Computational Robotics, Fall 2019

Lab 2 - Slide Supplement

Name: Andrew Choi, UID: 205348339

Github Link: **https://github.com/QuantuMope/ucla-computational-robotics**

1 Preliminaries

- Collaborators: Yuki Shirai, Yusuke Tanaka, Viacheslav Inderiakin
- All members made equal contributions of 25%

2 Problem Statement

The goal of this lab was to build autonomy into a two-wheeled non-holonomic differential drive robot. Using this robot model, the robot was assigned the task of navigating through a simulated parking lot environment and parking in the proper assigned space. As the robot resided within a continuous state space environment, sampling-based motion planning algorithms, RRT and RRT*, were employed.

3 Mathematical Setup

Before constructing the RRT tree, a metric for defining the nearest node as well as the drive policy was first defined with the nearest node metric defined as the node in the tree with the minimum L2 norm as shown below.

$$x_{nearest} \leftarrow \text{argmin}_{node}(node_x - sample_x)^2 + (node_y - sample_y)^2 + (node_\theta - sample_\theta)^2$$

This metric was chosen after pure euclidean distance, hence omitting θ, proved to be a poor indicator of what constitutes a node being "near" to a sample point due to the robot's differential drive constraints.

For the drive policy, a system of equations was then constructed from the differential drive kinematics as shown below. Through these equations, the left and right angular wheel velocities were computed by minimizing the least squares error of the sampled state and nearest node state.

$$\text{eqn. 1: } \frac{r}{L}u_r - u_l = \Delta\theta \tag{1}$$

$$\text{eqn. 2: } \frac{r}{2}(u_l + u_r)sin\theta = \Delta x \tag{2}$$

$$\text{eqn. 3: } \frac{r}{2}(u_l + u_r)cos\theta = \Delta y \tag{3}$$

$$u_l, u_r \leftarrow \text{argmin}_{u_r, u_l}\ (\Delta\theta)^2 + (\Delta x)^2 + (\Delta y)^2 \tag{4}$$

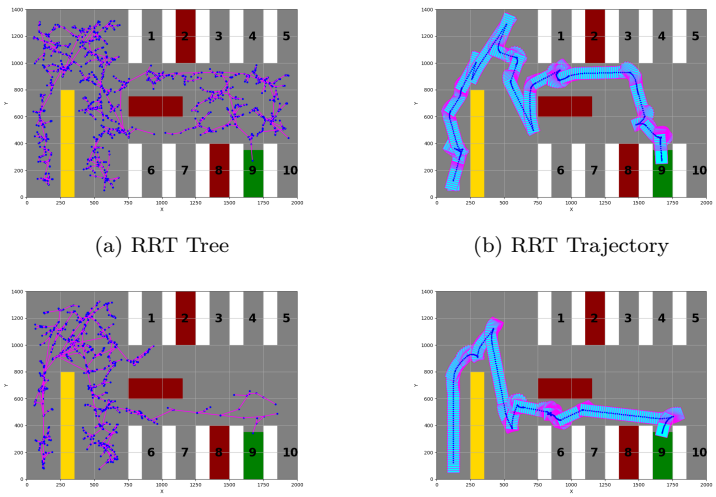
Lastly, the robot configuration space C/C_{obs} was computed using the Minkowski Sum of the robot and obstacle geometries for a discretized set of 3600 θ values, effectively reducing the robot shape to a point representation. The configuration space was then used to check for collisions during drive periods lasting 1 second.

A more detailed explanation of the mathematical setup as well as detailed psuedocode representations of RRT and RRT* can be found in the main report.

4 Results

As *probabilistically complete* algorithms, both RRT and RRT* were able to successfully find a viable path as $t \to \infty$ as long as such a path existed. Below, figure 1 displays two such viable trajectories produced by the algorithms for a relatively complex obstacle layout.

As can be seen, RRT* resulted in a more optimal path avoiding needless random motions compared to RRT. This is further reinforced when observing the tree structures of both algorithms where the rewiring procedures of RRT* result in a more concentrated branching point massed towards the initial state whereas RRT produces a tree with branch points spread out all throughout the map.



(a) RRT Tree (b) RRT Trajectory

(c) RRT* Tree (d) RRT* Trajectory

Figure 1: RRT and RRT* Run Comparison

Despite RRT* leading to an optimal path, it was seen to be much more computationally expensive due to the overhead of searching for optimal parents nodes and rewiring operations. In fact, for a similar number of nodes, RRT* would often take 2-3 times longer than RRT resulting in a trade-off between optimality and speed between the two approaches.

5 Key Contributions

Wrote the mathematical formulation and contributed significantly to the experimental setup section of the main report. Wrote a significant portion of the code and comments. Came up with the drive policy and nearest node metric.