Analysis of Path Planning Algorithms

Behrooz Bajestani
CCIS Department, NEU
Boston, USA
moradi.bajestani@husky.neu.edu

Bishwarup Neogy
CCIS Department, NEU
Boston, USA
neogy.b@husky.neu.edu

Abstract

Path planning and tracking is one of the most popular and crucial areas of research in mobile robotics. Many methods have been proposed and used in different applications. However, all suggested methods have their advantages and drawbacks and are limited to specific set of constraints. This project aims to study the performance of various path planning and obstacle avoidance methods. The comparison is performed against success rate, accuracy and cost effectiveness of the planned path. The planning is done for various closed environments simulated using Stage/Player simulation tool. The path tracking and motion toward goal is also simulated using a turtle-bot-like robot in the Stage/Player simulation tool. Finally, the paths from start to goal obtained from various methods within different environments are compared with each other and the results regarding the most efficient and complete path planning methods are discussed.

Introduction

Path finding and safe navigation toward goal is considered as one the fundamental issues in mobile robotics which is getting more significant with the rise of robotics applications in indoor and complex environments. Realization of human-robot collaboration is impossible unless collision free navigation of robots in environments designed for humans is achieved where robots are expected to encounter with a myriad of obstacles and complex paths e.g. objects, stairs, ramps, narrow hallways, etc.

In this paper, various path planning and path tracking methods are applied for collision free navigation of a simulated turtle-bot-like bot through various closed environments. Five methods, namely: Dijkstra shortest path, A*, Probabilistic Roadmap (PRM), Rapidly-exploring Random Tree (RRT) and Dynamic Window Approach (DWA) are evaluated. The results obtained from these methods are compared against each other with respect to their success rate and efficiency of the obtained path. All evaluated path planning algorithms are developed and executed in Python.

Obstacle avoidance and movement of robot using commands generated by motion control algorithms are simulated and observed within the Stage/Player simulation tool. The Stage/Player enables research in robot and sensor systems and one of the most common robot simulation and control interfaces across academia. The motion control algorithms are also developed using Python synced with Stage/Player simulation environment.

Environments

Four environments have been constructed and used for evaluating performance of path planning algorithms:

- Maze-like environment
- U-shaped environment
- Narrow path environment
- Blocked environment

Figure 1 below show the top view of these environments.

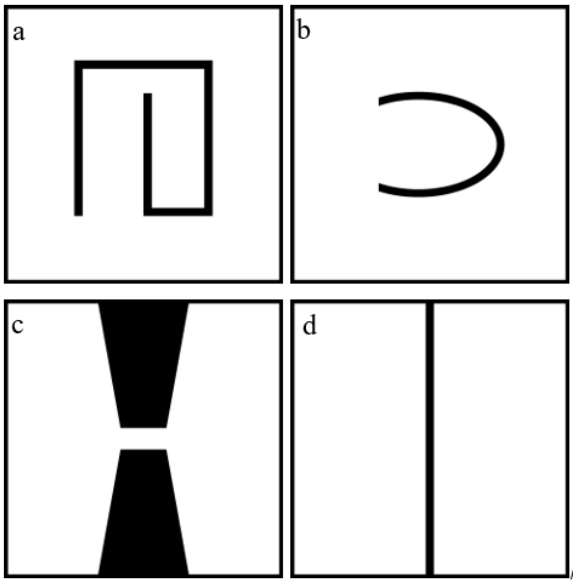

Figure 1. Top view of the environments a) Maze-like (World 1) b) U-shaped (World 2) c) Narrow path (World 3) d) Blocked (World 4)

All environments are closed, square with an edge size of 760 pixels. They are also simulated by the Stage/Player simulation tool which is a 2.5D environment that is the environments are imported as 2D black and white images and the dark shaded pixels of images are extruded vertically. Figure 2 shows a simulated version of environments generated by Stage/Player.

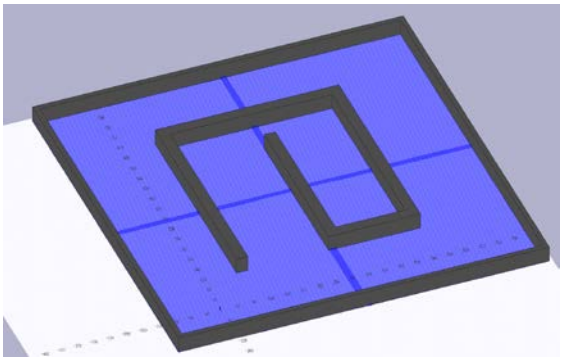

Figure 2. Simulated version of environments generated by Stage/Player simulation toll.

