Robot Motion Planning - Homework IV

David Akhihiero

November 3, 2024

1 Introduction

In this assignment, I implemented the Rapidly-Exploring Random Trees (RRT) algorithm using Dubins' paths as a local planner instead of straight-line paths. I wrote four functions that compute the LSL, LSR, RSL, and RSR Dubins' paths given an initial and final configuration and another that can select the shortest paths for local RRT planning.

2 Dubins' Paths Functions

I implemented four functions that compute the LSL, LSR, RSL, and RSR Dubins' paths given an initial and final configuration (x, y, θ) Giese (2012). The functions return the sequence of configurations (either LSL, LSR, RSL, or RSR) and the length of the path. Each function takes as input the initial and final configurations (x_init and x_goal), the minimum turning radius of the robot, r, and boolean arguments to decide if the Dubins' circles and tangents will be plotted for visualization. I wrote a script (called from the main script) for a user to test each function using the mouse. To select an initial configuration, the user needs to click two points. The first point clicked is the (x, y) coordinate for the configuration and the second point clicked is used to calculate the orientation, θ of the configuration using the arc-tangent function. The goal configuration is selected in the same way. After both configurations have been selected, the Dubins' path is plotted. Figure 2 shows some example Dubins' paths for LSL, RSR, LSR, and RSL.

I also wrote a function called $shortestDubins()$ that takes an initial and final configuration and returns the shortest Dubins' path between them that is collision-free and the length of the path. I modified the $collisionFree()$ function to account for the Dubins' path curves. A path is collision-free if the curves and straight lines in the Dubins' path do not intersect an obstacle. I wrote a script (called from the main script) for a user to test the $shortestDubins()$ function using the mouse. To select an initial configuration, the user needs to click two points. The first point clicked is the (x, y) coordinate for the configuration and the second point clicked is used to calculate the orientation, θ of the configuration using the arc-tangent function. The goal configuration is selected in the same way. After both configurations have been selected, the shortest Dubins' path is plotted. Figure **??** shows that the shortest path between an initial configuration $(0.759, 0.368, 55.305°)$ and a goal configuration $(0.229, 0.300, 137.386°)$ is an LSR configuration with path length 0.629 when the turning radius is 0.05.

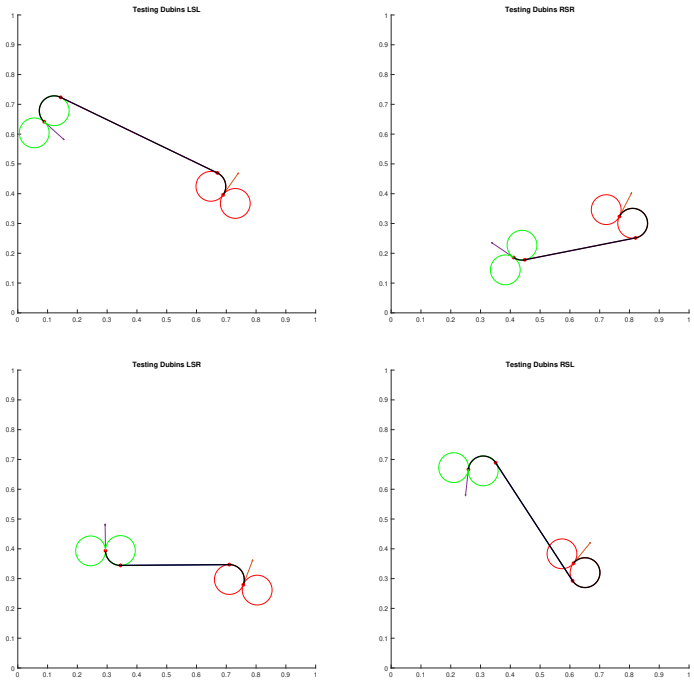
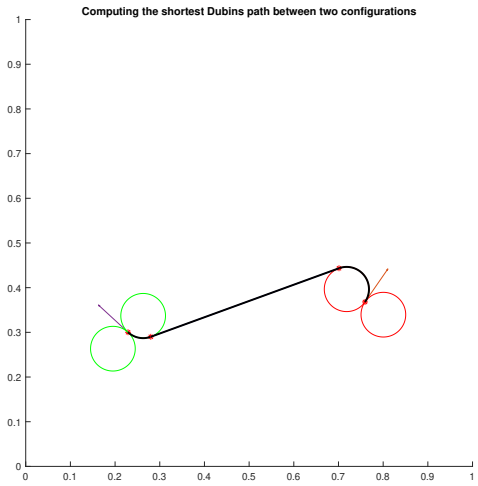


