Final Report: Dynamic Dubins-curve based RRT Motion Planning for Differential Constrain Robot

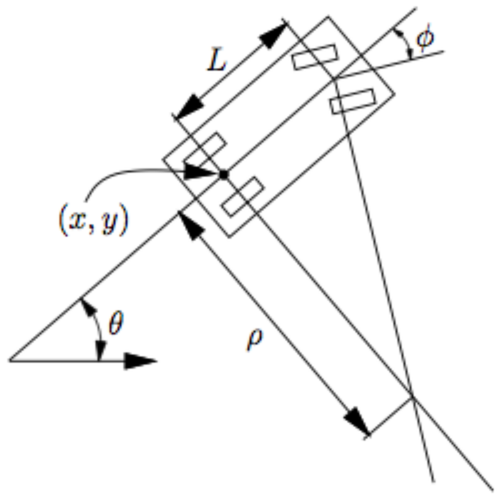
Abstract

This project develops a sample-based motion-planning algorithm for robot with differential constraints. In this problem, the robot needs to reach a destination without bumping into any polygon obstacles. And it also must obey the car's differential constraints, only moving forward or in its heading directions, and going straight, making left or right turns. In order to solve this problem, we uses RRT algorithm to explore the configuration space with its edges created by dynamic Dubins curve. After RRT tree is built, the shortest path is chosen as the optimal path from the initial region to the reward region. Dubins curve is one of the trajectories that car can drive on with differential constraint satisfied.

1. Problem

Since the robot obeys many different physical constraints, its movement is limited by its current angle and obstacles. To reach the reward region, the robot needs to move as a curve not a straight line.

The car can be imagined as a rigid body that moves in the plane. Therefore, its C-space is C = R2 × S1. A configuration is denoted by q = (x, y, θ). The body frame of the car places the origin at the center of rear axle, and the x-axis points along the main axis of the car. Let s denote the speed of the car. Let φ denote the steering angle. The distance between the front and rear axles is represented as L. If the steering angle is fixed at φ, the car travels in a circular motion, in which the radius of the circle is ρ.



x'= f1(x,y,θ,s,φ)
y'= f2(x, y, θ, s, φ)
θ'= f3 (x , y , θ , s , φ)

2. Methods

For the motion planning of differential constraint system, we combines Rapidly-exploring Random Tree algorithm with Dubins Curve algorithm. This Dubins-Curver-connecting RRT tree is built by connect the new sample point with the existing tree via dynamic Dubins curve.

Dynamic Dubins curve is a method that we generate a group of curves with random radius for Dubins curve, and then the best, that is the shortest curve would be chosen to the actual edge to be appended on the tree.

After time is out, The tree is established, the best path from root to the goal in the tree would be output.

2.1 Basic RRT

A Rapidly-exploring Random Tree (RRT) is a data structure and algorithm that is designed for efficiently searching non-convex high-dimensional spaces. RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly-chosen point to the tree. RRTs are particularly suited for path planning problems that involve obstacles and differential constraints. RRTs can be considered as a technique for generating open-loop trajectories for nonlinear systems with state constraints.

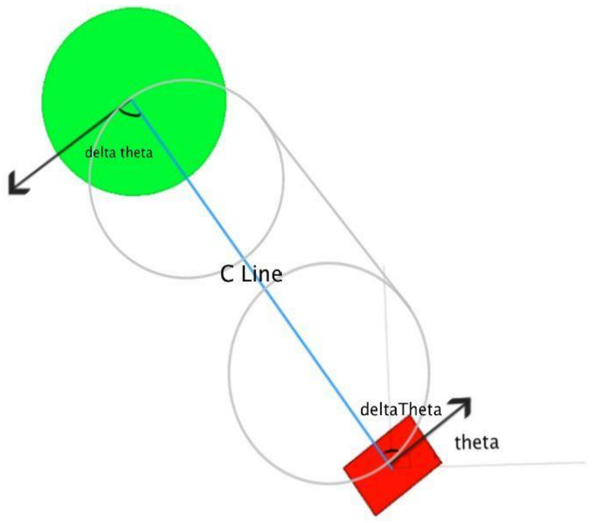
In this project, the whole algorithm is based on the structure of basic RRT algorithm.

