Comparing KF/EKF/PF for Mobile Robot Localization

Julian Haring

***Abstract*—In mobile robotics one of the key challenges is localization. [10] For any task it is fundamental to know the current position in the room. To achieve this, numerous algorithms have been developed over the decades of research. While more recent approaches to the problem surpass previous standards of accuracy and reliability, there isn't a jack of all trades. Therefore the goal of this paper is the implementation and comparison of three of the most common localization algorithms.**

I. Project

The goal of this project is to compare three of the most commonly used probabilistic localization algorithms in mobile robotics: the Kalman Filter (KF), Extended Kalman Filter (EKF), and Particle Filter (PF). While each of these algorithms has been extensively studied individually, there is a need for a direct comparison to understand their practical trade-offs in real-world applications.

The challenge in mobile robot localization is that there isn't a one-size-fits-all solution. Different algorithms excel in different scenarios, and the choice often depends on factors like computational resources, required accuracy, and the specific characteristics of the environment. This makes it difficult for practitioners to decide which approach to use for their particular application.

To address this, we implement all three algorithms in the same simulation environment using ROS2 and compare their performance on identical tasks. The robot follows a predefined path through a standard map, allowing us to evaluate how well each algorithm tracks the ground truth trajectory. We pay particular attention to challenging scenarios like sharp turns and rapid movements, where localization algorithms typically struggle.

The comparison focuses on two main aspects: accuracy and computational efficiency. Accuracy is measured by comparing the estimated path to the ground truth, while computational efficiency is evaluated by monitoring CPU usage during operation. This dual assessment provides insight into the practical trade-offs between precision and processing requirements.

By running all algorithms under the same conditions, we can observe their behavior directly and identify situations where one approach might be preferable over others. The results provide practical guidance for selecting the appropriate localization algorithm based on specific application requirements and constraints.

II. Probabilistic Robotics

The localization of mobile robots may seem straightforward at first. If the starting position is known the acquisition of the new position at t+1 is nothing more than a linear geometrical problem that can be solved by most with the use of trigonometric functions. While this works fine in simulations under optimized conditions, the challenge is in the implementation in realistic scenarios.

To mitigate the outside impacts on our motion model, additional sensors are used to compare the positions to. While a pure virtual sensor offers perfect data of its measured medium, in realistic implementations they are influenced by their surroundings resulting in deviations of the expected results called noise. Noise can occur in the form of different reflective surfaces when working with a laser scan, slippage of wheels for wheel encoders or uneven surfaces. The usage of electrical components automatically leads to process noise in every part of the system.

Therefore a localization based on the assumption of an idealized environment leads to large deviation to the actual pose due to the accumulative error. The noise of the system requires the assumption that a position is not a fixed point, but more a statistical distribution over a mean value. This probability based approach to robot localization is called probabilistic robotics. [10]

Most of the algorithms used in probabilistic robotics are based on Bayes' Rule. [2] When combining two different probabilities such as our motion model and our sensor result, Bayes allows the estimation of our position by comparing our prediction of our sensor, given the predicted position $P(A \mid B)$ with the actual results it collects.

$$P(H \mid E) = \frac{P(E \mid H)\, P(H)}{P(E)} \quad [2]$$

A. Kalman Filter

One of the oldest implementations of Bayes' Rule in mobile robot localization is the Kalman Filter (KF). It assumes our noise to be Gaussian distributed. [4] It compares the position assumed by the motion model with the position determined by the sensor based on Bayes' Rule and computes the so-called "Kalman-Gain" (K). This Kalman Gain describes the influence of each state of the prediction to each state of the measurement. The Kalman Gain matrix is tweaked each rendition of the algorithm providing the optimal tuning of the motion model to compensate for noise.

The Kalman Filter operates in two main steps: the **Prediction** and the **Update**. Each step is governed by linear equations that refine the robot's estimate of its position while accounting for uncertainty.

