An Introduction to EKF SLAM and FastSLAM

Hao Dong

1. Introduction

Simultaneous localization and mapping (SLAM) refers to building the map and localizing the robot in an environment at the same time. SLAM is the basis for a robot to achieve fully autonomous navigation. Therefore, it has a wide range of applications. Vacuum cleaner uses SLAM to navigate in the room and prevent collision with other objects in the environment. Unmanned air vehicles use SLAM for state estimation and robust control. SLAM can even be used in space, where rover uses SLAM for terrain mapping and safe move.

The history of SLAM can date back to the mid-eighties, when Smith et al. and Durrant-Whyte first described geometric uncertainty and relationships between features or landmarks. Kalman filter based approaches for SLAM appeared in mid-nineties. From 2000, wide research interests in SLAM have started. There are three main paradigms for SLAM problems now, which are Kalman filter based, particle filter based and graph-based. In this report, I will mainly focus on Kalman filter based and particle filter based approach.

2. Theory

2.1 SLAM problem

Estimating the robot's poses given the map, which is localization problem, and estimating the map given the robot's poses, which is mapping problem, are easy to solve separately. But estimating the robot's poses and the map at the same time is a hard problem. Since both the robot path and map are unknown, their estimations are correlated. Besides, we don't know the matches between observations and the map. Wrong matches will have catastrophic consequences.

According to [1], we can define SLAM problem as given the robot's controls $u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$ and observations $z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$, we want to estimate the map m of the environment and also the path $x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$ of the robot. We can describe it in a probabilistic formula as

$$p(x_{0:T}, m \mid z_{1:T}, u_{1:T})$$

This formula can be depicted as the graphical model in Figure 1 [1].

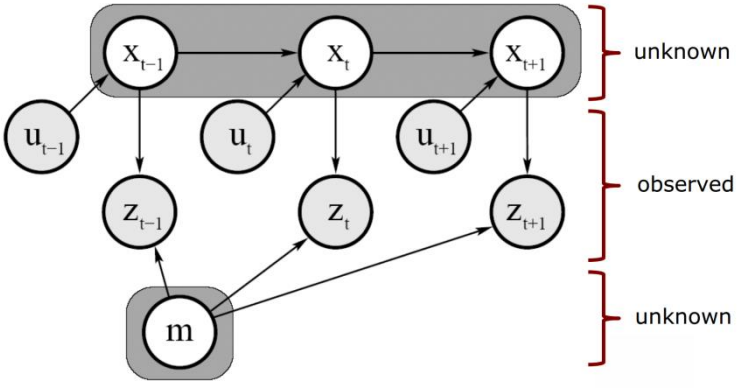


Figure 1: The graphical model of SLAM problem

2.2 Motion Model and Observation Model

To solve SLAM problem, we need to define the motion model and observation model first. Motion model is used to describe the relative motion of the robot given input u. We can describe it as a posterior probability

$$p(x_t \mid u_t, x_{t-1})$$

If the robot has wheel encoders, odometry-based model (Figure 2) can be used as motion model. Assume we have odometry information $u = (\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ and the robot moves from $(\bar{x}, \bar{y}, \bar{\theta})$ to $(\bar{x}', \bar{y}', \bar{\theta}')$, then we can describe this relation as

$$\begin{aligned} \delta_{trans} &= \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2} \\ \delta_{rot1} &= \operatorname{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \\ \delta_{rot2} &= \bar{\theta}' - \bar{\theta} - \delta_{rot1} \end{aligned}$$

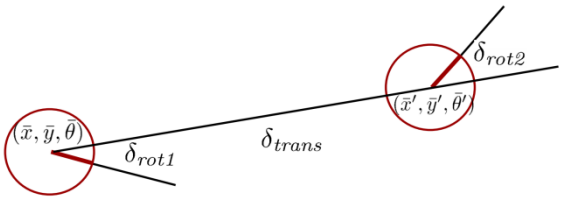


Figure 2: Odometry-based motion model

If there is no wheel encoder, velocity-based model (Figure 3) can be used as motion model. Assume we have velocity information and the robot moves from $(\bar{x}, \bar{y}, \bar{\theta})$ to $(\bar{x}', \bar{y}', \bar{\theta}')$, then we can describe this relation as

