**CPSC 8810: Motion Planning - Final Project Report**

**Project Title:** Nonholonomic Motion Planning for a Car in 2D Environments using RRT\*.

Team Members: Kalpit Madhusudan Vadnerkar, Vasanth Seethapathi

**Overview of the Project:**

The objective is to plan the motion of a car in 2D environment that is not confined to the roads,i.e an off road environment. The car is considered to be a Dubins Car which is subject to non-holonomic constraints where RRT\* algorithm is implemented for path planning, which can effectively generate paths to navigate through the 2D environment.

**Description of your implementation:**

**Status of Code:**

The RRT\* algorithm is created and integrated to run with the 2D environment. The code performance is compared with the other available path planning algorithms to find out the which is the optimized algorithm for Dubin’s Path planning.

By executing the RRT\* algorithm, we obtain an optimized path from start location to goal location of the vehicle based on the lowest cost of the generated paths as per the sampling and iteration.

**Usage of Third-Party Library:**

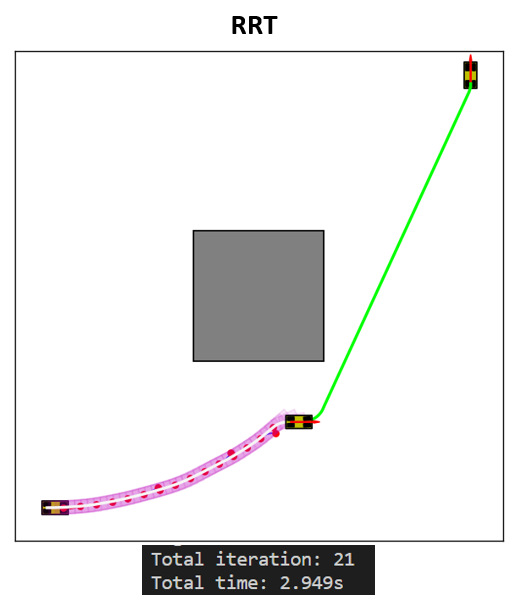
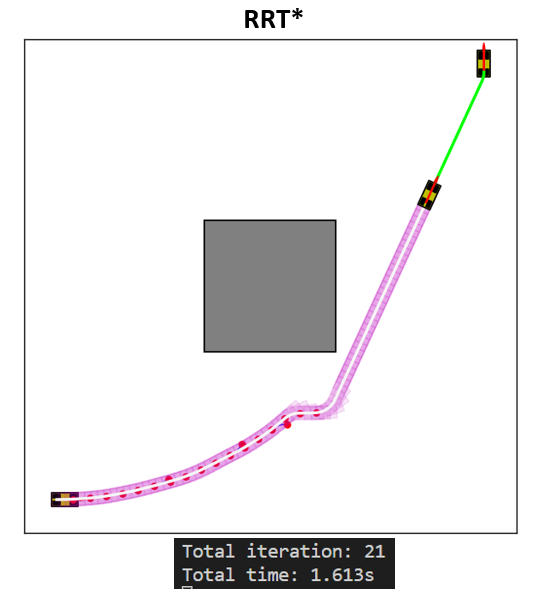
For the baseline version of the project, we have considered the environment and utility function code from this below Git path -[**https://github.com/jhan15/dubins\_path\_planning**](https://github.com/jhan15/dubins_path_planning)

The other algorithms in the existing directory of the Git folder will be used for comparison purpose.

**Results:**

In a 2D environment, the RRT\* algorithm has been shown to produce high-quality paths with efficient search times, making it a popular choice for path planning tasks in a variety of applications, such as robotics and autonomous vehicles. However, the performance of the algorithm can be affected by factors such as the complexity of the environment and the parameters used in the algorithm, and thus requires careful tuning to achieve optimal results.

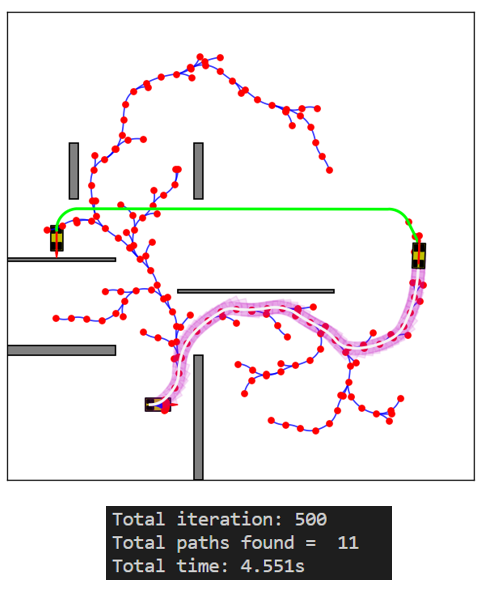
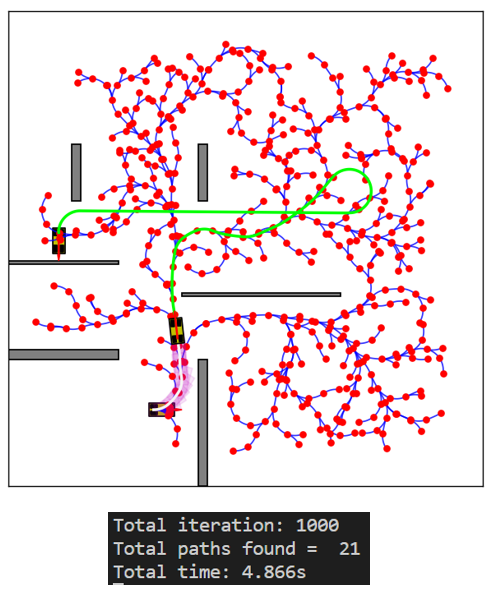
In the figure(1), we can see RRT\* outperforms RRT algorithm where the environment is a bit simple and the vehicle had to move from one vertex of square to another diagonally.

