# EECS498 Final Project Report
# Localization

Chun-Hsueh Lee, Zheng-Hang Yeh

January 2, 2022

## 1 Introduction

Pose estimation has long been widely used in the field of robotics. Not only robotics, states tracking of a nuclear power plant or a satellite is also very crucial. Due to the rapid rising of self-driving cars and other autonomous vehicles, pose estimation is a more and more popular topic in both research and industrial fields. In fact, pose estimation or localization plays a critical role in mobile robots, drones, humanoid robot, biped robots, and self-driving cars since it is a way of making these systems being aware of where they are. Knowing "where am I" is a foundation that further complex missions based on, for example, mapping and navigation tasks.

One of the biggest challenges in pose estimation, especially localization is that the sensors will never be perfect. As a result, methods which consider the uncertainty of the sensor measurement is popular in these decades. Probablistic filters such as Kalman Filter, Extended Kalman Filter, Uncented Kalman Filter, and Particle Filter are the most well known methods to handle pose estimation with linear or non-linear systems with different probability distributions.

Nevertheless, each method has its own defects, the users should pick the method that suits their scenarios the best instead of picking the best method. In this project, Extended Kalman Filter and Particle Filter are implemented in a simulated environment and compared by executing the exact same path and giving the same sensor measurement. The simulated environment setup and the equations used for implementing the two method will be discussed in the following sections.

## 2 Implementation

### 2.1 Particle Filter

#### 2.1.1 Structure of Particle Filter

In order to complete particle filter, there are 3 main parts, we need odometry data, sensor data, and particle filter function itself. Particles will move in each iteration base on odometry with some additional random walk. After that, the weight of each particles will be updated base on the sensor model. Then, particles are resampled base on their weight. Finally, we will get the estimated pose by calculating the mean of all particles. The flow chart of particle filter is shown below.

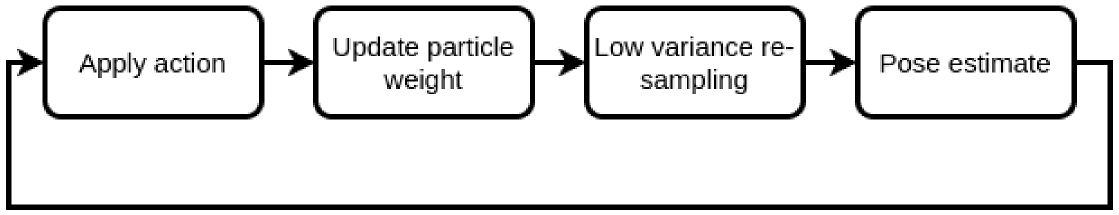


Figure 1: Particle filter flow chart

### 2.1.2 Odometry

In this project, we assume the robot is a differential drive mobile robot. Similar equations can be found in [1]. The state will be $X = [x, y, \theta]'$, and the control input $u = [v, \omega]'$.

To make the odometry drift like a real robot driving on the floor, we add some noise to the control input. The noise $R$ is defined as below.

$$R = \begin{bmatrix} c_1 v^2 + c_2 \omega^2 & 0 \\ 0 & c_3 v^2 + c_4 \omega^2 \end{bmatrix} \tag{1}$$

where $c1 = c2 = c3 = c4 = 0.5$, and the actual input becomes

$$u = \begin{bmatrix} v_{actual} \\ \omega_{actual} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} v \\ \omega \end{bmatrix}, R) \tag{2}$$

The basic kinematic model is defined as below.

$$x_{t+1} = x_t + v_{actual}\ \Delta t\ cos(\theta + \omega_{actual}\Delta t) \tag{3}$$

$$y_{t+1} = y_t + v_{actual}\ \Delta t\ sin(\theta + \omega_{actual}\Delta t) \tag{4}$$

$$\theta_{t+1} = \theta_t + \omega_{actual}\Delta t \tag{5}$$

### 2.1.3 Sensor Measurement

For sensor measurement, we are assuming that the measurement $z$ has a gaussian noise with zero mean. We will drive the robot ideally as a differential drive robot with ideal input $u = [v, \omega]'$, and treat the current location as ground truth. The equations are shown as following.

$$x_{t+1} = x_t + v\ \Delta t\ cos(\theta + \omega\Delta t) \tag{6}$$

$$y_{t+1} = y_t + v\ \Delta t\ sin(\theta + \omega\Delta t) \tag{7}$$

$$\theta_{t+1} = \theta_t + \omega\Delta t \tag{8}$$

$$z_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} + \epsilon \tag{9}$$

$$\epsilon \sim \mathcal{N}(0, Q) \tag{10}$$

$$Q = \begin{bmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & k \end{bmatrix} \tag{11}$$

where $k = 0.05$

### 2.1.4 Action Model

For the action model which is applied on every particle, RTR model is chosen. In order to maintain the diversity of the particles, noise is also added in the action model. The pose at time $t-1$ is donote as $[x_{t-1},\ y_{t-1},\ \theta_{t-1}]$, the pose at time $t$ is denote as$[x_t,\ y_t,\ \theta_t]$ The equations are shown below [2].

$$\Delta x = x_t - x_{t-1} \tag{12}$$

$$\Delta y = y_t - y_{t-1} \tag{13}$$

$$\Delta\theta = \theta_t - \theta_{t-1} \tag{14}$$

$$\Delta s^2 = \Delta x^2 + \Delta y^2 \tag{15}$$

$$\alpha = atan2(\Delta y, \Delta x) - \theta_{t-1} \tag{16}$$

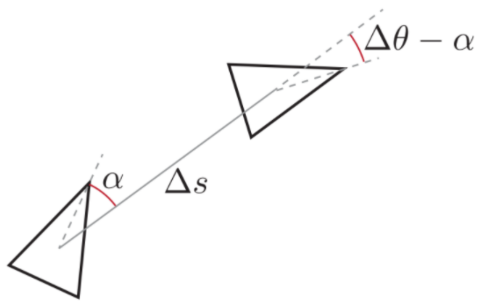


Figure 2: RTR action model [2]

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta s + \varepsilon_2)\cos(\theta_{t-1} + \alpha + \varepsilon_1) \\ (\Delta s + \varepsilon_2)\sin(\theta_{t-1} + \alpha + \varepsilon_1) \\ \Delta\theta + \varepsilon_1 + \varepsilon_3 \end{bmatrix} \tag{17}$$

$$\begin{gathered} \varepsilon_1 \sim \mathcal{N}(0, k_1|\alpha|) \\ \varepsilon_2 \sim \mathcal{N}(0, k_2|\Delta s|) \\ \varepsilon_3 \sim \mathcal{N}(0, k_1|\Delta\theta - \alpha|) \end{gathered} \tag{18}$$

where $k_1 = k_2 = 0.5$

#### 2.1.5 Weight Update

To calculate and update the weight of each particle, we update the weight based on $p(z_t|x_i)$, where $z$ is the sensor measurement and $x_i$ is the state of the $i$-th particle.

$$w_i = p(z_t|x_i) \tag{19}$$

After updating the weights of all particles, they should be normalized. Assume there are $N$ particles.

$$w_i' = \frac{w_i}{\sum_{i=0}^{N} w_i} \tag{20}$$

#### 2.1.6 Low Variance Re-sampling

For res-sampling method, instead of using sampling with replacement, which could sacrifice the diversity of particles, low variance sampler was chosen to preserve particle diversity in this project [3].

