Localization

Chien-lung Chou, Wan-Yi Yu

Prof. Dmitry Berenson

15 December 2021

1 Introduction

Robots and self-driving cars, unlike human beings, don't have five senses, e.g., sight, hearing, smell, taste, and touch, so they need to explore the world through a variety of sensors (e.g., LiDAR, Radar, Camera, and Ultrasound). Localization is the core task because before acting a corresponding reaction or making a control decision in the unknown environment, understanding where they are, surrounding information, and relative position with others in the world are the most critical issues.

However, localization is never a simple problem because it is impossible to predict the robot's actual position; people can only predict the state in which the robot would show up with a higher probability. Moreover, the sensors hardly collect and estimate all the information in the environment around the robot, so the sensor measurements are never a hundred percent accurate. Take Global Positioning System (GPS) as an example; although GPS is the most common localization method, the error sometimes would be up to two meters. In some shielding environments, the beacon may lose connection with the satellite at specific places.

In order to overcome the problems, it is necessary to combine other sensors to achieve the correction and apply different algorithms to compensate for the errors and compute a probabilistic estimation of the robot's state at a given time and update it based on the actions taken and sensor measurements perceived.

1.1 Application of localization

Localization is applied in various fields, such as self-driving cars, uncrewed aerial vehicles, robot vacuum cleaners, to plan and decide which direction. Additionally, robot-assisted surgery is also involved with localization and developed to overcome human limitations and eliminate impediments associated with conventional surgery. The robotic technology could assist minimally invasive procedures because autonomous probes or utility could navigate outside or inside the patient's body to the corresponding position, which could increase the operation precision.

1.2 Filtering Technique

To approach better localization, there are several kinds of filtering algorithms that can be applied to estimate the state of a robot probabilistically. For example, *Kalman Filter (KF), Extended Kalman Filter (EKF), and Particle Filter (PF)*.

KF works by a two-phase process: (i) prediction and (ii) update. KF first produces an estimate of the current state, along with the uncertainty. Once the outcome of the next measurement is observed, this estimate is updated using a weighted average, with more weight given to estimates with greater certainty. The advantage of this algorithm is it can operate in real-time, using only the present measurements and the state calculated previously and its uncertainty matrix. However, KF could not model nonlinear systems.

EKF is developed to deal with nonlinear systems by computing the Jacobian. However, since EKF approximates the motion model and sensor model with linear systems, if the system is highly nonlinear, then the approximation easily goes wrong, and the prediction diverges. On the other hand, KF and EKF have other limitations. The error of state and measurement must follow Gaussian distribution.

Unlike KF and EKF, PF can handle linear or nonlinear systems and both discrete or continuous distribution with a set of particles. The more particles spread, the better approximation would be generated, but require more time to compute and produce the path.

This project uses a linear system to execute the robot, so we choose KF and PF as our filtering algorithms to localize it along the predefined path. We have pushed our implementation on Github (https://github.com/chien-lung/localization).

2 Implementation

2.1 Models

To compare KF and PF fairly, we devise a path and an environment and design a linear system for the motion model and sensor model.

The motion model is described as follows,

$$x_t = x_{t-1} + v_t cos(\theta_{t-1})$$

$$y_t = y_{t-1} + v_t sin(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} + \delta_t$$

x is located in the x-axis, y is located in the y-axis, and θ is the heading angle of the robot.

v is velocity applied to the robot, and δ is the steering angle applied to the robot.

The sensor model we use is the Global Positioning System (GPS) which returns the location of the robot in x and y coordinates.

We formulate these two models with additional noises (i.e., motion noise and sensor noise) in a matrix form:

State:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Control:

$$\boldsymbol{u} = \begin{bmatrix} v \\ \delta \end{bmatrix}$$

Motion model:

$$\boldsymbol{x}_t = A_t \boldsymbol{x}_{t-1} + B_t \boldsymbol{u}_t + \boldsymbol{\epsilon}_t,$$

where

$$A_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B_t = \begin{bmatrix} cos(\theta_{t-1}) & 0 \\ sin(\theta_{t-1}) & 0 \\ 0 & 1 \end{bmatrix}$$

$\epsilon_t =$ motion noise at time t .

Sensor model:

$$z_t = C_t x_t + w_t,$$

where

$$C_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$w_t =$ sensor noise at time t .

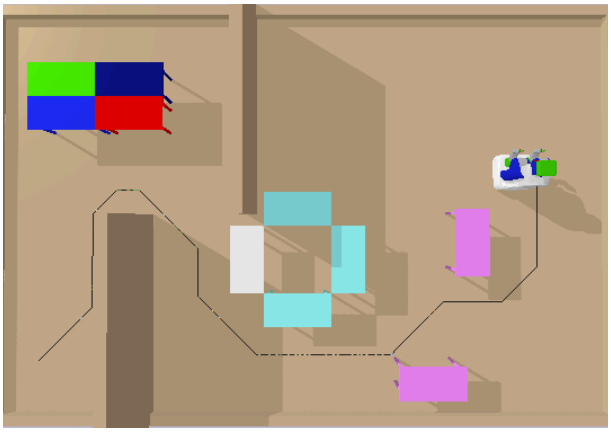
2.2 Pipeline

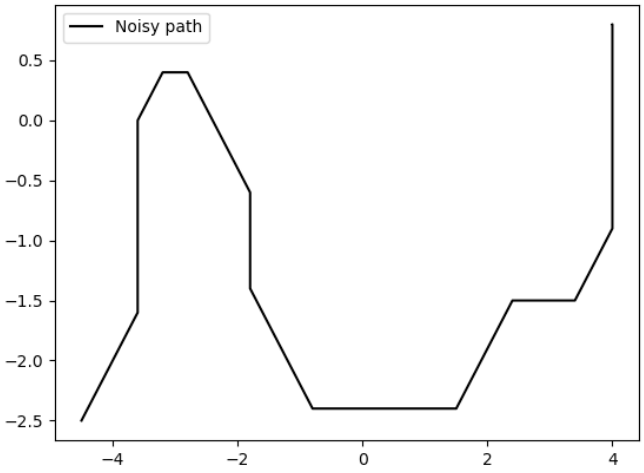
The pipeline of our project has three steps:

