HYBRID A* ALGORITHM

Ziad Essam

Introduction:

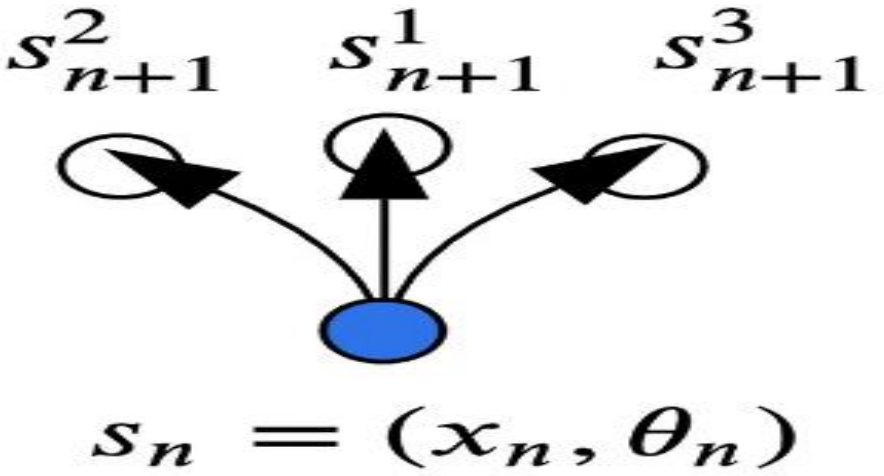
Hybrid A* is widely regarded as one of the most effective path-planning algorithms for car-like robots. Developed by Dmitri Dolgov, it has gained significant popularity in the field of robotics.

hybrid A star algorithm has gained popularity, and many developers start to use the algorithm in their work after its exceptional performance in the 2007 DARPA Urban Challenge, where robotic vehicles had to autonomously navigate parking lots, the algorithm was used by the Stanford Racing Teams robot, Junior, in the Urban Challenge and they demonstrated flawless performance in complex general path-planning tasks such as navigating parking lots and executing U-turns on blocked roads, with typical full cycle replanning times of 50–300ms.

The Hybrid A* algorithm helps self-driving cars, drones, and robots in hotels and restaurants move safely and efficiently. It combines smart path planning with smooth movement, allowing these machines to navigate around obstacles and find the best routes. Thanks to this algorithm, autonomous vehicles and robots can make quick decisions and travel through busy areas with ease.

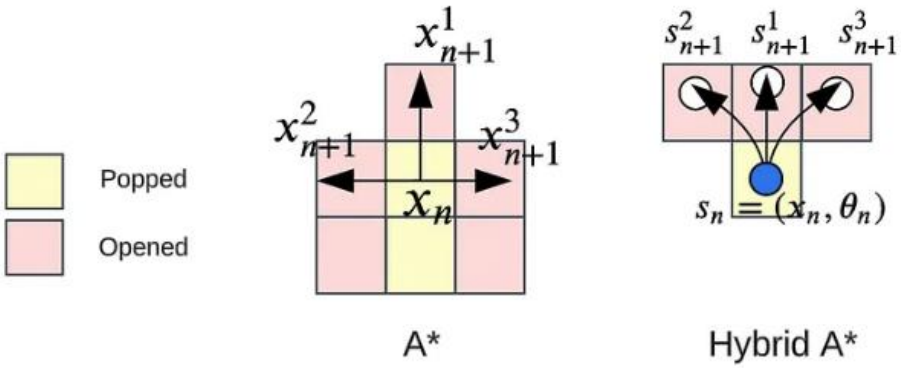
How hybrid A* algorithm work:

An improvement to the A* algorithm is hybrid A*. Hybrid A* searches for the safest and most effective route for a vehicle to travel between two points, taking the direction of travel into account, whereas A* looks for the shortest path between two points. The main difference lies in the fact that Hybrid A* determines the optimal path by considering the car's orientation, as well as its position.



- Moving to the next step (n+1) is done using one of three options: turning with the sharpest left or right turn or going straight. This means the shortest path between two points, can be calculated using a Dubins path. You can make the model more advanced by allowing the vehicle to move in reverse. This adds three more

movement options, giving a total of six possible directions.



