EECS 465 Introduction to Algorithmic Roboics Final Project Report: Kalman and Particle Filters Localization

Junhao TU, Wensong HU

***Abstract*— Comparing Kalman and Particle Filters in robotic localization, this study evaluates their performance across open spaces, some obstacles, and complex mazes, revealing their strengths and limitations.**

I. Introduction

In robotics, precise localization is a cornerstone for enabling autonomous systems to navigate and interact effectively with their environment. This challenge becomes even more critical in scenarios where GPS signals are unreliable or unavailable, such as indoor settings or densely built-up areas. The importance of robust and accurate localization is paramount in a variety of applications, ranging from autonomous vehicles navigating city streets to robots performing tasks in hazardous or inaccessible areas. These applications not only demand high accuracy but also the ability to function reliably under diverse and often unpredictable conditions.

This project is devoted to exploring and comparing two of the most prominent localization algorithms: the Kalman Filter and the Particle Filter. These methods will be scrutinized under three distinct scenarios to understand their practical applicability and limitations. The first scenario is an obstacle-free open space, which represents a controlled environment with minimal external influences on the localization process. This setting serves as a baseline to evaluate the fundamental performance of each algorithm. The second scenario introduces some obstacles, simulating more realistic conditions such as a typical indoor environment with furniture or other small impediments. This tests the algorithms' ability to handle slight complexities in the environment. The final and most challenging scenario is a complex maze, which represents a highly dynamic and unpredictable environment. This will push the algorithms to their limits, revealing their robustness and adaptability under extreme conditions.

Understanding the strengths and weaknesses of these algorithms is crucial. The Kalman Filter, known for its efficiency and accuracy in linear systems with Gaussian noise, may struggle in highly non-linear and complex environments. On the other hand, the Particle Filter, celebrated for its flexibility and ability to handle non-linear dynamics and non-Gaussian noises, might suffer from computational intensity and inefficiency in simpler scenarios. By analyzing these algorithms in varied contexts, this project aims to provide insights into their operational boundaries and optimal application environments, paving the way for more informed choices in the field of robotic localization.

II. Implementations

Generally, the implementation of this project could be divided into four sections, which are the motion model, sensor model, Kalman filter, and Particle filter:

A. Motion Model

First, the PR2 robot, equipped with a four-wheel omnidirectional drive, is modeled to understand its dynamics. The robot's state is defined by a three-dimensional vector $x = [x, y, \theta]^\top$, representing its position (x, y) and orientation (θ) in a two-dimensional space. In addition, control inputs, derived from sensor data, are expressed as changes in the robot's position and orientation, denoted as $u = [dx, dy, d\theta]^\top$. Commonly, the mobile robots are modeled as nonlinear systems that have the general equation:

$$x_{t+1} = g(u_t, x_{t-1}) \tag{1}$$

However, this study simplified the robot's nonlinear behavior while maintaining its core properties. Therefore, the robot's motion is described by a linear motion equation:

$$x_{t+1} = Ax_t + Bu_t + \varepsilon \tag{2}$$

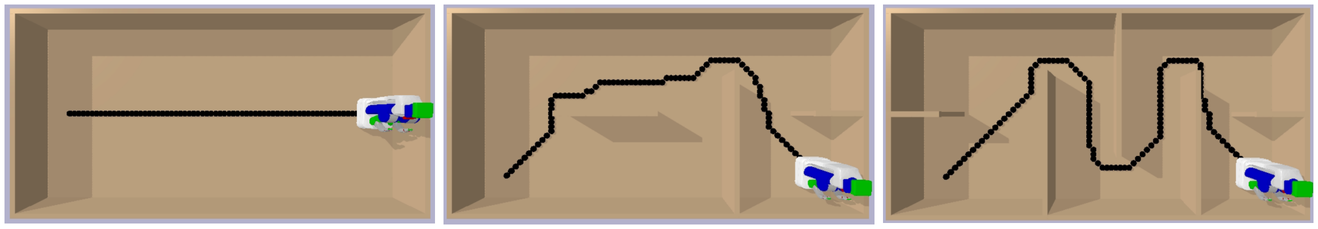
This formulation effectively links the robot's current state to its next state, factoring in the control inputs and the influence of its orientation. In this equation, A is the identity matrix, as the state has no change without controls or noise. A key aspect of our model is the use of a rotation matrix B, which transforms control inputs from the robot's local frame to the global frame. These inputs are mapped in the global frame, offering a coherent view of the robot's movement in relation to its environment. The B matrix is defined as:

