacobians:

- $g(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})$ replaces $A_t\boldsymbol{\mu}_{t-1} + B_t\mathbf{u}_t$:
Nonlinear motion model used to predict the next state.

- $G_t = \frac{\partial g}{\partial \mathbf{x}}\Big|_{\boldsymbol{\mu}_{t-1}, \mathbf{u}_t}$ replaces A_t:
Jacobian of the motion model with respect to the state, evaluated at the current estimate.

- $h(\bar{\boldsymbol{\mu}}_t)$ replaces $C_t\bar{\boldsymbol{\mu}}_t$:
Nonlinear observation model used to predict the expected measurement.

- $H_t = \frac{\partial h}{\partial \mathbf{x}}\big|_{\bar{\boldsymbol{\mu}}_t}$ replaces C_t:
Jacobian of the observation model with respect to the state, evaluated at the predicted mean.

C. Particle Filter (PF)

The Particle Filter, also known as Monte Carlo Localization (MCL), represents the robot's belief as a set of weighted samples (particles) [4]. It is capable of approximating arbitrary, non-Gaussian, and multimodal distributions, making it well suited for global localization and situations with ambiguous sensor data [5], [11]. PFs are robust to poor initial estimates and complex dynamics but require significantly more computational resources compared to KF and EKF. The quality of the approximation depends on the number of particles and the resampling strategy used [8].

Algorithm 3 MCL($X_{t-1}, \mathbf{u}_t, \mathbf{z}_t, m$), based on [1]

```
1: $\bar{X}_t = X_t = \emptyset$
2: for $m = 1$ to $M$ do
3:     $x_t^{[m]}$ = sample_motion_model($\mathbf{u}_t, x_{t-1}^{[m]}$)
4:     $w_t^{[m]}$ = measurement_model($\mathbf{z}_t, x_t^{[m]}, m$)
5:     $\bar{X}_t = \bar{X}_t \cup \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: end for
7: for $m = 1$ to $M$ do
8:     draw $i$ with probability $\propto w_t^{[i]}$
9:     add $x_t^{[i]}$ to $X_t$
10: end for
11:
12: return $X_t$ =0
```

Explanation of Variables (MCL)

- X_t: Set of particles representing the belief over states at time t.
- $x_t^{[m]}$: The m^{th} sampled particle at time t.
- $w_t^{[m]}$: Importance weight for particle $x_t^{[m]}$, based on likelihood of observation.
- M: Total number of particles.
- $\bar{X}_t$: Temporary set of weighted particles before resampling.
- $\mathbf{u}_t$: Control input (e.g., velocities).
- $\mathbf{z}_t$: Sensor observation at time t.
- m: The map of the environment.
- sample_motion_model(): Applies a probabilistic motion model to generate predicted state.
- measurement_model(): Calculates the likelihood of observation $\mathbf{z}_t$ given the predicted state and map.

A key aspect of the PF is the maintenance and normalization of particle weights. After each motion and sensor update, the importance weight $w_t^{[m]}$ of each particle $x_t^{[m]}$ is updated based on the likelihood of the current observation $\mathbf{z}_t$ given the particle's predicted state and the known map. This is typically computed using a measurement model, such as a Gaussian sensor model, which assigns higher weights to particles whose predicted measurements closely match the actual observations.

To address particle degeneracy,where a majority of the weights converge to near zero,it is common to perform resampling. However, resampling is typically triggered conditionally, based on the *effective sample size* N_{eff}, which estimates the number of meaningful (non-negligibly weighted) particles:

$$N_{\text{eff}} = \frac{1}{\sum_{m=1}^{M}(w_t^{[m]})^2} \tag{1}$$

If N_{eff} falls below a predefined threshold, resampling is performed to discard low-weight particles and replicate high-weight ones. This helps maintain particle diversity while preventing computational waste on unlikely hypotheses. Weights are normalized after each update step to ensure that

$\sum_{m=1}^{M} w_t^{[m]} = 1$.

