Path Planning Algorithms for Mobile Robots
CSE 523 Advanced Project Report

Professor Nilanjan Chakraborty

Professor Dimitris Samaras

Shrinand Thakkar 111483215

May 19, 2018

1 Introduction

A robot that is capable of locomotion is a mobile robot, and the path planning of the mobile robot has two goals in the most abstract level, Collision-free trajectories and reaching the goal location as fast as possible.

The problem of path planning of any mobile robot is stated as, given initial configuration of the robot, the final or goal configuration of the robot, A geometric description of the robot and A geometric description of the world.

So, the main goal of this project is to find a path that moves the robot gradually from start to goal while never touching any obstacle.

The Algorithms that I have implemented in Python assumes that the robot is a point robot. Although the path planning problem is defined in the regular world (which I have used in the Gazebo Simulation), it lives in the configuration space while doing calculations. The configuration space of the robot is a specification of the positions of all robot points relative to a fixed coordinate system.

I have worked on the path planning for the Differential drive robots, implementing Dubins Curves and RRT (based on Dubins Curves) and simulated them in Gazebo using ROS nodes.

2 Implementation

2.1 Dubins Path Algorithm

Dubins Path Algorithm is optimal path algorithm between two configurations traveled by a Dubins Car where Dubins Car is a forward only car which moves at constant forward speed v, haves a maximum steering angle ϕ_{max}, which results in a minimum turning radius r_{min} with the angular velocity $\omega = \frac{v}{r_{min}}$

Basically, if (x, y) is the initial position of the car with θ as the heading angle, our x coordinate changes as a function of $cos\,(\theta)$ and our y coordinate changes as a function of $sin\,(\theta)$. Therefore, we can say that,

$\dot{x} = cos\,(\theta)$
$\dot{y} = sin\,(\theta)$
$\dot{\theta} = \omega = \frac{v}{r_{min}}$

All the paths traced out by the Dubins car are combinations of essentially three controls, “turning

left at maximum" (L), "turning right at maximum" (R), and "going straight" (S). Lester Dubins has proved in his paper that there are only 6 combinations of these controls that describe all the shortest paths, which are: RSR, LSL, RSL, LSR, RLR, and LRL.

I implemented the Dubins Path considering a Differential Drive Robot from one configuration to the other with combinations of Curves and Straight Lines in Python.

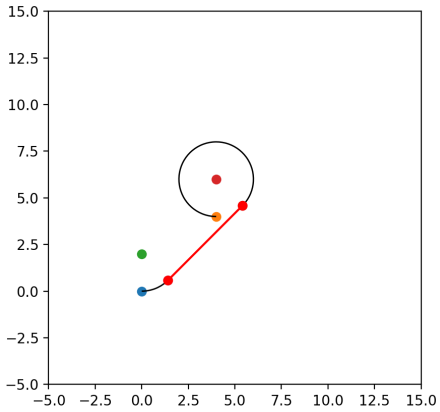

Figure 1: LSL from (0,0,0) to (4,4,0)

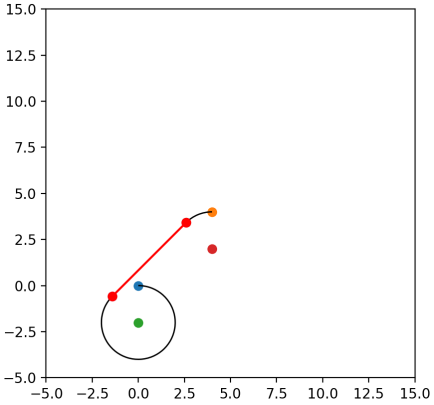

Figure 2: RSR from (0,0,0) to (4,4,0)

