Comparison between Kalman Filter and Particle Filter in Localization

Jiacheng Zhu
Mechanical Engineering
Email: jiachzhu@umich.edu

Xiaoke Wang
Electrical and Computer Engineering
Email: xiaoke@umich.edu

***Abstract*—In this paper, two of the most important localization algorithms are compared, Kalman Filter and Particle Filter, comparing their efficiency in tracking a moving robot in several complex environments and movement models. The accuracy of two algorithms is compared in terms of error distance as well as the rationality of results. This comparison is based on simulations in Openrave.**

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

I. Introduction

The mobile robot localization is a key problem in mobile robotics. It is known as estimating a robots pose according to its surrounding environment[16].

The mobile robot localization problem comes in many flavors. Position tracking[3, 21, 18, 20] is by far the simplest localization problem. In this problem, the initial position of the robot is given, and the problem is to eliminate the increment errors in robots movement, i.e. Odometry, in most cases.

The global localization[4] is a more challenging problem, in which the robot does not know its initial pose, and has to determine it from scratch.

The kidnapped robot problem is more difficult[9]. In this problem, a well-localized robot is teleported to a new position without being told. In this situation, the robot might still consider itself at somewhere at the time being teleported. The kidnapped robot problem is mostly used to test if a robot is able to recover from catastrophic localization failures.

Whats more, there are also multi-robot localization problem, where a team of robots seeks to localize themselves.

In order to solve these localization problems, a variety of algorithms are proposed[3]. Most of them address the Position tracking problem. The nature of small, incremental errors in the robotics makes algorithms such as Kalman filters [11, 12, 14, 19] applicable. Kalman filters estimate posterior distributions of robot poses according to sensor data, based on a list of restrictive assumptions such as Gaussian-distributed noise and also Gaussian-distributed initial uncertainty[16]. However, Kalman filter only deals with linear and Gaussian models, the Extanded Karman Filter is required for nonlinear systems. And due to the restrictive nature of belief representations, the Karman filter is inapplicable to global localization problems. [1]

Monte Carlo localization (MCL)[6, 10] is another novel mobile robot localization algorithm, and in the statistical literature, it is also known as Particle Filters [7, 8, 13, 15]. Particle Filter shares the same mathematical foundations with Karman filter, it computes approximate posteriors over robot according to sensor information [16]. Specifically, the Particle Filter represents the posteriors by a random collection of weighted particles which approximates the desired distribution [17]. The Particle Filter has a list of advantages,

1. Particle filter focus computational resources in areas that are most relevant, by sampling in proportion to the posterior likelihood.

2. Compared to other approaches, Particle Filters are universal density approximators, and does not rely on restrictive assumptions on the shape of the posterior density

Particle filters can be easily implemented as any-time algorithms[5, 22] by controlling the number of samples online. As a result, particle filters can adapt to the available computational resources.

II. Set up the Noisy Simulation Environment

In the real world, uncertainty is everywhere. There is no certain sensors nor motors. There is noise in the signals robot get from sensors. The robot can only know a possible location. So is the motor. If a robot walks along a direction, it will deviate the route. In our work, we make a simulation that a mobile robot moves in an environments with some obstacles. The robot tries to

reach a goal with a noisy sensor, which can roughly return the current position, and an unreliable motor.

A. Sensor

1) Sensor with Gaussian noise: On the mobile robot, there are different sensors to detect position, velocity, acceleration, etc. However, because of the existence of noise, the results are not accurate. In this simulation, due to the critical requirement of Kalman Filter, all the noise should be Gaussian. Taking a position sensor (e.g. GPS) as an example, the x direction and y direction are independent, so the result of sensor is just the true position value plus a Gaussian random number to each direction.

2) Sonar sensor: Sonar sensor is a more realistic sensor in real robotics. It emits pulses of sounds and listening for echoes, and then the robot can have a knowledge of whether there is obstacle and how far it is. The distance of of the obstacle can be computed by the time sounds takes to come back. In our simulation, A simple sonar is created. It will detect four direction just along the coordinate. The is a maximum detection distance and a detect step size. The step size is adjusted so that it will not go through collision. once the sonar detects a obstacle, it will return a distance, if there is no obstacle, then the sensor will return the maximum detection distance of that direction.

B. Move

1) Move Step by Step: The noise in moving is similar, which is a Gaussian noise. In this moving model, the robot wants to move to a desired position on each step according to a specific distance and direction. However, because of the noise, the robot reaches a neighbourhood around the desired position. Although this model is a two dimension simulation, the x direction and y direction are independent. So the noise is added independently to each direction. By moving step by step, the robot can check its position and corrects its direction to reach a goal. We are going to use this model to simulate a robot with a feedback to walk along a corridor and reach a final position.

$$x_{t+1} = xt + h_t + n$$

where $N = n \sim \mathcal{N}(0, \sigma^2)$.

2) Uniformly accelerated motion: This moving model is designed for Kalman filter, which works well in a continuous space with mutiple state variables. In this situation, the robot moves in uniform acceleration. So the next position depends on current location, velocity, acceleration, as well as noise.

