An Analysis of Probabilistic Localization Filters

Simon Vincent Kulovits
Department of Industrial Engineering
University of Applied Sciences Technikum Wien
Vienna, Austria
ORCID: 0009-0003-0689-1633

***Abstract*—This paper presents a comparative evaluation of three core probabilistic localization methods—Kalman Filter, Extended Kalman Filter, and Particle Filter implemented within a ROS2 based simulation using the TurtleBot3 platform. Each filter is developed from scratch and tested under varying motion patterns and noise levels in Gazebo. While the Kalman Filter performs well under linear conditions, it struggles with non-linear dynamics. The Extended Kalman Filter offers improved accuracy through local linearization, and the Particle Filter demonstrates robustness in complex scenarios but at the cost of higher computational overhead. Results highlight the trade-offs between accuracy, efficiency, and model flexibility, offering insight into the practical deployment of these algorithms in robotic applications.**

I. Introduction

Reliable self-localization is a core requirement for autonomous mobile robots, as it underpins their ability to move purposefully, plan trajectories, and interact with their surroundings. In practice, however, a robot's exact position is rarely known with certainty due to factors such as sensor inaccuracies, environmental changes, and deviations in motion execution. To handle this uncertainty, probabilistic methods have emerged that estimate the robot's pose by maintaining and updating a belief distribution over possible states. This report investigates three foundational approaches to probabilistic localization: the Kalman Filter (KF), the Extended Kalman Filter (EKF), and the Particle Filter (PF). These methods all rely on Bayesian reasoning to refine pose estimates by integrating noisy control commands with sensor feedback. KF and EKF provide efficient solutions under the assumption of Gaussian noise and linear or mildly nonlinear systems, whereas the Particle Filter accommodates highly nonlinear and non-Gaussian processes, making it suitable for more complex settings. The goal of this work is to explore and contrast the behavior of these filters in a unified experimental framework built around the TurtleBot3 robot simulated in Gazebo. Each filter is implemented as a standalone ROS 2 node, and the system is tested across various tasks involving different levels of sensor noise and initialization uncertainty. The main contributions of this paper are:

- A modular implementation of KF, EKF, and PF within a ROS2-based simulation framework
- A systematic comparison of their performance under different noise and initialization scenarios
- An analysis of simulation results and the author's observations implementing the filters

The remainder of this paper is structured as follows: Section II discusses the mathematical foundations of the three filters. Section III outlines the implementation details, software architecture and filter design. Section IV describes the experimental setup. Section V concludes with a summary and comparison of the author's observations.

II. State of the Art

Probabilistic localization plays a prominent role in mobile robotics and has remained a significant area of study for many years. At its core lies the use of Bayes filters, which estimate a robot's state by recursively updating a belief distribution based on incoming control data and sensor feedback. The foundational concepts for these methods are comprehensively explored in the work of Thrun, Burgard, and Fox [1], which outlines various filtering strategies applied across robotic domains.

The Kalman Filter (KF) is among the earliest and most well-established approaches for systems characterized by linear dynamics and Gaussian noise [4]. It offers an analytically tractable solution, providing direct updates for both the mean and covariance of the state estimate. Despite its efficiency, KF is limited in its applicability due to its reliance on strict linearity and unimodal Gaussian assumptions, which are rarely met in practical robotic applications. To overcome these limitations, the Extended Kalman Filter (EKF) was developed as an extension of KF. It handles nonlinear models by locally approximating them using Jacobians computed around the current estimate [3]. EKF has seen widespread adoption in robotics, especially in tasks such as Simultaneous Localization and Mapping (SLAM). Nevertheless, its performance degrades when the linearization is inaccurate or the underlying uncertainty is substantial.

The Particle Filter (PF), also referred to as Monte Carlo Localization (MCL), provides a more flexible alternative by representing the belief state as a collection of weighted samples. Unlike KF and EKF, PF can model complex, multi-modal, and non-Gaussian distributions, making it more suitable for dynamic and noisy environments. This flexibility, however, comes at the cost of increased computational demands, particularly with large particle sets.

Advanced implementations such as Adaptive Monte Carlo Localization (AMCL) [2] enhance PF with dynamic resampling mechanisms to improve performance. Yet for controlled experimental setups or educational purposes, implementing

KF, EKF, and PF manually remains valuable. Such implementations offer direct insight into the algorithmic structure and limitations of each approach.

All three filters; KF, EKF, and PF will be constructed from the ground up within a unified ROS2 simulation framework. By avoiding reliance on pre-packaged solutions like AMCL, a clearer understanding of each method's internal behavior stands to gain. This hands-on strategy enables one to perform a detailed evaluation of the trade-offs among accuracy, resilience to noise, and computational efficiency across different localization scenarios.

