海中ロボット学
濱松祐矢

```
from math import *
import random
import numpy as np
import matplotlib.pyplot as plt

landmarks   = [[0.0, 10.0]]
world_size = 200.0
t = 0 # sec

class robot:
    def __init__(self):
        self.x = np.random.normal(0, 2.0)
        self.y = np.random.normal(0, 1.0)
        self.orientation = pi /2 + np.random.normal(0, 0.026)
        self.x_noise = 0.0;
        self.y_noise = 0.0;
        self.yaw_noise      = 0.0;
        self.sense_noise     = 0.0;

    def set(self, new_x, new_y, new_orientation):
        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation)



    def set_noise(self, new_x_noise, new_y_noise, new_yaw_noise, new_sense_noise):
        self.x_noise = float(new_x_noise);
        self.y_noise       = float(new_y_noise);
        self.yaw_noise     = float(new_yaw_noise);
        self.sense_noise     = float(new_sense_noise);

    def sense(self):
        Z = []
```

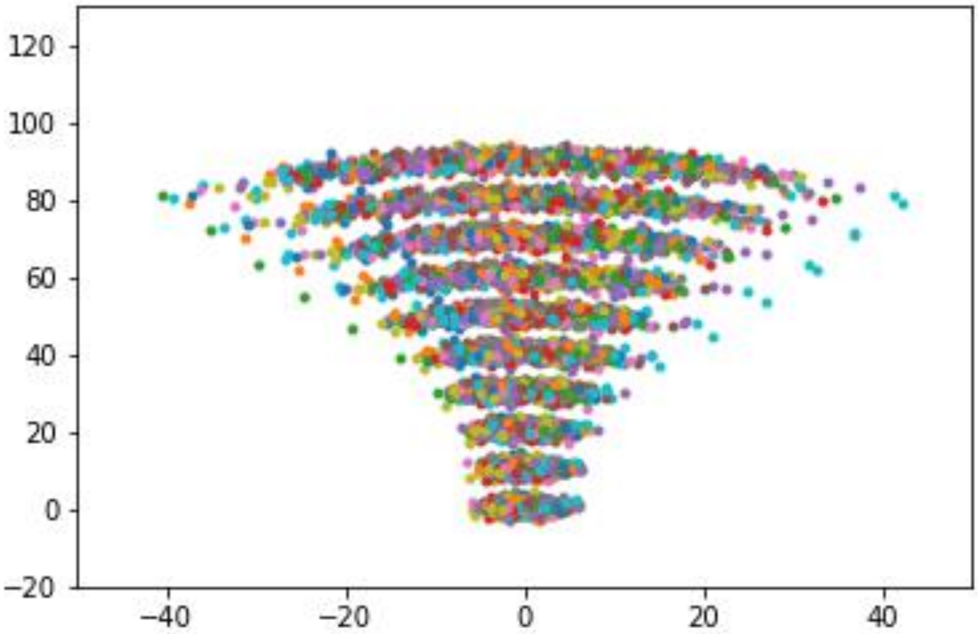
```
        for i in range(len(landmarks)):
            dist = np.sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
            dist *= np.random.normal(0.0, self.sense_noise)
            Z.append(dist)
        return Z


    def move(self, turn, forward):

        orientation = self.orientation + float(turn) + np.random.normal(0.0, 
self.yaw_noise)
        orientation %= 2 * pi

        dist = float(forward)
        x = self.x + (np.cos(orientation) * dist) + np.random.normal(0.0, self.x_noise)
        y = self.y + (np.sin(orientation) * dist) + np.random.normal(0.0, self.y_noise)


        # set particle
        res = robot()
        res.set(x, y, orientation)
        res.set_noise(self.x_noise, self.y_noise, self.yaw_noise, self.sense_noise)
        return res

    def Gaussian(self, mu, sigma, x):

        # calculates the probability of x for 1-dim Gaussian with mean mu and var. 
sigma
        return exp(- ((mu - x) ** 2) / (sigma ** 2) / 2.0) / sqrt(2.0 * pi * (sigma ** 2))


    def measurement_prob(self, measurement):

        # calculates how likely a measurement should be

        prob = 1.0 ;
```

```
        for i in range(len(landmarks)):
            dist = np.sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
+ np.random.normal(0.0, self.sense_noise)
           #    prob *= self.Gaussian(dist, self.sense_noise, measurement[i])
        return dist

    def __repr__(self):
        return '[x=%.6s y=%.6s orient=%.6s]' % (str(self.x), str(self.y),
str(self.orientation))



N = 1000
p = []

myrobot = robot()
myrobot.set(0.0, 0.0, pi / 2)
#for i in range(N):
   #    p[i].set(0.0, 0.0, pi / 2)




for i in range(N):
      x = robot()
      x.set_noise(0.1, 0.1, 0.026, 0.5) # x,y,rad,sense
      p.append(x)



for i in range(100):

  myrobot = myrobot.move(0.0, 1.0)
#    myrobot.set_noise(0.2, 0.2, 0.026, 0.5)

  p2 = []
  for i in range(N):
    p2.append(p[i].move(0.0, 1.0))
```

```
    p = p2




    if t % 10 == 0 :
        for i in range(N):
            plt.plot(p[i].x, p[i].y, '.')


    t =t + 1
    print(t)
plt.ylim(-20, 130)
plt.xlim(-50, 50)
plt.show()
```