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \omega\Delta t + \gamma\Delta t \end{pmatrix}$$

$$u = (v, \omega)^T$$

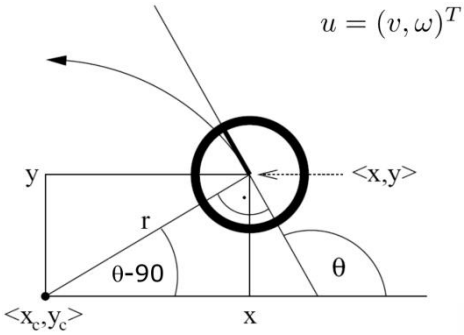


Figure 3: Velocity-based motion model

Observation model is used to relate measurements with the robot's pose. We can describe it as a posterior probability

$$p(z_t \mid x_t)$$

We always assume that individual measurements are independent given the robot position, which can be described as

$$p(z_t \mid x_t, m) = \prod_{i=1}^{k} p(z_t^i \mid x_t, m)$$

Assuming the robot is equipped with a range-bearing sensor to perceive landmarks. The current

robot's pose is

$$(x, y, \theta)^T$$

and the robot get a range-bearing observation

$$z_t^i = \left(r_t^i, \phi_t^i\right)^T$$

of feature j at location

$$(m_{j,x}, m_{j,y})^T$$

then we can describe this relation as

$$\begin{pmatrix} r_t^i \\ \phi_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x}-x)^2 + (m_{j,y}-y)^2} \\ \text{atan2}(m_{j,y}-y, m_{j,x}-x) - \theta \end{pmatrix} + Q_t$$

where Q_t is the noise term.

2.3 Bayes Filter

Bayes filter is used to estimate the state of the system given the observations and controls. In SLAM problem, it is

$$p(x_{0:T}, m \mid z_{1:T}, u_{1:T})$$

where $x_{0:T}$ and m are the states, $z_{1:T}$ are the observations and $u_{1:T}$ are the controls. There are two steps in a Bayes filter process which are prediction step and correction step. In prediction step, motion model is used to propagate the state, which can be described as

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\, dx_{t-1}$$

The prediction step will increase the covariance of the state. In correct step, the observation model is used to correct the error, which can be described as

$$bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$$

Bayes filter is a general framework for recursive state estimation. If the motion and observation model are non-linear with gaussian noise, we can use Extended Kalman Filter (EKF) realization.

2.4 Extended Kalman Filter

The extended Kalman filter (EKF) overcomes the linearity assumption of Kalman filter (KF) and calculates an approximation to the true belief. Figure 4 [1] is the extended Kalman filter algorithm. In EKF algorithm, g and h are nonlinear functions, G_t and H_t are the Jacobian matrix.

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 μ_t, Σ_t

Figure 4: The extended Kalman filter (EKF) algorithm

2.5 EKF SLAM

In EKF SLAM algorithm, we need to estimate the robot pose and the coordinates of all landmarks encountered. Therefore, we should include the landmark coordinates into the state vector. We can write the combined vector as

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} = (x \quad y \quad \theta \quad m_{1,x} \quad m_{1,y} \quad s_1 \quad \dots \quad m_{N,x} \quad m_{N,y} \quad s_N)^T$$

where x, y and θ are robot's position at time t, $m_{i,x}$ and $m_{i,y}$ are the position of the i-th landmark for $i = 1, \dots, N$, and s_i is its signature. Thus, for a map with N landmarks, the state vector has $3N + 3$ dimension. EKF SLAM calculates the online posterior

$$p(y_t \mid z_{1:t}, u_{1:t})$$

Like Bayes filter, there are two steps in a EKF SLAM process, which are prediction step and correction step. In prediction step, the state is updated based on the robot's motion model

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

Only the first three elements in the state, which represent the robot's position, is updated. Therefore, we need to map from the motion in the plane to the 2N+3 dimensional space by a matrix F_x.