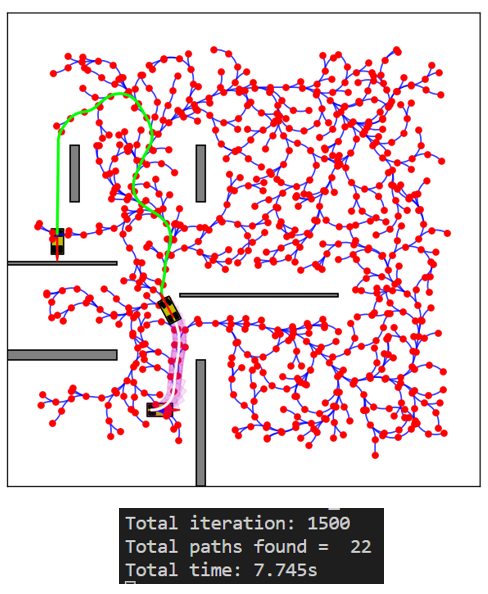
** **

**Figure (1): Comparison between RRT and RRT\* on a simple environment**

The path exploration of algorithm is mainly based on the number of iterations. It has a great influence on the percentage of the map being explored for trajectory generation.

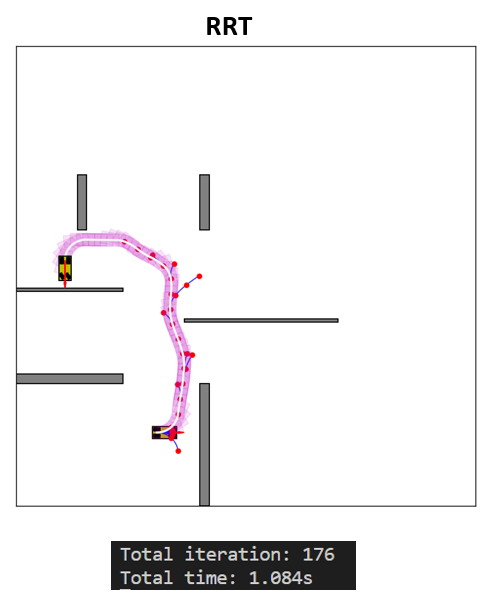
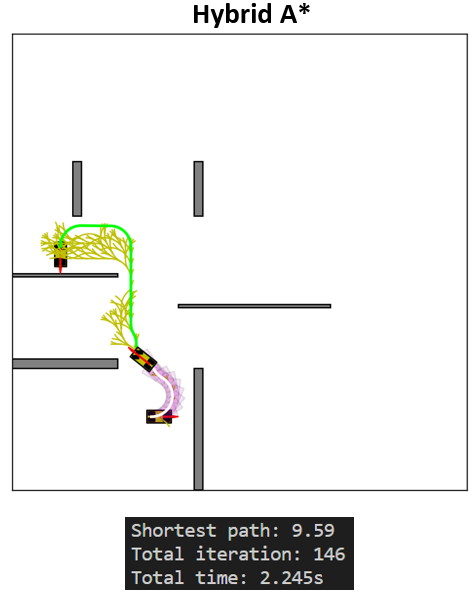
The figure (2) shows how the paths are getting optimized and efficient based on the number of iterations the algorithm is executed. As the number of iterations increase, the algorithm has better chance to find a shorter path with the cost of higher execution time. There should be a tradeoff between the generated path’s distance and execution time of algorithm while hyper tuning the parameters.

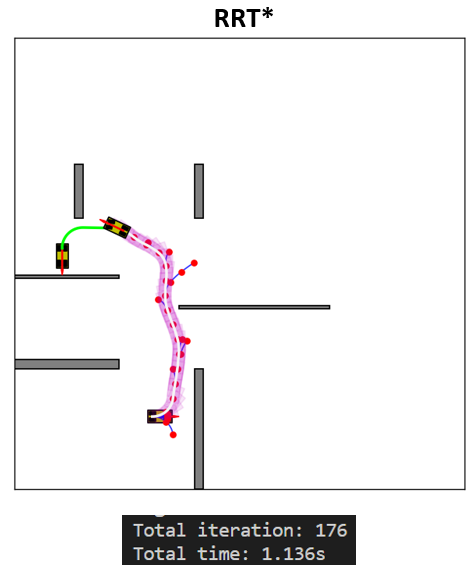
 



**Figure (2): Effect of Iteration on the execution Time and Tree exploration**

The figure (3) shows how various algorithms perform on the same scenario. As per the data collected on this instance – we are able to see Hybrid A\* having an execution time twice more than the RRT and RRT\* algorithm.



**Figure (3): Comparing Performance with other algorithms**

**Future Work:**

1. **Multi-Objective optimization –** RRT\* implementation should have multiple optimization constraints – such as Time to reach the goal, distance traveled and the execution time for algorithm.
2. **Dynamic Obstacle Avoidance -** To make the algorithm capable enough to handle dynamic obstacles into the environment as RRT\* is capable for adapting to changing environments.
3. **Non-Holonomic Variants Experiment -** We have planned to implement the similar algorithm in the Reed Shreep Car to understand how the Reverse direction have a positive impact on the final Trajectory generation.
4. **Extend to 3D environments –** Update the code to include the 3D space for the environment and expand the tree accordingly.