Motion Planning for a Dubin's Car Using RRT

Ashit Mohanty , Huzefa Shabbir Hussain Kagalwala, Rahil Modi, Rakshitaa Geetha Mohan
Department of Automotive Engineering, Clemson University
International Center for Automotive Research (CU-ICAR)
2020 Spring Course Report: CPSC 8810 Motion Planning, Instructor: Dr. Ioannis Karamouzas

Abstract

In recent times sampling-based planning algorithms like Rapidly - exploring Random Tree (RRT) are extensively used in path planning of robots. RRT for path planning uses a randomized data structure. RRT is designed keeping in mind the need to handle nonholonomic constraints and high degrees of freedom of the configurations space. Dubin's path is an algorithm in which we find the shortest curve between 2 points with the help of geometry. It can be done in 2-dimensional or 3-dimensional space. Our project aims at using RRT to create a roadmap and plans a path to maneuver a non-holonomic car (a Dubin's car model) to move from an initial state to a goal state in 2-dimensional space.

1. Introduction

RRT

Finding a path from the initial start point to a goal point within a configuration space is called path planning. It also involves finding a continuous path avoiding the obstacle region and the entire path should lie in the free configuration space. [1]

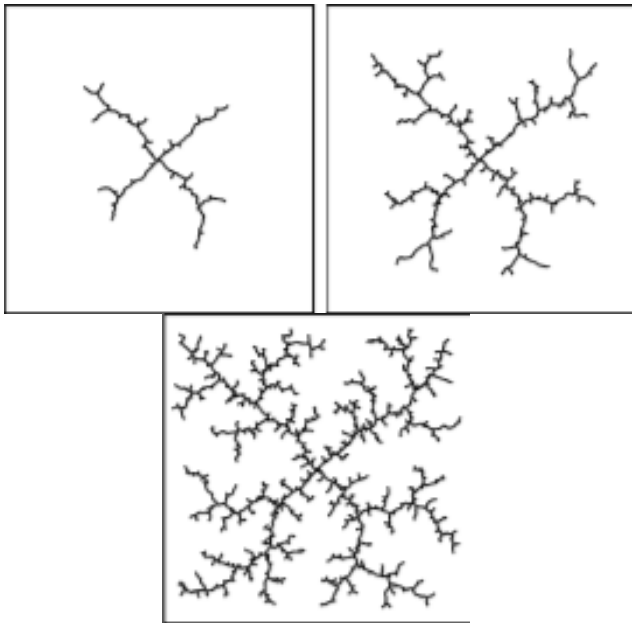

Figure 1 RRT tree expanding

RRT is unlike other path planning algorithm, in the sense that it is more inclined towards unexplored portions of the state space.

As seen in Figure 1 an RRT tree expands within a square by going towards the unvisited portions of the square than going to places that are already visited. This is one of the good properties of RRT.

RRT is proved to be faster than the regular PRM algorithm in experiments. It can generate a minimal path with the fewest edges which makes it faster as regular PRM algorithm may connect to additional edges just to make a connected path to the goal position. PRM algorithm uses k-nearest queries which are more costly than single nearest neighbor query which is used by RRT. Collision detection is one of the important aspects of path planning and RRT is very good in incremental collision detection. [2]

GENERATE_RRT($x_{init}, K, \Delta t$)
1 $\mathcal{T}$.init(x_{init});
2 **for** $k - 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

Figure 2 RRT Pseudocode *[2]*

Above mentioned is the pseudocode for an RRT algorithm.

Dubin's Path [3]

The problem of finding the shortest path between two points from start $(x_1 y_1 \theta_1)$ to goal $(x_2 y_2 \theta_2)$ with a constraint of turning radius was first solved by L.E. Dubins in 1957. According to him, the optimal path must contain only 3 segments which can be either a turning radius denoted by C or a straight path denoted by S. The type of optimal path is either a CSC or CCC.

There are 6 combinations which are known as the Dubin's curves and they are LSL, RSR, LSR, RSL, LRL, RLR. When the car configuration updates from the previous configuration to a new configuration there are dynamic

equations which help us define the new configuration based on the earlier configuration and the dynamic equations when translated into update equations it can be written as

$$x_{new} = x_{prev} + \delta * \cos(\theta) \quad [1]$$
$$y_{new} = y_{prev} + \delta * \sin(\theta) \quad [2]$$
$$\theta_{new} = \theta_{prev} + \frac{\delta}{r_{turn}} \quad [3]$$

CSC trajectories include LSL, RSR, LSR and RSL which is a turn then a straight path then again a turn as seen in Figure 3

