Robotic Navigation and Exploration Lab 3

Fast-SLAM

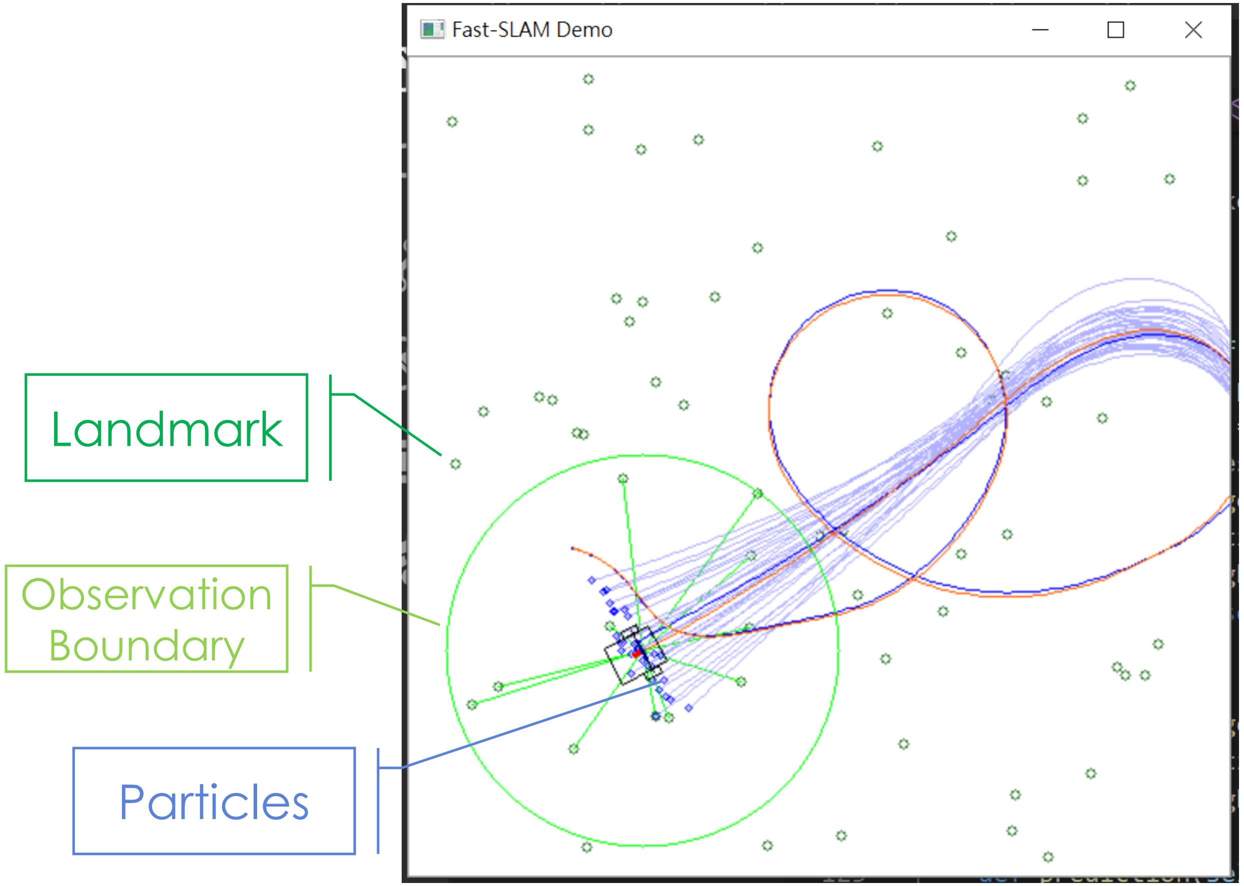
Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

Requirement

- Python 3.X (Suggest to install miniconda/anaconda)
- Numpy
- Opencv-Python

Run The Code

- Use WASD to control the car.



Fast-SLAM

- Steps of Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the distribution of each landmark $(\mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)})$ via measurement z_k.

$$Q = H\Sigma_{j,t-1}^{(i)} H^T + R, \qquad K_t = \Sigma_{j,t-1}^{(i)} H^T Q^{-1}$$
$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_k \left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)}) \right)$$
$$\Sigma_{j,t}^{(i)} = (I - K_t H)\Sigma_{j,t-1}^{(i)}$$

3. Update the importance weight of particles.

$$w^{(i)} \sim |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}\left(z_k - h\left(\mu_{j,t-1}^{(i)}, x_t^{(i)}\right)\right)^T Q^{-1} \left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)\}$$

4. Resampling.

Code Structure

- Particle
  - init_pos
  - deepcopy: Copy the memory of whole particle.
  - sampling: Sample from motion model.
  - observation_model
  - multi_normal
  - compute_H: Compute linearize observation model.
  - update_landmark: Update one landmark given the observation.
  - update_obs: Update all landmarks from the observation and return likelihood.
- Particle Filter
  - init_pf
  - prediction: Sample the next pose of each particle.
  - update_obs: Update the map and particle weight given the observation.
  - resample: Compute Neff and resample.

