Comparison of Mobile Robot Localization Methods

Huijie Zhang
Mechanical Engineering Department
University of Michigan
Ann Arbor, MI, USA
huijiezh@umich.edu

Yiting Zhang
Mechanical Engineering Department
University of Michigan
Ann Arbor, MI, USA
yitzhang@umich.edu

***Abstract*—The goal of this project is to do localization for the mobile robot PR2 on the OpenRAVE using different filters, evaluate and compare their performance in terms of the robustness in the environment with obstacles and noise. Filters, robot dynamic functions, sensors functions and path planning algorithms are implemented. Experiments are conducted on several interesting scenarios. We find that the Extended Kalman Filter (EKF) is more efficient than the Particle filter while the Particle filter performs more robust to the highly nonlinear dynamics and complex environments with many obstacles.**

***Index Terms*—Robotics Localization, Kalman Filter, Particle Filter**

I. Introduction

Estimation is a research area with a long history. It is a process of using some tractable data to estimate some unreachable value. The output would be usable even when the input is some kind of unstable, because the process makes best use of available information, in a probabilistic way. Starting before 21st century, many well-developed algorithms had been designed to solve estimation problem. Because of their explainability and robustness, even for some most state-of-art techniques, some estimation methods can be combined to make it more reliable. [1] [2] [3]

Although estimation has a wild range of applications, robot localization is the mainly focused area in this report. Robot localization is one of the bottlenecks in mobile robot. It is the foundation of many high level robot actions, such as navigation and mapping. While in a environment full of uncertainty, robot is hard to know its accurate location. So what estimation does is to estimate robot's position in a probability distribution. Locations with higher probability density would be more likely to be occupied by the robot.

Kalman filter might be one of the most classic estimation algorithms widely used in robotics. It treats any model as a Gaussian distribution. Kalman filter can be divided into two stage: use system dynamics to predict future state; use sensor data to correct predicted state. However, it could only work on linear system, so Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) are developed for nonlinear systems. Kalman filter is very computationally efficient, thus they are always used for robots with limited computing power. However, it can never estimate distribution other than Gaussian, which is useless in some non-guassian tasks. Additionally, the Kalman filter could only deal with some sensor measurement with explicit sensor model, data from sonar or laser would be intractable.

Particle filter is another useful algorithm, which is also named as Markov Chain Monte Carlo (MCMC) in some literature. Particle filter uses a set of particles (also called samples) to represent the posterior distribution of some stochastic process given noisy and partial observations. The state-space model can be nonlinear and the initial state and noise distributions can take any form required. Particle filter techniques provide a well-established methodology for generating samples from the required distribution without requiring assumptions about the state-space model or the state distributions. [4] [5] [6]

II. Method

A. Kalman Filter

A standard Kalman filter is implemented in this project. Kalman filter uses a series of measurements observed over time, containing sensor noise and other inaccuracies, and make predictions of the current state, which is more accurate than estimating only based on a single measurement.

Consider the dynamical system of a discrete-time controlled process:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{1}$$

the measurement is given by:

$$z_t = C_t x_t + \delta_t \tag{2}$$

where x_t, u_t and z_t denote the state, input and measurement at time t respectively; ϵ_t and δ_t denote the process and sensor noise applied to the state and measurement at time t. Kalman filter assumes that the noise, ϵ_t and δ_t, are independently and normally distributed with covariance R_t and Q_t respectively. In each time step, Kalman filter computes a Gaussian probability distribution of the state, with parameter μ_t and Σ_t. It firstly uses dynamics A_t and B_t to predict a state the robot will be, then it uses the measurement z_t to correct former prediction. The pseudocode of the Kalman filter is shown as Algorithm 1. [7]

B. Extended Kalman Filter

The Extended Kalman Filter (EKF) is the nonlinear version of the Kalman filter. Consider the system function:

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \epsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned} \tag{3}$$

Algorithm 1 Kalman Filter

Input: μ_{t-1}, Σ_{t-1}, u_t, z_t, A_t, B_t, C_t, Q_t, R_t
Output: μ_t, Σ_t

$\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$
$\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
$K_t = \bar{\Sigma}_t C_t^T \left(C_t\bar{\Sigma}_t C_t^T + Q_t\right)^{-1}$
$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$
$\Sigma_t = (I - K_t C_t)\,\bar{\Sigma}_t$
return μ_t, Σ_t

where the state transition function $g(u, x)$ and observation function $h(x)$ do not need to be linear but may instead be differentiable functions. ϵ_t and δ_t denote the process and sensor noise, which are independently and normally distributed with covariance R_t and Q_t respectively. Instead of dealing with nonlinear functions directly, the EKF applies a local linear approximation. Though the EKF still computes a Gaussian distribution each time step as the standard Kalman filter, it uses the Jacobians of the state and observation. It assumes that:

$$\begin{aligned} x_t &\approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \\ z_t &\approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned} \tag{4}$$

where $G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$ and $H_t = \frac{\partial h(\bar{\mu}_t)}{x_t}$ are the Jabocians of g and h respectively. Each time step of the EKF can also be separated into two parts: prediction and correction, which is the same as the Kalman filter. The pseudocode of the EKF is shown as Algorithm 2.