1. Environment setup and path generation
2. Noise and control generation
3. Filtering algorithm execution

2.2.1 Environment setup and path generation

We design our environment with several objects to let the robot easily collide with obstacles if the prediction is wrong. Then, we generate a path including the motion noises in this environment. The environment and the path are shown below. Also, to visualize them easily, we use matplotlib to show them on the *xy*-plane.





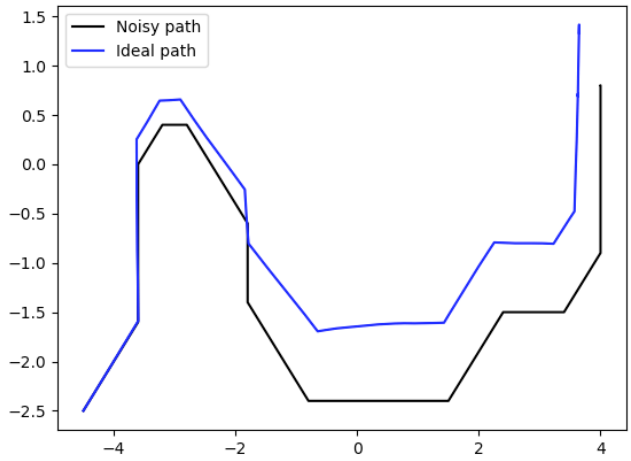
2.2.2 Noise and control generation

We first generate motion noise for each state as $\boldsymbol{\epsilon}_t$. Since we assume our path includes the motion noise, we can obtain the control of each state according to the following equation.

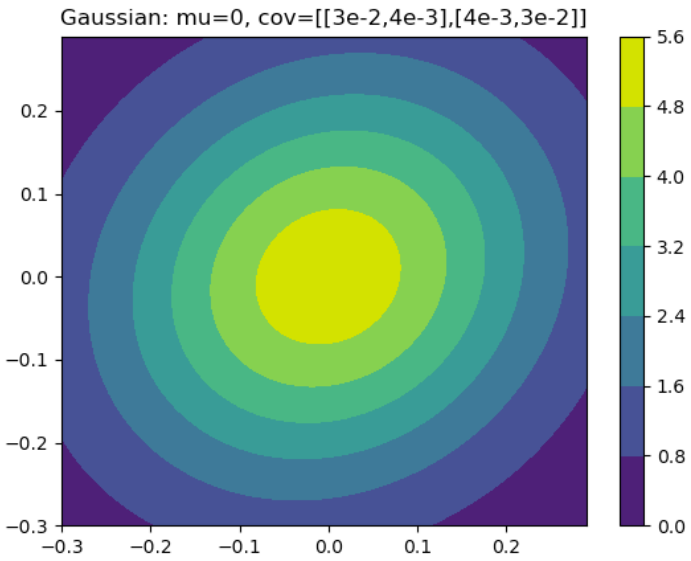
$$\boldsymbol{u}_t = B_t^{\dagger}(\boldsymbol{x}_t - A_t\boldsymbol{x}_{t-1} - \boldsymbol{\epsilon}_t),$$

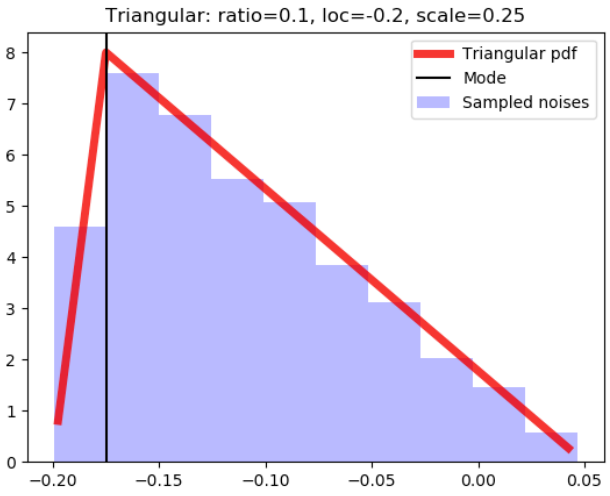
where $B_t^{\dagger}$ is pseudo-inverse of B_t

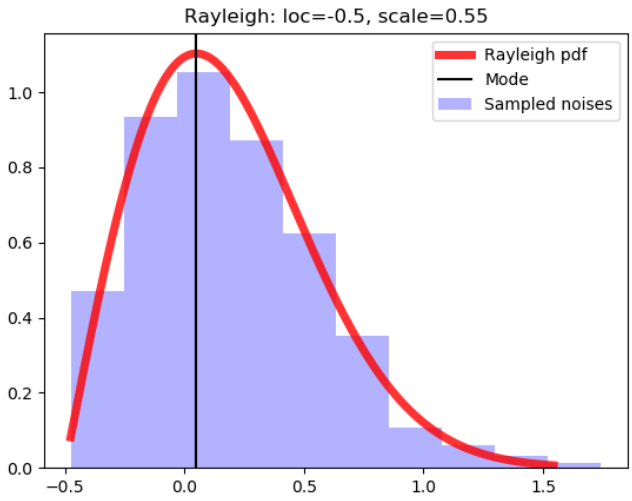
With control at each time step, we can generate the path without noise (i.e., ideal path), as shown below. Note that this ideal path doesn't get involved in our project.