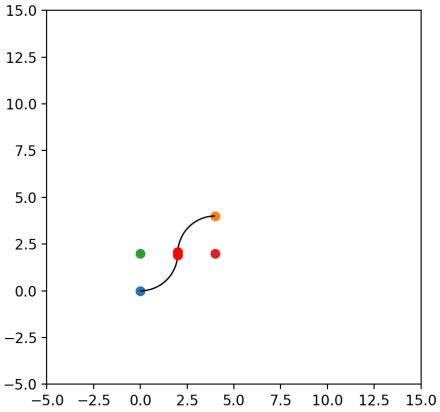

Figure 3: LSR from (0,0,0) to (4,4,0)

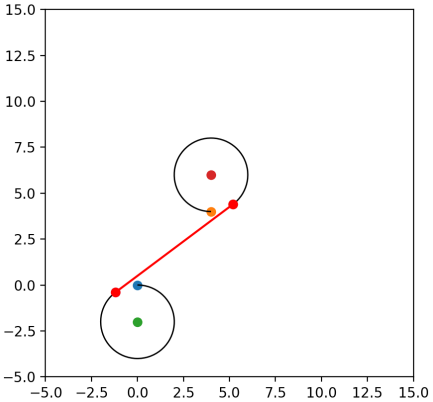

Figure 4: RSL from (0,0,0) to (4,4,0)

Above figures shows all combinations of curves and straight lines between the same configuration with the minimum turning radius of 2 units.

Observations As we can see from the figures (1,2,3 and 4), the optimal path in this particular example would be traveling Left with maximum steering for 87 degrees with positive X-axis then traveling straight for 0.159 units and then traveling Right with maximum steering for 87 degrees, which resulted in a total distance traveled, 6.28 units to reach our destination.

Simulating it with ROS and GAZEBO

- After implementing the Dubins Curves on python, I simulated the Dubins Path algorithm on Gazebo.
- I have used the Pioneer 3dx Differential Drive Robot which follows the path computed by the dubins algorithm.
- I have used various ROS nodes, Odometry Node (to get the current (x,y,θ) configuration of the Robot, Twist Node (to publish to the $cmd-vel$ the change in configuration over the next time step).

Challenges on Gazebo

- Since the Dubins car travels at constant velocity, the controlling of the car at the end of any of Curve and Straight line would result in slight deviation from what is computed by the algorithm.
- This resulted in a slight error with the goal configurations.

2.2 Dubins Curve based RRT(Rapidly Exploring Random Trees) Algorithm

Rapidly Exploring Random Trees A Rapidly-exploring Random Tree (RRT) is a data structure and algorithm that is designed for efficiently searching high-dimensional spaces by randomly building a space-filling tree. The trees in RRT's are constructed incrementally in a way that the expected distance of a randomly-chosen point to the tree as of now reduces quickly.

RRTs are particularly suited for path planning problems that involve obstacles in the environment and nonholonomic or kinodynamic differential constraints on the drive.

Dubins Curve based RRT(Rapidly Exploring Random Trees) Dubins Path provides optimal path from one configuration to the other but it cannot deal with any environment which has obstacles. So Dubins Curve based RRT generates the RRT and replace the branches with the dubins curves.

Although the Dubins Algorithm works optimally but the Dubins based RRT is a non optimal algorithm since it randomly selects the points in the environment and does not updates any nodes with their parents later, which results in non optimality.

I have implemented RRT which is based on step increase, which means that every node randomly selected would be at a distance of one step size from the tree nodes as of now.

The RRT algorithm generally avoid obstacles when it randomly selects a point in the environment, as in, if the point or the path to that points results in collision with any obstacles, the algorithm would not use that random node and will create a new random node.

The python implementation of the Algorithm assumes that the robot is a point which simplifies the mathematics behind the algorithm. To consider this case, the obstacles are applied with the Minkowski Sum of the size of Obstacles and the original size of the robot.

I have also implemented Minkowski Sums for basic obstacles which helps in considering the robot as a point robot in the configuration space.

Observations As we can see from figures above, the RRT avoids the obstacle and creates a path from start to goal but since it randomly selects the point with any replacement at later, the path is not optimal and so there can be many path for the same configuration space.

