Comparision and Analysis of Kalman Filter and Particle Filter in Robot Localization

Project Report, EECS 498 Introduction to Algorithmic Robotics, Fall 2021

Instructor: Dmitry Berenson, Graduate Student Instructor: Tianyi Li

December 20, 2021

Jiayi Pan*
EECS Department
University of Michigan
Ann Arbor, United States
jiayipan@umich.edu

Changyuan Qiu*
EECS Department
University of Michigan
Ann Arbor, United States
peterqiu@umich.edu

I. Introduction

Robot localization is the problem of estimating a robot's pose relative to a map of its environment [1], [2]. Applications of robot localization span the whole field of autonomous robotics. This problem is essential for autonomous robotics as a robot will not be able to execute commands and accomplish tasks successfully if it can not determine its current pose accurately, and it has been referred to as "the most fundamental problem to providing a mobile robot with autonomous capabilities" in [3].

In an unstructured real-world environment, when a robot executes an action, its actual pose will be different from the target pose due to the inevitable motion noise. Moreover, sensor noises add more uncertainties to the problem, making accurate localization even harder.

In this project, we introduced, compared and analyzed two canonical robot localization algorithms: Kalman Filter and Particle Filter [4]–[6]. In the scenario of a PR2 robot navigating in our designed environment with obstacles following a specific path, we implemented Kalman Filter and Particle Filter for localization. Based on simulation experiments results in Pybullet [7], we made a fair comparison of strengths and weaknesses of the two methods.

Index Terms

Robot Localization, Kalman Filter, Particle Filter

II. Approach & Implementation

A. Step by Step Quasi-static Motion Model

In this paper, we use the step by step quasi-static 2D motion to model the motion of the mobile robot. In this motion model, the robot moves step by step in a quasi-static manner, so that its velocity and acceleration converge to 0.

The state of robot s therefore can be represented only by its current position

$$\mu = \begin{bmatrix} x \\ y \end{bmatrix} \tag{1}$$

And the action (step) a as

$$u = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{2}$$

However, due to the existence of noise, after the robot takes an action towards the target, the resulting state is disturbed and the robot can only reach somewhere near the target, which we model as

$$\mu_{t+1} = A \times \mu_t + B \times u_t + n \tag{3}$$

*Equal Contributions.

where A is the state-transition model matrix

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4}$$

B is the control-input model matrix

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5}$$

n follows the multivariate normal (Gaussian) distribution with process noise covariance matrix R

$$n \sim \mathcal{N}(0, R) \tag{6}$$

B. Sensor Model with Gaussian Noise

Mobile robots are usually equipped with various sensors, like GPS, camera, microphone, LiDar to constantly update their knowledge about the surrounding environment.

In this setup, for the localization purpose, we consider the GPS sensor with input z representing its sensed position

$$z = \begin{bmatrix} x \\ y \end{bmatrix} + n' \tag{7}$$

It is noteworthy that due to the existence of noise, the sensed position z is not accurate. As Kalman filter relies on the assumption that errors obey normal (Gaussian) distribution, we model the noise n' accordingly

$$n \sim \mathcal{N}(0, Q) \tag{8}$$

where Q it its covariance matrix (also called Kalman gain under the setup of Kalman filter).

C. Kalman Filter

A general picture of the Kalman Filter method is illustrated in Algorithm 1, with detailed procedures and implementations of Kalman Filter illustrated in Algorithm 2.

Algorithm 1 Kalman Filter Method

Input: Gaussian of robot's initial state (μ_0, Σ_0)

while Robot takes action u_t at time t **do**
  get sensor input z_t
  estimate Gaussian of robot's state (μ_t, Σ_t) = Kalman Filter$(\mu_{t-1}, \Sigma t-1, u_t, z_t)$
end while

Algorithm 2 Kalman Filter

Input: Gaussian of the estimated current state (μ_{t-1}, Σ_{t-1}), action u_t, sensor input z_t

Set: state-transition model matrix A, control-input model matrix B, observation model matrix C, process noise matrix R, Kalman gain matrix Q

$\bar{\mu_t} \leftarrow A\mu_{t-1} + Bu_t$ ▷ Prediction
$\bar{\Sigma_t} \leftarrow A\Sigma_{t-1}A^T + R$
$K_t = \bar{\Sigma_t}C^T(C\bar{\Sigma_t}C^T + Q)^{-1}$ ▷ Correction
$\mu_t = \bar{\mu_t} + K_t(z_t - C\bar{\mu_t})$
$\Sigma_t = (I - K_tC)\bar{\Sigma_t}$

Output: Gaussian of estimated next state (μ_t, Σ_t),

D. Particle Filter

A general picture of the Particle Filter method is illustrated in Algorithm 3, with detailed procedures and implementations of Particle Filter method illustrated in Algorithm 4.

