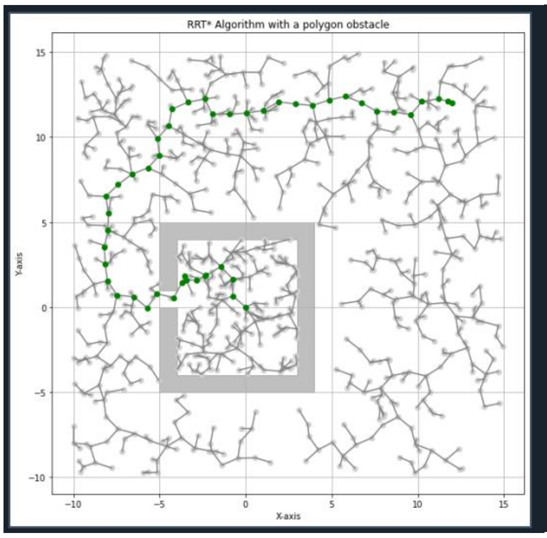
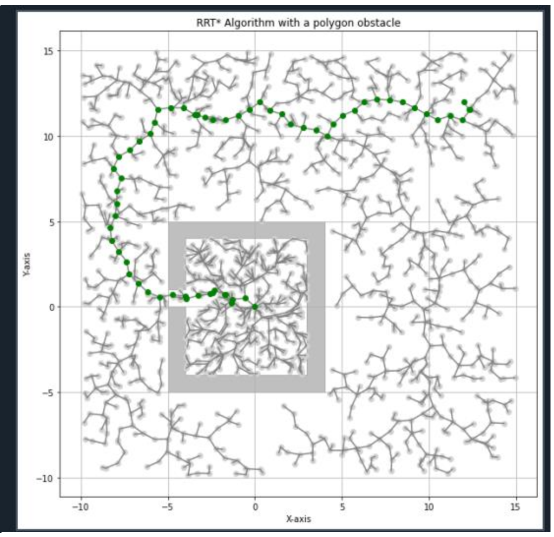
Gaurav Surtani – 017047576

First time we reach the path:





1 – First Time we reach Goal:

[44.4379327641378]: Goal distance

[(12, 12), (11.784822911497397, 12.091606983273039), (10.959973013181843, 13.162954792656718), (9.96912847432601, 11.425650242550557), (9.156783312927885, 9.59805747096465), (7.339866622786201, 10.43400827160102), (5.501598382317205, 11.22189777913556), (3.5169615977297433, 10.974477421722632), (2.2879067323971465, 10.618022501869717), (0.7946656326864034, 9.287522213205843), (-0.9163396748385326, 10.323119064119123), (-2.843292658705228, 10.85870490968194), (-4.769227599599925, 10.319469707343536), (-6.5529742844174805, 9.414899221199764), (-7.405825070212981, 9.15665369500849), (-8.737684903496602, 9.956823183411698), (-8.657316808363973, 7.958438593477931), (-8.938095744793404, 5.978245878397513), (-9.237037236376953, 4.000713581667621), (-8.89449122550536, 2.0302663658214417), (-6.947420374839332, 2.4873386822951815), (-6.866668598806131, 1.1340723906981367), (-5.094597356756147, 0.20683805510684294), (-3.400539125331936, 0.7257789743711208), (-1.493912832429691, 1.3297460092445843), (0, 0)]: Goal Nodes #1'

<u>2 –Recommended path after 2000 runs:</u>

[41.82648012577937]: Goal distance#2

[(12, 12), (11.729637588232759, 12.102156054236136), (11.215608627810182, 12.259124928403324), (10.229410403130897, 12.093556025406942), (9.592664008537094, 11.322482728649888), (8.605056294451067, 11.479425403145305), (7.605737458927342, 11.516328857827817), (6.740166760901198, 12.017115605574334), (5.817690022478777, 12.40316827953157), (4.841851735886873, 12.184674255765751), (3.8946030186378033, 11.864174852224977), (2.8981816363548027, 11.948699575999946), (1.9074319171726124, 12.084401432805263), (1.0462864073737501, 11.576042887137682), (0.05972161324647263, 11.412672427786541), (-0.9384578316425136, 11.352358285157177), (-1.93843891207939, 11.346206962673543), (-2.378032330369906, 12.244403837860526), (-3.3595954591352664, 12.053265604510011), (-4.2747798140054645, 11.650230126418046), (-4.4640484077887, 10.668304773141477), (-5.123279187223393, 9.916364097658413), (-4.99776525284932, 8.92427224088408), (-5.676236737318881, 8.189645502491262), (-6.606232760827886, 7.822075922333385), (-7.399458289930737, 7.2131479627359445), (-8.113515191904854, 6.513060296266147), (-7.967470996671405, 5.52378222967485), (-7.99181902356732, 4.524078686825134), (-8.193672168913341, 3.5446628876295536), (-8.157966876052475, 2.5453005248893312), (-8.022042604306215, 1.554581295061703), (-7.47612664943491, 0.7167413519836994), (-6.482001160451005, 0.6085077520596849), (-5.7152114014435815, -0.03339057749207375), (-5.140625664034905, 0.785053819132723), (-4.166195267265061, 0.560364251393224), (-3.700137296408506, 1.4451184363514362), (-3.5469275737708887, 1.8287255517702619), (-3.4455068555751893, 1.6035803576103191), (-2.8703283836295332, 1.5789507280084774), (-2.381844703715596, 1.8159708550981186), (-2.297494188037722, 1.8687312307886468), (-1.441313747895696, 2.385408164585973), (-0.7478958021931087, 1.6648725633991868), (-0.7469564162148455, 0.6648730046222922), (0, 0)]: Goal Nodes#2