Then, we generate sensor noise for each time step. In this project, we use Gaussian distribution, triangular distribution, and Rayleigh distribution as our sensor noise distribution, and each distribution is shown below.

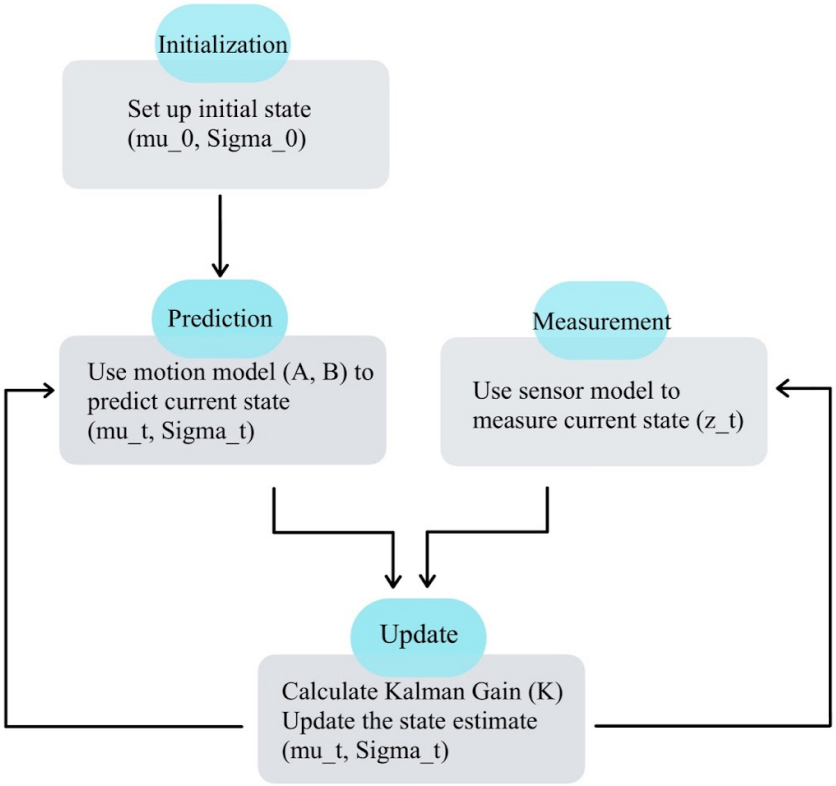




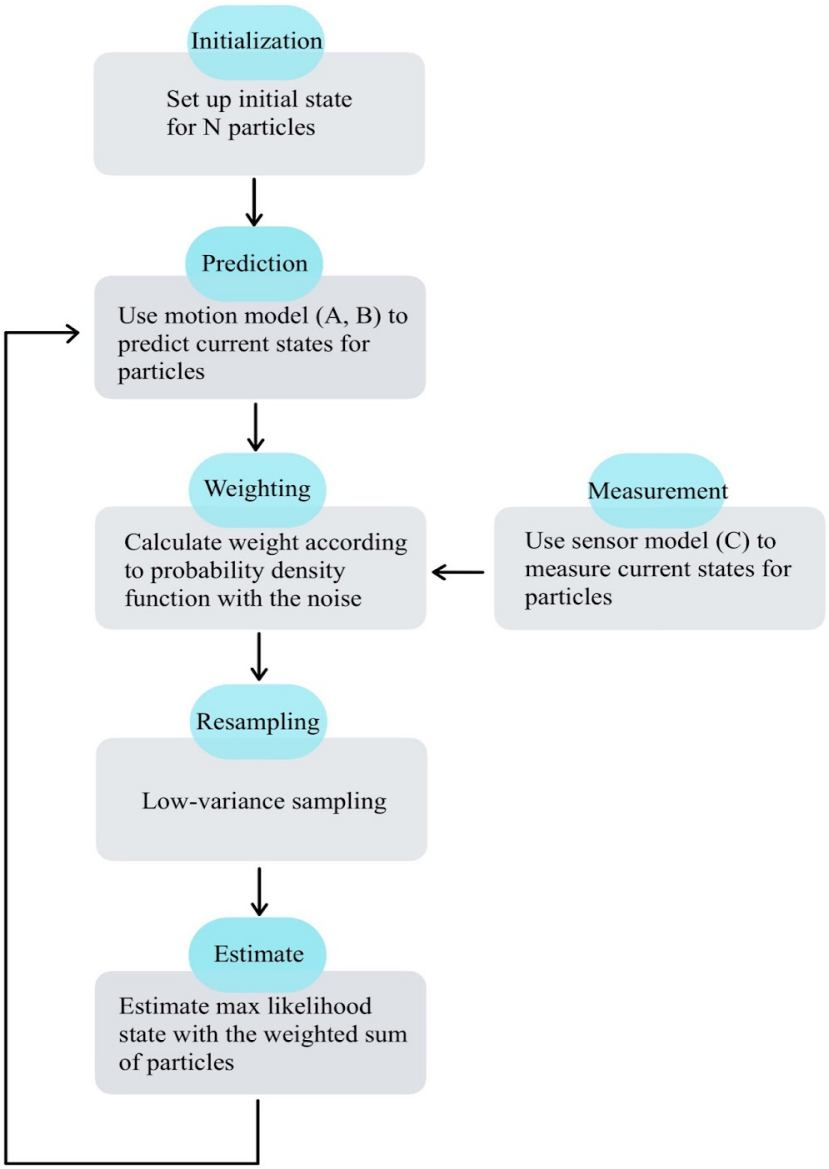


2.2.3 Filtering algorithm execution

With the given path, environment, controls, and measurements, we can run the filtering algorithms to localize the PR2 robot. Specifically, at each time step, we have the previous state and the control so that we can estimate the current state with the motion model. Then, we can use the measurement to update our estimated state to avoid diverging. The block diagram of each algorithm is shown below.