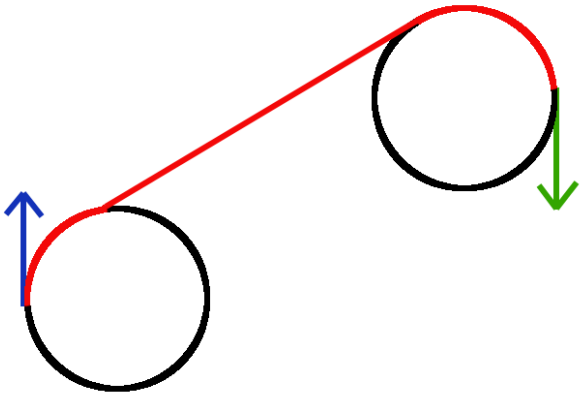

Figure 3 RLR curve *[3]*

There are four possible tangent lines for each pair of circles but only one of them valid for each case as it can be seen in Figure 4. To get the correct trajectory we need to get the tangents correctly.

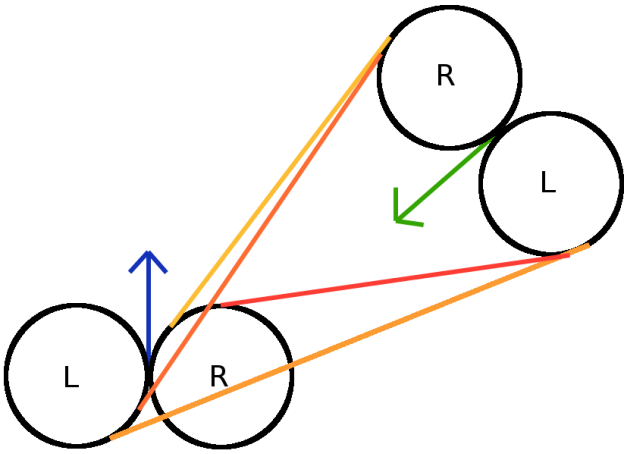

Figure 4 Possible Tangents *[3]*

CCC trajectories are different as it has a turn and then another turn in the opposite direction and again the turn in the same direction as the first one. According to Dubin's there are only 2 CCC trajectories RLR & LRL curves. These curves are sub-optimal. For these curves to be optimal the start and goal need to be very close to each other. In Figure 5 the RLR curve is shown which is a CCC trajectory

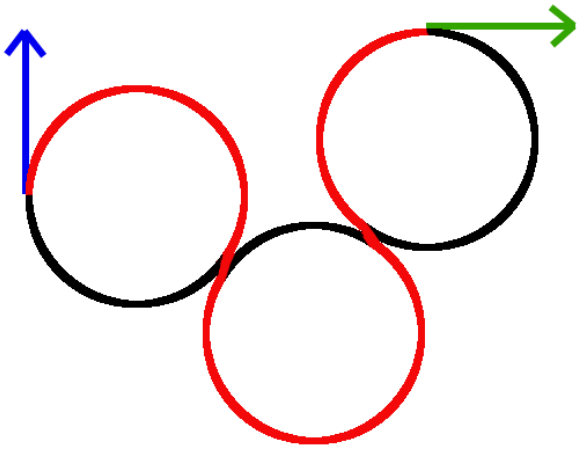

Figure 5 RLR curve *[3]*

2. Implementation

RRT

It is a path finding algorithm which searches the available space by building a tree using randomly sampled points. It is used to generate open loop trajectories for systems with state-constraints. These algorithms can handle multi-dimensional spaces (up to twelve degrees of freedom) with multiple queries and obstacles. RRT algorithm does not return the optimal path as it chooses a random node to build the tree.

For the given start and goal configuration, the RRT finds a random node in the space. Before adding it to the tree, the nearest node on the tree to the random configuration is found. If the distance between the nearest node and the random node is greater than the step distance, the nearest node on the tree is propagated in the direction of the random node with the step distance. This new node generated is added to the tree if the node is not on the obstacle. In this way, the RRT propagates. The RRT does not stop until it has reached the goal configuration. The path from the initial to the goal configuration is found by tracing the parent of the goal node and the following parent nodes.

Dubin's Path

For the derivation of Dubin's curves equations, we have followed the procedure mentioned in the document "*A Comprehensive, Step-by-Step Tutorial on Computing Dubin's Curves"* by Andy Giese [3]. First, we need to compute the center point for the circles corresponding to the start and endpoint as minimum turning radius as the radius of these circles.

