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.

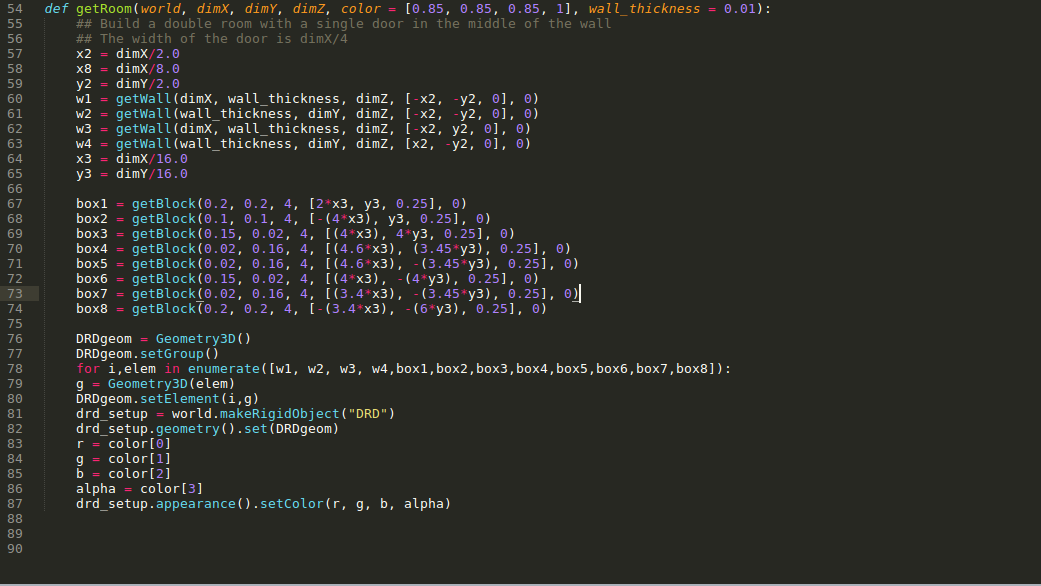
1. **IMPLEMENTATION STEPS:**

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:

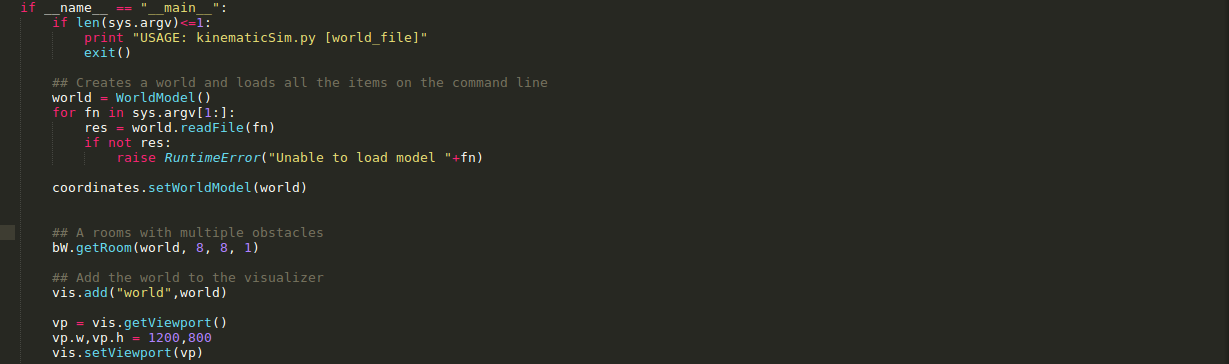
1. The Configuration of the robot at any point is considered in the format: (x, y, theta, t, u1, u2). Where t is the number of time steps required to reach the configuration. Each time step (dt) equals to 0.01 seconds. u1 is the linear velocity which is 0.5 m/sec and u2 is the angular velocity that is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

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.