Algorithm 3 Particle Filter Method

Input: Number of particles M, particles initial distributions P with weigts W

while Robot takes action u_t at time t **do**
  get sensor input z_t
  update particles and weights (P, W) = Particle Filter(P, W, M, u_t, z_t)
end while

Algorithm 4 Particle Filter

Input: particles from previous state P with weights W, number of particles M, action u_t, sensor input z_t
Set: state-transition model matrix A, control-input model matrix B, covariance matrix for sampling Σ, function to check if a particle is in reachable region $IsReachable(\cdot)$

for particle p in P **do** ▷ Update and sample particles (Propagate)
  $p' = A \times p + B \times u_t$
  sample new particle p'_s from distribution $N(p', \Sigma)$
  while not $IsReachable(p'_s)$ **do**
    re-sample a new particle p'_s from distribution $N(p', \Sigma)$
  end while
  $p = p'_s$
end for
for $m = 1$ to M **do** ▷ Update Weight
  $W[m] = \det(2\pi\mathbf{\Sigma})^{-\frac{1}{2}} e^{-\frac{1}{2}(z_t - P[m])^\top \mathbf{\Sigma}^{-1}(z_t - P[m])}$
end for
Construct empty particle list P' ▷ Resample
for $m = 1$ to M **do**
  draw i with probability $\propto W[i]$
  add $P[i]$ from P to P'
end for
$P = P'$

Output: updated particles P with weights W

III. Experiments & Results

A. Experiment Setup

In order to make a fair comparison between Kalman Filter and Particle Filter under different circumstances, we designed the environments as well as robot's execution path. An overview of the environment as well as the robot's execution path is illustrated in Figure 1.

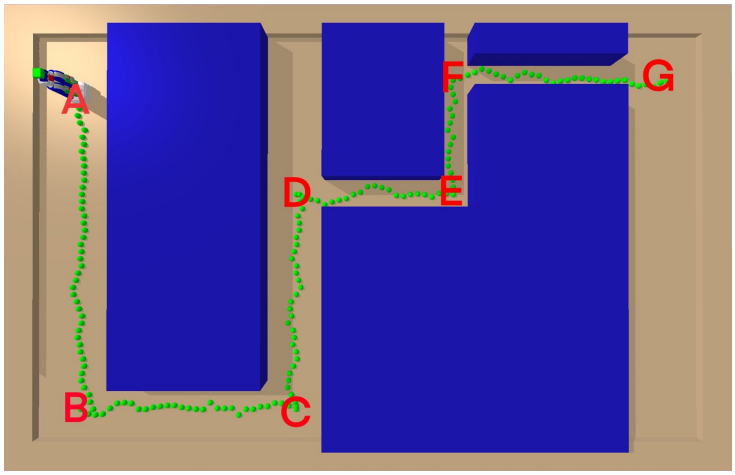


Fig. 1: An Overview of the map "Maze" and robot's execution path

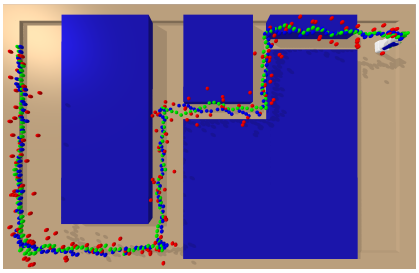
More specifically, we divide the path into 3 sub-paths:

- Path A-B. Where the path is most spacious and flat and the robot will not have any collisions with the walls.
- Path B-C-D. Where the path becomes narrower and based on the motion noise the robot might have some collisions with the walls.
- Path D-E-F-G. Where the path is the narrowest and the robot is expected to have a lot of collisions with the walls.

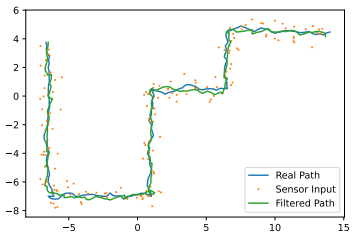
B. Result

Comprehensive experiments were conducted under this simulation environment.