A. Kalman Filter

The Kalman Filter (KF) is a recursive state estimator for linear systems with Gaussian noise. It maintains a belief about the system's state as a multivariate normal distribution, defined by a mean vector μ_t and a covariance matrix Σ_t. KF operates in two steps: prediction and correction.

Prediction step:

$$\mu_{t|t-1} = A\mu_{t-1} + Bu_t \tag{1}$$

$$\Sigma_{t|t-1} = A\Sigma_{t-1}A^\top + R \tag{2}$$

Here, u_t is the control input, A and B are the state transition and control matrices, and R is the process noise covariance.

Correction step:

$$K_t = \Sigma_{t|t-1}H^\top(H\Sigma_{t|t-1}H^\top + Q)^{-1} \tag{3}$$

$$\mu_t = \mu_{t|t-1} + K_t(z_t - H\mu_{t|t-1}) \tag{4}$$

$$\Sigma_t = (I - K_tH)\Sigma_{t|t-1} \tag{5}$$

The sensor measurement is z_t while H is the observation model, Q is the measurement noise covariance, and K_t is the Kalman gain. Equation (1) computes the predicted state $\mu_{t|t-1}$ by applying the control input u_t to the previous state μ_{t-1}. The matrix A models how the state evolves over time (e.g., constant velocity), and B maps control inputs to state changes (e.g., wheel velocities). Equ. (2) computes the predicted uncertainty. The previous uncertainty Σ_{t-1} is propagated through the system model using A. The term R adds process noise—uncertainty in the motion model (e.g., wheel slippage).

In the correction step, equations (3)–(5) incorporate the latest sensor measurement z_t to correct the predicted state. The Kalman gain K_t determines how much the filter should trust the measurement versus the prediction. If the measurement noise Q is small, K_t will give more weight to the sensor reading. The new state estimate μ_t is computed by correcting the prediction using the measurement residual $(z_t - H\mu_{t|t-1})$. This residual is the difference between the actual measurement and the expected measurement from the predicted state. The uncertainty is updated to reflect the new information. If K_t is large (i.e., the measurement is reliable), the uncertainty Σ_t is reduced accordingly.

Together, these equations allow the filter to continuously refine the robot's state estimate over time in a principled, probabilistic way. The assumptions of linearity and Gaussian noise make the filter efficient but limit its applicability to more complex systems—which motivates the use of nonlinear extensions like the EKF and PF.

B. Extended Kalman Filter

The Extended Kalman Filter (EKF) extends the Kalman Filter to systems with nonlinear motion and observation models. Instead of assuming linearity, the EKF approximates the models using first-order Taylor expansions around the current estimate. The EKF is widely used in mobile robotics, particularly for scenarios like landmark-based localization and nonlinear motion (e.g., differential drive robots).

$$\vec{x}_t = g(\vec{x}_{t-1}, \vec{u}_t) + w_t \tag{6}$$

$$\vec{z}_t = h(\vec{x}_t) + v_t \tag{7}$$

Here, x_t is the robot state, u_t is the control input, z_t is the observation, $g(\cdot)$ is the nonlinear motion model, and $h(\cdot)$ is the nonlinear observation model. w_t and v_t represent process and measurement noise, respectively. To apply the Kalman update in this nonlinear case, EKF computes Jacobians:

- $G_t = \frac{\partial g}{\partial x}\Big|_{x=\mu_{t-1}, u=u_t}$ — Jacobian of motion model
- $H_t = \frac{\partial h}{\partial x}\Big|_{x=\mu_{t|t-1}}$ — Jacobian of observation model

$$\mu_{t|t-1} = g(\mu_{t-1}, u_t) \tag{8}$$

$$\Sigma_{t|t-1} = G_t\Sigma_{t-1}G_t^\top + R \tag{9}$$

$$K_t = \Sigma_{t|t-1}H_t^\top(H_t\Sigma_{t|t-1}H_t^\top + Q)^{-1} \tag{10}$$

$$\mu_t = \mu_{t|t-1} + K_t(z_t - h(\mu_{t|t-1})) \tag{11}$$

$$\Sigma_t = (I - K_tH_t)\Sigma_{t|t-1} \tag{12}$$

In contrast to the linear Kalman Filter, the EKF uses nonlinear functions $g(\cdot)$ and $h(\cdot)$ to model how the robot moves and observes the world. For instance, in a differential-drive robot, forward motion and turning involve trigonometric relationships, which are captured naturally in g.