After re-sampling, the particles weights should be normalized again using the same equation as equation 20

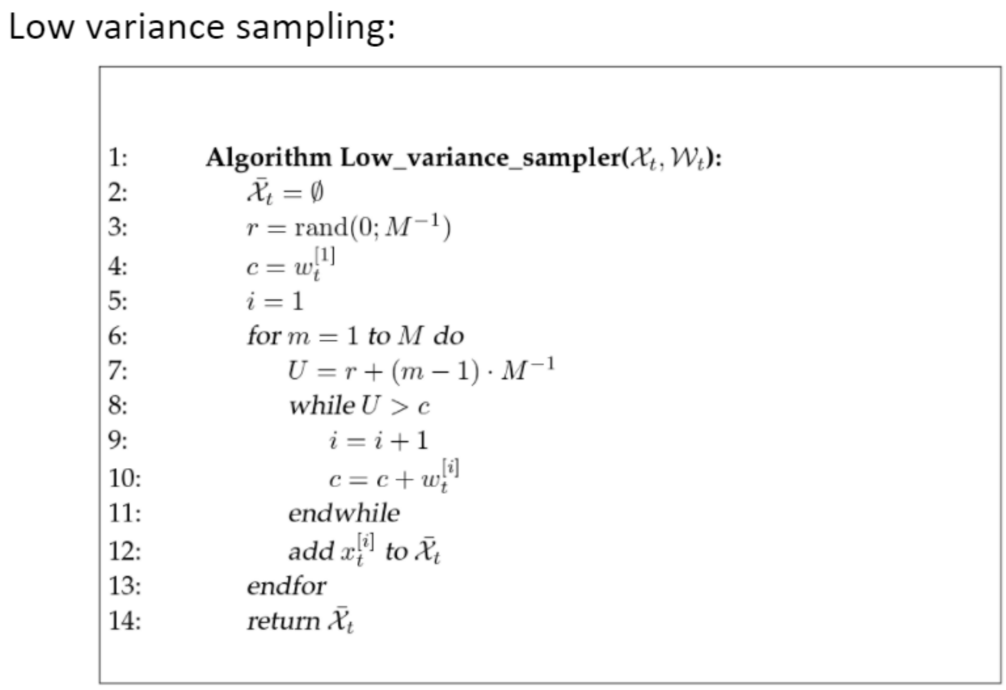
#### 2.1.7 Pose estimation

For pose estimation, the current pose should be the weighted sum of all particles. Assume there are $N$ particles, current pose $[\bar{x},\ \bar{y},\ \bar{\theta}]$ is shown as below [4].

$$\bar{x} = \sum_{i=0}^{N} w_i' x_i \tag{21}$$

$$\bar{y} = \sum_{i=0}^{N} w_i' y_i \tag{22}$$

$$\bar{\theta} = atan2\left(\sum_{i=0}^{N} w_i' \sin(\theta_i), \sum_{i=0}^{N} w_i' \cos(\theta_i)\right) \tag{23}$$


Low variance sampling:

```
1:  Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):
2:      $\bar{\mathcal{X}}_t = \emptyset$
3:      $r = \text{rand}(0; M^{-1})$
4:      $c = w_t^{[1]}$
5:      $i = 1$
6:      for $m = 1$ to $M$ do
7:          $U = r + (m-1) \cdot M^{-1}$
8:          while $U > c$
9:              $i = i + 1$
10:             $c = c + w_t^{[i]}$
11:         endwhile
12:         add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
13:     endfor
14:     return $\bar{\mathcal{X}}_t$
```

Figure 3: Pseudo code of low variance sampler[3]

## 2.2 Extended Kalman Filter

### 2.2.1 Motion Model

The nonlinear motion model for EKF in general form is:

$$x_t = g(x_{t-1}, u_t) + \tag{24}$$

Since the kinematic model we used is a differential drive model, it is nonlinear and thus we cannot use simple Kalman Filter. Instead, we use Extended Kalman Filter (EKF) to perform state estimation of a nonlinear system. The differential drive kinematic equation for our robot is below:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t + \gamma_t \Delta t \end{pmatrix} \tag{25}$$

A simple derivation [5] of this kinematic equation is shown below:

First we assume the robot move from time t to time t+1 with constant translational velocity v and rotational velocity w:

We know that the relation between v and w with respect to the radius of rotation is:

$$v = \omega \cdot r \tag{26}$$

Hence, we can rewrite the radius using v and w as below:

$$r = \left| \frac{v}{\omega} \right| \tag{27}$$

Also, from the geometry we can calculate the center of rotation (xc, yc) using current robot location (x, y) and current velocity command u = (v, w):

$$\begin{aligned} x_c &= x - \tfrac{v}{\omega}\sin\theta \\ y_c &= y + \tfrac{v}{\omega}\cos\theta \end{aligned} \tag{28}$$

After Δ t time of motion, the ideal robot location is:

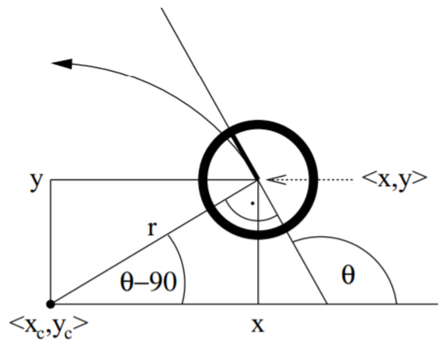


Figure 4: A figure showing the geometry analysis of the robot motion. Assume the robot is moving under constant translational and rotational velocities

$$\begin{aligned}\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} &= \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix} \\ &= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \omega\Delta t \end{pmatrix}\end{aligned} \tag{29}$$

This is how we derive our differential drive motion model kinematic equation.

