Solutions for HW8

Yunhai Han

March 7, 2020

1 1

1.1 a

As given in the title:A sensor measurement $z = (z_r, z_\theta)^T$is composed of the measured distance z_r and the measured bearing z_θ to the landmark l. Both the range and the bearing measurements are subject to zero-mean Gaussian noise with variances σ_r^2 , and σ_θ^2 , respectively. The range and the bearing measurements are independent of each other. From the lecture, we know if the measurements are independent, the sensor model could be described as:

$$p\left(\mathbf{z}_t|\mathbf{x}, \mathbf{l}\right) = \Pi_{k=1}^{K} p\left(z_t^k|\mathbf{x}_t, \mathbf{l}\right) \tag{1}$$

Here, $K = 2$ because there are only two measurements.

For each of them, there is only small different with respect to how to obtain $z_t^{k\star}$.

- For the range measurement z_r, we could simply compute the distance between the robot's position(x, y) and the landmark $l = (l_x, l_y)$:

$$z_r^{k\star} = \sqrt{\left(l_x - x\right)^2 + \left(l_y - y\right)^2} \tag{2}$$

- For the bearing measurement z_θ, we could compute the angle between the robot's heading direction and the connecting segment with robot and the landmark l:

$$z_\theta^{k\star} = \operatorname{atan2}\left(l_y - y, l_x - x\right) - \theta \tag{3}$$

Then, since in the part (a), we have no idea of the maximum range value, we just compute the probability from the Gaussian noise distribution:

$$p_{\text{hit}}\left(z_r^k|\mathbf{x}_t, \mathbf{l}\right) = \eta \frac{1}{\sqrt{2\pi\sigma_r^2}} \exp\left(-\frac{\left(z_r^k - z_r^{k\star}\right)^2}{2\sigma_r^2}\right) \tag{4}$$

Besides, we could assume that there are some unexpected objects between the robot and the landmark l, which means sometimes the measurements are not wrong but it hits some objects which are not marked in the map.

$$p_{\text{short}}\left(z_r^k|\mathbf{x}_t,\mathbf{l}\right)=\begin{cases}\eta\lambda_{\text{short}}\,\mathrm{e}^{-\lambda_{\text{stort}}\,z_r^k} & z_r^k\le z_r^{k\star}\\ 0 & \text{otherwise}\end{cases}\tag{5}$$

In the above two equations, η are the normalizers. Then, we need to combine the two probabilities:

$$p(z_r|\mathbf{x},\mathbf{l})=\begin{bmatrix}\alpha_{\text{hit}} & \alpha_{\text{short}} & \alpha_{\max} & \alpha_{\text{rand}}\end{bmatrix}\begin{bmatrix}p_{\text{hit}}\left(z|\mathbf{x},\mathbf{l}\right)\\ p_{\text{short}}\left(z|\mathbf{x},\mathbf{l}\right)\\ p_{\max}(z|\mathbf{x},\mathbf{l})\\ p_{\text{rand}}\left(z|\mathbf{x},\mathbf{l}\right)\end{bmatrix}\tag{6}$$

Indeed, in our case, since we have no idea of the maximum value, $\alpha_{\max}$ and α_{rand} could always be zeros. Besides, the sum of α_{hit} and α_{short} should be one.

Hence, the probability for the range measurement is $p(z_r|\mathbf{x},\mathbf{l})$.

The probability for the bearing measurement could be computed in the same way.

$$p\left(z_\theta|\mathbf{x},\mathbf{l}\right)=\eta\frac{1}{\sqrt{2\pi\sigma_\theta^2}}\exp\left(-\frac{\left(z_\theta^k-z_\theta^{k\star}\right)^2}{2\sigma_\theta^2}\right)\tag{7}$$

Finally, the probability $p\left(z^k|\mathbf{x}_t,\mathbf{l}\right)$ is the product of the twos:

$$p\left(z|\mathbf{x},\mathbf{l}\right)=p(z_r|\mathbf{x},\mathbf{l})*p\left(z_\theta|\mathbf{x},\mathbf{l}\right)\tag{8}$$

1.2 b

If we know the maximum value for the range measurement, we could add another two probability distributions into the range measurement and keep the bearing measurement model unchanged.