KF block diagram



PF block diagram

2.3 Evaluation metric

We use the sum of squared errors as our evaluation metric, as shown below.

$$Error = \sum_{t} \sqrt{(x_t - \hat{x}_t)^2 + (y_t - \hat{y}_t)^2},$$

where x_t and y_t are the real location, and $\hat{x}_t$ and $\hat{y}_t$ are the predicted location.

3 Results

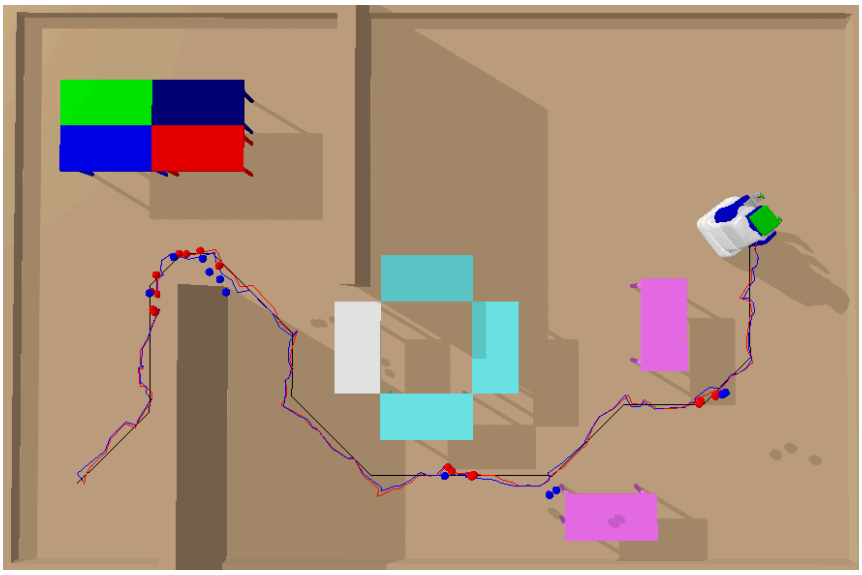
To demonstrate each algorithm's advantages and disadvantages, we use Gaussian noise (case 1) and non-Gaussian noise (case 2) as our sensor noise to compare the performance between KF and PF. Additionally, we demonstrate the tradeoff between execution time and error for different update frequencies of KF (case 3). And we also show the tradeoff between execution time and error for the number of particles used in PF (case 4). In the following cases, the blue color line and sphere represent KF's results, and the color red represents PF's results.

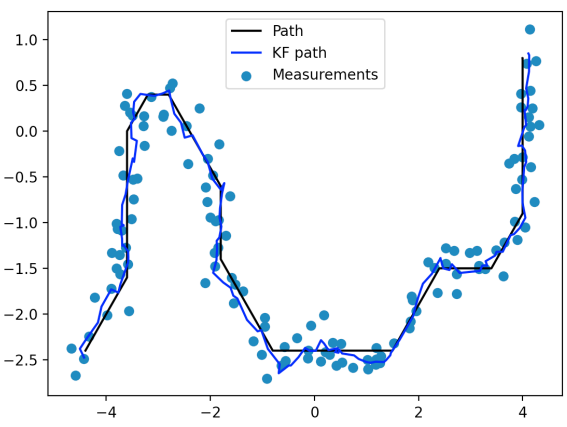
3.1 case 1

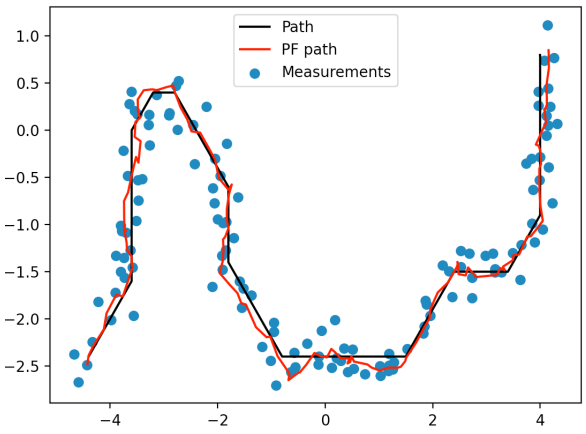
We first generate the path and measurements using the motion noise and sensor noise that both are Gaussian distributions with zero mean. The noise covariance matrix in the filter model is derived by ourselves included in the table. We set the number of particles to 100 in PF. In this case, we simulate the robot in both a 2D and 3D environment. Then, we compare the execution time and the error of KF and PF. The execution time is when the localization algorithm starts to estimate until it gets the reasonably estimated path. The error in the table represents the norm distance from the measurements to the path. In the 3D environment, we create spheres at the place where the robot collides during the movement.