Algorithm 2 Extended Kalman Filter

Input: μ_{t-1}, Σ_{t-1}, u_t, z_t, g, h, Q_t, R_t
Output: μ_t, Σ_t

$\bar{\mu}_t = g(u_t, \mu_{t-1})$
Compute the Jacobian G_t of $g(u_t, \mu_{t-1})$
$\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + R_t$
Compute the Jacobian H_t of $h(\bar{\mu}_t)$
$K_t = \bar{\Sigma}_t H_t^T \left(H_t\bar{\Sigma}_t H_t^T + Q_t\right)^{-1}$
$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
$\Sigma_t = (I - K_t H_t)\,\bar{\Sigma}_t$
return μ_t, Σ_t

C. Particle Filter

Different from the Kalman filter and the EKF which assumes all error only in Gaussian, the basic idea of the Particle filter is that any probability distribution can be represented as a set of samples (particles). In a complex environment with many obstacles around the robots, if our probability distribution looks like the two-humped line, we can represent that just by drawing a whole lot of samples from it, so that the density of the samples in one area of the state space represents the probability of that region. Each particle has one set of values for the state variables. This method can represent any arbitrary distribution, making it good for non-Gaussian, multi-modal probabilities. The body of Particle filter is shown as Algorithm 3. [9]

Algorithm 3 Body of Particle Filter

$X = Initialize(M)$
while True **do**
    $X.ApplyInput()$
    $X.UpdateWeight()$
    if $X.NeedReinitialize()$ **then**
        $X.Reinitialize()$
    end if
    $X.ReSample()$
end while

$Initialize$: the function returns M independent identically distribution samples from a state space $\mathbb{R}^N$ of the robot and put them into a set X, where:

$$X = x^{[1]}, x^{[2]}, \cdots, x^{[M]} \in \mathbb{R}^N \tag{5}$$

$ApplyInput$: During each iteration, input would be applied to each particles. Each particles' state would be update follow the dynamic function (Eq.1) for a linear system. This process could also preserving diversity because the randomness of ϵ_t in (Eq.6). In some related works, a function named $diffusion$ have the same purpose.

$$x_t^{[i]} = A_t x_{t-1}^{[i]} + B_t u_t + \epsilon_t \qquad i = 1, 2, \cdots, M \tag{6}$$

$UpdateWeight$: The weight of each particle represent the probability of the system in the same state of that particle. Given the sensor output $Z_{1:t-1} = \{Z_1, Z_2, \cdots, Z_{t-1}\}$, the weight particle $x_t^{[i]}$ could be denoted as $weight(x_t^{[i]}) = P(x_t^{[i]}|Z_{1:t})$. From Bayes's rules:

$$P(x_t^{[i]}|Z_{1:t}) = \frac{P(Z_t|x_t^{[i]})P(x_t^{[i]}|Z_{1:t-1})}{P(Z_{1:t}|Z_{1:t-1})} \tag{7}$$

In (Eq.7) $P(x_t^{[i]}|Z_{1:t-1})$ is the prior probability density for particle $x^{[i]}$ in time t, after $ReSample$ function, $P(x_t^{[i]}|Z_{1:t-1}) = 1$. And $P(Z_{1:t}|Z_{1:t-1})$ is just a normalized factor, to guarantee $\sum_{i=1}^{M} weight(x_t^{[i]}) = 1$. So the $weight(x_t^{[i]})$ is only related to $P(Z_t|x_t^{[i]})$, which is called sensor model. In this paper, we assume the observation $z_t^{[i]}$ for particle $x^{[i]}$ at time t should follow a normal distribution, that is $z_t^{[i]} \sim \mathcal{N}(Z_t, \sigma)$, σ is the sensor deviation, and by (Eq.2), $z_t^{[i]} = C_t x_t^{[i]} + \delta_t$, so:

$$\begin{aligned} P(Z_t|x_t^{[i]}) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-(\frac{z_t^{[i]} - Z_t}{\sqrt{2}\sigma})^2} \\ z_t^{[i]} &= C_t x_t^{[i]} + \delta_t \end{aligned} \tag{8}$$

In some other paper [2], sensor model is simplified to:

$$\begin{aligned} P(Z_t|x_t^{[i]}) &= e^{-(z_t^{[i]} - Z_t)^2} \\ z_t^{[i]} &= C_t x_t^{[i]} + \delta_t \end{aligned} \tag{9}$$

$Reinitialize$: The function works the same as $Initialize$ function, but would work only if the samples need to be

reinitialized. And if $\max_i (weight(x_t^{[i]})) < k$, samples need to be reinitialized. k is a constant and we set it to 10^{-5} in this paper.

$ReSample$: We use Low-variance sampling to re-sample the particles, it is shown in Algorithm 4. This re-sample algorithm could maintain the number of total particles. And after we apply input to them, the diversity would be preserved.

Algorithm 4 Low-variance sampling

$X_t = \{\}$
$r = random() * M^{-1}$
$n = 0$
$WeightSum = 0$
$index = 0$
while $n < M$ **do**
  if $r > WeightSum$ **then**
    $WeightSum = WeightSum + weight(x_t^{[index]})$
    $index = index + 1$
  else
    $r = r + M^{-1}$
    $X_t = X_t \cup (x_t^{[index-1]}, weight(x_t^{[index-1]}) = 1)$
    $n = n + 1$
  end if
end while
$X = X_t$

III. Experiment Setup

To realize the goal of this project, we consider and implement functions for multiple types of robot dynamics and sensors. We perform our experiments on two different environment and implement the A star and PID control for robot's path planning in addition to manually controlling the moving trajectory of the robot.