The 2D drawing of images are also imported to Python and converted into a binary matrix where each pixel is represented by a cell set to a value of 1 if the corresponding pixel of the image is occupied and 0 if there are no obstacles. Converting images to binary matrices with real numbers facilitate the process of analyzing the environments and planning path from start to goal. Upon converting the images to binary matrices, the obstacles are expanded vertically and horizontally as much as the robot radius and the cells within the radius are set to 0.5. Therefore, the free configuration space is achieved as all cells with values of exactly zero. Figure 3 shows the resulting environments where, the white, orange and black cells are obstacles, obstacle paddings and free configuration space (C_{free}), respectively.

Algorithms

Five path planning algorithms developed and evaluated are further explained below.

Dijkstra Shortest Path: Dijkstra's algorithm is an uninformed search algorithm based on breadth first search (BFS) principle (Mehlhorn and Sanders, 2008). The algorithm starts from the source node and proceeds to evaluate all adjacent nodes till the goal is reached. The order of visiting new nodes is that the closer the node to the source node the sooner it will be visited. Therefore, the Dijkstra algorithm is complete, meaning, it guarantees to find the shortest path, if one exists. However, it is exhaustive and in case of a dense and large environment with many nodes, the process of finding the shortest path can be time consuming requiring significant processing power. This algorithm outputs the order of nodes/coordinates representing the shortest possible collision-free path from start to goal. The output is not in the form of robot control commands and shall be transformed to control commands for the robot to traverse through the nodes toward the goal.

A*: A* is an informed search algorithm, and a modified version of BFS search algorithm where the order of the nodes to be expanded are determined by the distance from the start node as well as a manually defined heuristic function (Delling et. al., 2009). Given a starting node, A* will find a path to pre-determined goal node while visiting new nodes with minimal cost (least distance travelled) and heuristic value which is supposed to be an estimate of the cost required to reach the goal. The nodes expanded within a tree data structure with the start node as the root and extending paths to adjacent nodes, one edge at a time, until the goal is reached. In case of this study, the heuristic is defined to be the Euclidean distance to the goal coordinates. Similar to the Dijkstra algorithm, A* outputs an ordered set of nodes/ coordinates as the shortest collision-free possible path.

Also, in case of A* method, to make the whole process faster and more time efficient, the A* algorithm is equipped with Uniform Approximate Cell Decomposition method where the pixels are bundled together to generate bigger nodes. The value of nodes is set to 1 if it includes at least one occupied pixel (obstacle pixel) and to zero otherwise. As a benefit, a smaller number of nodes are explored by the algorithm and therefore, the goal is reached faster; however, the coarser the nodes the less accurate path will planned (comparing to the one obtained from Dijkstra algorithm). The imported environments are subdivided into coarse nodes initially and if no path was found, finer decomposition is tried. The process of subdividing the environment to finer nodes is continued until either a path is found or the minimum node size (each pixel one node) is reached. Figure 3 shows trials of A* algorithms with cell decompositions sizes manually set to various values.

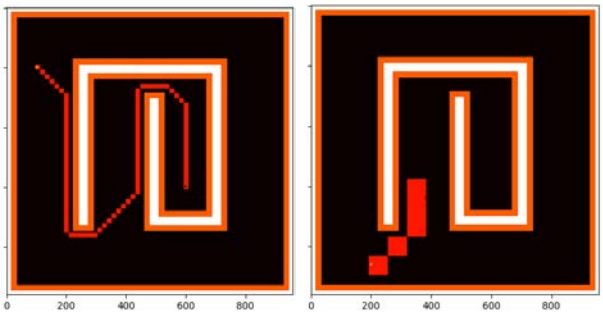

Figure 3. Trials of A algorithms with cell decompositions set to 16 (left) and 64 (right) pixels per cell*

Probabilistic Roadmap (PRM): PRM is one of the popular path planning algorithms used in robotics. It is efficient, relatively easy to implement and applicable for different types of motion planning problems. Here the PRM method was implemented to plan a path form the starting coordinates to the goal coordinates in different maze environments.

