zs2489 Zhou Shen
yx2725 YaoYao Xu

CAofR HW4

COMS 4733
November 24, 2021

Problem 1: Bayes Filter

(a) The pdf of the motion model is:

$$f = \frac{1}{\sqrt{2\pi}} e^{-0.5*(x_1 - x_0 - 0.5)^2} \tag{1}$$

The new belief distribution is:

$$\begin{aligned} B^{'}(x_1) &= \int_{-1}^{1} p(x_1|x_0, u_1) B(x_0) dx_0 \\ &= \int_{-1}^{1} \frac{1}{\sqrt{2\pi}} e^{-0.5*(x_1 - x_0 - 0.5)^2} * \frac{1}{2} dx_0 \\ &= \frac{(1.25331 erf(1.06066 - 0.707107 x_1) + 1.25331 erf(0.707107 x_1 + 0.353553))}{(2\sqrt{2\pi})} \end{aligned} \tag{2}$$

Then, the plot of the pdf is:

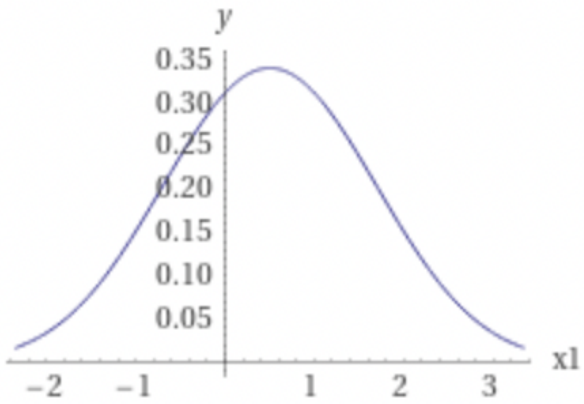

Figure 1: pdf of belief distribution

However, this distribution is over the domain $[0, 1]$, and we know that the entire infinite hallway initially has a uniform distribution. Thus, there is likelihood that the robot can be in the other location instead of $[-1, 1]$.

(b) We know that $p(z_1|x_1)$ is a uniform distribution over $[-0.5, 1.5]$, so we can make other

locations zero. First, we compute the normalization factor:

$$\begin{aligned}\eta &= \int_{-0.5}^{1.5} B^{'}(x_1)dx_1 \\ &= \int_{-0.5}^{1.5} \frac{1.25331erf(1.06066 - 0.707107x_1) + 1.25331erf(0.707107x_1 + 0.353553)}{2\sqrt{2\pi}}dx_1 \\ &= 0.624653\end{aligned} \tag{3}$$

Then, compute the posterior belief distribution:

$$\begin{aligned}B(x_1) &= \eta^{-1}p(z_1|x_1)B^{'}(x_1) \\ &= \frac{(1.25331erf(1.06066 - 0.707107x_1) + 1.25331erf(0.707107x_1 + 0.353553))}{(4\sqrt{2\pi}) * 0.624653} \\ &= 0.159666(1.25331erf(1.06066 - 0.707107x_1) + 1.25331erf(0.707107x_1 + 0.353553))\end{aligned} \tag{4}$$

The plot of the pdf is (Here, we only consider the range $[-0.5, 1.5]$, because in other locations, the probability is 0):

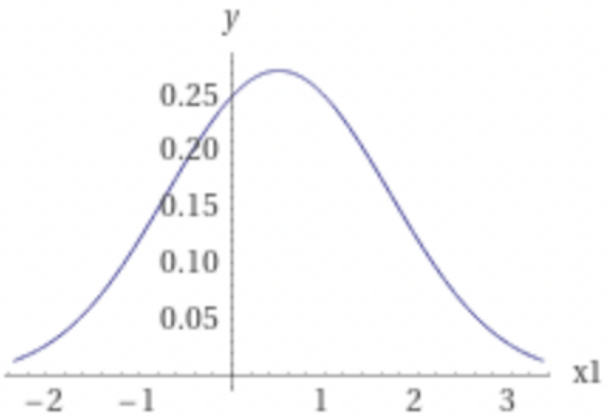

Figure 2: pdf of posterior belief distribution

(c) Because $B(x_0)$ are reweighted in an uniform distribution over these four state values, so $B(x_0) = (0.25, 0.25, 0.25, 0.25)$. Thus, we need to compute following equation:

$$B^{'}(x_1) = \sum p(x_1|x_0, u_1)B(x_0) \tag{5}$$

When $x_1 = -0.75$, we get the equation:

$$\begin{aligned} B^{'}(x_1 = -0.75) &= \sum_{x_0} \frac{1}{\sqrt{2\pi}} e^{-0.5*(-0.75-x_0-0.5)^2} * 0.25 \\ &\text{for each } x_0 \text{ location(-0.75,-0.25,0.25,0.75)} \\ &\approx 0.194386 \end{aligned} \tag{6}$$