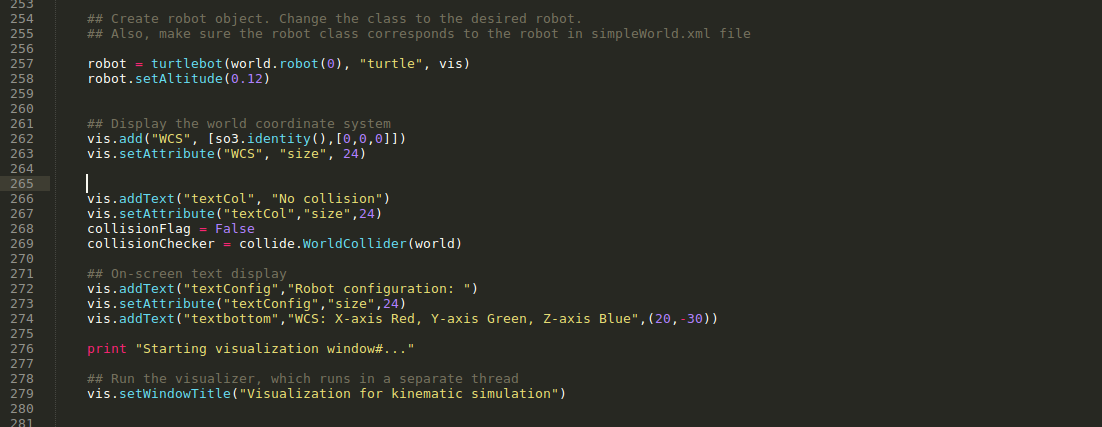
After making changes to the buildWorld.py file read this file in the kinematicSim.py file to create a world object.



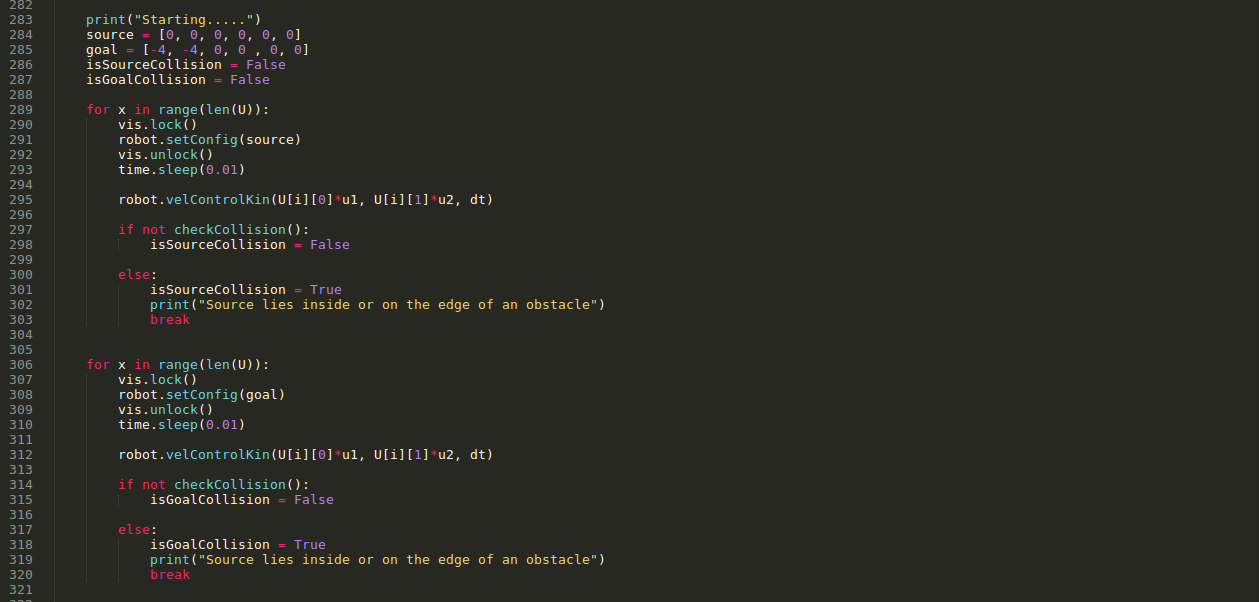
1. Make changes in the simpleWorld.xml file by adding the robot of your choice. For the project turtlebot was considered as the sample robot.



