

# Robotic Navigation and Exploration Lab 3

Fast-SLAM

Min-Chun Hu [anitahu@cs.nthu.edu.tw](mailto:anitahu@cs.nthu.edu.tw)  
CS, NTHU

# Requirement

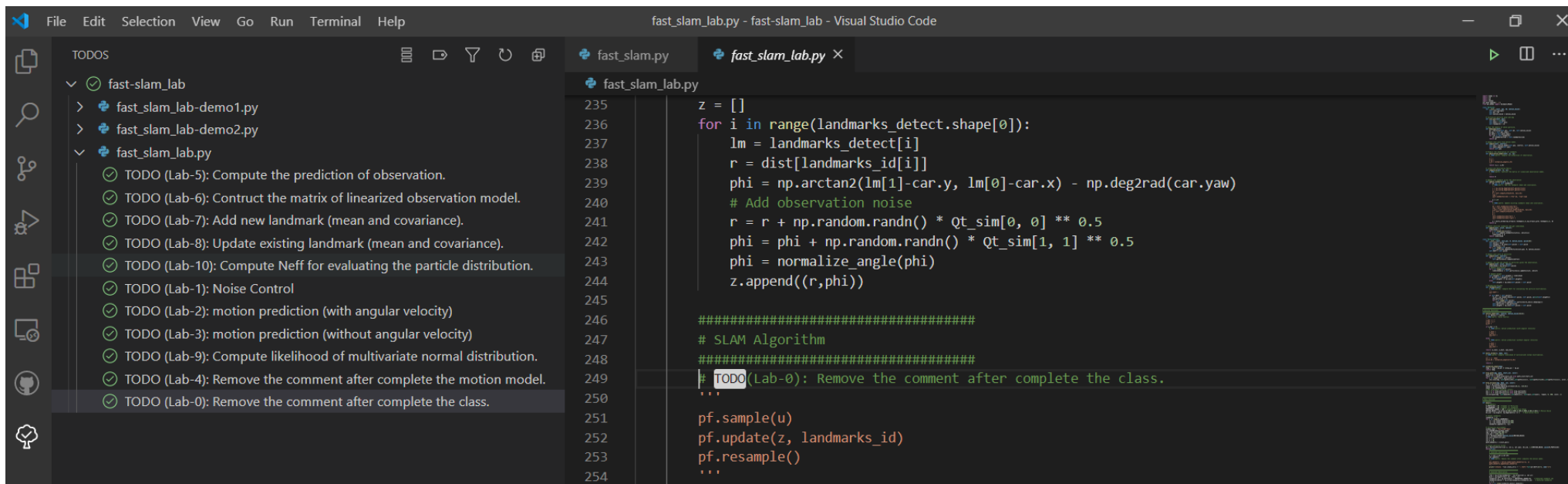
- Python 3.X
- Numpy
- Opencv-Python

# Useful NumPy Operation

- Matrix Multiply.
  - `np.matmul(A,B)` / `A.dot(B)` / `A @ B`
- Matrix Transpose.
  - `np.transpose(A)` / `A.T`
- Identity Matrix.
  - `np.eye(dimension)`
- Sample from normal distribution with mean **mu** and variance **var**.
  - `mu + np.sqrt(var)*np.random.randn()`
- Matrix Inverse.
  - `np.linalg.inv(A)`
- Matrix Determinant.
  - `np.linalg.det(A)`

# TODO List

- In this lab, you need to complete each “TODO” comment in the codes:  
`# TODO(Lab-<ID>): <Explanation>`
- The “Lab ID” implies the implementation order of the codes (except Lab-0).
- Suggest to install the “Todo Tree” extension in VSCode.



The screenshot shows the Visual Studio Code interface. On the left, the 'TODOs' sidebar lists tasks for the 'fast-slam\_lab' project, including computing observation predictions, constructing linearized observation models, adding landmarks, updating landmarks, computing Neff, noise control, motion prediction, computing likelihood, and removing comments. The main editor displays the 'fast\_slam\_lab.py' file, which contains Python code for a SLAM algorithm. The code includes a loop for processing landmarks, adding observation noise, and a TODO comment for Lab-0: 'Remove the comment after complete the class.' The code also shows particle filter operations like sampling, updating, and resampling.

```
fast_slam_lab.py
235 z = []
236 for i in range(landmarks_detect.shape[0]):
237     lm = landmarks_detect[i]
238     r = dist[landmarks_id[i]]
239     phi = np.arctan2(lm[1]-car.y, lm[0]-car.x) - np.deg2rad(car.yaw)
240     # Add observation noise
241     r = r + np.random.randn() * Qt_sim[0, 0] ** 0.5
242     phi = phi + np.random.randn() * Qt_sim[1, 1] ** 0.5
243     phi = normalize_angle(phi)
244     z.append((r,phi))
245
246 #####
247 # SLAM Algorithm
248 #####
249 # TODO(Lab-0): Remove the comment after complete the class.
250 ...
251 pf.sample(u)
252 pf.update(z, landmarks_id)
253 pf.resample()
254 ...
```