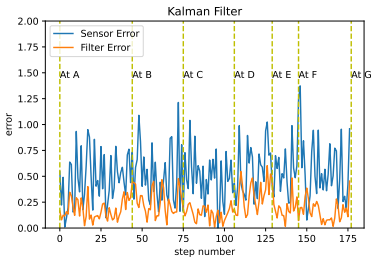
Result of Kalman Filter



(a) PyBullet Simulation

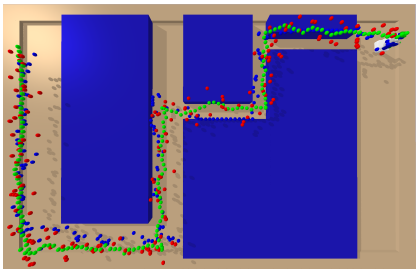


(b) Real Path, Sensor Input, Filter Result

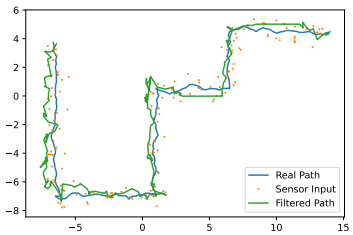


(c) Error after/before filtering

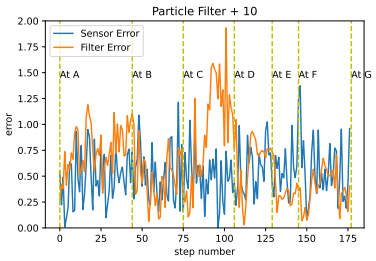
Result of Particle Filter with 10 Particles



(d) PyBullet Simulation

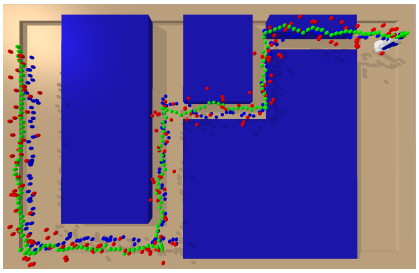


(e) Real Path, Sensor Input, Filter Result

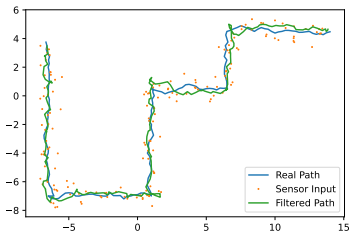


(f) Error after/before filtering

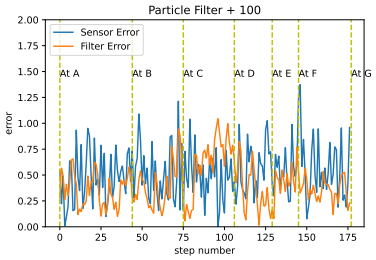
Result of Particle Filter with 100 Particles



(g) PyBullet Simulation

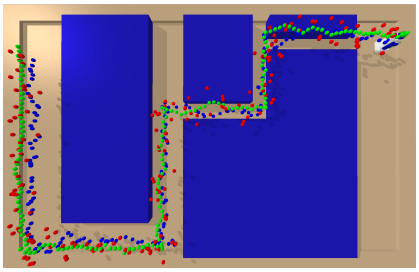


(h) Real Path, Sensor Input, Filter Result

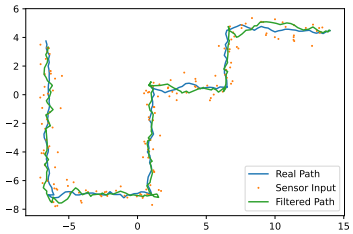


(i) Error after/before filtering

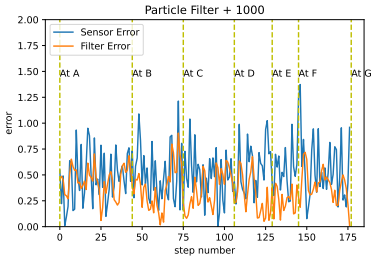
Result of Particle Filter with 1000 Particles



(j) PyBullet Simulation

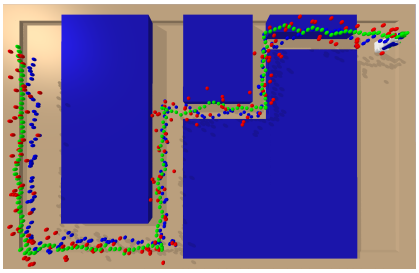


(k) Real Path, Sensor Input, Filter Result

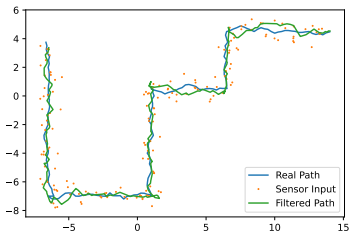


(l) Error after/before filtering

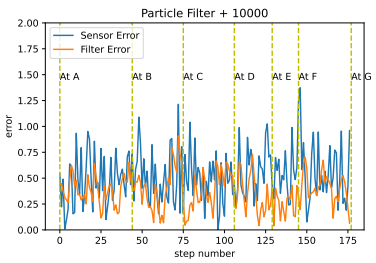
Result of Particle Filter with 10,000 Particles