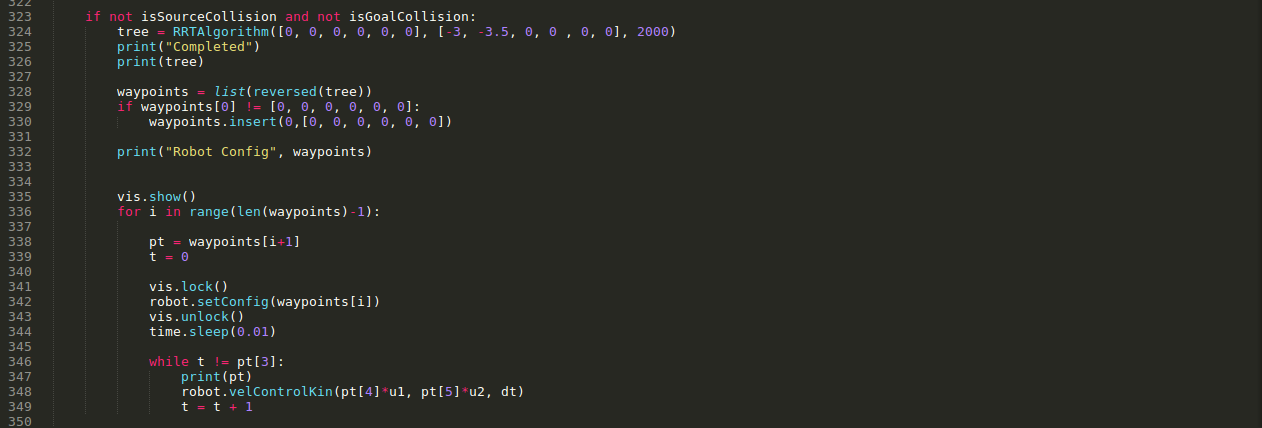
After making changes to the simpleWorld.xml file to create a robot object in the kinematicSim.py file and set its configurations.



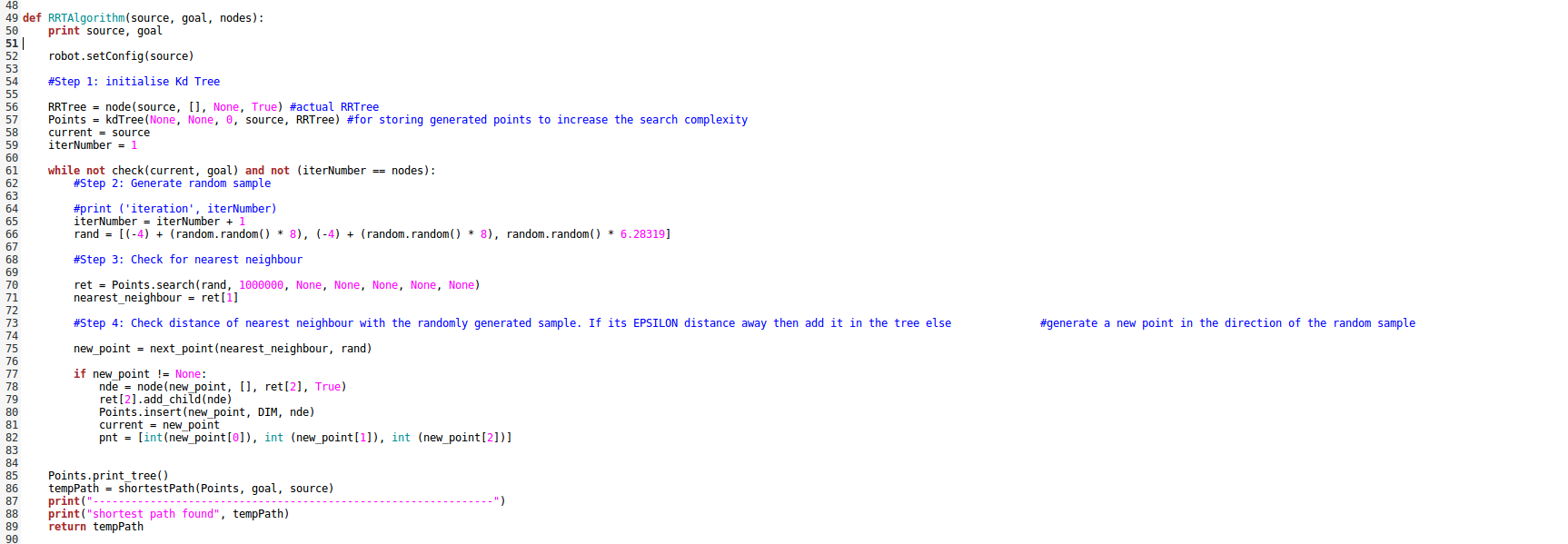
1. Set source and goal configuration for the algorithm and check if the lie in the Obstacle space.