# Main Function Workflow

- Set Parameters
- Create Landmarks
- Initialize Environment
- Repeat
  - Simulate Controlling
  - Simulate Observation
  - **SLAM Algorithm**
    - `pf.sample(u)`
    - `pf.update(z, landmarks_id)`
    - `pf.resample()`
  - Render Canvas
  - Keyboard Events

$$u = (v, \omega, \Delta t)$$

$$z = [(r_{id1}, \theta_{id1}), (r_{id2}, \theta_{id2}), \dots]$$

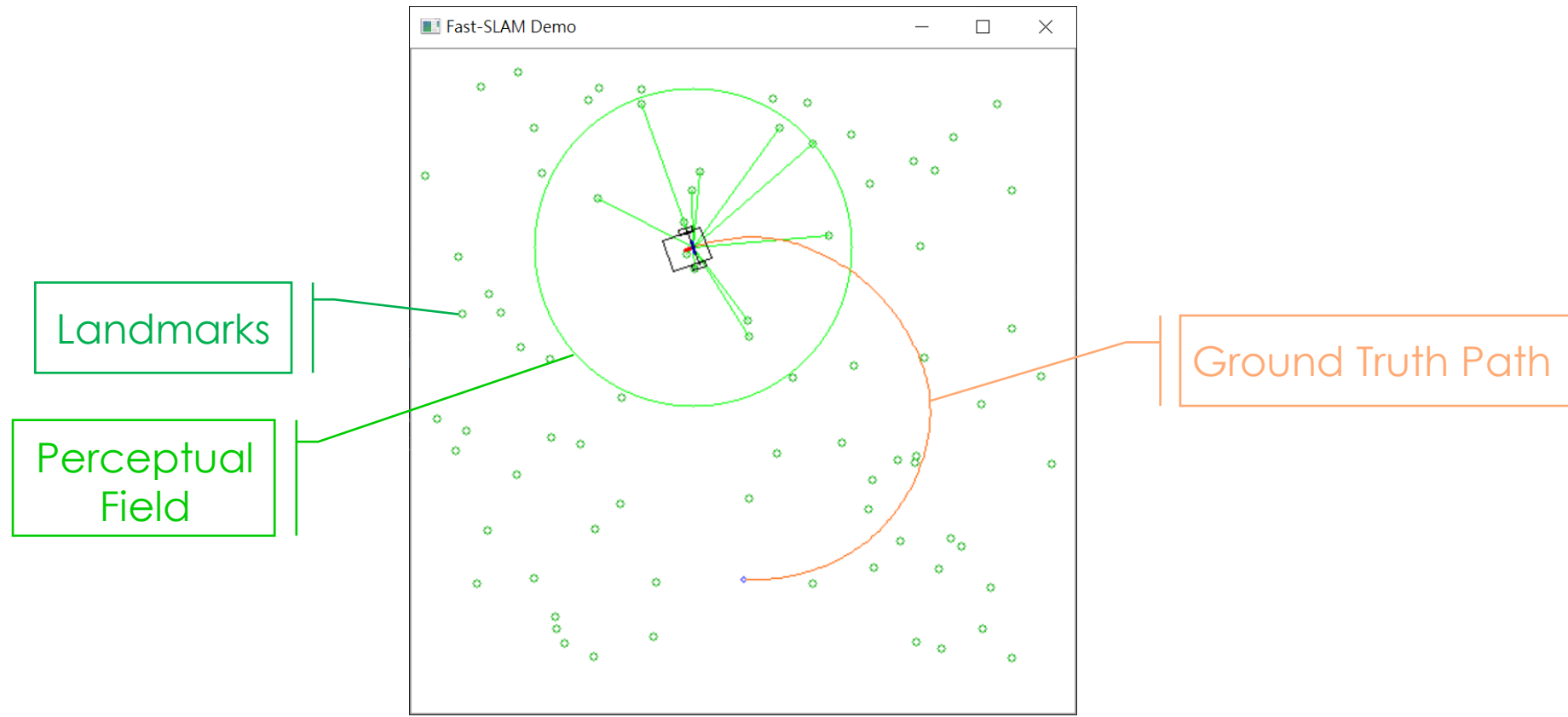
$$\text{landmarks\_id} = [id1, id2, \dots]$$

# Parameters Setting

```
# Parameters
N_PARTICLES = 40 # Number of Particles
N_LANDMARKS = 80 # Number of Landmarks
PERCEPTUAL_RANGE = 120 # Landmark Detection Range
MOTION_NOISE = np.array([1e-5, 1e-2, 1e-5, 1e-2, 1e-5, 1e-2]) # Motion Noise
Qt_sim = np.diag([4, np.deg2rad(4)]) ** 2 # Observation Noise
```

Initialize Pose: **(250,100,0)**  
Initialize Velocity: **24**  
Initialize Angular Velocity: **10**

# Run the Code



# 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  $z_t$ .

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

$$\mu_{j,t}^{(i)} = \mu_{j,t-1}^{(i)} + K_t (z_t - h(\mu_{j,t-1}^{(i)}, x_t^{(i)}))$$

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

3. Update the importance weight of particles.

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

4. Resampling.



# Class Design

- Particle
  - `init_pos`
  - `deepcopy`: Copy the memory of whole particle.
  - `sample (TODO)`: Sample next pose from motion model.
  - `observation_model (TODO)`: Predict the observation of landmark.
  - `compute_H (TODO)`: Construct the matrix of linearized observation model.
  - `update_landmark (TODO)`: Update one landmark given the observation.
  - `update`: Update observed landmarks and get likelihood.
- Particle Filter
  - `sample`: Sample next pose of particles.
  - `update`: Update the map and weight of particles given the observation.
  - `resample (TODO)`: Compute Neff for evaluating the particle distribution.

# Utility Functions

- `multi_normal()` **(TODO)**
  - Compute the probability of multivariate normal distribution.
- `normalize_angle()`
  - Normalize the angle to the range  $-\pi \sim \pi$ .