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} x \\ y \\ \theta \\ \vdots \end{pmatrix} + \underbrace{\begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \dots 0}_{2N cols} \\ 0 & 1 & 0 & 0 \dots 0 \\ 0 & 0 & 1 & 0 \dots 0 \end{pmatrix}^T}_{F_x^T} \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix}}_{g(u_t, x_t)}$$

Since function g only affects the robot's position and has no influence on landmarks, we can write the Jacobian matrix G_t as

$$G_t = \begin{pmatrix} G_T^x & 0 \\ 0 & I \end{pmatrix}$$

where

$$\begin{aligned} G_t^x &= \frac{\partial}{\partial (x, y, \theta)^T} \left[\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix} \right] \\ &= I + \frac{\partial}{\partial (x, y, \theta)^T} \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix} \\ &= I + \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t}\cos\theta + \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & -\frac{v_t}{\omega_t}\cos\theta + \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ 0 & 1 & -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Then we put g and Gt in the EKF prediction step and get the whole prediction step of EKF SLAM as in Figure 5 [2].

EKF_SLAM_Prediction$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, R_t)$**:**

2: $F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

3: $\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$

4: $G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$

5: $\bar{\Sigma}_t = G_t\,\Sigma_{t-1}\,G_t^T + \underbrace{F_x^T\,R_t^x\,F_x}_{R_t}$

Figure 5: The prediction step of EKF-SLAM algorithm

Next we move to the correction step. Given an observation

$$z_t^i = (r_t^i, \phi_t^i)^T$$

if the landmark has not been observed before, we can add it in the map by the following observation model

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i\,\cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i\,\sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$$

where $\bar{\mu}_{j,x}$, $\bar{\mu}_{j,y}$ is the observed location of landmark j, $\bar{\mu}_{t,x}$, $\bar{\mu}_{t,y}$ is the estimated robot's location. The expected observation given current state estimate is

$$\begin{aligned} \hat{z}_t^i &= \begin{pmatrix} \sqrt{q} \\ \mathrm{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix} \\ &= h(\bar{\mu}_t) \end{aligned}$$

where

$$\begin{aligned} \delta &= \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix} \\ q &= \delta^T\delta \end{aligned}$$

Next, we compute the Jacobian

$$\begin{aligned} {}^{\text{low}}H_t^i &= \frac{\partial h(\bar{\mu}_t)}{\partial \bar{\mu}_t} \\ &= \frac{1}{q}\begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix} \end{aligned}$$

and map it to the high dimensional space

$$H_t^i = {}^{\text{low}}H_t^i\,F_{x,j}$$

where

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 0 & \underbrace{0 \cdots 0}_{2N-2j} \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$$

Then in the same way as in prediction step, we put h and H_t in the EKF correction step and get

the whole correction step of EKF SLAM as in Figure 6 [2].

EKF_SLAM_Correction

6: $Q_t = \begin{pmatrix} {\sigma_r}^2 & 0 \\ 0 & {\sigma_\phi}^2 \end{pmatrix}$

7: *for all observed features* $z_t^i = (r_t^i, \phi_t^i)^T$ *do*

8: $\quad j = c_t^i$

9: $\quad$ *if landmark* j *never seen before*

10: $\quad \begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$

11: $\quad$ *endif*

12: $\quad \delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$

13: $\quad q = \delta^T \delta$

14: $\quad \hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix}$

15: $F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 0 & \underbrace{0 \cdots 0}_{2N-2j} \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

16: $H_t^i = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x \end{pmatrix} F_{x,j}$

17: $K_t^i = \bar{\Sigma}_t \, H_t^{iT} (H_t^i \, \bar{\Sigma}_t \, H_t^{iT} + Q_t)^{-1}$

18: $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$

19: $\bar{\Sigma}_t = (I - K_t^i \, H_t^i) \, \bar{\Sigma}_t$

20: *endfor*

21: $\mu_t = \bar{\mu}_t$

22: $\Sigma_t = \bar{\Sigma}_t$

23: *return* μ_t, Σ_t

Figure 6: The correction step of EKF-SLAM algorithm

2.6 Particle Filter

Particle filter is a non-parametric approach which models the distribution by samples. In prediction step, it draws particles from proposal distribution. In correction step, the importance weights are calculated by the ratio of target and proposal. Figure 7 [1] is the particle filter algorithm.