3 – Path Calculated Path by pen and paper:

1. Move from (0, 0) to (-5, 1):
   Distance=(−5−0)2+(1−0)2=(−5)2+12=25+1=26Distance=(−5−0)2+(1−0)2=(−5)2+12=25+1=26
2. Move from (-5, 1) to (-5, 5):
   Distance=(−5−(−5))2+(5−1)2=02+42=4Distance=(−5−(−5))2+(5−1)2=02+42=4
3. Move from (-5, 5) to (12, 12):
   Distance=(12−(−5))2+(12−5)2=172+72=289+49=338Distance=(12−(−5))2+(12−5)2=172+72 =289+49=338
4. **total cost of the path from (0, 0) to (-5, 1) to (-5, 5) to (12, 12) is approx 27.5 units.**

RRT* Algorithm Implementation

Overview

The Rapidly-exploring Random Tree Star (RRT*) algorithm is an incremental sampling-based motion planning algorithm optimized for fast exploration by incrementally building a space-filling tree. It converges towards the optimal solution, distinguishing it from the basic RRT algorithm.

Algorithm

The RRT* algorithm builds a tree rooted at the start node to explore the search space. Key steps include sampling a random state, finding the nearest node, steering towards the sampled state, checking collision, rewiring the tree, and adding the new node. The algorithm repeats these steps for a specified number of iterations, connecting the goal node to the tree when within range. The final path is extracted by following parent pointers from goal to start.

Implementation

RRTNode Class

- Represents each node in the tree.
- Contains x, y coordinates, parent node reference, and cost to reach the node.

distance(node1, node2)

- Calculates the distances between two nodes; random node and tree nodes.

find_nearest(nodes, target)

- Chooses the closest node in the 'nodes' list to the given 'target' node.
- Returns the nearest RRTNode class.

steer(node1, node2, max_dist)

- Creates a new node in the direction of node2 within max_dist from node1.
- Handles reaching the goal exactly or extending towards it.

is_path_clear(n1, n2, obstacles)

- Checks if the straight path between two nodes is collision-free.
- Returns True if clear, False if colliding.

extend_tree(tree, target, max_dist, obstacles, r)

- Grows the RRT tree towards a sample using the above functions.
- Rewires nodes within radius r if a lower cost path is found.

rrt_star(start, goal, iterations, max_dist, obstacles, r)

- Runs the RRT* algorithm for a given number of iterations.
- Calls extend_tree to expand the tree.
- Returns the final tree and optimal path to the goal.

visualize_rrt(tree, path, obstacles)

- Plots the RRT tree, obstacles, and the final path.
- Useful for visualizing progress and the result.

Pseudocode

```
Initialize RRT tree with start node for N iterations:

Sample a random point X in space

Find the nearest node Y in the tree to X

Steer from Y towards X to extend tree and get new node Z if the path between Y and Z is collision-
free:

Rewire:

for each node N in the neighborhood of Z: if cost(Z) + dist(Z, N) < cost(N): Update N.parent = Z
Update N.cost Add Z to the tree Attempt to connect the goal to the tree Extract the final path by
following parents from goal to start
```

Conclusion

This implementation of the RRT* algorithm includes obstacle avoidance and path optimization. It can be further enhanced with features like dynamic obstacles, kinodynamic constraints, and smoothing. The modular implementation allows easy adaptation for different environments and robot models.

Usage

- Read start, goal, and obstacle vertices from an input file.
- Set parameters like maximum extension distance, rewiring radius, and the number of iterations.
- The algorithm generates an RRT tree and an optimal path that is visualized at the end.
- The code can be adapted for different environments and robot models.

This concludes the documentation of the RRT* algorithm implementation. Modifications or expansions can be made based on specific requirements.