Equation (6) uses the nonlinear motion model g to predict the new state mean. The covariance update in Equation (7) propagates uncertainty through the Jacobian G_t, capturing how small deviations in state affect the prediction. In the correction step the Kalman gain Equation (8) is computed using the Jacobian of the observation model H_t. Equation (9) corrects the predicted state using the measurement residual — but this time using the nonlinear function h. Equation (10) updates the uncertainty as in KF. This linearization process allows EKF to approximate Bayesian updates for nonlinear systems. While more powerful than KF, EKF's performance depends heavily on the accuracy of the linearization. In cases where the system is highly nonlinear or uncertainty is large, the EKF may diverge or produce biased estimates.

C. Particle Filter

The Particle Filter (PF), also known as Monte Carlo Localization (MCL), is a non-parametric Bayes filter that approximates the belief distribution using a set of weighted samples

(particles). Unlike KF and EKF, which assume Gaussian distributions, PF can represent arbitrary, multi-modal distributions and is therefore well-suited for non-linear and non-Gaussian systems.

Each particle represents a possible state $x_t^{[i]}$ of the robot, and has an associated weight $w_t^{[i]}$ reflecting how likely that state is given the observations. For initialization N particles $\{x_0^{[i]}\}$ from the prior distribution get sampled. For each particle prediction, a new state based on the motion model gets sampled. The particles are moved according to a probabilistic motion model. This accounts for motion noise (e.g., wheel slippage, encoder drift) by sampling from a distribution instead of applying a deterministic update.

$$x_t^{[i]} \sim p(x_t \mid u_t, x_{t-1}^{[i]}) \tag{13}$$

As a correction step replacement, the weight of each particle based on the measurement likelihood gets updated. Each particle's weight reflects how well it explains the current sensor observation. For example, if the robot's laser scan matches well with the expected scan at the particle's position, it will receive a higher weight.

$$w_t^{[i]} = p(z_t \mid x_t^{[i]}) \tag{14}$$

Then the weights need to be normalized such that

$$\sum_i w_t^{[i]} = 1 \tag{15}$$

As a last step one needs to resample N particles from the current set with replacement, proportional to their weights. This step eliminates particles with low weights and duplicates those with high weights. Over time, this concentrates computational effort on the more likely regions of the state space.

The PF works by maintaining a cloud of particles that collectively represent the belief distribution over the robot's state. Each particle follows the robot's motion model and is evaluated using the current sensor measurement. PF's main advantage is its flexibility: it can handle non-linear motion and observation models and complex, non-Gaussian uncertainties. However, it is computationally expensive, especially as the number of particles grows. The accuracy of the filter is highly dependent on the number of particles and the effectiveness of the motion and measurement models.

III. Methods and Implementation

The three filters are implemented as individual ROS2 nodes in a localization filters package, each responsible for a specific filter implementation of the processing pipeline. Their core functionality is developed in C++ to ensure efficient performance and seamless integration with the ROS2 ecosystem. Each Filter is implemented via a class instance in a filter node class. The implementation is based on ROS2 Jazzy, which provides the necessary tools and communication infrastructure to support modular and scalable robotic applications.