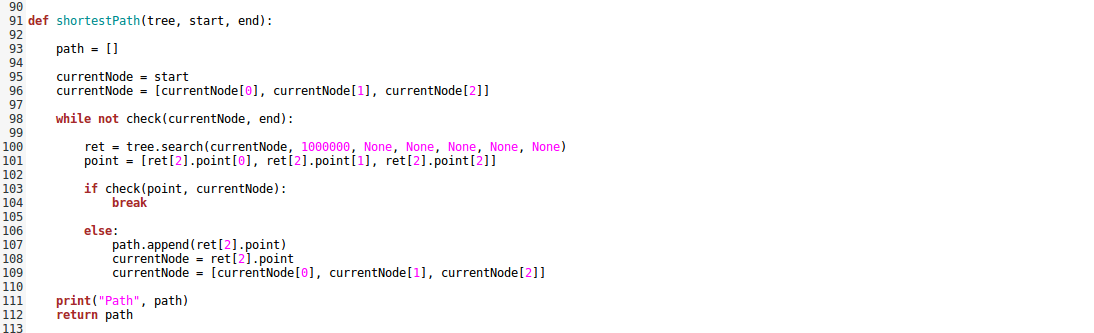


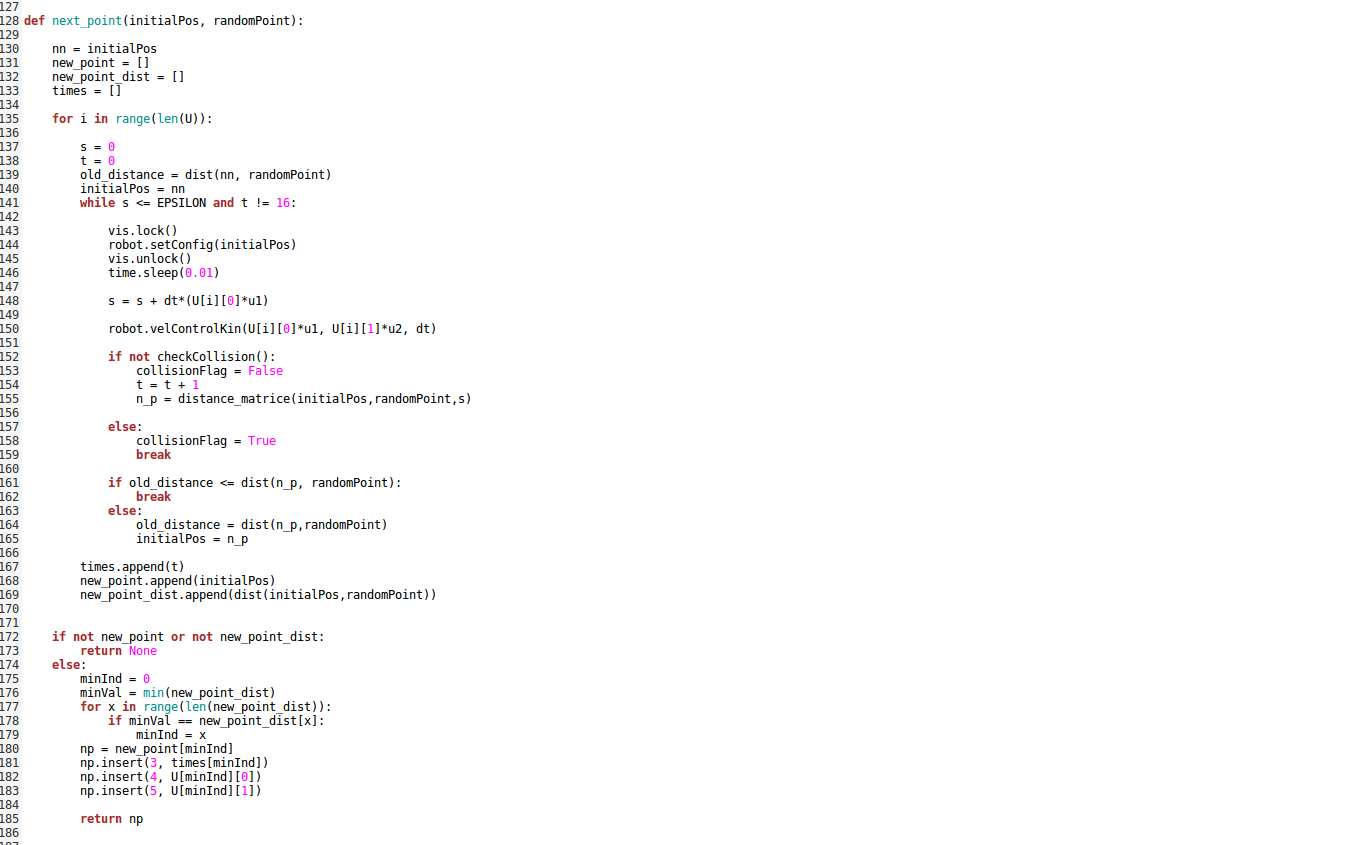
If both the source and goal are out of the obstacle space then call the RRTAlgorithm() method.