$$B = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

To add realism and account for inherent uncertainties in robotic movement, motion noise (ε) is also introduced in the equation (1). The noise is quantified by a covariance matrix, specifically designed for indoor mobile robots:

$$R = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \tag{4}$$

This matrix encapsulates potential variances in the robot's motion, such as sensor inaccuracies or wheel slippage, providing a more accurate and realistic simulation of the PR2 robot's capabilities and limitations in various indoor environments.



(a) Open space (b) Some obstacles (c) Maze

Fig. 1: PyBullet maps and PR2 robot ground truth path

B. Sensor Model

In addition to the motion model, a comprehensive sensor model is crucial for the effective implementation of Kalman and Particle Filters. In real-world applications, mobile robots typically amalgamate data from both LIDAR and IMU to construct a detailed understanding of their environment. Our simulation mirrors this approach, employing a combination of these sensors to gather environmental data. The sensor measurements, denoted as z_t, are formulated as follows:

$$z_t = Cx_t + \delta \tag{5}$$

Here, the matrix C is a $\mathbb{R}^{3\times 3}$ identity matrix. The identity matrix, in this context, serves to directly relate the sensor measurements to the state variables without transformation. The term δ represents the sensor noise. Incorporating sensor noise is vital for simulating real-world conditions, as it accounts for the inherent inaccuracies and uncertainties present in sensor data. Given the nature of the data from LIDAR (position coordinates x, y) and IMU (orientation θ), it is important to note that these sensors operate independently, without influencing each other's readings. Consequently, we have structured our sensor model to reflect this independence. The covariance matrix Q, designed to quantify the sensor noise, is specified as:

$$Q = \begin{bmatrix} 0.02 & 0.001 & 0 \\ 0.001 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{bmatrix} \tag{6}$$

The diagonal elements of the matrix Q have identical values, indicating equivalent accuracy levels across both sensors. The off-diagonal value of 0.001 in the upper-left 2×2 section of the matrix implies a marginal interdependence between the x and y measurements from the LIDAR. This factor is introduced to account for potential minor interactions between these two coordinates. Furthermore, the zeroes in the matrix highlight the independence between the LIDAR and IMU readings, as their covariance is effectively null.

By meticulously tuning the sensor noise levels in the matrix Q, we aim to strike a balance where the noise is sufficient to present a challenging yet solvable problem for location estimation. This balance is critical for creating a realistic and practical simulation environment, which is essential for the effective development and testing of autonomous navigation algorithms.

C. Kalman Filter

The Particle Filter is encapsulated within the `KalmanFilter` class, which recursively combines information from motion and sensor models to estimate the PR2 robot's state. It operates in two phases: Prediction and Measurement Update (see Algorithm 1).

- In the prediction phase, the Kalman Filter uses the motion model to predict the next state of the robot and estimate the uncertainties associated with this prediction. The equations for the prediction phase are:

$$\overline{\mu}_{t+1} = A\mu_t + Bu_t \tag{7}$$
$$\overline{\Sigma}_{t+1} = A\Sigma_t A^T + R \tag{8}$$

  Here, $\overline{\mu}_{t+1}$ is the predicted mean state, $\overline{\Sigma}_{t+1}$ is the predicted covariance, μ_t is the current mean state, Σ_t is the estimated covariance of the current state, and R is the motion noise covariance matrix.
- During the Measurement update phase, the filter refines its state estimate by incorporating new sensor measurements. The update equations are:

$$K_t = \overline{\Sigma}_{t+1} C^T (C\overline{\Sigma}_{t+1} C^T + Q)^{-1} \tag{9}$$
$$\mu_{t+1} = \overline{\mu}_{t+1} + K_t(z_t - C\overline{\mu}_{t+1}) \tag{10}$$
$$\Sigma_{t+1} = (I - K_t C)\overline{\Sigma}_{t+1} \tag{11}$$

  In these equations, μ_{t+1} is the next mean state, Σ_{t+1} is the next covariance, K_t is the Kalman gain, z_t is the sensor measurement, Q is the sensor noise covariance matrix, and I is the identity matrix. The Kalman gain K_t strikes a balance between the new measurement and the prediction.