### 2.2.2 Sensor Model

The nonlinear sensor model for the EKF algorithm in general form is:

$$z_t = h(x_t) + v \tag{30}$$

In our project, it is just the true pose of the robot.

$$z_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + v \tag{31}$$

where v is sensor noise with Gaussian distriution with sensor covariance Q

$$v \sim \mathcal{N}(0,\, Q)\,. \tag{32}$$

We add some noise to simulate real-world noisy measurements. The noise is assumed to be a Gaussian distribution with zero mean and covariance Q. Note that this is one of the limitations of the series of kalman filters since they assume the noises are Gaussian distribution, while particle filter doesn't have this assumption and limitation. Hence, particle filters can track arbitrary types of probability distribution.

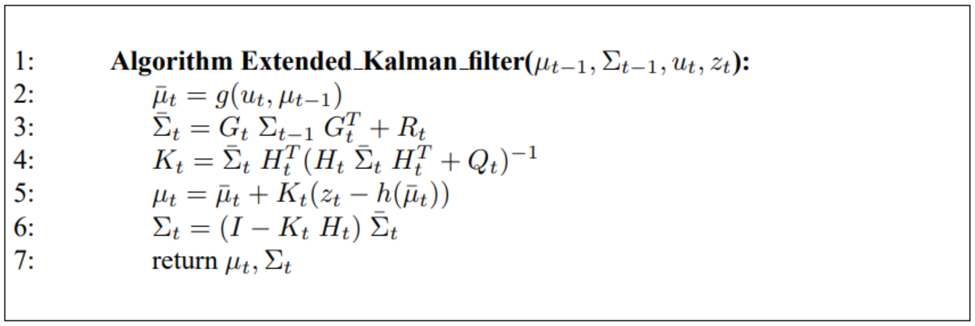
### 2.2.3 EKF algorithm

The general form of EKF algorithm is shown in Fig.5 [6] A flowchart is shown in Fig. 7 to better illustrate the process of running Extended Kalman Filter.

The main idea of EKF is to linearize the nonlinear system dynamics at its operating point. The linearization method is using the Taylor Series Expansion and keeping only the zero and first order term. That is, to compute the Jacobians of the nonlinear motion model and sensor model. The comparison between Kalman Filter and EKF is shown in Fig. 6

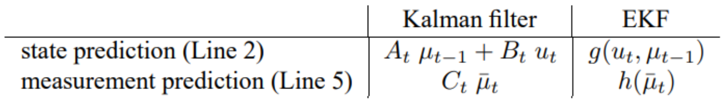
The EKF algorithm we implemented is shown in Fig.8 [7]

The EKF algorithm is split into 3 steps as in Kalman Filter: Prediction, Computation of Kalman Gain, and Correction using sensor measurements.



```
1:  Algorithm Extended_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:      $\bar{\Sigma}_t = G_t\ \Sigma_{t-1}\ G_t^T + R_t$
4:      $K_t = \bar{\Sigma}_t\ H_t^T (H_t\ \bar{\Sigma}_t\ H_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:      $\Sigma_t = (I - K_t\ H_t)\ \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

Figure 5: General algorithm of Extended Kalman Filter Algorithm



| | Kalman filter | EKF |
|---|---|---|
| state prediction (Line 2) | $A_t\ \mu_{t-1} + B_t\ u_t$ | $g(u_t, \mu_{t-1})$ |
| measurement prediction (Line 5) | $C_t\ \bar{\mu}_t$ | $h(\bar{\mu}_t)$ |

Figure 6: Comparison of motion and measurement models between KF and EKF

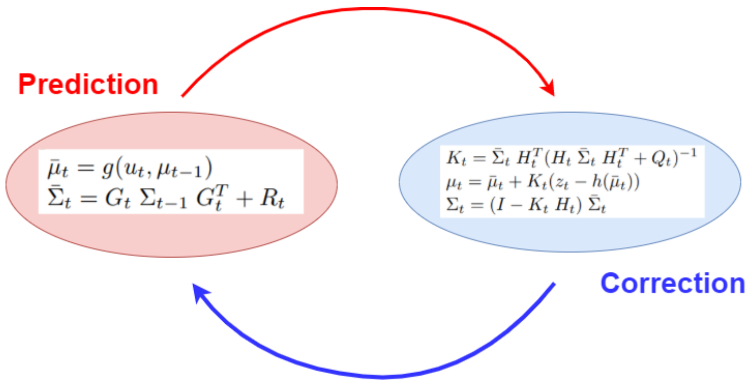
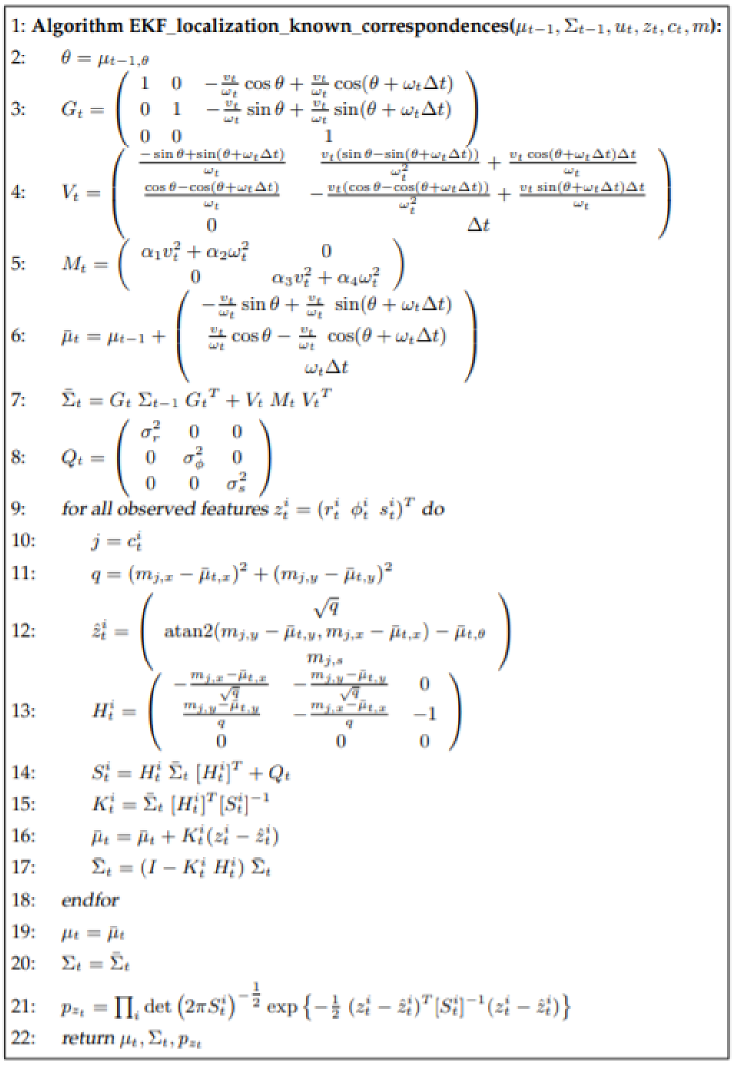

Prediction
$\bar{\mu}_t = g(u_t, \mu_{t-1})$
$\bar{\Sigma}_t = G_t\ \Sigma_{t-1}\ G_t^T + R_t$
$K_t = \bar{\Sigma}_t\ H_t^T (H_t\ \bar{\Sigma}_t\ H_t^T + Q_t)^{-1}$
$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
$\Sigma_t = (I - K_t\ H_t)\ \bar{\Sigma}_t$
Correction

Figure 7: Flowchart of EKF algorithm