Using the same equation(6), we can get discrete belief distribution:

$$\begin{aligned} B^{'}(x_1 = -0.25) &\approx 0.280624 \\ B^{'}(x_1 = 0.25) &\approx 0.336261 \\ B^{'}(x_1 = 0.75) &\approx 0.336261 \end{aligned} \tag{7}$$

(d) The observation model $p(z_1|x1)$ is an uniform distribution over the domain $[-0.5, 1.5]$. Thus, we get:

$$\begin{aligned} p(z_1 = -0.75|x_1) &= 0 \\ &\text{.-0.75 is out of sensor range} \\ p(z_1 = -0.25|x_1) &= 0.25 \\ p(z_1 = 0.25|x_1) &= 0.25 \\ p(z_1 = 0.75|x_1) &= 0.25 \\ p(z_1 = 1.25|x_1) &= 0.25 \end{aligned} \tag{8}$$

Thus, the discrete posterior belief distribution equation is:

$$\begin{aligned} B(x_1 = -0.75) &= \eta^{-1} p(z_1 = -0.75|x_1) B^{'}(x_1 = -0.75) \\ &= \eta^{-1} * 0 * 0.194386 \\ &= 0 \end{aligned} \tag{9}$$

Thus, we can get the others:

$$\begin{aligned} B(x_1 = -0.25) &\approx \eta^{-1}(0.070156) \\ B(x_1 = 0.25) &\approx \eta^{-1}(0.08406525) \\ B(x_1 = -0.25) &\approx \eta^{-1}(0.08406525) \end{aligned} \tag{10}$$

Then, the $\eta = 0.2382865$. Finally, we get $B(x_1) = (0, 0.29442, 0.35279, 0.35279)$

Problem 2: Homogeneous Transformations

(a)

$$A_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2^1 = \begin{bmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From Frame 2 to Frame, first rotate it along Z axis by 90 degrees, then rotate it along X axis by 180 degrees.

$$A_3^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^0 = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} 0 & 1 & 0 & -0.5 \\ 1 & 0 & 0 & 1.5 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b)

$$A_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From Frame 1 to Frame 2, first rotate along Z axis by 90 degrees and then do the transition.

$$A_2^1 = \begin{bmatrix} 0 & -1 & 0 & -0.5 \\ 1 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From Frame 2 to Frame 3, first rotate along X axis by 180 degrees and then do the transition.

$$A_3^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0.5 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(c)

$$A_3^0 = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} 0 & 1 & 0 & -0.5 \\ 1 & 0 & 0 & 1.5 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We get the same answer no matter we use the transformation from part(a) or part(b). For part(a), we first translate the base frame in X axis by -0.5, in Y axis by 1.5, and in Z axis by 1. Then it rotates along Z axis by 90 degrees and translates in Z axis by 2.

For part(b), we first translate the base frame in Y axis by 0.5, in Z axis by 1. Then rotate it along Z axis by 90 degrees and translate it along X axis by -0.5, in Y axis by 0.5. After this, we rotate it along X axis by 180 degrees, and translate it in Z axis by 2.

Problem 3: Forward kinematics

(a) The DH Table is shown in figure below.

DH	α_i	a_i	d_i	θ_i
1	$\frac{\pi}{2}$	0	0	θ_1
2	$-\frac{\pi}{2}$	0	d_2	0
3	0	a_3	0	θ_3

Figure 3: DH table

From Frame 0 to Frame 1, first rotate about Z axis by θ_1, then rotate it about X axis by 90 degrees. From Frame 1 to Frame 2, first rotate it about X axis by -90 degrees, the translate it along Z axis by d_2. From Frame 2 to Frame 3, first rotate about Z axis by θ_3, the translate it along X axis by a_3.

