A Combined A* Algorithm with Lattice Graphs that Guarantees Feasibility of the Motion

MinGyu Kim

Aerospace Engineering Department, Georgia Institute of Technology, Atlanta, GA, 30313

***Abstract*— In the simulation, our race car encountered a predicament while navigating a corner. Although the car initially charted a route toward its destination, it unexpectedly became wedged when making a turn. This issue arose from impractical movements generated by our code, prompting us to take action and enhance the A* search algorithm. Drawing inspiration from the lattice graph, a virtual pre-computed trajectory, we seamlessly integrated it into the original A* algorithm. As a result, the revised planner now calculates routes to the goal that are free from impractical maneuvers and guarantees efficient obstacle avoidance.**

I. INTRODUCTION

Finding viable paths in an unfamiliar environment can be a challenging task. Typically, in robotics, a robot is placed in a location, and the algorithm updates a static map within the robot system to compute routes to a specified destination. In our case, we selected the A* algorithm from a plethora of search algorithms to test the performance of a racing car in a virtual environment [1]. We meticulously configured and evaluated this algorithm within the Gazebo racing car simulation.

The car consistently succeeded in generating a path to the chosen destination. However, an issue surfaced when the car encountered specific scenarios, such as moving close to a wall or executing rapid turns near the wall. Despite generating an optimal path, the car found itself stuck at a corner, unable to progress. It required new directives to disengage from the wall, involving turning at a different location and plotting a fresh path.

To tackle this challenge, we propose integrating lattice graphs into the path-planning algorithm [2]. These pre-computed graphs guarantee the identification of paths that only involve feasible movements, ensuring that the generated paths are always executable. Moreover, the pre-computed trajectory substantially reduces computation time by exploring fewer potential states.

II. METHOD

The experiment took place within a 32 by 32 grid map populated with randomly placed square obstacles. Prior to the test, the start and destination positions were predetermined, and information regarding the obstacles was extracted from the static grid map. We introduced three new components to enhance the original spatial A* algorithm [1]. A detailed description of each component is provided in the following subsections.

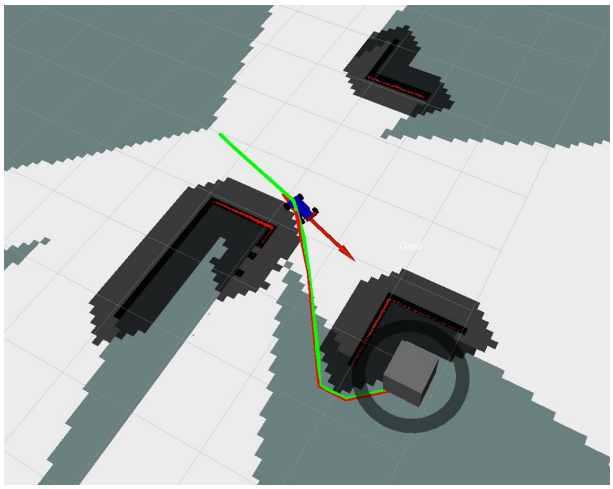


Fig. 1: A car in the Gazebo simulation is stuck at a corner, although the car successfully generated its trajectory to the designation (grey cube box).

Within the state nodes, a set of variables [x(s), y(s), $\theta(s), \kappa(s)$] representing arc length s was stored. Additionally, five distinct edges, generated as lattice graphs, were employed to explore the surrounding areas.

A. Generate a Lattice Graph

The lattice graphs were empirically generated using a methodology consistent with the reference [3]. The graph's curvature was computed using user-defined constants [p0, p1, p2, p3, and arc length s]. The planner exhibited various motion curves depending on the specific coefficients provided to the curvature function.

$$\kappa(s) = a + bs + cs^2 + ds^3$$

$$\kappa(s) = a(p) + b(p)s + c(p)s^2 + d(p)s^3$$

The user-defined coefficients were configured to be fractions of the curvature function. This choice was made to ensure continuity and meet boundary conditions, as demonstrated in the equations presented below.

$$\kappa(0) = p_0$$

$$\kappa(\frac{s}{3}) = p_1$$

$$\kappa(\frac{2s}{3}) = p_2$$

$$\kappa(s) = p_3$$

From that assumption, the coefficient values [a(p), b(p), c(p), d(p)] were derived using the following equations.