A. Robot dynamics

The robot has a state configuration $\mathbf{x} = (x, y, \theta)$, where x and y are the location of the robot in the Cartesian coordinate, θ is the rotation angle between the x direction in the world frame and the robot facing direction. We implement two types of robot dynamics including a simple linear dynamics and a nonlinear dynamics. We also implement different inputs corresponding for different types dynamics.

1) Linear dynamics: In terms of the linear dynamics, the robot uses input $\mathbf{u} = (u_x, u_y, u_\theta)$, which is definitely linear. Consider the robot can never 100% accurately performing the input people have coded in the real world, we include the process noise when the input is given and the state changes in each time step. The noise value is selected from a Gaussian distribution with zero mean and a certain covariance matrix R. Denote the noise as ϵ, the state transition function for the robot with simple linear dynamics for one step is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{u}_t + \epsilon_t \tag{10}$$

2) Nonlinear dynamics: We consider a robot which can only receive signals telling its to go forward (or backward) and make turns based on its current state. The input only includes two values: Δx, the step size of the robot going forward, and $\Delta\theta$, the rotation angle the robot perform based on the current state. Same as the linear dynamics, we also includes the process noise ϵ into the state transition function. Thus, we have the one-step state update function:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \begin{bmatrix} \Delta x_t \cos(\theta_t) \\ \Delta x_t \sin(\theta_t) \\ \Delta\theta_t \end{bmatrix} + \epsilon_t \tag{11}$$

It is obvious that this function is a nonlinear function, which means that the robot has a nonlinear dynamics.

3) Dynamical system noise: Assume all of the dynamical system noise ϵ_t in the experiment is independent and identically distributed from a Gaussian distribution, i.e. $\epsilon_t \sim \mathcal{N}(0, R_t)$, where the covariance R_t is given as (Eq.12).

$$R_t = \begin{bmatrix} 2.5*10^{-3} & 1.8*10^{-5} & 1.8*10^{-6} \\ 1.8*10^{-5} & 2.5*10^{-3} & 1.8*10^{-6} \\ 1.8*10^{-6} & 1.8*10^{-6} & 2.5*10^{-4} \end{bmatrix} \tag{12}$$

B. Sensors

The perception of the mobile robot is performed by sensors. Since all the simulation are on the OpenRAVE platform, we do not have real sensors and the robot cannot receive the measurement data for localization except we implement functions by ourselves to simulate sensors. Four types of sensors are considered in this project: standard GPS, IMU-enabled GPS, Landmark and Laser.

1) Standard GPS: A standard GPS sensor is implemented. The GPS sensor returns the robot location in the world frame directly. Besides, the measurement noise δ_t, with zero means and covariance matrix Q, is also considered and included in the measurement z_t. Thus, the measurement is given by:

$$\mathbf{z}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \delta_t \tag{13}$$

2) IMU-enabled GPS: An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. In the real world, an IMU always be integrated into GPS. [10] Thus, we implemented an IMU-enabled GPS in this project. In addition to the position of the robot, the sensor also gets the pose – the angle between the robot facing direction and the x-axis direction in the world frame. Similarly, the sensor noise δ_t is also considered. Then the measurement is given by:

$$\mathbf{z}_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \epsilon_t = \mathbf{x}_t + \delta_t \tag{14}$$

An interesting thing is that the IMU-enabled GPS can just get the complete state information of the robot.

3) Landmark: Consider a situation that the robot is in an environment with obstacles, a set of landmarks is located and all of their locations are known. Landmark sensor is applied for detecting the distance and direction between the robot and landmarks. If there is only one landmark, the measurement from the sensor includes two values (distance and direction); if there are N landmarks in total, then the measurement will contains $2 * N$ values. Including the sensor noise δ_t, the measurement is in the form as:

$$\mathbf{z}_t = \begin{bmatrix} \sqrt{(x_{l1} - x_t)^2 + (y_{l1} - y_t)^2} \\ \tan^{-1}((y_{l1} - y_t)/(x_{l1} - x_t)) \\ \cdots \\ \cdots \\ \sqrt{(x_{lN} - x_t)^2 + (y_{lN} - y_t)^2} \\ \tan^{-1}((y_{lN} - y_t)/(x_{lN} - x_t)) \end{bmatrix} + \delta_t \qquad (15)$$

where x_{li} and y_{li}, $i = 1, 2, ..., N$, are the location of landmarks.

4) Laser: Laser sensors are used where small objects or precise positions are to be detected. In this project, a laser sensor is implemented based on the built-in laser sensor in OpenRAVE. The sensor function output the distance from obstacles to the robots returned by laser beams.

5) Sensor noise: Assume the sensor noise are all independent and identically distributed over Gaussian distributions, i.e. $\delta_t \sim \mathcal{N}(0, Q_t)$ Different sensors have different covariance matrices Q_t.