----Run----

```
runfile('C:/Users/hama6767/Documents/underwater_robotics_5_particle_filter/pf1.py', wdir='C:/Users/hama6767/Documents/underwater_robotics_5_particle_filter')
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```


120
100
80
60
40
20
0
−20
−40
−20
0
20
40

```
from math import *
import random
import numpy as np
import matplotlib.pyplot as plt

landmarks   = [[0.0, 10.0]]
world_size = 200.0
t = 0 # sec

class robot:
    def __init__(self):
        self.x = np.random.normal(0, 1.0)
        self.y = np.random.normal(0, 1.0)
        self.orientation = pi /2 + np.random.normal(0, 0.026)
        self.x_noise = 0.2;
        self.y_noise = 0.4;
        self.yaw_noise      = 0.026;
        self.sense_noise     = 0.0;

    def set(self, new_x, new_y, new_orientation):
        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation)



    def set_noise(self, new_x_noise, new_y_noise, new_yaw_noise, new_sense_noise):
        # makes it possible to change the noise parameters
        # this is often useful in particle filters
        self.x_noise = float(new_x_noise);
        self.y_noise      = float(new_y_noise);
        self.yaw_noise     = float(new_yaw_noise);
        self.sense_noise     = float(new_sense_noise);
```

```
    def sense(self):
        Z = []
        for i in range(len(landmarks)):
            dist = np.sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
            dist *= np.random.normal(0.0, self.sense_noise)
            Z.append(dist)
        return Z


    def move(self, turn, forward):

        # turn, and add randomness to the turning command
        orientation = self.orientation + float(turn) + np.random.normal(0.0,
self.yaw_noise)
        orientation %= 2 * pi

        # move, and add randomness to the motion command
        dist = float(forward)
        x = self.x + (np.cos(orientation) * dist) + np.random.normal(0.0, self.x_noise)
        y = self.y + (np.sin(orientation) * dist) + np.random.normal(0.0, self.y_noise)



        # set particle
        res = robot()
        res.set(x, y, orientation)
        res.set_noise(self.x_noise, self.y_noise, self.yaw_noise, self.sense_noise)
        return res

    def Gaussian(self, mu, sigma, x):