The pseudo code of basic RRT tree T rooted at configuration qinit is like below:

```
Build_RRT(qinit, K, deltaQ){
      T.init(qinit);
      while(Time is not out){
            qrand = random_config();
            qnear = T.nearest_point_in_tree(qrand);
            qnew = new_config(qrand, deltaQ);
            if (line(qnear,qnew) collision free){
                  T.add_point(qrand);
                  T.add_edge(qnear,qnew);
            }
      }
      return T;
}
```

2.2 Guiding Sampling

In this project, we use the global random sampling method with a uniform distribution. It's probability complete and has good effect on the RRT algorithm. For the third parameter theta, we used the angle range from(-Pi/2,Pi/2) for random sampling. The limit of the angle sampling range can avoid creating a curve not sensible to a car-like robot. But different from the RRT with no constraints, it is very hard to get the exact sampling point inside reward region in the three dimensional space(x,y,theta). Even the it sampled correctly, the path is still waste lots of rotation.



To solve this problem, we introduce a Observer. The observer is used after the new sampling point added into the tree. The aim is connect the robot with the center directly. It works by checking whether this sampling point can reach the reward region directly. If the path between the sampling point and the center of region was collision free and can made a Dubins curve. Then connect the path and store it into the tree as a edge. The coordination of these two points are easily to get, but the theta will be hard to decide. We assign the angle of the center as:

$$\Delta\,\theta = slope(CLine) - \theta$$

This angle will make the path to be the best Dubins curve if there is a curve existed. And if there is no curve existed between the robot and the center, it will return to the loop of expending RRT tree.

2.3 Dubins Curve

Dubins Curve Type:

The original Dubins curve have different states for these two points. They are:

$$\{L_\alpha\, R_\beta\, L_\gamma,\ R_\alpha\, L_\beta\, R_\gamma,\ L_\alpha\, S_d\, L_\gamma,\ L_\alpha\, S_d\, R_\gamma,\ R_\alpha\, S_d\, L_\gamma,\ R_\alpha\, S_d\, R_\gamma\}$$

In our project, we try to make it easier by abstract them to two situation, which is

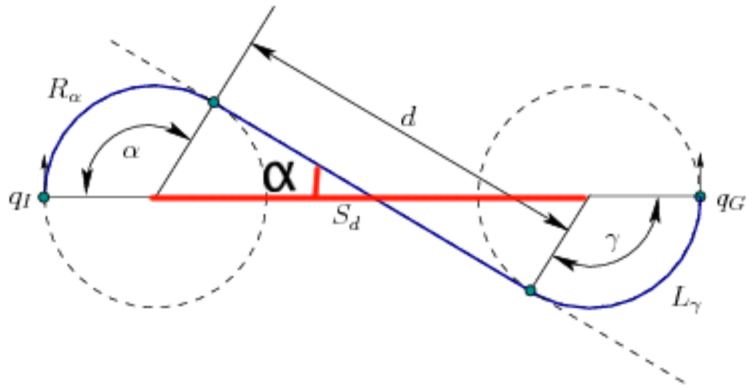
{co-orientation, anti-orientation}

In the co-orientation case, the tangent line will be parallel to the connection between two centers of Dubins circle. Which means the robot will leave the circle when the angle of it on the circle equals the slope of the connection. Then, it will follow this line until it reached the next circle. And the length of its straight movement is exactly the length of the connection line. Assume the length of connection between two centers is l.

$$leaveAngle = slope(l)$$

$$length = 2l$$

In the anti-orientation cast, the tangent line will be vertical with the connection between two centers. Then the length of this tangent line can be showed in the equation:



$$leveAngle = slope(l) \pm \arcsin\left(\frac{2dubin\ R}{l}\right)$$

$$length = 2\sqrt{\left(\frac{l}{2}\right)^2 - dubinR^2}$$

2.4 Curve-based RRT

Now we have basic RRT and Dubins curve, next step is to combine them together so that it generates the paths that car-like robot can run on. In the basic RRT algorithm, points in tree are connected by straight line and the obstacle collision detection is running on the staight line too. But in this project, there is not straight line. Every points are connected by Dubins curve. So, when we build the tree, we add curves as edges into the graph.