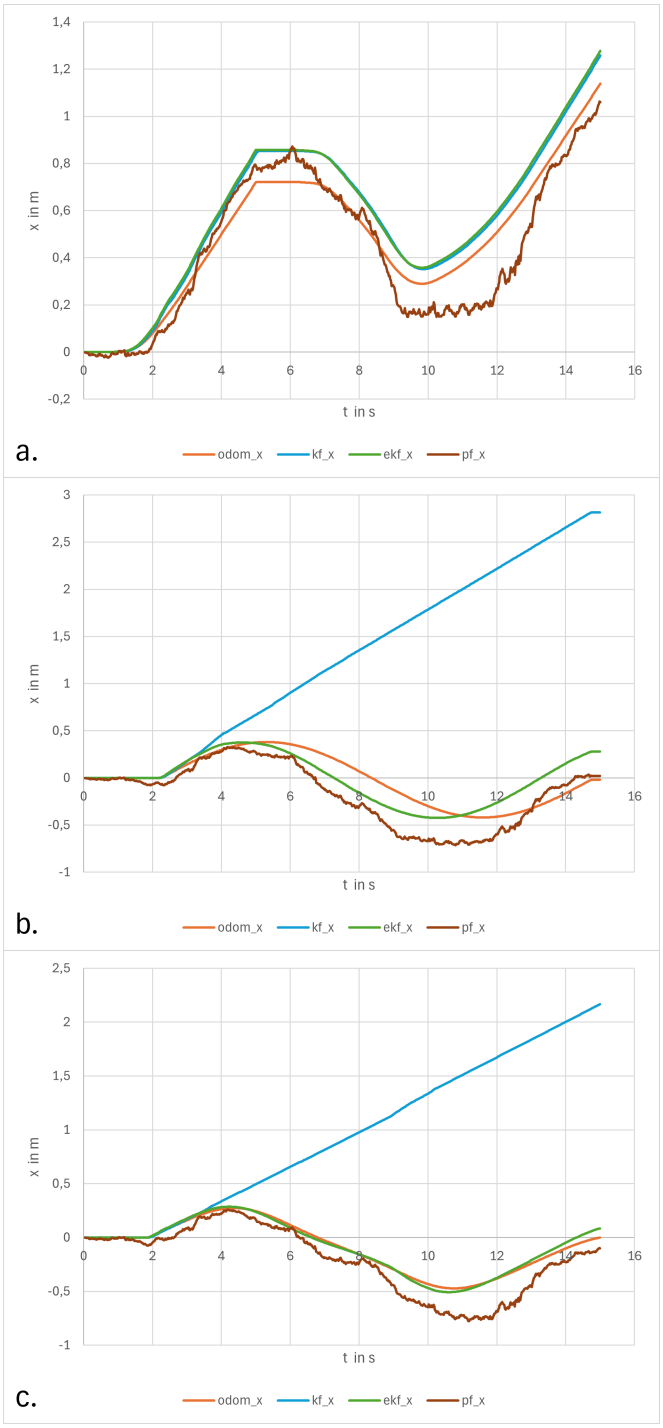

Fig. 1: Plotted Pose estimation with 'odom' as 'Ground Truth' Filtered Pose in x in blue,green and brown
a. shows a straight trajectory in x direction
b. shows the robot move in a circular pattern
c. the robot moves along a curved trajectory

A. Kalman Filter

In this implementation, the Kalman Filter was realized as a linear estimator using the classical two-step approach of prediction and correction. The predict method shown is part of a Kalman Filter implementation designed for robot localization using ROS2 and the 'geometry_msgs::msg::Twist' message as control input. It predicts the next state of the robot based on its current velocity and angular rate. The linear v and angular ω velocities are extracted from the input and organized into a control vector. The method retrieves the robot's current orientation angle θ and updates the state transition matrix A to reflect elapsed time dt, incorporating constant velocity motion. The control matrix B is structured to capture the influence of motion commands, with a basic assumption of near-zero orientation (thus ignoring non-linear trigonometric motion effects). The predicted state x is then computed by applying the motion model $A * x$ and the control input $B * u$. Simultaneously, the state covariance S is propagated forward using the process noise model, reflecting the uncertainty introduced during motion. To ensure consistent angular representation, the predicted orientation is normalized to remain within the range $[-\pi, \pi]$. This method reflects a simplified, linear motion model, appropriate for basic Kalman Filtering within the context of 2D localization as outlined in the lab project.

The correction step refines the predicted state using actual sensor measurements to improve localization accuracy. In this implementation, the Kalman Filter fuses data from wheel encoders via 'JointState' and the IMU to compute an estimated velocity-based measurement. The method begins by extracting angular velocities from the robot's wheel joints, converting them from radians per second to linear velocities using the known wheel radius. These are then used in a differential drive kinematics model to estimate the robot's forward linear velocity v and angular velocity ω, while the lateral velocity y is assumed to be zero due to the mechanical setup of the platform. These measurements are structured into a vector $\mathbf{z} \in \mathbb{R}^3$.

$$\mathbf{z} = [v, 0, \omega]^T \tag{16}$$

which represents the expected observations from the system. The observation matrix $\mathbf{C}$ is defined to map the current state vector $\mathbf{x}$ to the measurement space by selecting the appropriate state components corresponding to forward velocity, lateral velocity, and yaw rate. The measurement noise covariance matrix $\mathbf{Q}$ models uncertainty in the sensor data, with tunable parameters that reflect empirical or expected sensor performance.

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

$$\mathbf{Q} = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.02 \end{bmatrix} \tag{18}$$

The core of the correction step involves computing the innovation:

$$\mathbf{y} = \mathbf{z} - \mathbf{C}\mathbf{x} \tag{19}$$

It quantifies the discrepancy between predicted and measured values. This is followed by calculating the innovation covariance $\mathbf{S}$ and the Kalman Gain $\mathbf{K}$, which determine how much the new measurement should influence the current belief.

