Meets Specifications

Dear Student,
Congratulations ! You did a great job.

Here are some additional references and questions to look at if you are interested in SLAM beyond this project :

For another interesting tutorial refer this - https://www.youtube.com/watch?v=wVsfCnyt5jA
A tutorial style video by Cyril Stachniss.

Paper on Essential Algorithms for SLAM:
https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf

SLAM for Dummies - MIT:
https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

Q: If we were to translate this project into a real case scenario, would the landmarks be any walls, obstacles we have ? Because the mapping doesn't seem obvious, here we just used the relative position of the robot with the landmarks and its previous direction to localize itself.
A: Yes, in real case scenarios landmarks will be something your robot can detect Maybe some wall or other objects. For example for case of self driving car, the lane markers on road can act as landmarks.

The Github link to the RL agent to localize actively is very interesting do have a look!
If you are more interested please also refer this repo - https://github.com/kanster/awesome-slam

Also If this review helped you in some way, Please consider rating it positively.
I hope you enjoyed the nd so far and wish you the best. Great going !

`robot_class.py`: Implementation of `sense`

Implement the `sense` function to complete the robot class found in the `robot_class.py` file. This implementation should account for a given amount of `measurement_noise` and the `measurement_range` of the robot. This function should return a list of values that reflect the measured distance (dx, dy) between the robot's position and any landmarks it sees. One item in the returned list has the format: `[landmark_index, dx, dy]`.
Great that you checked for measurement_range edge case. I've seen students committing a common mistake to ignore the measurement_range parameter, but you made sure to add measurements that fall within this range using an logical statements. Good.
Some other checks which are not required but can be added for sanity:

```
if((self.x + dx) >= 0 and (self.x + dx) <= self.world_size

 and (self.y + dy) >= 0 and (self.y + dy) <= self.world_size):

# Then only add landmark points
```

It's really good to have them in general cases. But for this project you can be rest assured that data is good and doesn't violate these conditions.
Also you can add landmarks if `measurement_range == -1`. That is something missed in most cases.
Further reading:
Here is a link to the community with different implementations of the algorithm : https://openslam-org.github.io

Notebook 3: Implementation of `initialize_constraints`

Initialize the array `omega` and vector `xi` such that any unknown values are `0` the size of these should vary with the given `world_size`, `num_landmarks`, and time step, `N`, parameters.

Notebook 3: Implementation of `slam`

The values in the constraint matrices should be affected by sensor measurements *and* these updates should account for uncertainty in sensing.
The values of constraint matrices are affected by the measurements and it shows the robot is moving as per the measurements. It's the right implementation.
Further reading:
Concept of sensor fusion : https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see
The values in the constraint matrices should be affected by motion `(dx, dy)` *and* these updates should account for uncertainty in motion.
The values of constraint matrices are affected by the motion and it shows the robot is moving as per the motion. It's the right implementation.
Further Reading:
This is a paper on A General SLAM Framework and Benchmark https://arxiv.org/abs/1902.07995
Hope you enjoy it
The values in `mu` will be the x, y positions of the robot over time and the estimated locations of landmarks in the world. `mu` is calculated with the constraint matrices `omega^(-1)*xi`.
Compare the `slam`-estimated and *true* final pose of the robot; answer why these values might be different.
Based on this answer I can say that your understanding of SLAM is apt. Keep up the good work !
Just to elaborate more the importance of using the confidence -> that a low sigma imply that we have a higher confidence.For this projects' sake and to simply put it - The more the noise the less sure you are of your measurements/motion and hence it makes sense to use 1/noise as the confidence or "strength" with which each update you make.
More noise will affect your predictions adversely. That is the more motion/measurement noise you have the more will be the difference between predicted and real value of final pose. This is because of two reasons :
1) More noise implies less confidence (1/noise) while updating the matrices. In simple words when you are updating the matrices you are less confident because of more noise.
2) More randomness added while moving/sensing [as you implemented in sense function] hence more diversion from the actual values.
There are two provided test_data cases, test your implementation of slam on them and see if the result matches.
The test results are close and make sense. The difference in the results is due to floating point or matrix inverse calculations. Getting your test results right means your code is working well and its always nice to validate your implementation, its a very good coding practice !