In this process, we also apply some strategies to optimize the RRT tree, like the guilding sampling, no aggressive turn control, tree depth control, etc.

The pseudo code of Dubins-curve-based RRT tree T rooted at configuration qinit is like below:

```
Build_RRT(qinit){
      T.init(qinit);
      wihle(Time is not out){
            qrand = guiding_random_config();
            qnear = T.nearest_point_in_tree(qrand);
            dubins_curve = generate_curve(qrand, qnear);
            if (dubins_curve exist){
                  T.add_point(qrand);
                  T.add_edge(dubins_curve);
            }
      }
      return T;
}
```

The qrand point is generated by the guiding random sampling strategy.

The qnear is chosen from the existing points in tree, which is the “nearest” point to qrand. This “nearest” is not just about the distance from point A to point B. Because the configuration space have x, y, theta, we need to take the angle into consideration.

In order to build a RRT tree with proper size(not too deep and too wide), we need to use the factors of tree to affect the connecting of sample points.

First, we want to control the depth of the RRT tree. This depth means the path length from the current point backward to the root of tree. In order to controller it, we design a strategy to take the depth value of the point in tree into consideration when we looking for the “nearest” point to sample point. It encourages sample point to connect to the point closer to root and with less path length. This added factor make the final path improved so much, because it avoids most sample nodes stick on the longest path, and generates new path with shorter path.

And also, in order to limit the width of the RRT tree, we use some random factor to multiply the depth. In this way, the locations new sample point attached could be normally distributed on all the locations of the tree. So, width of the tree is controlled.

The final method to calculate the “distance” from point in tree to sample point is like below:

$$distance = \mathrm{dist}_{\mathrm{straight}}(\mathrm{p,s}) + \mathrm{A} * \Delta\mathrm{o}(\mathrm{p,s}) + \mathrm{B} * \mathrm{rand}(0,1) * \mathrm{depth}_{\mathrm{tree}}(\mathrm{p})$$

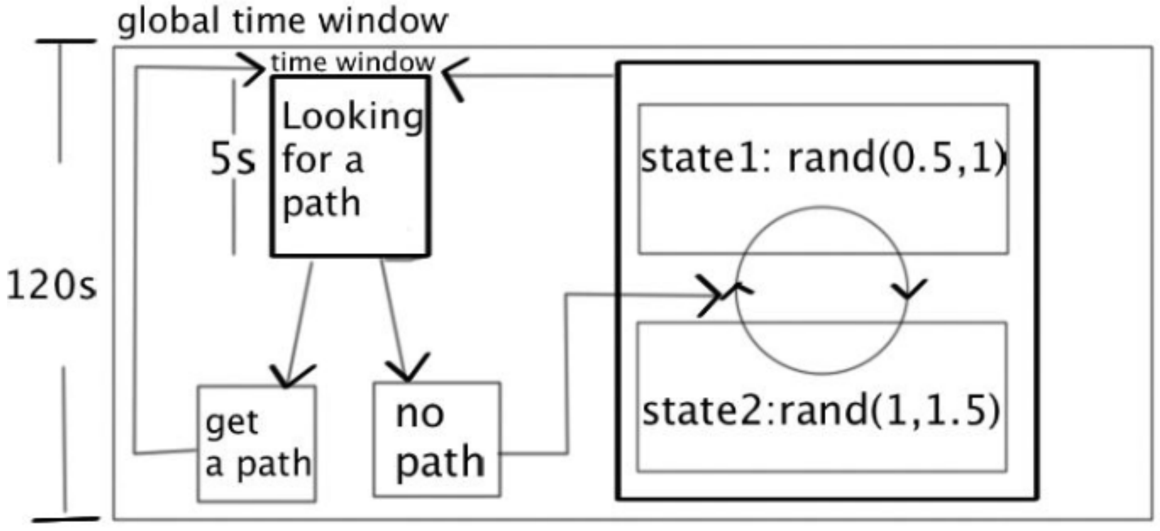
p: $point$ in tree
s: $sample$ point
o: $orientation$

Straight line distance, difference of orientations between sample point and point in tree, and the depth of point in tree all affect the building of RRT tree. note: the parameters A and B are used to scale the tree factors to the same order of magnitude.