Algorithm 1 Kalman Filter [10]

```
1: Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6: $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7: return $\mu_t, \Sigma_t$
```

The algorithm parameters are defined as follows:

- $\hat{x}_t$: Predicted state estimate using previous state x_{t-1}.
- A: State transition matrix describing system dynamics.
- $B\,u_t$: Control input affecting the system.
- P_t: Predicted covariance representing uncertainty in $\hat{x}_t$.
- Q: Process noise covariance capturing model inaccuracy.
- K_t: Kalman Gain
- z_t: Actual sensor measurement at time t.
- H: Observation matrix mapping state to measurement space.
- R: Measurement noise covariance indicating sensor uncertainty.
- x_t: Updated (posterior) state estimate.
- P_t: Updated covariance, representing refined uncertainty.
- I: Identity matrix ensuring correct dimensionality in update.

The algorithm steps are:

1) The function is called with the required parameters
2) The prediction of the position using the previous position and applying the control inputs to the motion model
3) The predicted covariance is calculated by taking the previous covariance and adding a predefined assumption of our noise
4) The Kalman-Gain equation dynamically determines how much the filter should trust the sensor measurement versus the predicted state, based on their respective uncertainties
5) The corrected position is calculated by applying the Kalman Gain to the prediction and the measured position
6) The new covariance is calculated in the same fashion
7) The corrected state and covariance is returned

B. Extended Kalman Filter

The Extended Kalman Filter (EKF) was introduced to overcome the limitation of the standard Kalman Filter in regards to its restrictions to linear distributions. In real-world robotics applications, most systems are represented by nonlinear dynamics, greatly reducing the effectiveness of the standard KF. [3]

The EKF tries to solve this problem by linearizing the nonlinear functions around the current estimate using a first-order Taylor expansion. It computes the Jacobian matrices of the motion model and the observation model. These Jacobians serve as linear approximations of the system's dynamics and sensor models.

The general workflow of the EKF is similar to the standard Kalman Filter, consisting of a prediction and an update step. The big difference is the substitution of A and B by the Taylor approximated G and H replacing the sensor motion model C.

Algorithm 2 Extended Kalman Filter [10]

```
1: function EKF($u_t$, $z_t$, $\mu_{t-1}$, $\Sigma_{t-1}$)
2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$
6: $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$
7: return $\mu_t, \Sigma_t$
```

Where:

- g: nonlinear motion model
- G: Jacobian of g in regards to the state space of the robot
- h: nonlinear measurement model
- H: Jacobian of the measurement model

C. Particle Filter / MCL

The Monte Carlo Localization Algorithm (MCL), an implementation of the Particle Filter (PF), is a powerful nonlinear and non-Gaussian alternative to the Kalman Filter family. Rather than requiring Gaussian distributions and linear models, or linearizing them, the PF approximates the posterior distribution using a set of weighted samples (particles). This allows the filter to follow multiple guesses at the same time, allowing room for correction of previous mistakes. [1]

The filter operates by maintaining a population of hypotheses about the system state and updating them over time using probabilistic principles.

Algorithm 3 Monte Carlo Localization (MCL) Algorithm [10]

```
1: Algorithm MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ $m = 1$ to $M$
3: $x_t^{[m]} =$ sample_motion_model($u_t, x_{t-1}^{[m]}$)
4: $w_t^{[m]} =$ measurement_model($z_t, x_t^{[m]}, m$)
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ $m = 1$ to $M$
6: draw $i$ with probability $\propto w_t^{[i]}$
7: add $x_t^{[i]}$ to $\mathcal{X}_t$
8: return $\mathcal{X}_t$
```

The algorithm uses the following notation:

- $\mathcal{X}_{t-1}$: Set of particles (state hypotheses) from the previous time step $t-1$
- m: Index for iterating through individual particles
- M: Total number of particles in the filter
- $\bar{\mathcal{X}}_t$: Temporary set of predicted particles at time t
- $x_t^{[m]}$: The m-th particle's state estimate at time t
- $w_t^{[m]}$: The importance weight associated with the m-th particle

III. Setup

The project is realized on basis of the Robot Operating System 2 (ROS2) jazzy. [7] Due to its extensive documentation as well as wide selections of sensors a Turtlebot 3 is used as a mobile robot. [9] The physics are simulated with gazebo iron [5] and the sensors are accessed using RVIZ2. [8]

The robot state is described as a 6x1 vector. For the KF as well as EKF the `cmd_vel` topic is used for applying the motion model while the correction is made using `odom` for linear speeds as well as `imu` for rotational speeds as well as accelerations.

