COMP 7308: Introduction to unmanned systems Fall 2023

Assignment 4: RRT and RRT* Algorithm

:

1 Introduction

In this project, you will start to implement the RRT and RRT* algorithm. The code for this project includes the following files, available on the Moodle.

Files you will edit and submit:

`/src/rrt_robot.py`: The location where you will fill in portions during the assignment. When you finish, you should submit **ONLY** this file without modifying its name.

Files you should NOT edit:

`/src/rrt_robot.yaml`: Information of the simulation environment.

`/hint/`: Some reference results you can utilise to compare with yours.

`/test/student_grader.py`: Project autograder of student version. Pass it can make sure that you get **most of** the grades.

Evaluation

Your code will be autograded for technical correctness. Please **DO NOT** change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

Academic Dishonesty

We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. Instead, contact the TA if you are having trouble.

Prerequisite

To install the simulation package, please check Assignment 0. You should be familiar with `numpy` and we name it `np` in our file.

2 Question 1: RRT algorithm (30 points)

First we will implement the Rapidly-exploring Random Tree (RRT) algorithm shown in Algorithm 1.

Algorithm 1: RRT (*Sampling-based Algorithms for Optimal Motion Planning, Karaman and Frazzoli*)

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3 $\quad x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4 $\quad x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5 $\quad x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
6 $\quad$ **if** $\texttt{CollisionFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7 $\quad\quad V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
8 **return** $G = (V, E)$;

The following is a description of the main loop of the RRT algorithm outlined in Figure 1:

Line 3: Generate a new node x_{rand} that does not collide with any obstacles.

Line 4: Determine the node x_{nearest} within the RRT tree that is closest to x_{rand}.

Line 5: Move a fixed step in the direction of $\overrightarrow{x_{\text{nearest}} x_{\text{rand}}}$ and obtain x_{new}.

Line 6-7: Verify if the new connection line $(x_{\text{nearest}}, x_{\text{new}})$ collides with any obstacles. If it does not, then add the node x_{new} and line $(x_{\text{nearest}}, x_{\text{new}})$ to the RRT tree for expansion.

The loop terminates when the new node x_{new} reaches either our goal state or the maximum iteration time n.

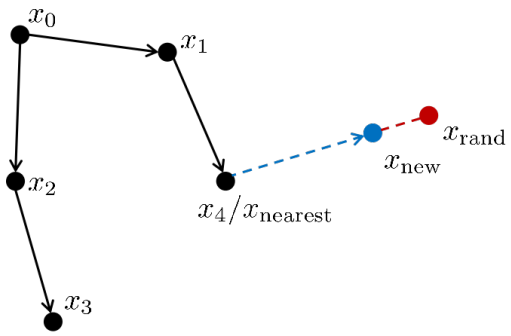

Figure 1: An illustration of RRT algorithm. x_0 is the start/root node.

In this section, you should:

(a) Finish the function `rrt_nearest()` following line 4.

(b) Finish the function `rrt_stear()` following line 5.

(c) Finish part of the function `rrt()` within the 'rrt' conditional statement following line 7.

For line 3 and 6, we have implemented them in our code, corresponding to two functions called `rrt_sample_free()` and `rrt_collision_free()`. You may utilise them later.

To test your implementation, run the autograder: `python /test/student_grader.py -q q1`. You can also visualise your trajectory result by running `python /test/student_grader.py -q q1 --graphics`, and compare it with the result `./hint/q1.gif` and `./hint/q1.png`.

Note:

1. In our implementation, the RRT tree is organized as a `list` to store all nodes. Each `node` is also represented as a `list` that contains its state x and the index of its parent node, which is the connected node with the closest path to the start/root node.

   For example, in Figure 1, the node with state x_{new} is saved as $[x_{\text{new}}, 4]$, where 4 is the index of the parent node $[x_4, 1]$ in the tree. Here, the path from x_0 to x_4 is shorter than the path from x_0 to x_{new}. You can retrieve node 4 by calling `nodes[4]` in the pseudocode.
2. You can use `for index, node in enumerate{nodes}` (pseudocode) to iterate each node with its index.

3 Question 2: RRT* algorithm

One of the concern of the RRT algorithm is that even though node x_{nearest} is the closest node to node x_{rand} within the RRT tree, it does not mean that node x_{nearest} is the closest node to node x_{new}. Therefore, it is likely that there exists better solution of the trajectory.

To improve on this, the RRT* algorithm introduces the concept of cost for each node, defined as the continuous path length from the start node to the target node. For instance, in Figure 2, the cost of node x_3 is the sum of length (x_0, x_2) and (x_2, x_3), denoted as $\texttt{Cost}(x_3) = \texttt{Cost}(x_2) + c(\texttt{Line}(x_2, x_3)) = c(\texttt{Line}(x_0, x_2)) + c(\texttt{Line}(x_2, x_3))$, where $c(x, y) = \|x - y\|_2$.