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ v_t \\ a_t \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} \quad (1)$$

n_1, n_2, n_3 are Gaussian distributed random number $N_i = n_i \sim \mathcal{N}(0, \sigma_i^2)$

With this model, we can test the accuracy of Kalman filter under different situations.

C. Collision Checking

1) No Collision Checking: In the real world, collison checking gives some information on the position of a robot. However a simple robot with a noisy GPS position signal cannot do this. When the robot gets a signal to move forward by one step to a collison free space, it may move forward or stuck on half way because of the noisy movement. However, the robot doesn't know it is stucked. It does not have sensor to return this stucked state. This causes problem when we implement Kalman Filter or Particle Filter. When the robot is stucked, the filter doesn't know this and do the calculation assuming the robot moving forward. The filter makes wrong prediction with moving information and causes error estiomation in this step. The error of estimation will accumulate while the robot is stucked at some place. This can be seen clearly when a robot moving along a corridor.

2) Moving with Collision Checking Information: If a robot has a sensor to return a signal when it collides with an obstacle, this will gives more information of the position when a robot moves in an environment with obstacles. With collision checking, a robot can check before it takes a step. If there is an obstacle, the robot will retry until it reaches a collision free position. The simulation will come close to reality if each step becomes infinitely small.

III. Kalman Filter

A. Kalman Filter in One Step Moving

When Kalman Filter is applied in the model of moving step by step, it can be simplified to a superposition of two one dimensional models with one state varible, position. For example, the filter along x driection, the distribution at time t is assumed to be Gaussian with variance σ_t^2:

$$P(x_t) = \alpha e^{-\frac{1}{2}(\frac{(x_t - \mu_0)^2}{\sigma_t^2})} \quad (2)$$

The transition model adds a Gaussian perturbation of constant variance σ_x^2 to the current state:

$$P(x_{t+1}|x_t) = \alpha e^{-\frac{1}{2}(\frac{(x_{t+1}-(x_t+h_{x,t}))^2}{\sigma_0^2})} \tag{3}$$

where $h_{x,t}$ means the desired movement along x direction at time t.The sensor model assumes Gaussian noise with variance σ_z^2

$$P(z_t|x_t) = \alpha e^{-\frac{1}{2}(\frac{(x_t-x_t)^2}{\sigma_0^2})} \tag{4}$$

The one-step predicted distribution comes from (2) and (3),

$$P(x_t+1) = \int_{-infty}^{\infty} P(x_{t+1}|x_t)P(x_t)dx_t \tag{5}$$

and given the observation z_t, using maximum likelihood estimation (MLE) to estimate the position and variance of x_{t+1}. The result is similar to [1], but slightly different.

$$\begin{aligned} \mu_{t+1} &= \frac{(\sigma_t^2+\sigma_x^2)z_{t+1} + \sigma_z^2(\mu_t + h_{x,t})}{\sigma_t^2+\sigma_x^2+\sigma_z^2} \\ \sigma_{t+1}^2 &= \frac{(\sigma_t^2+\sigma_x^2)\sigma_z^2}{\sigma_t^2+\sigma_x^2+\sigma_z^2} \end{aligned} \tag{6}$$

B. Kalman Filter in Uniformly accelerated motion

In the general temporal model used with Kalman filtering,

$$\begin{aligned} P(x_{t+1}|x_t) &= N(Fx_t, \Sigma_x)(x_{t+1}) \\ P(z_t|x_t) &= N(Hx_t, \Sigma_z)(z_t) \end{aligned} \tag{7}$$

where F and Σ_x are matrices describing the linear transition model and transition noise co-variance, and H and Σ_z are the corresponding matrices for the sensor model.

$$\begin{aligned} \mu_{t+1} &= F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t) \\ \Sigma_{t+1} &= (I - K_{t+1}H)(F\Sigma_t F^T + \Sigma_x) \end{aligned} \tag{8}$$

where $K_{t+1} = (F\Sigma_t F^T + \Sigma_x)H^T(H(F\Sigma_t F^T + \Sigma_x)H^T + \Sigma_z)^1$

Given the uniformly accelerated motion model (1), we have

$$F = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \quad \Sigma_x = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix} \tag{9}$$

t is the time interval we choose in simulation. The smaller is the time interval, the closer is the simulation to reality.

If the sensor can return the current position, velocity and acceleration with a zero mean σ_z variance Gaussian disturbance, which means

$$Z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \Sigma_z = \sigma_z \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

If the sensor is just a position sensor, like ultrasonic range finder, then

$$Z = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad \Sigma_z = \sigma_z \tag{11}$$

Comparing these two kinds of sensor, therotically, the multipurpose sensor can give more information of current state, so Kalman Filter can give a more accurate result.

Disturbance may influence the result.