```
1: Algorithm EKF_localization_known_correspondences($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, m$):
2:     $\theta = \mu_{t-1,\theta}$
3:     $G_t = \begin{pmatrix} 1 & 0 & -\frac{v_t}{\omega_t}\cos\theta + \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ 0 & 1 & -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$
4:     $V_t = \begin{pmatrix} \frac{-\sin\theta + \sin(\theta+\omega_t\Delta t)}{\omega_t} & \frac{v_t(\sin\theta - \sin(\theta+\omega_t\Delta t))}{\omega_t^2} + \frac{v_t\cos(\theta+\omega_t\Delta t)\Delta t}{\omega_t} \\ \frac{\cos\theta - \cos(\theta+\omega_t\Delta t)}{\omega_t} & -\frac{v_t(\cos\theta - \cos(\theta+\omega_t\Delta t))}{\omega_t^2} + \frac{v_t\sin(\theta+\omega_t\Delta t)\Delta t}{\omega_t} \\ 0 & \Delta t \end{pmatrix}$
5:     $M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2\omega_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4\omega_t^2 \end{pmatrix}$
6:     $\bar{\mu}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$
7:     $\bar{\Sigma}_t = G_t\,\Sigma_{t-1}\,{G_t}^T + V_t\,M_t\,{V_t}^T$
8:     $Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$
9:     for all observed features $z_t^i = (r_t^i\ \ \phi_t^i\ \ s_t^i)^T$ do
10:        $j = c_t^i$
11:        $q = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$
12:        $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \\ m_{j,s} \end{pmatrix}$
13:        $H_t^i = \begin{pmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 \\ 0 & 0 & 0 \end{pmatrix}$
14:        $S_t^i = H_t^i\,\bar{\Sigma}_t\,[H_t^i]^T + Q_t$
15:        $K_t^i = \bar{\Sigma}_t\,[H_t^i]^T[S_t^i]^{-1}$
16:        $\bar{\mu}_t = \bar{\mu}_t + K_t^i(z_t^i - \hat{z}_t^i)$
17:        $\bar{\Sigma}_t = (I - K_t^i\,H_t^i)\,\bar{\Sigma}_t$
18:    endfor
19:    $\mu_t = \bar{\mu}_t$
20:    $\Sigma_t = \bar{\Sigma}_t$
21:    $p_{z_t} = \prod_i \det\left(2\pi S_t^i\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\,(z_t^i - \hat{z}_t^i)^T[S_t^i]^{-1}(z_t^i - \hat{z}_t^i)\right\}$
22:    return $\mu_t, \Sigma_t, p_{z_t}$
```

Figure 8: EKF algorithm implementation

### 2.2.4 Prediction

The prediction step of the EKF is to generate a prior distribution using the motion model. Recall our motion model:

$$\underbrace{\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}}_{x_t} = \underbrace{\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin\left(\theta + \omega_t \Delta t\right) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos\left(\theta + \omega_t \Delta t\right) \\ \omega_t \Delta t \end{pmatrix}}_{g(u_t, x_{t-1})} + \mathcal{N}\left(0, R_t\right) \tag{33}$$

Same as the particle filter described above, we use the same predefined control inputs to drive the pr2 robot. Then, we use the v and w to calculate the prior estimation of the robot pose, that is, the prediction of the robot pose. Note that we do have process noise in our motion model, and the process noise comes from the uncertainty in control space. For the prediction step of the EKF algorithm, we need to compute the Jacobian of nonlinear function with respect to the state.