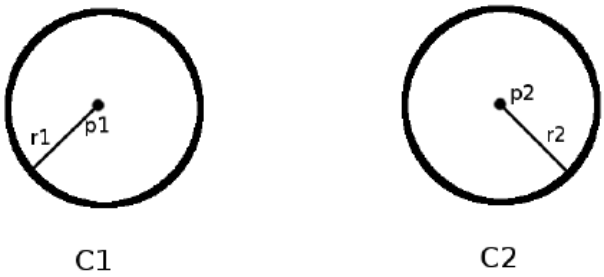

Figure 6 Center points of two circles *[3]*

$$P_{c1} = \left(x_1 + r_{min} * \cos\left(\theta_1 - \frac{\pi}{2}\right), y_1 + r_{min} * \sin\left(\theta_1 - \frac{\pi}{2}\right)\right) \quad [4]$$

$$P_{c2} = \left(x_2 + r_{min} * \cos\left(\theta_2 - \frac{\pi}{2}\right), y_2 + r_{min} * \sin\left(\theta_2 - \frac{\pi}{2}\right)\right) \quad [5]$$

After the computation of the center points of the 2 circles, we need to calculate θ and the tangent points on both the circles.

$$\theta = tan^{-1}\left(\frac{P_{c2y} - P_{c1y}}{P_{c2x} - P_{c1x}}\right) \quad [6]$$

$$P_{t1} = (x_1 + r_{min} * \cos(\theta_1), y_1 + r_{min} * \sin(\theta_1)) \quad [7]$$

$$P_{t2} = (x_2 + r_{min} * \cos(\theta_2), y_2 + r_{min} * \sin(\theta_2)) \quad [8]$$

Once the tangent points are calculated we need to calculate the distance between the centers of the two circles P_{c1}&P_{c2}. And it is denoted by D.

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad [9]$$

After computation of the D we need to compute the Arc lengths of the arcs on both the circles, first, we need to find out the turning radius of the two circles and the equations for them are:

$$\theta_i = tan^{-1}\left(\frac{y - P_{c1y}}{x - P_{c1x}}\right) - tan^{-1}\left(\frac{P_{t1y} - P_{c1y}}{P_{t1x} - P_{c1x}}\right) \quad [10]$$

$$\theta_f = tan^{-1}\left(\frac{P_{t2y} - P_{c2y}}{P_{t2x} - P_{c2x}}\right) - tan^{-1}\left(\frac{y_{goal} - P_{c1y}}{x_{goal} - P_{c1x}}\right) \quad [11]$$

Above equations give the turning angles when they are multiplied by the min turning radius, we get the arc lengths of both the circles. In *LSL* & *RSR* the straight path length is equal to the distance between the centers of the two circles. But in *RSL* & *LSR* the straight distance needs to be calculated with the help of trigonometry. The total distance of the path can be calculated by adding the two arc lengths and the straight path.

Implementation

The user has to enter the start and goal configurations (x,y,θ). The start node gets added to the tree initially. After the random vertex is generated, the nearest node (from the tree) is found. The nearest node is propagated along the direction of the random node by a growth factor/step distance (in this case, 1) and this becomes the new node. The path to reach the new node from its parent is found by using one of the four Dubin's curves (LSL, RSR,LSR,RSL). The trajectory that gives minimum distance is taken as the ideal path. The new node is checked for collision. If it lies in the collision-free space, then the node is added to tree. This way, the RRT gets propagated until the new node is within a set distance of the goal node.

A separate variable in the node class is used to store the index of the parent. The same process is repeated for connecting the goal node to the start node. The paths are connected by tracing the parent of the child node.