The first one is:

$$p_{\max}\left(z_r^k|\mathbf{x}_t,\mathbf{m}\right)=\eta\begin{cases}1 & \left|z_r^k-z_{\max}\right|\le\varepsilon\\ 0 & \text{otherwise}\end{cases}\tag{9}$$

The second one is:

$$p_{\text{rand}}\left(z_r^k|\mathbf{x}_t,\mathbf{m}\right)=\begin{cases}\frac{1}{z_{\max}} & 0\le z_r^k\le z_{\max}\\ 0 & \text{otherwise}\end{cases}\tag{10}$$

We could compute the probability using the same equation in equ 6. However, in this modified case, all the four parameters are not zero and the sum of them should be one.

The values of the four parameters could be obtained by experiments(real data).

The probability is also computed from equ 8.

2 2

2.1 a

In this part, we are required to implement a probabilistic motion model given a truncated Gaussian probability distribution.

We know the length of the wall is [-50,50], so the x position of the robot should be within this range. Suppose x_t and u_t are given, the mean value of x_{t+1} is $x_t + u_t$. Even with some random noise, the output should not be outside the range. In the title, there are five different situations and if we follow the five rules, the output could be guaranteed to be within the range.

The steps to compute the truncated Gaussian distribution are simple:

- Compute the mean value(given as 0) and variance(given as 0.5)
- Compute the integration of the Guassian probability from -1 to 1.(We could use python function *scipy.stats*)
- Take the integration value as the normalizer and set the probability to zero for all the regions outside [-1,1].

After we obtain the truncated Gaussian distribution, we need to use the rejection sampling method to draw samples from it. The procedures could be described as follow:

let g be the uniform distribution over $[-1, 1]$. Assume that $\max f(x) = y_{\max}$ and let $U\left[0, y_{\max}\right]$ be the uniform distribution over $[0, y_{\max}]$.

Basic rejection sampling for general distributions
Input: Distribution f, g and $U\left[0, y_{\max}\right]$, the uniform distribution over $[0, y_{\max}]$.
Output: a random sample distributed according to f
1: Obtain a random sample y distributed as g in $[x_{\min}, x_{\max}]$
2: Obtain a random saple u distributed as $U\left[0, y_{\max}\right]$
3: while $u > f(y)$:
4: reject the sample and repeat steps
5: accept the sample and return y

Also, at the very beginning, we need to check which situation the inputs belong to.

2.2 b

The sensor model has been provided and I only need to check the robot's position and the measurement data at that moment.

$P(z = |1|0.5 \leq x \leq 1, \text{ or } 2 \leq x \leq 2.5 \text{ or } 22.5 \leq x \leq 23) = 0.7$

$P(z = 0|x \text{ at door locations }) = 0.3$

$P(z = 0|x \text{ not at door locations }) = 0.85$

$P(z = 1|x \text{ not at door locations }) = 0.15$

2.3 c

In this part, we are required to implement a particle filter given the control sequence and the sensor sequence. The procedures are shown as follow:

- Consider $M = 300$ (or larger) Initialize at $t = 0$ with $\mathcal{P}_0 = \left\{ x_0^{[m]} | m \text{ goes from } 1 \text{ to } 300 \right\}$ such that
$$x_0^{[m]} = \text{ uniform } [-50, 50]$$
- After $t = 1$, assume we use data $u_1 = 1$ and $z_1 = 0$ Then, for each m from 1 to 300 We predict the next particles:
$$\bar{x}_1^{[m]} = x_0^{[m]} + 1 + \text{sample } \left(\sigma^2\right)$$
- Compute the weights
$$w_1^{[m]} = P\left(z_1 = 0|\bar{x}_1^{[m]}\right) = P\left(\text{ no door } |\bar{x}_1^{[m]}\right)$$
- We reassign the weights. For each m compute
$$\tilde{w}_1^{[m]} = \frac{w_1^{[m]}}{\sum_{n=1}^{300} w_1^{[n]}}$$
- For each m from 1 to M, apply basic rejection sampling:

  1 : Draw a random number from 1 to 300, say it is i

  2 : Draw a sample of the uniform distribution over $[0, 1]$, say it is c

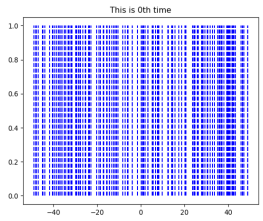
  3 : if $c \geq \tilde{w}_1^{[i]}$, reject and repeat steps $1 - 2$