(b)

$$A_1^0 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3 s_3 \\ s_3 & c_3 & 0 & a_3 c_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(c)

$$T_3^0 = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} c_1c_3 - s_1s_3 & -c_1s_3 - c_3s_1 & -s_1 & d_2s_1 + a_3c_1c_3 - a_3c_3s_1 \\ c_1s_3 + c_3s_1 & c_1c_3 - s_1s_3 & c_1 & a_3c_1c_3 - c_1d2 + a_3c_3s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

End effector position:

$$X_e = d_2s_1 + a_3c_1c_3 - a_3c_3s_1$$

$$Y_e = a_3c_1c_3 - c_1d2 + a_3c_3s_1$$

End effector orientation:

$$\phi = \theta_1 + \theta_3$$

Problem 4: Differential-Drive Car Localization

4.2 Discussion

4.2.1

Figure 4: EKF Path 1

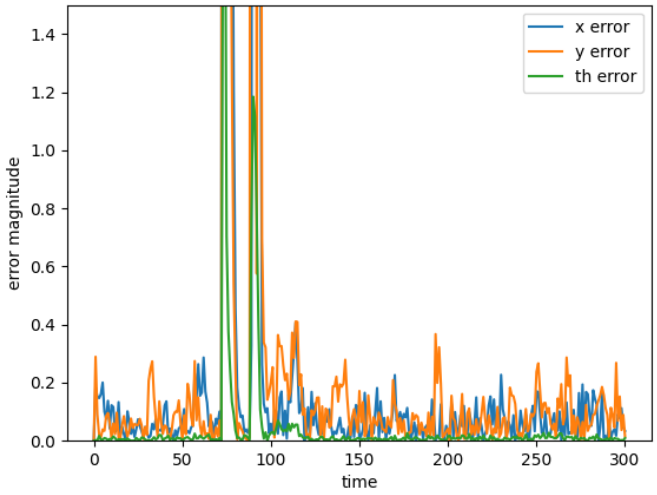

Figure 5: EKF error plot 1

The error is caused by the computation of the innovation. When the Bearing Sensor observes

two different reading 90 degree and -270 degree, the two degree represent a same observations; however, when calculating the innovation, the two degrees represent different observations in term of rad.In this case, we need to handle wraparound cases (-270° is equivalent to 90°), and the innovation should never exceed pi in magnitude.

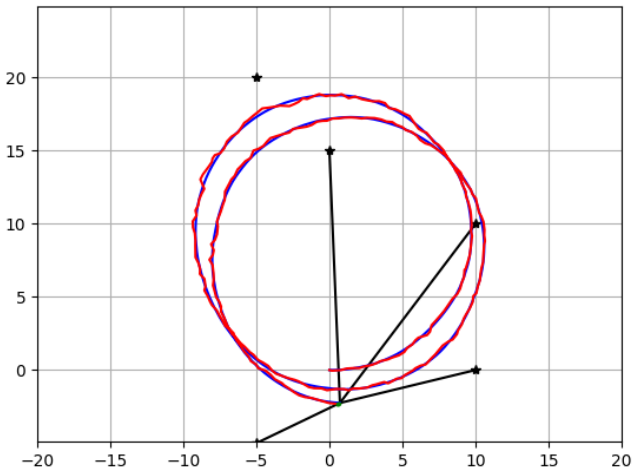

Figure 6: EKF Path 2

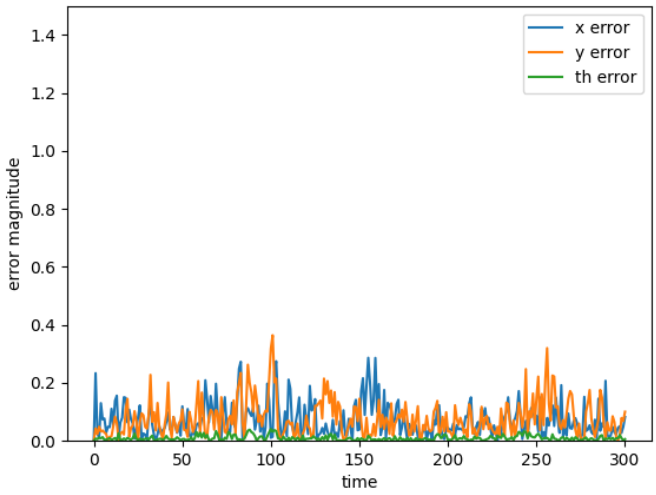

Figure 7: EKF error plot 2

The figure above shows the results after we normalize the innovation in the range of pi. We

observe that the error diminishes and the deviation between true data and predict data is close to zero.

