Autonomous Robotics
Lab Report 2 - Sampling-based algorithms (RRT)

Anirudh Puligandla

March 19, 2017

1 Introduction

The goal of this lab work is to program and test the Rapidly-Exploring Random Trees (RRT) algorithm, that is widely used for path planning in robotics. Using the RRT algorithm, that is based on smapling-based algorithms, a 2D path planning problem was to be solved. The algorithm is programmed on MATLAB and the code used is attached with the report. The algorithm was tested on the given maps, with the given coordinates, while results were compared for different values of parameters as well. The lab work was divided into two tasks, programming the algorithm followed by smoothing of the generated path. The following sections explain the proposed solution and results for each of the given tasks.

2 RRT

2.1 Approach

The task of programming the algorithm on MATLAB was achieved following the guidelines provided in the instructions. The algorithm is not explained in this report as it has already been given in the instructions. But, the approach for a few important steps is detailed below, followed by the results.

For locating q_near, firstly, the direction was obtained using the unit vector in the direction of the q_rand point, that was then multiplied by the $delta\_q$ distance thus providing another point at $delta\_q$ distance in the direction of q_rand. q_new was generated following the incremental method as given in the instructions.

2.2 Results

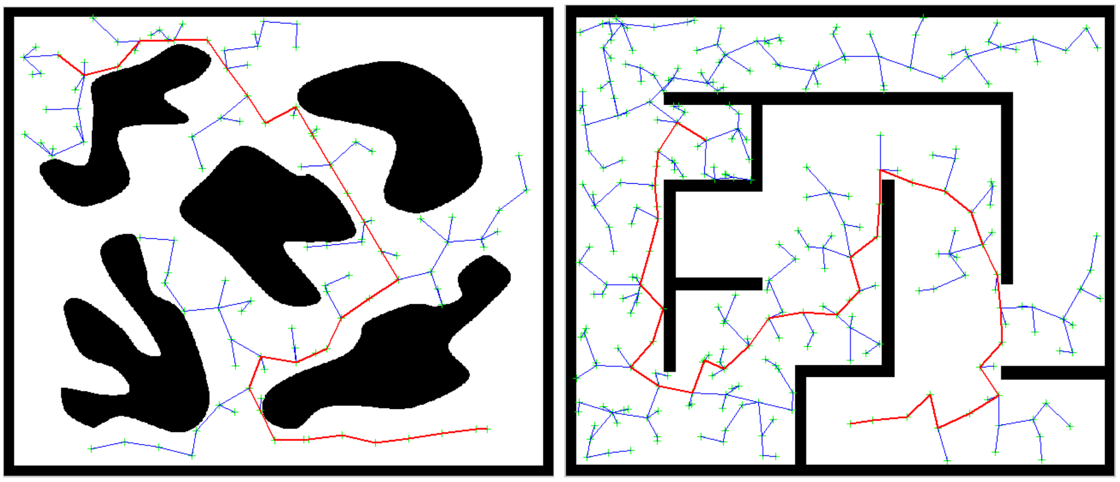
Figure 1: From Left: RRT results for (a) map; (b)maze;

The above images show the results for the RRT algorithm for the given environments. In the above figures, the computed path is shown in red. The algorithm usually converges within 1000 iterations. The given values for the parameters are well optimized for the given environments. The parameters were also changed and the observations are as follows:

- **probability** only changes the no. of iterations required for the algorithm to converge to the goal. No. of iterations increases with increase in probability.
- **k** does not have any significant effects on the result unless it is too low.
- **delta_q** has a little effect on the result as low *delta_q* would increase no. of iterations as well as computational cost, while high *delta_q* will reduce the accuracy by generating paths that cross through obstacles as shown in the figures below.

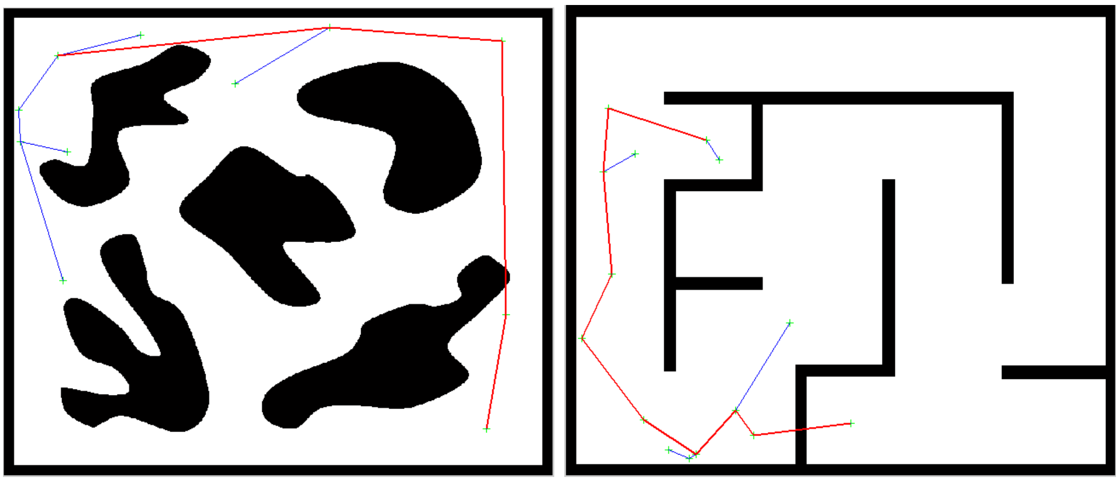

Figure 2: From Left: RRT results for (a) map with *delta_q*=400; (b)maze with *delta_q*=150;

As we can see from the images above, for higher *delta_q*values, the path crosses through obstacles.