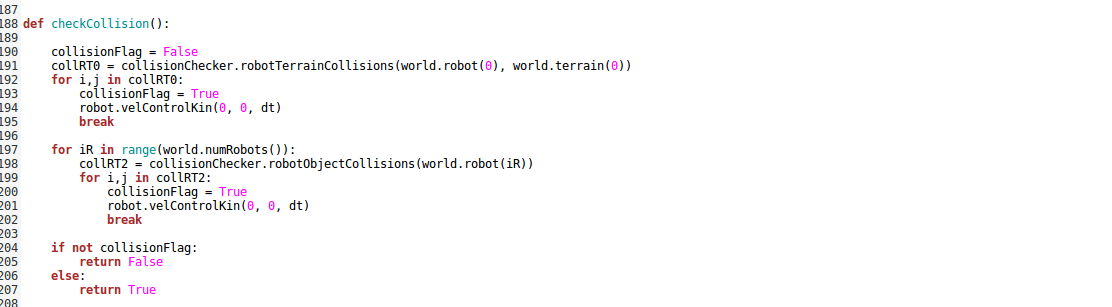


1. Following are the steps followed for RRT Algorithm:
2. First, initialize the KD Tree with the source node.
3. Now check if the source is equal to the goal configuration, if true, return to the main function.
4. If source if not equal to goal and the number of iteration has not surpassed the number of nodes then generate a random sample.
5. Look for the nearest neighbor in the tree for the generated sample.
6. Check the distance of the nearest neighbor from the random sample. If it is EPSILON distance away then at it to the tree else generate a new point in the direction of the random sample and add it to the tree.
7. While adding a new node to the tree make sure it’s not in the obstacle region.
8. Using back-tracking and nearest neighbor property find the shortest path from source to goal.