2.5 Dynamic Dubins Curve

The dynamic here means change the Dubins radius during path planning. In this project we use two methods arrange the radius dynamically.

The first method is using the strategy design pattern to adapt different situation of the relatively free space. For example, in some situations the car can’t pass a narrow channel with a big radius but in other situations the small radius will slow down the path planning and make too regressive turning. Here we define a time window for 5 seconds. If the planner can not find a reasonable path reached the goal in 5 minutes, we change the radius but we still keep the tree.



The pseudo code of dynamic Dubins curve is as below:

```
time_window_start = time.now
while(Time is not out){
      code to add sample point and dubins curve into tree.
      if(Tree don’t have path to reward region){
            if(time.now - time_window_start > 5s){
                  change_to_another_dubins_radius_range();
                  time_window_start = time.now;
            }
      }
}
```

After defining the reasonable range of the radius. The second method can be used to look for the best performance curve between a small range of radius. In this method, we iterate a array which begins with the result of first method. The elements in the array will be incremented with a small value(such as 0.1). During the test we found even small variance of the radius has a great influence of the path efficiency.

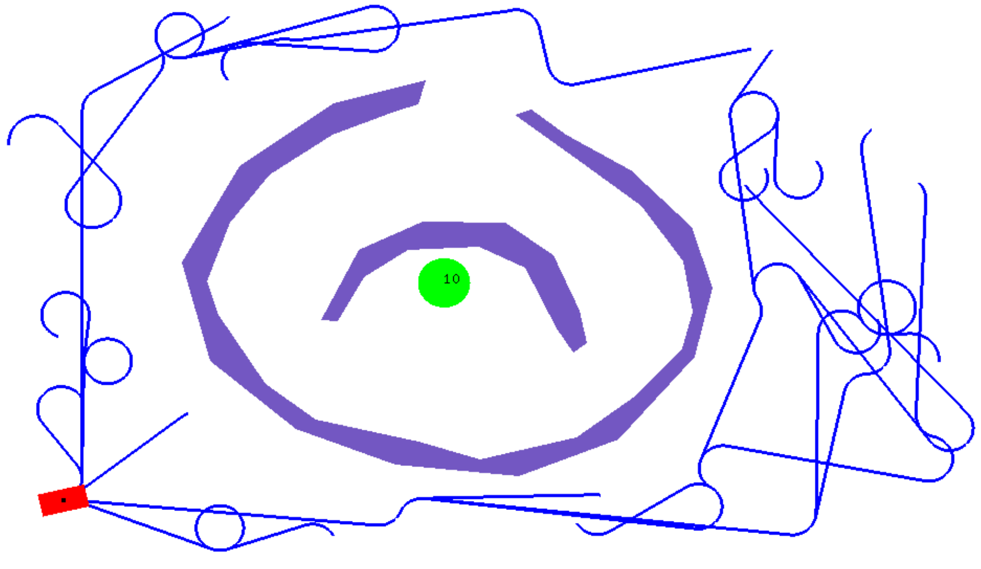
2.6 Solve Optimal Path

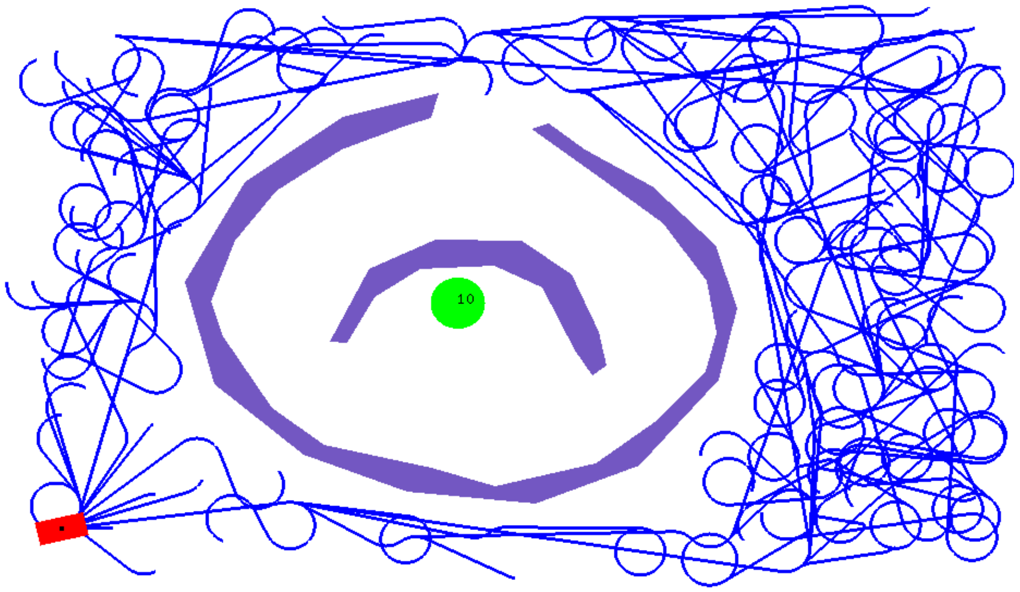
After we have the curve-based RRT in the free space, we need to find the shortest path from root to the reward region. If the time is long enough for the building of RRT tree, there should be many paths in tree connecting root and reward region. We iterate all this kind of path and sum the lengths of all edges in path, then choose the shortest as the best path.

