Path Planning Algorithm Report

0.1 Introduction

This is a latex report contains analysis of Question 1,2 of Take home end semester examination.

0.2 Path Planning with RRT Algorithm:

Path planning is a crucial aspect of autonomous robotics systems, enabling them to navigate complex environments efficiently. RRT algorithm is a sampling-based approach widely used for path planning when the environment map is available. This report explores enhancements to the RRT algorithm and their impact on path planning efficiency.

Objective of the question:

- we have to write a program that plots a rectangle of some arbitrary width and height, which acts as the walls of the environment, and place 3-4 obstacles, which are rectangles of around one-tenth the size of the bounding wall.
- with that we have to perform RRT algorithm, perform smoothing on it, find a greedy approach from start to the goal point.
- after this we have to run the stimulation 1000 times for each variant and plot average number of nodes needed to reach the goal.

Setting up the environment:

We import matplotlib module (for graph and plotting purpose), numpy module (for numerical computations) and random module. After this we create a rectangle which acts as the walls of the environment.with the help of random.randint function we generate obstacles at random points(assuring that the obstacles don't overlap each other).

0.3 Implementing RRT algorithm:

The RRT algorithm builds a tree structure by iteratively sampling random points in the environment and connecting them to the nearest existing node. The algorithm continues until a path to the goal is found.

code explanation

- `class Node`: Represents a point in the RRT tree.
- `def distance(p1, p2)`: This function calculates the distance between two points using numpy's `hypot` function.
- `def nearest_node()`: This function finds the node in a list of nodes that is closest to a given random point.
- `def steer()`: This function extends the tree from the nearest node.
- `def line_collision_check()`: This function verifies path feasibility.
- `def extract_path()`: This function retrieves the path to the goal.

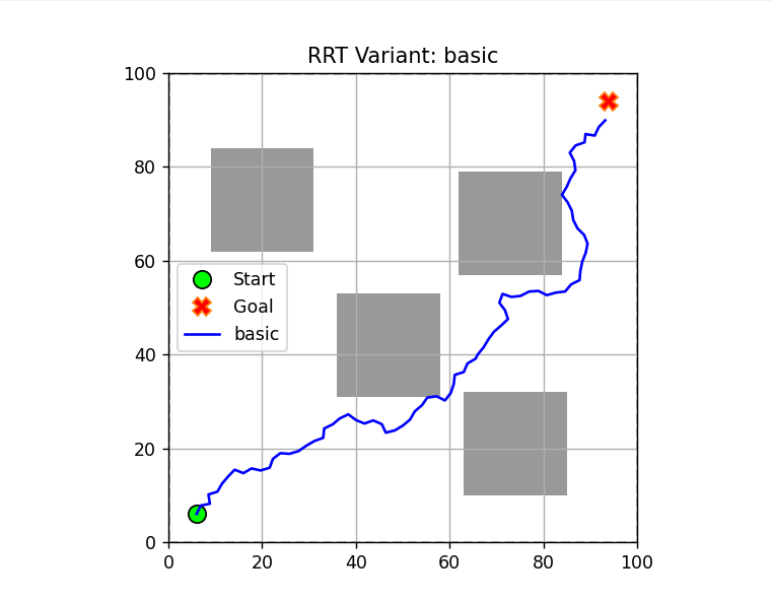


Figure 1: basic

0.4 Trajectory smoothing:

Trajectory smoothing is essential to ensure that the generated path is feasible for robotic motion, reducing unnecessary sharp turns and improving navigation efficiency.

code explanation

- def smooth_path(): this function attempts to smooth a given path by iteratively removing intermediate points if a direct path between randomly selected pairs of points is possible and does not collide with any obstacles. It repeats this process until either the path cannot be further optimized or a maximum number of iterations is reached.

0.5 Spacing between obstacles :

We have to take it to account about spacing while path planning is done to prevent collisions and a safe navigation for the robot. This is achieved by Implementing a buffer around obstacles due to which the random point sampling is done away from obstacle edges.

code explanation

- def find_free_point():This function generates random points within a defined space, ensuring they are not within any obstacles with a specified buffer zone.

0.6 Greedy approach :

Greedy algorithm prioritizes points closest to the goal, favoring paths that show promising progress towards achieving the objective. By favoring such paths,this algorithm can potentially reduce the overall search effort and converge faster towards the goal compared to random selection strategies.

code explanation

- The ' rrt ' function is modified to bias random point sampling towards the goal with a certain probability .This encourages the algorithm to explore areas closer to the goal more frequently .