The PRM approach is normally formulated in terms of the configuration space C, which represents the entire environment including obstacles and the spaces where the agent can move freely. Each obstacle in the environment transforms to an obstacle in the configuration space. Let us represent the set of obstacles as well as obstacle paddings as, C_{obs}. A path for the moving object corresponds to a curve in the configuration space connecting the start and the goal configuration. A path is collision-free if it does not intersect C_{obs}. The probabilistic roadmap planner does random sampling in the free configuration space, C_{free}. These samples in the free space form the nodes of a graph called the *Roadmap* (see Figure 4).

Algorithm 1 CONSTRUCTROADMAP

Let: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
1: **loop**
2: $c \leftarrow$ a (useful) configuration in $\mathcal{C}_{\text{free}}$
3: $V \leftarrow V \cup \{c\}$
4: $N_c \leftarrow$ a set of (useful) nodes chosen from V
5: **for all** $c' \in N_c$, in order of increasing distance from c **do**
6: **if** c' and c are not connected in G **then**
7: **if** the local planner finds a path between c' and c **then**
8: add the edge $c'c$ to E

Figure 4. Algorithm to construct the Road Map for PRM (Geraerts, 2004)

After randomly sampling some points in the free configuration space, each point was taken and connected to k=10 nearest neighbors with in a predetermined distance, r. To efficiently find the k-nearest neighbors a KD-Tree of all the sampled points was constructed and a search was performed on the tree for *k* nearest neighbors (Yershova and LaValle, 2008). This process is continued until this graph represents a connectivity in C_{free}, and can then be user to answer motion planning queries. In this implementation, Dijkstra's path search algorithm was employed to find the shortest path between the start and the goal within the constructed Roadmap.

One of the challenging aspects of implementing PRM is to check for collisions while connecting the samples to construct the roadmap. On option was to use the KD-Tree approach to check if the sampled neighbors lied within any of the obstacles or not. But since the environment had already been preprocessed and the obstacle nodes had been marked with a value greater that zero, only a simple check was performed: if the neighbor was part of an obstacle, value > 0 otherwise it is in C_{free}.

Rapidly-exploring Random Tree (RRT): RRT is another path planning algorithm designed to efficiently search for paths by randomly building a tree like structure with many branches. RRT constructs a tree using random sampling in the free configuration space denoted by C_{free} (Noreen et. al. 2008). The root of the tree is the starting coordinate in the maze. The idea is to keep expanding branches iteratively starting form this point until we reach our goal node.

Let us consider a node *randNode* which is selected from C_{free} during some intermediate iteration. If *randNode* lies in C_{free} which is the obstacle free space, then a nearest neighbor node, *nearestNode* is searched in the tree confined within a pre-determined radius, *r*. If *nearestNode* can be accessed from *randNode* according to the predefined step size, then the tree can be expanded by connecting *randNode* and *nearestNode* as shown in Figure 5. If *nearestNode* is not reachable, another node is sampled within the predefined radius.

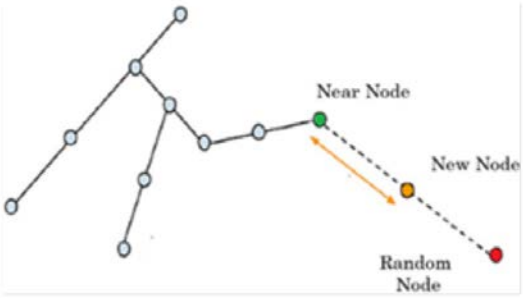

Figure 5: RRT Tree expansion process (Noreen et. al. 2008)

But so far, only a random tree within the free space is created. Although this tree starts at the start node, it will probably fail to find the goal as it has no sense of direction. To provide a sense of direction to the algorithm a steering function needs to be introduced. This function will slowly steer the path towards the goal. This is done by controlling the direction in which the nodes are sampled. Nodes are sampled in random directions with a probability *P* and while the goal node is sampled with a probability of *1 - P* for choosing the direction. In this implementation *P* was chosen to be 0.95. This ensures that the algorithm picks the goal node for direction at least 5% of the time while randomly picking samples 95% of the time. This ensures that the algorithm will eventually converge and find a path to the goal node.