IV. Particle Filter

A. Algorithm

In order to deal with the non-Gaussian and nonlinear problems. The particle filter are formulated on the concepts of the Bayesian theory and the sequential importance-sampling[23]. The Particle Filter approximates recursively the posterior distribution by a sets of weighted samples. So that the the required posterior density function can be represented by a set of random samples and weights and to compute estimates based on these samples and weights. Then the estimated state of the system can be computed based on the particles and weights. The particle filter uses the probabilistic system transition model $P(X_t|X_{t-1})$ to describes the transition for state vector X_t and then predict the posterior at time t as

$$P(X_t|Z_{1:t-1}) = \int P(X_t|X_{t-1})P(X_{t-1}|Z_{1:t-1})dX_{t-1} \tag{12}$$

In which $Z_{1:t-1} = \{Z_1, Z_2, ..., Z_{t-1}\}$ are the available observations at times $1, 2, ..., t-1$, and $P(X_t|X_{t-1})$ expresses the motion model, $P(X_{t-1}|Z_{1:t-1})$ is posterior probability density function at time $t-1$ and $P(X_t|Z_{1:t-1})$ is the prior Probability Density Function (PDF) at time t. Now since the observation Z_t is available, we can updated the stateby Bayes's rule as:

$$P(X_t|Z_{1:t}) = \frac{P(Z_t|X_t)P(X_t|Z_{1:t-1})}{P(Z_t|Z_{1:t-1})} \tag{13}$$

In which $P(Z_t|X_t)$ is described by the observation equation. The posterior PDF $P(X_{t-1}|Z_{t-1})$ is approximated recursively as a set of N weighted samples $\{X^{(s)}, W^{(s)}\}^N$ and $W^{(s)}$

Using a Monte Carlo approximation of the integral we get,

$$P(X_t|Z_t) = P(Z_t|X_t)\sum_{s=1}^{N} W_{t-1}^{(s)} P(X_t|X_{t-1}^{(s)})$$

B. Implementation

The process of our particle filter are composed by the following parts.

1. Initialization, randomly generate a set of particles. Particles may have position, heading, accelerate and/or any other variables. They are initialized to have the same wright.
2. First, Prediction. For each inner iteration, the particles are moved according to how to predict to the robots behavior in this step. In our case, the prediction is made based on the heading of the robot.
3. Second, Update. The weight of the particles is updated. Generally, particles that closely match the measurement are weighted higher.
4. Resample. Highly improbable particles are replaced by more probable particles
5. Estimate. Compute and estimate the variables based on the mean and state and covariance.

Algorithm 1 PARTICLEFILTER(e, N , dbn)

Require: **e**, the new incoming evidence
N, the number of samples to be maintained
dbn, a DBN with prior $P(X0)$,
transition model, $P(X1|X0)$,
sensor model, $P(E1|X1)$
Ensure: a set of samples for the next time step
if not initialize **then**
$S \leftarrow$ Initialize $P(X_0)$
end if
for $i = 0$ to N **do**
Prediction $S[i] \leftarrow$ sample from $P(X_1|X_0 = S[i])$
Update and reweight $W[i] \leftarrow P(e|X_1 = S[i])$
end for
Discarded $S \leftarrow S\{x_{discard}\}$
Resample $S \leftarrow S \cup \{x_{new}\}$
return S

Algorithm 2 Function EuclidianWeighting(S, e)

Require: **S**, a vector of samples of size N
e, the new incoming evidence
Ensure: W a vector of weights of size N
for $i = 0$ to N **do**
$W[i]$ = constant / distance$(S[i], e, N)$
end for
return W

Algorithm 3 Resample(S, W)

Require: **S**, a vector of samples of size $M(M < N)$
W, a vector of samples of size $M(M < N)$
Ensure: S, a vector of samples of size N
Sum = **ComputeWeightSum**(W)
while Number of $(S) < N$ **do**
for particle in S **do**
S = S + **GenerateNewPartice**(particle, weight)
end for
end while
return S

V. Simulation

A. Simulation I: Comparison: Reach a Goal WithOUT Collision Checking

In this environment, the robot starts in a room, walking though a narrow corridor and reaches a goal in another room with stepsize equal to 1. The sensor signals (x_s, y_s) are added with a Gaussian noise (n_x, n_y) with zero mean and variance equal to 1.6.

$$(x_s, y_s) = (x_{true}, y_{true}) + (n_x, n_y)$$

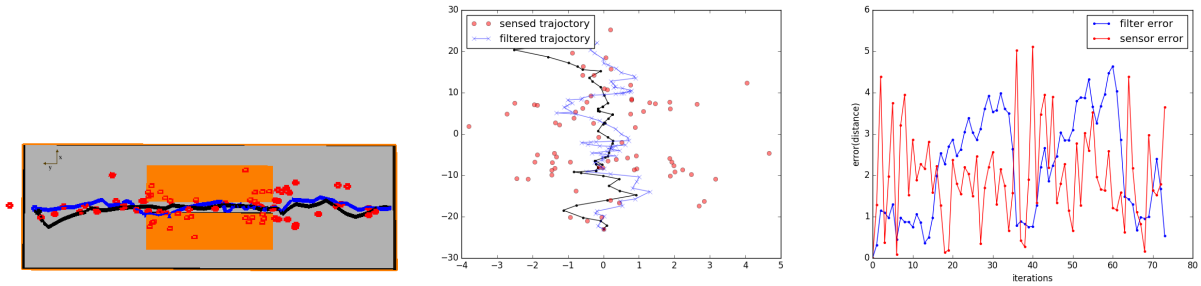
Similarly, the true position of destination of one step movement is added with a Gaussian noise (n_x, n_y) with zero mean and variance equal to 0.9.