4.2.2

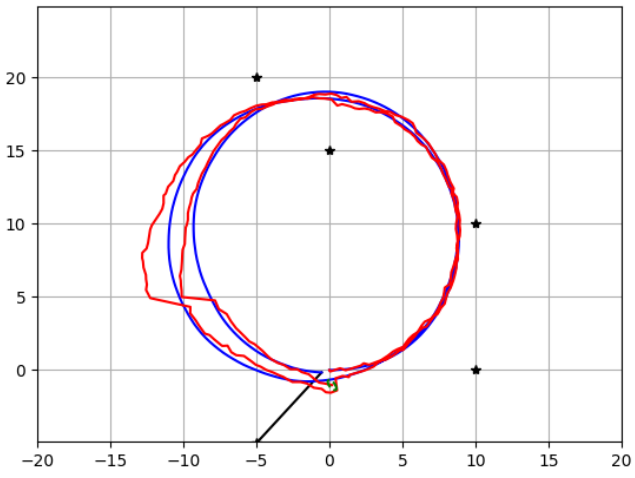

Figure 8: EKF with small sensor range path

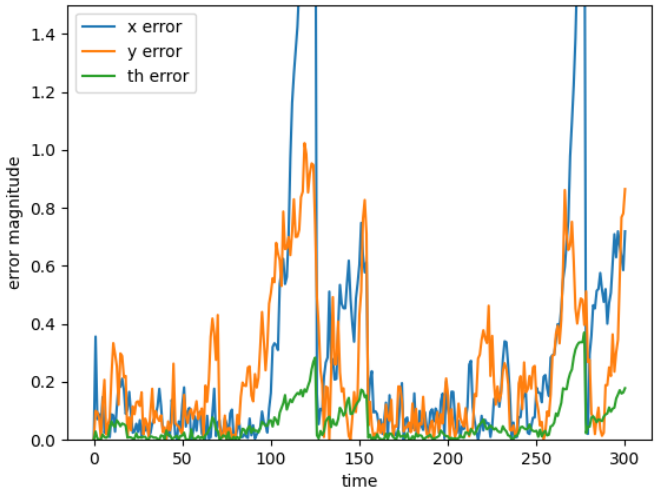

Figure 9: EKF with small sensor range error plot

When the sensor range decreases to 10, the visible observations decreases. From the plot above, we observe that the predicted value deviate from the true value when no landmarks observed. And the uncertainty(green ellipse) increased as we decrease the sensor range. The deviation is caused by the update step. Since the sensor does not observe any landmark, the

EKF only execute prediction step, and the noise accumulates in the prediction step causing the prediction deviate from true value.