D. Comparison

While the KF is optimal for linear systems, the EKF extends applicability to mildly non-linear environments at a low additional cost [8]. The PF offers the greatest flexibility and robustness, particularly in complex or uncertain scenarios, but at the cost of computational overhead [5]. In practice, the choice among these filters depends on the system model, computational constraints, and the degree of non-linearity and noise present in the environment.

III. Materials and Methods

This project involved the implementation and comparison of three probabilistic localization algorithms KF, EKF, and PF within a ROS 2-based simulation using the TurtleBot3 robot. The experiments were conducted in a Gazebo environment integrated with ROS 2 Humble. A custom launch file was created to spawn the robot, initialize the navigation stack with a precomputed map, and open RViz for live visualization of the robot's pose, coordinate frames, and waypoints. This launch file also manages the execution of the three localization algorithms, each implemented in a separate C++ node.

The **KF** assumes linear motion and observation models, applying Gaussian prediction and correction steps. The **EKF** extends this by linearizing nonlinear models via Jacobians. The **PF** uses a set of weighted particles to approximate the belief distribution, offering robustness in the presence of non-Gaussian noise and ambiguous observations. Each filter node subscribes to control input topics (`/odom`, `/imu`) and optionally to laser scan data (`/scan`). An object-oriented architecture was followed, with each filter encapsulating `predict(...)` and `correct(...)` methods. The estimated poses are published to `/prediction` using the `geometry_msgs::PoseWithCovarianceStamped` message type.

During simulation setup, collision-related issues were encountered due to Gazebo's default `step_size`, which limited the physics update rate. Reducing this parameter increased simulation fidelity and prevented the robot's base from clipping through the ground, avoiding downstream errors in mapping and obstacle detection. To minimize the amount of custom setup, the predefined `turtlebot3_gazebo` package was used to spawn the robot. In addition to launching Gazebo and RViz GUIs, this configuration enabled seamless switching between the custom localization filters for evaluation.

To produce a high-fidelity reference map, simulated noise in lidar, IMU, and odometry sensors was disabled. The lidar's maximum range was increased to ensure comprehensive coverage of the environment. These changes allowed for more accurate alignment of scanned obstacles with the environment geometry. When generating the map using the Cartographer SLAM node, its range parameters were adjusted accordingly to avoid mismatches with the robot's scan configuration. After the mapping process, all altered sensor parameters were reset to their default values to ensure a fair and realistic evaluation of each filter under noisy conditions.

The `nav2` stack was used as a foundational framework to integrate the custom localization filters into a ROS 2-compatible system. Although `nav2` provides full navigation capabilities, including Adaptive Monte Carlo Localization (AMCL) and path planning, these features were deliberately not used. Instead, only the map server and costmap infrastructure were utilized to support the localization pipeline. A separate C++ node was implemented to send a fixed sequence of four waypoints to the robot, ensuring consistent motion for evaluating the KF, EKF, and PF filters independently of the navigation stack.

A. Motion and Sensor Models

The motion model was based on velocity commands derived from odometry and IMU twist data, which are commonly fused in mobile robot localization systems [12]. In the linear Kalman Filter (KF), the robot's motion was modeled with a constant velocity assumption and a linear state transition, which is valid only for small or linear motions [8].

The Extended Kalman Filter (EKF) employed a nonlinear motion model, linearized using a first-order Taylor expansion. The robot was modeled as a unicycle with state vector $\mathbf{x} = [x, y, \theta]^T$, where x and y are the robot's position and θ its orientation. The control input was $\mathbf{u} = [v, \omega]^T$ (translational and angular velocity), and the continuous-time motion model is standard for differential-drive robots [13]:

$$\begin{aligned} \dot{x} &= v \cdot \cos(\theta) \\ \dot{y} &= v \cdot \sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{2}$$

This model was discretized for use in prediction:

$$\begin{aligned} x_{k+1} &= x_k + v \cdot \cos(\theta_k) \cdot \Delta t \\ y_{k+1} &= y_k + v \cdot \sin(\theta_k) \cdot \Delta t \\ \theta_{k+1} &= \theta_k + \omega \cdot \Delta t \end{aligned} \tag{3}$$

