UDACITY

Return to "Computer Vision Nanodegree" in the classroom

Landmark Detection & Tracking (SLAM)

REVIEW

CODE REVIEW 1

HISTORY

robot_class.py 1

```
from math import *
import random
import numpy as np

### ------------------------------------ ###
# Below, is the robot class
#
# This robot lives in 2D, x-y space, and its motion is
# pointed in a random direction, initially.
# It moves in a straight line until it comes close to a wall
# at which point it stops.
#
# For measurements, it  senses the x- and y-distance
# to landmarks. This is different from range and bearing as
# commonly studied in the literature, but this makes it much
# easier to implement the essentials of SLAM without
# cluttered math.
#
class robot:

    # --------
    # init:
    #   creates a robot with the specified parameters and initializes
    #   the location (self.x, self.y) to the center of the world
    #
    def __init__(self, world_size = 100.0, measurement_range = 30.0,
                 motion_noise = 1.0, measurement_noise = 1.0):
        self.measurement_noise = 0.0

        self.world_size = world_size
        self.measurement_range = measurement_range
        self.x = world_size / 2.0
        self.y = world_size / 2.0
```

```
        self.motion_noise = motion_noise
        self.measurement_noise = measurement_noise
        self.landmarks = []
        self.num_landmarks = 0


    # returns a positive, random float
    def rand(self):
        return random.random() * 2.0 - 1.0


    # --------
    # move: attempts to move robot by dx, dy. If outside world
    #       boundary, then the move does nothing and instead returns failur
    #
    def move(self, dx, dy):

        x = self.x + dx + self.rand() * self.motion_noise
        y = self.y + dy + self.rand() * self.motion_noise

        if x < 0.0 or x > self.world_size or y < 0.0 or y > self.world_size
            return False
        else:
            self.x = x
            self.y = y
            return True


    # --------
    # sense: returns x- and y- distances to landmarks within visibility rar
    #        because not all landmarks may be in this range, the list of me
    #        is of variable length. Set measurement_range to -1 if you want
    #        landmarks to be visible at all times
    #

    ## make sure the indentation of the code is correct
    def sense(self):
        ''' This function does not take in any parameters, instead it refer
            (such as self.landamrks) to measure the distance between the ro
            that the robot can see (that are within its measurement range).
            This function returns a list of landmark indices, and the measu
            between the robot's position and said landmarks.
            This function should account for measurement_noise and measurem
            One item in the returned list should be in the form: [landmark_
            '''
        measurements = []
        ## iterate through all of the landmarks in a world
        for i, l in enumerate(self.landmarks):
        ## For each landmark
        ## 1. compute dx and dy, the distances between the robot and the la
            dx = l[0] - self.x
            dy = l[1] - self.y
        ## 2. account for measurement noise by *adding* a noise component t
        ##    - The noise component should be a random value between [-1.0,
        ##    - Feel free to use the function self.rand() to help calculate
            dx += self.rand() * self.measurement_noise
            dy += self.rand() * self.measurement_noise
        ## 3. If either of the distances, dx or dy, fall outside of the int
        ##    then we cannot record them; if they do fall in the range, the
        ##    as list.append([index, dx, dy]), this format is important for
            d = sqrt(dx*dx + dy*dy)
```

SUGGESTION

All though this is not wrong, keep in mind that the distance is measured in L1 norm in the `move` func is a good practice to be consistent throughout the project. Thus, you would need to check for the absc calculating Euclidean distance.

```
            if d > self.measurement_range and self.measurement_range != -1:
                continue
            measurements.append([i, dx, dy])
        ## return the final, complete list of measurements
        return measurements



    # --------
    # make_landmarks:
    # make random landmarks located in the world
    #
    def make_landmarks(self, num_landmarks):
        self.landmarks = []
        for i in range(num_landmarks):
            self.landmarks.append([round(random.random() * self.world_size),
                                   round(random.random() * self.world_size)])
        self.num_landmarks = num_landmarks


    # called when print(robot) is called; prints the robot's location
    def __repr__(self):
        return 'Robot: [x=%.5f y=%.5f]'  % (self.x, self.y)



####### END robot class #######
```

RETURN TO PATH

Rate this review