# 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  $z_k$ .

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

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

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

3. Update the importance weight of particles.

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

4. Resampling.

# Sample from Velocity Motion Model

With Angular Velocity:

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

Without Angular Velocity:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \hat{v} \cos(\theta) * \Delta t \\ \hat{v} \sin(\theta) * \Delta t \\ \hat{\gamma}\Delta t \end{bmatrix}$$

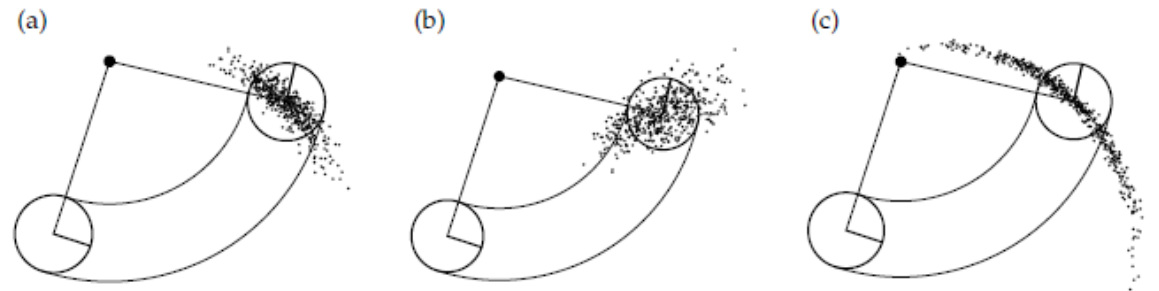
$$\begin{aligned} \hat{v} &\sim \mathcal{N}(v, \sigma_v^2) \\ \hat{\omega} &\sim \mathcal{N}(\omega, \sigma_\omega^2) \\ \hat{\gamma} &\sim \mathcal{N}(\gamma, \sigma_\gamma^2) \end{aligned}$$

$$\begin{aligned} \sigma_v^2 &= \alpha_1 v^2 + \alpha_2 \omega^2 \\ \sigma_\omega^2 &= \alpha_3 v^2 + \alpha_4 \omega^2 \\ \sigma_\gamma^2 &= \alpha_5 v^2 + \alpha_6 \omega^2 \end{aligned}$$

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

```

2:    $\hat{v} = \overset{\mu}{v} + \text{sample}(\overset{\sigma^2}{\alpha_1 v^2 + \alpha_2 \omega^2})$ 
3:    $\hat{\omega} = \omega + \text{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$ 
4:    $\hat{\gamma} = \text{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$ 
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  $x_t = (x', y', \theta')^T$ 
    
```



# Sample from Velocity Motion Model

- Lab-1~Lab-3: Complete the velocity motion model.

```
def motion_model(pos, control, motion_noise=[0]*6):
    x, y, yaw = pos
    v, w, delta_t = control

    # TODO(Lab-1): Noise Control
    ...

    v_hat = v +
    w_hat = w +
    g_hat =
    ...

    if w_hat != 0:
        # TODO(Lab-2): motion prediction (with angular velocity)
        ...

        x_next = x +
        y_next = y +
        yaw_next = yaw +
        ...

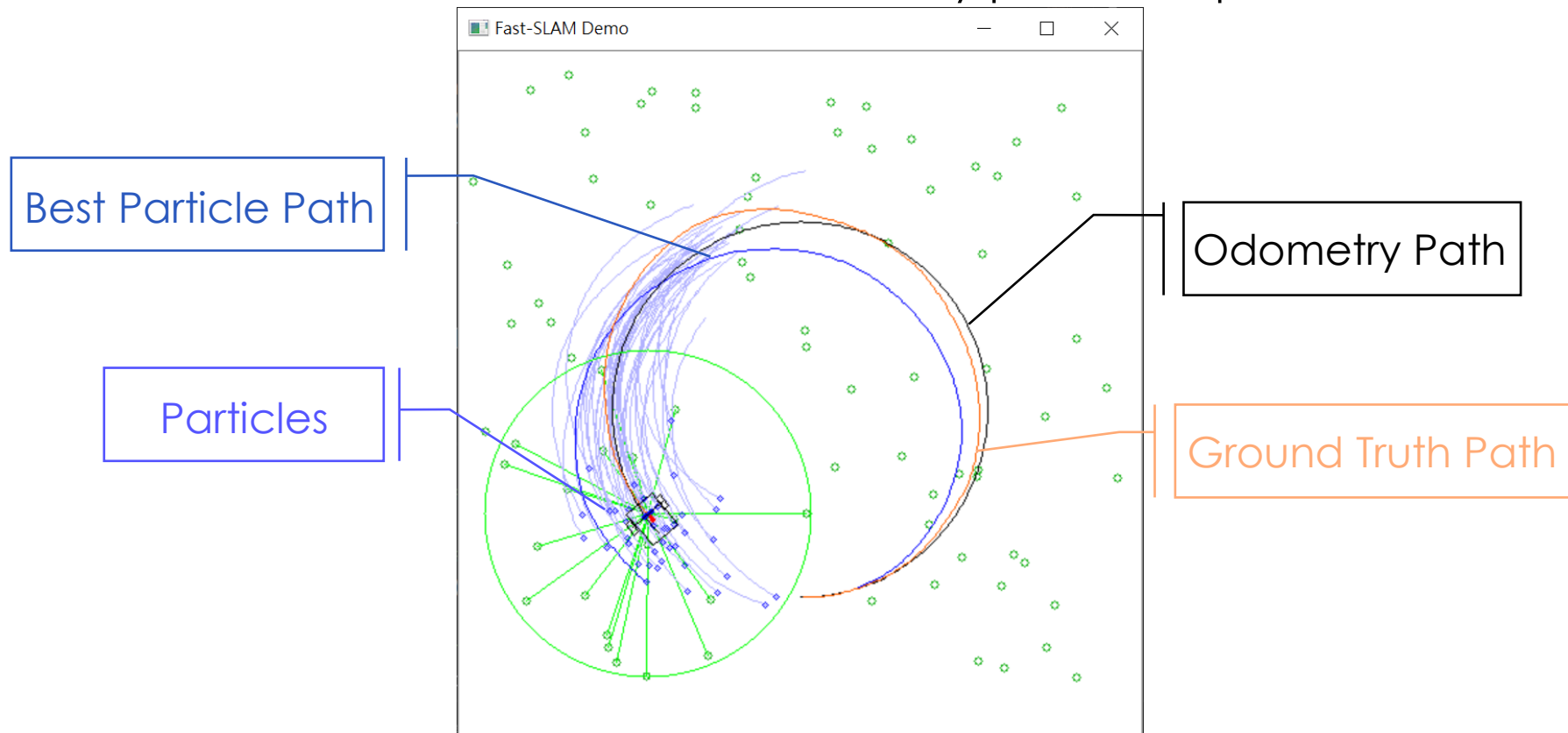
    else:
        # TODO(Lab-3): motion prediction (without angular velocity)
        ...

        x_next = x +
        y_next = y +
        yaw_next = yaw +
        ...

    return [x_next, y_next, yaw_next]
```

# Run the code

- Lab-0: Remove the comment of `“pf.sample(u)”`.
- Lab-4: Remove the comment of odometry path computation.