To apply the EKF, the model was linearized around the current estimate using the Jacobian of the motion model with respect to the state:

$$F_k = \begin{bmatrix} 1 & 0 & -v \cdot \sin(\theta_k) \cdot \Delta t \\ 0 & 1 & v \cdot \cos(\theta_k) \cdot \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Optionally, the Jacobian with respect to the control input can be expressed as:

$$G_k = \begin{bmatrix} \cos(\theta_k) \cdot \Delta t & 0 \\ \sin(\theta_k) \cdot \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \tag{5}$$

These Jacobians were used to propagate the state covariance and integrate motion uncertainty during the prediction step [1].

The observation model measured the difference between the predicted and observed poses and added Gaussian noise based on sensor uncertainties. In the Particle Filter (PF), particle weights were computed from a Gaussian likelihood function over pose error [4], and resampling was performed using the normalized weights. This approach allowed the PF to represent multimodal beliefs and recover from poor initializations, which is essential in global localization and recovery scenarios [14].

B. Visualization and Ground Truth Alignment

To compare the estimated poses against the true robot position, a custom node extracted the ground truth from Gazebo's `/gazebo/model_states` topic and published it to `/ground_truth_pose`. This allowed real-time visualization of both prediction and ground truth in RViz, independently of the odometry pose given by the localized robot [15]. Coordinate frames, laser scans, and map overlays were also displayed for reference.

Localization performance was quantitatively evaluated using Root Mean Square Error (RMSE) between each filter's predicted pose and Gazebo's ground truth, as is common in benchmarking localization algorithms [16]. Since the nodes all get launched with the same launchfile, they were tested under identical conditions, including the same initial pose and waypoint sequence. For robustness testing, scenarios with incorrect initializations (kidnapped robot problem) were also simulated [1].

C. Software and Tools

The implementation was completed in C++ using ROS 2 Humble on Ubuntu 22.04 running inside a Docker container. Core ROS packages used include `nav2`, `tf2`, and `geometry_msgs`. Visualization was handled with RViz2, and simulation was run in Gazebo. Version control was maintained via Git. This ROS-Gazebo toolchain is widely adopted for simulation-based benchmarking in mobile robotics [17], [18].

IV. Results

V. Experimental Results

To evaluate the performance of the implemented localization filters, a series of experiments were conducted in simulation. Each filter's estimated trajectory was compared against the Gazebo-provided ground truth, and the results were visualized with matplotlib by overlaying the robot paths on a static map image. This section discusses the outcome of three distinct test scenarios.

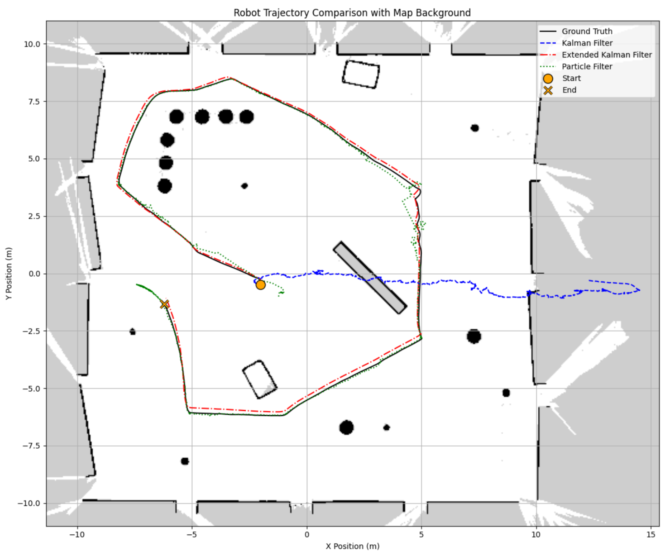

Fig. 1: Robot trajectory comparison during standard waypoint navigation. Includes Kalman Filter (Blue), Extended Kalman Filter (Red), Particle Filter (Green), and Ground Truth (Black).