$$G_t = \frac{\partial g\left(u_t, \mu_{t-1}\right)}{\partial x_{t-1}} = \begin{pmatrix} \frac{\partial x'}{\partial \mu_{t-1,x}} & \frac{\partial x'}{\partial \mu_{t-1,y}} & \frac{\partial x'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y'}{\partial \mu_{t-1,x}} & \frac{\partial y'}{\partial \mu_{t-1,y}} & \frac{\partial y'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta'}{\partial \mu_{t-1,x}} & \frac{\partial \theta'}{\partial \mu_{t-1,y}} & \frac{\partial \theta'}{\partial \mu_{t-1,\theta}} \end{pmatrix} \tag{34}$$

$$G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\left(-\cos\mu_{t-1,\theta} + \cos\left(\mu_{t-1,\theta} + \omega_t \Delta t\right)\right) \\ 0 & 1 & \frac{v_t}{\omega_t}\left(-\sin\mu_{t-1,\theta} + \sin\left(\mu_{t-1,\theta} + \omega_t \Delta t\right)\right) \\ 0 & 0 & 1 \end{pmatrix} \tag{35}$$

Also, we need to map the uncertainty from the control space into the state space. Hence, we compute following matrices:

$$M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 \omega_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_3 \omega_t^2 \end{pmatrix} \tag{36}$$

$$\begin{aligned} V_t &= \frac{\partial g\left(u_t, \mu_{t-1}\right)}{\partial u_t} \\ &= \begin{pmatrix} \frac{\partial x'}{\partial v_t} & \frac{\partial x'}{\partial \omega_t} \\ \frac{\partial y'}{\partial v_t} & \frac{\partial y'}{\partial \omega_t} \\ \frac{\partial \theta'}{\partial v_t} & \frac{\partial \theta'}{\partial \omega_t} \end{pmatrix} \\ &= \begin{pmatrix} \frac{-\sin\theta + \sin(\theta + \omega_t \Delta t)}{\omega_t} & \frac{v_t(\sin\theta - \sin(\theta + \omega_t \Delta t))}{\omega_t^2} + \frac{v_t \cos(\theta + \omega_t \Delta t)\Delta t}{\omega_t} \\ \frac{\cos\theta - \cos(\theta + \omega_t \Delta t)}{\omega_t} & -\frac{v_t(\cos\theta - \cos(\theta + \omega_t \Delta t))}{\omega_t^2} + \frac{v_t \sin(\theta + \omega_t \Delta t)\Delta t}{\omega_t} \\ 0 & \Delta t \end{pmatrix} \end{aligned} \tag{37}$$

After computing all the Jacobian matrices we need, we can compute our prior distribution of the state:

$$\begin{aligned} \bar{\mu}_t &= \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin\left(\theta + \omega_t \Delta t\right) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos\left(\theta + \omega_t \Delta t\right) \\ \omega_t \Delta t \end{pmatrix} \\ \bar{\Sigma}_t &= G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T \end{aligned} \tag{38}$$

### 2.2.5 Kalman Gain

For the Kalman Gain, we use the general formula

$$K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t\right)^{-1} \tag{39}$$

The matrix H is the first order partial derivative of nonlinear sensor model *h(xt)* with respect to state *xt*. However, since our sensor model is actually **linear**, we simply set H to be equal to C, which is the identity matrix I.

| Parameters | Description | Values |
|---|---|---|
| c1, c2, c3, c4 | Stochastic gains in motion model | 5 |
| Q | Sensor model covariance (diagonal matrix) | 0.01 |
| k | Noisy measurements covariance (diagonal matrix) | 0.05 |
| dt | Sampling period | 0.1 |

Table 1: A table of parameters set in both algorithms.

#### 2.2.6 Correction

After calculating the kalman gain K, we can update the posterior estimation of the state using our sensor measurments. We simply apply the formula in Fig. 8 :

$$\begin{aligned} \mu_t &= \bar{\mu}_t + K_t\left(z_t - h\left(\bar{\mu}_t\right)\right) \\ \Sigma_t &= \left(I - K_t H_t\right)\bar{\Sigma}_t \end{aligned} \tag{40}$$