$$(x_m, y_m) = (x_{true}, y_{true}) + (\delta x, \delta y) + (n_x, n_y)$$

The result of Kalman Filter and Particle Filter is shown in Fig1, Fig2, Fig3, Fig4.

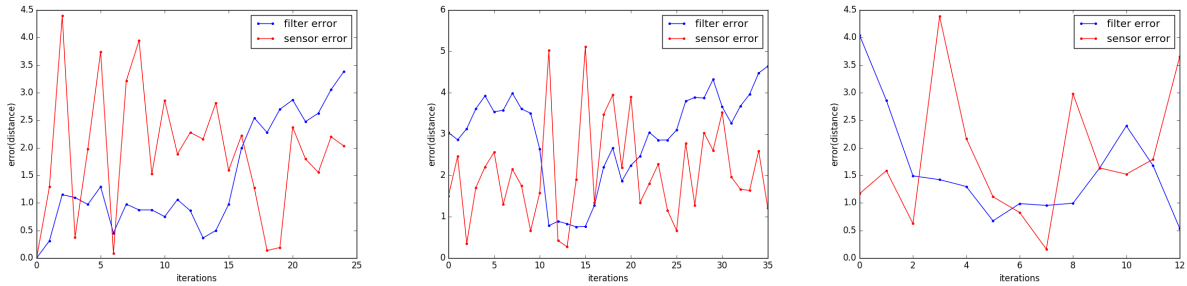
At beginning, the Kalman filter works very well. Based on the posterior estimation, it makes more accurate estimation when the sensor gives absurd measurement. The average of the distance error is around 0.7433, 74.33% of the step length, which is 1.

However, when the robot goes through the corridor, it is stucked when it takes noisy moves (the darker black dots). The error of Kalman filter becomes even much larger the error of the sensor becuase of the



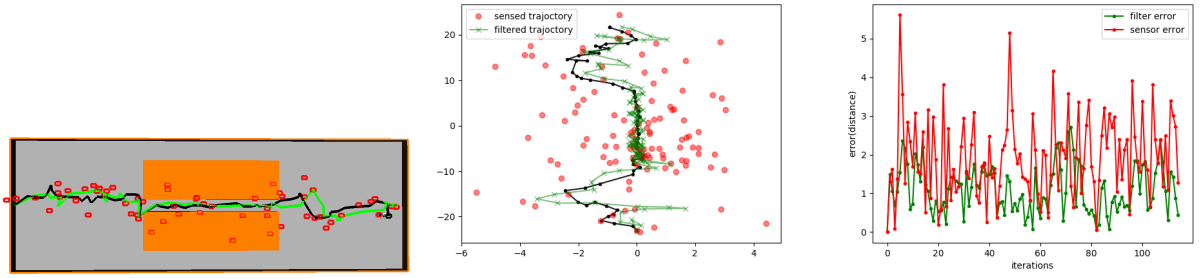
((a)) OpenRAVE ((b)) 2D demostration ((c)) Error

Fig. 1. Kalman Simulation WithOUT Collision Checking



((a)) In Room 1 ((b)) Through Corridor ((c)) In Room 2

Fig. 2. The Error of Kalman Filter withOUT Collision Checking



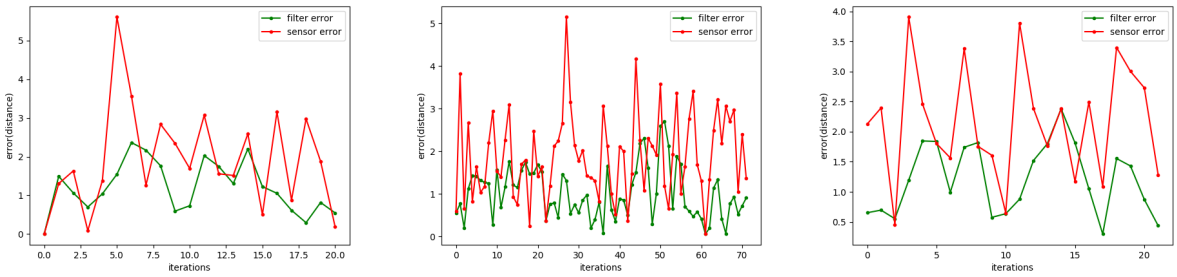
((a)) OpenRAVE ((b)) 2D demostration ((c)) Error

Fig. 3. Particle Filter Simulation WithOUT Collision Checking

accumulation of errors when it is stucked. Meanwhile, Particle Filter performs very well. Although the robot is stucked sometimes in the corridor, the Particle filter resamples and the filter location becomes more accurate.