For Standard GPS sensor:

$$Q_t = \begin{bmatrix} 4.87 * 10^{-1} & -5.86 * 10^{-3} \\ -5.86 * 10^{-3} & 4.87 * 10^{-1} \end{bmatrix} \qquad (16)$$

For IMU-enabled GPS sensor:

$$Q_t = \begin{bmatrix} 4.87 * 10^{-1} & -5.86 * 10^{-3} & -5.86 * 10^{-5} \\ -5.86 * 10^{-3} & 4.87 * 10^{-1} & -5.86 * 10^{-5} \\ -5.86 * 10^{-5} & -5.86 * 10^{-5} & 4.87 * 10^{-3} \end{bmatrix} \qquad (17)$$

For a single Landmark sensor:

$$Q_t = \begin{bmatrix} 4.87 * 10^{-1} & -5.86 * 10^{-3} \\ -5.86 * 10^{-3} & 4.87 * 10^{-2} \end{bmatrix} \qquad (18)$$

And covariance between different Landmark sensors output would be 0 (they are independently work).

C. Maps

To better investigate features of the filters and be more generalized, we setup two different maps. One is a simple square space with several cylinder obstacles. The other one is a more complicated office environment, which is a commonly-used environment for the test of the performance of localization.

1) Space with obstacles: Consider a square space with size $12m \times 12m$, 6 cylinder obstacles are located sparsely around the center of the space. Each obstacle has the radius $0.2m$ and the height $1.0m$. The minimum distance between obstacles are $2.0m$. Particularly, when the landmark sensor is applied to the robot, the location of landmarks can be the same as the location of the obstacles, since it is reasonable to regard landmarks also as obstacles in the view of the robot.

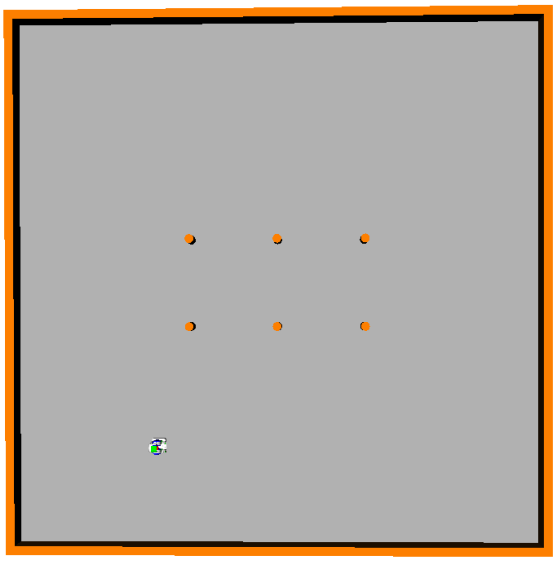


Fig. 1. Square space with 6 cylinder obstacles

2) Office environment: Since the environments in the real life are usually more more complicated than a square space with obstacles, an office environment is considered in this project. Here Fig. 2 shows a floor plan of a $27m \times 20m$ section office space. We try our best to make the environment more realistic by adding tables and shelves in the office. This can also make the localization easier for the robot since it can more easily recognize the difference from the measurement of surrounding environment. [8]

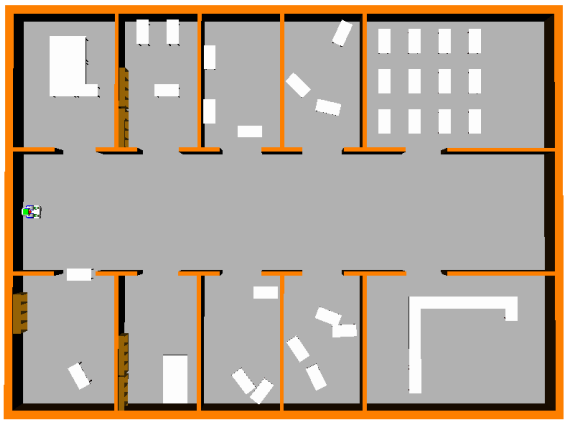


Fig. 2. Office environment with doors, tables and shelves

D. Path planning

To control the path of the robot, we design two practical ways. One is manually giving the input value of the robot

Algorithm 5 Greedy best-first search

$AllNodes = Initialize(dist_{start} = \text{inf}, parent_{start} = \text{None}, visited_{start} = \text{False})$
$StartNode = Initialize(dist_{start} = 0, parent_{start} = \text{None}, visited_{start} = \text{True})$
$VisitQueue = StartNode$
while $VisitQueue \neq$ empty and $CurrNode \neq Goal$ **do**
    $CurrNode = Dequeue(VisitQueue, Fscore)$
    $Visited_{CurrNode} = \text{True}$
    for each nbr in $NotVisited(Adjacent(CurrNode))$ **do**
        $Enqueue(nbr \text{ to } VisitQueue$
        if $dist_{nbr} > dist_{CurrNode} + distance(nbr, CurrNode)$ **then**
            $parent_{nbr} = CurrNode$
            $dist_{nbr} = dist_{CurrNode} + distance(nbr, CurrNode)$
            $Fscore = LineDistance(nbr, CurrNode)$
        end if
    end for
end while

each time step. The other one is giving a target and the robot searching for a path based on the greedy algorithm.