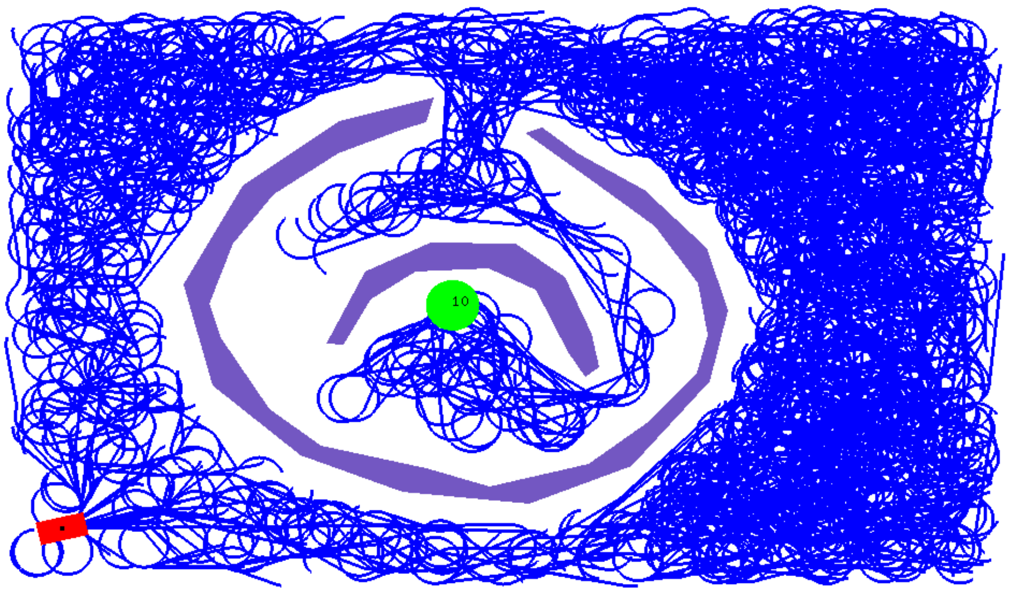
3. Experiments & Results

3.1 Dubins Curve Based RRT Tree

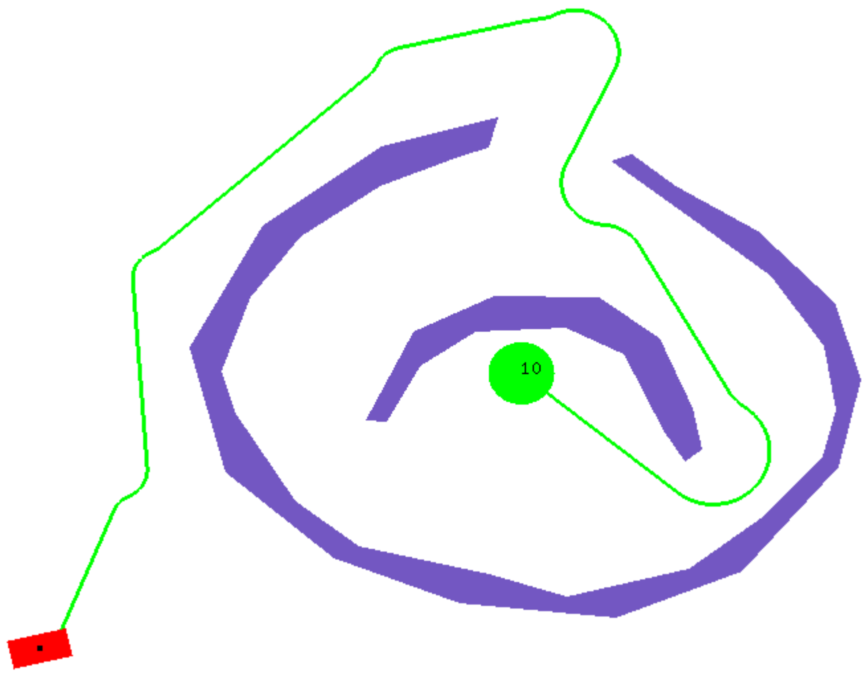
The pictures of Dynamic Dubins curve based RRT trees with different run time is as below: (their run times are 0.3s, 1s, 10s.)





