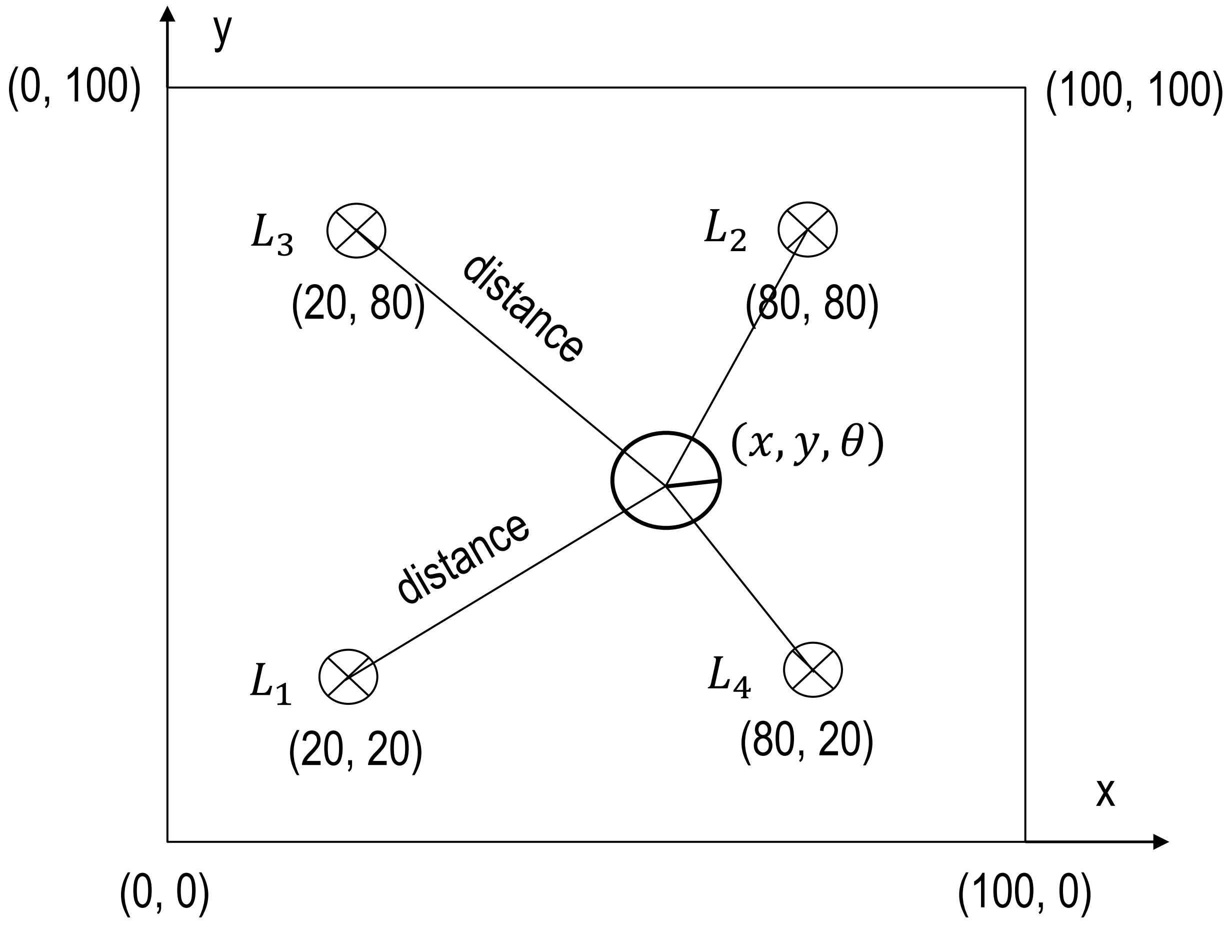
Mobile Robotics

Particle Filter Assignments

Robot World

- One robot and four landmarks inside a square are shown below.



