ROBOT MOTION PLANNING

PROJECT REPORT

ON

RAPIDLY-EXPLORING RANDOM TREE

FOR NON-HOLONOMIC ROBOT

BY

SAKSHI SHRIVASTAVA

(800987926)

1. **INTRODUCTION:**

The basic motion planning problem asks for computing a collision-free, feasible motion for an object (or kinematic device) from a given start to a given goal placement in a workspace with obstacles. One of the philosophy to address this problem is sampling-based motion planning. The main idea here is to avoid the explicit construction of C-obstacle and instead conduct a search that probes the C-space with a sampling scheme. Sampling-based Methods can be divided into two categories: multi-query methods and single-query methods. For multi-query methods, a probabilistic roadmap is built during the preprocessing phase and stored inside the robot. After given the start and a goal configuration, the robot will search the roadmap for a path joining the two nodes. For single-query methods, no roadmaps are built ahead of the task. The robot constructs the roadmap online during the searching, with sampling conducted in the process to avoid being stuck in the local minima due to any deterministic approach.

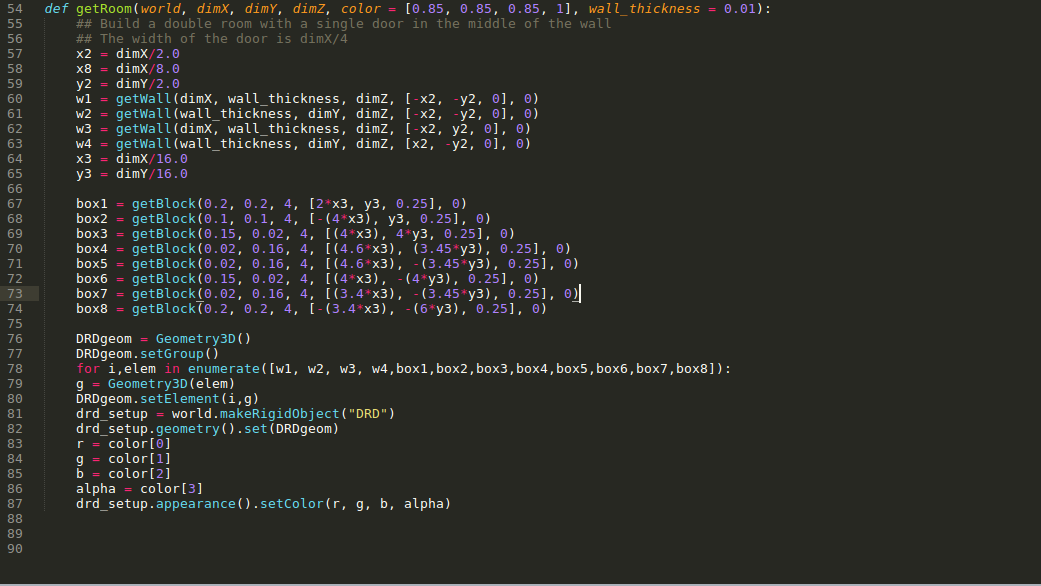
1. **IMPLEMENTATION STEPS:**

For the purpose of this project, the “Randomized Kinodynamic Planning” research paper authored by Steven M. LaValle and James J. Kuffner, Jr was taken as reference. Following are some of the specifications for the algorithm implementation:

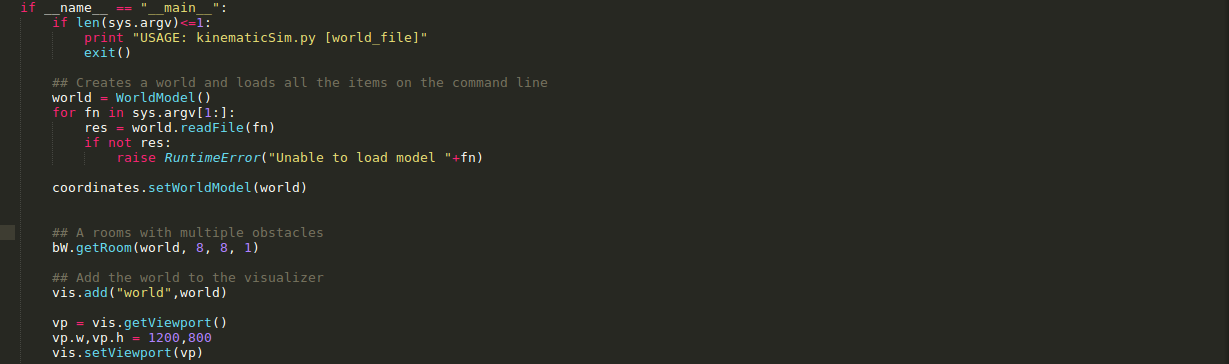
1. The Configuration of the robot at any point is considered in the format: (x, y, theta, t, u1, u2). Where t is the number of time steps required to reach the configuration. Each time step (dt) equals to 0.01 seconds. u1 is the linear velocity which is 0.5 m/sec and u2 is the angular velocity that is pi/sec.
2. All distances are considered as Euclidean distances.
3. KD Tree is used to store all the configurations and find the nearest neighbor to the randomly generated point while finding the new node.