1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$)**:**

2: $\quad \bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$

3: $\quad$ for $m = 1$ to M do

4: $\quad\quad$ sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$

5: $\quad\quad$ $w_t^{[m]} = p(z_t \mid x_t^{[m]})$

6: $\quad\quad$ $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$

7: $\quad$ endfor

8: $\quad$ for $m = 1$ to M do

9: $\quad\quad$ draw i with probability $\propto w_t^{[i]}$

10: $\quad\quad$ add $x_t^{[i]}$ to $\mathcal{X}_t$

11: $\quad$ endfor

12: $\quad$ return $\mathcal{X}_t$

Figure 7: The particle filter algorithm

2.7 FastSLAM

In FastSLAM algorithm, we use the particle set only to model the robot's path, each sample is a path hypothesis. For each sample, we can compute an individual map of landmarks. By the factorization of the SLAM posterior we get

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t})p(m_{1:M}|x_{0:t}, z_{1:t})$$
$$= p(x_{0:t}|z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i|x_{0:t}, z_{1:t})$$

Each landmark is represented by a 2×2 EKF. Therefore, each particle has to maintain M individual EKFs. Figure 8 [2] is the complete FastSLAM algorithm.

FastSLAM1.0_known_correspondence$(z_t, c_t, u_t, \mathcal{X}_{t-1})$**:**

for $k = 1$ *to* N *do* // *loop over all particles*

Let $\left\langle x_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} \right\rangle, \ldots \right\rangle$ *be particle* k *in* $\mathcal{X}_{t-1}$

$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ // *sample pose*

$j = c_t$ // *observed feature*

if feature j *never seen before*

$\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ // *initialize mean*

$H = h'(\mu_{j,t}^{[k]}, x_t^{[k]})$ // *calculate Jacobian*

$\Sigma_{j,t}^{[k]} = H^{-1}\, Q_t\, (H^{-1})^T$ // *initialize covariance*

$w^{[k]} = p_0$ // *default importance weight*

else

else

$\hat{z}^{[k]} = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // *measurement prediction*

$H = h'(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // *calculate Jacobian*

$Q = H\, \Sigma_{j,t-1}^{[k]}\, H^T + Q_t$ // *measurement covariance*

$K = \Sigma_{j,t-1}^{[k]}\, H^T\, Q^{-1}$ // *calculate Kalman gain*

$\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z}^{[k]})$ // *update mean*

$\Sigma_{j,t}^{[k]} = (I - K\, H)\Sigma_{j,t-1}^{[k]}$ // *update covariance*

$w^{[k]} = |2\pi Q|^{-\frac{1}{2}}\ \exp\left\{-\frac{1}{2}(z_t - \hat{z}^{[k]})^T\right.$
$\left. Q^{-1}\,(z_t - \hat{z}^{[k]})\right\}$ // *importance factor*

endif

for all unobserved features j' *do*

$\langle \mu_{j',t}^{[k]}, \Sigma_{j',t}^{[k]} \rangle = \langle \mu_{j',t-1}^{[k]}, \Sigma_{j',t-1}^{[k]} \rangle$ // *leave unchanged*

endfor

endfor

$\mathcal{X}_t = \text{resample}\left(\left\langle x_t^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \right\rangle, \ldots, w^{[k]} \right\rangle_{k=1,\ldots,N}\right)$

return $\mathcal{X}_t$

Figure 8: The FastSLAM algorithm

3. Results

The implementation is based on the framework provided by Robot Mapping course at University of Freiburg [2]. The complete code can be found at https://github.com/donghao51/SLAM-intro.

