|  |
| --- |
| # Now add noise to your robot as follows:  # forward\_noise = 5.0, turn\_noise = 0.1,  # sense\_noise = 5.0.  #  # Once again, your robot starts at 30, 50,  # heading north (pi/2), then turns clockwise  # by pi/2, moves 15 meters, senses,  # then turns clockwise by pi/2 again, moves  # 10 m, then senses again.  #  # Your program should print out the result of  # your two sense measurements.  #  # 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))  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))  #### DON'T MODIFY ANYTHING ABOVE HERE! ENTER CODE BELOW ####  myrobot = robot()  # enter code here  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() |
| # Please only modify the indicated area below!  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))  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))  #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/MODIFY CODE BELOW ####  myrobot = robot()  myrobot = myrobot.move(0.1, 5.0)  Z = myrobot.sense()  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 = []  for i in range(N):  w.append(p[i].measurement\_prob(Z))  p3 = []  index = int(random.random() \* N)  beta = 0.0  mw = max(w)  for i in range(N):  beta += random.random() \* 2.0 \* mw  while beta > w[index]:  beta -= w[index]  index = (index + 1) % N  p3.append(p[index])  p = p3  #enter code here, make sure that you output 10 print statements.  print eval(myrobot,p) |