$$\mathbf{S} = \mathbf{C}\mathbf{S}\mathbf{C}^T + \mathbf{Q} \tag{20}$$

$$\mathbf{K} = \mathbf{S}\mathbf{C}^T\mathbf{S}^{-1} \tag{21}$$

The updated state estimate is then computed as:

$$\mathbf{x} \leftarrow \mathbf{x} + \mathbf{K}\mathbf{y} \tag{22}$$

and the state covariance is refined by applying the standard covariance update formula:

$$\mathbf{S} \leftarrow (\mathbf{I} - \mathbf{K}\mathbf{C})\mathbf{S} \tag{23}$$

Finally, the method returns a 'PoseWithCovarianceStamped' message populated with the corrected position (x, y) and orientation θ, converted to quaternion form. Only the diagonal elements of the covariance matrix corresponding to x, y, and θ are included in the ROS2 message for simplicity. This correction routine ensures the filter remains grounded in real sensor data, maintaining accuracy over time despite prediction drift. But the Filter cannot account for nonlinear dynamics inherent in real-world robot motion. While the filter performs well during periods of linear motion as shown in Fig. 1a., it lacks the flexibility required to handle curved trajectories or orientation changes like in Fig. 1b.-c., which motivates the use of more advanced techniques like the Extended Kalman Filter or Particle Filter in such scenarios.

B. Extended Kalman Filter

The Extended Kalman Filter (EKF) extends the classical KF framework to handle the nonlinear motion and measurement models typical of a differentially driven mobile robot. In the EKF the prediction step is adapted to handle nonlinear motion models by linearizing the system dynamics around the current state estimate. The robot's forward velocity v and angular velocity ω are extracted from the control input once again, and the robot's current heading θ is used to propagate the state forward in time using a differential drive motion model. The position updates are computed via:

$$g(\vec{u}, \vec{x}) = \begin{bmatrix} x + v\cos(\theta) \cdot dt \\ v\cos(\theta) \\ y + v\sin(\theta) \cdot dt \\ v\sin(\theta) \\ \theta + \omega \cdot dt \\ \omega \end{bmatrix} \tag{24}$$

The velocity components within the state vector are also updated to reflect the effect of the control input:

$$v_x = v\cos(\theta), \quad v_y = v\sin(\theta), \quad \omega = \omega \tag{25}$$

To account for the nonlinearity of the motion model, the state transition Jacobian $\mathbf{G}$ is recomputed at each step. This Jacobian represents the first-order partial derivatives of the motion model with respect to the state and is used to propagate uncertainty. It includes derivatives with respect to the orientation θ and time dt, and captures how small changes in the state affect the prediction:

$$\mathbf{G} = \begin{bmatrix} 1 & dt & 0 & 0 & -v\sin(\theta)\cdot dt & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & v\cos(\theta)\cdot dt & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

The predicted state covariance is then updated using this Jacobian:

$$\mathbf{P} \leftarrow \mathbf{G}\mathbf{P}\mathbf{G}^T + \mathbf{R} \tag{27}$$

Finally, the heading angle θ is normalized to the interval $[-\pi, \pi]$ to ensure continuity in angular representation. This nonlinear prediction framework allows the EKF to more accurately track robot motion over curved paths or during rotations, providing an advantage over the standard Kalman Filter in real-world robotic navigation scenarios.

In the correction step of the EKF, nonlinear sensor measurements are incorporated to refine the predicted state estimate. Wheel encoder velocities are used to compute the actual robot motion. The left and right wheel speeds are extracted from the 'JointState' message and converted from angular to linear velocities using the known wheel radius. These are then used to estimate the robot's linear velocity v and angular velocity ω through differential drive kinematics:

$$v = \frac{v_{\text{left}} + v_{\text{right}}}{2}, \quad \omega = \frac{v_{\text{right}} - v_{\text{left}}}{\text{wheelbase}} \tag{28}$$

The measurement model $h(\vec{x})$ is also non-linear, mapping the current state to expected sensor observations. In this case, the measurement vector includes the robot's velocity components in x and y directions, as well as its angular velocity, calculated as:

$$h(\vec{x}) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \tag{29}$$

The nonlinear measurement model is linearized around the current state estimate using the Jacobian matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{30}$$

This matrix maps the velocity and angular rate components from the full state vector $\mathbf{x}$ to the measurement space. The innovation (measurement residual) is computed as the difference between the actual measurement and the predicted measurement:

$$\mathbf{y} = \mathbf{z} - \mathbf{H}\mathbf{x} \tag{31}$$

The innovation covariance is then computed as:

$$\mathbf{S} = \mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{Q} \tag{32}$$

From this, the Kalman Gain is derived:

$$\mathbf{K} = \mathbf{P}\mathbf{H}^T\mathbf{S}^{-1} \tag{33}$$

Using the Kalman Gain, the state is updated as:

$$\mathbf{x} \leftarrow \mathbf{x} + \mathbf{K}\mathbf{y} \tag{34}$$

and the covariance matrix is updated accordingly:

$$\mathbf{P} \leftarrow (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P} \tag{35}$$

Finally, the robot's orientation θ is normalized to the interval $[-\pi, \pi]$. The updated state is returned as a 'PoseWithCovarianceStamped' message, containing the corrected (x, y) position and orientation as a quaternion. For simplicity, only the variances of the x-position, y-position, and orientation are included in the message's covariance matrix. This correction step enables the EKF to incorporate nonlinear motion information from wheel encoders, significantly improving estimation robustness and accuracy, especially during non-straight trajectories.

