Path planning using RRT

David FILLIAT - ENSTA Paris

12 décembre 2022

1 Introduction

In this practical work, we will work on the Rapidly Exploring Random Trees (RRT) algorithm [1]. For this, we will use the python code available on the course Moodle that implements RRT and one of its variant RRT* [2], on different environments. This code is modified from the code of the repository of Huiming Zhou [1] that implements and illustrates many path planing algorithms.

Upload your report as a pdf file that includes your answers to the questions and the code you wrote on the Moodle.

2 RRT vs RRT*

In this part, use the default starting and goal position provided in the code and the default environment (`environment = env.Env()` in the main function).

Question 1 : Test the two algorithms `RRT` and `RRT*` on this problem by varying the maximum number of iterations. What can you see on the average lengths of the paths ? On the computation times ? Remember to disable the display and to make several experiments to have significant results as these algorithms are stochastic.

Question 2 : Change the `step_len` parameter (default value is 2 in the provided code). What are the consequences of small values and large values on the two algorithms ?

3 Planification in narrow corridors

In this part, use the default starting and goal position provided in the code, the Env2 environment (`environment = env.Env2()`) and the RRT algorithm with the initial parameters (`rrt = Rrt(environment, x_start, x_goal, 2, 0.10, 1500))` . You will see that the narrow corridor in the middle of the map makes it difficult to find a path.

Question 3 : Explain why it is difficult to grow the tree rapidly in this environment (in particular think about what happens when the tree tries to grow towards a random point from the nearest node).

Question 4 : To improve this, modify the `rrt.py` file to implement a simple variant of the `OBRRT` [3] algorithm. In this algorithm, the idea is to sample points taking into account the obstacles in order to increase the chances that the tree passes through difficult areas.

Implement a very simple version in which you will sample a part of the points randomly in the obstacle free area around the corners of the obstacles. To do this, you must modify the function `generate_random_node(self, goal_sample_rate)`. You will need to use the following variables and functions :

- `self.env.obs_rectangle` : a list of tuples (x, y, w, h) describing the obstacles : x, y are the coordinates of the bottom left corners of the obstacles, w, h are the width and height of the obstacle
- `self.utils.is_inside_obs(node)` : a function that checks if a node is in the obstacle free area
- `np.random.randint(n)`, `np.random.random()` and `np.random.randn()` : functions giving a random integer, random value between 0 and 1 with uniform probability and a random value following a unit gaussian.

Show the performance variation as a function of the percentage of points sampled using this strategy (from 0% to 100%).

[1] https://github.com/zhm-real/PathPlanning

Références

[1] Steven M. Lavalle, James J. Kuffner, and Jr. Rapidly-Exploring Random Trees : Progress and Prospects. In *Algorithmic and Computational Robotics : New Directions*, pages 293–308, 2000.

[2] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7) :846–894, June 2011.

[3] S. Rodriguez, Xinyu Tang, Jyh-Ming Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 895–900, May 2006.