Algorithm 1 Kalman Filter Algorithm

```
Require: Initial state estimate $\mu$, initial covariance $\Sigma$
Ensure: Updated state estimate and covariance
1: Initialize $\mu$ and $\Sigma$
2: for each time step $t$ do
3:     Prediction Step
4:     Measurement Update
5: end for
6: return Updated state estimate and covariance
```

D. Particle Filter

The Particle Filter is encapsulated within the `ParticleFilter` class, which initializes a set of particles to represent possible states of the robot. Each particle represents a hypothesis of the robot's position and orientation in the environment, and update them based on the prior predicted by action model, posterior provided by sensor model, and particle resampling . `ParticleFilter` class and its methods are utilized in `main_pf` function.

- The function `__init__` initializes all the necessary variables for `ParticleFilter` class, including `num_particles`, which represents the number of particles in each step; `particles_t`, representing all particles that will be updated in the `action_model` method; `weight_t`, representing the weight of the corresponding particles that are updated in the `sensor_model` method; and `particles_tminus1`, which is updated in the `low_var_resample` method.
- The `random_init` method is also called during class initialization. This method initializes the particles at time zero randomly and stores them in `particles_t0`. The particles inside the obstacle will be excluded considering collision constraints if they are enabled.
- The filter updates the particles' states based on the robot's movements (via the `action_model` method), sensory observations (via the `sensor_model` method) and resampling (via the `low_var_resample` method).
- The `action_model` updates the state of each particle by simulating the effect of a control action, the system input is the position difference `dx, dy, dtheta`, incorporating uncertainty through a Gaussian noise model, and updating the possible positions of each particle in `particles_t`. In addtion, if collision checking is enabled, it will also run the `check_collision` to ensure no particles inside obstacle will be considered.
- The `sensor_model` updates the weights of the particles based on how well their predicted states align with actual sensor readings, again considering uncertainty. This is implemented by using the `multivariate_normal` function in scipy, which returned an object that has distribution with mean `z_t` and covariance `r`. The output probabilities is calculated by using `pdf` method.
- Low variance resampling is used for resample process, as shown in Algorithm 3. `low_var_resample` method to allow the particle filter to focus on more probable states. This algorithm chose a random number r and select those particles that correspond to $r+(m-1)M^{-1}$, where $m=1,...,M$ and $r \in [0, 1/M]$
- The `estimate_config` method then estimates the robot's state by computing a weighted average of all particles, which could be roughly considered as the center of a particle cluster.

The `main_pf` function provides the way to use the ParticleFilter class, which follows the psudo code of particle filter (see Algorithm 2), and it is also be called in demo.

Algorithm 2 Particle Filter

```
Require: $X_{t-1}, u_t, z_t$
Ensure: $X_t$
 1: $\hat{X}_t = X_t = \emptyset$
 2: for $m = 1$ to $M$ do
 3:     sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$
 4:     $w_t^{[m]} = p(z_t | x_t^{[m]})$
 5:     $\hat{X}_t = \hat{X}_t \cup \{(x_t^{[m]}, w_t^{[m]})\}$
 6: end for
 7: $X_t$ = Low Variance Resampling($\hat{X}_t$, $w_t$)
 8: return $X_t$
```

Algorithm 3 Low Variance Sampler

```
Require: $X_t, W_t$
Ensure: $\hat{X}_t$
 1: $\hat{X}_t = \emptyset$
 2: $r = \text{rand}(0, M-1)$
 3: $c = w_t^{[1]}$
 4: $i = 1$
 5: for $m = 1$ to $M$ do
 6:     $U = r + (m-1) \cdot M^{-1}$
 7:     while $U > c$ do
 8:         $i = i + 1$
 9:         $c = c + w_t^{[i]}$
10:     end while
11:     add $x_t^{[i]}$ to $\hat{X}_t$
12: end for
13: return $\hat{X}_t$
```

For the PyBullet part, it initializes the PyBullet environment, loads the robot and obstacle configurations, and defines the robot's kinematic constraints. After that, it reads the map and path, and interpolate the path to a specific number which allows the better control of steps in the whole algorithm. It then runs the Particle Filter over a series of timesteps, using control inputs and sensor data to update the particles by calling `action_model`, `sensor_model`, `low_var_resample`, and `estimate_config` in case of the robot is moved and sensor has measurements. It also visualizes the results, showing the ground truth path, sensor readings, estimated path by the Particle Filter, and particle distributions at various timesteps.

