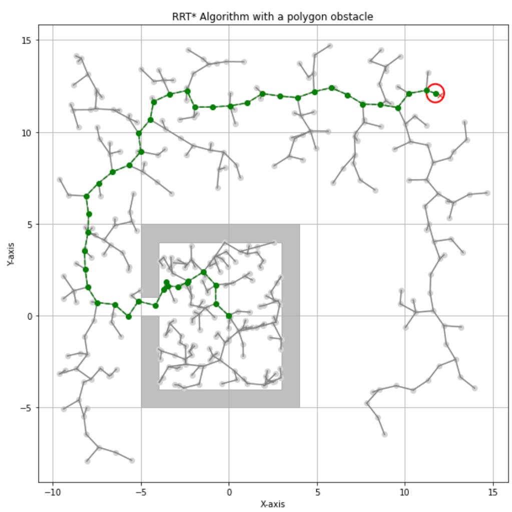
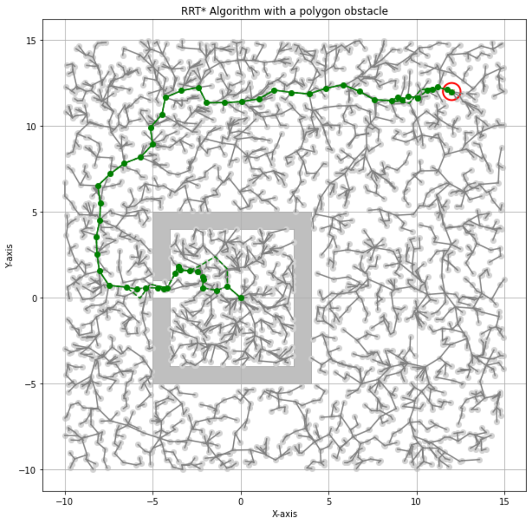
HWK Assignment 3 – RRT* Algorithm

Gaurav Surtani – 017047576

First time we reach the path & Optimal Path





1 – First Time we reach Goal:

[41.82648012577937]: Goal distance

<u>2 –Recommended path after 4000 runs:</u>

[41.53746162617477]: Goal distance#2

<u>3 – Path Calculated Path by pen and paper:</u>

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

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.

is_inside_circle(point, center, radius):

- The radius around the target circle.

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.

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.

.