A Combined A* Algorithm with Lattice Graphs that Guarantees Feasibility

MinGyu Kim

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

***Abstract*— Our race car was trapped at a corner in the simulation. The car successfully created its path and was on the way to the destination, however, it got into being stuck at a corner at the moment when it made a turn. This was caused by infeasible motions produced by the code, therefore, we decided to modify the A* search algorithm. With the motivation from the lattice graph which is a kind of virtual pre-computed trajectory, we successfully merged the lattice graph into the original A* algorithm. The new planner found routes to the goal with feasible motions only and ensured efficient obstacle avoidance.**

I. INTRODUCTION

It is hard to find paths that can be operated in an unknown environment. In general, a robot is put in a place, a static map on a robot system is updated, and the algorithm tries to calculate the ways to the given destination. To do that, we chose the A* algorithm [1] among many search algorithms to test a racing car in a virtual environment and configured and tested the algorithm on the Gazebo racing car simulation. The car always successfully generated the path toward the destination that we selected, however, a tracking problem was found. The car was stuck at a corner several times when it was given a prompt that it should move close to the wall or a prompt that it should make a rapid turn near the wall. The car did not manage to move forward even if it generated an optimal path. The car had to receive a new prompt to escape from the wall. It had to make a turn at the location and had to generate a new path at a new location.

To resolve this issue, we propose to use lattice graphs [2] in a path planning algorithm. The pre-computed graphs ensure to find a path with feasible motions only, therefore, it can guarantee that paths generated with the graph are always executable. The pre-computed trajectory also will save much time enabling it to scan less possible states.

II. METHOD

This experiment was conducted in a 32 by 32 grid map with a random set of square obstacles. The start and destination positions were given before the test, and the obstacle information was extracted from the static grid map. There are three components that were newly added to the original spatial A* algorithm [1]. Each component is described specifically below in the sub-section. Each state node contains a set of variables [x(s), y(s), $\theta(s), \kappa(s)$] of arc length s. Also, 5 different edges generated as lattice graphs were used to explore the surrounding regions.

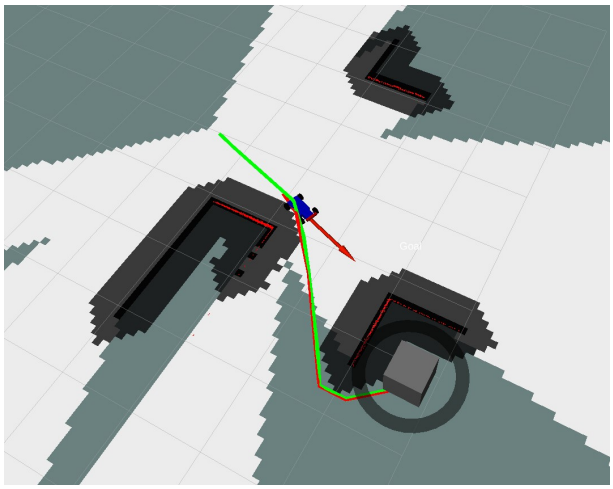


Fig. 1: A car in the Gazebo simulation is stuck at a corner, although the car successfully generated its trajectory to the designation (grey cube box).

A. Generate a Lattice Graph

The lattice graphs were empirically created the same way as the reference [3]. The curvature of the graph was calculated with the user-defined constants [p0, p1, p2, p3, arc length s]. The planner had different motion curves depending on what coefficients the curvature function was given.

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

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

The user-defined coefficients were set to be equal to fractions of the curvature function so as to guarantee continuity and satisfy boundary conditions, as shown in the equations 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.

$$a(p) = p_0$$

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

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

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

Once the coefficients were calculated, the state of a node was estimated. The orientation function at the node was derived by taking derivatives of the curvature function that was gotten in the previous step. The positions on the x and y axes were derived by using the kinematic bicycle model. Taking integral of the orientation function from the current position to a point of the arc length can determine 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

However, it was hard to get the digit numbers of the state information using the continuous method like taking integral of a function. To solve this issue, the Simpson's Rule was introduced. Discrete numbers generated by Simpson's rule were needed, and each node saved the numbers where a potential node should be explored.

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

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

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

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

C. Obstacle detector

In the original condition, it was hard to consider if the object collides with an obstacle or not. The dot point in the grid map was an indefinite position of the state node. A virtual boundary around the state node was needed. As a result, the dimension of a car-shape square around a state point was generated in a function, and with the dimension and obstacle information from the static map, the obstacle detector function determined if the object would crash into the wall or not.

In this test, a 0.5 by 0.5 square box was used which was half size of the unit wall, and each vertex of the square (4 vertices in total) was used to detect the obstacle.

III. Result and Discussion

Before the test, the length of the edge and search rate must be determined. The length of the edge refers literally to the length of the motion trajectory generated, and the search rate refers to the number of intervals at which it wants to stop in the middle of the trajectory to explore another surrounding region. The experiment has proceeded with a total of 9 sets of different arc length and search rate values, and the search algorithm performed differently depending on the variables.

In general, the planner which contains a longer edge found its path faster than the other ones with shorter curves, also, the planner with shorter edges created more potential trajectories and had a larger size of the control set.