III. Simulations and Results

To rigorously evaluate the performance of the two filter localization algorithms, a series of three progressively complex maps were crafted. The initial map showcases a spacious open space, measuring 8 meters in length and 4 meters in width. Building upon this foundational open space map, two additional maps were developed: one interspersed with various obstacles and another representing a more intricate maze-like structure. Furthermore, to navigate these environments, a search-based planning algorithm, specifically the

A^* algorithm, was employed to determine optimal paths from designated start points to end points, as depicted in Figure 1. Each setting was executed by both localization filter, with the sensor measurement, estimated path plotted, and with Root Mean Square Error (RMSE) which has three values representing the RMSE in the x, y, and θ (orientation) dimensions, number of collisions and time spent recorded.

A. Open space map

The path we set in open space map start from $(-3.4, 0, 0)$ and end up with $(3.4, 0, -\pi/2)$, as shown in Figure 1 (a). The experiment result is displayed in Figure 2, and an example of particle clusters in given in Figure 3. Statistical result is recorded in Table I and II.

1) Kalman Filter (KF) Results: The blue line represents the actual path the robot took. It's a straight line, which suggests a consistent, unidirectional movement. The green dots represent raw sensor measurements. They are scattered around the ground truth, showing variability and noise inherent to sensor data. The red dots, which represent the Kalman Filter's estimation of the robot's path, are closely aligned with the blue ground truth line. This indicates that the Kalman Filter has effectively corrected the sensor noise and provided a good estimate of the robot's actual path. The Kalman Filter appears to perform well in this scenario, maintaining a tight cluster of estimates around the true path with a RMSE of $[0.072, 0.083, 0.079]$. The consistency of the estimations suggests that the process and measurement noise are well-tuned and the model accurately represents the system.

2) Particle Filter (PF) Results: As with the KF results, the blue line shows the ground truth path, which remains consistent. Green dots again show the sensor readings with a similar spread of noise around the ground truth. The red dots for the Particle Filter's estimation also follow the ground truth closely, but there appears to be a slight initial divergence at the start of the path. This divergence may indicate that the Particle Filter took some time to converge to the correct estimate, which can happen if the initial particle distribution is not representative of the actual state. Notably, PF also spent more time than KF.

Both filters successfully estimate the path of the robot in open space with noisy sensor data with no collides with obstacle. The Kalman Filter have a more consistent and faster estimation but particle filter's estimation has less variance from the ground truth. The Particle Filter shows an initial estimation error, which could be due to the nature of particle filters where the correct distribution of particles is achieved over time as more sensor data is integrated. These makes PF is less applicable than KF for simple scenario.

B. Obstacles map

The movement in map with some obstacles was set from $(-3.4, -1.4, 0)$ to $(3.4, -1.3, -\pi/2)$ (Figure 1 (b)). The graphical experiment plot is shown in Figure 4, and an example of particle clusters in given in Figure 5. The statistical result, including RMSE and number of collides in shown if Table I, and II.