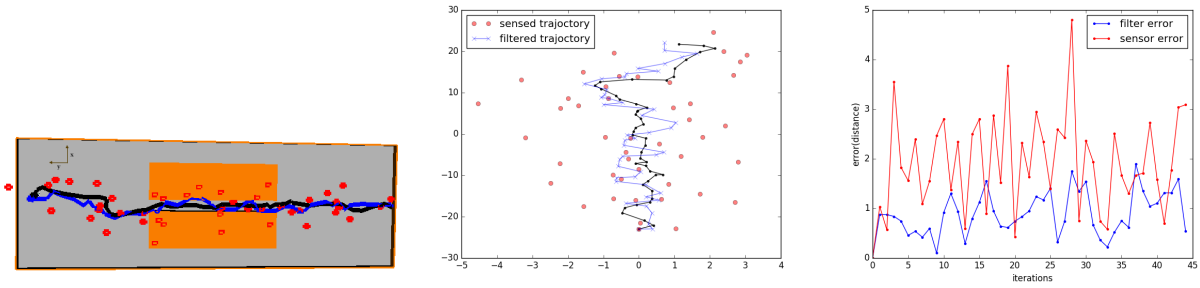
Also the simulation shows, the filtered position of Kalmn filter could be in the wall sometimes, while partial filter never returns unrealistic results. The reason is when Particle Filter randomly takes samples, it droppes the samples in obstacles. In this procedure, Particle Filter gains more information of unpossible places than Kalman Filter.

When the robot struggles out of the corridor, the Kalman Filter works well again. At each move, based on the information of the sensor and 'right' heading prediction, the robot corrects its postion and reduces error accumulated before. The situation of robot just



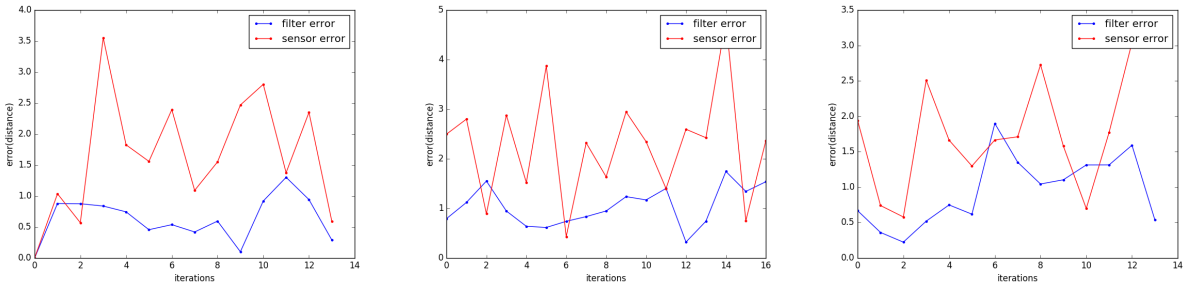
((a)) In Room 1 ((b)) Through Corridor ((c)) In Room 2

Fig. 4. The Error of Particle Filter withOUT Collision Checking



((a)) OpenRAVE ((b)) 2D demostration ((c)) Error

Fig. 5. Kalman Simulation With Collision Checking



((a)) In Room 1 ((b)) Through Corridor ((c)) In Room 2

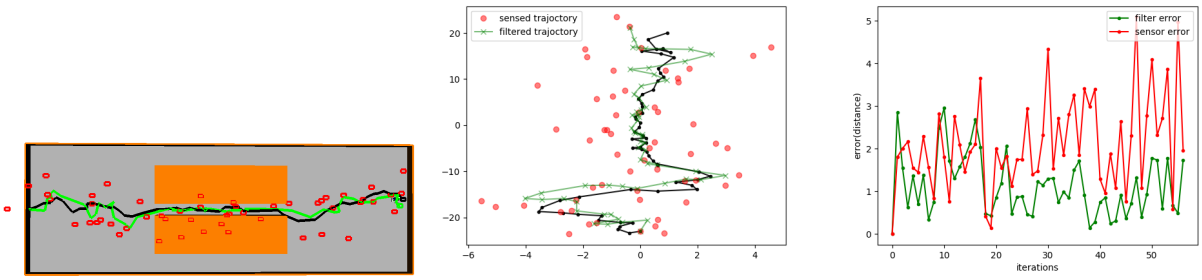
Fig. 6. The error of Kalman Filter With Collision Checking

walking to the end of the corridor equals to a simple kidnap problem. The robot believes itself at some place with a Gaussian probability, which, however, is not the real position. The robot corrects its estimation as it moves.