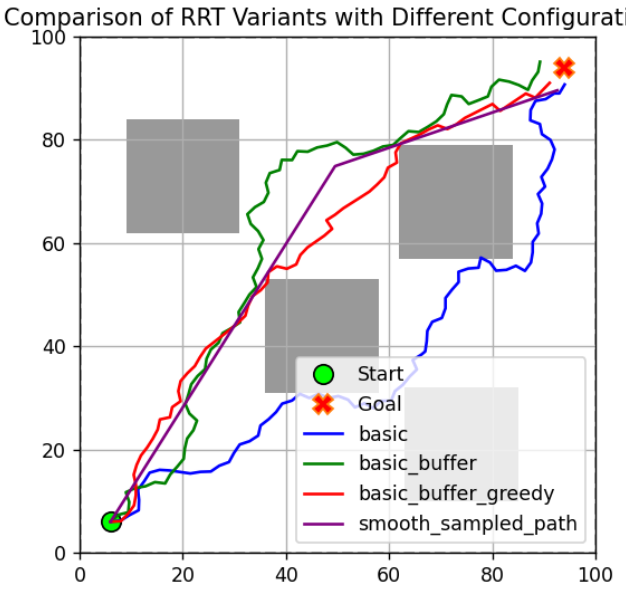


Figure 2: All algorithms

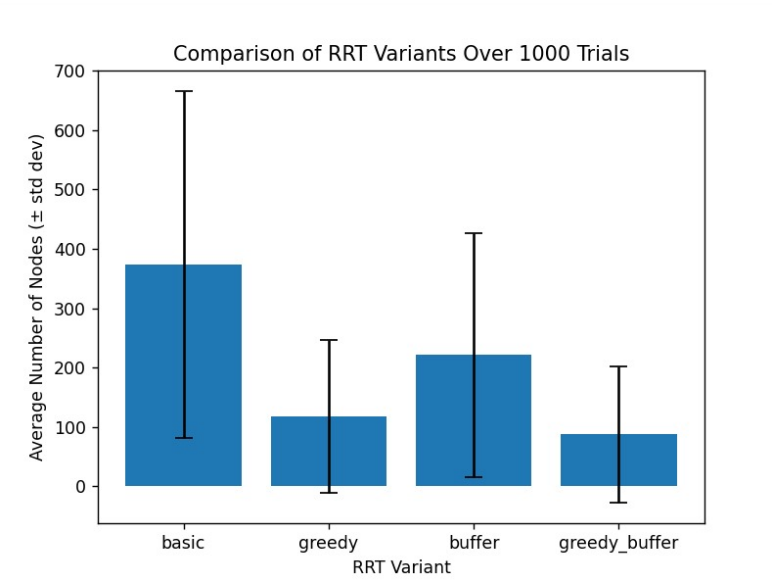
0.7 Simulation and Results:

We conducted simulations with different variants of the RRT algorithm, varying parameters such as environment size, obstacle density, and sampling strategies. The simulation results show the average number of nodes required to reach the goal for each RRT variant, indicating the performance and efficiency of each approach. We discuss the effectiveness of each RRT variant in terms of path planning performance and computational complexity. Insights are provided on potential improvements and future research directions.

- **Basic RRT:** Serving as the reference point, this variant relies on purely random sampling, typically resulting in a higher node count compared to other variants.
- **RRT with Buffer:** Adding buffer zones increases safety but it can also increase the path length due to reduced space for sampling.
- **Greedy RRT:** Employing a greedy algorithm, this variant notably reduces the average node count by prioritizing sampling towards the goal configuration.

- **Greedy RRT with Buffer:** Striking a balance between efficiency and safety, this variant offers a compromise by combining the advantages of the greedy algorithm with the safety enhancement provided by buffer zones.

RESULT



It takes about 5-10 min to run 1000 simulations and get the out put of the plot

0.8 Closure:

This assignment demonstrated the versatility and adaptability of the RRT algorithm in addressing various challenges in robotic path planning, from generating feasible paths to optimizing for efficiency and safety. By combining theoretical concepts with practical implementations and analysis, valuable insights were gained into the complexities of real-world navigation scenarios and the importance of algorithmic optimizations for autonomous systems.