4.2.3

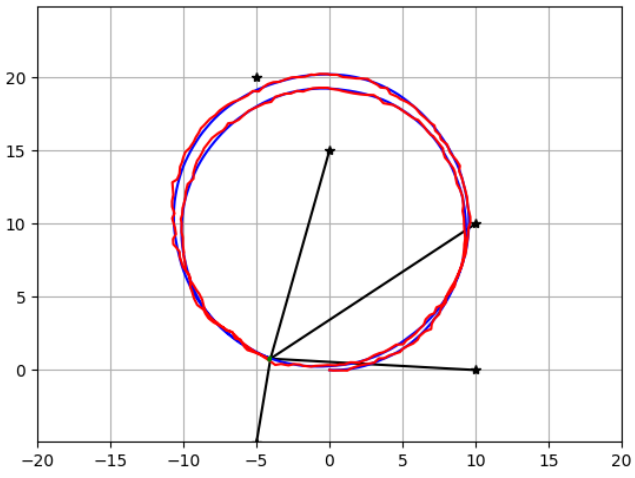

Figure 10: EKF with P to be 1000

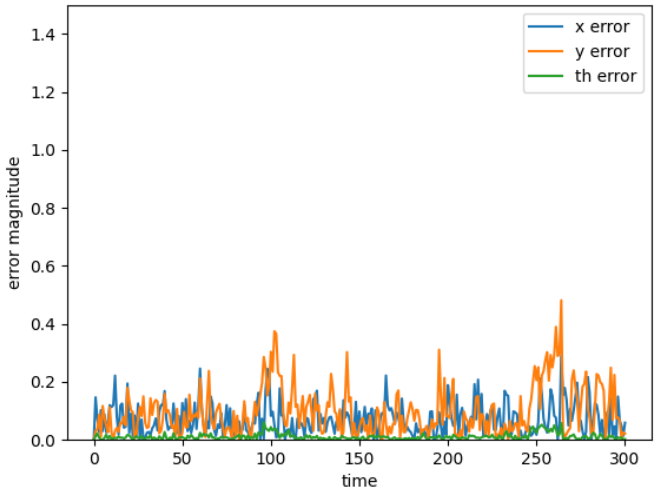

Figure 11: EKF with P to be 1000 error plot

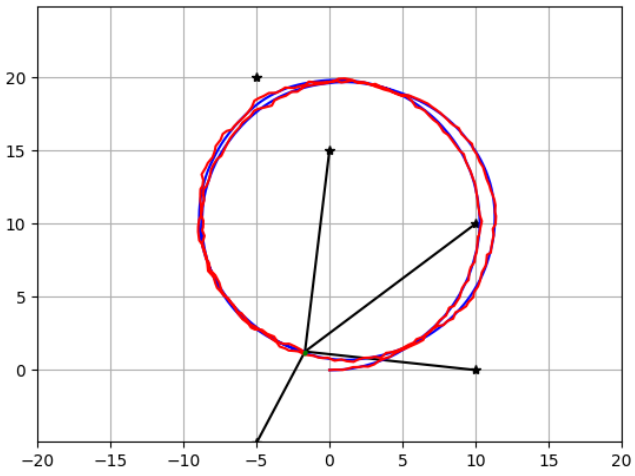

Figure 12: EKF with P to be 0.001

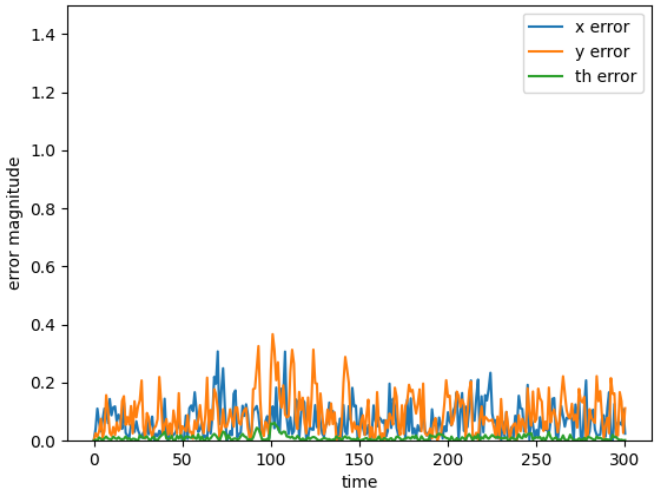

Figure 13: EKF with P to be 0.001 error plot

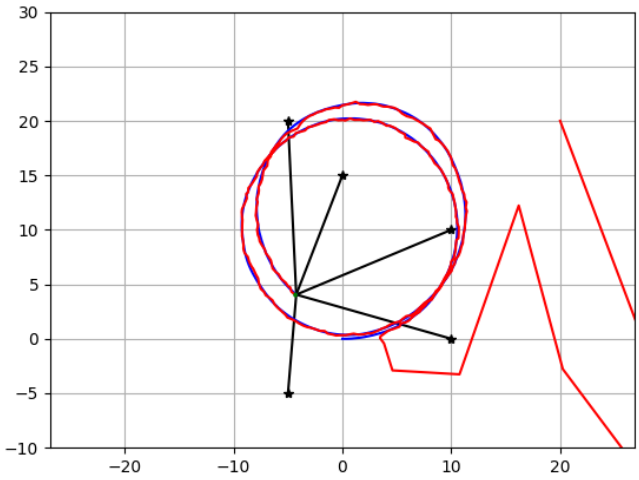

Figure 14: EKF with new initial x

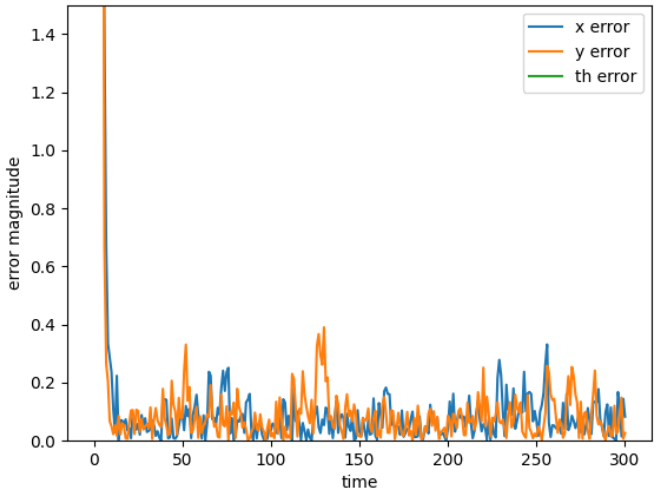

Figure 15: EKF with new initial x error plot

From Figure 10(using P as 1000) and 11(using P as 0.001), change P does not change the accuracy of EKF. Setting the initial estimated position to be 20,20,20 causes the initial estimation deviate from the true estimation; however, the prediction converges to the actual in a few steps. Since the predicted Gaussian distribution merges with the observation model, different initial P converges to the same value after a large amount of iterations, which also hold true for different starting point estimation.