Dynamic Window Approach (DWA): The Dynamic Window Approach is an online path planning, tracking strategy for collision-free navigation of mobile robots (Fox et. al. ,1997). Unlike previously discussed algorithm, where the output is set of coordinates representing the path, the dynamic window approach is derived directly from the dynamics and motion control of the robot. Also, dynamic window is a local approach meaning it use only a fraction

of surrounding environmental information, to plan path, avoid collision and generate robot control. Whereas, previous methods are global which assume that robot's configuration space is completely available and constant throughout the motion task. Therefore, DWA is suitable for unknown environments or environments with dynamic obstacles spaces.

DWA is performed by firstly populating a valid and admissible translational and angular velocities search space, and secondly choosing the most optimal pair within the search space as the robot motion control. The optimality is evaluated based on heading toward goal, preventing obstacles and maximizing transitional speed.

Stage/Player Simulation

The results obtained from path planning algorithms are transformed to motion control commands readable by Stage/Player simulation tool and the robot movement toward the goal while avoidance obstacle is simulated and recorded for presentation. The simulation robot had a height and radius of 20 units. The motion commands were transferred to robot as transitional and angular speeds with maximums of 15 and 1 units and radians per seconds respectively.

Results and Discussion

To compare the algorithms in an unbiased manner, the start and goal coordinates for each of the worlds are fixed. The start and goal coordinates were set to ensure a non-straightforward path. The obtained path from studied algorithms through various environments are shown in Figure 6

