Bi-Directional RRT*

Introduction

Bi-Directional RRT* is a sampling-based algorithm which is a modification of the RRT* which in-turn is a modification of the RRT algorithm. In order to understand the working behind Bi-Directional RRT*, let us first understand the basis of the RRT and RRT*.

Theory and Explanation

RRT - Rapidly exploring Random tree (RRT) is a sampling-based algorithm whose premise is quite straight forward. RRT algorithm generates random points in the configuration space and are connected to the closest available node. Each time a vertex is created, a collision check must be performed to ensure that the vertex lies outside of the obstacle. Additionally, another check must be made to ensure that the connecting line between the randomly generated child node and the closest neighboring node must be free from obstacles. Iterating this process multiple times populates the area with nodes, and subsequently reaches in the vicinity of the destination after which the algorithm traces the final path from the source to destination using backtracking the parent node of the goal node and so on until the source node is finally reached. This outputs the final path.

The method of generating random nodes in the configuration space is a design decision. Simple methods such as using a built-in random number generators can be used for basic applications. During the process of generating random nodes, if a node is generated a bit too far from the nearest node, there must exist multiple nodes at a step interval of distance delta to reach that node.

Moreover, the method of chaining the randomly generated vertex varies too. One method involves calculating the vector that forms the shortest distance between the new valid randomly generated node and the closest node. Alternatively, the randomly generated vertex can be chained to its closest link by storing discrete nodes. This method requires less computation and is much straight forward and easy to implement, hence we have used the latter method in our program.

There also exists a limit on the maximum number of nodes that can be generated, and if the algorithm exceeds that limit, the algorithm declares that it failed to find a path and terminates.

*RRT** - The major modification of RRT* from RRT is that whenever the random node is generated, it searches for other nodes inside a circle of fixed radius and calculates the cost of each of those nodes from the source node and links the new node to whichever has the least cost.

*Bi-Directional RRT** - Bi-Directional RRT* is simply as the name suggests. There are two RRT* trees; one extending from the source node and the second extending from the goal node. When the newly generated nodes of either of the tree comes in the neighborhood of each other, the planner attempts to connect between the two trees using the new extension and the closest node on the opposite tree. When nodes of both the trees are at sufficiently close to each other the trees are merged, and a unique path is derived by backtracking, thus obtaining a path from the source to goal node.

Complexity of Bi-Directional RRT* is given by O(n)

Key building blocks

- Sampling – Samples are assumed to be of uniform distribution, even though the random number generator has a certain level of bias. Each sample is independent and identically distributed.

- Nearest Neighbor – Given a graph G=(V,E), where $V \subset X$, a point $x \in X$, the function Nearest: (G, x) $\rightarrow$ $v \in V$ returns the vertex in V that is closest to point x in terms of distance. We have assumed Euclidean distance in our program.

- Near Vertices - Given a graph G=(V,E), where $V \subset X$, a point $x \in X$, and a positive radius ‘r’, the function Near Vertices: (G,x,r) returns all vertices inside the circle of radius r centered at point x.

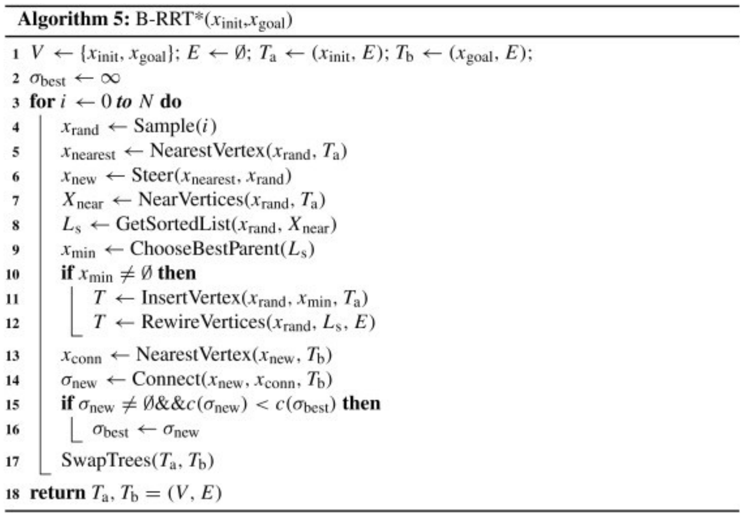
- Collision Test – Given two points $x,y \in X$, the Boolean function CollisionFree(x,y) returns True if the line segment between x and y lies in X_{free} and False otherwise.