A. Benchmark Evaluation

In the first test (Fig. 1), the robot was guided through a fixed sequence of waypoints while the estimated positions from all three filters were recorded and plotted. This benchmark scenario served as the foundation for calculating the root mean square (RMS) error for each method.

- **Kalman Filter (KF)**: RMS Error = 18.28 m
- **Extended Kalman Filter (EKF)**: RMS Error = 0.43 m
- **Particle Filter (PF)**: RMS Error = 0.33 m

While the EKF initially achieved a lower RMS error ($0.43\,\text{m}$) compared to the Particle Filter ($0.33\,\text{m}$), it was observed that the EKF's estimate gradually drifted over time due to the accumulation of linearization errors. In contrast, the PF, although slightly less accurate in terms of RMS in this benchmark, consistently recovered from deviations and maintained reliable tracking throughout the trajectory. This robustness makes the PF more suitable for long-term localization in nonlinear and uncertain environments. The Kalman Filter, due to its strict assumption of linear system dynamics, quickly diverged from the ground truth after the robot made its first turn. It effectively models only linear motion and thus fails to incorporate angular changes, resulting in a path that drifts primarily along the x-axis. In contrast, the EKF and PF remained closely aligned with the ground truth throughout the path, with the PF providing slightly better accuracy.

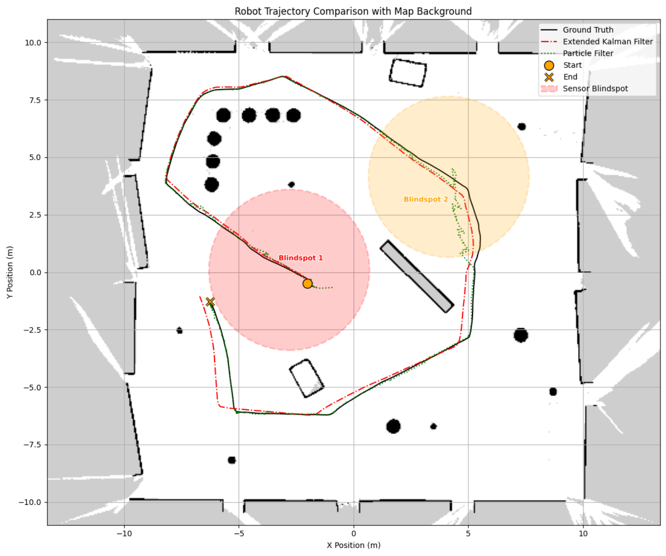

Fig. 2: Trajectory visualization in presence of sensor blind spots. Shows Extended Kalman Filter (EKF), Particle Filter (PF), and Ground Truth.

B. Sensor Blind Spot Evaluation

In the second scenario (Fig. 2), the two blind spots were highlighted in the image, where no lidar measurements were available due to the sensor's 3.5 m range limitation. This significantly affects the performance of the Particle Filter, particularly in "Blindspot 2", where the robot momentarily lost track due to the absence of nearby observable landmarks. This led to a temporary deviation from the true path, which was visually evident. However, once the robot re-entered a region with sufficient detectable features, the PF rapidly recovered its position estimate.

Meanwhile, the EKF demonstrated a tendency to accumulate error over time. This can be attributed to linearization artifacts: as the system propagates through nonlinear dynamics without new measurements (or with uninformative measurements), the EKF's internal model becomes increasingly misaligned with reality, especially when the Jacobians fail to capture higher-order system behavior.

C. Kidnapped Robot Evaluation

The final experiment (Fig. 3) involved forcibly relocating the robot mid-execution in the Gazebo simulation environment commonly referred to as the "kidnapped robot problem". Two distinct relocation events were performed and marked in the plot.