$$a(p) = p_0$$

$$b(p) = \frac{-11p_0 - 18p_2 + 9p_2 - 2p_3}{2s}$$

$$c(p) = \frac{9(2p_0 - 5p_1 + 4p_2 - p_3)}{2s^2}$$

$$d(p) = \frac{-9(p_0 - 3p_1 + 3p_2 - p_3)}{2s^3}$$

After calculating the coefficients, the state of a node was approximated. The orientation function at the node was obtained by taking derivatives of the previously obtained curvature function. The x and y positions were determined using the kinematic bicycle model, with the integral of the orientation function computed from the current position to a point along the arc length, yielding the x and y positions. Curve parameter set $p = [p_1, p_2, p_3, s]$

$$x_p(s) = \int_0^s cos(\theta_p(s))\, ds$$

$$y_p(s) = \int_0^s sin(\theta_p(s))\, ds$$

$$\theta_p(s) = a(p)s + \frac{b(p)s^2}{2} + \frac{c(p)s^3}{3} + \frac{d(p)s^4}{4}$$

$$\kappa_p(s) = a(p) + b(p)s + c(p)s^2 + d(p)s^4$$

B. Define a Discrete State Using Simpson's Rule

Nevertheless, obtaining discrete numerical values for the state information proved challenging when using a continuous method, such as integrating a function. To address this challenge, Simpson's Rule was introduced. This method generated discrete numerical values, and each node stored these values to identify potential locations for exploration.

$$\int_0^s f(x)\, dx$$

$$\approx \frac{\Delta x}{3}(f(x_0) + \sum^{n-1}(4f(x_{2n}) + 2f(x_{2n-1})) + f(x_{2n}))\Delta x$$

$$(\Delta x = \frac{s}{3})$$

Each state node was sampled by the lattice graph generator in a form of a state set of the discrete numbers.

C. Obstacle detector

Under the original conditions, it was challenging to assess whether the object might collide with an obstacle. The grid map's dot point represented an ambiguous state node position, necessitating the creation of a virtual boundary around the state node. Consequently, a function was employed to define the dimensions of a square, resembling the shape of a car, around a state point. Using this dimension information in conjunction with the obstacle data from the static map, the obstacle detection function determined whether the object would collide with the wall.

In this particular test, a square box measuring 0.5 by 0.5 was utilized, which represented half the size of the unit wall. All four vertices of this square (comprising four in total) were employed to detect obstacles.

III. Result and Discussion

Before conducting the test, it was essential to determine both the length of the edge and the search rate. The term "length of the edge" directly pertains to the extent of the motion trajectory generated, while the "search rate" signifies the number of intervals at which the trajectory paused in order to explore the surrounding regions. The experiment encompassed a total of nine sets, each featuring distinct combinations of arc length and search rate values, and the search algorithm exhibited varying behavior contingent on these variables.

In general, planners with longer edges were capable of finding their path more swiftly in comparison to those with shorter trajectories. Conversely, planners employing shorter edges generated a greater number of potential trajectories and boasted a larger control set size.

Arc length	size of control set	elapse time [sec]
1	65	15.46
2	22	9.96
3	9	9.05

TABLE I: The search performance results with the search rate = 1

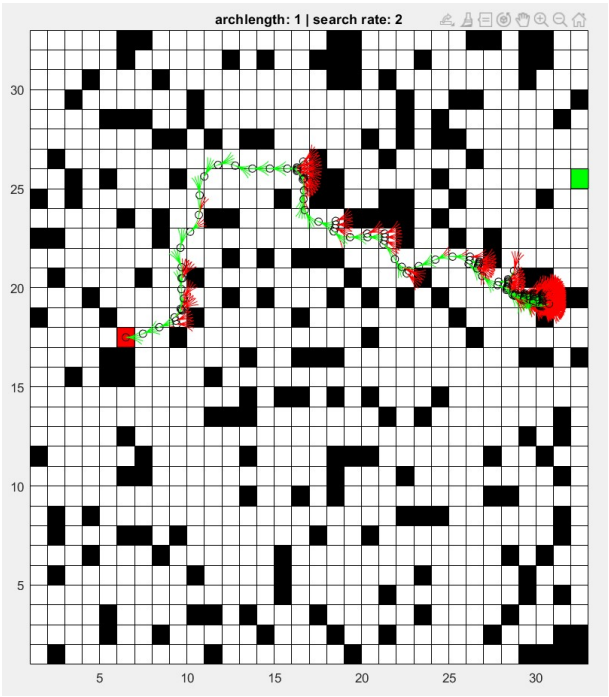


Fig. 2: At the right-side corner, the car generated numerous potential trajectories but exhibited hesitance when attempting a wide turn in the passage. In this context, a red curve represents an impractical trajectory, while a green curve signifies a workable, feasible trajectory. The black box indicates the presence of an obstacle.

Notably, when the search rate was set to 2, the planner with a curve length of one expended significantly more time analyzing potential optimal paths compared to the other planners under the same conditions (Table II). This delay stemmed from an unfortunate situation where the car found itself trapped on three sides, as illustrated in Figure 2. Because of the limited trajectory size, the car's planner had to meticulously explore its surrounding regions, incrementally updating its optimal current state. This process resembled falling into a deep rut, leading to a state of stagnation and causing an infinite search time.

Arc length	size of control set	elapse time [sec]
1	181	> 60
2	82	8.08
3	43	3.59

TABLE II: The search performance results with the search rate = 2

Seeing the result from the table I-III, the planner with a longer motion trajectory found its path faster and did not need to consider many edges relatively.