C. Particle Filter

The Particle Filter represents the robot's state using a set of discrete hypotheses, or particles, each with an associated weight reflecting its likelihood. At initialization, N particles are uniformly distributed at the origin $(x = 0, y = 0, \theta = 0)$, each assigned an equal weight of $\frac{1}{N}$. This ensures that no initial bias is introduced into the system:

$$\forall i \in \{1, \ldots, N\}: \quad \begin{aligned} x_i &= 0 \\ y_i &= 0 \\ \theta_i &= 0 \\ w_i &= \frac{1}{N} \end{aligned} \tag{36}$$

Each particle is also associated with a Gaussian noise model to account for uncertainty in motion, defined independently for the x and y directions as well as the orientation θ. During the prediction step, the motion model is applied to each particle individually using the control input (v, ω) for linear and angular velocities, along with a time step dt. The updated position and orientation of each particle are computed as:

$$\begin{aligned} x_i &\leftarrow x_i + v\cos(\theta_i)\cdot dt + \mathcal{N}(0, \sigma_x^2) \\ y_i &\leftarrow y_i + v\sin(\theta_i)\cdot dt + \mathcal{N}(0, \sigma_y^2) \\ \theta_i &\leftarrow \theta_i + \omega\cdot dt + \mathcal{N}(0, \sigma_\theta^2) \end{aligned} \tag{37}$$

where $\mathcal{N}(0, \sigma^2)$ denotes Gaussian noise sampled with zero mean and variance σ^2. This process enables the particle filter to explore multiple possible future states and is particularly well-suited for handling non-linear and non-Gaussian motion dynamics. By evolving each particle independently, the prediction step preserves the multi-modal nature of the belief distribution, which is one of the key strengths of the Particle Filter over parametric filters such as the Kalman Filter.

The correction step in the Particle Filter updates the weights of each particle to reflect how well its predicted motion aligns with observed sensor measurements. In this implementation, the angular velocity predicted by the motion model is compared to the actual measurement obtained from the IMU. For each particle, a Gaussian likelihood function is used to assign a weight:

$$w_i \leftarrow \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\hat{\omega}_i)^2}{2\sigma^2}\right) \tag{38}$$

where z is the measured angular velocity, $\hat{\omega}_i$ is the particle's predicted angular velocity, and σ is the assumed standard deviation of the IMU noise. The weights are then normalized:

$$\sum_{i=1}^{N} w_i = 1 \tag{39}$$

If all weights collapse (i.e., sum to zero), the filter resets to a uniform distribution to avoid degeneracy. Following correction, the resampling step draws a new set of particles based on the normalized weights. This step is implemented using a discrete distribution sampling process that probabilistically favors particles with higher weights, effectively duplicating likely hypotheses and discarding unlikely ones. Resampling addresses particle depletion and ensures the filter continues to track plausible state estimates. Finally, the estimated pose of the robot is computed as the mean of all particles:

$$\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \hat{y} = \frac{1}{N}\sum_{i=1}^{N} y_i, \quad \hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} \theta_i \tag{40}$$

This average is used to populate a 'PoseWithCovarianceStamped' message, where the orientation is converted to quaternion form for ROS2 compatibility. The result should provide a robust pose estimate even under high uncertainty, leveraging the non-parametric nature of the Particle Filter.