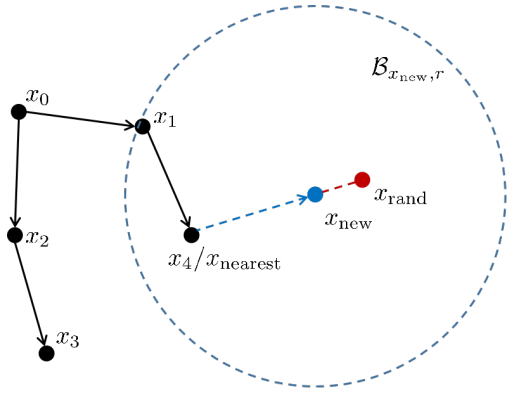

Figure 2: An illustration of RRT algorithm

Algorithm 2 reveals all details. The difference between RRT and RRT* is that RRT* considers the cost of the path. If line $(x_{\text{nearest}}, x_{\text{new}})$ does not collide with obstacles, RRT* does the following:

Line 7: Find all nodes in the RRT* tree whose distance to x_{new} is within r: $X_{\text{near}} = \texttt{Near}(G = (V, E), x_{\text{new}}, r) = \{x \in G | x \in \mathcal{B}_{x_{\text{new}},r}\} = \{x \in G | \|x - x_{\text{new}}\|_2 < r\}$, where $r = \min\{\gamma_{\text{RRT}^*} \cdot \left(\frac{\log(\text{card}(V))}{\text{card}(V)}\right)^{1/d}, \eta\}$ in base e, $\text{card}(V)$ means the number of nodes in the RRT* tree, and $\gamma_{\text{RRT}^*}, \eta$ are two constants.

Line 9-12: Iterate nodes within X_{near} to find a shortest-cost path for x_{new} with collision check. Assign the target to node $x_{\min}$ and cost $c_{\min}$ to be the parent node for x_{new}.

Line 8, 13: Add the node x_{new} and the corresponding line $(x_{\min}, x_{\text{new}})$ into our RRT* tree for expansion.

Algorithm 2: RRT* (*Sampling-based Algorithms for Optimal Motion Planning, Karaman and Frazzoli*)

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3 $\quad x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4 $\quad x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5 $\quad x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
6 $\quad$ **if** $\texttt{CollisionFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7 $\quad\quad X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*} \cdot \left(\frac{\log(\text{card}(V))}{\text{card}(V)}\right)^{1/d}, \eta\})$;
8 $\quad\quad V \leftarrow V \cup \{x_{\text{new}}\}$;
9 $\quad\quad x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
10 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
11 $\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
12 $\quad\quad\quad\quad x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
13 $\quad\quad E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
14 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
15 $\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then**
$\quad\quad\quad\quad x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
16 $\quad\quad\quad E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
17 **return** $G = (V, E)$;

Line 14-16: Iterate nodes within X_{near}, check whether the cost of each node x_{near} is reduced and collision does not exist if it connects to x_{new}. If so, update the connection relationship of x_{near} by changing its parent node x_{parent} to x_{new} within RRT* tree.

In this section, you should:

(a) Finish the function `rrt_near()` following line 7.

(b) Finish part of the function `rrt()` within the 'rrt_star' conditional statement following line 7-16.

For `cost`, we have implemented it in our code called `rrt_cost()`. You may utilise its and other functions we provide above.

To test your implementation, run the autograder: `python /test/student_grader.py -q q2`. You can also visualise your trajectory result by running `python /test/student_grader.py -q q2 --graphics`, and compare it with the result `./hint/q2.gif` and `./hint/q2.png`.

Note:

1. In our code, the data structure is slightly different than the pseudocode. The RRT* tree is organised as a `list` to save all nodes. Each `node` is also a `list` containing its state x, the index of its parent node, and its cost. The parent node means the connected node with a closest path to the start/root node.

   As an instance in Figure 2, for node with state x_3, we will save it as $[x_3,\ 2,\ c(\texttt{Line}(x_0, x_2)) + c(\texttt{Line}(x_2, x_3))]$, where 2 means the index of the parent node $[x_2,\ 0,\ c(\texttt{Line}(x_0, x_2))]$ in our tree. Here $\texttt{Cost}(x_2) < \texttt{Cost}(x_3)$. You can find node 2 by calling `nodes[2]` (pseudocode).

2. You can use `for index, node in enumerate{nodes}` (pseudocode) to iterate each node with its index.