Motion Planning for a Non Holonomic Car using RRT

- An Approach to Autonomous Parking

AKSHAY DESAI
ROHITH KRISHNAN

4/29/2018
SECTION 1: INTRODUCTION

1.1 WHAT IS THE PROBLEM?

In this project, we consider motion planning for a non-holonomic mobile robot (car-like robot). The objective is to develop and implement a path planning system that can be used for a mobile robot to take it from an initial configuration to a goal configuration. The application of the planning algorithm is mainly focused towards autonomous parking of cars. This project is to be used as a broad representation of the solution to the problem.

1.2 PROPOSED SOLUTION

After assessing the various options available for this type of non-holonomic planning, we decided to implement a Rapidly-exploring Random Tree (RRT) with a Dubins car model for the maneuver till the parking space. Local navigation into the parking space is done using a Type 1 maneuver, which completes the parking maneuver of the vehicle.

1.3 RRT

A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem. It is this algorithm that we have chosen to implement for our application.

1.4 DUBINS CURVES

Dubins path or Dubins curves is a geometrical concept which describes the shortest curves that connect any two points in the two-dimensional Euclidean space. The constraints imposed on such a model are that the curvature of the path and with prescribed initial and terminal tangents to the path, and an assumption that the vehicle traveling the path can only travel forward. In 1957, Lester Eli Dubins (1920–2010) proved using tools from analysis that any such path will consist of maximum curvature and/or straight line segments. In other words, the shortest path will be made by joining circular arcs of maximum curvature and straight lines.

1.5 NON HOLONOMIC MANEUVER FOR LOCAL NAVIGATION – TYPE 1

Since it is highly likely that an RRT will find a non-optimal path, it is not desirable to use the RRT generated path to execute a parking maneuver. Hence, an RRT is used to merely guide the vehicle till it is near the available parking space, where a non-holonomic maneuver called the Type 1 maneuver is used to execute the final parking maneuver.
SECTION 2: IMPLEMENTATION

In a parking lot, the general behavior of a driver is to first look for a parking space. Once the driver has come close that spot, he/she then plans the local maneuver into the parking space. To accomplish the global planning, we use a simple RRT with a Dubins car model to navigate close to the parking space. The RRT was implemented by carefully studying the steps of implementation. The following pseudocode gives a basic structural understanding of the algorithm:

Algorithm BuildRRT  
Input: Initial configuration $q_{init}$, number of vertices in RRT $k$, incremental distance $\Delta q$  
Output: RRT graph $G$

$G$.init($q_{init}$)

for $k = 1$ to $k$
    $q_{rand} \leftarrow$ RAND_CONF()
    $q_{near} \leftarrow$ NEAREST_VERTEX($q_{rand}, G$)
    $q_{new} \leftarrow$ NEW_CONF($q_{near}, q_{rand}, \Delta q$)
    $G$.add_vertex($q_{new}$)
    $G$.add_edge($q_{near}, q_{new}$)

return $G$

Using the pseudocode as a reference, we implemented the following key functions that were used to build the basic RRT. The following are the main functions that constitute the algorithm:

1. Initialize (__init__)  
   This function initializes key parameters such as the start, end nodes, the lower and upper limit of the area, the percentage bias of searching towards the goal and the number of iterations the search has to take place.

2. Expansion  
   This function creates a list of all the nodes that are getting expanded in the search tree. Every time a random point is picked in the given area, its nearest neighbors are inspected and the index of the neighbor closest to the randomly generated node is found. Another function 'carnode' is called to find the most feasible maneuver between two given points amongst the Dubins six options. If the path is collision free, the node is added to the list of all the feasible nodes and stores the parent of the previously expanded node.
3. **Carnode**

This function calls the Dubins curve function from another file to generate the optimal combination of maneuvers to get the car from point A to B. This newly generated node is expanded similarly.

4. **Random_Point**

This function randomly picks a point from the given area, while picking nodes oriented towards the goal based on the specified goal rate.

5. **Last_Index**

This function carries out three main functions:
- It finds the distance of each of the nodes in the list from the goal node.
- It checks which of the listed nodes fall within the acceptable goal radius and stores the indices in a list
- Among the stored goal radius nodes, the node with the least cumulative path cost is expanded so that the cumulative cost of the path from the start to the goal is minimum.

6. **Pathgen**

Once the goal region has been reached, this function is called to start from the goal and trace the path back to the start. Remember that we had stored the parent node of every expanded node. These parent nodes are used to trace back the path to the start.

7. **Nearest Node**

This function takes as input, a node and a nodelist and returns the nearest neighbor from the nodelist to the node. Although simple, this is a key function that effectively expands the RRT while keeping it connected to the start.

8. **Collision**

This function checks if the given node collides with the given list of obstacles. To make this function faster and easier to compute we use.

9. **Parking Space simulation**

The parking space was created using the circular obstacles. A parking space was simulated using these obstacles.

10. **Inputs and Outputs to the RRT**

The inputs given to the RRT are the start and the goal position. Note that the position includes the x,y coordinates and the orientation of the vehicle. Using this information, the algorithm segments
the search into two halves. The first half implements the RRT with Dubins car model till an offset position from the parallel parking spot. The vehicle then moves from the offset position to the actual spot by executing the type 1 maneuver.

11. **Simulation:**

A vehicle model is characterized by its width, length and the heading direction. Since, implementing a rectangular region for the vehicle proved to be challenging, we used an arrow to depict the heading direction of the vehicle while specifying its width and height.

### SECTION 3: RESULTS

The results are presented in a sequence that represents the development phases.

1. **Simple RRT implementation**

   The goal biased sample rate chosen for the scenario is 12 percent.
2. **RRT with Dubins Implementation**

This search was run with a goal bias rate of 10 percent.

![Figure 3 RRT implementation along with the Dubins curve to describe vehicle motion](image1)

3. **Type 1 Maneuver**

This is a test case of the Type 1 maneuver with the start node $\rightarrow (9,8)$ and goal node $\rightarrow (18,8)$

![Figure 4 Trajectory of the Type 1 maneuver](image2)
4. **RRT with Dubins model and Type 1 maneuver**

This search was run with a goal bias rate of 10 percent.

![Figure 5](image)

**Figure 5** RRT implementation with Dubins model and Type1 maneuver

### SECTION 4: OBSERVATIONS AND FUTURE WORK

1. It is observed that the RRT algorithm is highly likely of finding a non-optimal path. Furthermore, experimental research has shown that under mild technical conditions, the cost of the best path in the RRT converges almost surely to a non-optimal value. For that reason, it is desirable to find variants of the RRT that converges to the optimum. This behavior has been observed with our implementation as well, hence there is a necessity to implement an optimized version such as an RRT* for real world parking applications.

2. The use of Dubins cars as model to represent a non-holonomic vehicle model is seen to be constrictive and mis-representation of the parking problem. Relaxing the vehicle path constraints by using a Smooth Car model would be necessary to find a more optimal path.

3. The Type 1 non-holonomic maneuver is a good representation of the final parallel parking maneuver. It shows good similarity with the parking trajectory typically followed by human drivers.
5. REFERENCES


[3] Atsushi Sakai, GitHub repository - AtsushiSakai/PythonRobotics

[4] Lester Eli Dubins, On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents, 1957