B. Simulation II: Comparison: Reach a Goal With Collision Checking

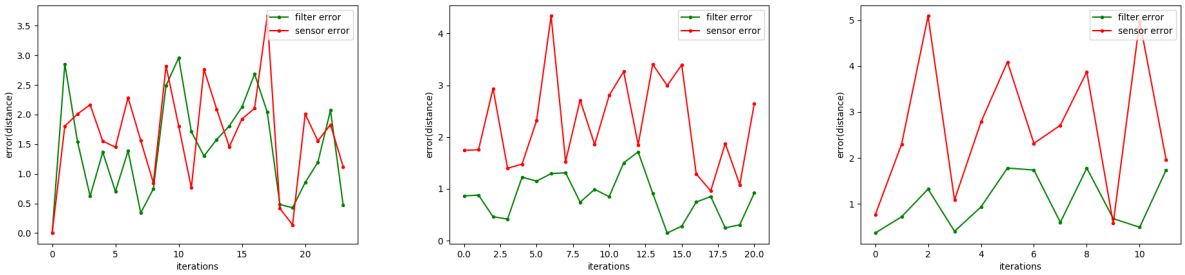
The environment and initial parameter is the same with Simulation I. The difference is the robot has a collision sensor, which means the robot won't collide to the obstacles and stucked at someplace.

The result in open area is the same with Simulation I. The interesting part is the robot moving through the



((a)) OpenRAVE ((b)) 2D demostration ((c)) Error

Fig. 7. Particle Filter Simulation WithOUT Collision Checking



((a)) In Room 1 ((b)) Through Corridor ((c)) In Room 2

Fig. 8. The Error of Particle Filter withOUT Collision Checking

corridor. With the information of collision checking, Kalman Filter performs much better than in Simulation I, comparing Fig8(b) and Fig2(b), even better than in the open area. Because of the narrow corridor, the robot cannot moves sideward and has higher probability moves along the desired direction.

C. Simulation III: Kalman Filter in Uniformly accelerated motion

In this part, we want to implement Kalman Filter with multiple states as discribed in Uniformly accelerated motion section. We use Kalman filter to tracking a uniformly accelerated moving particle under Gaussian disturbance. We just simply assume all the measurement have the same Gaussian noise with $\sigma_i^2 = 25$ variance and all the movement have the same Gaussian noise with $\sigma_i^2 = 1$ variance. First, Kalman Filter gets all the sensor results of three states, discribed in Equation 10. Fig 9(a) and Fig 9(b) is the result of the simulation.

We calculate the average of filter distance error is 2.7389 and the average of sensor distance error is 4.8734. Kalman Filter works well with 3 state variables.

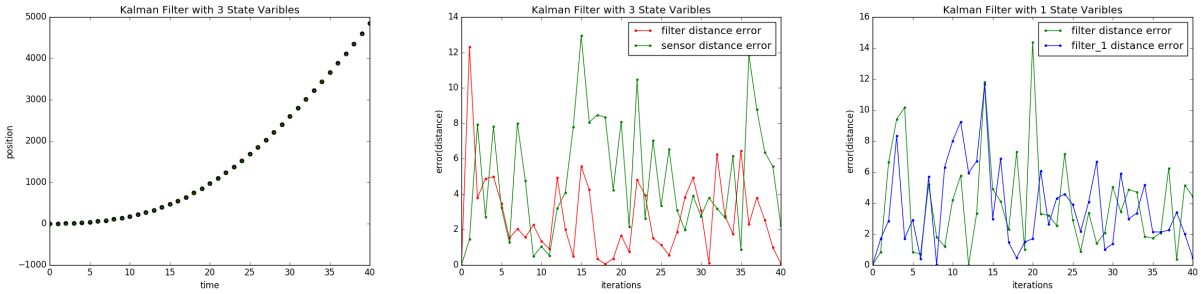
We run Kalman Filter with one state variable as discribed in Equation 11 under the same intial condition and get the following result, as shown in Fig 9(c).

The average of filtered distance error is 3.7316 and the average of sensor distance error is 3.9644. We add a lot of noise on position, velocity and acccelaration, Kalman Filter could only get the information of position. Less information leads to inaccuracy. figure

D. Simulation IV: Sonar Sensor

Most of the times, we do not have a have GPS-like sensor that will directly gives robot a position. However, we may have sonar or radar. Typically, A sonar emitting pulses of sounds and listening for echoes, so the robot can know whether there is a collision at the certain direction.

However, the result of a sonar is non Gaussian and non linear. Thus we can not imply a Kalman filter to handle with a sonar sensor. Particle filter, due to its versatility, can be used in this situation. Whats more, when the robot



((a)) Simulation

((b)) Error of Kalman Filter with three state varibles

((c)) Error of Kalman Filter with one state varibles

Fig. 9. The simulation of Kalman Filter with mutiple state variables

is using particle filter and sonar sensor for localization, some interesting feature will appear.

In order to simulate this problem, a simple sonar sensor is made. It will emit pulses along the coordinate. And return whether there is a collision and the distance of the collision in this direction.

A interesting map is created where a lot symmetric features are included. There features will cause some interesting result for sonar sensor.