Arc length	size of control set	elapse time [sec]
1	314	38.40
2	144	50.82
3	83	6.73

TABLE III: The search performance results with the search rate = 3

Nonetheless, the paths generated by longer arc lengths were not as concise and precise as those generated with shorter arc lengths. As depicted in Figure 3, the path explored by the shorter trajectory yielded a more direct route to the goal, even though it involved a larger control set for computational consideration.

This raises the question of whether the path is genuinely optimal. If the user prioritizes finding a path quickly, a longer arc length variable might be preferred. On the other hand, if the user seeks a shorter, more accurate path, a shorter arc length variable would be the choice. This prompts a discussion about whether to opt for a route that minimizes the time the robot takes to reach the destination (short path with a longer search time) or one that is rapidly discovered by the robot to reach the destination (longer path with a shorter search time).

IV. Conclusion

The experimental results clearly demonstrated that the new algorithm, combined with lattice graphs, effectively identified routes that steered the robot clear of obstacles (black boxes), guiding it to the destination exclusively using feasible motions. Prior to the test, certain variables needed to be configured, specifically the arc length and search rate. The planner's performance exhibited variations based on the chosen variables. In one scenario, the algorithm spent considerable time updating its current state, and the robot found itself unable to escape due to the small arc length value, as it had to explore all locations enclosed by walls on three sides (Figure 2).

Excluding this specific scenario, the trajectory with a size equivalent to the unit length wall (s = 1) generally resulted in the most satisfactory path (Figure 3(a)), while the planner with a longer trajectory computed fewer potential routes and reached its destination earlier (Table I-III).

Consequently, the robot would not generate a path if it determined that it couldn't reach the goal solely through executable motions. In such cases, the robot required new instructions. Despite this, the lattice graph significantly expedited the process by determining, right from the start when the robot received new instructions, whether the robot could reach the goal using only feasible motions.

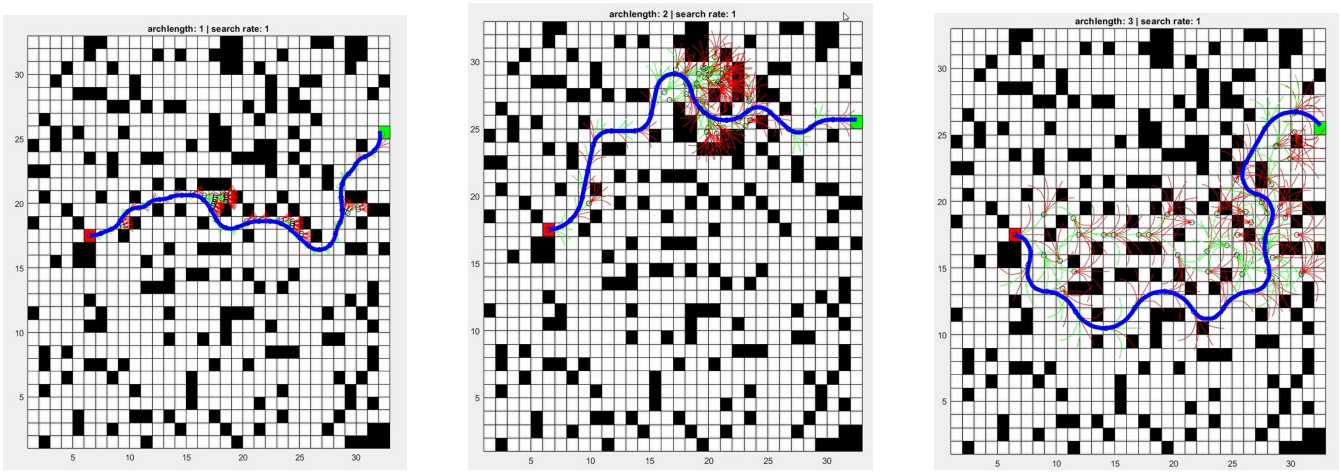
Regrettably, when analyzing the results, it remained uncertain which path to choose in real-world applications. The forthcoming report will address a new algorithm capable of making decisions between the shortest path and the fastest path.

References

[1] *P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.*

[2] *M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 2611-2616, doi: 10.1109/IROS.2008.4651220.*

[3] *M. McNaughton, C. Urmson, J. M. Dolan and J. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 4889-4895, doi: 10.1109/ICRA.2011.5980223.*



(a) arc length = 1 (s = 1) (b) arc length = 2 (s = 2) (c) arc length = 3 (s = 3)

Fig. 3: The planner produced optimal paths to the destination when the search rate was set to 1. Each planner operated with a distinct arc length value. In the visual representation, a red line indicates an impractical motion, a green line represents a feasible motion, and a blue line signifies the optimal path. The red box marks the starting point, the black box designates an obstacle, and the green box denotes the destination.