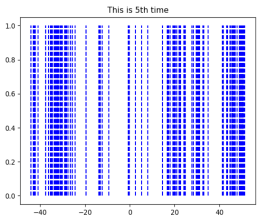
  4 : else, accept i

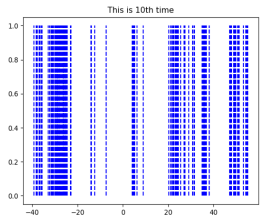
  5 : Insert $\bar{x}_1^{[1]}$ as particle number m, that is, $x_1^{[m]} = \bar{x}_1^{[1]}$ in the set $\mathcal{P}_1$

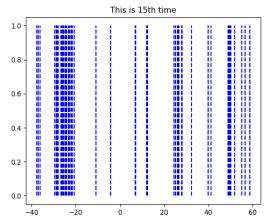
2.4 Results

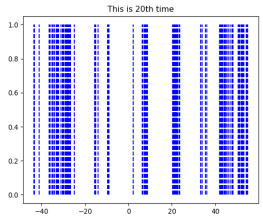
The particle filter could be function of the number of particles. In the following case, I select 200 particles and show how they evolve at different snapshots.

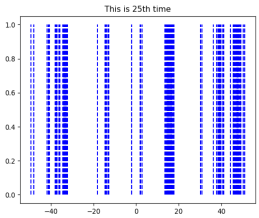


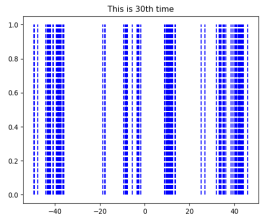


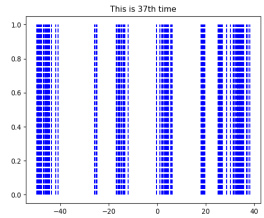












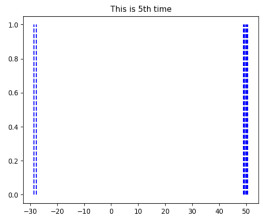
In the above figures, the first one represents the distribution of all the particles generated from the uniform probability function. Indeed, there are in total 38 figures, but I only put some of them here. As mentioned in the slides, there are some problems which arise from the finite number of particles and the resampling procedures.

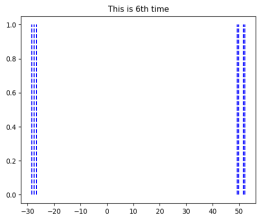
A repeated resampling in the absence of new sensor measurements or for static states ($x_t = x_{t-1}$) can lead to a problem of loss of diversity in the particles, or a particle deprivation problem. What this means is that we will end up having many repeated particles, being an extreme case that, after some iterations, a single particle survives and the rest are discarded.

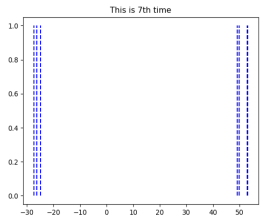
I think this problem is much more severe when the number of particles is small, because it could greatly lose diversity which is significant for the success of the algorithm.

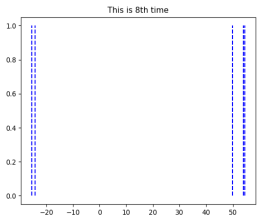
This problem could happen when the robot is static and no new observations are obtained. There are two special cases:1.when the robot is around the corners, it would not move outside the range(from the motion model);2.when $t = 15, 16$, the control input is zero.

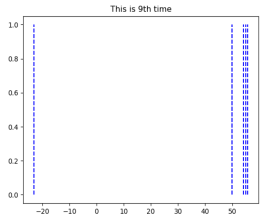
However, since we use 200 particles here, the problem is not severe. Hence, I decide to change the number of particles and this time, it is 10.

