

[< Back to Computer Vision Nanodegree](#)

Landmark Detection & Tracking (SLAM)

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

I think you did a great job! I think making the slight change in the `sense()` function that I mentioned earlier is all you need meet specification for everything. Hope this helps you out. All the best with the revision.

`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]
```

I think you have the intuition for how to setup the sense function. I looked at the your notebook 3. Your slam function made good estimations for the test data 1 and 2. The slam-estimations in the "Run Slam" sections were off. I read your answer to the question, I think it was good reasoning. However, I pinpointed the issue of why your values were off.

- In the `sense()` function in the `robot_class.py` file, change

```
landmark_dx=self.x - landmarks[landmark][0]
landmark_dy=self.y - landmarks[landmark][1]
```

to

```
landmark_dx=landmarks[landmark][0] - self.x
landmark_dy=landmarks[landmark][1] - self.y
```

Just swaping what is being subtracted from what.

- Another note, `if landmark_dx<self.measurement_range and landmark_dy<self.measurement_range:` will probably work, but the values can be negative so you might want to use absolute values like this `if abs(landmark_dx)<self.measurement_range and abs(landmark_dy)<self.measurement_range:`.

Looking at the rest of the code, I think if you make these changes everything else should work as is.


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.


 Great! The implementation looks good.

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.

 Well done! The values in the constraint matrices should be affected by sensor measurements and these updates should account for uncertainty in sensing.

The values in the constraint matrices should be affected by motion `(dx, dy)` *and* these updates should account for uncertainty in motion.

 Good job! The values in the constraint matrices should be affected by motion (dx, dy) and these updates should account for uncertainty in motion.

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`.



Perfect! The code looks good.

Compare the `slam`-estimated and *true* final pose of the robot; answer why these values might be different.

It took me a bit of time to understand what is causing the discrepancy in your slam-estimated values and the true final pose of the robot and landmarks. I believe your slam implementation is correct. The estimations, however, are really off. However, the 2 test cases at the end of the notebook gives good results. I think the issue is related to the implementation of the `sense()` function.

Real Landmarks: [[10, 59], [7, 24], [34, 20], [23, 89], [25, 39]]

Your slam-estimated landmarks:

Estimated Landmarks:

[90.226, 35.305]

[91.342, 70.161]

[65.103, 72.701]

[77.675, 5.073]

[74.505, 54.367]

Real Robot Pose: [x=65.58328 y=58.34148]

The slam-estimation of pose:

[69.715, 66.623]

I believe you just have to fix the `sense()` function and this will work. I gave my recommendations earlier in the `sense()` function section.

There are two provided `test_data` cases, test your implementation of slam on them and see if the result matches.



Excellent! There are two provided `test_data` cases. Your slam-estimations are close. This is a good indication that your slam function is correct.

RESUBMIT

[DOWNLOAD PROJECT](#)



Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

[Watch Video](#) (3:01)

RETURN TO PATH

Rate this review