1) Manually: To directly giving the input value for robots, we design a series of input value in the beginning, then collect them together in an array. Each time step the robot moves, we only select one piece of the input from the array in the time sequence. This method is simple but requires lots of human work in designing a trajectory, especially when we want the robot to arrive a specific goal in a complicated environment.

2) Greedy algorithm: To make up the shortage of manually designing the path, we implement the greedy algorithm for path planning. The structure of the greedy algorithm is very similar to the A*. The only difference is the F-score. The F-score is equal to the heuristic (H-score) in Greedy algorithm while it is equal to the sum of the distance between the current node and the start node (G-score) and the H-score in A*.

IV. Experiment Results

A. Noise visualization

In Fig. 3, a linear dynamic robot hangs out in an obstacle-free ground. The black line is the target path, the blue line is the actual path for robot. From the figure, after several iterations, robot would deviate from the designed path significantly in open-loop control, due to the dynamic function noise.

In Fig. 3, the red points are the sensor observations (GPS sensor is used here). Fig. 4 shows the error between observation state and actual state $\|x_{observe} - x_{actual}\|^2$ and the average error of the scenario in Fig. 3. From the figure, the error is ranging from 0 to 2.0

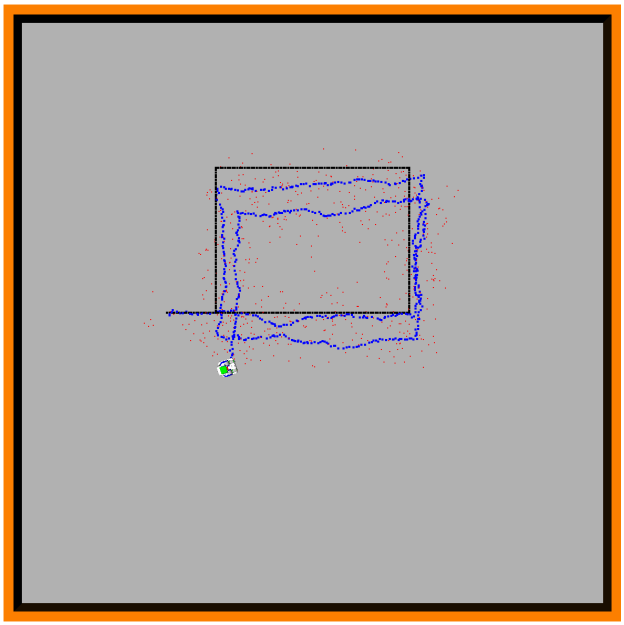


Fig. 3. Path of the PR2 robot moving on an obstacle-free ground. Red points are the GPS sensor's measurement.

B. Comparison about two algorithms

We design several scenarios to compare these two estimation algorithms, they are shown in Table I.

In this subsection, Scenario 1 is chosen, in which the robot dynamics is nonlinear, the sensor is IMU-enabled GPS, the map is the open space with obstacles. And we manually set a trajectory. In addition to the dynamic function, there is also a local obstacles-avoidance policy: the robot will turn a little

TABLE I
Scenarios used in this paper

Scenario Index	Scenarios' configurations		
	Robot Dynamics	***Sensors***	***Maps***
Scenario 1	Nonlinear	IMU-enabled GPS	obstacles
Scenario 2	Nonlinear	one Landmark	obstacles
Scenario 3	Nonlinear	six Landmark	obstacles
Scenario 4	Linear	GPS	office

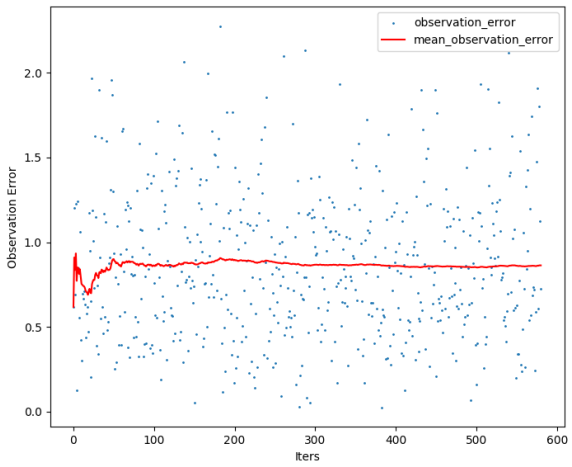


Fig. 4. Observation error

angle if it hits an obstacle. Because of this policy, the robot seems to hang out randomly in Fig. 5

From Fig. 6(a), the errors of the Particle filter and the EKF are in the same magnitude, and they are much less than the sensor error in Fig. 4. However, for the Particle filter, during the first few iterations, the error is very high. This is probability due to the limitation of particle number M. The algorithm could not sample a state near the actual state during some initialization, so it would take some time to converge. For EKF, there are some peaks during the process. We could also see some spiral line segments in Fig. 5(b). A more detailed discussion is in the Error Analysis section.

From Fig. 6(b), the computation time of the Particle filter is ten times higher than that of the EKF. Obviously, the Particle filter is more time consuming and the Kalman filter is more efficient.

C. Special case

However, there are some special cases where the Kalman filter fails. When the system is highly nonlinear, the EKF would become unstable – it could not estimate any external sensor data, e.g. data from laser or sonar sensors; Kalman filters are only available for Gaussain-distributed estimation, it is powerless for some multi-distribution task. The report chooses the highly non-linearity and goes into details.