Python Code: Robot Class - Sense

```
landmarks  = [[20.0, 20.0], [80.0, 80.0], [20.0, 80.0], [80.0, 20.0]]
world_size = 100.0

class robot:
    def __init__(self):
        self.x = random.random() * world_size
        self.y = random.random() * world_size
        self.orientation = random.random() * 2.0 * pi
        self.forward_noise = 0.0;
        self.turn_noise    = 0.0;
        self.sense_noise   = 0.0;

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

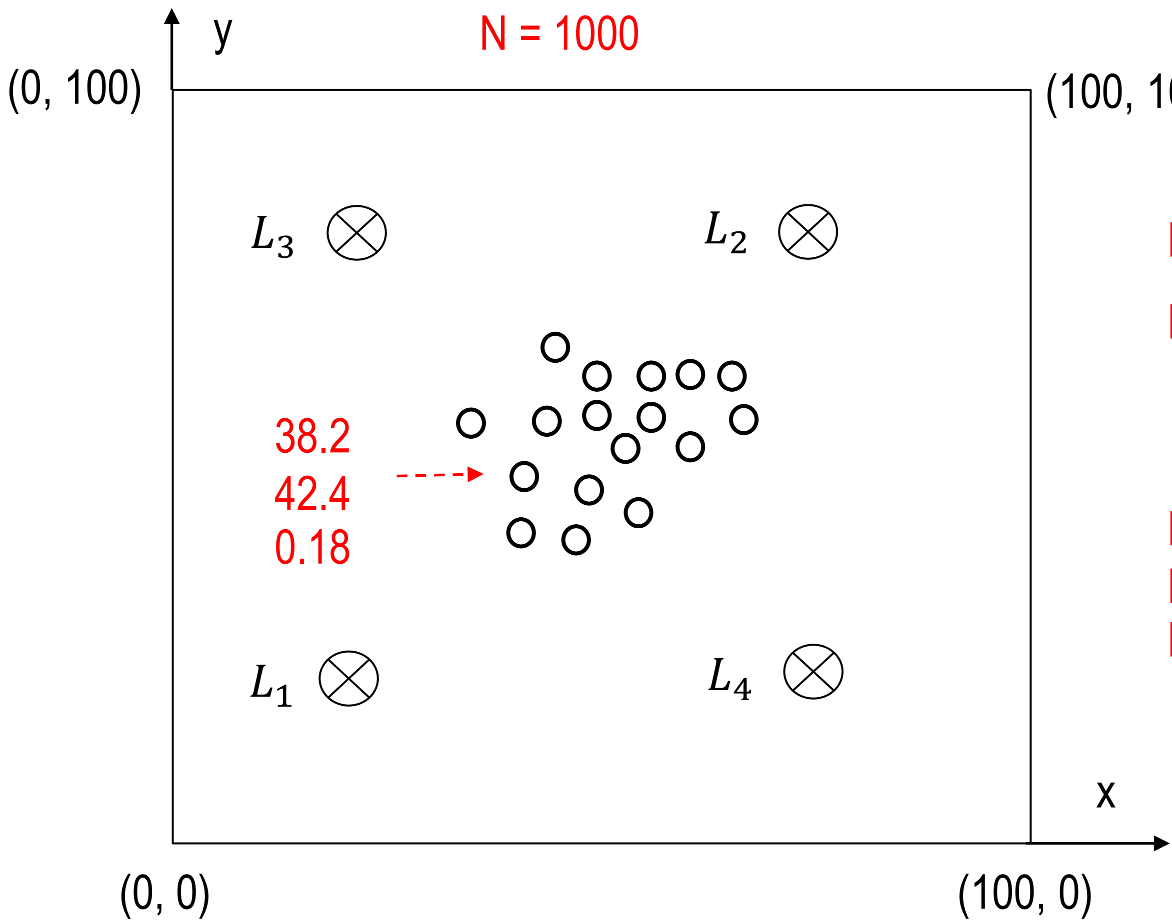
```
myrobot = robot()
myrobot.set(20, 30, pi/2)
```

⇒

Creating a robot at location (20, 30) and facing north.

Creating Particles