Implementation Steps:

1. Make changes in the buildWorld.py file. Create walls and other obstacles that suites the problem. For the project and simple room with 4 walls and multiple obstacles with different shapes were created.



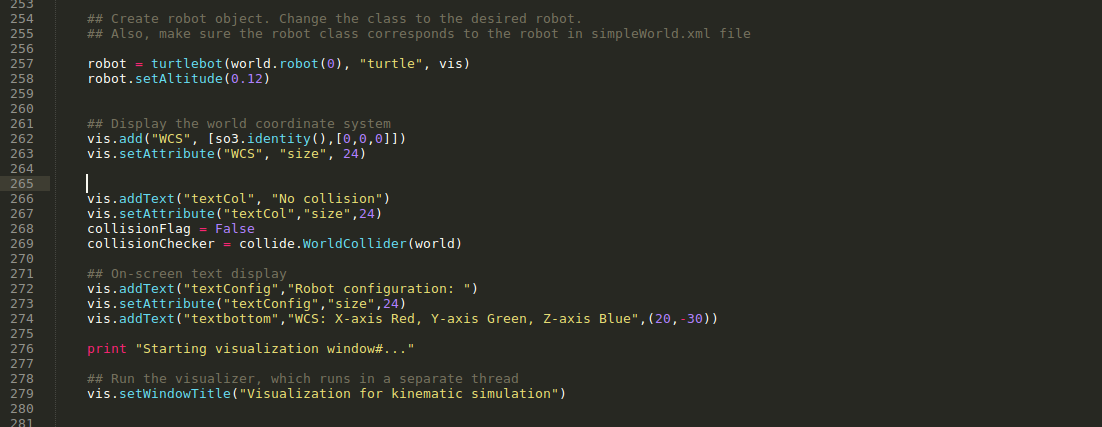
After making changes to the buildWorld.py file read this file in the kinematicSim.py file to create a world object.



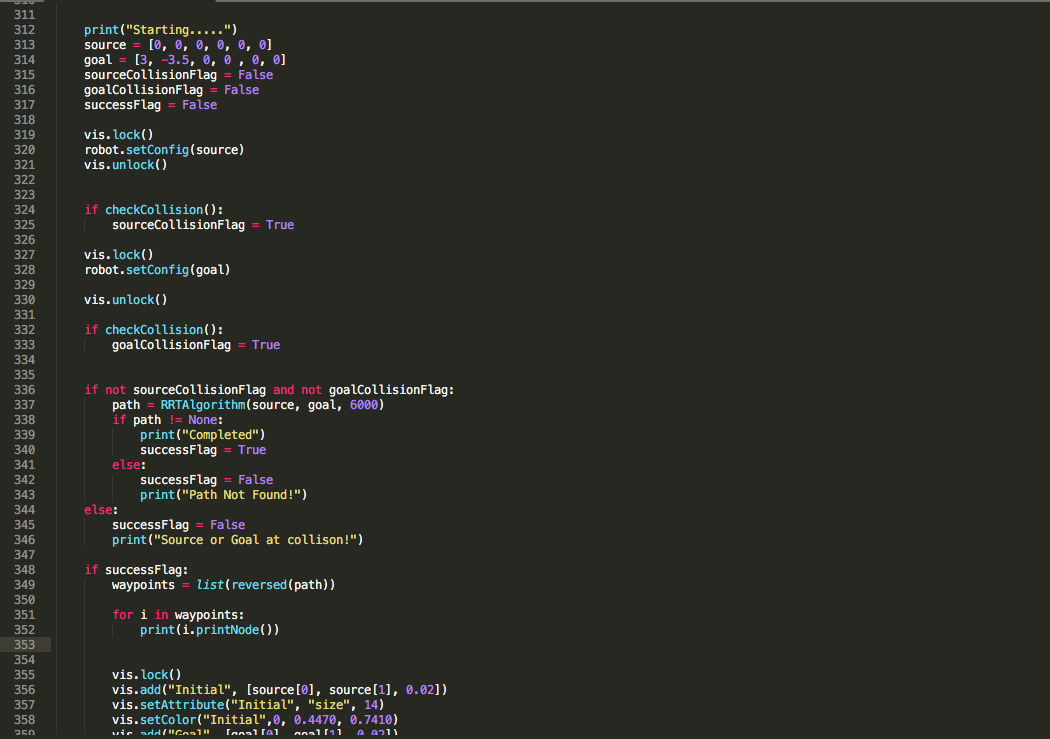
1. Make changes in the simpleWorld.xml file by adding the robot of your choice. For the project turtlebot was considered as the sample robot.



