RBE 550 - Standard Search Algorithms Implementation

Overview

In this assignment, you are going to implement **PRM**, **RRT** and **RRT*** algorithms. For **PRM**, you are required to implement 4 different sampling methods - **uniform sampling**, **random sampling**, **gaussian sampling** and **bridge sampling**. These three algorithms are the basic ones for sampling-based planning. This template is provided to you as a start point. After you finish coding, you would be able to run these algorithms to find a path in a map, and visualize the result.

Files included:

PRM.py is the file where you will implement a PRM class with four different sampling methods.

RRT.py is the file where you will implement a RRT class for RRT and RRT*.

main.py is the scrip that provides helper functions that load the map from an image and call the classes and functions from **PRM.py** and **RRT.py**. You are not required to modify anything but you are encouraged to understand the code.

WPI_map.jpg is a binary WPI map image with school buildings. You could replace it with some other maps you prefer.

Instruction

Before starting any coding, please run the code first:

```
python main.py
```

The **main.py** loads the map image **WPI_map.jpg** and calls classes and functions to run planning tasks. As you haven't written anything yet, there should be no path shown in the graph, but only the map image, start point and end point.

Please keep in mind that, the coordinate system used here is **[row, col]**, which is different from [x, y] in Cartesian coordinates. In README and the code comment, when the word '**point**' is used, it refers to a simple list [row, col]. When the word '**node**' or '**vertex**' is used, it refers to either the Node class in RRT ,or a node/vertex in a graph in PRM.

PRM

The two main phases of PRM are **Learning Phase** and **Query Phase**.

Learning Phase

You would code **Learning Phase** in the function `sample`, where it samples points in the map according to different strategy, and connect these points to build a graph. In this template, the graph library Networkx is used to store the result graph.

There are four different sampling methods to be implemented - `uniform_sample`, `random_sample`, `gaussian_sample` and `bridge_sample`. Please refer to the lectures and make sure you understand the ideas behind these sampling methods before coding.

After sampling, you would need to connect these sampling points to theirs k nearest neighbors. To find their neighbors, you should NOT just use brutal force algorithm (This will take way too long), but use K-D tree as mentioned in the class. Here is an example of how to use scipy K-D tree structure.

Finally, you will need to use all the sampled points and their connection with neighbors as nodes and edges to build a Networkx graph.

Query Phase

You would code **Query Phase** in the function `search`, where it search for a path in the constructed graph given a start and goal point.

As start and goal points are not connected to the graph, you will first need to add the start and goal node, find their nearest neighbors in the graph and connect them to these two nodes. Practically, as some of the graphs don't have a good connectivity, we will not only connect the start and goal node to their nearest node, but all the nodes within a certain distance, in order to increase the chance of finding a path.

Having connected start and goal node in the graph, we could use Dijkstra algorithm or any other algorithms we learn before to search for a valid path. This part is similar to the first assignment, so is already done by using the Dijkstra function Networkx provided.

Finally, as PRM is a multi-query planning algorithms, one could call `search` with other start and goal point. So the previous start and goal nodes and their edges need to be removed in the end of each query phase. This part is also implemented already.

Read the description of the functions for more details before implementing.

RRT

For simplicity, this template uses a class 'Node' and a list 'vertices' in class 'RRT' as a tree structure. If you prefer to use other tree structure, please feel free to do so.

You would code RRT in the function `RRT`. In each step, get a new point, get its nearest node, extend the node and check collision to decide whether to add or drop this node. When you add a new node to the tree, remember to set the cost and parent of the new node, and add the new node to the list 'vertices'. You will also need to check if it reaches the neighbor region of the goal. If so, connect to the goal directly and set the found flag to be true.

You would code RRT* in the function `RRT_star`. The first few steps are pretty much the same as RRT. Besides, when a new node is added, you will need to rewire the new node AND all its neighbor nodes. Even a path is found, the algorithm should not stop as adding new nodes will possibly optimize the current found path.

Read the description of the functions for more details before implementing.

Until now, I hope you have a basic understanding of the template code and what to do next.

This template is only provided as a start point, feel free to make any modification of the codes or code structures if needed. After you finish coding, your algorithms should produce similar results as the images in **demo** folder.

Rubrics

- (3 pts) Your PRM is implemented correctly
    - Four sampling methods produce the correct sampling points
    - Connect the sampling points, start and goal into a graph using a proper method
    - Given start and goal, find a path if feasible.

- (3 pts) Your RRT and RRT* are implemented correctly
    - Get proper new nodes in each step
    - Connect and rewire (RRT*) new nodes
    - Find a path if feasible

- (1 pts) Documentation

  Besides the code, you should also include a documentation with the following content:

    - Briefly answer the following questions
        - For PRM, what are the advantages and disadvantages of the four sampling methods in comparison to each other?
        - For RRT, what is the main difference between RRT and RRT*? What change does it make in terms of the efficiency of the algorithms and optimality of the search result?
        - Comparing between PRM and RRT, what are the advantages and disadvantages?
    - Algorithm results and explanation

      Run your code with `python main.py` and **save your results as png images**. Briefly explain why your algorithms would produce these results. Why RRT and RRT* result in different trees? How different sampling methods lead to different sample sets in the graph?
    - Reference paper and resources if any

  Include the documentation as a pdf file, or if you prefer, a md file.