- R = Motion noise, Q = Sensor Noise

	Motion model	Sensor model	Filter model	Exec. time (second)	Error
KF	$mean = 0$ $cov = \begin{bmatrix} 3e-3 & 1e-3 & 0 \\ 1e-3 & 3e-3 & 0 \\ 0 & 0 & 1e-4 \end{bmatrix}$	$mean = 0$ $cov = \begin{bmatrix} 3e-2 & 4e-3 \\ 4e-3 & 3e-2 \end{bmatrix}$	$R = \begin{bmatrix} 1e-2 & 1e-4 & 0 \\ 1e-4 & 1e-2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $Q = \begin{bmatrix} 8e-2 & 1e-3 \\ 1e-3 & 8e-2 \end{bmatrix}$	0.03	13.7
PF	$mean = 0$ $cov = \begin{bmatrix} 3e-3 & 1e-3 & 0 \\ 1e-3 & 3e-3 & 0 \\ 0 & 0 & 1e-4 \end{bmatrix}$	$mean = 0$ $cov = \begin{bmatrix} 3e-2 & 4e-3 \\ 4e-3 & 3e-2 \end{bmatrix}$	$R = \begin{bmatrix} 1e-2 & 1e-4 & 0 \\ 1e-4 & 1e-2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $Q = \begin{bmatrix} 8e-2 & 1e-3 \\ 1e-3 & 8e-2 \end{bmatrix}$	5.8	13.63







We find out that the error in both filters is about the same, but the PF's execution time is 200 times the same as the execution time of the KF. Therefore, KF has better performance in this case since KF can complete the task more efficiently and generate the same result as the PF.

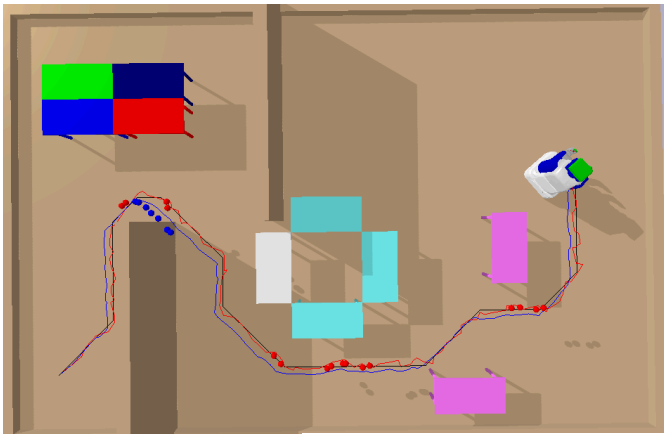
3.2 Case 2

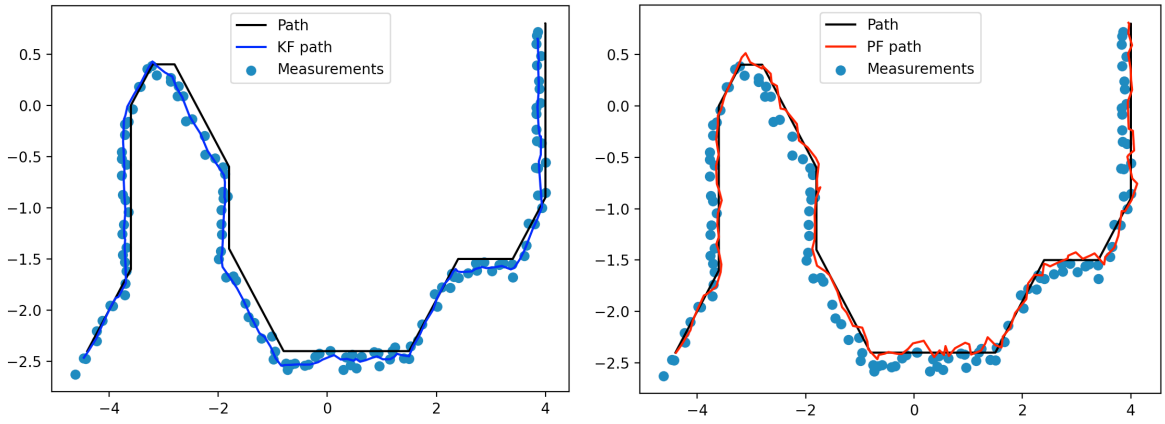
In case 2, the motion noise from Gaussian distribution is the same as the one in case 1, and we also compare the execution time and the error between KF and PF and the collision situation of the PR2. But the sensor noise is sampled from different distributions. In this case, we apply two distributions individually: triangular distribution and Rayleigh distribution. Most of the settings in KF and PF are the same as the one in case one, but in PF, the weighting function is changed to the same distribution as the sensor noise.

- R = Motion noise, Q = Sensor Noise

		Motion model	Sensor model	Filter model	Exec. time (second)	Error
Triangular	KF	$mean = 0$ $cov = \begin{bmatrix} 3e-3 & 1e-3 & 0 \\ 1e-3 & 3e-3 & 0 \\ 0 & 0 & 1e-4 \end{bmatrix}$	$mean = 0$ $cov = \begin{bmatrix} 3e-2 & 4e-3 \\ 4e-3 & 3e-2 \end{bmatrix}$	$R = \begin{bmatrix} 1e-2 & 1e-4 & 0 \\ 1e-4 & 1e-2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $Q = \begin{bmatrix} 8e-2 & 1e-3 \\ 1e-3 & 8e-2 \end{bmatrix}$	0.03	21.27
	PF	$mean = 0$ $cov = \begin{bmatrix} 3e-3 & 1e-3 & 0 \\ 1e-3 & 3e-3 & 0 \\ 0 & 0 & 1e-4 \end{bmatrix}$	ratio = 0.1 loc = -0.2 scale = 0.25	$R = \begin{bmatrix} 1e-2 & 1e-4 & 0 \\ 1e-4 & 1e-2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ Q : ratio = 0.1 loc = -0.2 scale = 0.25	10.36	7.92
Rayleigh	KF	$mean = 0$ $cov = \begin{bmatrix} 3e-3 & 1e-3 & 0 \\ 1e-3 & 3e-3 & 0 \\ 0 & 0 & 1e-4 \end{bmatrix}$	$mean = 0$ $cov = \begin{bmatrix} 3e-2 & 4e-3 \\ 4e-3 & 3e-2 \end{bmatrix}$	$R = \begin{bmatrix} 1e-2 & 1e-4 & 0 \\ 1e-4 & 1e-2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $Q = \begin{bmatrix} 8e-2 & 1e-3 \\ 1e-3 & 8e-2 \end{bmatrix}$	0.035	37.35
	PF	$mean = 0$ $cov = \begin{bmatrix} 3e-3 & 1e-3 & 0 \\ 1e-3 & 3e-3 & 0 \\ 0 & 0 & 1e-4 \end{bmatrix}$	loc = -0.5 scale = 0.55	$R = \begin{bmatrix} 1e-2 & 1e-4 & 0 \\ 1e-4 & 1e-2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ Q : loc = -0.5 scale = 0.55	7.642	19.85