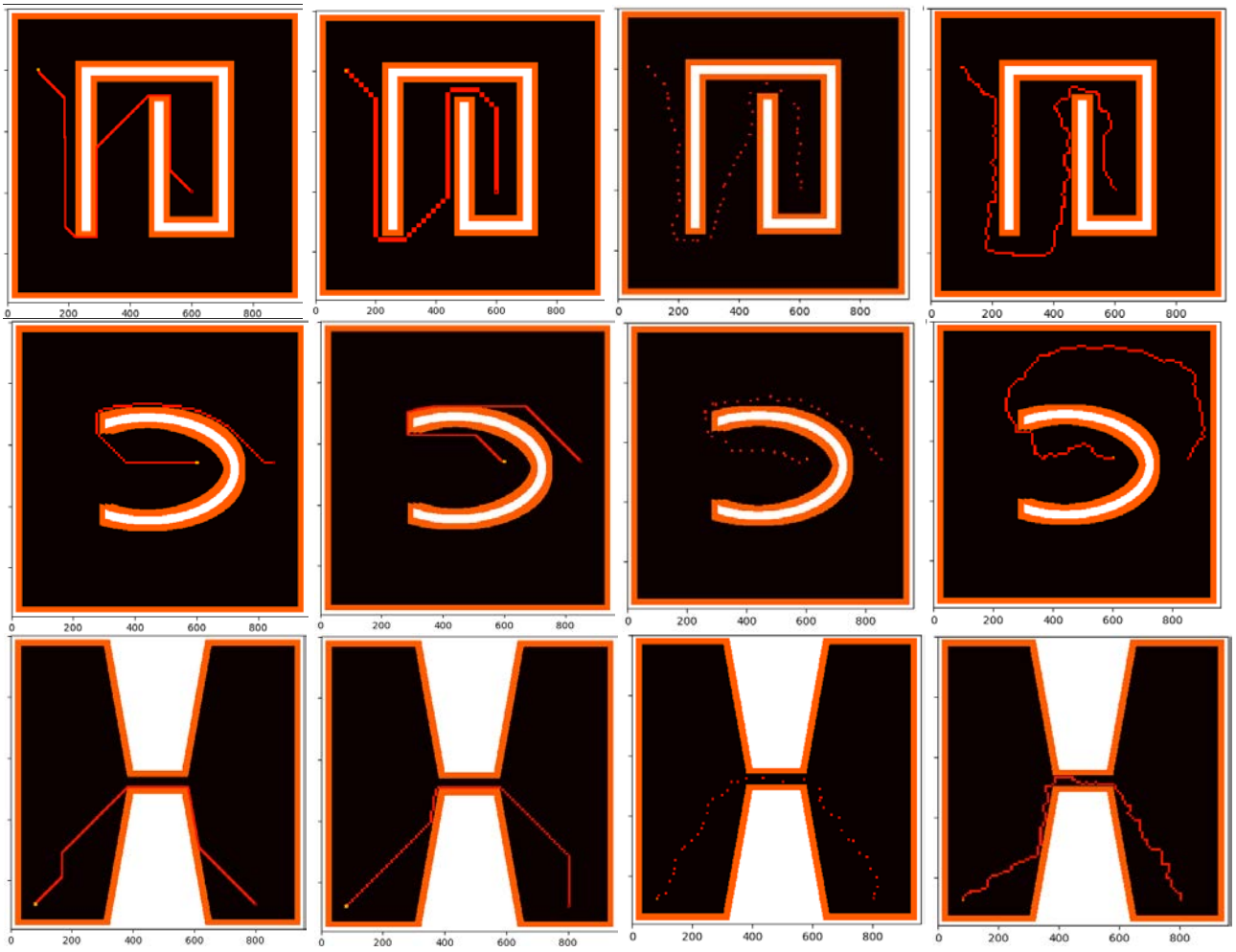

Figure 6. Path obtained from Dijkstra, A, PRM and RRT algorithms shown on first, second, third and fourth from left, respectively.*

According to Figure 6 while RRT is always successful in finding a path between the start and goal nodes, the paths that it calculates are inconsistent and longer than those found by the other algorithms. This erratic behavior arises from the random sampling being performed to calculate the branches while constructing the path.

PRM is another sample-based approach, and just like RRT, it also results in a different path with each execution. But according to Figure 7, PRM is better than RRT when it comes to length of the path. It consistently finds a path that is shorter and more efficient than RRT. This accuracy however comes with a small chance of failure. The probability that PRM will find a path is directly influenced by the number of samples. This behavior has been tested and plotted on Figure 8, which shows that as the number of samples increases, the failure rate of PRM decreases. PRM was tested with 2000 samples for 20 iterations of each environment which resulted in an overall success rate of 95%. Path obtained from PRM method using 500 and 3000 samples is shown in Figure 9.

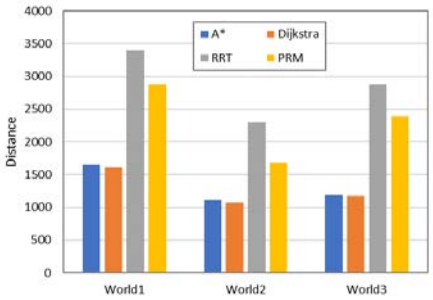

Figure 7. Path lengths obtained by various algorithms for World 1 (top), World 2 (middle) and World 3 (bottom)

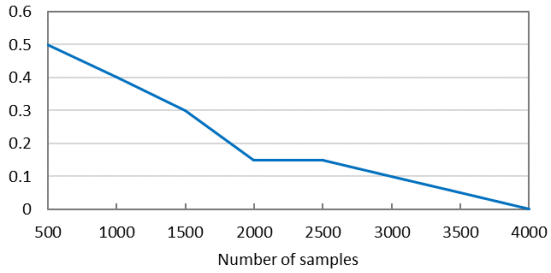

Figure 8. Percentage of failures vs number of samples for PRM

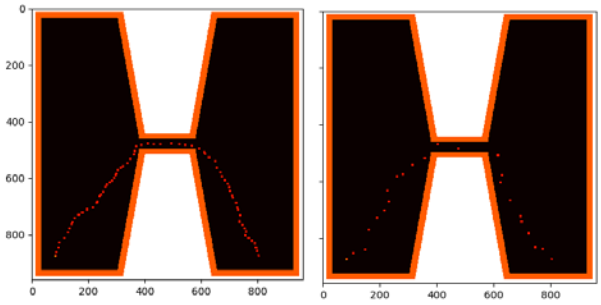

Figure 9: PRM using 3000 (left) and 500 (right) samples

Furthermore, Figure 10 summarizes performances of all studied algorithms with respect to length of the paths obtained for various environments. It is noticeable that the success rate of PRM was affected by environments that had narrower passages. That is in narrower regions of C_{free}, many of the random samples will lie in the C_{obs} region. Thus, there may arise a situation when the roadmap gets disconnected because in a given region, there are not enough samples in the free space (see Figure 9). In such a scenario, the PRM approach may fail to find a path between start and goal. This can be accommodated by increasing the number of samples, and thus increasing the probability of finding a neighbor node inside C_{free}.

