Build a 2D 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. The starting location of the robot will be random. To help it navigate, a set of beacons have also been randomly placed in the environment. The beacons are flat, so the robot can roll over them. In addition, the location of the beacons will remain fixed in each session. The robot should make its own map of the environment, using its **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 moves around. Please note the movement of the robot is noisy (i.e., the robot may not turn or move the exact distances you command).

In part A (worth 33%), the 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 the 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. Special beacons (called "sites") are determined 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. The 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. The robot will have to discover the mapping between its 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 the 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.)

Your code should NOT display a GUI or visualization when I import or call your function under test. Please create a separate ".py" file for visualization.

You are provided with testing suites similar to the one that will be used to test your submission. These testing suites **are NOT complete**, and you will need to develop other, more complicated, test cases to fully validate your code. 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. For each test case, your code must complete execution within the proscribed time limit (5 seconds).

Frequently Asked Questions

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 the 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 the robot cannot see any sites from its initial starting position (outside the horizon distance)?
A: You will 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 starts?
A: Although slightly unrealistic, the robot will always have a bearing of zero degrees at start.