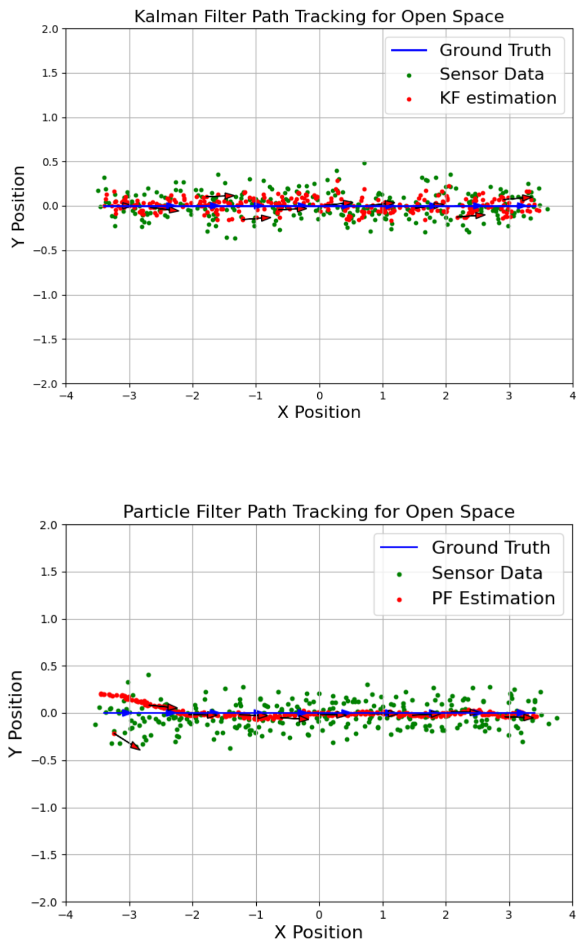


Fig. 2: Kalman filter and particle filter localization result on open space

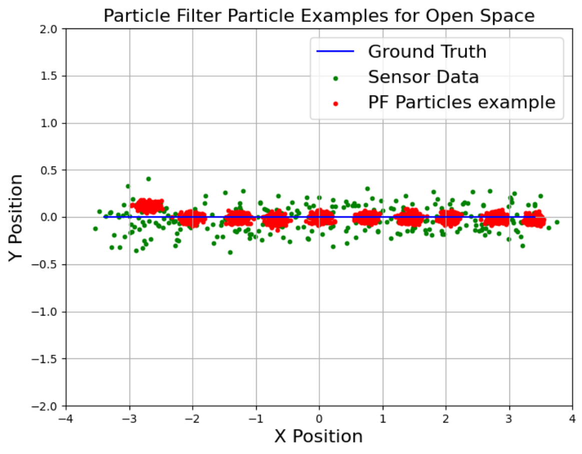


Fig. 3: Particle filter's particle cluster example on open space

TABLE I: Localization result: RMSE (x, y, θ)

Map	RMSE KF [m]	RMSE PF [m]
Open space	$[0.072, 0.083, 0.079]$	$[0.065, 0.031, 0.029]$
Some obstacles	$[0.083, 0.076, 0.081]$	$[0.076, 0.056, 0.048]$
Maze	$[0.085, 0.069, 0.073]$	$[0.062, 0.075, 0.050]$

TABLE II: Localization result: Collision Count and Time Spent

Map	Collision Count KF	Collision Count PF	Time KF [s]	Time PF [s]
Open space	0	0	1.36	4.06
Some obstacles	33	0	1.31	4.47
Maze	41	0	1.53	4.72

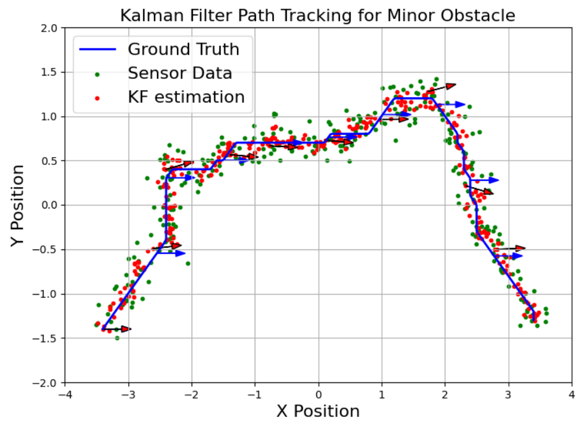
1) Kalman Filter (KF) Results: The KF estimation (red dots) closely follows the ground truth (blue line) but shows deviations, especially near the turns and corners, where the estimation seems to cut corners rather than precisely following the path. The red arrow shows the estimated robot orientation θ, which aligned quite good with ground truth red arrow. In addition, KF has an RMSE of $[0.083, 0.076, 0.081]$, indicating that the overall path estimation is acceptable and near to the ground truth. The KF estimation resulted in 33 collisions with obstacles, suggesting that the filter does not handle obstacle avoidance well.