- In regular A* when a state x_n in R2 (R2 refers to 2-dimensional Euclidean space) s expanded on a grid, it opens up neighboring cells by moving along the grid, and the states line up with the center of each cell.

  In hybrid A*, it's different. When the state sn=(xn,θn) moves, it opens new cells by simulating the vehicle's movement. As a result, the next state sn+1=(xn+1,θn+1) rarely lands exactly in the center of a cell (as shown by the white circles).

- Calculating the cost for each new state sn+1 is quite straightforward. The cost is based on the length of the path between sn and sn+1. However, we can add extra costs to discourage certain movements. For example, in the diagram(the previous bullet point), we can apply a

higher cost to turning movement (s^2n+1 and s^3n+1) compared to going straight (s^1n+1). Additionally, if reverse movements are allowed, they will also have a higher penalty to discourage moving backward unless necessary.

- In Hybrid A*, each state expands by simulating different possible movements, including turns and straight lines. As a result, the new state might not land exactly in the center of the grid cells, as shown by the circles in illustrations. The cost of moving between states is determined by the length of the movement arc, with additional penalties for movements that involve turning or reversing.
- When choosing the next state to expand, Hybrid A* uses a combination of the current cost and an estimate of the remaining distance to the goal. This estimate, or heuristic, can be calculated using either Dubins path or Reeds Shepp path, which are adapted for the robot's movement constraints. However, obstacles can affect these estimates, so alternative methods may be used to avoid inaccurate cost predictions.
- To account for obstacles, the algorithm can employ grid-based methods. For example, dynamic programming like Dijkstra's algorithm or A* can be used to calculate the shortest path on a 2D grid. This

approach helps refine the heuristic by incorporating obstacle information.

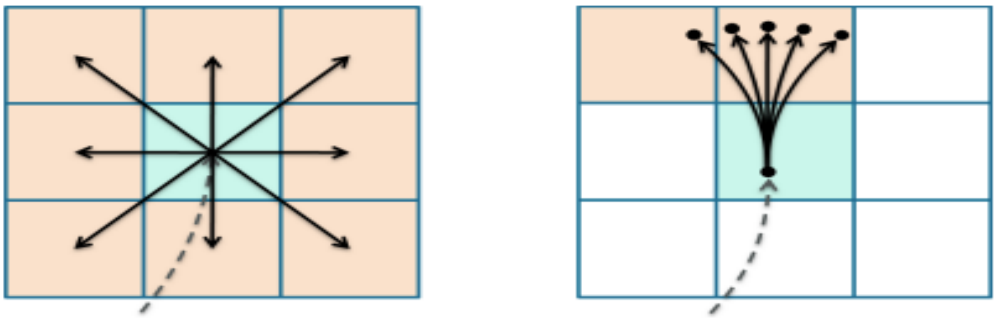
- Finally, Hybrid A* combines two heuristics: one for kinematics and one for obstacles. It uses the maximum value of these heuristics to select the most promising state for the next expansion. This process is managed by a priority queue, which ensures that states with the lowest combined cost and heuristic value are explored first.

Key Differences between A* and hybrid A*:

1. The regular A *algorithm works well in two-dimensional systems and has many applications in computer games. However, it faces limitations when applied to non-holonomic systems. Hybrid A* is an improved version of the regular A*, designed to be effective when applied to non-holonomic systems.

2. The conventional A* algorithm aims to find the shortest path from an initial position x_0 in 2D plane to a goal position xf in 2D plane. In contrast, Hybrid will try to find a sequence of safe poses from $s_0 = (x_0, \theta_0)$ in 2D plane with orientation to sf = (xf, θf) in 2D plane with orientation, minimizing the travel length. Consequently, our search domain becomes 2D plane with orientation.

3. In regular A*, states move from one cell to its neighboring cells, with each state being at the center of a cell. In Hybrid A*, states are based on the vehicle's movement and orientation, which means the new state often doesn't land exactly at the center of a cell. This approach allows Hybrid A* to handle more complex paths better than the standard A* method.



(a) regular A*

(b) Hybrid A*

Possible neighbors for a cell