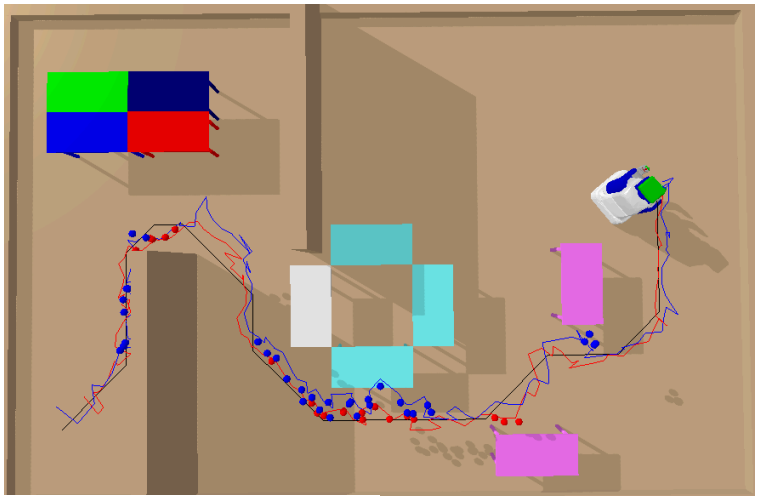
1. Triangular distribution

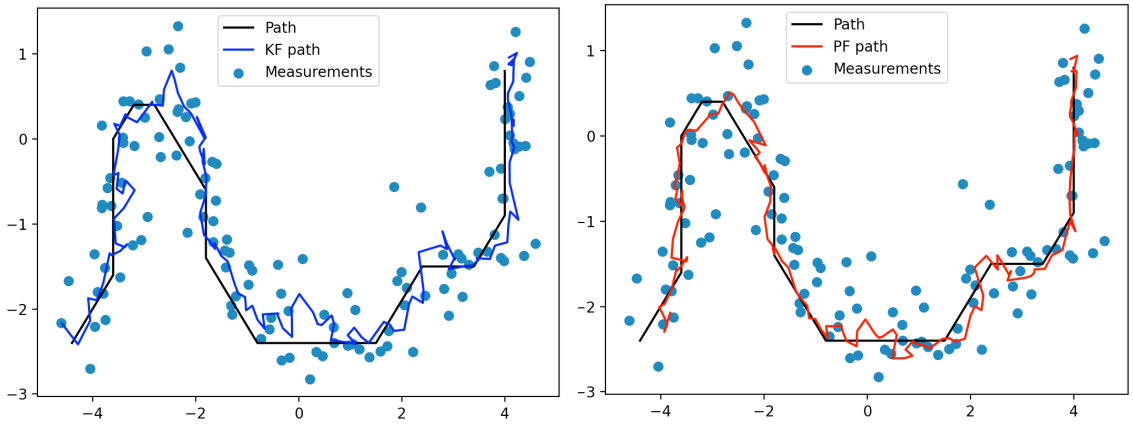




From the triangular distribution, although the execution time of the PF is 300 times more than KF, the error of the PF is three times less than the error of the KF. Therefore, applying PF can localize better for the noise generated from triangular distribution. The result also shows that PF achieves excellent results in both Gaussian and non-Gaussian distribution.

2. Rayleigh distribution





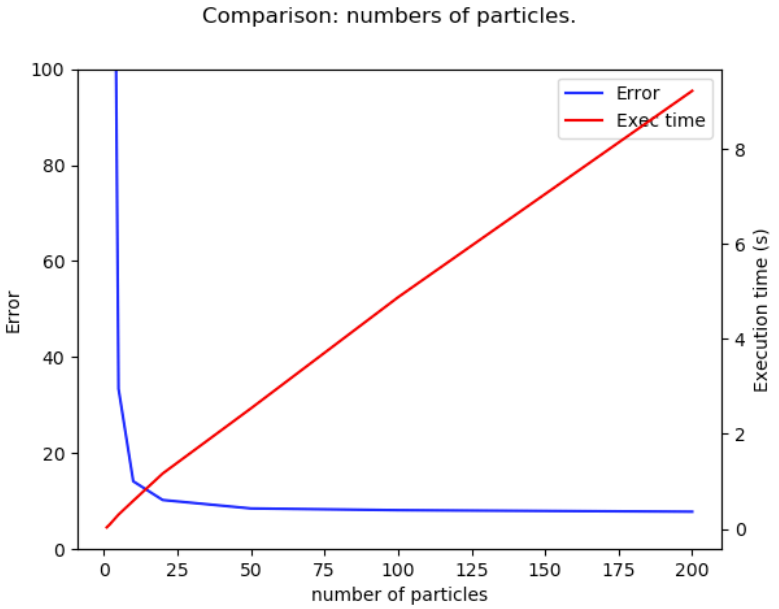
In this case, we apply Rayleigh distribution to set the sensor noise. We observe that although the execution time of KF is 200 times faster than the execution time of PF, the error produced by KF is almost twice as much as the error made by PF. Additionally, the number of collisions in PF is much less than KF. Therefore, PF produces a better path than KF.