As shown in Fig10(a), at first, the particle filter is initialized by spreading normalized random particles in the rooms.

From Fig10(b) we can see that after one iteration. The particles are resampled. Some of the particles moves towards to the while a lot of particles are at everywhere.

In Fig10(c), the result of the first localization is rather bad. The reason is that. Since there are a lot of symmetric features in this map. Some particles or states in different location has a similar weight compare to the real location.

After some iterations, like Fig10(d), we can see there are still some outliner particles.

However, since the particle filter is always predicting, updating and resampling according its own prediction of movement and sensor. The particle trends to converge to the real location(Fig 10(e)).

The particles are being resampled at the real position(Fig10(f)).

Now we can see that, all the particles are just around the real location(Fig10(g)).

At first the particle filter is hard to localize itself, but after some iterations, the filter starts get the real position(Fig10(h)). Particle filter is able to handle with non Gaussian non linear problems such as sonar sensor. While the kalman filter can not deal with this problem.

VI. Conclusion

From the simulation we can see that:

1. Both karman filter and particle filter can deal with linear and Gaussian models. If a sensor with Gaussian error is given, a position tracking localization problem can be solved by both filters.

2. Kalman Filter performs well in open area, which can make a reasonable estimation when the sensor gives too inaccurate results. But when the robot stucked in some place, the error will accumulate. Also, Kalman Filter does not know the environment and may give a unreasonable result is obstacles.

3. There are many methods for weighting and resampling in Particle Filter. The result of the the particle filter is highly sensitive to these methods.

4. When there is a lot of collisions, the particle filter performs better than Kalman filter since it avoids collision when sampling.

5. When dealing with non Gaussian and non linear problemssuch as using a sonar sensor to solve for a kidnap problem, only particle filter works since it does not rely on restrictive assumptions.