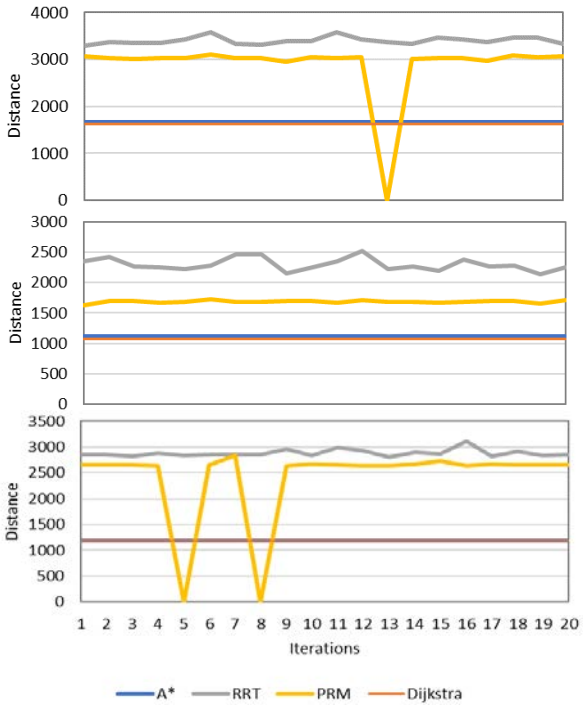

Figure 10. Path lengths obtained by various algorithms for World 1 (top), World 2 (middle) and World 3 (bottom)

Finally, all studied algorithms were executed in case of World 4 (blocked environment) where no feasible path existed. In all cases the algorithms successfully detected no viable paths from start to goal.

In case of DWA, the algorithm was successful only for simple scenarios where the position of start and goal nodes were not complex. Figure 11 shows the results of DWA for two different settings of start and goal nodes. Regarding complex scenarios, DWA is stuck in a local minimum and ceased to proceed towards the goal. Further analysis is required to completely implement this algorithm for passing all scenarios successfully.

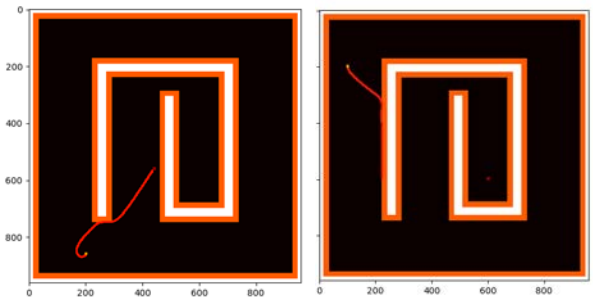

Figure 11. Results of DWA for two simple and complex settings of start and goal nodes.

Conclusion:

In this paper, various path planning and path tracking algorithms are applied for collision free navigation of a simulated turtle-bot-like bot through different closed environments. According to the obtained results, unlike PRM, A*, Dijkstra and RRT algorithms has excellent success rates, meaning a path would be found if it exists. However, the paths obtained from these algorithms are all different, with Dijkstra returning the shortest possible path and A* slightly longer. Sample-based algorithms e.g. RRT and PRM return longer paths. RRT converges faster by increasing the rate at which the goal is sampled; however, this would run the risk of getting stuck in a corner or a turn where the goal is just behind the obstacle, similar to the U-shaped maze. In short, RRT will always find the goal, provided a suitable sample rate, but it takes a longer path comparing to other algorithms. Compared to RRT, PRM was able to find a more economic path. For future studies, application of composable reinforcement learning for path planning and navigation through highly complex environments is suggested.

References

Geraerts, R. 2004. A Comparative Study of Probabilistic Roadmap Planners. Springer.

Yershova, A., LaValle, S. M. 2008. Improving Motion Planning Algorithms by Efficient Nearest-Neighbor Searching.

Noreen, I., Khan, A., Habib, Z. 2008. A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms

Fox, D.; Burgard, W.; Thrun, S. 1997. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*. 4(1): 23–33.

Mehlhorn, K., and Sanders, P. 2008. Chapter 10. Shortest Paths, Algorithms and Data Structures: The Basic Toolbox. Springer.

Delling, D., Sanders, P., Schultes, D., Wagner, D. 2009. Engineering Route Planning Algorithms. Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation. Lecture Notes in Computer Science. Springer. pp. 11