2) Particle Filter (PF) Results: The PF estimation (red dots) also closely follows the ground truth and appears to handle turns and corners more accurately than the KF. Like KF, PF yields the orientation estimation also in a reasonable way. The estimations seem to adapt better to the shape of the path, reflecting the PF's ability to deal with non-linear motion and multi-modal distributions. The PF has a better (lower) RMSE of $[0.076, 0.056, 0.048]$ compared to the KF. This suggests that the PF has a more accurate localization, particularly when navigating around obstacles, which is often the case in real-world environments. Significantly, the PF estimation resulted in zero collisions with obstacles. This indicates that the PF is more adept at capturing the complexities of the environment and avoiding collisions, likely due to its ability to represent and update a probability distribution over the space of possible locations.

This result suggests that while the KF provides a decent estimation of the robot's path but have wrong estimation near obstacles. The PF outperforms it in complex environments where obstacle avoidance and non-linear movements are essential but it takes more time to compute. The lack of collisions and lower RMSE scores for the PF highlight its robustness and effectiveness for this particular task.

C. Maze map

The maze map is a more complex version of obstacle map, which makes the robot travel through more corners. The path we have in maze map is also started from $(-3.4, -1.4, 0)$ and end with $(3.4, -1.3, -\pi/2)$. The top view figure result is provided in Figure 6 and 7, with the RMSE, number of



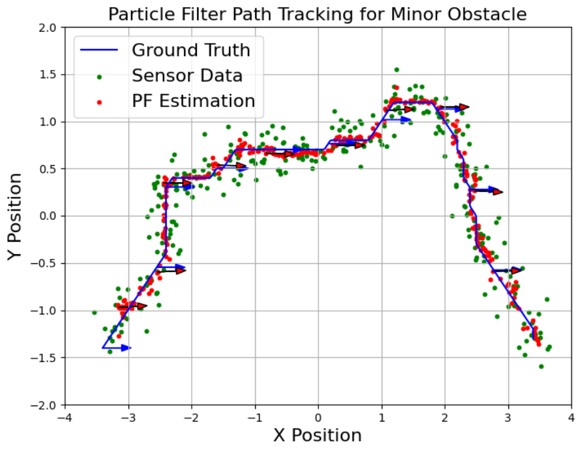


Fig. 4: Kalman filter and particle filter localization result on map with obstacles

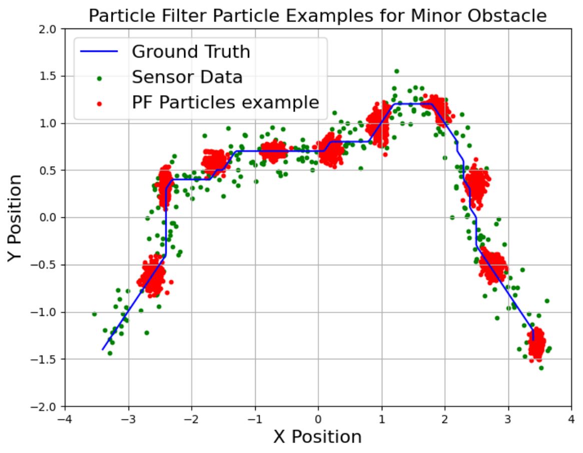


Fig. 5: Particle filter's particle cluster example on map with obstacles

collides and time spent recorded in Table I, and II.

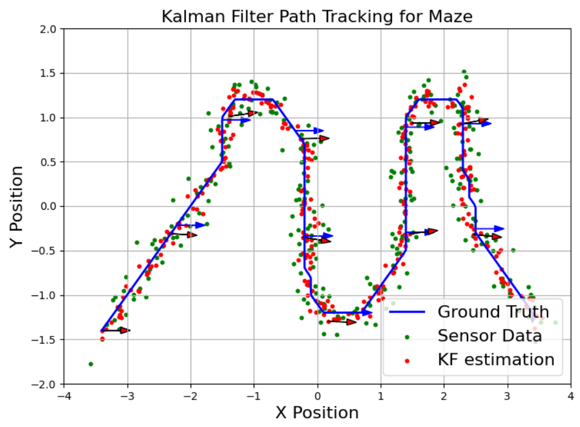
1) Kalman Filter (KF) Results: Similar to the result in map with some obstacles, KF's estimated path (red dots) on maze generally follows the ground truth (blue line), but there are notable deviations, particularly around corners and turns. The reported RMSE values for the KF are $[0.085, 0.069, 0.073]$. These values suggest that the KF has a moderate level of accuracy in estimating the robot's path. The first and third values are higher compared to the PF, indicating more significant errors in those dimensions. The KF estimation resulted in 41 collisions with obstacles. This high number of collisions indicates that the KF struggles to accurately navigate around obstacles, and more obstacles might leads to a higher number of collides.