- Particles are robots. They are instances of the robot class.



p[i] = robot()

p[i].set(38.2, 42.4, 0.18)

p[i].x	=	38.2
p[i].y	=	42.4
p[i].orientation	=	0.18

Holonomic Robot Move: Turn and Forward

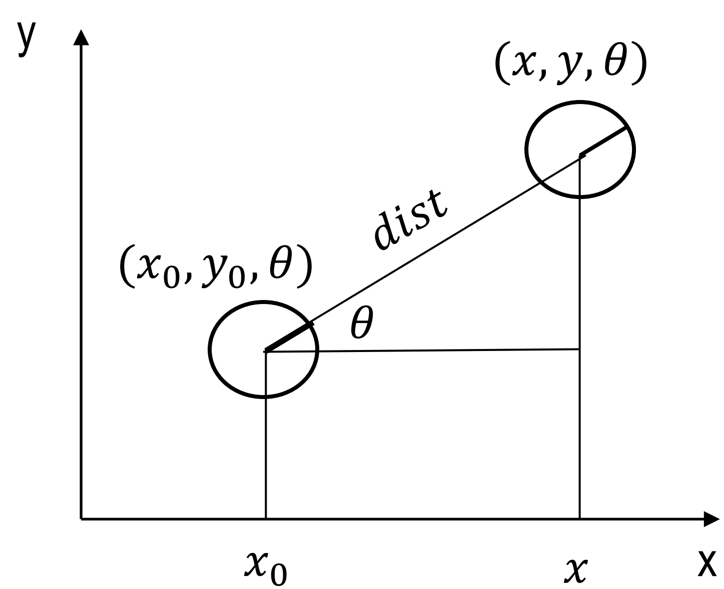
Turn $angle$ in place from $(\boldsymbol{x_0}, \boldsymbol{y_0}, \theta_0)$ to $(\boldsymbol{x_0}, \boldsymbol{y_0}, \boldsymbol{\theta})$

$$\theta = \theta_0 + angle$$

Forward $dist$ from $(\boldsymbol{x_0}, \boldsymbol{y_0}, \boldsymbol{\theta})$ to $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$

$$x = x_0 + dist * \cos(\theta)$$

$$y = y_0 + dist * \sin(\theta)$$



Python Code: Robot Class - Move

