CS 7638: Artificial Intelligence for Robotics

Gem Finder Project

Summer 2021 - Deadline: Monday July 26th, 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 40%) 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 landmarks). Complete the SLAM class in the *gem_finder.py* file. The movements and measurements will have noise.

Your robot will be dropped into an unknown territory by a team seeking to extract special gems in the area. The robot has special sensors that are able to detect the gems in the region. The measurements from these sensors can be used to help the robot navigate. The sensors will provide the bearing and distance to the gem (relative to the robot's location). Both of these measurements contain noise. The amount of noise will be proportional to the distance away from the robot. For example a close gem will have a small amount of noise, while a far away gem will have a larger amount of noise. Since this noise value is not provided, you will need to come up with a way to estimate the noise, while maintaining this proportional constraint.

Your robot should make a map of the environment, using it's starting location as the origin (0,0) by taking advantage of the measurements to the gems to maintain a good estimate of its own position relative to the initial (0,0) drop-off location.

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 measurement and movement (relative to the arbitrary (0,0) staring point). [You do not guide the robot in part A and do not extract any gems, this part is only to test your SLAM module.]

Part B (worth 60%) asks you to navigate your robot around the environment to extract gems provided to you in a list. Complete the GemExtractorPlanner class in the *gem_finder.py* file.

Your commander will provide you with a list of needed gems. You will need to collect one of each kind of gem in the list. You can collect them in any order, you don't have to follow the order provided in the list. You do not need to extract all gems in the environment, only the ones in the provided list.

There is a time penalty for extracting dirt (failing to extract a gem). This case happens when you attempt to extract in a location that does not contain a gem of the type you specified (within a minimum extraction distance). Your robot must be within this pre-defined distance (0.15) of the gem in order to extract it successfully. As soon as the extraction is successful, the gem will be removed from the environment. When a gem is extracted from the terrain, it no longer exists in the terrain, and thus won't return a measurement. You must specify the type of gem you are intending to extract when you attempt to extract. Only this kind of gem will be extracted if it is within the defined radius, if not there will be a time penalty.

Your robot has a maximum turning angle [pi/2] and distance [passed using a parameter] that it can move each turn. Movement commands that exceed these values will be ignored and cause the robot to not move.

Note that your robot will never have access to the map of where the randomly scattered gems are located, only a list of the gems needed.

Submitting your Assignment

Your submission will consist of the *gem_finder.py* file (only) which will be uploaded to Gradescope. Do not archive (zip,tar,etc) it. Your code must be valid python version 3.6 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']

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.

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.

A visualization file has been provided to aid in debugging. The visualization will plot 5 pieces of data: real location of robot, estimated location of robot, real location of gems, estimated location of gems, type of gem ('A', 'B', . . . etc) present in environment. The real location of the robot will be a large red dot. The estimated location of the robot will be a small blue dot. The real location of a gem will be a large grey square. The estimated location of a gem will be a small black dot. The type of gem will be next to the real location. When the robot extracts a gem from the environment, the letter will no longer exist next to the real location (to indicate it has been removed).

The estimated points to plot need to be returned from `next_move` as a 2nd (optional) value in the form of a dictionary. The keys will be the landmark id and the values will be x,y coordinates. The key representing the robot's estimated location will be 'self'. `{'self': (.2, 1.5), '123': (.4,1.9)}`

Usage: `python visualize.py [-h] [--part {A,B}] [--case {1,2,3,4,5,6,7,8,9}]`

Example to run the visualization: `python visualize.py --part B --case 5`

We are using the Gradescope autograder system which allows you to upload and grade your assignment with a remote / online autograder. 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: How can I uniquely identify gems in the environment? *A:* Each gem will have a unique id. Although there may be more than one of the same type of gem in the area, each will have a different unique id.

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: 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.)