# 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  $z_k$ .

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

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

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

3. Update the importance weight of particles.

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

4. Resampling.

# Observation Model

- Given observation model

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

- Linearized the observation model :

$$\begin{aligned} \text{➤ } H^i &= \frac{\partial z_i}{\partial (x, y, \theta, m_{i,x}, m_{i,y})} = \begin{bmatrix} \frac{\partial \sqrt{q}}{\partial x} & \frac{\partial \sqrt{q}}{\partial y} & \dots \\ \frac{\partial \text{atan2}(\delta_x, \delta_y)}{\partial x} & \frac{\partial \text{atan2}(\delta_x, \delta_y)}{\partial y} & \dots \end{bmatrix} \\ &= \frac{1}{q} \begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \boxed{\sqrt{q}\delta_x} & \boxed{\sqrt{q}\delta_y} \\ \delta_y & -\delta_x & -q & \boxed{-\delta_y} & \boxed{\delta_x} \end{bmatrix} \\ &\quad \begin{matrix} \partial/\partial x & \partial/\partial y & \partial/\partial \theta & \partial/\partial r & \partial/\partial \phi \end{matrix} \end{aligned}$$

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

$$\begin{aligned} \frac{\partial}{\partial x} \text{atan2}(y, x) &= \frac{\partial}{\partial x} \arctan\left(\frac{y}{x}\right) = -\frac{y}{x^2 + y^2}, \\ \frac{\partial}{\partial y} \text{atan2}(y, x) &= \frac{\partial}{\partial y} \arctan\left(\frac{y}{x}\right) = \frac{x}{x^2 + y^2}. \end{aligned}$$

- Only Consider the Landmarks

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



# Observation Model

- Lab-5: Compute the prediction of observation.
- Lab-6: Construct the matrix of linearized observation model.

```
# Predict the observation of landmark.
def observation_model(self, lx, ly):
    x, y, yaw = self.pos
    # TODO(Lab-5): Compute the prediction of observation.
    # [Hint 1] The parameter "lx,ly" is the location of landmark.
    ...

    z_r =
    z_th =
    z_th = normalize_angle(z_th)
    ...

    return (z_r, z_th)

# Linearized Observation Matrix
def compute_H(self, lx, ly):
    x, y, yaw = self.pos
    # TODO(Lab-6): Construct the matrix of linearized observation model.
    # [Hint 1] The parameter "lx,ly" is the location of landmark.
    ...
    ...

    return H
```

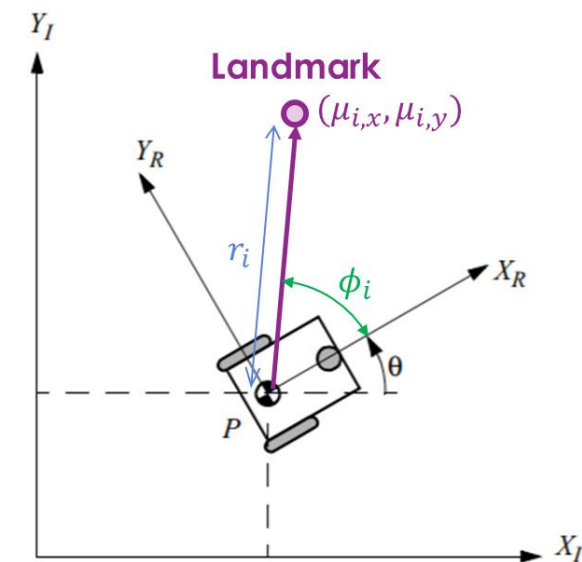
# Add New Landmark

- Obtain the relative measurement of landmarks:  $z_i = (r_i, \phi_i)^T$ 
  - $\triangleright h^{-1}(z, x) = \begin{bmatrix} \mu_{i,x} \\ \mu_{i,y} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} r_i \cos(\phi_i + \theta) \\ r_i \sin(\phi_i + \theta) \end{bmatrix}$
- Given observation noise  $Q_t$ , the covariance of the observation will be scaled by the inverse of linearized observation matrix  $H^{-1} Q_t (H^{-1})^T$ .

$$\triangleright H^{-1} = \frac{\partial h^{-1}(z, x)}{\partial x} = \begin{bmatrix} \cos(\phi_i + \theta) & -r_i \sin(\phi_i + \theta) \\ \sin(\phi_i + \theta) & r_i \cos(\phi_i + \theta) \end{bmatrix}$$

$$\begin{aligned} \mu_{j,t}^{[k]} &= h^{-1}(z_t, x_t^{[k]}) \\ H &= h'(x_t^{[k]}, \mu_{j,t}^{[k]}) \\ \Sigma_{j,t}^{[k]} &= H^{-1} Q_t (H^{-1})^T \\ w^{[k]} &= p_0 \end{aligned}$$