2) Particle Filter (PF) Results: The PF's estimated path (red dots) closely aligns with the ground truth, showing tight tracking even around complex turns in the maze. The PF has RMSE values of $[0.062, 0.075, 0.050]$, which are generally lower than those of the KF, indicating a more accurate path estimation. While the second value is slightly higher than the corresponding KF value, the first and third are lower, suggesting better performance in those dimensions. Notably, the PF estimation still had zero collisions with obstacles. This suggests that the PF is more adept at handling the non-linearity and complexities of the maze environment, maintaining an accurate estimate of the robot's position without intersecting with obstacles. However, the computation time spent by PF is also much higher than KF.

PF outperforms the KF in the maze environment by providing a more accurate estimation of the robot's path and avoiding any collisions with obstacles, suggesting that it is better suited for tasks requiring robust navigation and handling of complex, non-linear dynamics if the higher computational cost is acceptable.

IV. Conclusions

In this project, we designed three increasingly complex maps featuring a variety of obstacles. We programmed a robot to navigate these maps using the optimal route determined by the A^* algorithm, thereby simulating real-world scenarios. For the robot's movement, we implemented an omnidirectional action model with a diagonal covariance matrix. Additionally, our sensor model incorporated a covariance that accounted for the impact of different sensor data sources. The results of our simulation highlighted the efficiency and robustness of both the Kalman Filter and Particle Filter. Specifically, the Kalman Filter demonstrated lower computational costs across all scenarios and performed exceptionally well when system dynamics and sensor measurements are linear and maps are less complex. In contrast, the Particle Filter exhibited varying performance, often larger than the Kalman Filter, due to initial estimation errors. Nevertheless, it consistently showed low Root Mean Square Error (RMSE) values in both linear and non-linear scenarios for all maps, highlighting its precision under various conditions.



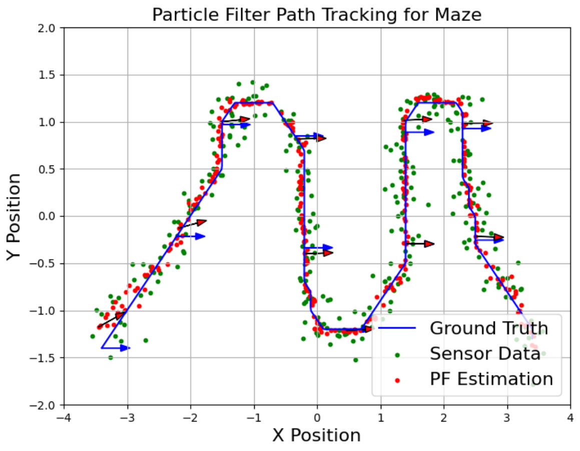


Fig. 6: Kalman filter and particle filter localization result on maze

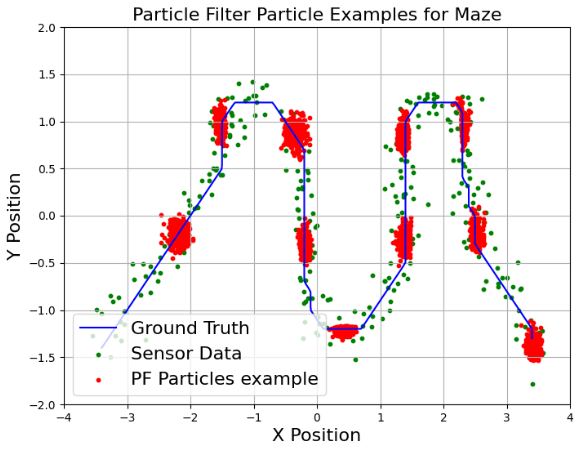


Fig. 7: Particle filter's particle cluster example on maze map