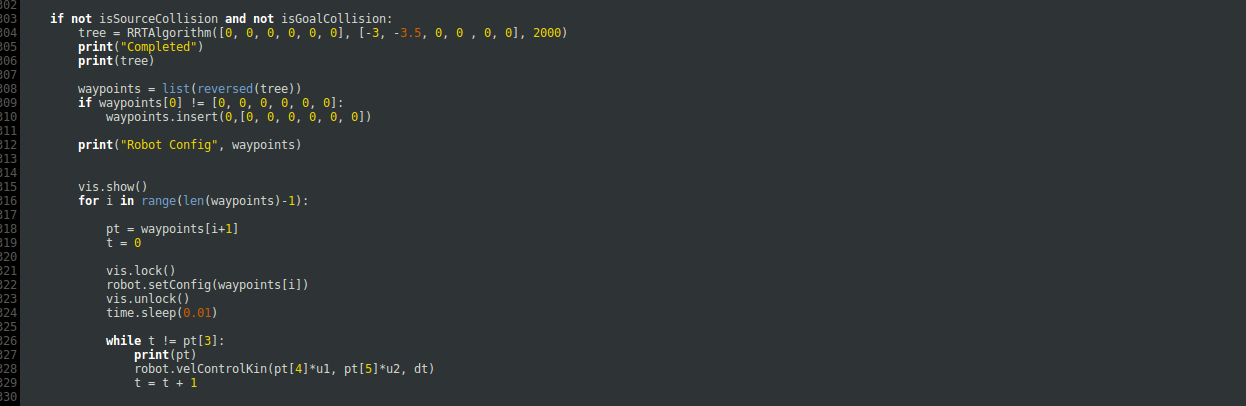


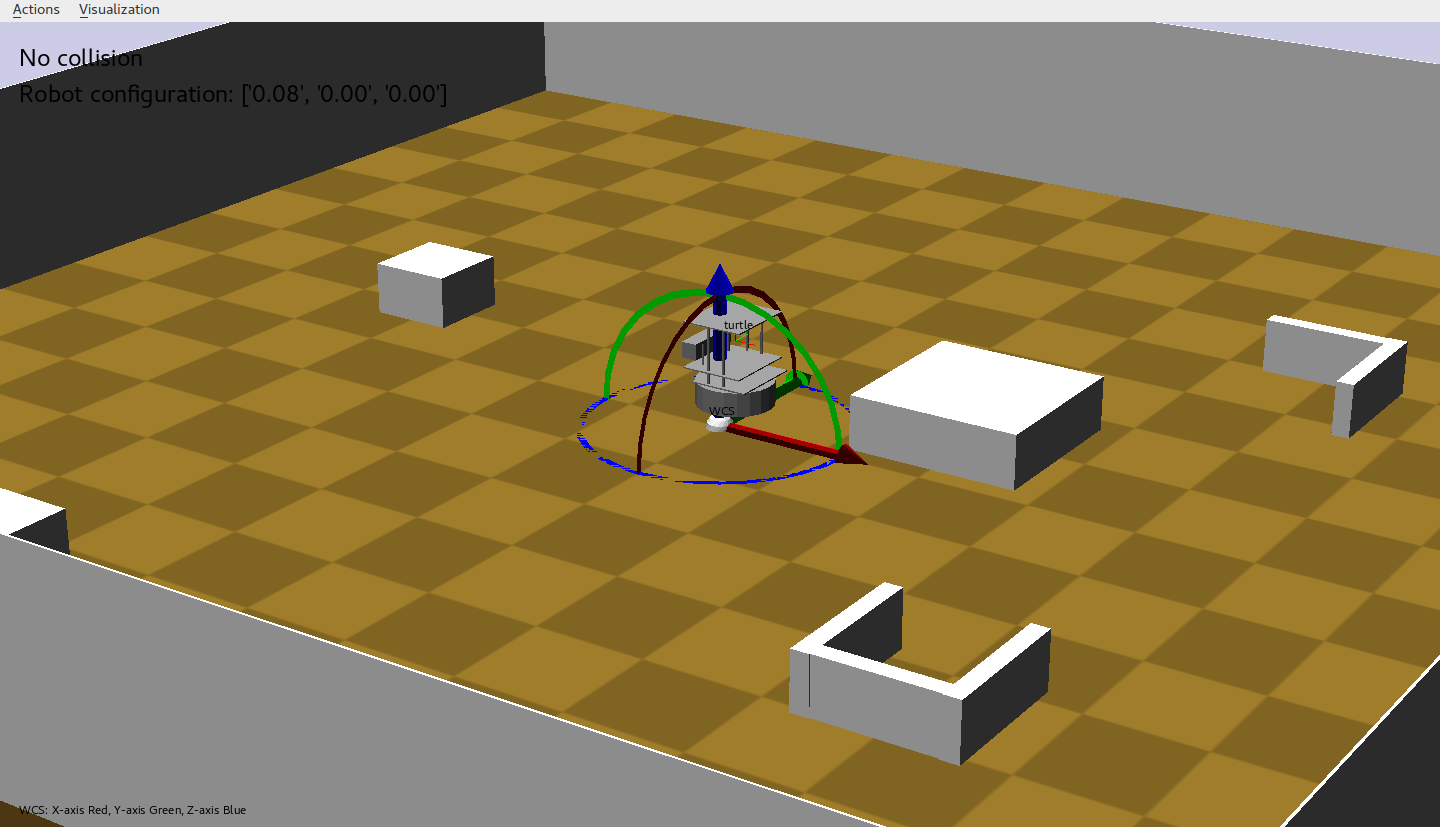






1. In the end, using the waypoints generated in the shortestPath() method set the configuration of the robot to the source and give inputs to move it to the goal position.





1. **TASK PROPOSED IN PROJECT PROPOSAL:**

Following tasks were proposed to be completed in the project proposal:

1. Definitely do: implement the RRT algorithm using Klamp’t on turtlebot or another car-like robot. Collision detection will be handled with Klamp’t.
2. Likely to do: implement RRT using bidirectional search approach to increase the efficiency of the algorithm.
3. Ideally do: implement the RRT algorithm and test on an actual robot.
4. **TASK COMPLETED:**

Out of the 3 tasks mentioned in the above section, the first one i.e. implementing RRT for a car-like (turtlebot) robot is completed in the proposal it was mentioned that the turtlebot can take 4 actions at any given configuration i.e. it can move forward, reverse, left and right. But while implementing the algorithm 9 possible actions were considered. 3 possible linear and angular velocities were considered, u1 = [1, -1, 0] and u2 = [1, -1, 0] and with the combination of these 9 possible actions were created.

Because of the difficulties and failures (mentioned below) encountered in the implementation process (b) and (c) tasks were not performed.

1. **DIFFICULTIES:**

The first and the most difficulty that was encountered was the lack of resources and tutorials on klamp’t. It’s a very sophisticated software with a lot of functionality and in-build libraries for possible every task but the biggest is the un-availability of tutorials and documentation. For example, apart from drawing a circle for the waypoints, there is no other way to show the trajectory the robot is following.

