Combinational Path Planning Algorithms For Robot Considering Temporal Constraints

Piyush Kumar Bhuva
Robotics
University of Maryland
College Park, USA
piyush63@terpmail.umd.edu

Vamshi Kumar Bogoju
Robotics
University of Maryland
College Park, USA
vamshikumar0401@gmail.com

Abstract—This draft discusses path planning algorithms used in autonomous ground vehicles like cars and trucks. Two path planning algorithms are discussed, A* and Hybrid A* . In the current conditions autonomous vehicles are using multiple methods to explore possible nodes and reach from point A to point B. The nut n a cell is that it is rather interesting to use multiple/hybrid algorithms at the same time that can have a switch is better than single algorithm completing the task. For proving we have used famous A* algorithm and hybrid A* algorithm to reach in region of interest. We have also compared performance considering factors like time complexity, accuracy and robustness. This must be following that it should avoid the obstacles with proper margin and after reaching the goal node, give an path exploration. After evaluating both algorithms we will reach at conclusion considering various factors. This will lead to an optimal path in both the algorithm avoiding obstacles.

Keywords- A* motion planning, Path planning, Self-driving cars, Autonomous Vehicle Agents, Heuristic algorithms, Automobiles, hybrid algorithms, Autonomous vehicles

I. INTRODUCTION

Every year almost 40000 accidents happen in the USA alone and 90% of these are due to drivers' negligence. These numbers are not decreasing nor expected to decrease as almost every day new 7000 vehicles are bought in united states. These numbers still make us forceful to solve the issue and this numbers are given that US has comparatively stricter road rules. We are on the verge of changing the automation age and self-driving cars are on the front line of it. Not only self-driving cars are better, but also convenient and safe option, as it is also safer compared to current standards of human driver as due to artificial intelligence cars learns faster than human driver. Perception, Localization, Obstacle avoidance and Path Planning are key components of self-driving cars [1]. Few more aspects also impact self-driving cars like kinematics [2] That is why there is a surge in algorithms involved with motion planning and control. In this paper, we are introducing a comparison between A* and hybrid A* algorithms and how the result and time complexity varies between them. This algorithm will give us a clear idea of which scenarios where best planning algorithm is applied considering various factors.

Self-driving car models have different degrees of autonomy depends on the car itself. For example, Tesla cars have level 2 autonomy and where the constant need for a drivers' attention. Although this car has achieved a reputation of achieving high level of autonomy. The main matter is finding the paths with more robustness, acuteness and should be dynamic meaning it should be responsive even when environment has some anomaly.There is always a need of constant change in algorithms and this is called 'Algorithm switching'. For an instance, in freeways reaching A to B ,time is the main factor consideration then we will use faster tree expansion algorithms expanding almost all the options. Sometimes reaching A to B with rank hierarchy(nothing but distance) we consider cost as a main factor and use an algorithm which prioritizes the Euclidean or Manhattan distance [3].

So, the target that we are trying to achieve is exploring various methods, algorithms and gist of report is comparison of performance with various standards. We have taken a very simple algorithm like A*,which uses costs, and explained that using it in a different way is so much powerful and accurate[4][5]. For example, A* limits us to use discretized space and hybrid A* allows us to make a choice and easily applicable to continuous space[6][7].

So, A* must be used in n*n grid and hybrid A* deletes that limitation. Here hybrid algorithm explores the moves that is restricted in the normal A* algorithm. Many other features can be unlocked using the modification of algorithms. And that is why nowadays we have stressed on the hybrid complex algorithm.

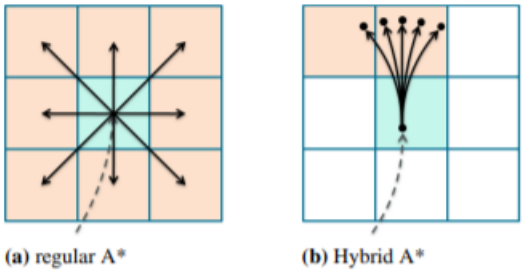

Fig. 1: A* and Hybrid A*

The obstacle space that is used as shown below which consists of 2 walls and displaying goal point besides the cross point, node exploration using the hybrid A* code.