In Scenario 2, the robot has nonlinear dynamics, sensor is single-landmark sensor, and the map is the open space with obstacles. From Eq. 15, the model for landmark is a nonlinear function, and would present highly nonlinear properties when $x_l - x_t \to 0$. From Fig. 7(a), during the first few iterations, the EKF works well. But when the robot comes near the point $x_t = x_l$ (the x axis is vertical for this figure), the algorithm would give some unreasonable estimation. From Fig. 8(a), the error grows explosively after 400 iterations. But from Fig. 7(b) and Fig. 8(a), particle filter performs much more stable, since it is able to tackle any kind of dynamic function and sensor function.

In Fig. 7(c), with five more landmark sensors are added, the EKF can also perform the localization task well. Since six landmarks are in different positions, there is hardly a highly nonlinear state for all of them, thus the EKF can solve the problem.

D. Error analysis

From previous sections, we find that for the EKF, there exists some spiral line segments in Fig.5(a). Fig.5 is calculated in Scenario 1 with the EKF, the blue point is prediction error and the red line is one of the input signal: angular velocity. From the figure, the beginning of the peak is coincide with the step of the input signal, which means the growing error might be caused by the step of the input.

Since the nonlinear dynamics only represents a one-dimensional system (displacement and velocity) in this report, it would be sensitive to some higher order signal (acceleration). As for the step angular velocity, the angular acceleration would be infinite at the step point. So the EKF would be unstable around such a point. However, this phenomenon wouldn't always happen because of the randomness of the system and the stability of the EKF to a certain extent.

E. Localization and Navigation

In this section, Scenario 4 is designed to display the strength for estimation algorithm on robot localization and navigation. The EKF is used in this scenario. Different from previous scenarios, the trajectory is generated using greedy algorithm based on the target location, rather than designed manually. Additionally, PID control is implemented to guide the robot moving through the target trajectory. Results are shown in Fig. 10, the robot is localized and navigated well, and can even pass some narrow passages.

V. Summary and Future Work

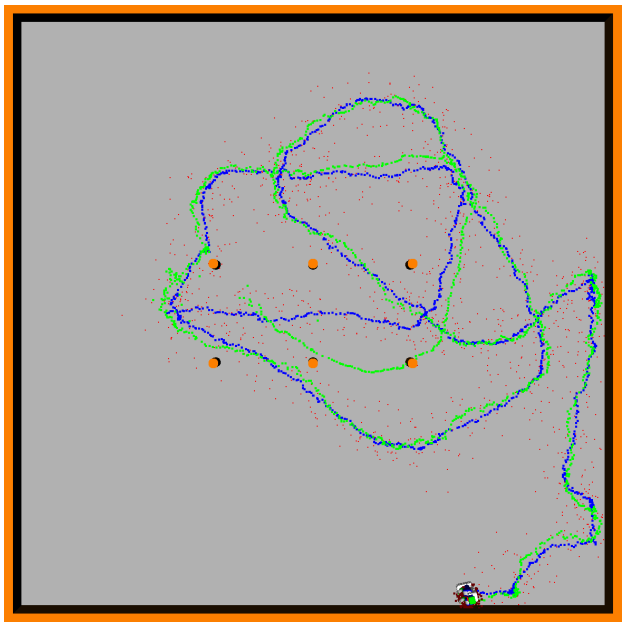
From the experiment results above, we can get the following conclusions:

1. The Extended Kalman Filter (EKF) performs a similar prediction accuracy with the Particle filter when the robot moves in an open space with a few obstacles, and the dynamics will never be highly nonlinear. The EKF is more efficient than the Particle filter. A serious drawback of the EKF is that it may fail when the dynamics is highly nonlinear or the environment is very complicated with many obstacles. Increasing the number of sensors can be a method to solve this problem.

2. The Particle filter usually costs more computation time than the EKF. But the Particle filter is more robust to the non-linearity and keeps a high accuracy even if the environment becomes more complicated, such as in a office environment.

3. Path planning algorithms such as the Greedy algorithm and PID control are successfully applied in the context of localization, which gives us more possibilities to the research on target localization.

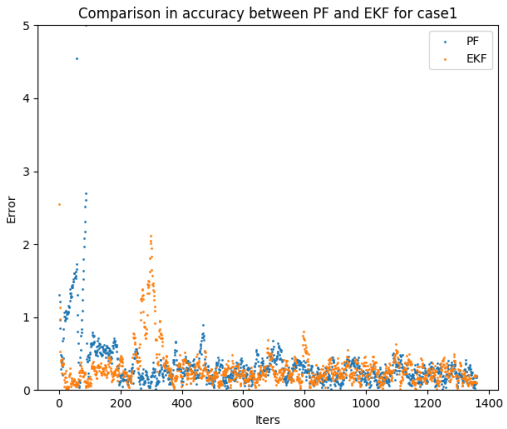
Considering the future work, there are several things can be improved further:



(a) Particle Filter Localization on Scenario 1

(b) Extended Kalman Filter Localization on Scenario 1

Fig. 5. (a), (b) Nonlinear dynamic robot localization in open space with obstacles, using IMU-enabled sensor. The estimation algorithms are Particle Filter for (a) and Extended Kalman Filter for (b). In the figure, the blue line is the actual path, the green line is the estimated path, the red points are sensor output.