Initialized probability of new landmark, set to 1



# Add New Landmark

- Lab-7: Add new landmark, compute mean and covariance.

```
if lid not in self.landmarks:
    # TODO(Lab-7): Add new landmark (mean and covariance).
    # [Hint 1] The parameter "z" is a list of one landmark [r, phi].
    # [Hint 2] The observation noise is "self.Qt (numpy array)".
    # [Hint 3] The mean of landmark "mu" is a numpy array with shape (2,1).
    ...

    c = np.cos(yaw+z[1])
    s = np.sin(yaw+z[1])
    mu =
    H = self.compute_H(mu[0,0], mu[1,0])
    Hinv = np.linalg.inv(H)
    sig =
    self.landmarks[lid] = {"mu":mu, "sig":sig}
    ...

    p = 1.0
```

# Update Existing Landmark

- Extended Kalman Filter

$$x_t^{pre} = f(x_{t-1}^{est}, u_t) \rightarrow \mu_{t-1} \text{ (Landmarks do not move)}$$

$$P_t^{pre} = F_t P_{t-1}^{pre} F_t^T + Q \rightarrow \Sigma_{t-1} \text{ (Landmarks do not move)}$$

$$K_t = P_t^{pre} H^T (H P_t^{pre} H^T + \overbrace{R}^Q)^{-1}$$

$Q_t$  (Observation Noise)

$$x_t^{est} = x_t^{pre} + K_t (z_t - \underbrace{H x_t^{pre}}_{\hat{z}})$$

$\hat{z}$  (Predict Observation)

$$P_t^{est} = (I - K_t H) P_t^{pre}$$

$$\begin{aligned} \hat{z} &= h(\mu_{j,t-1}^{[k]}, x_t^{[k]}) \\ H &= h'(x_t^{[k]}, \mu_{j,t-1}^{[k]}) \\ Q &= H \Sigma_{j,t-1}^{[k]} H^T + Q_t \\ K &= \Sigma_{j,t-1}^{[k]} H^T Q^{-1} \\ \mu_{j,t}^{[k]} &= \mu_{j,t-1}^{[k]} + K(z_t - \hat{z}) \\ \Sigma_{j,t}^{[k]} &= (I - K H) \Sigma_{j,t-1}^{[k]} \\ w^{[k]} &= |2\pi Q|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - \hat{z}_n)^T Q^{-1} (z_t - \hat{z}_n) \right\} \end{aligned}$$

# Update Existing Landmark

- Lab-8: Update existing landmark, compute mean and covariance.

```
else:
    # TODO(Lab-8): Update existing landmark (mean and covariance).
    ...

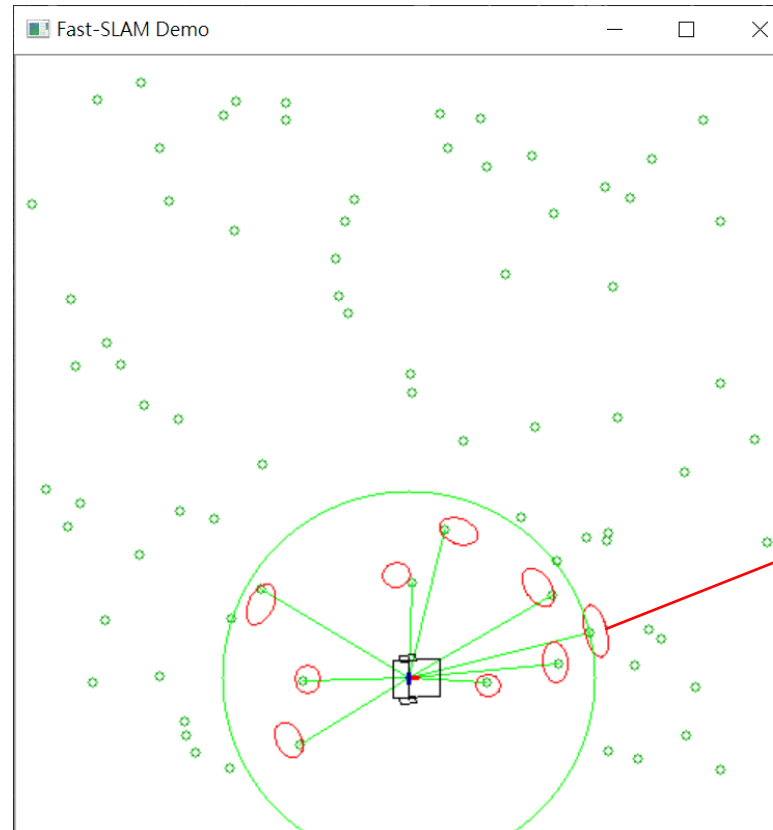
    mu = self.landmarks[lid]["mu"]
    sig = self.landmarks[lid]["sig"]
    z_hat = self.observation_model(mu[0,0], mu[1,0])
    H = self.compute_H(mu[0,0], mu[1,0])
    Q =
    K =

    e = np.array(z) - np.array(z_hat)
    e[1] = normalize_angle(e[1])
    self.landmarks[lid]["mu"] =
    self.landmarks[lid]["sig"] =
    ...

    p = multi_normal(np.array(z).reshape(2,1), np.array(z_hat).reshape(2,1), Q)
```