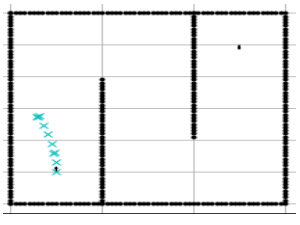

Fig. 2: Obstacle Space

II. METHODOLOGY

The pipeline followed in this process is actually very simple
Where a common map is used for both the algorithms and to compare fairly, same map and same computational machine are used. This minor step allows us to compare them on fair base.
The main aim of suggested comparison is quickly to find out the comparison between 2 well-established algorithms and inform user to make choice between them. This paper will clarify when, where and how to compare these algorithms considering performance and time complexity at the same time. This knowledge will provide a pre-testing data and features to consider when implementing in the real world. So, a map is created as shown in figure 2 where mainly it is of algebraic equations which is lines in a 2-dimensional plane and implemented via equation of lines. This equation is an obstacle space mathematically which will reflect in images as well(as shown in figure 2).
The next step in the pipeline is to define set of points from where the robot will start moving and the goal node/point is also hardcoded. For more complex cases we have chosen points at the extremes(2 black dots as mentioned before) which make an absolute representation of robustness in which point robot has to work through 2 walls of a different dimension. The bottom left point is chosen to be the origin and all coordinated frame is with respect to that only.

The second step in the pipeline is to design an A* algorithm for which a ranking algorithm is heart. A* algorithm explores the child node and at every step, it ranks a distance (cost-Euclidean distance) and put them in best of the cost approach path. Just keep a record for a final trajectory and record time run. While running the code it calculates every path possible and cost will decide the optimal path. The explored nodes can be seen here as a blue cross tick marks.

Now the next step in the pipeline is to run a hybrid algorithm and explore the power it wields for us. In this part, the curves are taken as per Reeds and Shepp [8]. A* does a nice job to find local optimum but it is not always important that hitting local feasible point every time gives you overall optimum performance. That is why we use dynamic programming to understand why we have to look for a bigger picture and that is what hybrid A* offers- an optimal path with global minimum cost.

III. PREVIOUS WORKS

According to research paper that we have referred one possible approach is designing them and fairly compare them to give optimum results as explained earlier. Hybrid A* algorithm has steps as below which is in pseudo code and can be implemented in other language as well [9]. Hybrid A* is also successful in aerial path planning [10] and can be used in urban crowded scenarios.
These examples here will clarify the idea how different modifications of A* are used and sharing a glimpse of results will hep to make a better conclusion.

Speaking of time complexity of both the algorithm which resembles can be described as heuristic functions and this is represented as follows.

Algorithm 1 Standard version of Hybrid A*

1: **procedure** PLANPATH(m, μ, x_s, θ_s, G)
2: $n_s \leftarrow (\tilde{x}_s, \tilde{\theta}_s, x_s, 0, h(x_s, G), \text{-})$
3: $O \leftarrow \{n_s\}$
4: $C \leftarrow \emptyset$
5: **while** $O \neq \emptyset$ **do**
6: $n \leftarrow$ node with minimum f value in O
7: $O \leftarrow O \setminus \{n\}$
8: $C \leftarrow C \cup \{n\}$
9: **if** $n_x \in G$ **then**
10: **return** reconstructed path starting at n
11: **else**
12: UPDATENEIGHBORS(m, μ, O, C, n)
13: **end if**
14: **end while**
15: **return** no path found
16: **end procedure**

17: **procedure** UPDATENEIGHBORS(m, μ, O, C, n)
18: **for all** δ **do**
19: $n' \leftarrow$ succeeding state of n using $\mu(n_\theta, \delta)$
20: **if** $n' \notin C$ **then**
21: **if** $m_o(n'_{\tilde{x}}) =$ obstacle **then**
22: $C \leftarrow C \cup \{n'\}$
23: **else if** $\exists n \in O : n_{\tilde{x}} = n'_{\tilde{x}}$ **then**
24: compute new costs g'
25: **if** $g' < g$ value of existing node in O **then**
26: replace existing node in O with n'
27: **end if**
28: **else**
29: $O \leftarrow O \cup \{n'\}$
30: **end if**
31: **end if**
32: **end for**
33: **end procedure**

This is the standard procedure followed to implement hybrid A* algorithm and it can be described shortly like this:
You are taking a current pose of robot from the user and which has initial angles of rotation which in our case becomes unnecessary as we are considering a point robot here. We use while loop here which is same as we use it in the standard A* algorithm but updating neighbor here is a little updating procedure where we are again.