This test clearly demonstrates the strength of the Particle Filter in handling global uncertainty. After each relocation, the PF successfully re-localized the robot as soon as enough unique features were observed. In contrast, the EKF produced significantly skewed estimates. This is expected behavior: EKF's unimodal Gaussian belief cannot represent multiple hypotheses, and it must "drag" its estimate toward the new pose using a correction process based on noisy observations. This results in a curved recovery path, with the EKF never fully converging back to the true position.

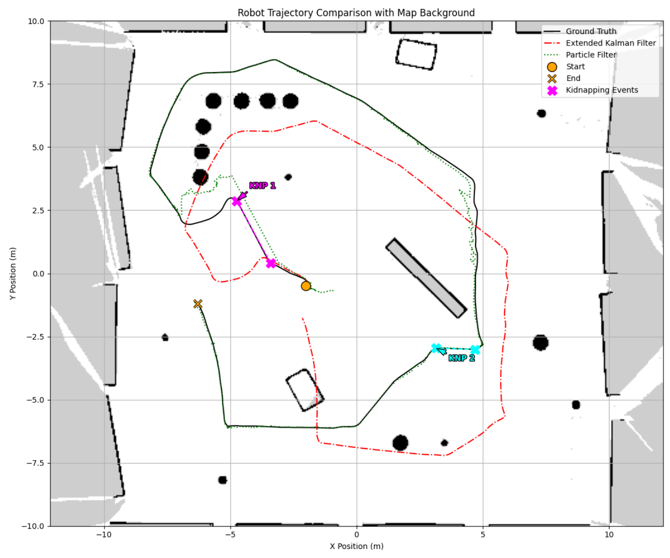

Fig. 3: Trajectory visualization under kidnapped robot scenario. EKF, PF, and Ground Truth are shown.

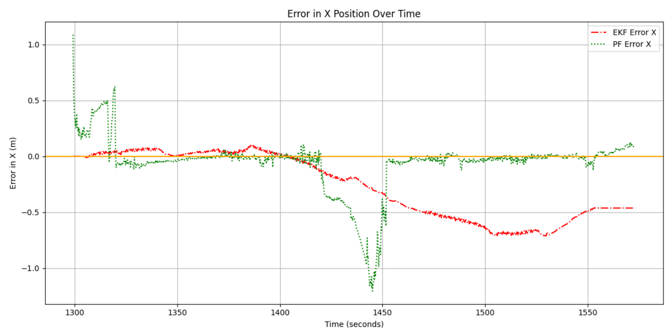

Fig. 4: Error in X position over time for Extended Kalman Filter (EKF) and Particle Filter (PF) relative to ground truth.

The position error plots in Fig. 4 and Fig. 5 provide further insight into the behavior of the EKF and PF during the blind spot scenario. Two major error spikes are clearly visible, corresponding to the periods when the robot passed through regions without lidar coverage due to its limited sensing range.

In the X direction (Fig. 4), the EKF gradually drifts away from the ground truth over time, showing its typical sensitivity to poor or missing observations. In contrast, the PF is briefly disrupted during blind spots but quickly re-converges to the ground truth once sufficient environmental features reappear.

In the Y direction (Fig. 5), the PF shows more pronounced short-term fluctuations during unobserved phases, but like in

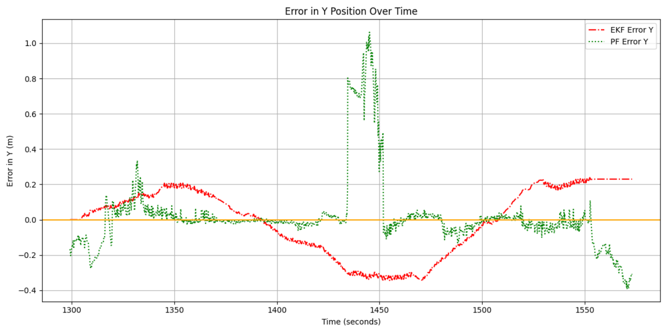

Fig. 5: Error in Y position over time for Extended Kalman Filter (EKF) and Particle Filter (PF) relative to ground truth.