```
def move(self, turn, forward):
    if forward < 0:
        raise ValueError('Robot cant move backwards')

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

    # move, and add randomness to the motion command
    dist = float(forward) + random.gauss(0.0, self.forward_noise)
    x = self.x + (cos(orientation) * dist)
    y = self.y + (sin(orientation) * dist)
    x %= world_size    # cyclic truncate
    y %= world_size

    # set particle
    res = robot()
    res.set(x, y, orientation)
    res.set_noise(self.forward_noise, self.turn_noise, self.sense_noise)
    return res
```

myrobot = myrobot.move(-pi/2, 20) ⟹ turns clockwise by pi/2, and moves forward 20 meters

Programming Assignment 5: Moving Robot and Adding Noise

PA3A-MovingRobot.py

Make a robot called myrobot that
Starts at coordinates 30, 50 heading north (pi/2).
Have your robot turn clockwise by pi/2, move 15 m, and sense.
Then have it turn clockwise by pi/2 again, move 10 m, and sense again.
#
Your program should print out the result of
your two sense measurements.

PA3B- AddingNoise.py

Now add noise to your robot before it moves as follows:
forward_noise = 5.0, turn_noise = 0.1, sense_noise = 5.0.

Programming Assignment 6: Creating Particles

PA3C-CreatingParticles.py

Now we want to create particles, p[i] = robot().

In this assignment, write code that will assign 1000 such particles to a list.

#

Your program should print out the length of your list

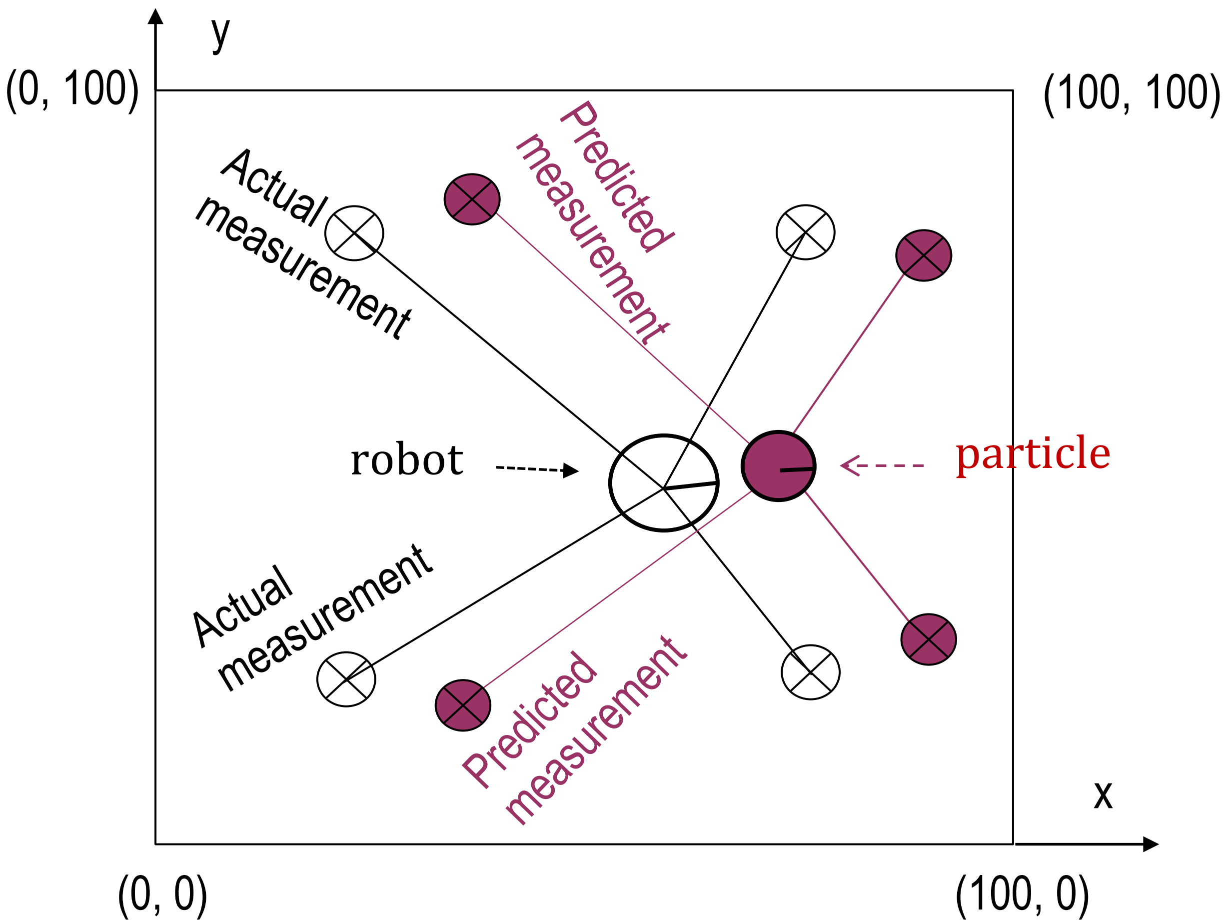
PA3D-RobotParticles.py

Now we want to simulate robot motion with our particles.

Each particle should turn by 0.1 and then move by 5.

Importance Weight

- The closer the predicted measurement of a particle is to the actual measurement of the robot, the more important the particle is.



Python Code: Robot Class - Importance Weight