## 3 Results

### 3.1 Experimental Setup

In the experimental setup, pybullet simulated environment was used. In the simulation, one PR2 robot was used, the obstacles are six IKEA table. The start position of the robot is the bottom left corner and the goal point is the corner of the 2 tables at the right side of the environment. The robot path is set to be close to the obstacles so that collision can be easily checked. The environment setup is shown as Fig. 9 and the parameters of the experiments is shown in Table 1.

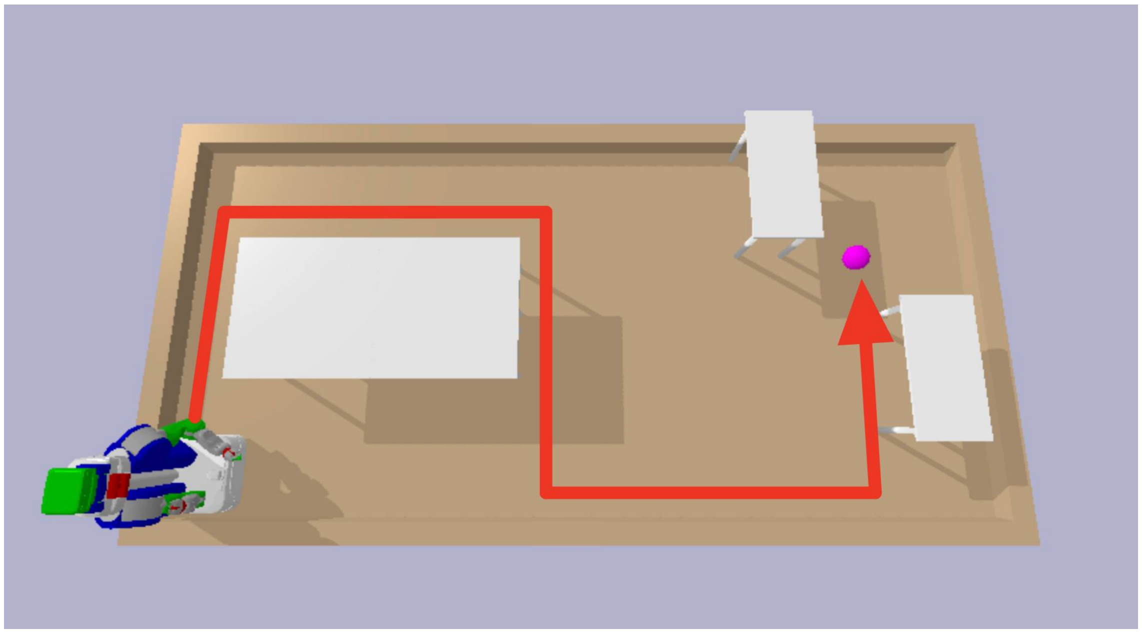


Figure 9: Robot path and obstacle setup

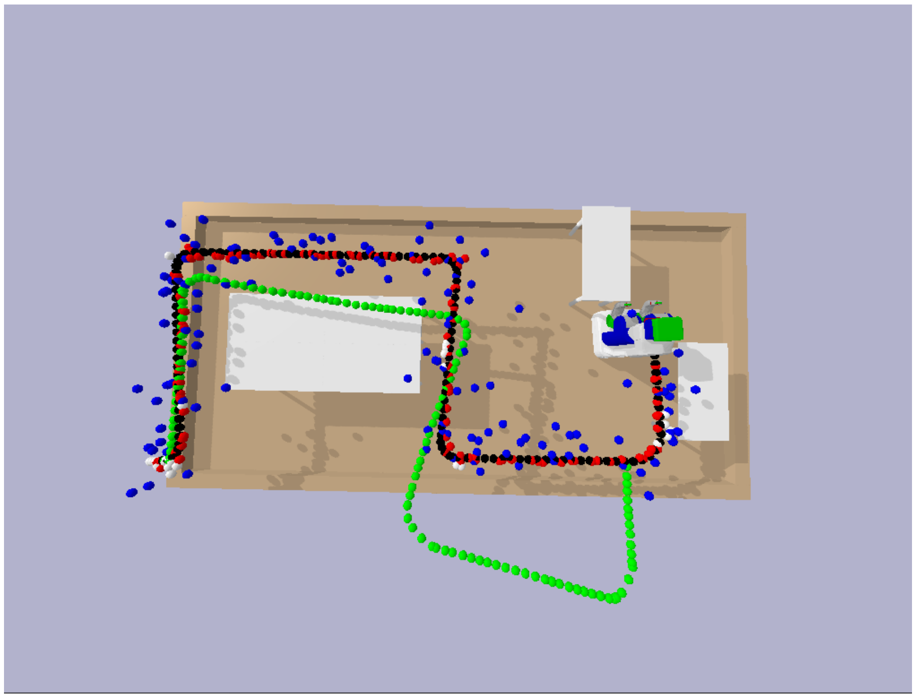


Figure 10: Simulation result of running particle filter. The red balls are the estimated poses with no collision, white balls are estimated poses with collision, yellow balls are odometry poses, black balls are true poses ,and blue balls are noisy measurements

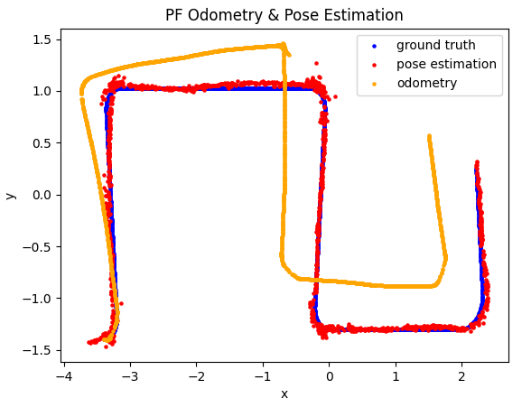


Figure 11: 2D plot of the estimated poses(red dots) versus true poses(blue dots), and odometry(orange dots).

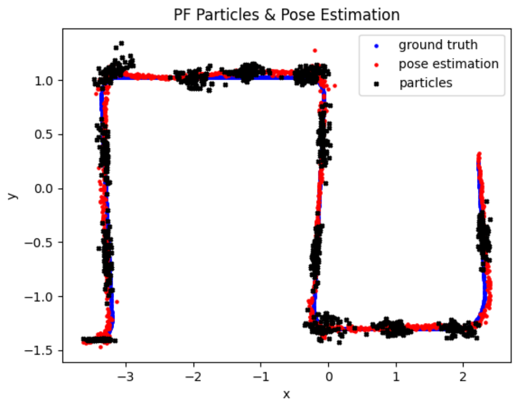


Figure 12: 2D plot of the estimated poses(red dots) versus true poses(blue dots), and particle distribution every 100 iterations(black cross).

## 3.2 Experiment Result

### 3.2.1 Result of Particle Filter

Fig 11 shows that even the odometry drifts a lot, the particle filter estimates a better pose. The estimation is less accurate when the robot is turning. The reason might be that the particles are more divergent due to bigger $\Delta\theta$ and $\Delta\alpha$ in equation 14 and equation 16, which leads to larger covariance in $\varepsilon_1$ and $\varepsilon_3$ in equation 18. The result can also be seen in Fig. 12, where particles are more spread out when the robot is turning.

In the lecture, we learned that for particle filter, the more particles are used, the more accurate the estimation, Table 2 shows the same result. The error and the numbers of collision decrease as the

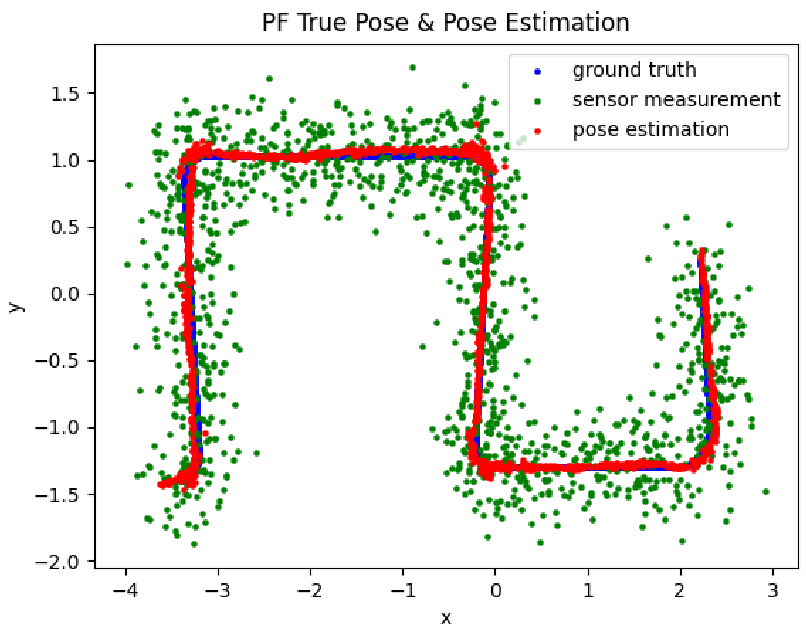


Figure 13: 2D plot of the estimated poses(red dots) versus true poses(blue dots), noisy sensor measurements(green dots)

| # of particles | error | execute time | # of collision |
|---|---|---|---|
| 100 | 152.65 | 21.472 | 78 |
| 200 | 149.118 | 31.234 | 52 |
| 500 | 146.569 | 68.145 | 48 |
| 1000 | 145.472 | 134.88 | 23 |

Table 2: Comparison of different number of particles.

particle increase. However, the execution time increase which make the particle filter less efficient in real-time application.

In Fig. 12, the particles spread out in a long shape when the robot is going in a straight line. This phenomenon is due to the fact that when the robot is moving forward without turning, $\Delta s$ in equation 15 will be big, $\Delta\theta$ and $\Delta\alpha$ in equation 14 and 16 will be small. Leads to small $\varepsilon_1$ and $\varepsilon_3$ and big $\varepsilon_2$ in equation 18, which make the particles distribute in a long shape along the moving direction.