For an instance, UpdateNeighbor function is a special in a sense as it compares this state and successive states as well as scene in line no.19 and again we will check that our updated state should be out of obstacle space. If a new cost is not meeting a threshold then we keep new state and cost as it was before.

This is one of the ramification of how we can add extra terms which will gives us a better path considering a cost. The other modification for the same we can give it like below.

Algorithm 2 Extended version of Hybrid A*

1: **procedure** PLANPATH($m, \mu, x_s, \theta_s, G_1, G_2$)
2: $n_s \leftarrow (\tilde{x}_s, \tilde{\theta}_s, x_s, 0, h(x_s, G_1, G_2), \text{-}, 1)$
3: $O \leftarrow \{n_s\}$
4: $C \leftarrow \emptyset$
5: **while** $O \neq \emptyset$ **do**
6: $n \leftarrow$ node with minimum f value in O
7: $O \leftarrow O \setminus \{n\}$
8: $C \leftarrow C \cup \{n\}$
9: **if** $n_s = 1$ **then**
10: **if** $n_x \in G_1$ **then**
11: UPDATENEIGHBORS($m, \mu, O, C, n, 2$)
12: **else**
13: UPDATENEIGHBORS($m, \mu, O, C, n, 1$)
14: **end if**
15: **else if** $n_s = 2$ **then**
16: **if** $n_x \in G_2$ **then**
17: **return** reconstructed path starting at n
18: **else**
19: UPDATENEIGHBORS($m, \mu, O, C, n, 2$)
20: **end if**
21: **end if**
22: **end while**
23: **return** no path found
24: **end procedure**

25: **procedure** UPDATENEIGHBORS(m, μ, O, C, n, s)
26: **for all** δ **do**
27: $n' \leftarrow$ succeeding state of n using $\mu(n_\theta, \delta)$
28: $n'_s \leftarrow s$
29: **if** $n' \notin C$ **then**
30: **if** $m_o(n'_{\tilde{x}}) =$ obstacle **then**
31: $C \leftarrow C \cup \{n'\}$
32: **else if** $\exists n \in O : n_{\tilde{x}} = n'_{\tilde{x}} \wedge n_s = n'_s$ **then**
33: compute new costs g'
34: **if** $g' < g$ value of existing node in O **then**
35: replace existing node in O with n'
36: **end if**
37: **else**
38: $O \leftarrow O \cup \{n'\}$
39: **end if**
40: **end if**
41: **end for**
42: **end procedure**

Here we have extended our change on the procedure function. The change is the frequency we with which we update the successive node here. As you can see from line 10 and 14 we can modify the nodes where if the first criterion is met we keep s=1 otherwise we keep s=2.

IV. OUR RESULTS

After comparing standard A* and hybrid A* we have numerous conclusions and we have classified it in following ways. The very first way that we have tried is the time complexity and how much time it takes with various complexity of maps

For an instance, when we used a standard A*, it was comparatively a less time taking to implement. A reed-sheep map is taking around 130 seconds to implement. On the other hand, if we are using hybrid A* which is a little modification of standard A*(Hybrid A*) gives a little poor timing to execute and it also depends complexity of an map. For simple 2 obstacle map it have taken around 170 seconds to execute and when we have taken multiple walls as objects this time has shoots almost 3 times.

The seconds comparison that we have observed is direct comparison considering multiple parameters. Considering the performance of both algorithms we have observed that A* is faster to implement but not so accurate compared to hybrid A*. Below images shows the path A* has followed and the second photo shows path for hybrid A*.