```
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 = sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
        prob *= self.Gaussian(dist, self.sense_noise, measurement[i])
    return prob
```

p[i].measurement_prob(z))

Here, z is actual measurement

⇒ Importance weight of particle p[i]

Programming Assignment 7: Weight to Particles

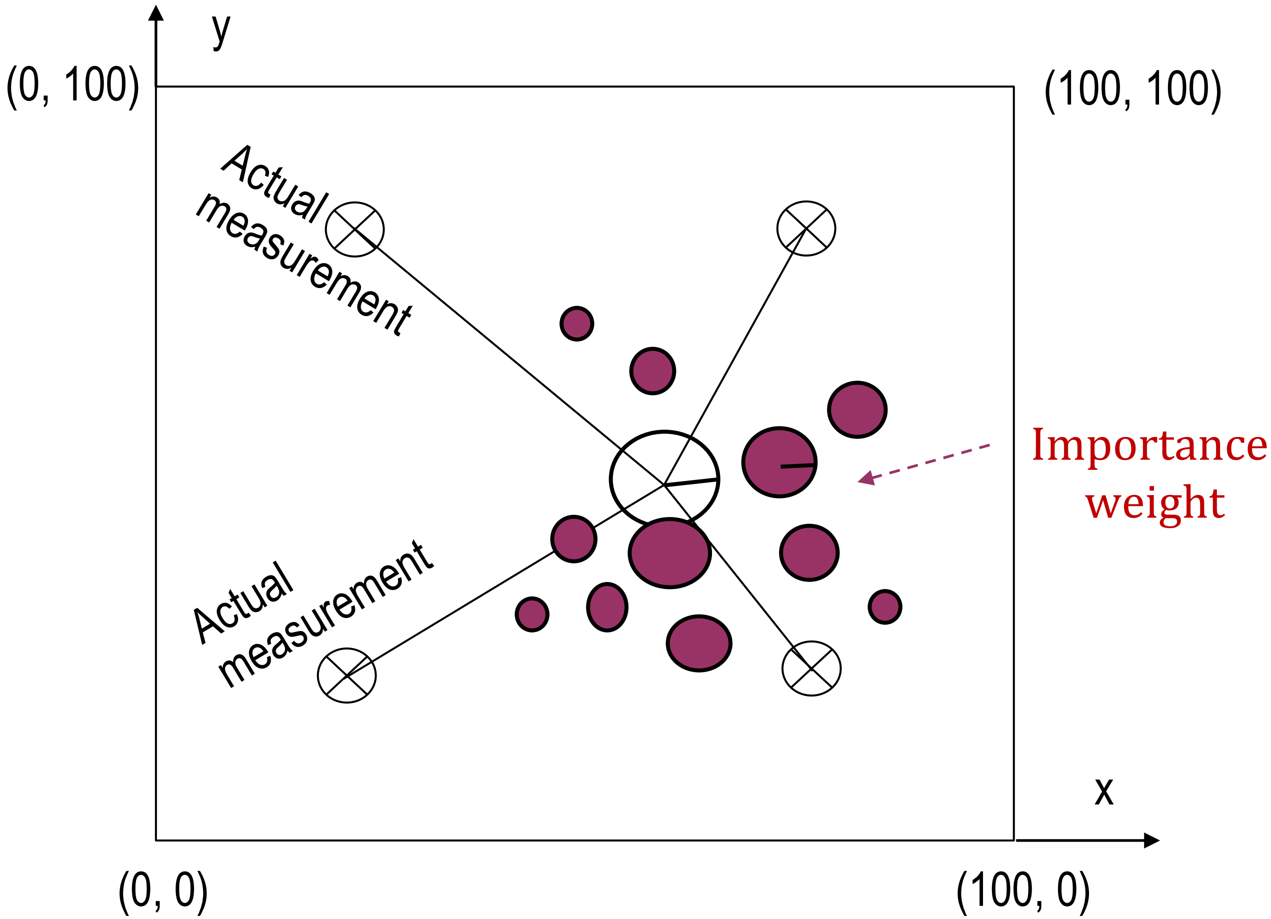
PA3E-ImportanceWeight.py

Now we want to give weights to our particles.

```
####   DON'T MODIFY ANYTHING ABOVE HERE! ENTER CODE BELOW ####
myrobot = robot()
myrobot = myrobot.move(0.1, 5.0)
Z = myrobot.sense()

N = 1000
p = []
for i in range(N):
    x = robot()
    x.set_noise(0.05, 0.05, 5.0)
    p.append(x)