### 3.2.2 Result of Extended Kalman Filter

The overall pose estimation of using Extended Kalman Filter is quite acceptable. For each step we got the norm of squared error between the true pose and the estimated pose approximately 0.3. In addition, from Fig. 15 we can observe that the posterior covariances (black eclipses) are maintained within a certain magnitude, which is good since it indicates that our estimation uncertainty is under control and our pose estimation is accurate and reliable. On the contrary, Fig. 16 shows a 2D plot of the estimation covariances with no measurement correction. As observed, the covariance eclipses keep growing since there's no measurement correction to reduce the covariances, meaning the estimation uncertainty grows with time.

However, the number of collisions between the estimated poses and the obstacles are quite large. We ran the algorithm for 10 times and got 112 as the average number of collisions using EKF localization, as shown in Table 3. In fact, these true poses didn't hit into obstacles in the real world. Hence, the estimated poses are not as accurate as the ones particle filter estimated.

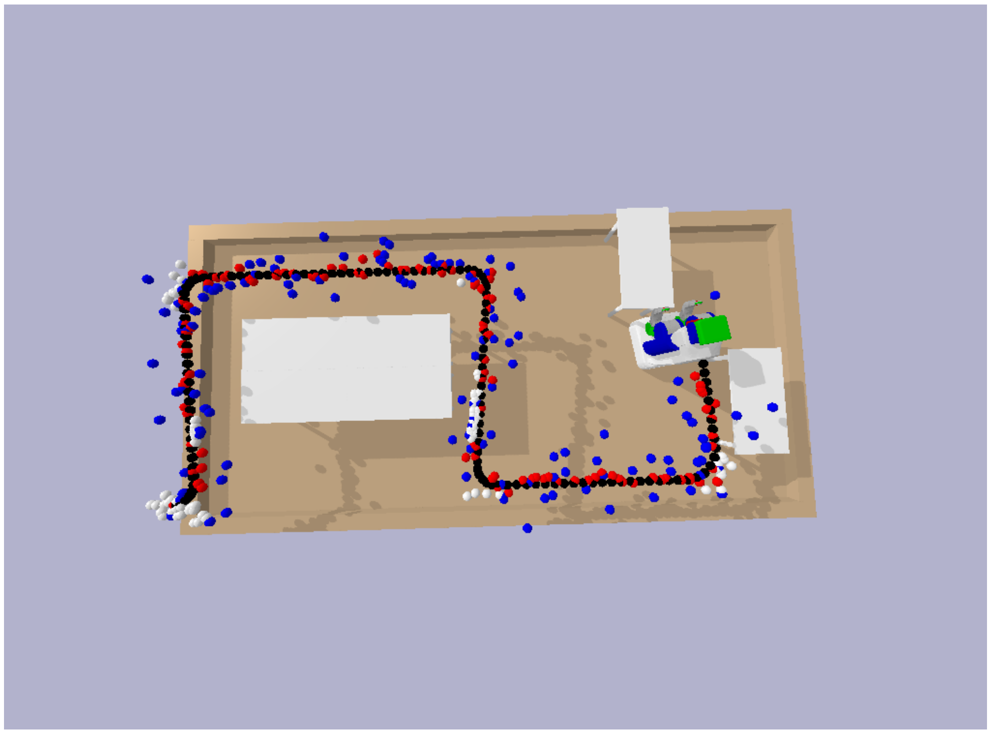


Figure 14: Simulation results of running EKF localization. The red balls are the estimated poses with no collision, white balls are estimated poses with collision, black balls are true poses, and blue balls are noisy measurements

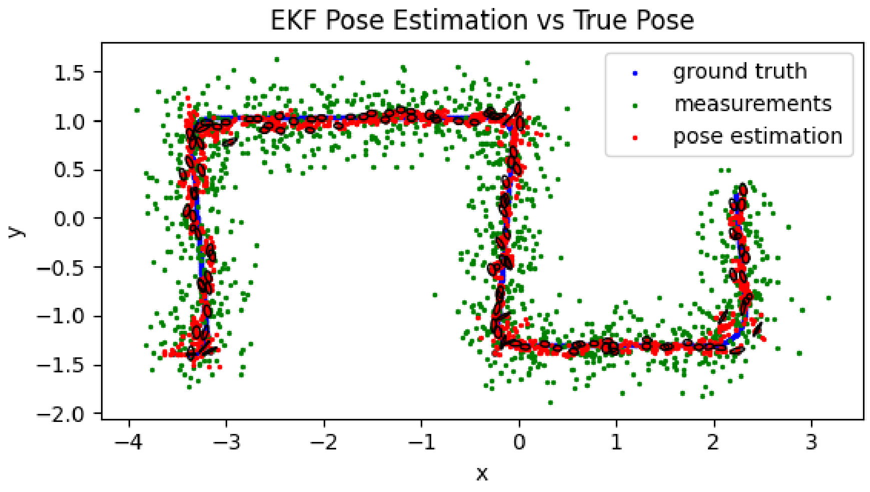


Figure 15: 2D plot of the estimated poses(red dots) versus true poses(blue dots), noisy measurements(green dots) and posterior covariance(eclipses)

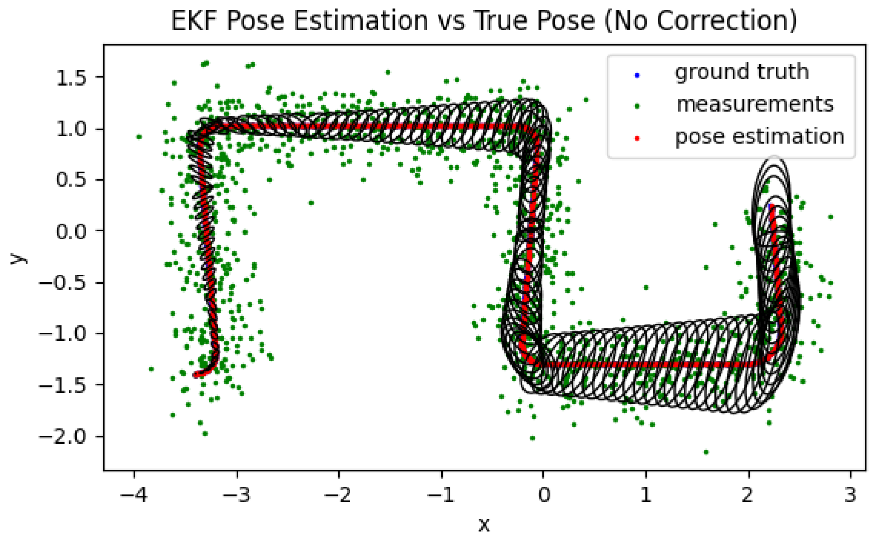


Figure 16: 2D plot of the estimated poses(red dots) versus true poses(blue dots), noisy measurements(green dots) and posterior covariance(eclipses) with no measurement corrections.

| Filter Type | Error | # of Collision | Execute time |
|---|---|---|---|
| Particle Filter | 150.97 | 70 | 18.65 |
| Extended Kalman Filter | 163.19 | 112 | 6.36 |

Table 3: Comparison of PF and EKF in terms of error, number of collisions in estimated poses, and execution time (average of 10 runs).

## 3.3 Discussion

Table 3 show that EKF has a higher efficiency than PF, which is consistent with the lecture. The reason is that EKF only performs matrix multiplications, while Particle Filter has to sample and apply actions to every particle. The time and space complexity of Particle Filters is proportional to the number of particles used in a particle set. Thus, Particle Filter would be a less efficient algorithm compared to EKF.

In terms of the estimation error and the number of collisions, PF performs better than EKF. There might be several reasons: 1) EKF is a linear approximation of a non-linear model, the approximation might not be accurate at some specific state, especially when the kinematics is highly non-linear, then the performance would be very poor. 2) Kalman filters(including EK, EKF, UKF), unlike PF, are based on a strong assumption that the state distribution and all uncertainty should be Gaussian distributions, and all noise should be zero mean. However, it is seldom true in real world situation, this would also lead to poor performance.

In Fig. 18, there are more points with collision in EKF estimation than those of PF estimation (white balls are more than blue balls). This indicates that, in our case, Particle Filter performs better than Extended Kalman Filter in terms of accuracy.