3 Smoothing

The smoothing algorithm follows the below shown steps:

- In each iteration, at first *q_start* is always set to the start point.
- Get the unit vector from the *q_start* to *q_end*
- check if it is free space between *q_start* and *q_end* (add delta*unit vector)
- If yes, directly set *q_end* to *q_start*, put the *q_end* into *path_smooth* and this iteration finishes
- If no, update *q_start* to the next element in the path list, start from step 2 again

3.1 Results

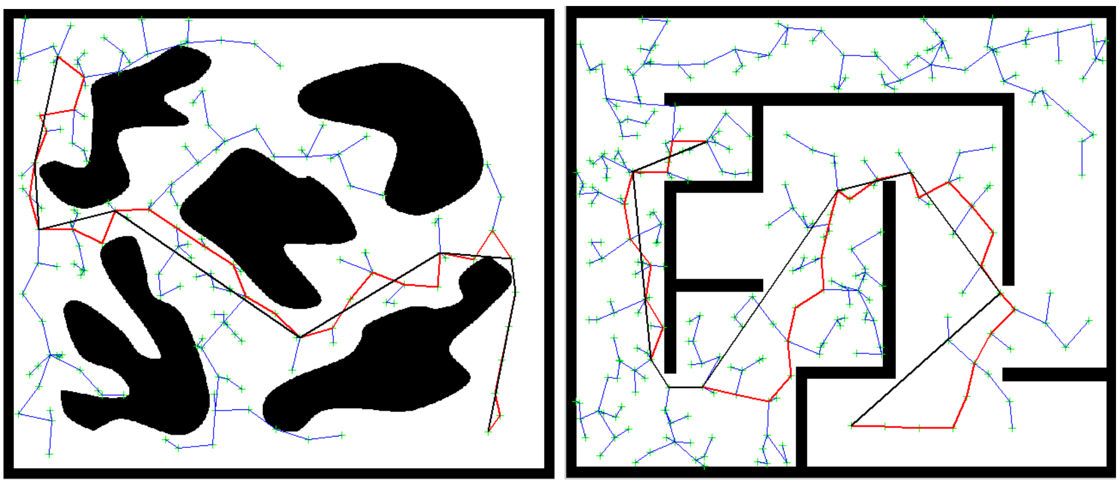

Figure 3: From Left: smoothing results for (a) map; (b)maze;

The above shown images show the smoothed path in black. Similarl to RRT, the parameter *delta* being passed to the function effects the result as high delta would provide path that would cross through obstacles. Hence, we can say that the given value $delta = 5$ is apt for the given maps. The images below show the results for different values of *delta*

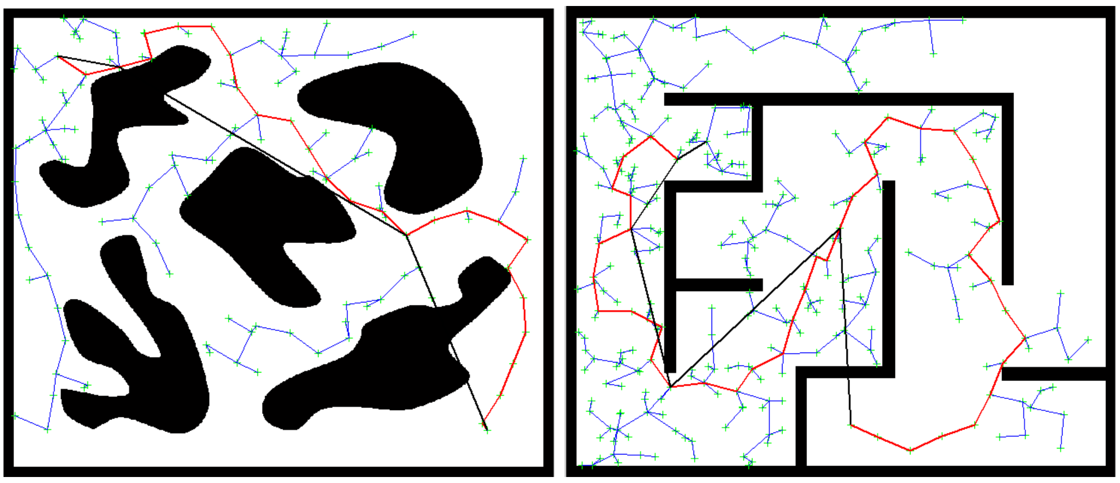

Figure 4: From Left: Smoothing results for (a) map with *delta*=100; (b)maze with *delta*=20;

4 Problems Faced

The following problems were faced while completing the lab work:

- In the beginning, the algorithm was not converging with *q_new* never coming equal to *q_goal*. After rigorous debugging, the problem was traced to computing the *q_new* point from *q_near* as *q_new* was being computed using the ratio method. Later it the norm of the unit vector was used to generate the same point, which was giving correct results.
- The algorithm was also tested on the *maze.mat* file given in *lab*1. But the algorithm did not provide correct results as it would require many more iterations to compute a path on such a dense maze with a most of the map occupied by obstacles.

5 Conclusion

RRT algorithm was successfully programmed on MATLAB along with the smooth function. The programs were giving accurate results that are shown in the above sections. Parameters were tested with different values and also the problems faced were discussed.