$$\mu = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \omega \end{bmatrix}$$

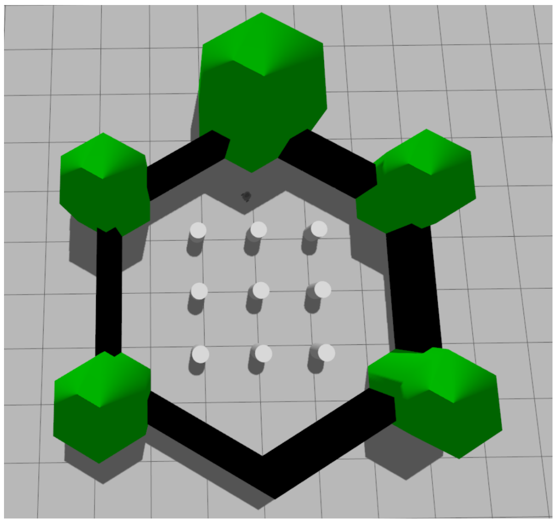
Fig. 1: Default map used in the simulation environment

The navigation is done using the nav2 package [6] and controlled using a separate `nav_node` node sending the robot along 4 predefined goal poses on the gazebo standard map. The three localization algorithms are calculated in three separate nodes publishing their position and covariance as well as previous path to RVIZ. All four nodes can be launched using start.launch.py and the noise matrices can be modified in the settings.yaml file to be accessed by both KF and EKF at the same time.

IV. Implementation

A. Kalman Filter

The Kalman Filter is implemented in C++ using the `Eigen` library for matrix operations. A ROS2 node subscribes to `/cmd_vel`, `/imu` and `/odom` topics. The node maintains the current state vector and covariance matrix. At a fixed frequency the nodes are updated and the prediction step is executed using the velocity commands. The Kalman Gain matrix is dynamically calculated at each cycle, adjusting the trust between the motion and sensor models. The final pose estimate is published on a custom topic `/kf_pose`.

B. Extended Kalman Filter

The EKF implementation extends the KF by incorporating nonlinear models. Here, the state transition and measurement functions are explicitly defined to handle the robot's circular motion. The Jacobians of these functions are recalculated at each timestep. The EKF is implemented in a similar node as KF but additionally computes the Jacobian matrices on-the-fly. This allows more realistic motion handling compared to the linear assumptions of the standard KF. The output of the EKF is published on the `/ekf_pose` topic.

C. Particle Filter

The Particle Filter (MCL) is implemented with the help of the `random` module for sampling new particles and calculating weights based on laser scan similarity. Each particle represents a hypothesis of the robot's position. Initially, particles are distributed uniformly in the environment. For each motion command, the particles are updated by applying a sampled motion with added noise to simulate uncertainty. The weights are computed by comparing the expected laser scan at the particle's position with the actual scan data, using a likelihood field. A resampling wheel algorithm is used to preferentially select particles with higher weights. The particles are published on `/pf_particles` for visualization in RVIZ2.

V. Performance Evaluation

When evaluating the performance of the algorithms the main method used is the comparison of paths. This was achieved by publishing the pose at every time step to a path topic. When comparing the ground truth received from the tf robot state to the path of the algorithm one can see the performance characteristics of each method.

A. Kalman Filter

Analyzing the path of the KF one can notice the high accuracy the KF achieves compared to the ground truth. Only in the end when performing a 270-degree turn the algorithm loses precision and starts to drift. Relying on a linear motion model with `/cmd_vel` inputs and IMU measurements, it struggles when the robot's actual dynamics deviate from pure linear motion. Cumulative errors in prediction and correction steps manifest as a growing offset, especially during sharp turns or rapid accelerations.

B. Extended Kalman Filter

The Extended Kalman Filter achieves the goal of tracking the robot's position as well. Errors occur in similar moments as with the KF mainly in turns above 180 degrees. This is usual in differentially driven robots due to inaccuracies in the wheel encoders. The results are sufficient, but errors spike whenever the linearization point poorly reflects the true pose.

