CS 7638: Artificial Intelligence for Robotics

Ice Rover Project

Fall 2019 - Deadline: Monday December 2nd, Midnight AOE

Project Description

The goal of this project is to give you practice implementing a SLAM module and a robot control system that uses it to navigate through a world.

Part A (worth 33%) asks you to build a SLAM system that can keep track of where your robot is located after a series of movements (using measurements to landmark beacons). Complete the SLAM class in the ice_rover.py file. The movements and measurements will have noise.

Your robot will drop via parachute onto the ice sheet. To help it navigate, a set of solar powered beacons have also been dropped onto the ice (they are flat, and the rover can roll over them). These beacons are scattered randomly, but will melt themselves into the ice upon placement, so do not move around.

Your robot should make it's own map of the environment, using it's starting location as the origin (0,0), and taking advantage of the signals from the beacons to maintain a good estimate of its own position relative to the initial (0,0) origin as it slips around on the ice. (The ice may introduce movement noise, in that your robot may not turn or move the exact distances you command.)

In part A, your robot will receive a set of measurements as it follows a pre-scripted set of movements, and your SLAM module will need to calculate and report your robot's position after each movement and measurement (relative to the arbitrary (0,0) staring point). [You do not guide the robot in part A, this part is only to test your SLAM module.]

Part B (worth 67%) asks you to navigate your robot around the environment to take a sample at each of a list of sample sites. Complete the WayPointPlanner class in the ice_rover.py file.

The science team has manually planted a set of special beacons (called "sites") in specific locations which your robot must navigate to and sample.

Note: There is a time penalty for attempting to sample an invalid site. Your robot has a maximum turning angle and distance that it can move each turn. Movement commands that exceed these values will be ignored and cause the robot to not move.

Your robot will not have a map of the environment, but (in part B) it will receive a list of the absolute locations of these sample points. Your robot will have to discover the mapping between it's own "relative" map that places the origin (0,0) at the point where the robot landed and the "absolute" locations where sample sites are located. Hint: Once you sample a site, it will remove itself from the "todo_sample" list, so as soon as you sample your first site, you should have a very good idea of where you are in "absolute" coordinates.

Note that your robot will never have access to the map of where the randomly scattered beacons are located, only a list of the absolute locations of the manually planted sample sites. (Each sample site is marked with a beacon that behaves in exactly the same way as the scattered beacons, but is called a "site" instead of a "beacon" to indicate that it was manually placed and marks a sample site.)

Submitting your Assignment

Your submission will consist of the ice_rover.py file (only) which will be uploaded to Canvas. Do not archive (zip,tar,etc) it. Your code must be valid python version 2.7 code, and you may use external modules such as numpy, scipy, etc that are present in the Udacity Runaway Robot auto-grader. [Try to ensure that your code is backwards compatible with numpy version '1.13.3' and scipy version '0.19.1']

Your python file must execute NO code when imported. We encourage you to keep any testing code in a separate file that you do not submit. Your code should also NOT display a GUI or Visualization when we import or call your function under test. If we have to manually edit your code to comment out your own testing harness or visualization you will receive a -20 point penalty.

Testing Your Code

We have provided testing suites similar to the one we'll be using for grading the project, which you can use to ensure your code is working correctly. These testing suites are NOT complete, and you will need to develop other, more complicated, test cases to fully validate your code. We encourage you to share your test cases (only) with other students on Piazza.

You should ensure that your code consistently succeeds on each of the given test cases as well as on a wide range of other test cases of your own design, as we will only run your code once per graded test case. For each test case, your code must complete execution within the proscribed time limit (5 seconds) or it will receive no credit. Note that the grading machine is relatively low powered, so you may want to set your local time limit to 2.5 seconds to ensure that you don't go past the CPU limit.

We are using the bonnie autograder system which allows you to upload and grade your assignment with a remote / online autograder. See the submit.py file posted as part of the assignment on Canvas for details. You are not required to use this online/autograder feature, but if you do, it will give you more assurance that your code will work correctly with our autograder when we grade the files you submit on Canvas. We **may also choose to use the last grade you receive via the remote autograder as your final grade** at our discretion. (See the "Online Grading" section of the Syllabus.)

Academic Integrity

You must write the code for this project alone. While you may make limited usage of outside resources, keep in mind that you must cite any such resources you use in your work (for example, you should use comments to denote a snippet of code obtained from StackOverflow, lecture videos, etc).

You must not use anybody else's code for this project in your work. We will use code-similarity detection software to identify suspicious code, and we will refer any potential incidents to the Office of Student Integrity for investigation. Moreover, you must not post your work on a publicly accessible repository; this could also result in an Honor Code violation [if another student turns in your code]. (Consider using the GT provided Github repository or a repo such as Bitbucket that doesn't default to public sharing.)

Frequently Asked Questions (F.A.Q.)

Q: Measurements to beacons are relative, how do I find my absolute location? *A:* You have a list of absolute locations for sites. When you successfully sample a site, it will disappear from the list of sites you still need to process. You can use this information to figure out your absolute coordinate (within the sampling size tolerance) once you make a successful sample.

Q: How do I tell if a measurement is from a new landmark or one I have seen previously? *A:* Each landmark has a unique ID (hash value) which you can use to identify landmarks that have been seen before.

Q: How do I handle landmark beacons that are outside the horizon distance (and not returning measurements)? *A:* Do not add beacons to your matrix until you see them, and if you stop receiving measurements from them, stop updating the appropriate spots in your matrix until they come back into view.

Q: Is there a minimum possible value for the horizon_distance parameter? *A*: You will always be able to see beacons from your initial position, and since beacons are a minimum of 1 unit apart, you can assume that the horizon_distance will never be smaller than 1.0.

Q: What are the (x,y) return values from process_measurement() and process_movement() in part A relative to? And how are they different? *A:* Both of them return your best guess for the position of the robot relative to the starting location (0,0). The difference between them is that one of them gives an estimation of the robot's position after a measurement, and the other after the robot has moved.

Q: What should I do if my robot can't see any sites from its initial starting position (outside the horizon distance)? *A*: You'll need to search for the signal from your first site by moving the robot in a search pattern, such as a slowly expanding spiral.

Q: Which way is the robot facing when it lands? *A*: Although slightly unrealistic, your robot will always have a bearing of zero degrees when it lands. (You are welcome.)