Code Structure

- In each iteration, we get the control information of car, landmark observation and landmark ids.

$$u = (v, \omega, \Delta t) \qquad z = [(r_{id1}, \theta_{id1}), (r_{id2}, \theta_{id2}), \dots] \qquad detect\_ids = [id1, id2, \dots]$$

- For SLAM process, we first prediction the next pose by control information, update the map, and resample the particle.

```
###################################
# SLAM Algorithm
###################################
pf.prediction(u)
pf.update_obs(z, detect_ids)
pf.resample()
```

Fast-SLAM

- Steps of Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the distribution of each landmark $(\mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)})$ via measurement z_k.

$$Q = H\Sigma_{j,t-1}^{(i)}H^T + R, \qquad K_t = \Sigma_{j,t-1}^{(i)}H^T Q^{-1}$$
$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_k\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)$$
$$\Sigma_{j,t}^{(i)} = (I - K_t H)\Sigma_{j,t-1}^{(i)}$$

3. Update the importance weight of particles.

$$w^{(i)} \sim |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}\left(z_k - h\left(\mu_{j,t-1}^{(i)}, x_t^{(i)}\right)\right)^T Q^{-1}\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)\}$$

4. Resampling.

Sample the Motion Model

```python
def sampling(self, control):
    v, w, delta_t = control
    v_hat = v + random.gauss(0, self.params[0]*v**2+self.params[1]*w**2)
    w_hat = w + random.gauss(0, self.params[2]*v**2+self.params[3]*w**2)
    w_rad = np.deg2rad(w_hat)
    g_hat = random.gauss(0, self.params[4]*v**2+self.params[5]*w**2)

    if w_hat != 0:
        x_next = self.pos[0] - (v_hat/w_rad)*np.sin(np.deg2rad(self.pos[2])) + (
            v_hat/w_rad)*np.sin(np.deg2rad(self.pos[2]+w_hat*delta_t))
        y_next = self.pos[1] + (v_hat/w_rad)*np.cos(np.deg2rad(self.pos[2])) - (
            v_hat/w_rad)*np.cos(np.deg2rad(self.pos[2]+w_hat*delta_t))
        yaw_next = self.pos[2] + w_hat*delta_t + g_hat
    else:
        x_next = self.pos[0] + v_hat * \
            np.cos(np.deg2rad(self.pos[2]))*delta_t
        y_next = self.pos[1] + v_hat * \
            np.sin(np.deg2rad(self.pos[2]))*delta_t
        yaw_next = self.pos[2] + g_hat

    self.pos = [x_next, y_next, yaw_next]
    self.path.append(self.pos)
    return self.pos
```

1: **Algorithm sample_motion_model_velocity**(u_t, x_{t-1}):

2: $\hat{v} = v + \mathbf{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$

3: $\hat{\omega} = \omega + \mathbf{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$

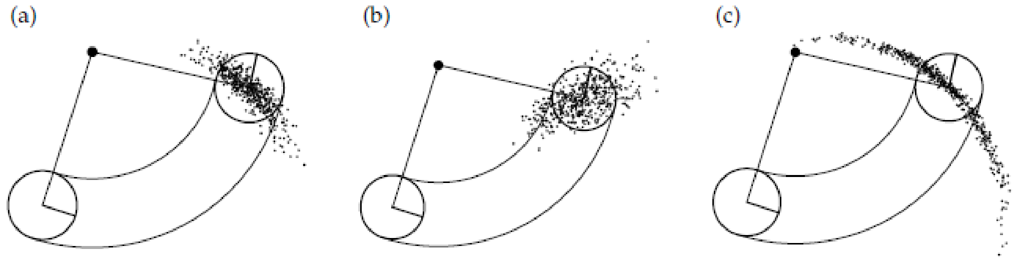
4: $\hat{\gamma} = \mathbf{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$

5: $x' = x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\Delta t)$

6: $y' = y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\Delta t)$

7: $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$

8: *return* $x_t = (x', y', \theta')^T$



Fast-SLAM

- Steps of Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the distribution of each landmark $(\mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)})$ via measurement z_k.