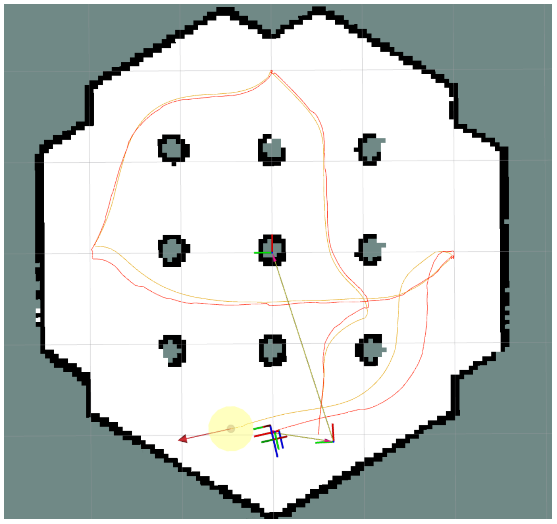

Fig. 2: Path tracking performance of the Kalman Filter

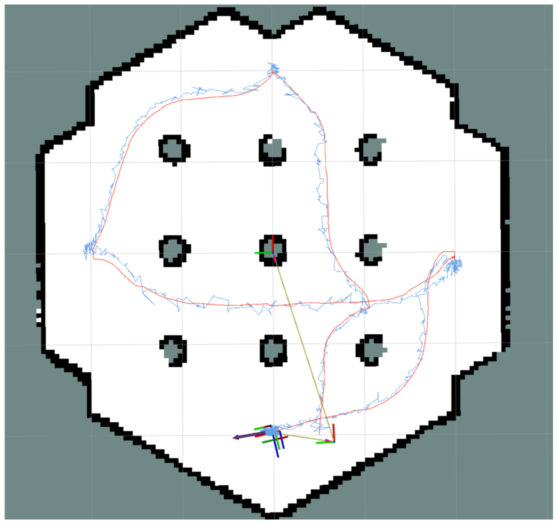

Fig. 4: Path tracking performance of the Particle Filter

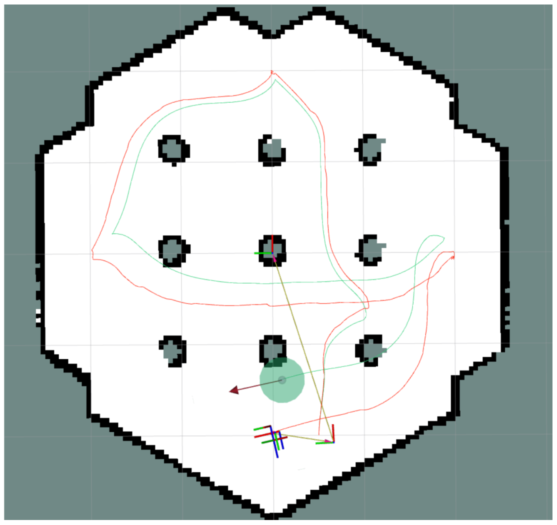

Fig. 3: Path tracking performance of the Extended Kalman Filter

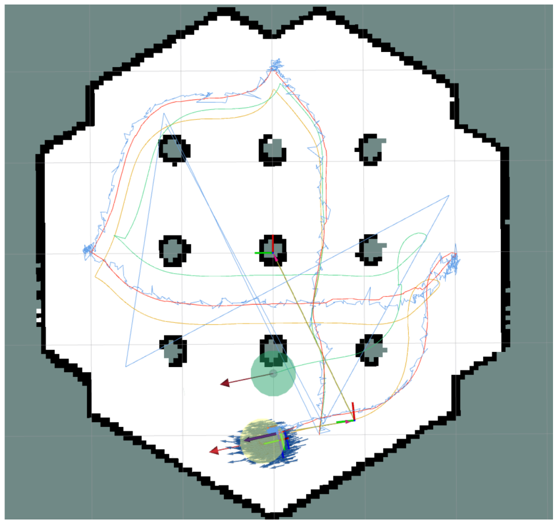

Fig. 5: Comparison of all three algorithms: KF (yellow), EKF (green), PF (blue) and ground truth (red)

C. Particle Filter

The Particle Filter achieves noticeably the best results of the three tracking the ground truth at a very high accuracy. This is partly due to the laser scan allowing the model to correct itself after starting to drift. The high-frequency variations of the path are because of the particle filter switching particles at each cycle not focusing on one pose. In rare occurrences the particle filter fails to detect the correct side of the map and converges at a wrong position before being able to correct itself. While particle filters have the ability to rectify wrong assumptions it fails at this, if there are no poses left at the right area.