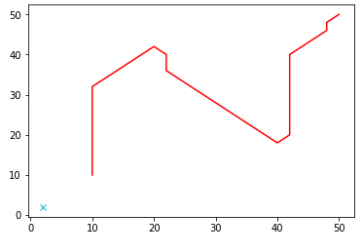

Fig. 3: A* Algorithm

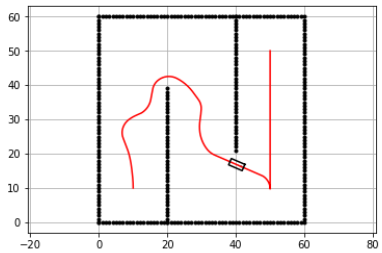

Fig. 4: Hybrid A* Algorithm

These 2 images shows a stark difference between the same start and goal point, but great difference can be seen in the ability for car traverse in the paths. The first image shows an path followed by standard A* which is considering a workspace as a discrete space and path has some sharp curvatures and angles. On the other hand, the second image shows path followed by hybrid A*, which shows how the workspace is considered as continuous space and the turns that robot has taken here are realistic and doable.

The second observation is reverse taken in the hybrid A*, where the robot has reversed itself to go at goal point. This is not shown inn standard A* algorithm because standard A* follows forward path exploration (even if you take reverse path exploration).

The conclusion that we have made here was that for real world scenario and self-driving car it is blunder to take real space as discrete space (even dividing the scene in numerous pixels). So, in real self-driving car we try to use modifications of standard A* algorithm and we have done the same which has given a satisfactory results as shown in the second image.

The next result was taken in consideration was a cost that has been involved. So, far we have used a simple Euclidian distance as a cost for standard A* which shows no deflection of cost when obstacle is nearby. On the other hand, for hybrid A* when obstacle is nearby cost increases in a rapid rate. The reason can be followed as:

In the continuous space we have used a car has given multiple dimension like [left_steer, forward, right_steer] which gives robot a freedom to stir in proper direction. The left_steer and right_steer are the angles in radian and decides the direction to move. Another observation that we want to add here is that after selecting one ramification in hybrid A* the other part

remains unexplored (which makes an algorithm faster to go ahead).

The last observation that we want to conclude is within the hybrid A* algorithm alone. We have designed a car model and we wanted to add extra layer of complexity, which is adding features which we can change regarding the car model. We have changed the max steering angle that it can reach and observed the results and it can be juxtaposed as follows:

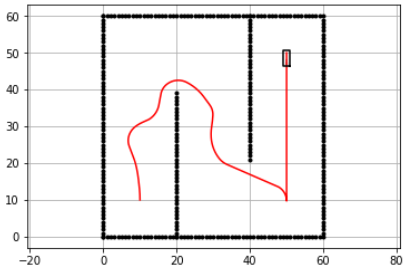

Fig. 5: Steering angle 35

This image shows an path followed in hybrid A* with 35 degrees of steering angle at maximum. This second image shows the path when we have taken steering angle roughly 70 degrees.

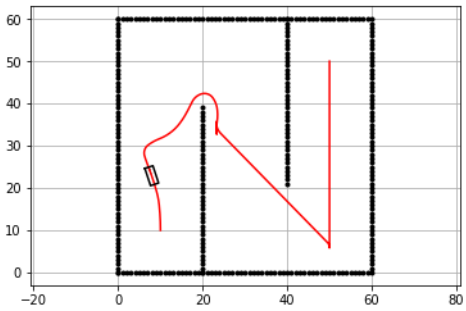

Fig. 6: Steering angle 70

This image shows how coming around middle robot takes a shortcut to reach at goal point and this scene is a little unrealistic for real car as this kind of steering angle near to impossible to achieve but it is one of the observation.

Another little tweak we have performed is changing distance between front and rear wheels. And if we keep this difference between front and rear wheels, we can simply see that our car deviates from trajectory instead of keeping in the middle and this can be expressed in following images.

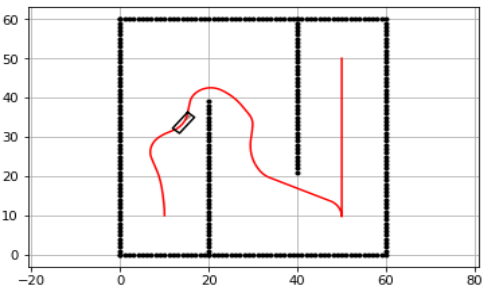

Fig. 7: After distance between wheels changed

Observing the image, we can observe that comparing it to other images the car is deviated from the center lane.

