CS 7638: Artificial Intelligence for Robotics

Indiana Drones Project

Spring 2022 - Deadline: Monday April 25th, Midnight AOE

Project Description

Welcome to the Indiana Drones team!

In this project, you were chosen to part of a team of treasure hunters who are seeking to extract an invaluable piece of ancient treasure. Unfortunately for us, the treasure is located in a dense and dangerous jungle - making a typical safari impossible. That's where you come in. As a robotics expert, your mission should you choose to accept it, is to program a drone to map the jungle environment (using SLAM) and navigate it through the jungle to extract the treasure while avoiding tree crashes.

The drone has special sensors that are able to detect the trees relative locations and size. The sensors will provide the bearing in radians (i.e. angle) and distance to the trees' center in meters (relative to the drone's location and orientation) as well as the radius of the tree in meters. You will also be able to move the drone by directly specifying the next location (i.e. by specifying distance in meters and steering in radians relative to the drone's previous location and orientation).

Part A (worth 60%) asks you to complete the SLAM class in the *indiana_drones.py* file. Your mission is to implement a SLAM module that calculates and reports your drone's position after each measurement and movement (relative to the arbitrary (0,0) staring point). All you need to do is fill in the appropriate member functions. The testing suite will call those functions at appropriate times to process the measurements (using `process_measurements`) and movement (using `process_movement`) and grade your SLAM systems accuracy in estimating the drone's location (using `get_coordinates`).

This part is only to test your SLAM module. You do not guide the drone in part A and do not extract any treasure. A series of pre-scripted movements are performed and those movements will be provided to you one by one. For each test case, you will recieve 50% credit if your SLAM system accurately estimates the position of your drone within a distance threshold and 50% of the credit if each of the landmark (tree) locations are within a distance threshold. Points are deducted for each innacuarcy.

Note :

1. You will only see trees that are within your drone's measurement sensor's horizon (in meters). So previously unseen trees may appear in your measurements as you move through the environment and get closer to new trees.
2. Both measurements and movements contain noise. Since this noise value is not provided, you will need to come up with a way to estimate the noise
3. Your drone will never have access to the map of where the randomly scattered trees are located.

Part B (worth 40%) asks you to complete the IndianaDronesPlanner class in the *indiana_drones.py* file. Your commander will provide you with the location of the treasure. Your goal in this part of the project is to use your newly developed SLAM module in conjuction with a navigation algorithm (that you need to code) to navigate the drone to the treasure's location while avoiding trees in its path and extracting the treasure when it is reached.

The output of your navigation algorithm will be one of two actions in the next_move function: namely `extract` and `move`. The `extract` action extracts the treasure at your location if it exists. The `move` action moves the drone by the distance and steering you prescribe.

When you issue your `extract` action you should supply 3 arguments total, including the treasure type (*) and current estimated location (x, y) of the drone as follows: `extract * 1.5 -2.1` [command treasure_type x y]. The treasure will only be extracted if it is within the defined radius (0.25 meters), if not there will be a time penalty for extracting dirt.

You should specify the movement as follows: `move 1 1.57`. [command distance steering] which means the drone will turn counterclock-wise 90 degrees [1.57 radians] first and then move a distance of 1 meter.

For each test case, you will receive 50% of the credit by extracting the treasure within the time limit, and 50% of the credit by avoiding any crash with the trees. However, if the drone doesn't extract the treasure within the time limit, there will be no credit given. For each crash with one tree (multiple crashes with the same tree will be counted as one crash for grading purpose), there will be a 25% credit deduction. For example, if the drone extracted the treasure within the time limit but crashed into one tree and one tree only, you will receive 75% of the credit.

Note :

1. In this project we assume the drone is a point (even though in the visualization it occupies some area).
2. There is penalty for each tree the drone crashes into (whenever the drone enters within the radius of the tree's center/the canopy of a tree). Remember that the drone moves on a path, so even if the starting and ending points of your movement aren't inside the tree's radius, the path could intersect it, which would result in a penalty.
3. Your drone has a maximum turning angle [in radians] and a maximum distance [in meters] that it can move each timestep [both passed using a parameter]. Movement commands that exceed these values will be ignored and cause the drone to not move.

Submitting your Assignment

Your submission will consist of the *indiana_drones.py* file (only) which will be uploaded to Gradescope. Do not archive (zip,tar,etc) it. Your code must be valid python code, and you may use external modules such as numpy, scipy, etc.

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 help 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. For each test case, your code must complete execution within the prescribed time limit (10 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 5 seconds to ensure that you don't go past the CPU limit. Note that if VERBOSE is on in the *testing_suite_indiana_drones.py* file, printing will take a lot of time and slow down your execution. So please feel free to increase the time limit while debugging with the VERBOSE on, but when you submit your code, it should run within the 10 second time limit on Gradescope.

Please note that the *testing_suite_indiana_drones.py* will run all cases from both part A and part B 10 times, remove the lowest score and average the rest to calculate your score. It will do this automatically (i.e. you do not need to loop your code).

A visualization file has been provided to aid in debugging. The visualization will plot 6 pieces of data: the real location of drone, the estimated location of drone, the real location of trees, the estimated location of trees, the types of the trees ('A', 'B', . . . etc) and the location of treasure present in environment. The real location of the drone will be a drone with 4 rotors. The estimated location of the drone will be a small blue dot. The real location of trees will be represented by green circles of varying radii. The estimated location of a tree will be a small black dot. The type of tree/treasure will be next to the real location. The treasure is represented by a red triangle.

The estimated points to plot need to be returned from `next_move` as a 2nd (optional) value in the form of a dictionary. This is needed to show your SLAM system's estimates of drone and landmark location in the visualization. The keys should be the landmark id and the values should be its x,y coordinates. The key representing the drone's estimated location will be 'self'. `{'self': (.2, 1.5), landmark id 1: (.4,1.9)}`

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

Example to run the visualization: `python visualize.py --part B --case 3`

The *visualize.py* also has a VERBOSE FLAG, similar to the testing suite.

We are using the Gradescope autograder system which allows you to upload and grade your assignment with a remote / online autograder. You must submit your *indiana_drones.py* file to Gradescope to receive credit.

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 trees in the environment? *A:* Each tree will have a unique id. Although there may be more than one of the same type of tree 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 drone relative to its start location (0,0). process_measurement() gives an estimation of the drone's position after a measurement, and process_movement() provides the drone's position estimate after the drone has moved.

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

Q: I'm confused. We are given so many files. What exactly should we do again and in which file? *A*: The main file you are concerned with is *indiana_drones.py.* This is what you fill and submit to gradescope. It contains two classes (SLAM and IndianaDronesPlanner) whose methods are used by the testing suite to run SLAM and Obstacle Avoidance respectively in various test cases to generate your score. *drone.py* and *matrix.py* contain helper classes and methods that you are free to use in your implementation. *visualize.py* is provided to help you debug your code with a visualization.

Things to think about

1. GraphSLAM estimates the X and Y locations of the drone, but the orientation accumulates noise too. Do you need to handle that? If so, think about how to handle that.
2. The noise in movements and measurements are for the distance and angle to a point. What does this mean for the variance of the noise in the X and Y position? Is it constant? If not, how does it scale? Do you need to handle that? If so, think about how to handle that.
3. How can the drone detect a potential crash of its path with a tree? And when it detects a potential crash with a tree, how can the drone avoid the crash? Could it figure out what options it has and choose one way it could go? Or does it need to find an exact path to avoid the crash?