D. Comparison

When comparing all three nodes next to the ground truth it is clear that the PF achieves by far the best results in accuracy. This is analogous to the results in literature. KF and EKF struggle with the same kind of drift in areas of large rotational changes and linear accelerations.

E. Computational Power

The computational requirement is one of the most important properties of a localization algorithm. Especially when trying to implement the code on real robots one often faces limitations in the computational power of the onboard CPU. When comparing the required resources for all three models the significantly higher usage of computational power for the particle filter can be noticed. This is mainly due to the algorithm requiring calculating a pseudo measurement of each ray of each particle. Using 300 particles and 36 rays at 10 Hz the processor needs to calculate 108,000 rays per second resulting in a CPU usage of 73.8% on a Ryzen 5 5600.

Tab. 1: Performance metrics of filtering algorithms

Metric	**KF**	**EKF**	**PF1**	**PF2**	**PF3**	**PF4**
CPU Usage (%)	10.3	7.0	17.9	33.9	41.4	73.8

Where PF1 = 100 particles/18 rays, PF2 = 100 particles/36 rays, PF3 = 300 particles/18 rays, PF4 = 300 particles/36 rays.

VI. Discussion and Outlook

The experiments demonstrate that all three probabilistic localization techniques—Kalman Filter (KF), Extended Kalman Filter (EKF), and Particle Filter (PF)—are capable of robustly estimating the pose of a mobile robot in simulated environments. However, each method exhibits distinct strengths and limitations tied to their underlying assumptions and computational complexities.

The KF, relying on linear and Gaussian assumptions, performs well in scenarios where the system dynamics and sensor models closely align with these properties. Its simplicity makes it attractive for real-time applications with limited processing resources. However, it becomes inadequate when the motion or observation models exhibit pronounced nonlinearity, leading to systematic errors in localization.

The EKF extends the applicability of KF by linearizing around the current estimate through Jacobian matrices, which significantly improves performance in moderately nonlinear systems. Nonetheless, the EKF remains sensitive to poor linear approximations, particularly when large initial uncertainties exist or highly nonlinear behaviors dominate. Divergence can occur if the linearization poorly captures the underlying system, which emphasizes the need for carefully designed models and tuning.

In contrast, the PF (MCL) shines in complex environments with nonlinear, non-Gaussian characteristics. By maintaining a diverse set of particles, it can represent multimodal distributions and recover from ambiguous or incorrect state estimates. This makes it highly suitable for global localization and kidnapped robot problems. The tradeoff is a substantially higher computational load, as a large number of particles are typically required to achieve accurate and stable estimates. This might limit its use on resource-constrained platforms without careful optimization.

A. Future Work

Future work could explore hybrid approaches that combine the complementary strengths of these algorithms. For example, leveraging a PF to globally localize the robot while switching to an EKF for efficient local tracking may provide a balanced tradeoff between robustness and computational efficiency.

Additionally, integration with more diverse sensor modalities could enhance the resilience of these filters against specific types of noise and uncertainty. Machine learning techniques might also be employed to adaptively model noise characteristics or even replace traditional motion models with learned dynamics.

Finally, extending this study to real-world experiments on physical robots would be a natural progression, validating the simulated results under realistic sensor noise, wheel slippage, and unforeseen disturbances, thereby closing the gap between theory and practical deployment.

References

[1] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999.

[2] E. T. Jaynes, *Probability Theory: The Logic of Science*. Cambridge, UK: Cambridge University Press, 2003.

[3] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–193.

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

[5] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004, pp. 2149–2154.

[6] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, 2020, pp. 2718–2725.

[7] Open Robotics, "ROS 2 documentation," https://docs.ros.org, 2024, accessed: 2024-07-06.

[8] ——, "RViz2 — ROS 2 visualization tool," https://docs.ros.org/en/rolling/Tutorials/Intermediate/RViz2.html, 2024, accessed: 2024-07-06.

[9] Robotis, "Turtlebot 3 e-manual," https://emanual.robotis.com/docs/en/platform/turtlebot3 2024, accessed: 2024-07-06.

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.