For EKF SLAM, in prediction step, we update the belief concerning the robot pose according to the motion model.

```
function [mu, sigma] = prediction_step(mu, sigma, u)
% mu: 2N+3 x 1 vector representing the state mean
% sigma: 2N+3 x 2N+3 covariance matrix
% u: odometry reading (r1, t, r2)

% Compute the new mu based on the noise-free (odometry-based) motion model
Fx = [eye(3) zeros(3, length(mu)-3)];

mu = mu + Fx' * [u.t*cos(normalize_angle(mu(3) + u.r1));
                 u.t*sin( normalize_angle(mu(3) + u.r1));
                 normalize_angle(u.r1 + u.r2)];
```

```
% Compute the 3x3 Jacobian Gx of the motion model
Gx = [0, 0, -u.t*sin(mu(3) + u.r1);
      0, 0, u.t*sin(mu(3) + u.r1);
      0, 0, 0 ];

% Construct the full Jacobian G
G = eye(length(mu)) + (Fx' * Gx * Fx);

% Motion noise
motionNoise = 0.1;
R3 = [motionNoise, 0, 0;
     0, motionNoise, 0;
     0, 0, motionNoise/10];
R = zeros(size(sigma,1));
R(1:3,1:3) = R3;

% Compute the predicted sigma after incorporating the motion
sigma = G*sigma*G' + R;

end
```

Figure 9: The prediction step of EKF-SLAM algorithm

In correction step, we update the belief, i. e., mu and sigma after observing landmarks, according to the sensor model.

```
function [mu, sigma, observedLandmarks] = correction_step(mu, sigma, z, observedLandmarks)
% mu: 2N+3 x 1 vector representing the state mean.
% The first 3 components of mu correspond to the current estimate of the robot pose [x; y; theta]
% The current pose estimate of the landmark with id = j is: [mu(2*j+2); mu(2*j+3)]
% sigma: 2N+3 x 2N+3 is the covariance matrix
% z: struct array containing the landmark observations.
% Each observation z(i) has an id z(i).id, a range z(i).range, and a bearing z(i).bearing
% The vector observedLandmarks indicates which landmarks have been observed
% at some point by the robot.
% observedLandmarks(j) is false if the landmark with id = j has never been observed before.

% Number of measurements in this time step
m = size(z, 2);

% Z: vectorized form of all measurements made in this time step: [range_1; bearing_1; range_2;
bearing_2; ...; range_m; bearing_m]
% ExpectedZ: vectorized form of all expected measurements in the same form.
% They are initialized here and should be filled out in the for loop below
Z = zeros(m*2, 1);
expectedZ = zeros(m*2, 1);

% Iterate over the measurements and compute the H matrix
% (stacked Jacobian blocks of the measurement function)
% H will be 2m x 2N+3
H = [];

for i = 1:m
    % Get the id of the landmark corresponding to the i-th observation
    landmarkId = z(i).id;
    % If the landmark is obeserved for the first time:
    if(observedLandmarks(landmarkId)==false)
        % Initialize its pose in mu based on the measurement and the current robot pose:
        mu(3+landmarkId*2-1) = mu(1) + z(i).range * cos(normalize_angle(z(i).bearing+mu(3)));
        mu(3+landmarkId*2)   = mu(2) + z(i).range * sin(normalize_angle(z(i).bearing+mu(3)));
        % Indicate in the observedLandmarks vector that this landmark has been observed
        observedLandmarks(landmarkId) = true;
    endif

    % Add the landmark measurement to the Z vector
    Z(2*i-1:2*i) = [z(i).range;z(i).bearing]
    % Use the current estimate of the landmark pose
    % to compute the corresponding expected measurement in expectedZ:
    delta = [mu(3+landmarkId*2-1) - mu(1); mu(3+landmarkId*2) - mu(2)];
    q = delta'*delta;
    expectedZ(2*i-1:2*i) = [sqrt(q); normalize_angle(atan2(delta(2),delta(1)) - mu(3,1))]
    % Compute the Jacobian Hi of the measurement function h for this observation
    N = (length(mu) - 3) / 2;
    Fx = [eye(3) zeros(3, 2*N); zeros(2, 2*landmarkId+1) eye(2) zeros(2, 2*N-2*landmarkId)];
    Hi = 1/q * [-sqrt(q)*delta(1), -sqrt(q)*delta(2), 0, sqrt(q)*delta(1), +sqrt(q)*delta(2);
                delta(2), -delta(1), -q, -delta(2), delta(1) ] * Fx;
```