- Choose Best Parent – Given parent vertex $X_{parent} \in X$, the function returns a vertex that has a collision free path with minimum cost from source vertex x_{init} to new randomly generated point $x_{rand.}$

Inputs

- Maximum number of nodes
- Start Node Location
- Goal Node location
- Delta step-distance
- Maximum merging distance between two trees, ‘e’
- Window resolution

Algorithm Pseudo code

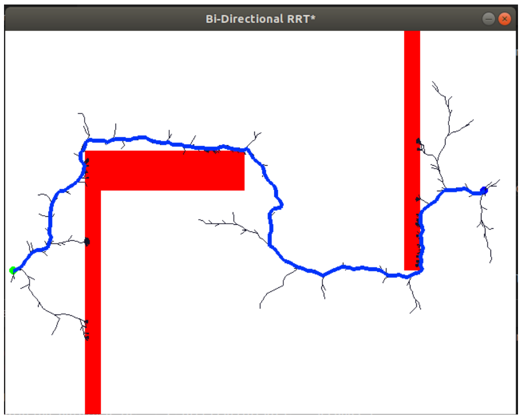


```
Algorithm 5: B-RRT*($x_{init}$,$x_{goal}$)
$V \leftarrow \{x_{init}, x_{goal}\}$; $E \leftarrow \emptyset$; $T_a \leftarrow (x_{init}, E)$; $T_b \leftarrow (x_{goal}, E)$;
$\sigma_{best} \leftarrow \infty$
for $i \leftarrow 0$ to $N$ do
    $x_{rand} \leftarrow \text{Sample}(i)$
    $x_{nearest} \leftarrow \text{NearestVertex}(x_{rand}, T_a)$
    $x_{new} \leftarrow \text{Steer}(x_{nearest}, x_{rand})$
    $X_{near} \leftarrow \text{NearVertices}(x_{rand}, T_a)$
    $L_s \leftarrow \text{GetSortedList}(x_{rand}, X_{near})$
    $x_{min} \leftarrow \text{ChooseBestParent}(L_s)$
    if $x_{min} \neq \emptyset$ then
        $T \leftarrow \text{InsertVertex}(x_{rand}, x_{min}, T_a)$
        $T \leftarrow \text{RewireVertices}(x_{rand}, L_s, E)$
    $x_{conn} \leftarrow \text{NearestVertex}(x_{new}, T_b)$
    $\sigma_{new} \leftarrow \text{Connect}(x_{new}, x_{conn}, T_b)$
    if $\sigma_{new} \neq \emptyset \&\& c(\sigma_{new}) < c(\sigma_{best})$ then
        $\sigma_{best} \leftarrow \sigma_{new}$
    SwapTrees($T_a$, $T_b$)
return $T_a, T_b = (V, E)$
```

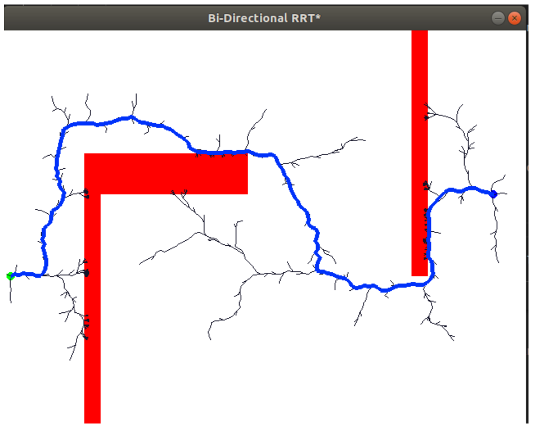
Results

The Bi-Directional RRT* was successfully implemented. The source node is represented by a green dot while the goal node is represented by the blue dot on the right side of all the below images. The path is highlighted and the obstacles are represented in red.

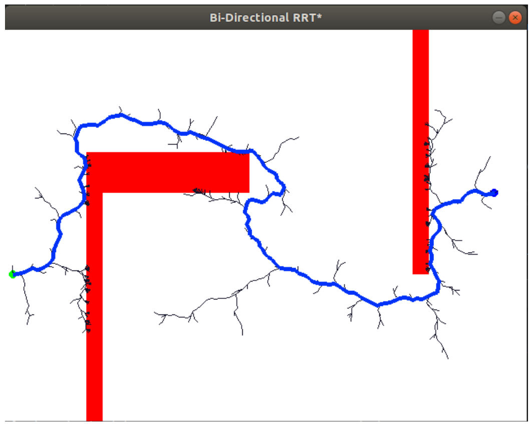
Below images depict total nodes(n) used to derive solution and time elapsed(t) for it.



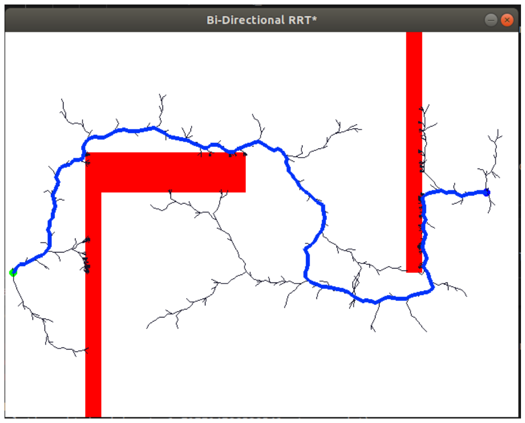
n = 902   t = 1.0805 s



n = 1017   t = 1.8702 s



n = 1107   t = 1.7177 s



n = 1011   t = 2.0297 s

References

[1] Incremental Sampling-based Algorithms for Optimal Motion Planning
[2] Sampling-based Algorithms for Optimal Motion Planning

Conclusion

Bi-Directional RRT* is an efficient way of finding optimal path solutions by extending two trees from their initial points and merging them to obtain a final path connecting the two initial points. We observed as the tree expands and the number of nodes increase, it does not necessarily mean that the solution will be obtained faster, although it does increase the probability of finding an optimal solution. In any case, Bi-Directional RRT* guarantees shortest path.