Path Planning

Ibrahim Beshr

6 September 2023

1 Introduction to Path Planning

Path planning is a non-deterministic polynomial-time hard problem with the task of finding a continuous path connecting a system from an initial to a final goal configuration ensuring that the robot moves smoothly while avoiding obstacles in a cluttered environment. This problem has attracted attention since the 1970s, and, in the years since, it has been used to solve problems across fields from simple spatial route planning to the selection of an appropriate action sequence that is required to reach a certain goal.

Path planning can be used in fully known or partially known environments, as well as in entirely unknown environments where information is received from system-mounted sensors and update environmental maps to inform the desired motion of the robot. As a result, path planning algorithms are differentiated based upon available environmental knowledge.

2 Types of Path Planning Algorithms

An effective path planning algorithm needs to satisfy four criteria:

1. The motion planning technique must be capable of always finding the optimal path in realistic static environments.
2. It must be expandable to dynamic environments.
3. It must remain compatible with and enhance the chosen self-referencing approach.
4. It must minimize the complexity, data storage, and computation time.

2.1 *Dijkstra Algorithm*

The Dijkstra algorithm works by solving sub-problems computing the shortest path from the source to vertices among the closest vertices to the source. It finds the next closest vertex by maintaining the new vertices in a priority-min queue and stores only one intermediate node so that only one shortest path can

be found. We can generalize that the Dijkstra algorithm is best suited for a static environment and global path planning as most of the data required are pre-defined for computation of shortest path.

Another improved Dijkstra algorithm reserves all nodes with the same distance from the source node as intermediate nodes, and then searches again from all the intermediate nodes until traversing successfully through to the goal node. Through iteration, all possible shortest paths are found and may then be evaluated. Moreover, the Floyd algorithm is another improved Dijkstra algorithm a popular graph algorithm for finding the shortest path in a positive or negative weighted graph, whereas Dijkstra works best for finding the single-source shortest path in a positive weighted graph.

2.2 *D* Algorithm*

Path planning in partially known and dynamic environments in an efficient manner is increasingly critical. To solve this problem, the D* algorithm is used to generate a collision-free path amidst moving obstacles. D* is an informed incremental search algorithm that repairs the cost map partially and the previously calculated cost map. D* is over 200-times faster than an optimal re-planner, but the main drawback of the D* algorithm is its high memory consumption when compared with other D* variants.

2.2.1 Applications

- Automated vehicles.
- Path planning of autonomous vehicles in cluttered environments, where the algorithm can reach a quick re-planning result when unexpected obstacles are encountered.

2.3 *A* Algorithm*

The A* Algorithm is a popular graph traversal path planning algorithm. A* operates similarly to Dijkstra's algorithm except that it guides its search towards the most promising states, potentially saving a significant amount of computation time. A* is the most widely used for approaching a near optimal solution with the available data-set. It is widely used in static environments; there are instances where this algorithm is used in dynamic environments. The base function can be tailored to a specific application or environment based on our needs.

2.3.1 Applications

- Gaming industry.
- Development of Artificial Intelligence.
- Robot path planning.

- Urban intelligent transportation
- Automatic control.

2.4 *Rapidly-Exploring Random Trees*

We have discussed algorithms like A*, which are static in nature and require a path specified to them upfront. Let us now discuss dynamic and online algorithms like RRT, which do not require a path to be specified upfront. Rather than that, they expand in all regions, and, based on weights assigned to each node, create a path from start to goal. RRT's were introduced to handle broad classes of path planning problems. RRT expands heavily in unexplored portions of the configuration space of the robot when compared to naive random trees, who tend to expand heavily in places that are already explored. Thus, we can say that RRT is biased to unexplored regions. The vertices of RRT follow a uniform distribution and the algorithm is relatively simple and RRTs always stay connected even though the number of edges is minimal.

2.4.1 Applications

- Any wheeled system.
- Robotic arms.

2.5 Genetic Algorithm

Discrete path planning algorithms, such as grid based algorithms and potential fields, require substantial CPU performance and require significant memory. In this section, we introduce genetic algorithms, which help to overcome such limitations. For example, genetic algorithms can be applied with the advantage that such algorithms cover a large search space and use a minimal memory and CPU resources. They are also able to adapt to changing environments. One disadvantage is that the solution found for the optimization problem may not always be a global minimum.

3 Local Planner and Global Planner

3.1 *Local Planner*

Local path planning is most typically performed in unknown or dynamic environments. It is performed while the robot is moving, taking data from local sensors. In this case, the robot has the ability to generate a new path in response to the changes within the environment. Obstacles, if any exist, may be static (when its position and orientation relative to a known fixed coordinate frame is invariant in time), or dynamic (when the position, orientation, or both change relative to the fixed coordinate frame).

3.2 *Global Planner*

Global path planning seeks an optimal path given largely complete environmental information and is best performed when the environment is static and perfectly known to the robot. In this case, the path planning algorithm produces a complete path from the start point to the final end point before the robot starts following the planned trajectory. Global motion planning is the high level control for environment traversal.

4 Conclusion

In this paper, we presented several widely used path planning algorithms and their variants, categorized based primarily upon whether the environment in which the robot operates is static or dynamic. In each section, we covered the methodology of all the algorithms as well as their advantages and disadvantages.

We conclude that, for a given map the algorithm A* is a strong-performing option for static environments because of the low memory usage, computational speed, less implementation complexity, and efficiency making it suitable also for use in embedded system deployment.

Also, it could be concluded that D* is one such algorithm that stands out as one of the best algorithms suited for a dynamic environment as this is shown to decrease the computational time significantly. This helps in the presence of dynamic obstacles where the reaction time is very important, particularly with faster moving dynamic obstacles.