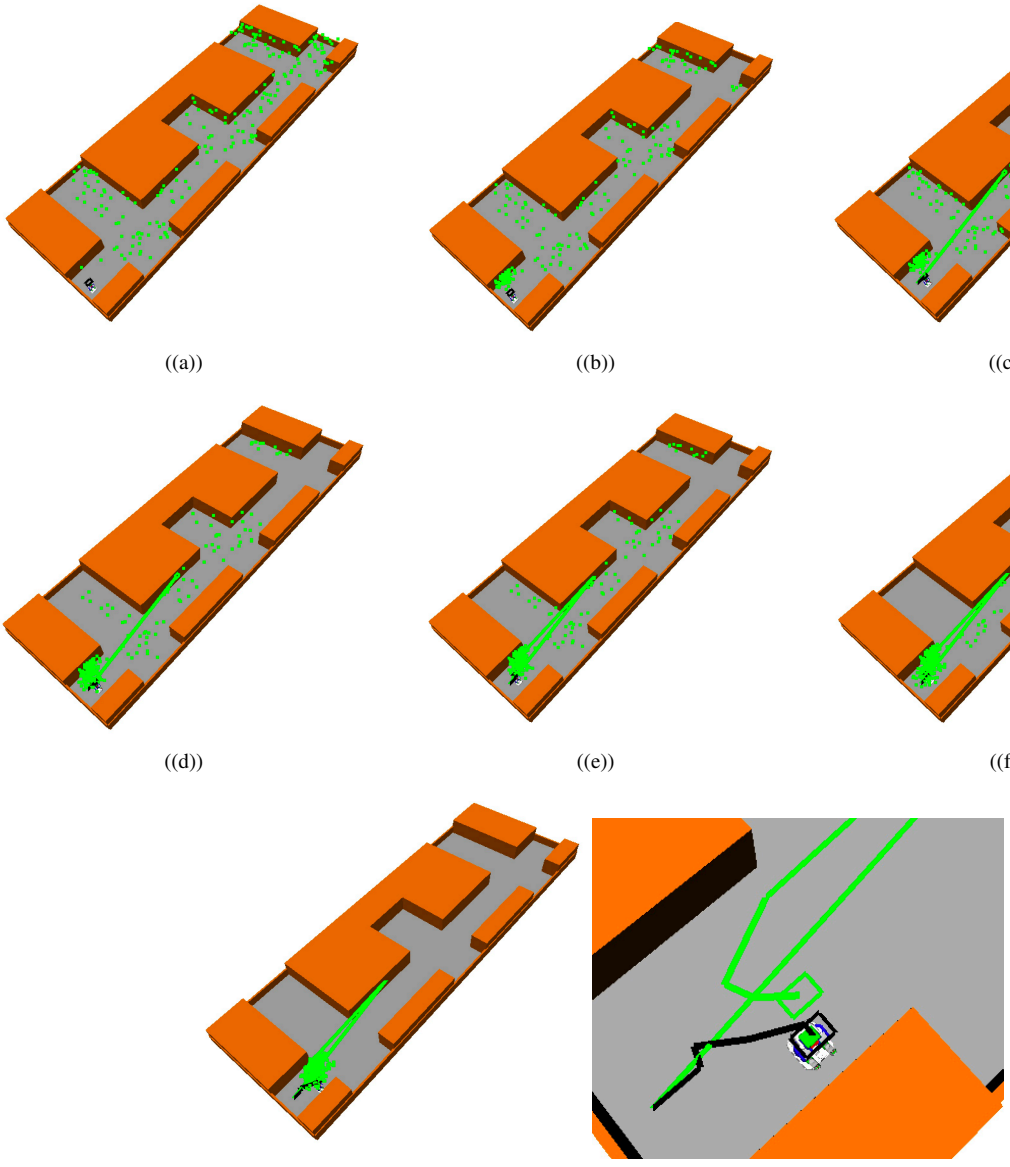
References

[1] Russell S, Norvig P, Intelligence A. A modern approach[J]. Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs, 1995, 25: 27.

[2] Thrun S, Fox D, Burgard W, et al. Robust Monte Carlo localization for mobile robots[J]. Artificial intelligence, 2001, 128(1-2): 99-141.

[3] J. Borenstein, B. Everett, and L. Feng. Navigating Mobile Robots: Systems and Techniques. A. K. Peters, Ltd., Wellesley, MA, 1996.

[4] W.Burgard,A.Derr,D.Fox,andA.B.Cremers.Integrating global position estimation and position tracking for mobile robots: The dynamic markov localization approach. In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS98), 1998. To appear.



((a)) ((b)) ((c))

((d)) ((e)) ((f))

((g)) ((h)) Sonar Result

Fig. 10. The simulation of Particle Filter with sonar sensor

[5] T. L. Dean and M. Boddy. An analysis of time-dependent planning. In Proceeding of Seventh National Conference on Artificial Intelligence AAAI-92, pages 4954, Menlo Park, CA, 1988. AAAI, AAAI Press/The MIT Press.

[6] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In Proceedings of the IEEE International Conference on Robotics and Au- tomation (ICRA), 1999.

[7] A Doucet. On sequential simulation-based methods for bayesian filtering. Technical Report CUED/F-INFENG/TR 310, Cambridge University, Department of Engineer- ing, Cambridge, UK, 1998.

[8] A.Doucet,N.J.Gordon,andJ.F.G.deFreitas,editors.Sequential Monte Carlo Methods In Practice. forthcoming, 2000.

[9] S. Engelson and D. McDermott. Error correction in mobile robot map learning. In Proceedings of the 1992 IEEE International Conference on Robotics and Automation, pages 25552560, Nice, France, May 1992.

[10] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In Proceedings of the National Conference on Artificial Intelligence (AAAI), Orlando, FL, 1999. AAAI.

[11] J.-S. Gutmann and C. Schlegel. Amos: Comparison of scan matching approaches for self-localization in indoor environments. In Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots. IEEE Computer Society Press, 1996.

[12] R. E. Kalman. A new approach to linear filtering and prediction problems. Trans. ASME, Journal of Basic Engineering, 82:3545, 1960.
[13] J. Liu and R. Chen. Sequential monte carlo methods for dynamic systems. Journal of the American Statistical Association, 93, 1998.
[14] F. Lu and E. Milios. Globally consistent range scan alignment for environment map- ping. Autonomous Robots, 4:333349, 1997.
[15] M. Pitt and N. Shephard. Filtering via simulation: auxiliary particle filter. Journal of the American Statistical Association, 1999. Forthcoming.
[16] W.D. Rencken. Concurrent localisation and map building for mobile robots using ultrasonic sensors. In Proceedings of the IEEE/RSJ International Conference on In- telligent Robots and Systems, pages 21292197, Yokohama, Japan, July 1993.
[17] D.B. Rubin. Using the SIR algorithm to simulate posterior distributions. In M.H. Bernardo, K.M. an DeGroot, D.V. Lindley, and A.F.M. Smith, editors, Bayesian Statistics 3. Oxford University Press, Oxford, UK, 1988.
[18] B. Schiele and J. Crowley. A comparison of position estimation techniques using occupancy grids. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation, pages 16281634, San Diego, CA, May 1994.
[19] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I.J. Cox and G.T. Wilfong, editors, Autonomous Robot Vehnicles, pages 167193. Springer-Verlag, 1990.
[20] G. Wei, C. Wetzler, and E. von Puttkamer. Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In Proceedings of the International Conference on Intelligent Robots and Systems, pages 595601, 1994.
[21] B. Yamauchi and R. Beer. Spatial learning for navigation in dynamic environ- ments. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cyber- netics, Special Issue on Learning Autonomous Robots, 1996. also located at http://www.aic.nrl.navy.mil/ yamauchi/.
[22] S. Zilberstein and S. Russell. Approximate reasoning using anytime algorithms. In S. Natarajan, editor, Imprecise and Approximate Computation. Kluwer Academic Publishers, Dordrecht, 1995.
[23] Mahmoud I I, Salama M, El Tawab A A. Particle/Kalman Filter for Efficient Robot Localization[J]. International Journal of Computer Applications, 2014, 106(2).