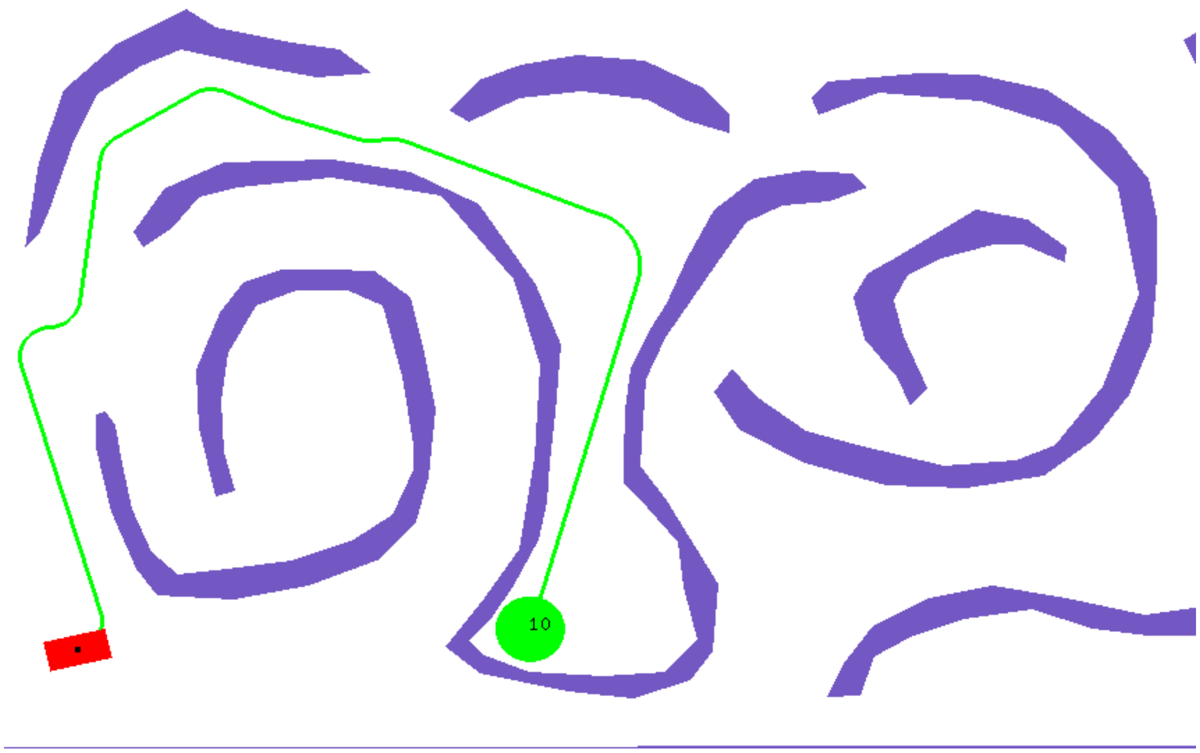


3.2 Optimal Path



Map1 test results:

```
adjustment of dubinsRs------------------------------------------ 1.484
arrival at the goal
 reachGoal path distance:61.2486
met a shorter path
arrival at the goal
 reachGoal path distance:62.2589
adjustment of dubinsRs------------------------------------------ 0.697213
arrival at the goal
 reachGoal path distance:66.3195
arrival at the goal
 reachGoal path distance:83.0606
arrival at the goal
 reachGoal path distance:70.2649
arrival at the goal
 reachGoal path distance:78.3265
adjustment of dubinsRs------------------------------------------ 1.44371
```



Map2 test Result:

test1:

```
adjustment of dubinsRs------------------------------------------- 1.4339
arrival at the goal
 reachGoal path distance:47.9594
met a shorter path
adjustment of dubinsRs------------------------------------------- 0.670922
adjustment of dubinsRs------------------------------------------- 1.38887
adjustment of dubinsRs------------------------------------------- 0.921272
adjustment of dubinsRs------------------------------------------- 1.26734
Motion planner has been run for 30.003197 seconds
```

test2:

```
adjustment of dubinsRs------------------------------------------- 1.15703
adjustment of dubinsRs------------------------------------------- 0.557396
arrival at the goal
 reachGoal path distance:45.8643
met a shorter path
adjustment of dubinsRs------------------------------------------- 1.19834
adjustment of dubinsRs------------------------------------------- 0.787025
arrival at the goal
```

```
 reachGoal path distance:46.1459
adjustment of dubinsRs------------------------------------------ 1.29464