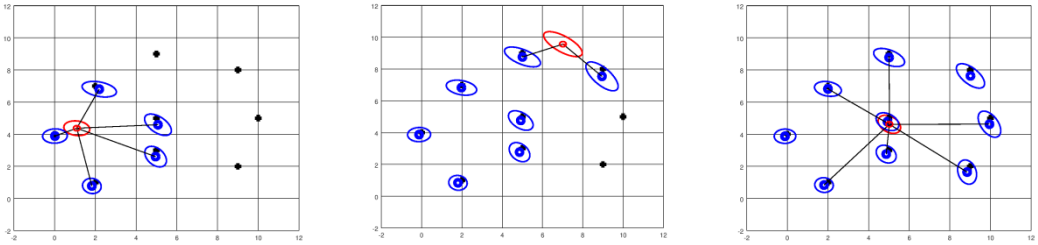
```
    % Augment H with the new Hi
    H = [H;Hi];
endfor

% Construct the sensor noise matrix Q
Q = 0.01*eye(2*m);
% Compute the Kalman gain
K = sigma*H'*inv(H*sigma*H'+Q);
% Compute the difference between the expected and recorded measurements.
Zdiff = normalize_all_bearings(Z - expectedZ);
% Finish the correction step by computing the new mu and sigma.
mu = mu + K*Zdiff;
sigma = (eye(2*N+3) - K*H)*sigma;

end
```

Figure 10: The correction step of EKF-SLAM algorithm

Figure 11 depicts some example images of the state of the EKF SLAM at different time indices. The black crosses are the ground-truth landmark positions in the environment. The blue ellipses are the estimated states of landmarks and the red ellipse is the estimated states of the robot. We can find that the estimation is quite accurate and the RMSE of landmark position is 0.20191.



t = 50 t = 150 t = 330

Figure 11: Example images of the state of the EKF SLAM at certain time indices

For FastSLAM, in prediction step, we update the particles by drawing from the motion model.

```
function particles = prediction_step(particles, u, noise)
% Use u.r1, u.t, and u.r2 to access the rotation and translation values
% which have to be pertubated with Gaussian noise.
% The position of the i-th particle is given by the 3D vector
% particles(i).pose which represents (x, y, theta).

% noise parameters
r1Noise = noise(1);
transNoise = noise(2);
r2Noise = noise(3);

numParticles = length(particles);

for i = 1:numParticles
  % append the old position to the history of the particle
  particles(i).history{end+1} = particles(i).pose;
  % sample a new pose for the particle
  r1 = normrnd(u.r1, r1Noise);
  r2 = normrnd(u.r2, r2Noise);
  trans = normrnd(u.t, transNoise);
  particles(i).pose(1) = particles(i).pose(1) + trans*cos(particles(i).pose(3) + r1);
  particles(i).pose(2) = particles(i).pose(2) + trans*sin(particles(i).pose(3) + r1);
  particles(i).pose(3) = normalize_angle(particles(i).pose(3) + r1 + r2);
end

end
```

