Robot Automation Using ROS

Week 4 Assignment: Path Planning Algorithms

Path planning is an important path of a simulation development and environment as it traces out what direction the robot needs to follow to ensure a safe route from the start point to the end point in such a way that no harm is brought to the environment or the robot, as both could be very expensive to maintain. Keeping this in mind, many basic algorithms have been developed to help the machine understand where it should go and what it should do to keep away from such mishaps and ensure safety. Discussed below are 4 such path planning algorithms:

1. **Probabilistic Road Map:**
   PRM path planning algorithm is one of the simplest of algorithms for deciding a path between the start and end nodes of a particular environment. To create a roadmap using PRM, what we need to pass the number of nodes (vertices) to the function. The algorithm will randomly choose a point, that is not on an obstacle, and try to connect it to the nearest already existing node of the graph. The initial point is taken at random, and the consequent points are chosen in way that the graph stretches to all regions of the environment, not colliding with any obstacles. If suppose a new point chosen can be connected to multiple nodes (vertices) of the already existing graph, the algorithm chooses the node which has the least distance, and connects the new point to that node. This process repeats till N nodes are formed, and a graph connects all the nodes. What is to be done next is just input the starting vertex and the end point and the algorithm traverses through the graph, choosing the Euclidean distance (shortest possible path) from the start point to the end point.

2. **Rapidly- expanding Random Tree:**
   This path planning algorithm is similar to the PRM path planning algorithm, except that instead of a graph, the vertices form a tree. Each vertex can have atmost 1 parent. The initial point of the tree is chosen, and the next point is selected in such a way that it does not overlap with any of the

obstacles in the environment. Next, the algorithm tries to find the shortest path from the new point to the already existing tree. If a path is found where no obstacles are overlapped, the point is connected to the tree to its nearest neighbouring node. If such a path is not found, the algorithm discards that point and randomly chooses a new point. The reason it's called Rapidly-expanding Random Tree is because the points of the environment are chosen at random and the tree is connected immediately, expanding in all directions from the initial point. This is one of the easier to code algorithms and is computationally fast, however a lot of points may be connected in this tree. The path is expanded from the initial point till the end goal point is not reached. The algorithm just picks out the nearest points of the tree that connect to the start and end points and trace out the path between the two points. This point is chosen in such a way that the distance between the new point and the already existing point is greater than a parameter 'delta'. It is good for speed computing of the path but it does not return back the shortest or the quickest route between the two points, so keeping it running for more time may give us another answer which may or may not be faster than the already given path. This is where RRT* has an upper hand over RRT.

3. **Rapidly- expanding Random Tree *:**
   Being an extended version of the RRT path planning algorithm, the initial steps of the algorithm work in the same way as the RRT algorithm. However it defers in the computing path step, where the node connects to the nearest neighbouring point, but it goes back in the tree and tries to find a shorter path to the new point. If such a point is found, it chooses that point, or else continues the current path as a normal RRT algorithm. The major drawback of this algorithm is that the initial node (ROOT) has a lot of subtrees and can get confusing when trying to find shortest path. Since it returns the shortest path, the time complexity is more than that of RRT, however it returns the shortest path possible between the start and the end point correctly. It takes into account the 'cost' of each edge, and whether or not a path can be formed with a smaller cost. So in a general scenario, the RRT* would take a lot more time than RRT, but return the most feasible path from the start to the end point. RRT* has a very

important property of ‘re-connecting’ the tree, if a shorter distance is found. For example, if there are 3 points A, B and C, where B and C are connected, and a new random point D comes between A and B, the algorithm will first connect D to A or B, whichever has the shorter distance. Suppose it connects to A, it then compares the distance of D to B and B to C. If DB is shorter than BC, the algorithm removes connection BC and connects DB instead. In this way, the algorithm changes the tree too.

4. **Rapidly-expanding Random Graph:**
   As the name suggests, this type of algorithm is similar to the RRT and the RRT* algorithms, except that it creates a graph instead of a tree. The working of this algorithm is similar to RRT, where the new node is connected to multiple nodes, whichever is the nearest to it. The algorithm expands in all directions, covering the entire environment. RRT* is like a tree implementation of RRG. Which means that even RRG has the feature of ‘re-connecting’ the graph, as and when shorter paths are found. Due to this reason, the computation time of the algorithm increases, but it returns back the shortest distance between the start point and the end goal point. Thus this algorithm creates a completely interconnected graph in the environment, avoiding the obstacles, whenever they arise. It keeps in mind that the size of the edg should not be less than ‘delta’, to prevent overcrowding of nodes and the graph.

Conclusion:

RRT, RRT* and RRG are similar path planning algorithms, and as mentioned in the individual topics, have their own advantages and disadvantages. PRM and RRT are the simplest form of path planning algorithms, and others are generally adapted versions of these algorithms. These algorithms are used indefinitely in robotic path planning for general movement and traversing to robotic arm simulation, where the robots don’t have a free environment but have obstacles or conditions that they need to follow/satisfy. All these codes can be implemented in Python and in C++, making them easier to test in ROS and other development and simulation platforms.