        # calculates the probability of x for 1-dim Gaussian with mean mu and var.
sigma
        return exp(- ((mu - x) ** 2) / (sigma ** 2) / 2.0) / sqrt(2.0 * pi * (sigma ** 2))
```

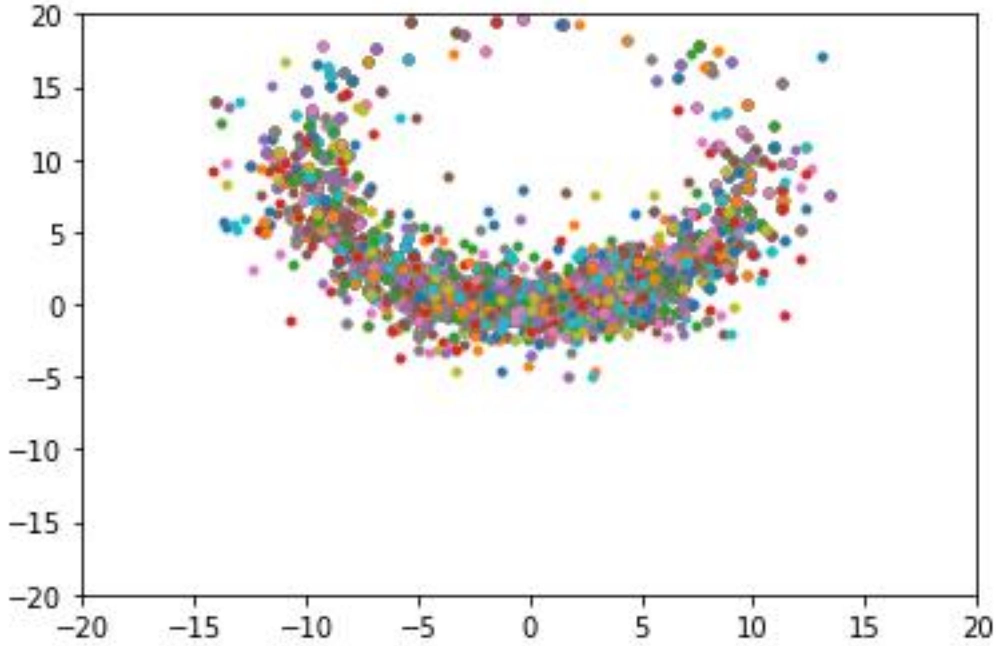
```
    def measurement_prob(self, measurement):

        # calculates how likely a measurement should be

        prob = 1.0 ;
        for i in range(len(landmarks)):
            dist = np.sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
+ np.random.normal(0.0, self.sense_noise)
           #    prob *= self.Gaussian(dist, self.sense_noise, measurement[i])
        return dist

    def __repr__(self):
        return '[x=%.6s y=%.6s orient=%.6s]' % (str(self.x), str(self.y),
str(self.orientation))




myrobot = robot()
myrobot.set(0.0, 0.0, pi/2)

N = 700
p = []
for i in range(N):
      x = robot()
      x.set_noise(1.0, 1.0, 0.026, 0.5) # x,y,rad,sense
      p.append(x)



for i in range(100):

  myrobot = myrobot.move(0.0, 0.0)
  myrobot.set_noise(1.0, 1.0, 0.026, 0.5)

  p2 = []
  for i in range(N):
    p2.append(p[i].move(0.0, 0.0))
```

```
p = p2

if t % 10 ==0 :
    Z = myrobot.sense()
    w = []
    wp = 0
    for i in range(N):
      w.append(p[i].measurement_prob(Z))
    p3 = []
    p3.append(p[1])

    for i in range(N-1):
        if abs(10.0 - w[i]) >    abs(10.0 - w[i+1]) :
                p3.append(p[i+1])
        else:
                if wp < 7:
                    p3.append(p[i])
                    p[i+1].x = p[i].x #+ np.random.normal(0.0, 0.05)
                    p[i+1].y = p[i].y #+ np.random.normal(0.0, 0.05)
                    w[i+1] = w[i]
                    wp += 1

                else:
                    p3.append(p[i])
                    wp = 0

    p = p3
    random.shuffle(p)

if t % 10 == 0 :
   for i in range(N):
    plt.plot(p[i].x, p[i].y, '.')



t =t + 1
```

```
    print(t)
plt.xlim(-20, 20)
plt.ylim(-20, 20)
plt.show()
```

----Run-----

```
runfile('C:/Users/hama6767/Documents/underwater_robotics_5_particle_filter/pf.py',
wdir='C:/Users/hama6767/Documents/underwater_robotics_5_particle_filter')
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20