Figure 12: The prediction step of FastSLAM algorithm

In correction step, we weight the particles according to the current map of the particle and the landmark observations z.

```
function particles = correction_step(particles, z)
% z: struct array containing the landmark observations.
% Each observation z(j) has an id z(j).id, a range z(j).range, and a bearing z(j).bearing
% The vector observedLandmarks indicates which landmarks have been observed
% at some point by the robot.

% Number of particles
numParticles = length(particles);
% Number of measurements in this time step
m = size(z, 2);
% Construct the sensor noise matrix Q_t (2 x 2)
Qt = 0.01*eye(2);
% process each particle
for i = 1:numParticles
  robot = particles(i).pose;
  % process each measurement
  for j = 1:m
    % Get the id of the landmark corresponding to the j-th observation
    % particles(i).landmarks(l) is the EKF for this landmark
    l = z(j).id;
    % If the landmark is observed for the first time:
    if (particles(i).landmarks(l).observed == false)
      % Initialize its position based on the measurement and the current robot pose:
      particles(i).landmarks(l).mu = [robot(1) + z(j).range*cos(z(j).bearing + robot(3));
                                      robot(2) + z(j).range*sin(z(j).bearing + robot(3))];
      % get the Jacobian with respect to the landmark position
      [h, H] = measurement_model(particles(i), z(j));
      % initialize the EKF for this landmark
      particles(i).landmarks(l).sigma = inv(H)*Qt*inv(H)';
      % Indicate that this landmark has been observed
      particles(i).landmarks(l).observed = true;
    else
      % get the expected measurement
      [expectedZ, H] = measurement_model(particles(i), z(j));
      % compute the measurement covariance
      Q = H*particles(i).landmarks(l).sigma*H' + Qt;
      % calculate the Kalman gain
      K = particles(i).landmarks(l).sigma*H' * inv(Q);
      % compute the error between the z and expectedZ (remember to normalize the angle)
      z_diff = [z(j).range - expectedZ(1); normalize_angle(z(j).bearing - expectedZ(2))];
      % update the mean and covariance of the EKF for this landmark
      particles(i).landmarks(l).mu = particles(i).landmarks(l).mu + K*z_diff;
        particles(i).landmarks(l).sigma = (eye(2) - K*H)*particles(i).landmarks(l).sigma;
      % compute the likelihood of this observation, multiply with the former weight
      % to account for observing several features in one time step
      particles(i).weight = particles(i).weight*(1/sqrt(det(2*pi*Q)))
                            *exp(-1/2*z_diff'*inv(Q)*z_diff);
    end
  end % measurement loop
end % particle loop

end
```

Figure 13: The prediction step of FastSLAM algorithm

Figure 14 depicts the final estimated states of the FastSLAM algorithm. The black crosses are the ground-truth landmark positions in the environment. The blue ellipses are the estimated states of landmarks and the red ellipse is the estimated states of the robot. The red line is the trajectory of the robot. The green points are particle samples. We can find that the estimation is quite accurate and the RMSE of landmark position is 0.47435.

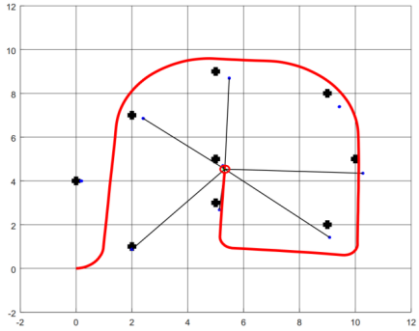


Figure 14: The estimated state of the FastSLAM algorithm

4. Summary

From the result we can find that EKF SLAM can achieve good performance in small scale scenes. But it has cubic complexity depending on the dimension of the landmarks in the environment. If there are N landmarks, the cost complexity per step and the memory consumption would be $O(N^2)$. Therefore, EKF SLAM is computationally intractable in large scale scenes.

The complexity of FastSLAM is $O(NlogM)$, where N is the number of particles and M is the number of map features. It scales well up to 1 million features and is robust to ambiguities in the data association. Compared to EKF approach, it can also work well with high non-linear models.

References

[1] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. The MIT Press, 2005.

[2] W. Burgard, Robot Mapping course. University of Freiburg. http://ais.informatik.uni-freiburg.de/teaching/ws20/mapping/