(a) Accuracy for two algorithm on Scenario 1

(b) Computational time for two algorithm on Scenario 1

Fig. 6. comparison between two algorithm on (a) accuracy and (b) efficiency. (a) the error is the Euclidean distance between actual position and predicted position per iteration,(b) the computation time is the time consuming per iteration.

1. Investigate methods to improve the robustness for the Kalman filter. Though the Particle filter can solve many highly nonlinear problem, it costs huge amount of time. For tasks need fast computation speed, such as self-driving cars, improving the robustness for the Kalman filter is very significant.

2. Combine different sensors for localization. In this project we use different types of sensors to get measurement data separately. We can just combine several of them together and test the performance based on the combined sensor.

3. Use Simultaneous Localization And Mapping (SLAM) to build the map of the environment and estimate the location of the robot. [11]

REFERENCES

[1] Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., ... & Corke, P. (2018). The limits and potentials of deep learning for robotics. The International Journal of Robotics Research, 37(4-5), 405-420.

[2] Sui, Z., Xiang, L., Jenkins, O. C., & Desingh, K. (2017). Goal-directed robot manipulation through axiomatic scene estimation. The International Journal of Robotics Research, 36(1), 86-104.

[3] Chen, X., Chen, R., Sui, Z., Ye, Z., Liu, Y., Bahar, R., & Jenkins, O. C. (2019). Grip: Generative robust inference and perception for semantic robot manipulation in adversarial environments. arXiv preprint arXiv:1903.08352.

[4] Del Moral, Pierre (1996). Non Linear Filtering: Interacting Particle Solution. Markov Processes and Related Fields. 2 (4): 555–580.

[5] Del Moral, Pierre (1998). Measure Valued Processes and Interacting Particle Systems. Application to Non Linear Filtering Problems. Annals of Applied Probability (Publications du Laboratoire de Statistique et Probabilités, 96-15 (1996) ed.). 8 (2): 438–495.

[6] Del Moral, Pierre (2004). Feynman-Kac formulae. Genealogical and interacting particle approximations. https://www.springer.com/gp/book/9780387202686: Springer. Series: Probability and Applications. p. 556. ISBN 978-0-387-20268-6.

[7] Ribeiro, M. I. (2004). Kalman and extended kalman filters: Concept, derivation and properties. Institute for Systems and Robotics, 43.

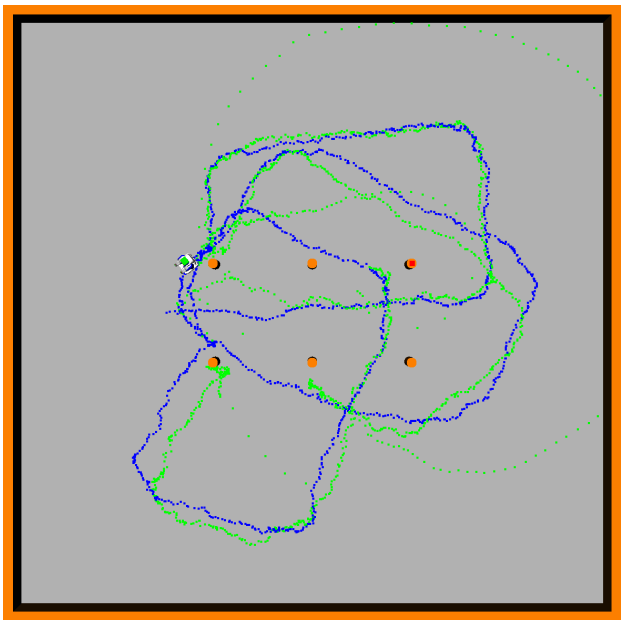
[8] Gutmann, J., Burgard, W., Fox, D., & Konolige, K. (1998). An experimental comparison of localization methods. Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190), 2, 736-743 vol.2.

[9] Fox D., Thrun S., Burgard W., Dellaert F. (2001) Particle Filters for Mobile Robot Localization. In: Doucet A., de Freitas N., Gordon N. (eds) Sequential Monte Carlo Methods in Practice. Statistics for Engineering and Information Science. Springer, New York, NY. https://doi.org/10.1007/978-1-4757-3437-9

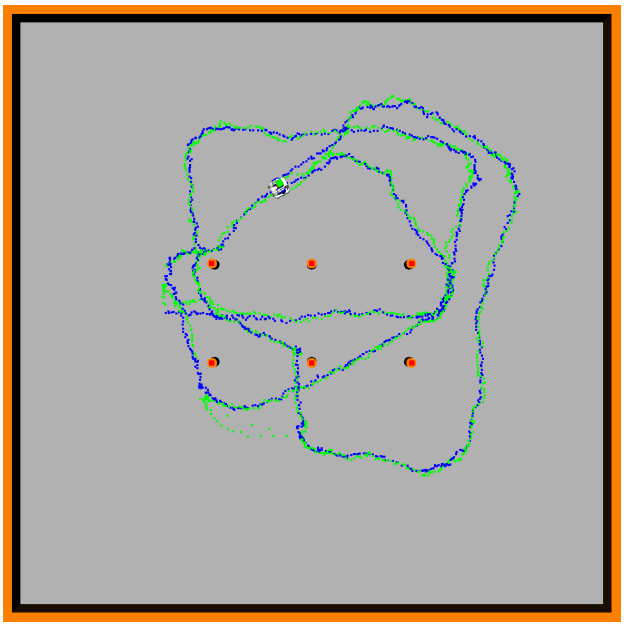
[10] Fakharian, A., Gustafsson, T., & Mehrfam, M. (2011, April). Adaptive Kalman filtering based navigation: An IMU/GPS integration approach. In 2011 International Conference on Networking, Sensing and Control (pp. 181-185). IEEE.