The next observation is with the complicated maps and how A* and hybrid A* performs in each map. This is better visualization that A* failed to accomplish and described as follows:

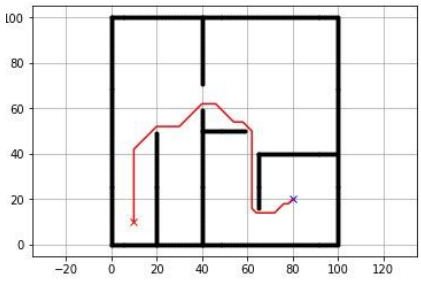

Fig. 8: A* with complex obstacle map

This image shows how mechanical steps standard A* take in workspace. It has very drastic changes and again may not applicable to real world car.(For an instances an action at last stages where car takes almost 90 degrees turn). Now, comparing it to the hybrid A* have astounding results and as seen follows:

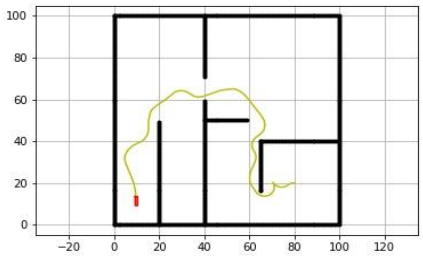

Fig. 9: Hybrid A* with complex obstacle map

Here observing we can see that we are also taking a reverse in last phases in reaching goal point which is not possible to standard A* algorithm and path exploration is more realistic and having more real feel to it.

V. Conclusion

To conclude, we can infer from the results that A* is a faster algorithm but is not available to application in real self-driving car. Here, it is important to have an algorithm which works for continuous algorithm and hybrid A* fits well. The time taken for the same (kind of) map taken by standard A* is less compared to hybrid A*. Comparing the robustness, it is evident that standard A* is not up to the bar and hybrid A* has impressive performance. Obstacle avoidance for standard A* is poor compared to hybrid A* and boundary clearance is poor in standard A*. There is no reverse option for normal A* whereas hybrid A* as shown in the results performs a lot better. Changing the car parameter produces a different kind of results every time when we change angles at max which can produce a different results and which matches to real world scenarios.

VI. ACKNOWLEDGEMENTS

We love to thank our professor Dr. Reza Monfaredi to his constant support as our mentor. He provided unwavering faith and support to complete this mammoth task. He also provided insights to us where we were stuck in python simulations. He also encouraged us to constantly update the obstacle map and aim to make our algorithm to provide more robustness. We have learned and enjoyed working with him and looking forward to learning a lot.

VII. REFERENCES

[1] Sreenatha G. Anavatti, Sumana Biswas and Jedd T. Colvin, "A Hybrid Algorithm for Efficient Path Planning ofAutonomous Ground Vehicles".

[2] Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli, "A Survey of Motion Planning and Control"

[3] Peter E. Hart, Nils J Nilsson, and Bertram Raphael, "A Formal basis for the Heuristic Determination of Minimum Cost Paths".

[4] LaValle, S. M.: Planning Algorithms, Cambridge University Press, 2006.

[5] N. Sariff and N. Buniyamin, "An overview of autonomous mobile robot path planning algorithms," In Research and development, 2006.SCORed 2006. 4th student conference on IEEE, pp. 183-188, 2006.J.Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2.Oxford: Clarendon, 1892, pp.68–73

[6] Michael Montemerlo,Jan Becker, "Junior: The Stanford Entry in the Urban Challenge"

[7] Dmitri Dolgov, Sebastian Thrun, "Practical Search Techniques in Path Planning for Autonomous Driving"

[8] J.A.Reeds, L.A.Shepp, "Optimal Paths for a Car that Goes both Forwards and Backwards"

[9] Janko Petereit, Thomas Emter, Christian W. Frey,Thomas Kopfstedt, Andreas Beutel, "Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments".

[10] Richads, N. D.; Sharma, M.; Ward, D. G, "Hybrid A*/Automaton Approach On-Line Path Planning with Obstacle Avoidance", AIAA 1st Intelligent Systems Technical Conference, 200