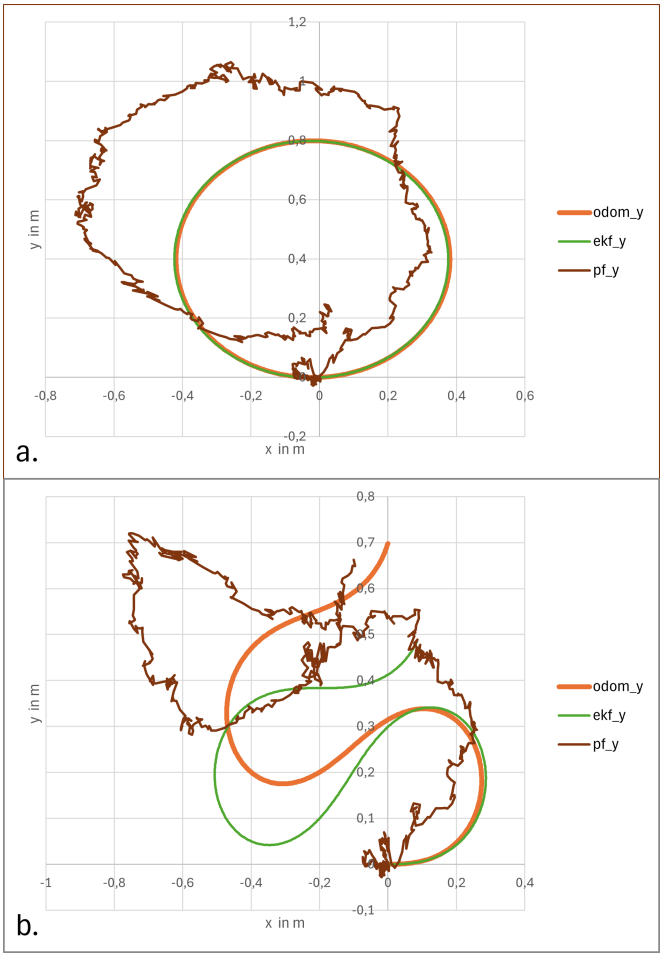

Fig. 2: Capured robot trajectory in x and y starting at $(0, 0)$
a. shows the robot move in a circular pattern with $u = (0.2, 0.5)$
b. robot moves along non linear path $u = (0.15, sin(0.05t))$

IV. Results and Discussion

The time based tracking of the estimated pose forming the robots trajectory for the KF can be seen in Fig. 1a.-c. The two-dimensional tracking of the trajectory for EKF and PF are plotted in Fig. 2a.-b. The EKF tracks the ground truth from the odometry of the robot closely, almost overlapping the graphs. The PF has bigger discrepancies as can be seen in Fig. 3a.-c. as the RMSE of the three filters as plotted over the simulation time. Fig. 3a. shows the straight trajectories when the robot is only moving in x direction, having zero-velocities entries for ω. The KF's problem with drift is directly visible from tracking its RMSE on Fig. 3b.-c. proving the KF weakness with non-linear movements. The Particle Filter (PF) is inherently more computationally demanding than the Kalman Filter (KF) and Extended Kalman Filter (EKF), particularly as the number of particles increases or when the update rate is high. Each particle requires individual prediction, weighting, and resampling, leading to significant processing overhead. In a simulation environment, this can result in reduced loop rates when initialized with many particles, delayed message handling, or visible lag in the system's response. While the PF offers greater flexibility and robustness in complex, non-linear scenarios, its performance trade-off becomes especially apparent in resource-constrained or real-time applications. So generally, a PF can scale with map size as more particles are required to keep the same localization probability density.

The jagged and misaligned appearance of the PF trajectory, as seen in Fig. 2b., can be attributed to several inherent characteristics of the algorithm and its sensitivity to configuration parameters. One major factor is the use of a relatively small number of particles, which can lead to fluctuations in the estimated pose after forming the average, particularly in areas with high uncertainty. Since the pose estimate is typically derived as the weighted mean of all particles, a sparse or spread-out particle set can cause visible jitter in the trajectory. Furthermore, aggressive resampling may have lead to sample impoverishment, reducing particle diversity and causing the estimated pose to jump between multiple clusters. Timing issues in the simulation, such as delayed

or unsynchronized sensor messages, can also degrade filter stability. Since the filters only use 'Estimated TimerSync', as the use of 'Time Synchronizer' wouldn't allow runtime execution as the time differences between the sensor messages was too big. Additionally, averaging over particles without considering their spatial distribution can amplify artifacts when the belief is multimodal or skewed.

To mitigate these effects, several improvements could be applied in the furute: increasing the particle count, fine-tuning the noise parameters, using adaptive resampling strategies, or estimating the pose based on the most likely particle rather than the mean. These adjustments can significantly reduce the jaggedness and improve the stability of PF-based localization in dynamic or uncertain environments.

V. Summary and Outlook

The Kalman Filter demonstrated fast execution and reliable performance for linear trajectories, but exhibited increasing estimation errors in curved motion due to its inability to model non-linear dynamics. The Extended Kalman Filter, through its use of local linearization and Jacobian matrices, offered improved performance in curved and rotational motion, closely tracking the ground truth in most scenarios. The Particle Filter, while in theory the most flexible and capable of modeling multi-modal distributions, was the most sensitive to configuration parameters and exhibited higher computational demands. Notably, its trajectory displayed jitter and reduced accuracy under low particle counts and aggressive resampling. These results underscore the importance of selecting a state estimation algorithm based on task-specific requirements. In environments where real-time performance and low complexity are critical, the KF or EKF may be sufficient. For more complex, noisy, or ambiguous environments, PF provides a powerful alternative—albeit at greater computational cost.