# Run the Code

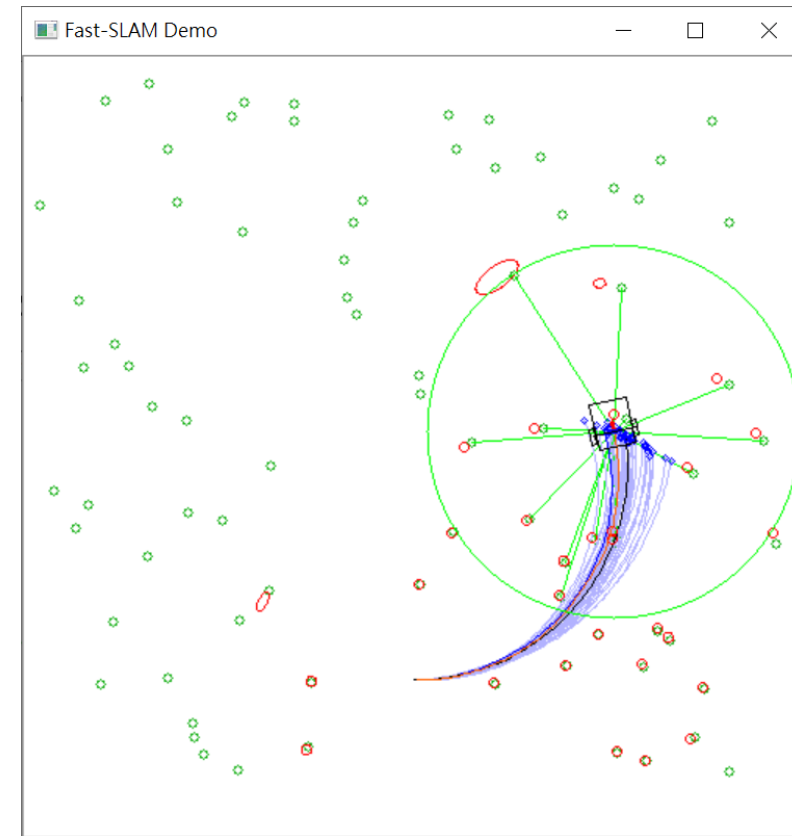
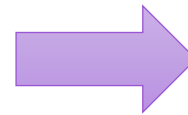
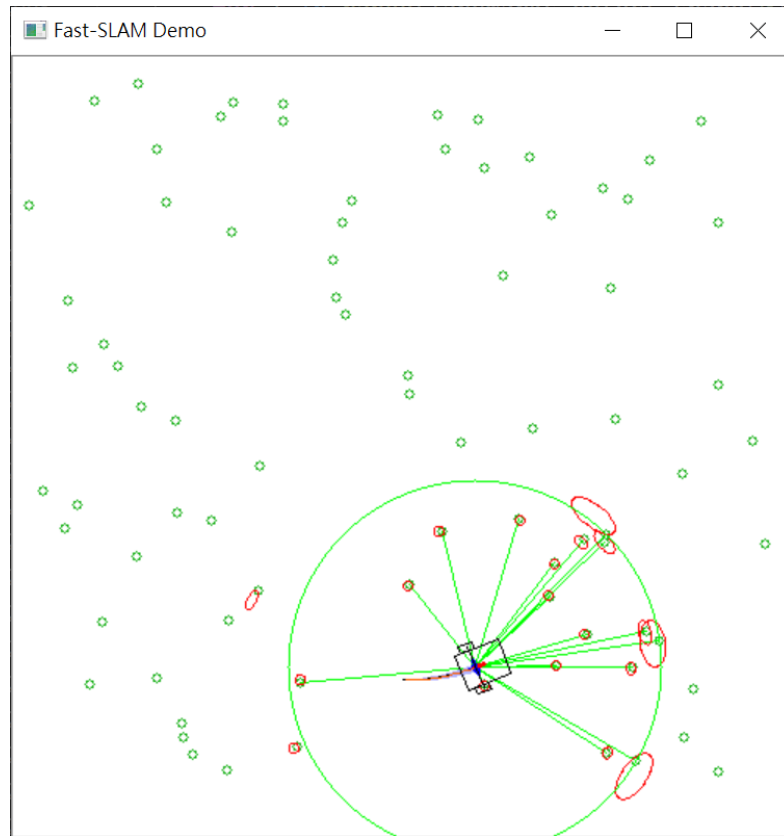
- Lab-0: Remove the comment of `pf.update(z, landmarks_id)`.
- Set `cv2.waitKey(1)` to `cv2.waitKey(0)` to observe the distribution of new landmarks.



Landmark of best particle

# Run the Code

- Set back to “`cv2.waitKey(1)`” to observe the updating of landmarks.



# 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  $z_k$ .

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

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

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

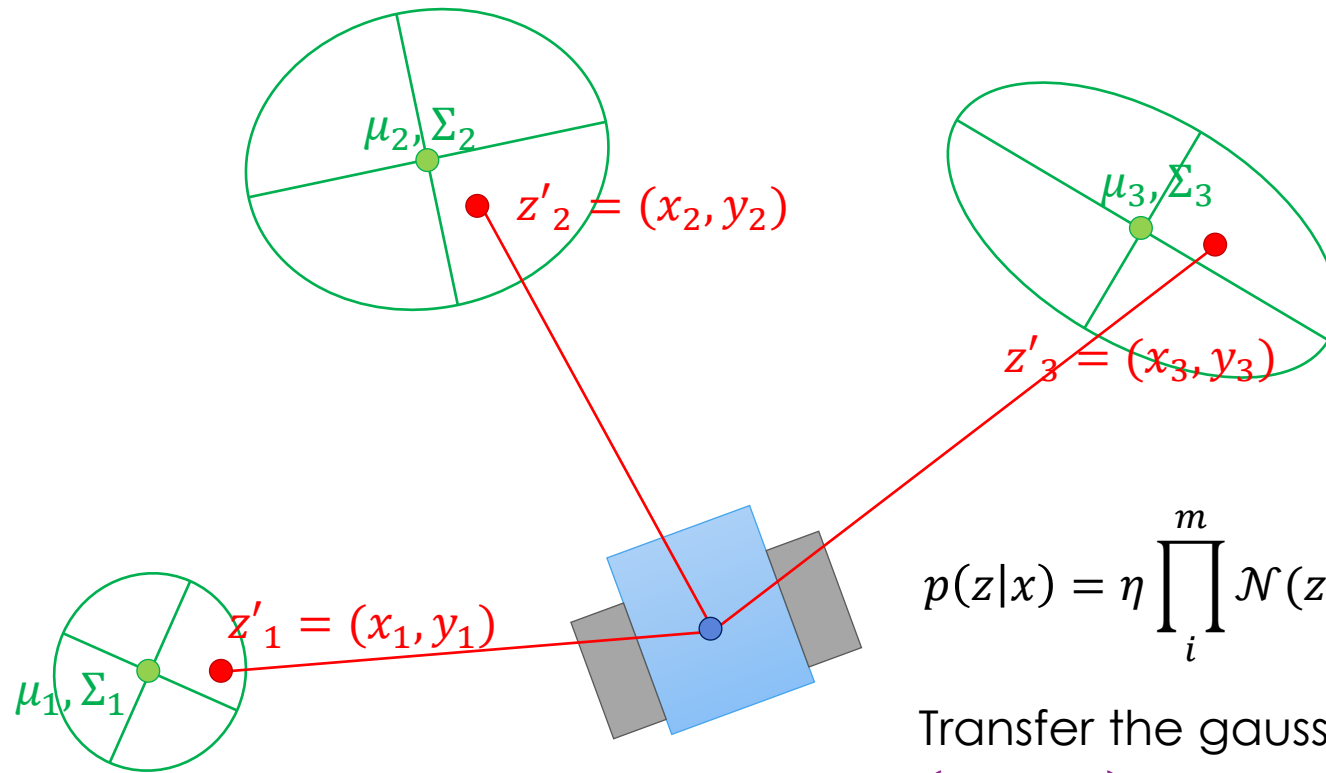
3. Update the importance weight of particles.