p2 = []
for i in range(N):
    p2.append(p[i].move(0.1, 5.0))
p = p2

w = []
#insert code here to compute weights!
```

Resampling Based on Importance Weight

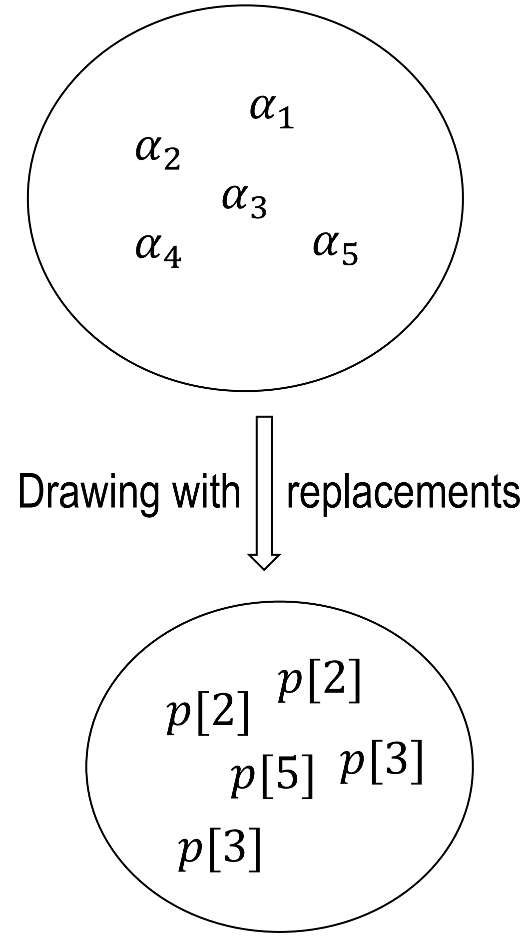
- Drawing particles N times from N existing particles with replacement.



Resampling Based on Importance Weight

- Drawing particles N times from existing N particles with replacement.

Particles	Weights	Normalized weights
p[1]	w_1	$\alpha_1 = {w_1}/{W}$
p[2]	w_2	$\alpha_2 = {w_2}/{W}$
p[N]	w_N	$\alpha_N = {w_N}/{W}$
	$W = \sum_i w_i$	$\sum_i \alpha_i = 1$

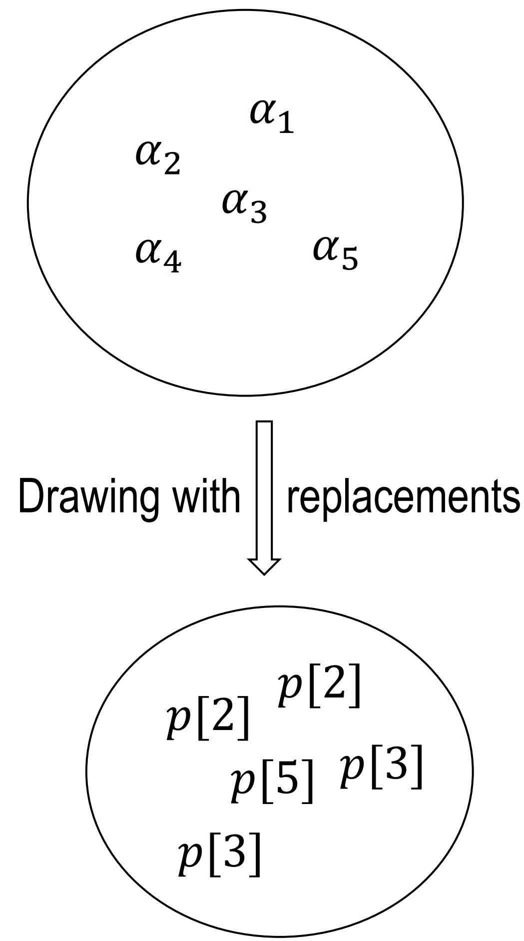


Example (1/2)

- Drawing particles N times from N existing particles with replacement.

N = 5

Particles	Weights	Normalized weights
p[1]	$w_1 = 0.6$	$\alpha_1 =$
p[2]	$w_1 = 1.2$	$\alpha_2 =$
p[3]	$w_1 = 2.4$	$\alpha_3 =$
p[4]	$w_1 = 0.6$	$\alpha_4 =$
p[5]	$w_5 = 1.2$	$\alpha_5 =$



Example (2/2)

- Drawing particles N times from existing N particles with replacement.

N = 5

Particles	Weights	Normalized weights
p[1]	$w_1 = 0.6$	$\alpha_1 = 0.1$
p[2]	$w_2 = 1.2$	$\alpha_2 = 0.2$
p[3]	$w_3 = 2.4$	$\alpha_3 = 0.4$
p[4]	$w_4 = 0.6$	$\alpha_4 = 0.1$
p[5]	$w_5 = 1.2$	$\alpha_5 = 0.2$

Is it possible that p[2] is never sampled?

So what is the probability of NEVER sampling p[2]?

Resampling Algorithm and Example