4.4 Particle Filter Discussion

4.4.1

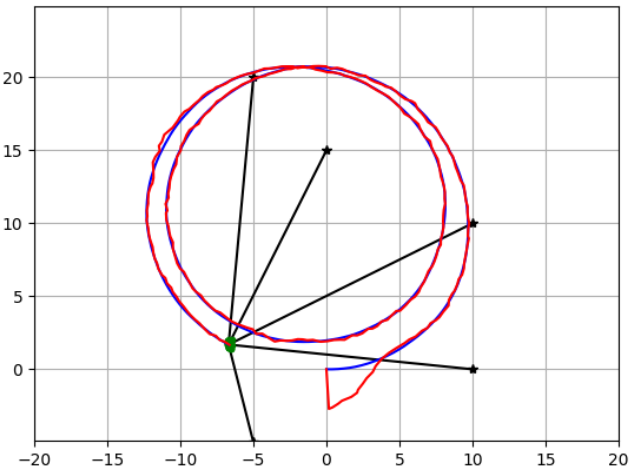

Figure 16: PF path

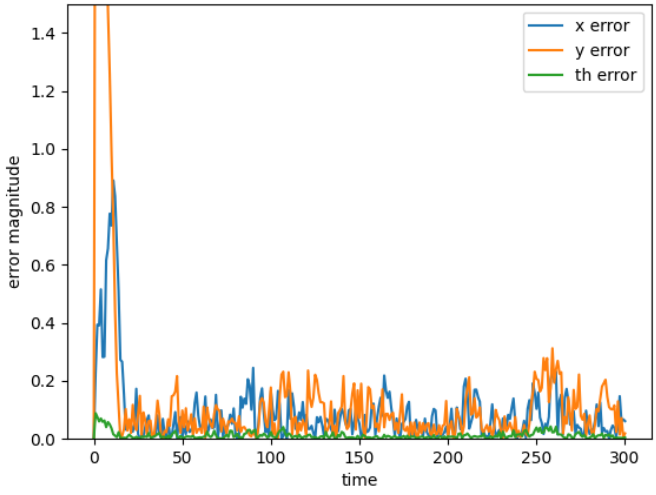

Figure 17: PF error

Initially, all the particles is evenly distributed in the space, but as the particle filter re samples the particle, new particles converge to the true position in a few iterations. From the error plot, we conclude that the particle filter tracks the object with a low error.

4.2.2

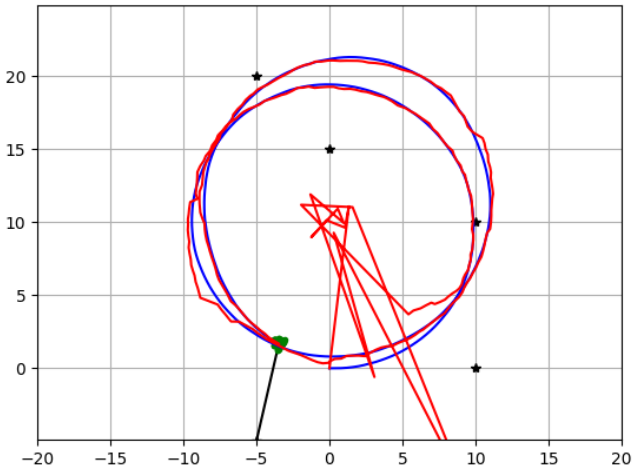

Figure 18: PF path with sensor range to be 10

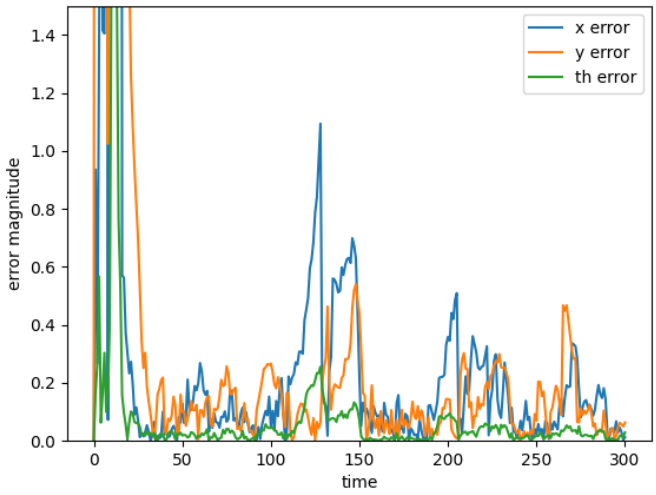

Figure 19: PF error with sensor range to be 10

After decrease the MAXRANGE to 10, the particle filter experience the same problem as the EKF does. Moreover, particles takes a longer time to converge to the real position since the number of visible landmarks decrease. Also, when the sensor does not detect any landmarks, particles updates using the motion model. The noise in the motion model cause the prediction deviates from the true value.