The second difficulty was finding the shortest path. The process that was chosen was to back-track the nearest neighbors from goal to the source configuration. This method, though works for most of the cases, at times do not produce any waypoints. One of the reasons for this is the distance we consider finding the nearest neighbor in the tree. If the distance is too large the algorithm will not be able to find any nearest neighbor for a point and return itself as nearest neighbor.

1. **ANALYSIS:**
2. An efficient cost function needs to be defined to build the KD Tree in order to generate waypoints at close intervals. If the waypoints and too far then the time to move from one waypoint to another will be higher and in that case the robot.velControlKin() function will abort making the robot stationary at one point.
3. Defining the distance matric is one of the most important aspects of the RRT algorithm. A distance metric is a simple metric on the state space based on the weighted Euclidean distance for the position, linear velocity and, angular velocity along with a weighted metric on the unit quaternion for orientation distance.
4. **CONCLUSION:**

The implemented RRT algorithm was able to create the waypoints but the turtlebot was not actually moving across the waypoints. After analysis of the algorithm and tweaking the parameter, it was concluded that there could be 3 possible reasons for such results. First, the distance considered to find the nearest neighbor is inaccurate which in turn results in less or no waypoint generation. Second, the distance matric is not correct which results generating inaccurate nodes for the KD Tree. And third, the klamp’t ‘s trajectory function is not used to generate the trajectory for the robot because of which the robot only gets a linear velocity as input.