3. Results

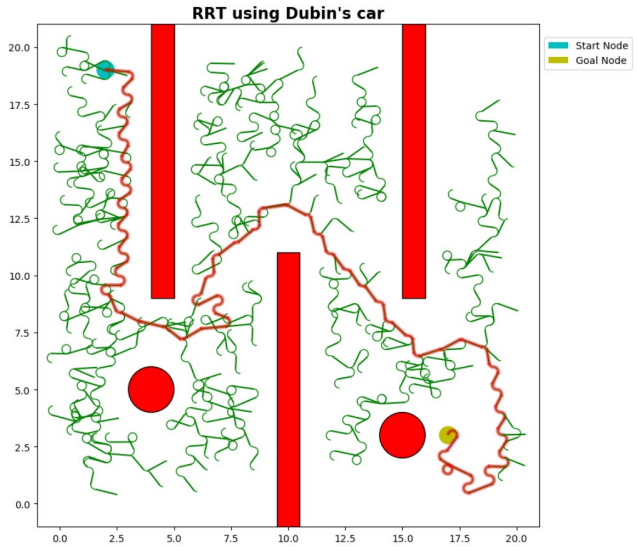

Figure 7. RRT with Dubin's Car - Iteration 1

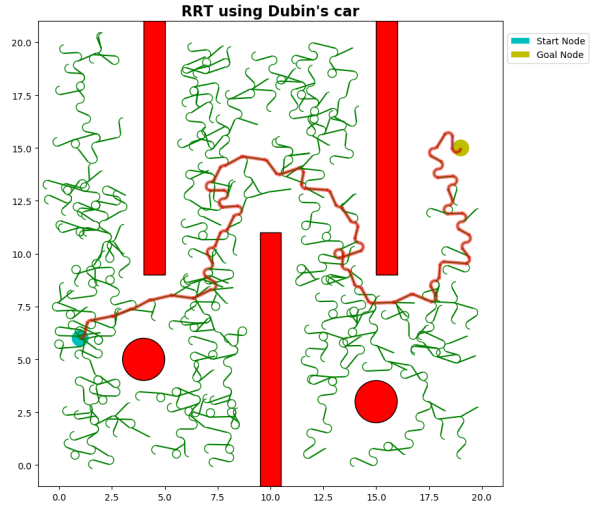

Figure 8. RRT with Dubin's Car - Iteration 2

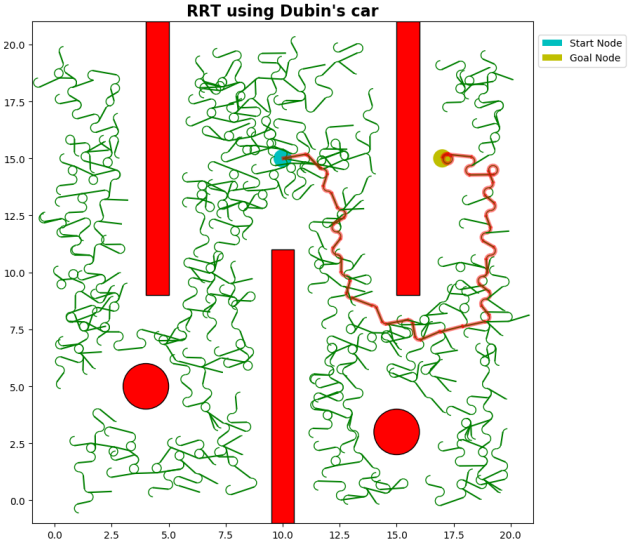

Figure 9. RRT with Dubin's Car - Iteration 3

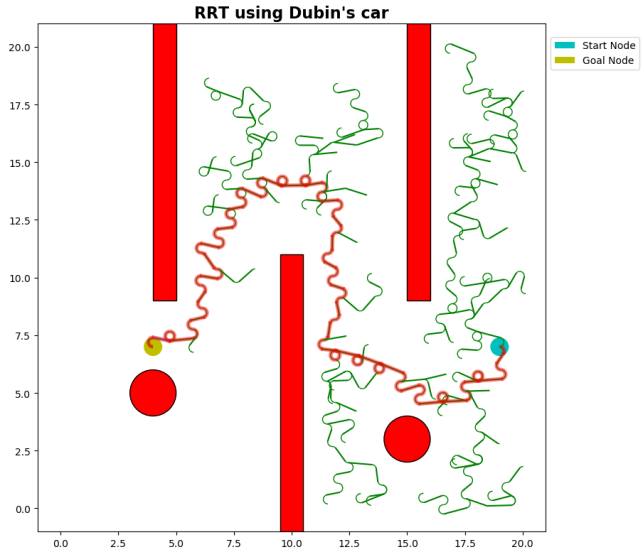

Figure 10. RRT with Dubin's Car - Iteration 4

4. Limitations

There are a few limitations when using the RRT algorithm for path planning. The primary one being that there is no heuristic to guide the creation of the new nodes. This leads to the graph exploring the space randomly which might even take it away from the goal configuration. This brings us to the second limitation of the algorithm which is that it is a time-consuming search. The random and unguided nature of the nodes being created creates loops in the final path.

The primary limitation of the Dubins curves approximation is that there is no reverse directionality. Due to this, the paths returned are not optimal.

5. Conclusion

We have used algorithms like RRT and Dubin's path to calculate the most optimum path between the start and the goal position. Based on our experience with the PRM done in our assignments we have concluded that RRT with Dubin's path gives faster results than the PRM. It has better collision avoidance and gets a continuous path with lesser edges which leads to a lesser total length of the path from start to the goal position. Also, we learned to integrate two algorithms to obtain the shortest path between two points.

6. Future Scope

To further enhance this work, we can implement RRT* or RRT*-. These algorithms use heuristics such as the path cost (which are computed based on the distances between nodes) and rewire the tree based on the heuristics to give a clutter free graph without any loops. Instead of Dubin's Car, a

Reed-Shepp car model can be used as it incorporates reverse motion too.

References

[1] A. K. Z. H. Iram Noreen, "A Comparison of RRT, RRT* and RRT*-Smart Path Planning," *IJCSNS International Journal of Computer Science and Network Security,* vol. 16, no. 10, p. 20, 2016.

[2] S. M. L. Valle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," [Online]. Available: http://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf.

[3] A. Giese, "A Comprehensive, Step-by-Step Tutorial on," [Online]. Available: https://gieseanw.wordpress.com/2012/10/21/a-comprehensive-step-by-step-tutorial-to-computing-dubins-paths/.