Autonomous Robots
Lab 3 Report: Sampling-based algorithms (RRT)

Mohammad Rami Koujan
M.Sc. VIBOT
University of Girona

April 13, 2016

1 Introduction

The aim of this lab assignment is to program a sampling-based path planning algorithm, which is RRT (randomly exploring random trees). RRTs are particularly suited for problems that involve differential constraints. Additionally, they are useful for problems with a lot of Degrees of Freedom or constraints. In fact, Instead of finding an optimal solution considering the whole environment, this algorithm only takes into account few samples, where each sample is a robot configuration. Basic properties of RRTs are established, including convergence to a uniform coverage of nonconvex spaces.

2 Rapidly-Exploring Random Trees

In this section the implemented RRT algorithm is explained in details. The implementation starts by initializing the "vertices" variable with the value of q_start. Then for each sample k the followings tasks are performed:

- A random number between 0 and 1 is drawn from the standard uniform distribution by the Matlab random number generator. This number is checked whether it is smaller to the probability variable "p" or not. If yes, which happens with probability p, q_rand is set to q_goal. Otherwise, q_rand takes random pair of numbers (x and y coordinates) in the dimension of the map.
- Next, the euclidean distance between each vertex in the "vertices" variable and q_rand is computed and the vertex that has the minimum distance is considered as q_near.
- Thereafter, q_new is set by giving it the coordinates of the head vertex of the vector [delta_q, 0] after rotating and translating this vector by θ and q_near respectively, where θ is the angle between q_rand and q_near vertices, as shown in the following figure.

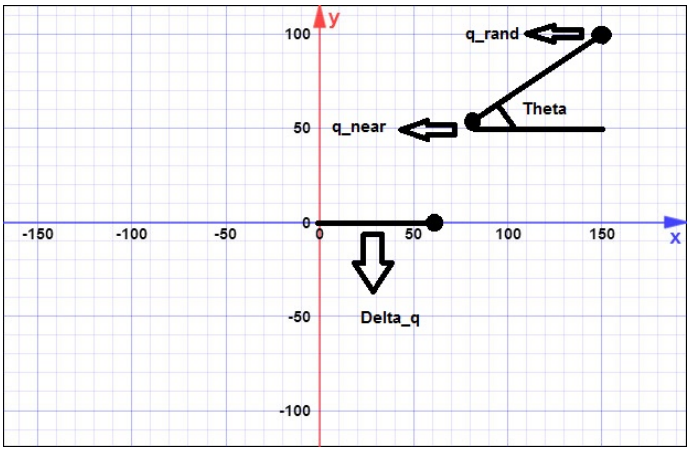
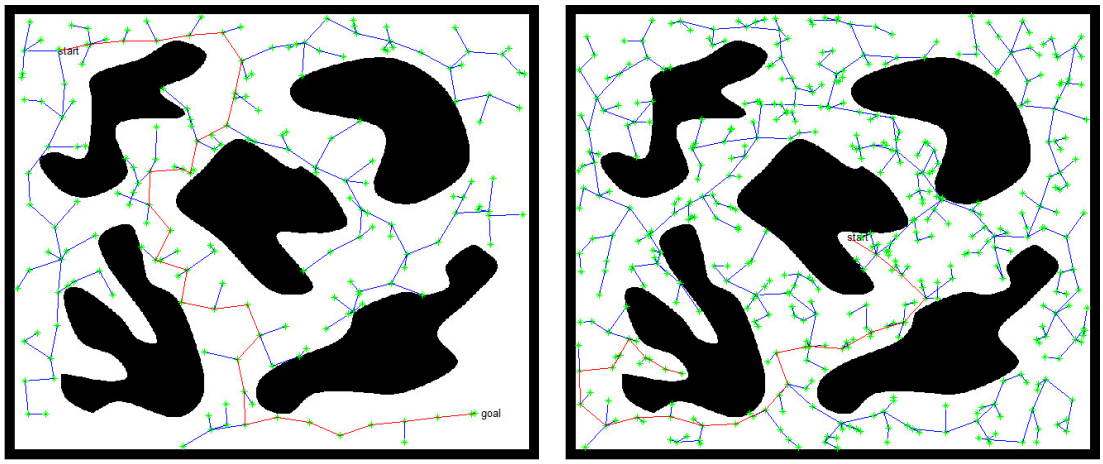