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

4. Resampling.



# Likelihood of Measurement



$$p(z|x) = \eta \prod_i^m \mathcal{N}(z'_i; \mu_i, \Sigma_i)$$

Transfer the gaussian of  $xy$  to  $r\theta$ :  
 $(\mu_{xy}, \Sigma_{xy}) \rightarrow (\mu_{r\theta}, \Sigma_{r\theta}) = (\hat{z}, Q)$

$$Q = H\Sigma_{xy}H^T + Q_t$$

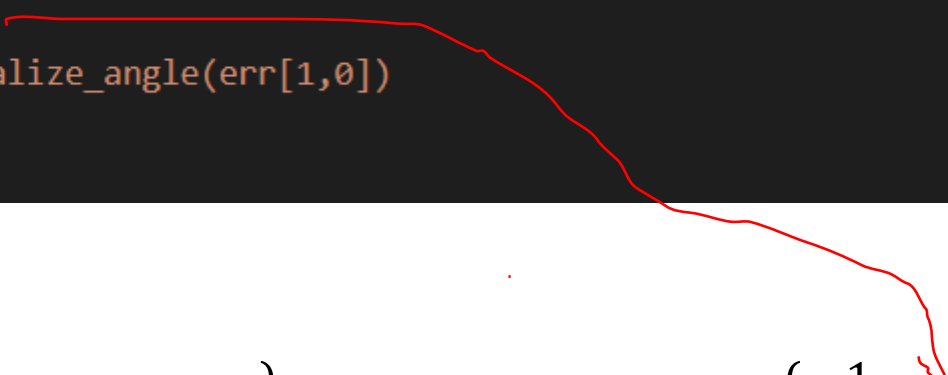
# Likelihood

- Lab-9: Compute likelihood of multivariate normal distribution.

```
def multi_normal(x, mean, cov):  
    # TODO(Lab-9): Compute likelihood of multivariate normal distribution.  
    ...  
    err = x - mean  
    err[1,0] = normalize_angle(err[1,0])  
    ...  
    return 1
```

$$\mathcal{N}_k(\mu, \Sigma) = \frac{\exp\left\{-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right\}}{\sqrt{(2\pi)^k |\Sigma|}}$$

Determinant  $\rightarrow$  行列式


$$w^{(i)} \sim \frac{\exp\left\{-\frac{1}{2}(\underline{x} - \mu)^T Q^{-1}(x - \mu)\right\}}{2\pi\sqrt{|Q|}}$$

# 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  $z_k$ .

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

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

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

3. Update the importance weight of particles.

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

4. Resampling.

# Evaluating the Particle Weights

- 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.