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

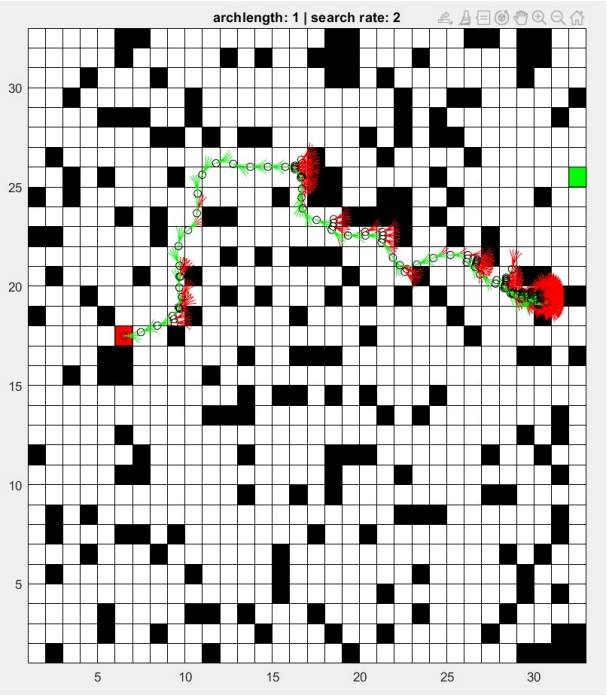
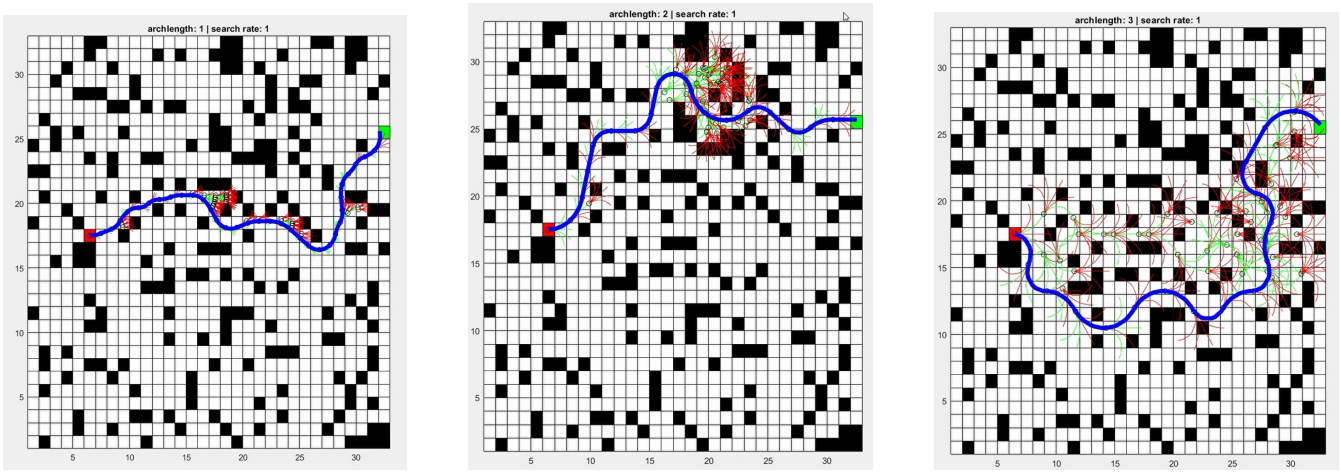


Fig. 2: The car yielded a lot of potential trajectories at the corner on the right side, and it was being reluctant to make a large turn at the passage. A red curve is an infeasible trajectory, and a green curve is a feasible trajectory. The black box is an obstacle.

Particularly, at the search rate of 2, the planner whose curve length is one has consumed much time examining a possible optimal path, compared to the other planners in the same condition (Table II). This was because the car, unfortunately, got into a location walled in on 3 sides, as shown in Figure 2. Due to the small size of the trajectory, the planner of the car had to highly precisely explore its surrounding regions by updating its optimal current state gradually, which looked like it had fallen into a deep loophole. The stagnation in the loophole triggered an infinite search time.



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

Fig. 3: The optimal paths to the destination are generated by the planner at the search rate = 1. Each planner was operated with a different arc length value. A red line is an infeasible motion, a green line is a feasible motion, and a blue line is an optimal path. The red box is the start point, the black box is an obstacle, and the green box is the destination.

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

However, the paths generated by longer arc lengths were not as short and precise as the ones generated by a shorter arc length. As seen in Figure 3, the path explored by the shorter trajectory developed a shorter path to the goal, although the shorter trajectory had a larger control set to consider in computation. This was the point to discuss whether the path was truly optimal. If the user wants to find a path faster, he will use a longer arc length variable. If the user wants to find a short, precise path, he will use a shorter arc length variable. It should be discussed whether to choose a route with a shorter time for the robot to actually move to the destination (a short path & long search time) or a route that is quickly discovered for the robot to move to the destination (a long path & short search time).

IV. Conclusion

The experiment result has shown that the new algorithm merged with lattice graphs clearly found a route avoiding the obstacles (black box). It successfully guided the robot to the destination using feasible motions only. Before the test, there were some variables to be set; arc length and search rate. The planner showed a different performance depending on what variables to choose. In one case, the algorithm code spent much time updating its current state. The robot was not able to escape from the hole with the small arc length value because the robot had to explore all locations bounded by walls from 3 sides (Figure 2). Except for this scenario, generally, the trajectory with the same size as the unit length wall (s = 1) accomplished establishing the most satisfying path (Figure 3(a)), and the longer trajectory planner computed fewer potential routes and found its path earlier (Table I-III). As a result, the robot will not generate a path if it determines that the robot cannot move to the goal with executable motions only. If that happens, the robot should receive a new order. In spite of it, the lattice graph will save you much time by being able to determine if the robot can go to the goal with the feasible motions only in the beginning when the robot receives a new prompt. Unfortunately, while analyzing the result, it was uncertain about what kind of path to select in the real world. The next report will cover a new algorithm that is able to make a decision 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.*