Several improvements and extensions could further enhance the filter implementations. Incorporating adaptive resampling in the PF could reduce sample impoverishment, while increasing the number of particles would likely improve trajectory smoothness and estimation fidelity. Integrating more sophisticated sensor fusion, such as visual odometry or LIDAR, could also enhance robustness.

Overall, this study highlights the trade-offs between accuracy, efficiency, and complexity in probabilistic localization and provides a practical foundation for extending these techniques in more advanced robotics applications.

References

[1] J. Bongard, "Probabilistic robotics. sebastian thrun, wolfram burgard, and dieter fox. (2005, mit press.) 647 pages," *Artificial Life*, vol. 14, no. 2, pp. 227–229, 2008.

[2] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI '99)*, July 1999, pp. 343–349.

[3] A. Gelb, J. F. Kasper, R. A. Nash, C. F. Price, and A. A. Sutherland, Eds., *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.

[4] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

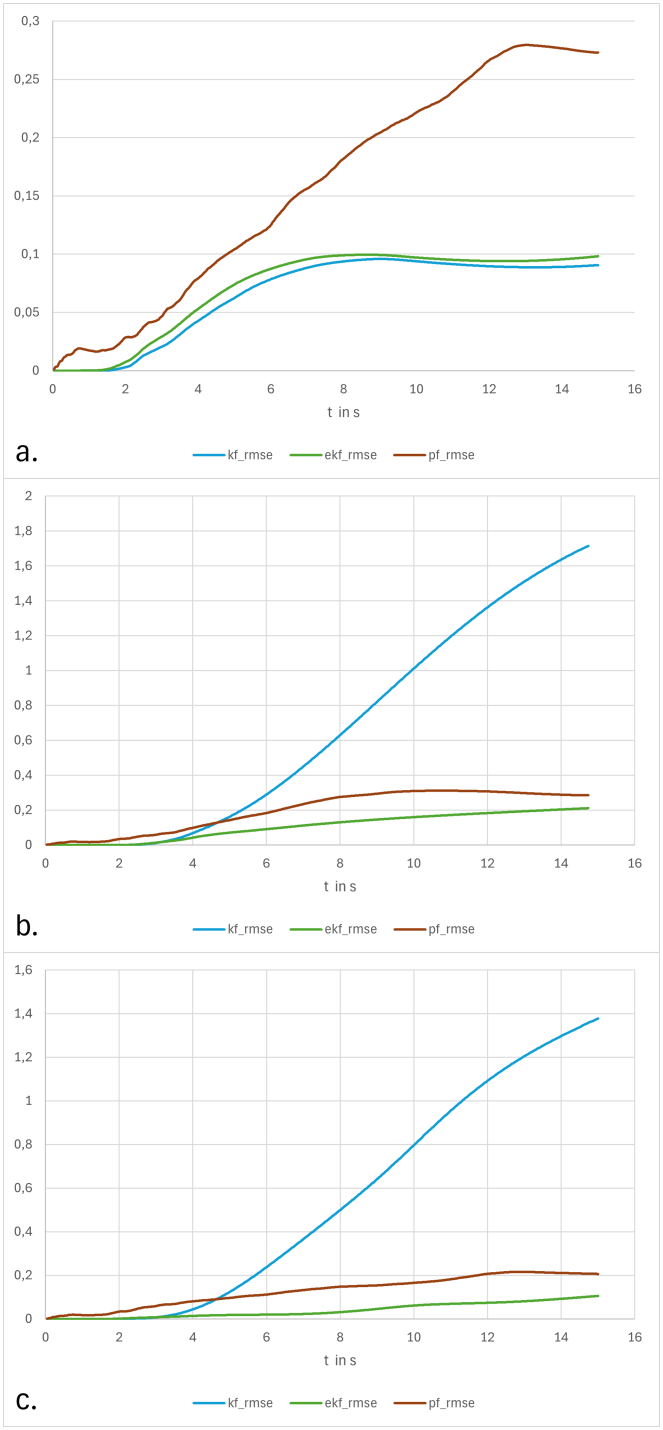

Fig. 3: Root Mean Square Error (RMSE) KF, EKF, and PF over time for three different motion trajectories.
a. Linear motion along the x-axis with zero angular velocity, highlighting KF's low error under linear conditions.
b. Circular motion, where EKF outperforms KF due to its ability to handle nonlinear dynamics.
c. Non linear path with changing rotational rate where PF performs more robustly despite higher variance, while KF exhibits significant drift.