4.4.3

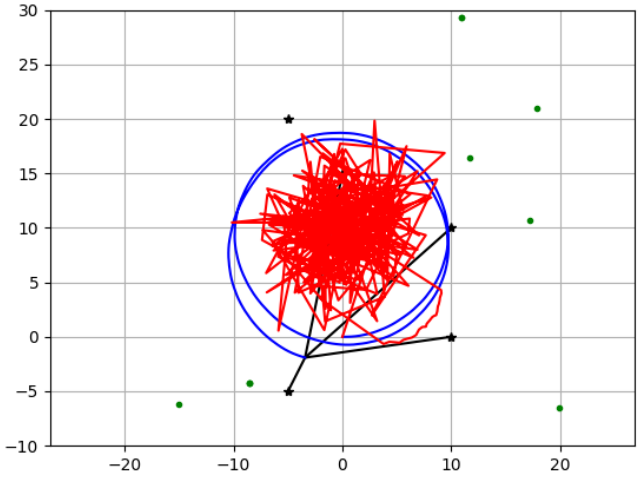

Figure 20: PF path with NP to be 10

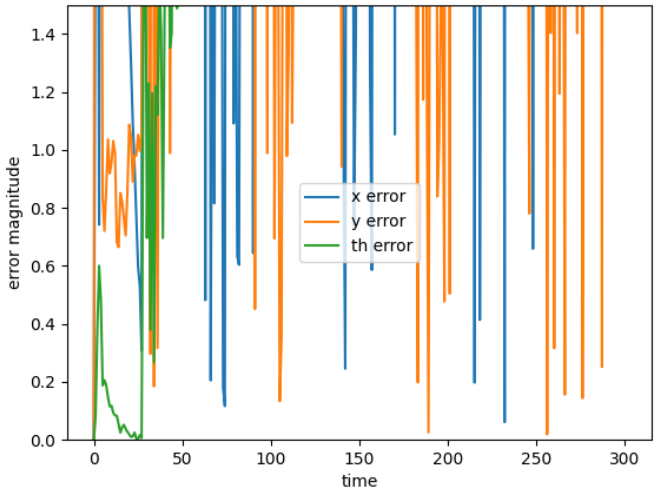

Figure 21: PF error with NP to be 10

After change the number of particle to 10, we observe the particle filter performs poorly. The reason for such behavior is caused by decreasing the number of particles, since particle filter uses particles to estimate the real distribution, 10 particles may be bad prediction which significantly deviate from the true distribution. The 10 particles are re sampled as

the particle filter executes without adding new particles, but no particle is close to the real distribution. Thus the particle filter output garbage result.

Appendix:

1. Question4 EKF-localization Code

```
"""
Extended Kalman filter procedure
"""

def EKF(x, P, u, z):
    x, P = predict(x, P, u)
    x, P = update(x, P, z)
    print(x[2])
    return x, P


def predict(x, P, u):
    """
    :param x: State mean (x,y,theta) [size 3 array]
    :param P: State covariance [3x3 array]
    :param u: Robot inputs (u1,u2) [size 2 array]
    :return: Predicted state mean and covariance x and P
    """
    theta = x[2]
    u1 = u[0]
    u2 = u[1]
    motion_model = np.array([DT * (RHO/2) * np.cos(theta) * (u1+u2)
                          ,DT * (RHO/2) * np.sin(theta) * (u1+u2)
                          ,DT * (RHO/L) * (u2-u1)])
    x = x + motion_model + np.random.multivariate_normal(np.zeros(3), Q)
    #print(x)

    Fk = np.array([[1,0,-DT * RHO * 0.5 * np.sin(theta) * (u1+u2)]
                , [0,1,DT * RHO * 0.5 * np.cos(theta) * (u1+u2)]
                , [0,0,1]])
    P = Fk @ P @ Fk.T + Q
    return x, P


def update(x, P, z):
    """
    :param x: State mean (x,y,theta) [size 3 array]
    :param P: State covariance [3x3 array]
    :param z: Sensor measurements [px3 array]. Each row contains range, bearing,
       and landmark's true (x,y) location.
    :return: Updated state mean and covariance x and P
```

```
    """
    #compute the innovation error, Hk and Rk
    true_r_phi, indexs = z[:,:2], z[:,2]
    true_r_phi = true_r_phi.reshape((-1))
    xk, yk, theta = x[0], x[1], x[2]
    #print(xk, yk, theta)
    p = len(z[:,0])
    predict_r_phi = []
    Hk = np.zeros([0,3])
    Rk = np.eye(2 * p)
    for i in range(p):
        index = int(indexs[i])
        landmark_x, landmark_y = RFID[index,:]
        #print(landmark_x, landmark_y)
        #innovation error
        predict_r = np.sqrt((landmark_x - xk)**2 + (landmark_y -yk)**2)
        predict_phi = np.arctan2(landmark_y - yk, landmark_x - xk) - theta
        #print(predict_r, predict_phi)
        predict_r_phi.append(predict_r)
        predict_r_phi.append(predict_phi)
        #Hk
        Hk_i = np.array([[(xk-landmark_x)/predict_r, (yk-landmark_y)/predict_r, 0]
                        ,[-(yk-landmark_y)/(predict_r)**2,
                            (xk-landmark_x)/(predict_r)**2,-1]])
        Hk = np.vstack((Hk, Hk_i))
        #Rk
        Rk[2*i:2*i+2, 2*i:2*i+2] = R
    #innovation_error
    innovation_error = true_r_phi - predict_r_phi
    for i in range(p):
        if innovation_error[2*i+1] < -np.pi:
            innovation_error[2*i+1] += 2*np.pi
        elif innovation_error[2*i+1] > np.pi:
            innovation_error[2*i+1] -= 2*np.pi

    if not len(Hk):
        return x, P

    Sk = Hk @ P @ Hk.T + Rk
    Kk = P @ Hk.T @ np.linalg.inv(Sk)
    KkHk = Kk @ Hk
    P = (np.identity(KkHk.shape[0]) - KkHk) @ P
    x = x + (Kk @ innovation_error).ravel()
    return x, P
```