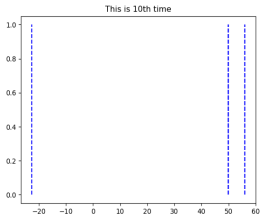


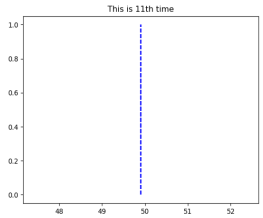


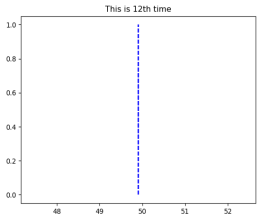


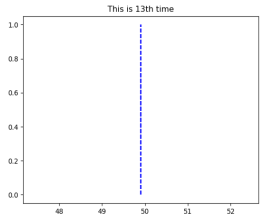


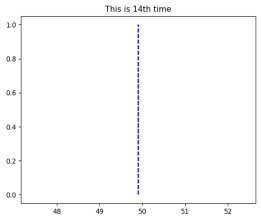


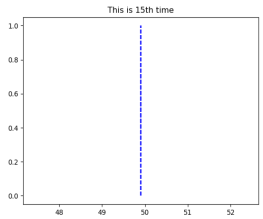


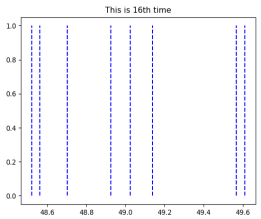












In my algorithm, I start from 0th, so the 16th represent the 17th control input(-1). You could see during several iterations, all the ten particles represent the same robot's position, greatly decreasing the diversity.

3 3

In the problem description, the constant h should represents the distance between the ceiling and the camera plane. These two planes are parallel to each other, so there is only an offset along z-axis.

Here I put global coordinate system on the camera plane and the three axis are fixed(x forward, y left and z up). So the world coordinates of the marker in the global coordinate system should be x_m, y_m, h. Besides, the marker is not a point, so the orientation matters.

3.1 a

The camera perspective equation is simple:

$$h \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ h \\ 1 \end{pmatrix} \tag{11}$$

I assume the origin of the image plane is the center of the image, so I don't have to consider c_x, c_y. In the above equation, all the variables are known except x_v, y_v. They represent the marker's position in the camera frame and they could be obtained by the above equation. Besides, the orientation in the image coordinates could be computed as $\theta_i = \arctan \frac{y_v}{x_v}$

Since we know the relative position of the camera frame(it is fixed on the robot) to the global coordinate by x_r, y_r, θ_r, we could compute the marker's position in the global coordinate:

$$\begin{aligned} x_m &= x_r + x_v * \sin\theta_r - y_v * \cos\theta_r \\ y_m &= y_r + x_v * \cos\theta_r + y_v * \sin\theta_r \\ \theta_m &= \theta_r - \theta_i \end{aligned} \tag{12}$$

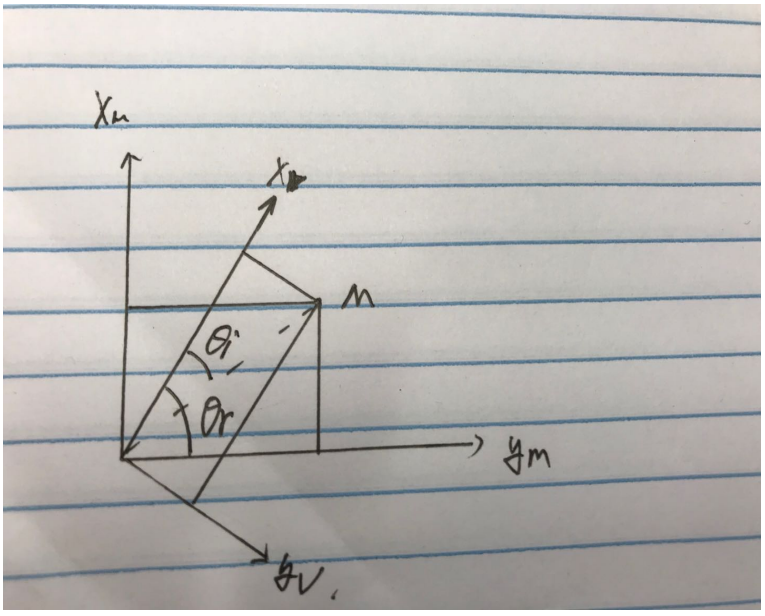


Figure 1: The transformation

3.2 b

There is only a small difference between part(b) and part(a). If we change the computing order, we could obtain what part(b) requires. With the same equation:

$$\begin{aligned} x_m &= x_r + x_v * \sin\theta_r - y_v * \cos\theta_r \\ y_m &= y_r + x_v * \cos\theta_r + y_v * \sin\theta_r \\ \theta_m &= \theta_r - \theta_i \end{aligned} \tag{13}$$

This time, x_m, y_m and θ_m are known, and we could compute x_v, y_v and θ_i. After this, by the camera's perspective model:

$$h \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ h \\ 1 \end{pmatrix} \tag{14}$$

We could compute x_i, y_i.