X, it recovers promptly. The EKF, on the other hand, oscillates around the ground truth and exhibits a more sustained positional bias, reflecting the limitations of its unimodal belief structure in ambiguous or partially observed environments.

These plots confirm the qualitative observations made earlier: the EKF is prone to error accumulation in sparse measurement regions, whereas the PF demonstrates strong recovery behavior, particularly after short-term observation losses.

These experiments demonstrate that while the EKF performs reliably under moderate noise and in well-observed regions, the Particle Filter provides significantly more robustness in the presence of uncertainty, measurement dropout, and major localization failures.

VI. Summary and Outlook

This work presented the implementation and evaluation of three probabilistic localization algorithms, Kalman Filter (KF), Extended Kalman Filter (EKF), and Particle Filter (PF) , within a simulated 2D mobile robot framework using ROS 2 and Gazebo. Each filter was integrated into a common infrastructure and tested under identical conditions, including standard waypoint navigation, sensor blind spots, and artificial re-localization scenarios.

The Kalman Filter, while computationally efficient, proved unsuitable for realistic robot motion due to its assumption of linear system dynamics. The Extended Kalman Filter initially delivered better accuracy, but was prone to drift and divergence in nonlinear or poorly observed environments due to limitations introduced by linearization. The Particle Filter consistently demonstrated strong performance across all test scenarios, including its ability to recover from sensor dropout and global pose uncertainty, such as in the kidnapped robot problem.

Future work may focus on improving the computational efficiency of the PF, possibly by exploring adaptive resampling strategies or integrating scan-matching techniques to improve likelihood estimation. Additionally, extending this framework to 3D localization, real-world experiments, or incorporating visual sensors would further increase its relevance for practical autonomous systems.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[2] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[3] E. Kou and A. Haggenmiller, “Extended kalman filter state estimation for autonomous competition robots,” *arXiv preprint arXiv:2310.04459*, 2023.

[4] A. Doucet, N. de Freitas, and N. Gordon, “Sequential monte carlo methods in practice,” in *Springer*. Springer, 2001.

[5] N. Yadaiah, K. Sharma, K. Yadav, A. L. N. Rao, A. Srivastava, and S. K. Shah, “Particle filter-based localization algorithm for autonomous robots in smart factories,” *2024 4th International Conference on Innovative Practices in Technology and Management (ICIPTM)*, pp. 1–6, 2024.

[6] R. Contributors, “map_server: Ros package,” http://wiki.ros.org/map_server, 2024, accessed: 2024-06-29.

[7] S. M. Malagon-Soldara, M. Toledano-Ayala, G. Soto-Zarazua, R. V. Carrillo-Serrano, and E. A. Rivas-Araiza, “Mobile robot localization: A review of probabilistic map-based techniques,” *International Journal of Robotics and Automation (IJRA)*, vol. 4, no. 1, pp. 73–81, 2015.

[8] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley & Sons, 2001.

[9] G. Welch and G. Bishop, “An introduction to the kalman filter,” *SIGGRAPH Course Notes*, vol. 8, p. 41, 1995.

[10] S. J. Julier and J. K. Uhlmann, “A new extension of the kalman filter to nonlinear systems,” in *Proc. of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management II*, 1997.

[11] N. Adurthi, “Scan matching-based particle filter for lidar-only localization,” *Sensors*, vol. 23, no. 8, p. 4010, 2023.

[12] Y. Bar-Shalom and T. E. Fortmann, “Tracking and data association,” 1988, often cited under data fusion in robotics and filtering literature.

[13] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. MIT Press, 2011.

[14] S. Thrun, “Particle filters in robotics,” in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 511–518.

[15] DREAMS-lab, “Gazebo ground truth,” https://github.com/DREAMS-lab/gazebo_ground_truth, 2024, accessed: 2024-04-08.

[16] R. J. Hyndman and A. B. Koehler, “Another look at measures of forecast accuracy,” *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.

[18] ROBOTIS, *ROBOTIS e-Manual for TurtleBot3*, 2024, accessed: 2024-04-08. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/