1. Question4 PF-localization Code

```
import math
from re import T
from typing import List
import matplotlib.pyplot as plt
import numpy as np
import collections
from scipy.sparse.construct import random
from scipy.stats import multivariate_normal
import random

"""
Particle filtering procedures
"""
def gaussianest(x):
    size = 2
    det = np.linalg.det(R)
    norm_const = 1.0/ ( (2*math.pi) * math.pow(det,1.0/2) )
    inv = np.linalg.inv(R)
    result = np.exp(-0.5 * (x.dot(inv).dot(x.T)))
    return norm_const * result

def motionModel(x, u):
    theta = x[2]
    u1, u2 = u[0], u[1]

    deltax = DT * (RHO/2) * np.cos(theta)*(u1 + u2)
    deltay = DT * (RHO/2) * np.sin(theta)*(u1 + u2)
    deltahTheta = (RHO/L)*(u2-u1)*DT

    return np.array([deltax, deltay, deltahTheta])

def predict(x, u):
    """
    :param x: Particle state (x,y,theta) [size 3 array]
    :param u: Robot inputs (u1,u2) [size 2 array]
    :return: Particle's updated state sampled from the motion model
    """
    x = x + motionModel(x, u) + np.random.multivariate_normal(np.zeros(3), Q)

    return x


def update(x, z):
```

```
    """
    :param x: Particle state (x,y,theta) [size 3 array]
    :param z: Sensor measurements [px3 array]. Each row contains range, bearing,
       and landmark's true (x,y) location.
    :return: Particle's updated weight
    """
    trueR, trueB, IDList = z[:,0], z[:,1], z[:,2]
    X, Y, Theta = x[0], x[1], x[2]
    observedList = []
    for ID in IDList:
        observedList.append(int(ID))

    weight = 1
    for k in range(len(IDList)):
        index = int(IDList[k])
        trueX, trueY = RFID[index]
        r = np.sqrt( (trueX - X)**2 + (trueY - Y)**2 )
        phi = np.arctan2(trueY-Y, trueX-X) -Theta

        if r <= MAX_RANGE and index in observedList:
            deltaPhi = trueB[k] - phi
            nu = np.array([trueR[k] - r, deltaPhi])
            weight *= gaussian.pdf(nu)
        else:
            weight = 0
            break

    return weight


def resample(px, pw):
    """
    :param px: All current particles [3xNP array]
    :param pw: All particle weight [size NP array]
    :return: A new set of particles obtained by sampling the original particles
       with given weights
    """
    pw = pw[0]
    idx2data = collections.defaultdict(list)
    NORM = sum(pw)
    indexlist = []
    for i in range(len(pw)):
        pw[i] /= NORM
        idx2data[i] = [px[0][i], px[1][i], px[2][i]]
```

```
            indexlist.append(i)
        x = np.random.choice(indexlist, len(pw), p=pw)
        for i in range(NP):
            px[0][i] = idx2data[x[i]][0]
            px[1][i] = idx2data[x[i]][1]
            px[2][i] = idx2data[x[i]][2]

        return px
```