$$Q = H\Sigma_{j,t-1}^{(i)} H^T + R, \qquad K_t = \Sigma_{j,t-1}^{(i)} H^T Q^{-1}$$

$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_k \left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)}) \right)$$

$$\Sigma_{j,t}^{(i)} = (I - K_t H)\Sigma_{j,t-1}^{(i)}$$

3. Update the importance weight of particles.

$$w^{(i)} \sim |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}\left(z_k - h\left(\mu_{j,t-1}^{(i)}, x_t^{(i)}\right)\right)^T Q^{-1} \left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)\}$$

4. Resampling.

Update the Landmarks

- Initialize the covariance if the landmark is new and update the existing landmarks by EKF.

```python
def update_landmark(self, z, lid):
    if lid not in self.landmarks:
        # Add New Landmark
        c = np.cos(np.deg2rad(self.pos[2]+z[1]))
        s = np.sin(np.deg2rad(self.pos[2]+z[1]))
        mu = np.array([[self.pos[0] + z[0]*c],[self.pos[1] + z[0]*s]])
        sig = np.eye(2)*100
        self.landmarks[lid] = {"mu":mu, "sig":sig}
        p = 1.0
    else:
        # Update Old Landmark
        mu = self.landmarks[lid]["mu"]
        sig = self.landmarks[lid]["sig"]
        H = self.compute_H(mu[0,0], mu[1,0])
        Q = [redacted]
        K = [redacted]
        z_pre = self.observation_model(self.landmarks[lid])
        e = np.array([[z[0]-z_pre[0]],[z[1]-z_pre[1]]])
        self.landmarks[lid]["mu"] = [redacted]
        self.landmarks[lid]["sig"] = [redacted]
        p = self.multi_normal(np.array(z_pre).reshape(2,1),np.array(z).reshape(2,1), Q)
    return p
```

Linearize Observation model

```python
def compute_H(self, fx, fy):
    dx = fx - self.pos[0]
    dy = fy - self.pos[1]
    d2 = dx**2 + dy**2
    d = np.sqrt(d2)
    H = np.array([[dx / d, dy / d],
                  [-dy / d2, dx / d2]])
    return H
```

- Given observation model

$$z_i = \begin{bmatrix} \sqrt{q} \\ atan2(\delta_x, \delta_y) - \theta \end{bmatrix}, \delta = \begin{bmatrix} m_{i,x} - x \\ m_{i,y} - y \end{bmatrix}, q = \delta^T \delta$$

- Linearized the observation model :

  - $$H^i = \frac{\partial z_i}{\partial(x,y,\theta,m_{i,x},m_{i,y})} = \begin{bmatrix} \frac{\partial\sqrt{q}}{\partial x} & \frac{\partial\sqrt{q}}{\partial y} & \cdots \\ \frac{\partial atan2(\delta_x,\delta_y)}{\partial x} & \frac{\partial atan2(\delta_x,\delta_y)}{\partial y} & \cdots \end{bmatrix}$$

$$= \frac{1}{q} \begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix}$$

$$\frac{\partial\sqrt{q}}{\partial x} = \frac{1}{2}\frac{1}{\sqrt{q}} 2\delta_x(-1) = \frac{1}{q}(-\sqrt{q}\delta_x)$$

$$\frac{\partial}{\partial x}\operatorname{atan2}(y, x) = \frac{\partial}{\partial x}\arctan\left(\frac{y}{x}\right) = -\frac{y}{x^2+y^2},$$
$$\frac{\partial}{\partial y}\operatorname{atan2}(y, x) = \frac{\partial}{\partial y}\arctan\left(\frac{y}{x}\right) = \frac{x}{x^2+y^2}.$$

- Only Consider the Landmarks

$$H = \begin{bmatrix} \delta_x/\sqrt{q} & \delta_y/\sqrt{q} \\ -\delta_y/q & \delta_x/q \end{bmatrix}$$

Fast-SLAM

- Steps of Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the distribution of each landmark $(\mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)})$ via measurement z_k.

$$Q = H\Sigma_{j,t-1}^{(i)}H^T + R, \qquad K_t = \Sigma_{j,t-1}^{(i)}H^T Q^{-1}$$

$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_k\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)$$

$$\Sigma_{j,t}^{(i)} = (I - K_t H)\Sigma_{j,t-1}^{(i)}$$

3. Update the importance weight of particles.

$$w^{(i)} \sim |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}\left(z_k - h\left(\mu_{j,t-1}^{(i)}, x_t^{(i)}\right)\right)^T Q^{-1}\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)\}$$