Figure 1: The idea of computing q_new

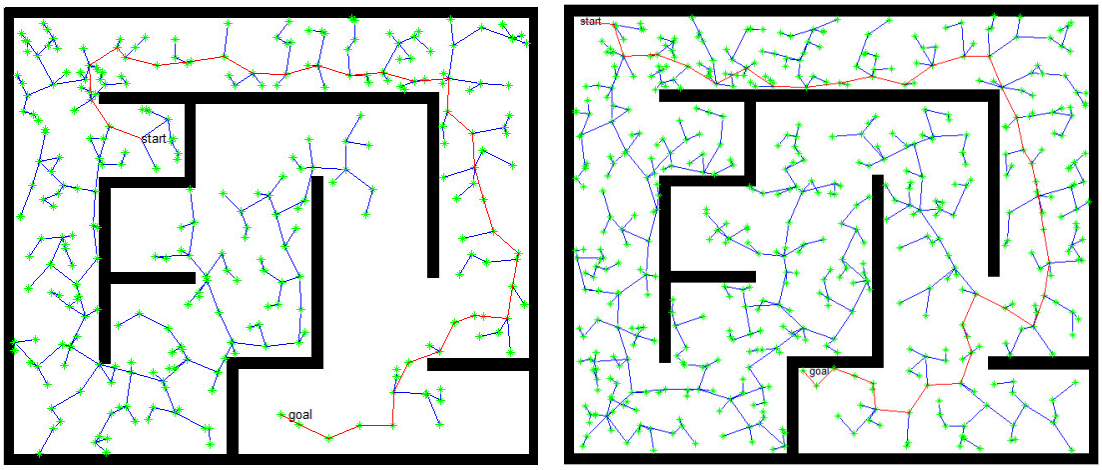
- If the distance between (q_near, q_new) is greater than the distance between (q_near, q_rand), then q_new is given the value of q_rand since in this case the distance between q_near and q_rand is smaller than delta_q.
- The next step is to check if the value of the map at q_new is equal to 1 (obstacle). If yes, the program goes back to the step of generating a random number. Otherwise, the edge connecting q_near and q_new is tested to make sure that it does not pass through an obstacle. This is done by using an incremental strategy by which the distance between q_near and q_new is divided into e.g 10 and then this distance is increased gradually starting from q_near and each time the new vertex is checked weather it lies on an obstacle or not. In case it lies, the program goes back to the step of generating a new random number. Otherwise, q_new is stored in "vertices" variable and [index_of_q_new, index_of_q_near] is stored in "edges" variable.
- Finally, q_new is checked if it is equal to q_goal. If it is equal, the "path" variable which should store the entire path between the start and goal vertex is filled in a reverse manner, from the index of the goal back to the start node, and path variable is returned by the program. Otherwise, new iteration starts.
  When the program does no succeed in finding a solution during k iterations, it returns the vertices that have been explored during those iterations. Figure 2 shows the result of testing the implemented algorithm on the shown map for the following parameters:
  k=10000, delta_q=50 and p=0.3
  q_start=[80, 70]; q_goal=[707, 615] (a)
  q_start=[424,350]; q_goal=[175,555] (b)



(a) (b)

Figure 2: testing the implemented RRT algorithm on different start&goal pairs

Figure 3 shows the result of testing the implemented algorithm on another map for the following parameters:
k=10000, delta_q=50 and p=0.3
q_start=[206, 198]; q_goal=[416, 612] (a)
q_start=[25, 25]; q_goal=[360,548] (b)



(a) (b)

Figure 3: testing the implemented RRT algorithm on different start&goal pairs

3 Smoothing

This section focuses on the implementation of a smoothing algorithm for obtaining a shorter and less noisy path. The idea is mainly to connect q_goal from q_start. If this fails, a closer position should be tried until it connects. Once q_goal is connected, the process starts again with its directly connected position.

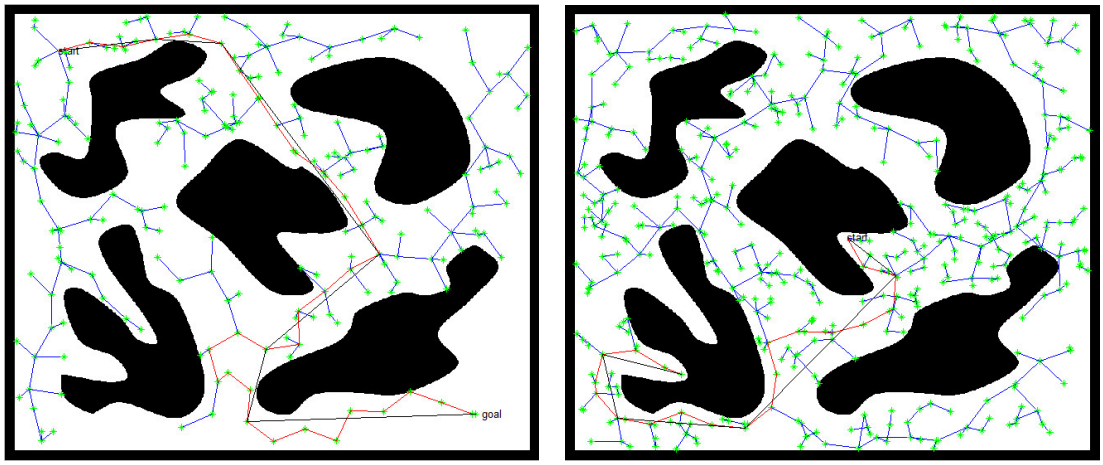
At the beginning, the following parameters are defined:

1. flip_path: it contains the path from the starting vertex to the goal vertex and every element in this parameter represents the index of the corresponding path vertex.
2. v_end: it stores initially the value of the goal vertex.
3. v_end_index: it keeps initially the index of the last vertex in flip_path, the goal vertex index.
4. path_smooth: variable used to store the final smoothed path and it contains initially the goal point.

A "while" loop is then used, which stops when the last element of the "path_smooth" variable is equal to one (the index of the start point), to generate the smoothed path as follows:

- An index "i" is initialized to 1.
- Another loop is used to iterate from the i'th vertex of the flip_path parameter till the last checked vertex for connectivity, initially it is the goal vertex.
- A variable v_start is set to the coordinates of the i'th vertex of the flip_path parameter.
- A for loop is used to check incrementally if the path connecting v_start and v_end passes through an obstacle or not using the same principle stated in the RRT function (based on a transformation matrix and gradual increase by delta of the distance between the mentioned vertices). If yes, "i" is incremented by one and another iteration of the second nested loop starts. The purpose of increasing "i" is to check the connectivity with a closer vertex to v_end. Otherwise, the value of the flip_path at index "i" is used to update the path_smooth and v_end_index variables while v_end is updated with the coordinates of the vertex whose index is stored in the i'th position of the flip_path variable. Then, another iteration of the first loop starts.
- This process keeps repeating until the index of the start vertex is stored in path_smooth parameter, which causes the first loop to stop iterating.

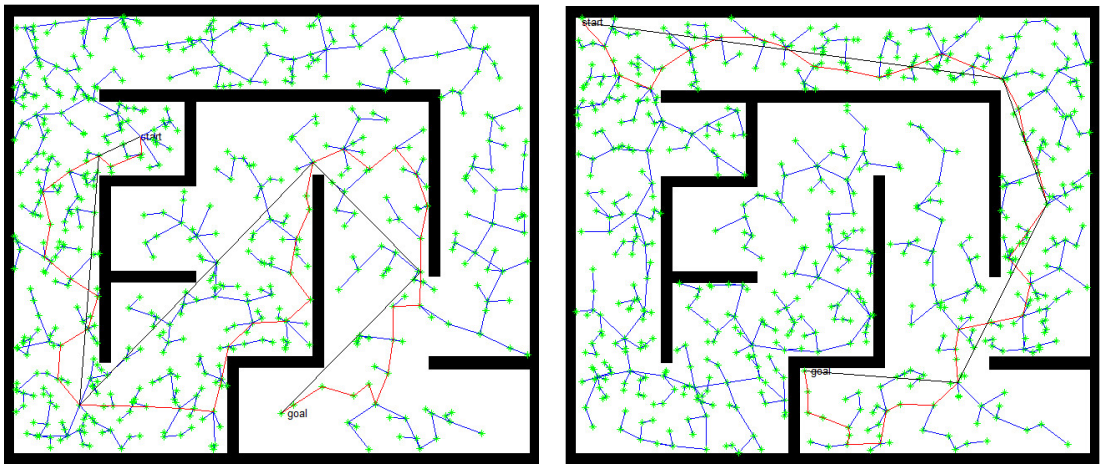
Figure 4 shows the results of testing the "smooth" function on map 1 with delta=5 and the same input parameters used to generate figure 2.



(a) (b)

Figure 4: testing the implemented RRT & smooth algorithms on different start&goal pairs

Figure 5 shows the results of testing the "smooth" function on map 2 with delta=5 and the same input parameters used to generate figure 3.



(a) (b)

Figure 5: testing the implemented RRT & smooth algorithms on different start&goal pairs

Note: black lines shown in figures 4&5 represent the smoothed path.

4 Conclusion

In this lab assignment, a sampling-based path planning algorithm (RRT) is designed, implemented, and tested successfully on two different maps. A smoothing function is also implemented in order to smooth the given path by the RRT function which helps the robot to travel a smoother path with less distance.