(m) PyBullet Simulation



(n) Real Path, Sensor Input, Filter Result



(o) Error after/before filtering

Fig. 2: Experiment Results in Pybullet Simulation

We apply Kalman filter and particle filter methods with 10, 100, 1000, 10000 particles respectively in this setup and summarized the result in Figure 2 and Table I.

For ablation study purposes, the sensor noise was generated beforehand and was the same for all methods.

Method Name	Accumulated Error	Runtime
Base	-	16.698s
Kalman Filter	35.718	17.176s
Particle Filter with 10 particles	109.984	17.721s
Particle Filter with 100 particles	73.732	19.542s
Particle Filter with 1000 particles	66.172	30.330s
Particle Filter with 10,000 particles	66.544	140.861s

TABLE I: Quantitative Experiment Results. Base denotes running the simulation without applying any localization algorithm

C. Discussion & Conclusion

- Kalman filter is more efficient than Particle Filter.
  We discovered that Kalman filter is more efficient than particle filter. In our experiment, from Table I, Kalman filter consumes only 24% of time for particle filter with 10 particles and adds little cost to the overall simulation runtime. However, running particle filter with more particles (eg: 1000, 10000) will significantly increasing the simulation time. For particle filter with 10,000 particles, it takes about 790X more time than the Kalman filter.
  The underlying reason is that while Kalman filter uses Gaussian distribution to model the state of robot, particle filter relies on sampling various particles to model it, which is less computationally efficient.
- Kalman filter has lower overall error under this setup.
  We discovered that Kalman filter generally has lower error under this setup. From Table I, we can see that the accumulated error for Kalman filter is lower than any setup of the particle filter, demonstrating the strength of Kalman filter when facing Gaussian noise.
- Particle filter generates realistic result while Kalman might not.
  Though Kalman filter produces lower error overall, we discovered that the output of particle filter is more realistic. From Figure 2, we can find that though Kalman filter generates satisfactory results during Path A-D, during Path D-G, where the path is the narrowest, the Kalman filter sometimes predicts that the robot is within the wall, which is unrealistic. On the other hand, for the particle filters, whatever the setup, the output is always realistic (meaning that the predicted location is within reach).
  The underlying reason is that while particle filter always samples points in the permissible area, Kalman filter does not take it into consideration .
- Particle filter is more robust to when most of surrounding space is unreachable.
  Due to the same reason and analysis of the last discussion, during Path D-G, where most of the surrounding space is unreachable, particle filter generates better result than the Kalman filter.

IV. Conclusion

In this project, we introduced, compared and analyzed two robot localization algorithms: Kalman Filter and Particle Filter. Extensive experiments were conducted under a curated simulation environment which covers scenarios including PR2 robot moving forward in a spacious environment, turning around, and moving in corridors with different widths. After comparison and analysis, we conclude that while Kalman filter is more efficient and generates lower error overall under Gaussian noise, particle filter is more robust and generates more realistic outputs when most of the surrounding space is unreachable.

REFERENCES

[1] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle filters for mobile robot localization," in *Sequential Monte Carlo Methods in Practice*, ser. Statistics for Engineering and Information Science. Springer, 2001, pp. 401–428.

[2] M. Betke and L. Gurvits, "Mobile robot localization using landmarks," *IEEE Trans. Robotics Autom.*, vol. 13, no. 2, pp. 251–263, 1997.

[3] I. J. Cox, "Blanche-an experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Trans. Robotics Autom.*, vol. 7, no. 2, pp. 193–204, 1991.

[4] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[5] O. Stepanov, "Kalman filtering: Past and present. an outlook from russia.(on the occasion of the 80th birthday of rudolf emil kalman)," *Gyroscopy and Navigation*, vol. 2, no. 2, pp. 99–110, 2011.

[6] P. Del Moral, "Nonlinear filtering: Interacting particle resolution," *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, vol. 325, no. 6, pp. 653–658, 1997.

[7] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.