3.3 c

According to the same equations, we could obtain the results required in this part.

$$\begin{aligned} x_m &= x_r + x_v * \sin\theta_r - y_v * \cos\theta_r \\ y_m &= y_r + x_v * \cos\theta_r + y_v * \sin\theta_r \\ \theta_m &= \theta_r - \theta_i \end{aligned} \tag{15}$$

This time, we already know x_m, y_m, θ_m and x_v, y_v, θ_i(from x_i, y_i, θ_i using camera's perspective model). Thus, we could compute x_r, y_r, θ_r.

However, the difference is that there may be multiple solutions. I draw a picture to illustrate it:

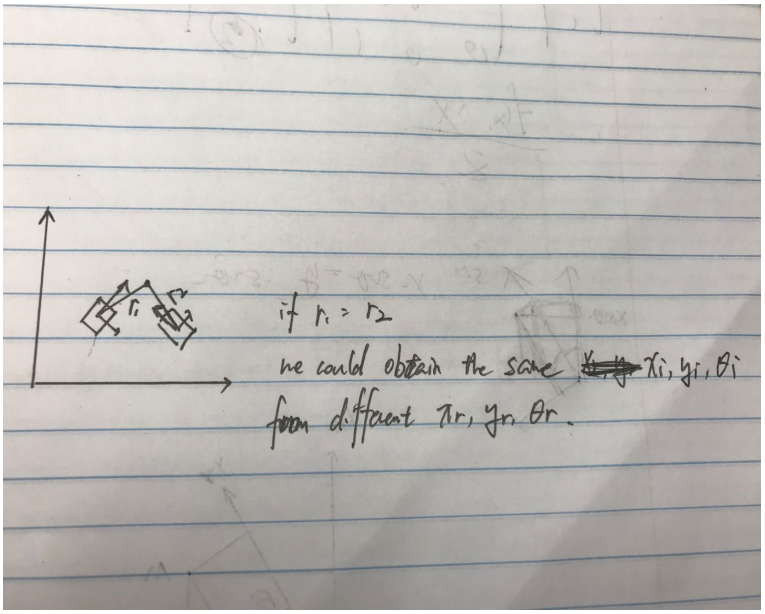


Figure 2: An example

The reason behind it is very simple:

When the robot is on the different sides of the marker, θ_m could be computed in different ways. For example, in the above figure, if the robot is on the left side, θ_m could be computed by:$\theta_m = \theta_r - \theta_i$. However, if the robot is on the right side, it could be computed by:$\theta_m = \pi - (\theta_r - \theta_i)$.

Indeed, if we rotate the robot around the marker at a certain distance and adjust its orientation, we could obtain the same results.

Thus, if wa want to unique localize the robot, we need to either know the relative position of the robot to the marker or detect multiple markers at the same time.

3.4 d

If we could detect multiple markers at the same time, it could uniquely help us identify robot's pose.

Two marker detected at the same time are enough. Here I draw such configuration to explain it.