After completing this case, we can find out that PF produces less error with the sensor noise from different distributions. Thus, if we are going to process localization with non-Gaussian distributed noises and expect showing a lower error, applying Particle Filter will be a better choice.

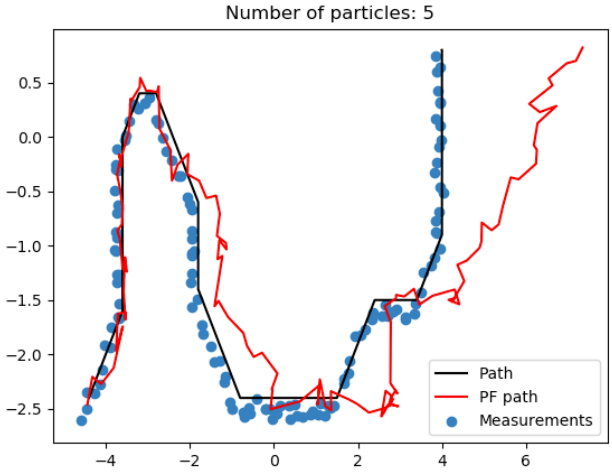
3.3 Case 3

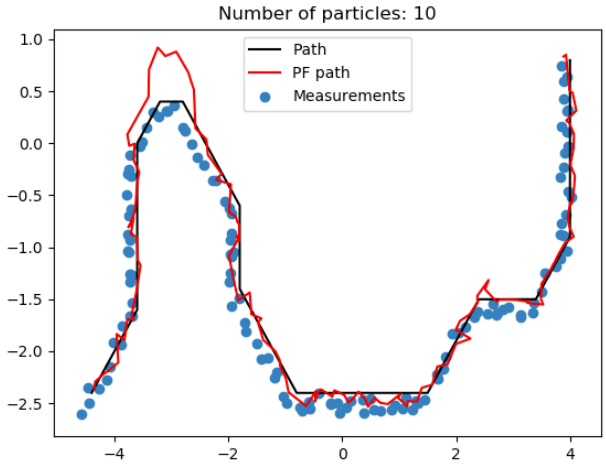
In this case, we use triangular distribution to generate the sensor noise and simulate PF with different numbers of particles. As the experimental results show, the error is very high when using less than ten particles. However, as the number of particles increases, the error decreases sharply and converges to 7.5. This comparison shows that the number of particles is important to the error and it will converge to some value if there is no more information obtained. Also, as the number of particles increases, the execution time grows linearly, making this algorithm hard to run in real-time.

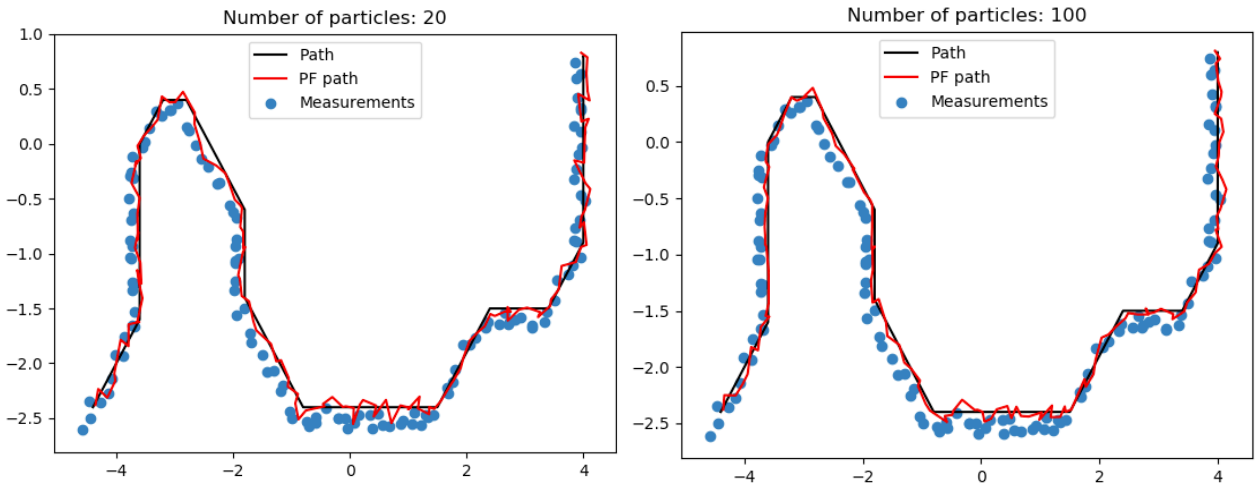
	Number of particles	Execution time (s)	Error
PF	1	0.027	378.54
	2	0.093	111.16
	5	0.331	108.53
	10	0.667	14.21
	20	1.289	10.03
	50	2.069	8.59
	100	3.912	7.96
	200	7.588	7.6



To better understand the estimation, we also sample the estimated paths using different particles below. We find that PF with 5 particles can estimate the states well in the beginning, but when the particles cannot describe the real state well, the estimated path diverges largely.