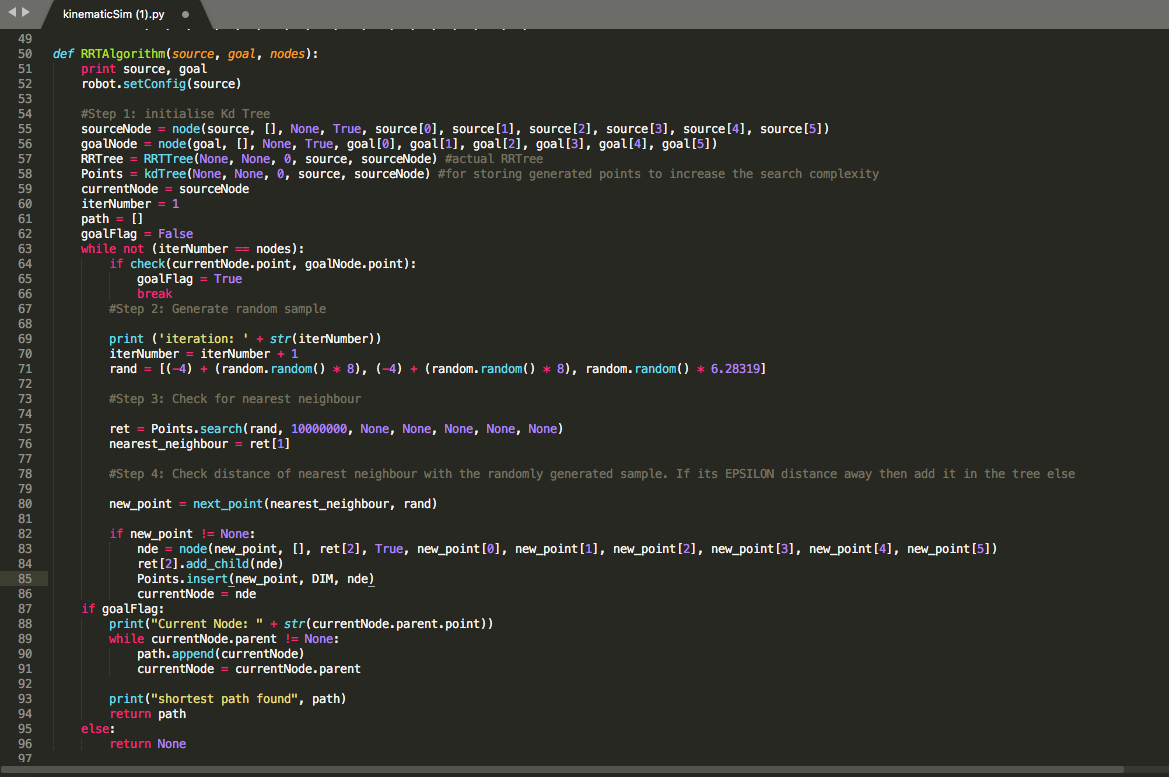
After making changes to the simpleWorld.xml file to create a robot object in the kinematicSim.py file and set its configurations.