```
p3 = []
index = int(random.random() * N)
beta = 0.0
mw = max(w)
for i in range(N):
    r = random.random()*2*mw
    beta += r
    while beta > w[index]:
        beta -= w[index]
        index = (index + 1) % N
    p3.append(p[index])
p = p3
```

N = 5
mw = 0.4
index = 2

i	0	1	2	3	4
w	0.10	0.20	0.40	0.10	0.20
r	0.25	0.40	0.20	0.25	0.15
beta	0.25	0.65-0.40 = 0.25 0.25-0.10 = 0.15	0.35-0.20 = 0.15 0.15-0.10 = 0.05	0.30-0.20 = 0.10	0.3
index	2	2->3->4	4->0->1	1->2	2
p3	x2	x4	x1	x2	x2

Resampling Algorithm and Example

```
p3 = []
index = int(random.random() * N)
beta = 0.0
mw = max(w)
for i in range(N):
    r = random.random()*2*mw
    beta += r
    while beta > w[index]:
        beta -= w[index]
        index = (index + 1) % N
    p3.append(p[index])
p = p3
```

N = 10

mw = 0.25

index = 2

i	0	1	2	3	4	5	6	7	8	9
w	0.10	0.15	0.05	0.25	0.15	0.07	0.06	0.04	0.10	0.05
r	0.22	0.03	0.07	0.24	0.10	0.17	0.15	0.04	0.11	0.21
beta	0.22-0.05=0.17	0.20	0.27-0.25=0.02	0.26-0.15-0.07=0.04	0.14-0.06-0.04=0.04	0.21-0.10-0.05=0.06	0.21-0.10=0.11	0.15-0.15=0.00	0.11-0.05=0.06	0.27-0.25=0.02
index	2/3	3	3/4	4/5/6	6/7/8	8/9/0	0/1	1/2	2/3	3/4
p3	x3	x3	x4	x6	x8	x0	x1	x2	x3	x4

Python Code: Evaluate Particles

Average distance of particles { p[i] | i = 0, 1, …, N-1 } to the robot r

```
def eval(r, p):
    sum = 0.0;
    for i in range(len(p)): # calculate mean error
        dx = (p[i].x - r.x + (world_size/2.0)) % world_size - (world_size/2.0)
        dy = (p[i].y - r.y + (world_size/2.0)) % world_size - (world_size/2.0)
        err = sqrt(dx * dx + dy * dy)
        sum += err
    return sum / float(len(p))
```

Programming Assignment 8: Resampling Particles

PA3F-NewParticle.py

Now, you should implement the resampling algorithm shown in the class.

```
p3 = []
index = int(random.random() * N)
beta = 0.0
mw = max(w)
for i in range(N):
    r = random.random()*2*mw
    beta += r
    while beta > w[index]:
        beta -= w[index]
        index = (index + 1) % N
    p3.append(p[index])
p = p3
```

```
print (eval(myrobot, p2)) # Evaluate P2
print (eval(myrobot, p3)) # Evaluate P3
```

⇨

```
38.41411465530672
5.765216996191339
```

Programming Assignment 9: Particle Filter

PA3G-Oreigntation.py

Print 10 evaluation results

```
myrobot = robot()
N = 1000
T = 10 #Leave this as 10 for grading purposes.

p = []
for i in range(N):
    r = robot()
    r.set_noise(0.05, 0.05, 5.0)
    p.append(r)

for t in range(T):
    myrobot = myrobot.move(0.1, 5.0)
    Z = myrobot.sense()

    p2 = []
    for i in range(N):
        p2.append(p[i].move(0.1, 5.0))
    p = p2
```

```
    w = []
    wsum = 0
    for i in range(N):
        wt = p[i].measurement_prob(Z)
        w.append(wt)
        wsum += wt

    for i in range(N):
        w[i] = w[i]/wsum

    p3 = []
    index = int(random.random() * N)
    beta = 0.0
    mw = max(w)
    for i in range(N):
        r = random.random()*2*mw
        beta += r
        while beta > w[index]:
            beta -= w[index]
            index = (index + 1) % N
        p3.append(p[index])
    p = p3
```

```
    #enter code here,
```