Simulating it with ROS and GAZEBO

- After implementing the Dubins Curves based RRT on python, I simulated the Dubins Curve based RRT algorithm on Gazebo.
- I have used the Pioneer 3dx Differential Drive Robot which follows the path computed by the algorithm.
- I have used various ROS nodes, Odometry Node (to get the current (x,y,θ) configuration of the Robot, Twist Node (to publish to the cmd_vel the change in configuration over the next time step).

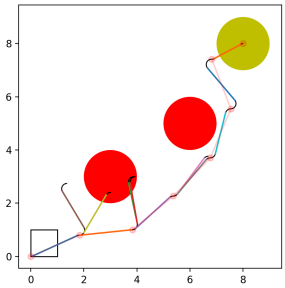

Figure 5: RRT from (0,0,0) to (8,8,0) with two red obstacles at (3,3) and (6,5)

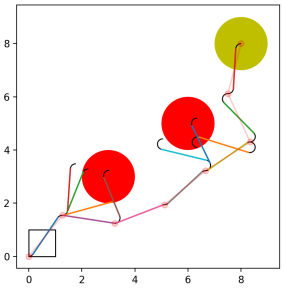

Figure 6: RSR from (0,0,0) to (8,8,0) with two red obstacles at (3,3) and (6,5)

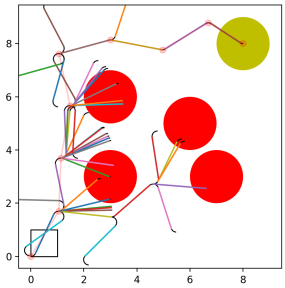

Figure 7: LSR from (0,0,0) to (8,8,0) with four red obstacles at (3,3) and (6,5), (7,3) and (3,6)

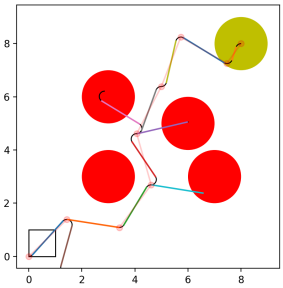

Figure 8: RSL from (0,0,0) to (8,8,0) with four red obstacles at (3,3) and (6,5), (7,3) and (3,6)

Above figures shows all combinations of curves and straight lines between the same configuration with the minimum turning radius of 2 units.

Challenges on Gazebo

- Since the Dubins car travels at constant velocity, the controlling of the car at the end of any of Curve and Straight line would result in slight deviation from what is computed by the algorithm.
- And since there are many Dubins Path from one point to the other, this accumulates a slight more error than the Dubins Path Algorithm.

2.3 Future Work

- Putting the Dubins Curve based RRT algorithm on real-time differential drive (P3dx) mobile robot.
- Work on optimization of Dubins Based RRT algorithm, implementing RRT* algorithm.
- Research on other optimal path policies for mobile robots.

2.4 References

Anytime Motion Planning using the RRT*, *https* : //*personalrobotics.ri.cmu.edu*/*files*/*courses*/*papers*/*Karaman*11
anytimerrtstar.pdf

The RRT Page,*http* : //*msl.cs.uiuc.edu*/*rrt*/*about.html*

A Comprehensive, Step-by-Step Tutorial to Computing Dubin's Paths, *https* : //*gieseanw.wordpress.com*/2012/10/21
comprehensive − *step* − *by* − *step* − *tutorial* − *to* − *computing* − *dubins* − *paths*/

Minkowski Sums and Offsets of Polygons, *https* : //*resources.mpi*−*inf.mpg.de*/*departments*/*d*1/*teaching*/*ss*10/*Semi*

Robot Motion Planning, *http* : //*ais.informatik.uni*−*freiburg.de*/*teaching*/*ss*11/*robotics*/*slides*/18−
robot − *motion* − *planning.pdf*