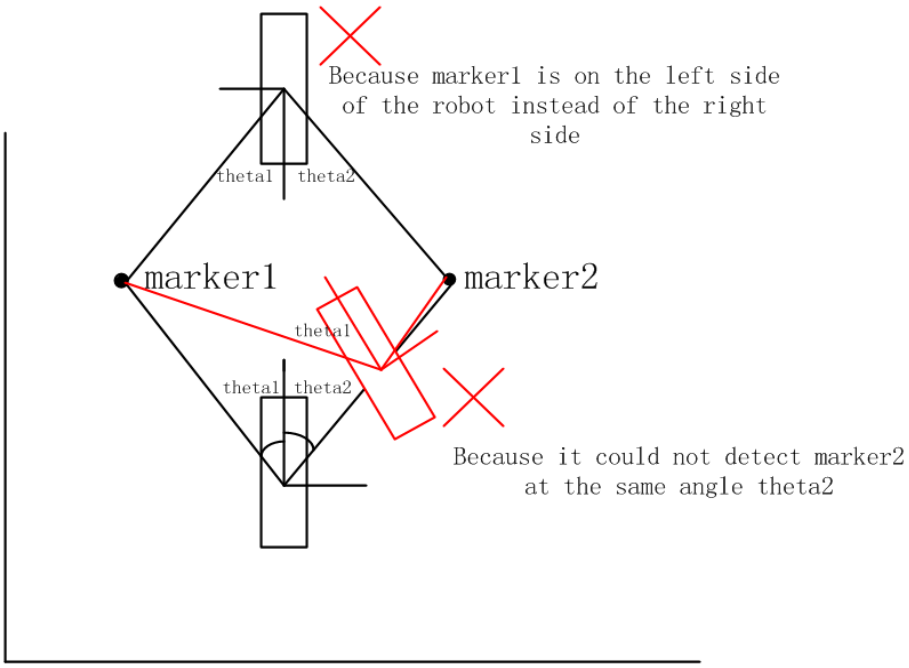


Figure 3: Configuration for two markers

From the above figure, if we can detect two markers at the same time and associate them perfectly, we could uniquely identify the robot's pose.

4 Codes

Here I put the codes in problem2.

```
#A53307224
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
import intervals as I
import matplotlib.pyplot as plt

def samplemethod(mu=0, sigma=0.5, x_t = 0, u_t = 0):
    if u_t == 0:
        x_t_1 = x_t
    elif x_t in I.closed(-50, -49) and u_t == -1:
        x_t_1 = x_t
```

```
    elif x_t in I.closed(49, 50) and u_t == 1:
        x_t_1 = x_t
    else:
        tmp1 = scipy.stats.norm(0,sigma).cdf(-1)
        tmp2 = scipy.stats.norm(0,sigma).cdf(1)
        n = tmp2 - tmp1 #the cumulation over [-1,1]
        n = 1/n #normalizer
        y_max = n/np.sqrt(2*np.pi * sigma) * np.exp(-(0)**2/(2*sigma))
        u = np.random.uniform(0, y_max)
        v = np.random.uniform(-1, 1)
        proba = n/np.sqrt(2*np.pi * sigma) * np.exp(-(v)**2/(2*sigma))
        while u > proba:
            u = np.random.uniform(0, y_max)
            v = np.random.uniform(-1, 1)
            proba = n/np.sqrt(2*np.pi * sigma) *
                np.exp(-(v)**2/(2*sigma))
        x_t_1 = x_t + u_t + v
    return x_t_1
def sensor_method(x_t_1 = 0, z_t = 0):
    if x_t_1 in I.closed(0.5, 1) or x_t_1 in I.closed(2, 2.5) or
        x_t_1 in I.closed(22.5, 23):
        if z_t == 1:
            return 0.7
        elif z_t == 0:
            return 0.3
        else:
            print("Invalid input z_t")
            return 0
    else:
        if z_t == 1:
            return 0.15
        elif z_t == 0:
            return 0.85
        else:
            print("Invalid input z_t")
            return 0
def draw_particles(index, particles):
    plt.title("This is " + str(index) + "th time")
    for i in particles:
        plt.vlines(i, 0, 1, colors = "b", linestyles = "dashed")
    plt.savefig(str(index)+".png")
    plt.pause(0.1)
def rejection_sampling(x_t, proba):
    c = np.random.uniform(0, 1)
    if c >= proba:
        return True
    else:
        return False
def particle_fileter(u_t, z_t, number = 5):
```

```
    plt.ion()
    #initialization
    particles = np.random.uniform(-50, 50, number)
    proba = [1 / number] * number
    plt.cla()
    draw_particles(-1, particles)
    #prediction & update
    for i in range(0, len(u_t)):
        particles_tmp = [0] * number
        for j in range(0, len(particles)):
            particles_tmp[j] = samplemethod(mu = 0, sigma = 0.5, x_t =
                particles[j], u_t = u_t[i])
            proba[j] = proba[j] * sensor_method(particles_tmp[j],
                z_t[i])
        sum_cr = sum(proba)
        for j in range(0, len(particles)):
            proba[j] = proba[j] / sum_cr #normalization
        for j in range(0, len(particles)):
            index = int(np.floor(np.random.uniform(0, len(particles))))
            repeat = rejection_sampling(particles_tmp[index],
                proba[index])
            while repeat:
                index = int(np.floor(np.random.uniform(0,
                    len(particles))))
                repeat = rejection_sampling(particles_tmp[index],
                    proba[index])
            particles[j] = particles_tmp[index]
            proba[j] = 1 / number
        plt.cla()
        draw_particles(i, particles)
    plt.ioff()
    plt.show()
if __name__ == "__main__":
    u_t = [1]*14+[0]*2+[-1]*22
    z_t =
        [0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0]
    number = 10
    particle_fileter(u_t, z_t, number)
```