[11] Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. IEEE robotics automation magazine, 13(3), 108-117.

(a) Particle Filter Localization on Scenario 2

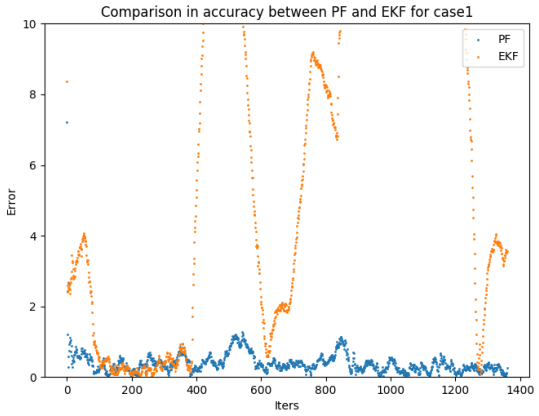


(b) Extended Kalman Filter Localization on Scenario 2

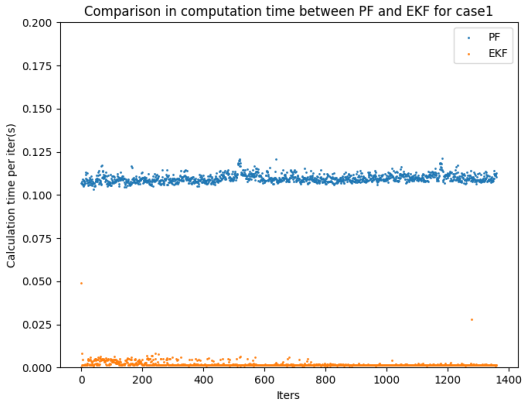


(c) Extended Kalman Filter Localization on Scenario 3

Fig. 7. (a), (b) Nonlinear dynamic robot localization in open space with obstacles, using one landmark sensor. The estimation algorithms are the Particle filter for (a) and the EKF for (b). (c) Nonlinear dynamics robot localization in an open space with obstacles, using the six-landmark sensor, estimation algorithm is the EKF. In the figure, the blue line is the actual path, the green line is the estimated path, there is a red point on the top-right obstacles(represent the land mark position).



(a) Accuracy for two algorithms on Scenario 2



(b) Computational time for two algorithms on Scenario 2

Fig. 8. comparison between two algorithms on (a) accuracy and (b) efficiency. (a) the error is the Euclidean distance between actual position and predicted position per iteration, (b) the computation time is the time consuming per iteration.

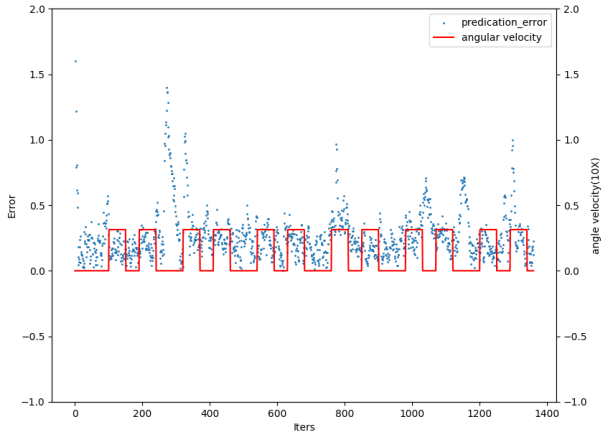
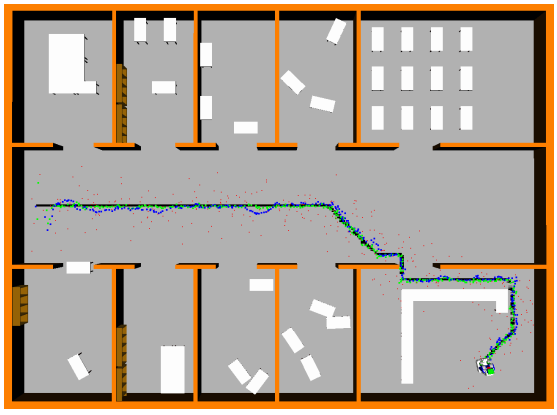


Fig. 9. EKF error analysis



(a) Target 1

(b) Target 2

Fig. 10. Robot localization and navigation on Scenario 4. The robot has linear dynamics with GPS sensor. The test is performed on the office map and the EKF is chosen for localization. The red point is the sensor observation, the blue point is the actual trajectory, the black point is the target trajectory and the green point is the predicted trajectory. PID is applied between the target trajectory and predicted trajectory.