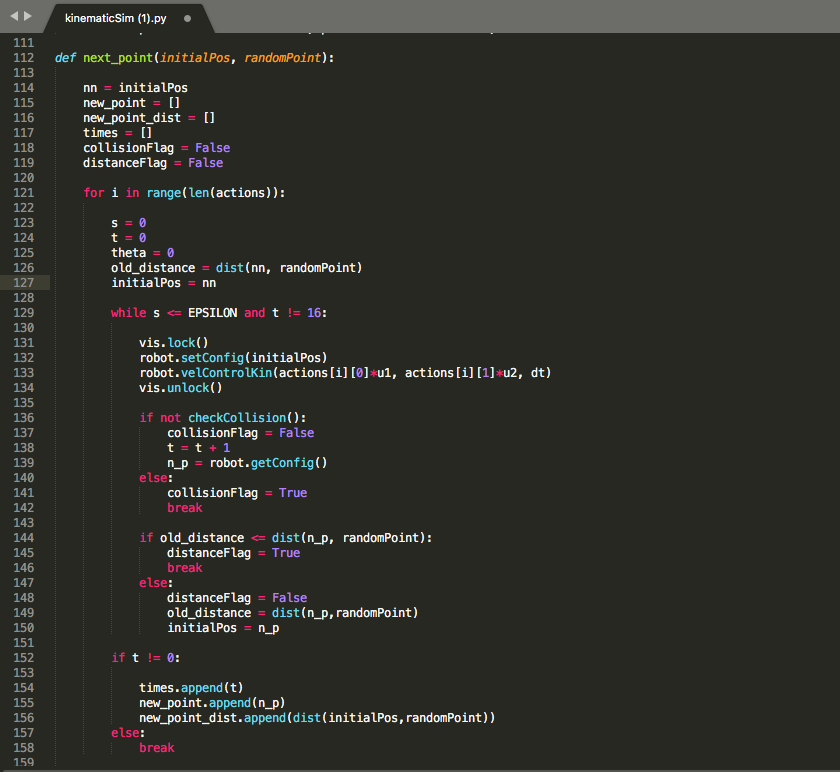


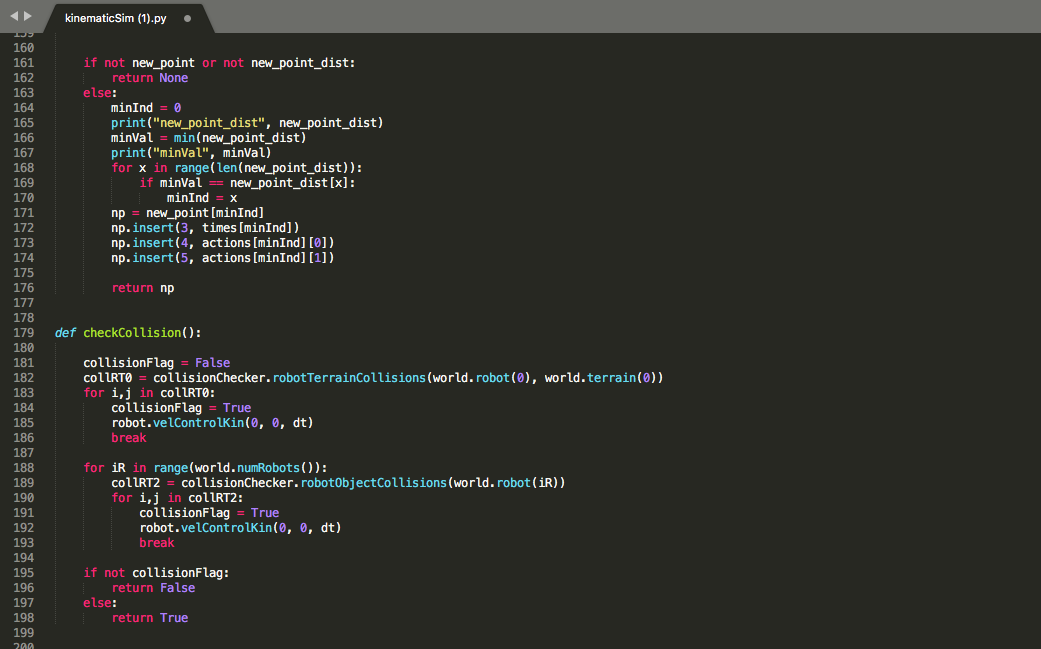
1. Set source and goal configuration for the algorithm and check if the lie in the Obstacle space. If both the source and goal are out of the obstacle space then call the RRTAlgorithm() method.