4. Resampling.

Likelihood

- Compute log-likelihood of the joint probability.

```
def update_obs(self, zlist, idlist):
    loglike = 0
    for i in range(len(zlist)):
        p = self.update_landmark(zlist[i], idlist[i])
        loglike += np.log(p + 1e-10)
    return loglike
```

Fast-SLAM

- Steps of Fast-SLAM

1. Predict the next pose $x_t^{(i)}$ by motion model.

$$x_t^{(i)} \sim p(x_t^{(i)} | x_{t-1}^{(i)}, u_{t-1})$$

2. Update the distribution of each landmark $(\mu_{j,t}^{(i)}, \Sigma_{j,t}^{(i)})$ via measurement z_k.

$$Q = H\Sigma_{j,t-1}^{(i)}H^T + R, \qquad K_t = \Sigma_{j,t-1}^{(i)}H^T Q^{-1}$$
$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_k\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)$$
$$\Sigma_{j,t}^{(i)} = (I - K_t H)\Sigma_{j,t-1}^{(i)}$$

3. Update the importance weight of particles.

$$w^{(i)} \sim |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}\left(z_k - h\left(\mu_{j,t-1}^{(i)}, x_t^{(i)}\right)\right)^T Q^{-1}\left(z_k - h(\mu_{j,t-1}^{(i)}, x_t^{(i)})\right)\}$$

4. Resampling.

Fast SLAM

- Measure of how well the target distribution is approximated by samples drawn from the proposal.

$$N_{eff} = \frac{1}{\sum_i \left(w_t^{(i)}\right)^2}$$

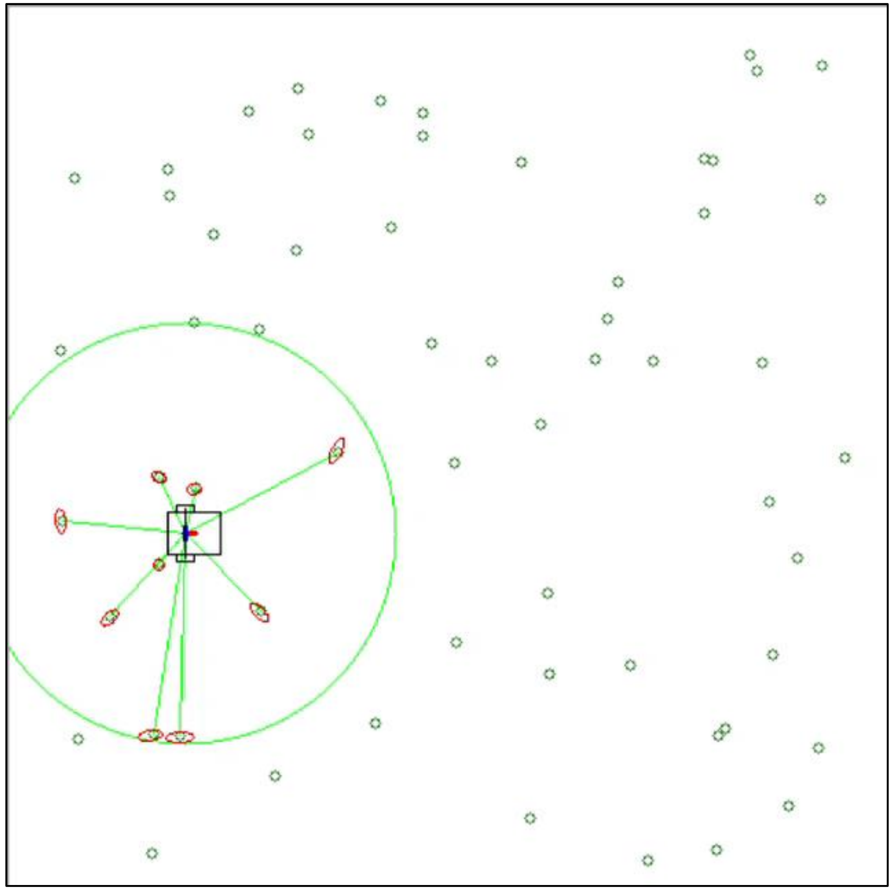
- N_{eff} denotes the inverse variance of the normalized particle weights. For equal weights, the results is the number of the particles. And the sample approximation is close to the target.

$$N_{eff}^* = \frac{1}{\sum_i \frac{1}{N^2}} = \frac{1}{N \frac{1}{N^2}} = N$$

- If N_{eff} drops below a given threshold (usually set to half of the particles), we will resample the particle.

$$N_{eff} < \frac{N}{2}$$

Demo



https://youtu.be/bLKG8aSdLRo