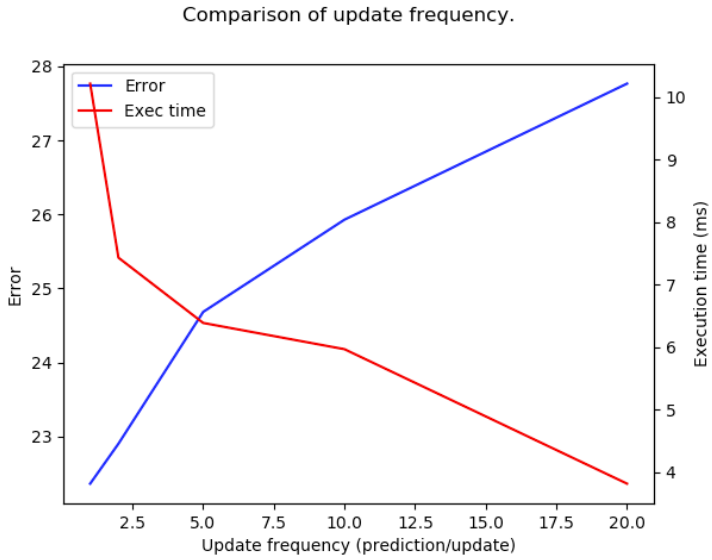




3.4 Case 4

We use different update frequencies of KF to show the tradeoff between execution time and error. As the update frequency increases, the error increases, but the execution time decreases. This is because the update step of KF takes some time to update the current state, which will increase the execution time.

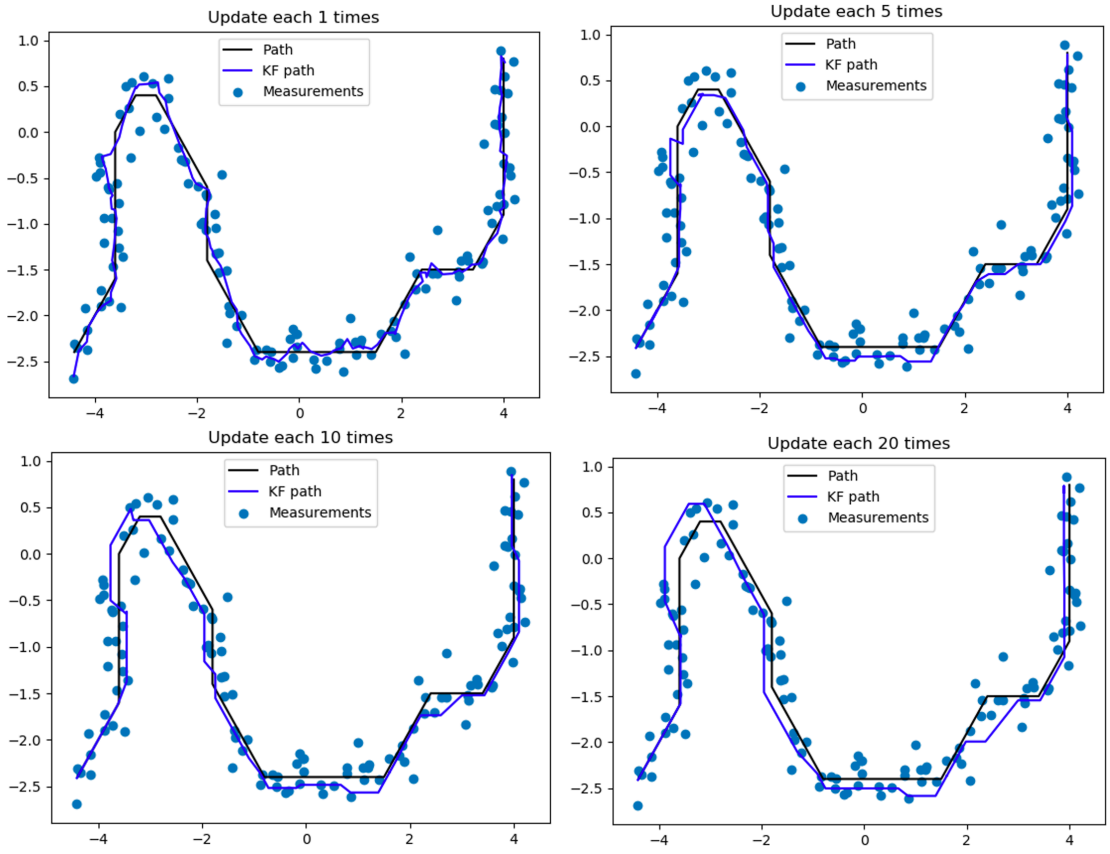
	Update frequency	Execution time (ms)	Error
KF	1	10.21	22.36
	2	7.43	22.91
	5	6.39	24.68
	10	5.97	25.93
	20	3.82	27.76



According to the experimental results, we can find that the algorithm runs faster by discarding some measurements although this will inject more error into our estimation. Also, this shows that if there is no measurement, we can still estimate the state other than waiting for the measurement to update.

We sample the estimated path of KF with different update frequencies. The estimated path whose update frequency is 20 deviates from the real path. However, the estimated path whose update frequency is 5 is still close to the real path.

This fact can be beneficial to some applications; for example, if the robot moves in a high speed, but the measurement can not be obtained on time, KF can still handle this kind of situation, instead of letting the estimated path diverging.



4 Conclusion

In this project, we implement Kalman Filter and Particle Filter to estimate the location of a PR2 robot by the design motion and sensor model in the pybullet environment. We also apply different noise distributions to compare the localization performance between KF and PF. Furthermore, we demonstrate the tradeoff between execution time and error for the number of particles used in PF.

The results show that Kalman Filter only works efficiently and precisely if the noise distribution is Gaussian distributed. Moreover, we find out that if the update frequency is less than some threshold, we can still obtain a reasonable path close to the actual path.

On the other hand, although Particle Filter is slow and does not come up with better results in the Gaussian noise case, PF is more able to deal with various distributions and can generate promising results mainly. Additionally, the more particles that PF spread, the higher precision would generate. However, the drawback of the PF is that it requires more time to create the path, and the cost of the calculation depends on the quantity of the particles it spreads. Therefore, it is an issue that deserves a sophisticated discussion to design the corresponding amount of particles to generate.

The code can be found on https://github.com/chien-lung/localization. We also write README.md to explain files and how they work.