## 4 Conclusion

The goal of this project was to realize two kinds of probablistic filtering methods that track the states of a mobile robot, and compare the performance of the two localization methods. Overall, the Particle

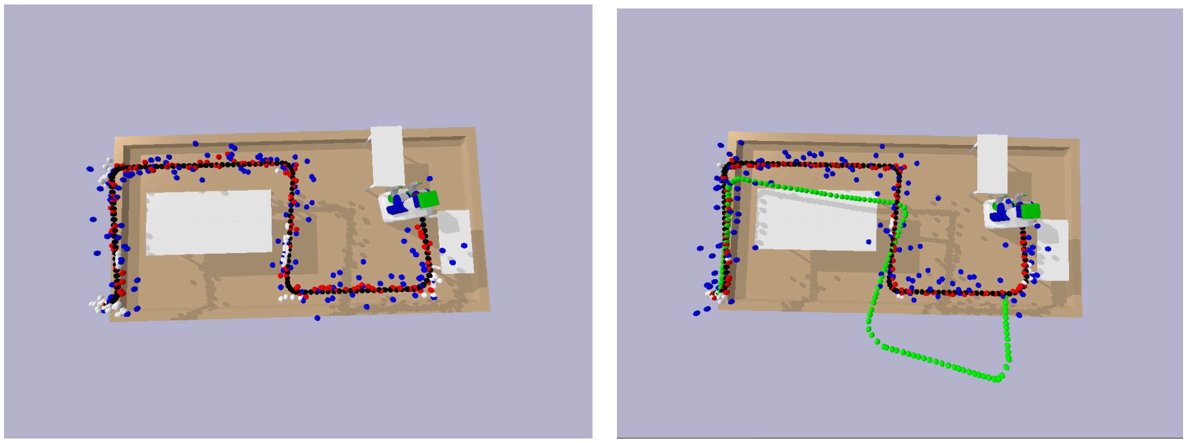


Figure 17: Simulation results of running two filters. The left figure is the result of running EKF localization and the right figure is the result of running particle filter. The red balls are the estimated poses with no collision, white balls are estimated poses with collision, green balls are odometry poses, black balls are true poses, and blue balls are noisy measurements

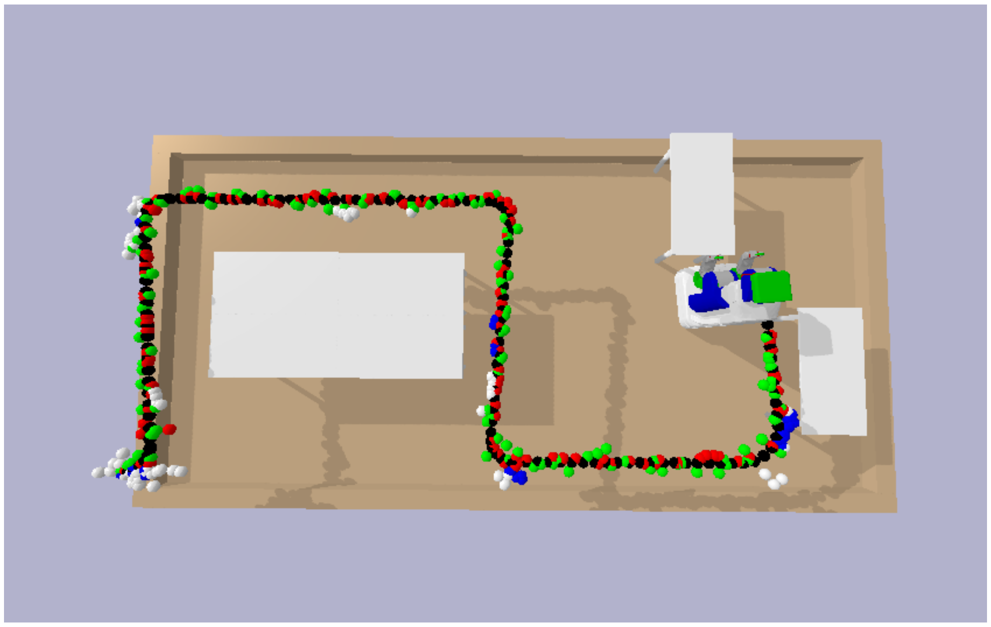


Figure 18: Simulation results of running EKF and PF together, black balls are ground truth poses, red balls are PF estimation without collision, blue balls are PF estimation with collision, green balls are EKF estimation without collision, white ball are EKF estimation with collision.

| Filter Type | Advantages | Disadvantages |
|---|---|---|
| Particle Filters | Can handle non-linear model<br>Can estimate arbitrary distribution | Time Consuming<br>Risk of diversity loss |
| Extended Kalman Filters | Can handle non-linear model<br>More efficient in time and memory | Less accurate if highly non-linear<br>Limited to Gaussian distribution |

Table 4: Advantages and disadvantages of Particle Filters and Extended Kalman Filters

| Filter Type | Computational cost | Type of dynamic model | Uncertainty distribution |
|---|---|---|---|
| Particle Filter | High, the more particles, the higher cost | linear & non-linear model | not limited to gaussian distribution |
| Extended Kalman Filter | Low, only matrix multiplication | linear & non-linear model | Gaussian distribution |

Table 5: Comparison of PF and EKF in terms of computational cost, types of dynamic model, and uncertainty distribution

Filter estimates the pose more precisely since it gets less collision estimations compared with the results from running Extended Kalman Filter. Though, both Extended Kalman Filters and Particle Filters are powerful state estimators that solve lots of real-world localization problems and both algorithms are relatively simple to implement. Two filters have its own advantages and disadvantages and can be selected under different scenarios, including the kinematics of the systems, the specification of the performance in accuracy and efficiency, and the computation resources of the onboard computer.

Ultimately, each member of the project team advanced their knowledge of all aspects of odometry, sensor uncertainty, Particle Filter, and Extended Kalman Filter in general and will apply these skills to all future engineering endeavors.

## References

[1] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics,” pp. 132–134, 2005.

[2] P. Gaskell, “Localization, particle filter  action model, lecture notes for rob 550.” University of Michigan, 2021.

[3] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics,” pp. 96–113, 2005.

[4] P. Gaskell, “Localization  sensor model, lecture notes for rob 550.” University of Michigan, 2021.

[5] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics,” pp. 118–129, 2005.

[6] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics,” pp. 54–64, 2005.

[7] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics,” pp. 203–215, 2005.