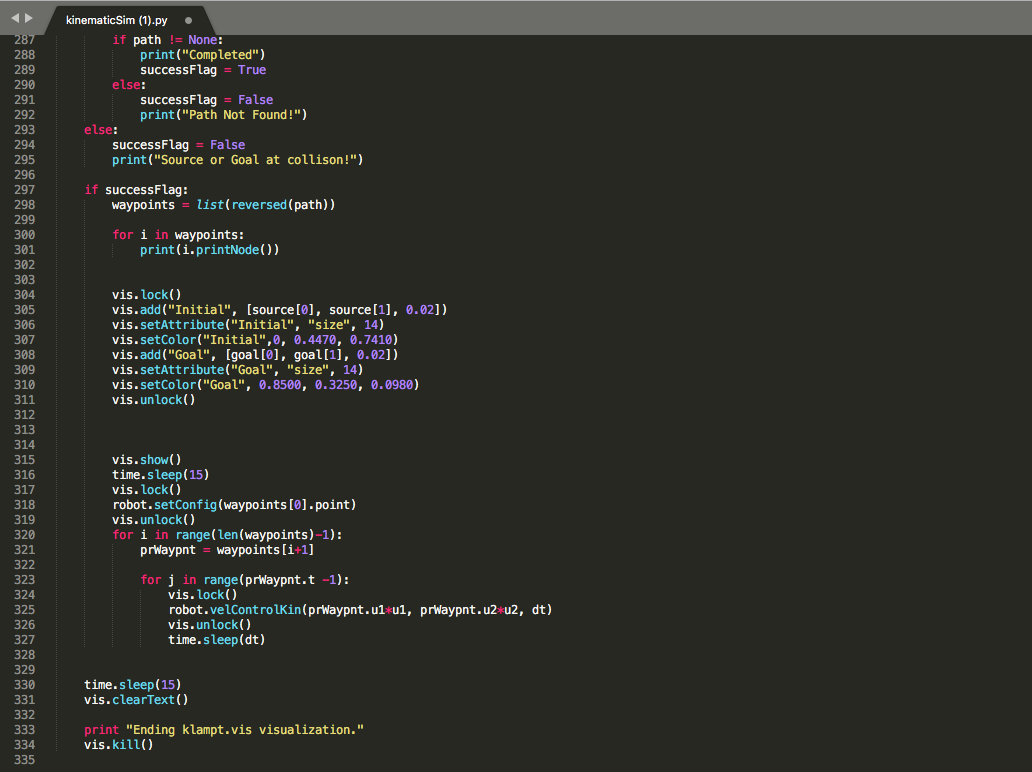
1. Following are the steps followed for RRT Algorithm:
2. First, initialize the KD Tree with the source node. At the same time initialize the RRT Tree with the source node. We will use the RRT Tree to trace back from goal node to source node to find the path between the two points.
3. Now check if the source is equal to the goal configuration, if true, return to the main function.
4. If source if not equal to goal and the number of iteration has not surpassed the number of nodes then generate a random sample.
5. Look for the nearest neighbor in the KD tree for the generated sample.
6. From the nearest neighbor, we will use these 9 combinations of linear and angular velocity: [(1,1), (1, -1), (1,0), (-1,1), (-1, -1), (-1,0), (0,1), (0, -1), (0,0)], one-by-one to move EPSILON distance in the corresponding 9 directions. While moving in any of the directions we will check for collision, if collision is encountered we will stop a configuration just before collision. We will calculate the distance of the final configuration reached in all the 9 directions to the randomly generated sample and select the one with minimum distance as our new node. We will add this new node to the RRT Tree.
7. Using back-tracking (moving from goal to source node) in RRT Tree we will find the path from source to goal configuration.

