# 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 (don't cheat by making an

# arbitrary list of 1000 elements!)

#

# Don't modify the code below. Please enter

# your code at the bottom.

from math import \*

import random

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 set(self, new\_x, new\_y, new\_orientation):

if new\_x < 0 or new\_x >= world\_size:

raise ValueError, 'X coordinate out of bound'

if new\_y < 0 or new\_y >= world\_size:

raise ValueError, 'Y coordinate out of bound'

if new\_orientation < 0 or new\_orientation >= 2 \* pi:

raise ValueError, 'Orientation must be in [0..2pi]'

self.x = float(new\_x)

self.y = float(new\_y)

self.orientation = float(new\_orientation)

def set\_noise(self, new\_f\_noise, new\_t\_noise, new\_s\_noise):

# makes it possible to change the noise parameters

# this is often useful in particle filters

self.forward\_noise = float(new\_f\_noise);

self.turn\_noise = float(new\_t\_noise);

self.sense\_noise = float(new\_s\_noise);

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

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

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

def \_\_repr\_\_(self):

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

#myrobot = robot()

#myrobot.set\_noise(5.0, 0.1, 5.0)

#myrobot.set(30.0, 50.0, pi/2)

#myrobot = myrobot.move(-pi/2, 15.0)

#print myrobot.sense()

#myrobot = myrobot.move(-pi/2, 10.0)

#print myrobot.sense()

#### DON'T MODIFY ANYTHING ABOVE HERE! ENTER CODE BELOW ####

N = 1000

p = []

#enter code here

for i in range(N):

ro\_particle = robot()

p.append(ro\_particle)

p 2= []

for i in range(N):

p2.append(p[i].move(0.1, 5.0))

p = p2

print len(p)

#print p