Motion planner has been run for 30.000733 seconds
```

3.3 Result Analysis

The data above shows the motion planner running on the map1 and map2. The RRT tree expended very quickly but didn't get to the goal at the first time. After a certain time, the RRT can connect the reward region and begin to find the best path to the goal. That obeys the unform distribution which also get blocked where the collision happened. The narrow channel will slow down the expanding and it go across the channel if there are enough time.

The result outputted by terminal shows the dynamic Dubins works and help reconstructing the tree very good. In the Map2, test2: we can see the robot can not find a correct path with 1.15 in 5 seconds, so the program change the strategy to another side, which got the new radius 0.55. After running few seconds, the path was founded. That shows this algorithm resolve the problem that car-like robot can't work out a good performances in different free spaces.

3.4 Demo Videos

Demo Video 1

Demo Video 2

4. Conclusion

The RRT algorithm have a very good performance to explore the free space rapidly. The dubins curve can be used to build an effective curve for differential constraint robot. Our algorithm utilizes both of their benefits and combine them together with some creative strategies such as guiding sampling, dynamic adjustment of Dubins radius, etc. The results are very decent for most environments. However, for some environments with very long narrow channels and sharp turns, it is very hard for the Dubins-curve RRT tree to reach the goal in our tests.

The future direction would be to refactor our code to make it running faster to do more sampling, so that we can have better path on the tree.