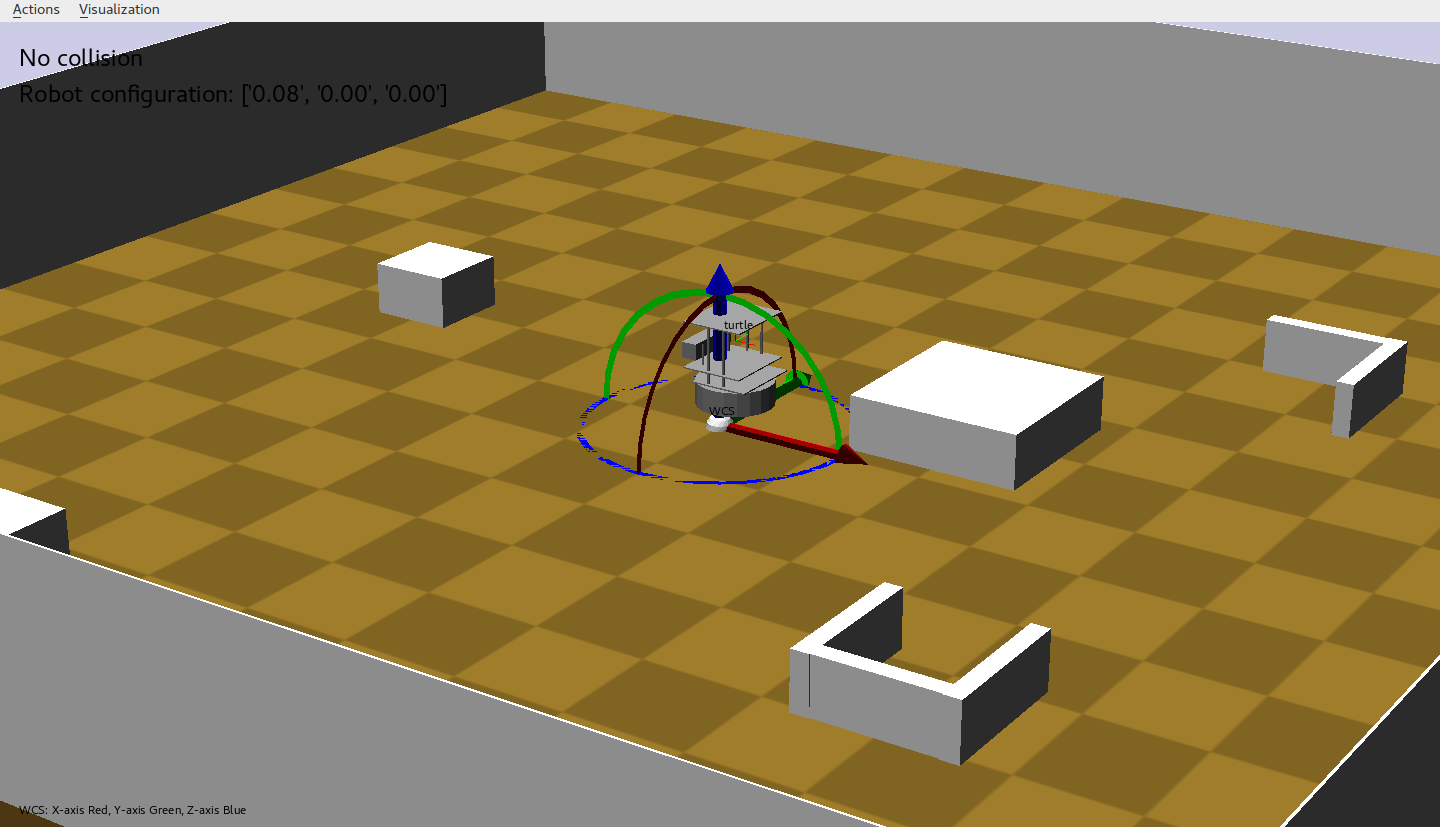






1. In the end, using the configurations (nodes) as waypoints move the robot to the goal position.





1. **ANALYSIS:**
2. While testing the algorithm different pairs of source and goal configurations were used. Also, the threshold for number of samples was changed for many of the test cases. If was found that with the initial threshold of 2000, the algorithm was able to find a path where the source and destination were not far apart. But for cases where the source and destination were far apart (let’s say source is at the center and destination is near one of the corners of the room) this many samples were not enough. Even by raising the number of samples to 6000 didn’t assured that the algorithm will be able to find a path. Because of this reason Bidirectional Search seems to be a better approach. With Bidirectional Search approach the samples generated are biased towards goal, which means that even with small number of samples the algorithm will be able to find a path in less time.
3. As can be seen in the demo video, the robot takes many unnecessary turns while moving towards the goal configuration. This is because the path consists of random configurations. To improve the movement of the robot path smoothing need to be implemented.
4. **CONCLUSION:**

There is no doubt that RRT is an efficient algorithm for planning path for robots with higher degree of freedom at the same time considering the differential constraints of the robot. The implemented RRT algorithm was able to generate path for different initial and final configurations. After analysis of the algorithm and tweaking the parameters, Bidirectional Search and path smoothing were found to be the two important improvement to the algorithm.