Figure 1: Dubins' Paths for LSL, RSR, LSR, and RSL for some random initial (in red) and final (in green) configurations with a turning radius of 0.05




```
Testing for Shortest Dubins Path
Click two points to indicate the position and angle of the start configuration.
Start configuration is at (0.759, 0.368). Click another point to determine the angle
Start configuration is (0.759, 0.368, 55.305 deg)
Click two points to indicate the position and angle of the goal configuration.
Goal configuration is at (0.229, 0.300). Click another point to determine the angle
Goal configuration is (0.229, 0.300, 137.386 deg)
The shortest path length is 0.629 with combination LSR
```

Figure 2: Shortest Dubins' Paths (LSR) between an initial configuration (in red) $(0.759, 0.368, 55.305°)$ and a goal configuration (in green) $(0.229, 0.300, 137.386°)$ with a turning radius of 0.05

3 Rapidly-Exploring Random Trees (RRT) With Dubins' Paths

I modified my $RRT()$ function to implement the RRT algorithm using Dubins' paths with turning radius $r = 0.01$. The graph was stored in a $PGraph$ object. I used a 2D $PGraph$ object and the orientation for each node was stored in the node's data field along with the cost and parent of the node. I chose this option rather than use a 3D graph. I modified the $sampleFree()$ function to sample orientation as well (scaled to 2π). The function generated 800 nodes for both environments, the radius used for the goal region was 0.05 and the seed for the random number generator was 7. The $steer()$ function step size was 0.05 and it computes x_new along a straight-line path from $x\_nearest$ to x_rand.

I used the smallest Dubins' cost between two nodes to select $x\_nearest$ and not the Euclidean distance by modifying the $Near()$ function to find the shortest Dubins length between a node v and all the other nodes in the graph and sort them by distance to get $x\_nearest$.

The function started the tree's root from a q_{init} specified by a user using two points as described above. My RRT used straight lines instead of Dubins' curves between two nodes if the distance between those nodes is less than $4r$ (0.04) because the four Dubins' paths will fail in this scenario. It also stored the parent and the cost of each node in the graph using the $PGraph\ setvdata()$ function and the Dubins' path type of each edge using the $PGraph\ setedata()$ function. This data was used to recover the path (and Dubins' path type) from the start node to the node in the goal region with the smallest cost from q_{init} to q_{goal} (also user-specified using two points). I plotted the RRT with Dubins' path edges instead of the default straight-line edges

Example paths for both environments are shown in Figure 3

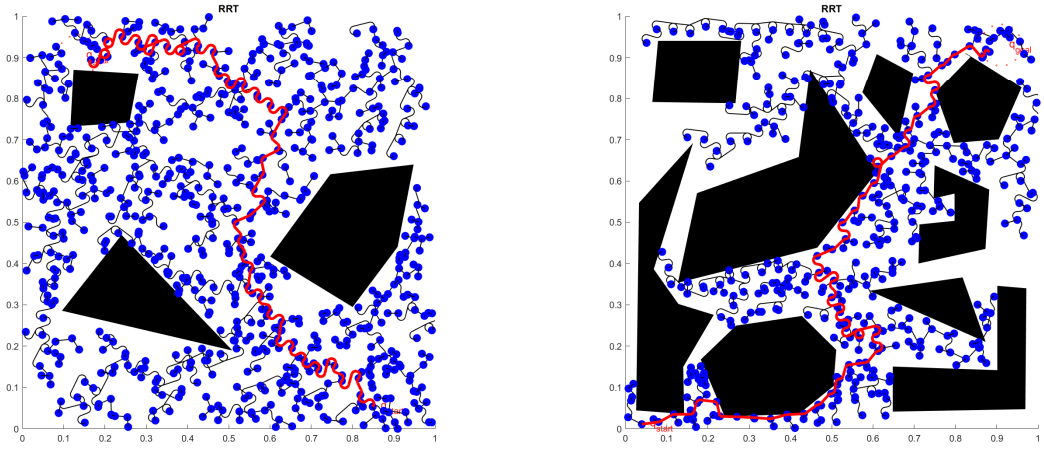


Figure 3: Dubins' Paths in an RRT graph from a start configuration (user-specified) to a goal configuration (user-specified) for sparse and dense environments.

References

Giese, A. (2012). A comprehensive, step-by-step tutorial on computing dubin's curves.