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

# Evaluating the Particle Weights

- Lab-10: Compute Neff

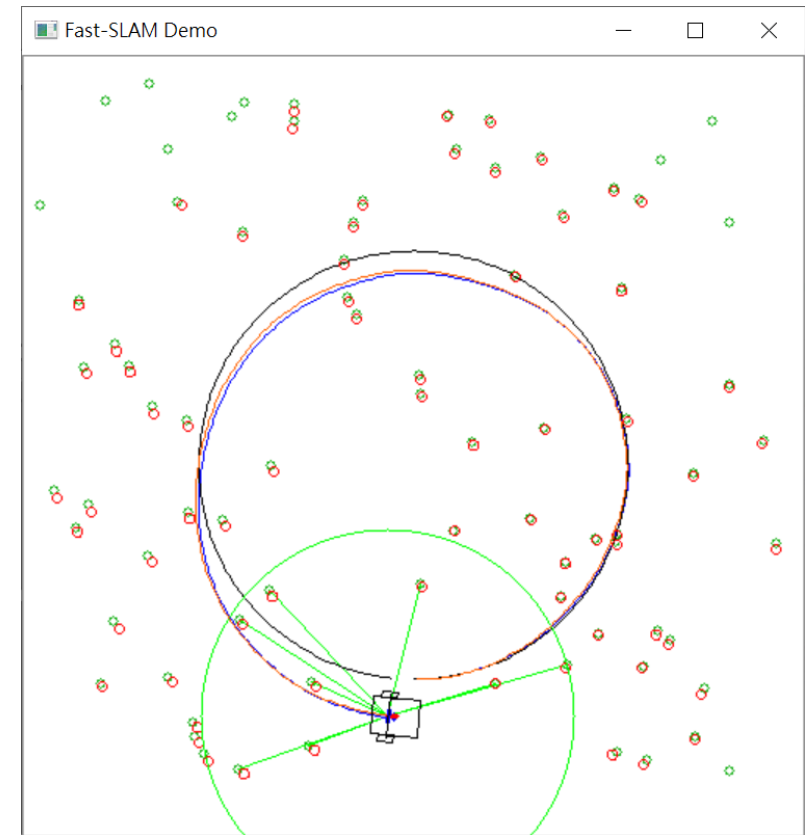
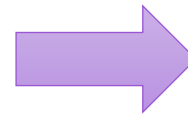
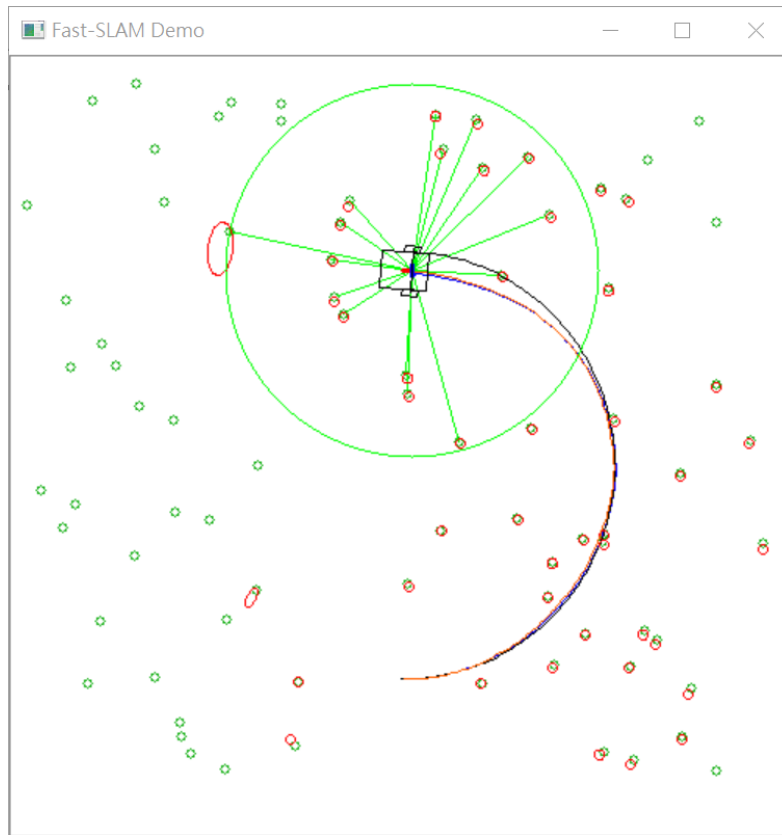
```
# Resampling Process
def resample(self):
    # TODO(Lab-10): Compute Neff for evaluating the particle distribution.
    ...

    self.Neff =
    ...

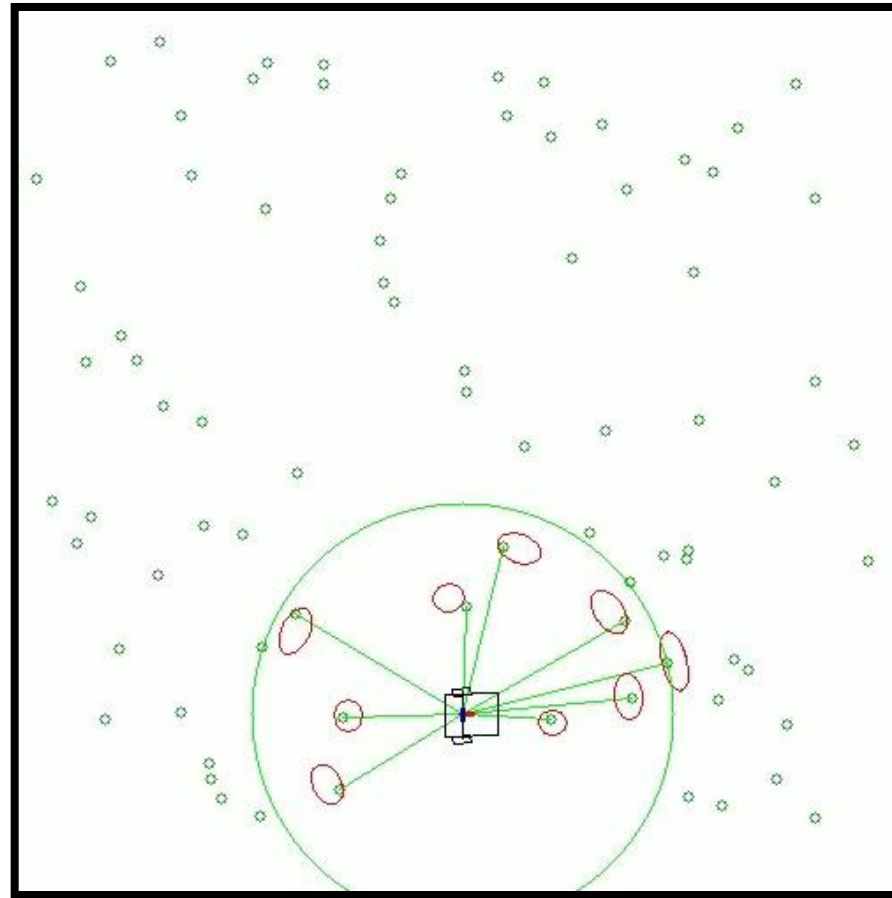
    if self.Neff < self.psize/2:
        re_id = np.random.choice(self.psize, self.psize, p=list(self.weights))
        new_particles = []
        for i in range(self.psize):
            new_particles.append(self.particles[re_id[i]].deepcopy())
        self.particles = new_particles
        self.weights = np.ones(self.psize) / self.psize
```

# Run the Code

- Lab-0: Remove the comment of `“pf.resample()”`.



# Demo



**Hint:**

You can press "R" to reset the environment and control the car by keyboard (WSAD).

<https://youtu.be/eFjTG5mVpJI>