AER1516 A2: Planning in Practice

Ali Mahdavifar
1002251582

Table of Contents

Problem Setup

The setup for the comparison runs were as follows:

- Default starting and end points as in the starter code
- 1000 runs for each planner
- Maximum iteration for each planner at each run was set to 10,000
- random.seed(1)

The norm of distances between two nodes along a path were used to calculate the total path length for each run. Both planners exited the first time they reached the goal. In the Appendix, the main function of dubins_path_planning file is included to show the setup and also the obstacles list.

Planner Results

In one of the runs displayed as an example here, we can see that RRT has more curves and turns, while RRT* has more straight paths and benefits from more pruned branches. This results in a shorter path, which is not always the case due to the randomness associated with these algorithms. There are some iterations when RRT performs better than RRT*, while on average the latter yields more favourable results.

An adaptive distance threshold for choosing the goal node versus a random node was chosen such that as we get closer to the known goal node, the threshold becomes smaller. Overall, this improved the performance of both planners to reach the goal node at fewer iterations.

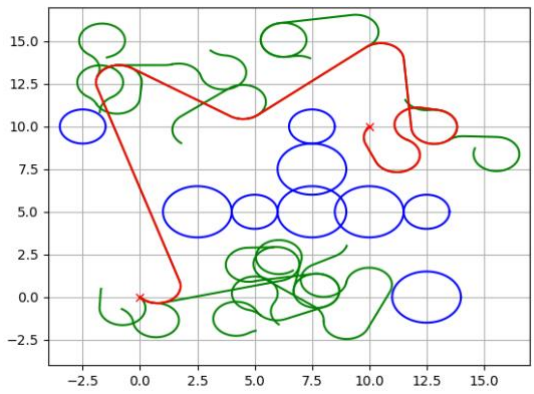

Figure 1 - RRT = 150.97

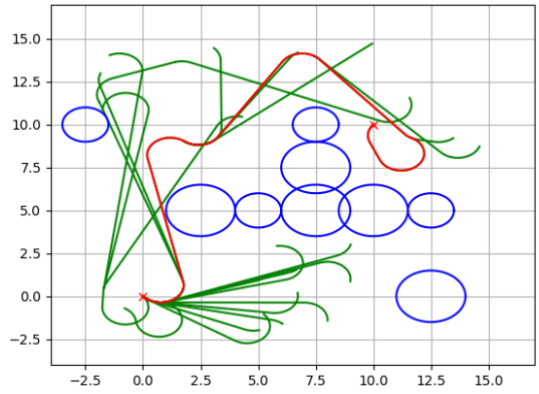

Figure 2 - RRT = 91.96*

Comparison Charts

As mentioned, RRT* for this environment yields a shorter path length on average. See Figure 3. Path distribution for all 1000 runs is shown in Figure 4 below.

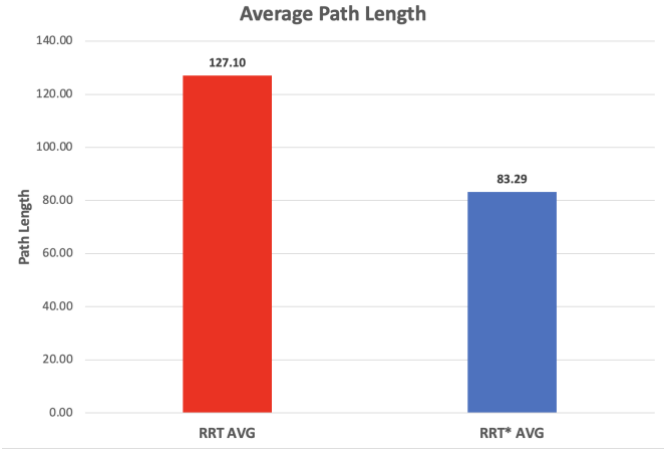

Figure 3

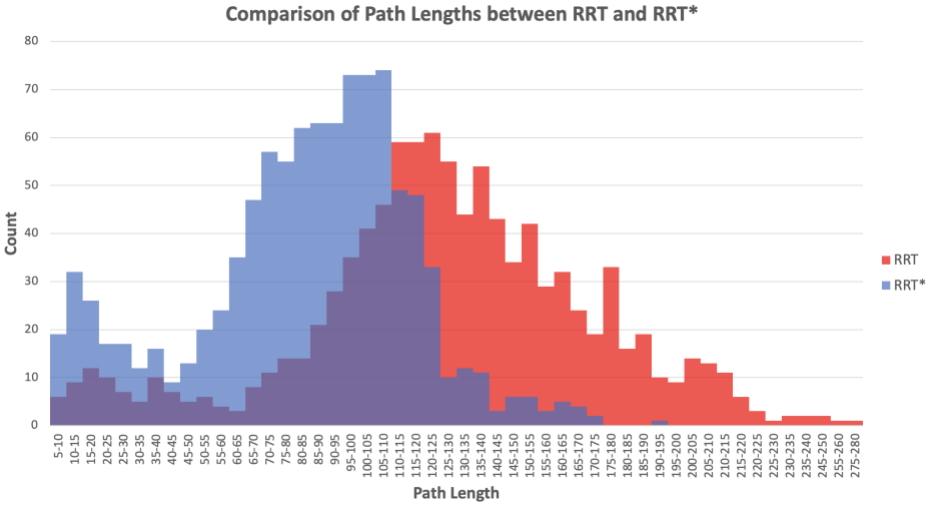

Figure 4 - RRT mean = 127.1 and STDV. = 44.9, RRT mean = 83.3 and STDV. = 33.9*

Appendix: Main Function Code

```
def main():
    print("Executing: " + __file__)

    # ====Search Path with RRT====
    obstacleList = [
        (-2.5, 10, 1),
        (2.5, 5, 1.5),
        (5, 5, 1),
        (7.5, 5, 1.5),
        (12.5, 5, 1),
        (12.5, 0, 1.5),
        (7.5, 7.5, 1.5),
        (7.5, 10, 1),
        (10, 5, 1.5)
    ]  # [x,y,size(radius)]

    # Set Initial parameters
    start = [0.0, 0.0, np.deg2rad(-50.0)]
    goal = [10.0, 10.0, np.deg2rad(50.0)]

    # Run the planner on the obstacle map
    path_len_list = []
    N = 1000
    for i in range(N):
        rrt_dubins = RRT_dubins_problem(start=start, goal=goal,
                                        obstacle_list=obstacleList,
                                        map area=[-2.0, 15.0, -2.0, 15.0],
                                        max_iter=10000)

        # Choose planner type
        # path_node_list = rrt_dubins.rrt_star_planning(display_map=False)
        path_node_list = rrt_dubins.rrt_planning(display_map=False)
        is_path_valid = check_path(rrt_dubins, path_node_list)
        path = get_path(path_node_list)

        if not path:
            print(f'Test Failed: Given path is empty')
            continue
        if not is_path_valid:
            print(f'Test Failed: Given path is not valid')
            continue

        # Calculate path length
        x_vec = []
        y_vec = []

        for j in range(len(path)):
            x_vec.append(path[j][0])
            y_vec.append(path[j][1])

        x_vec = np.array(x_vec)
        y_vec = np.array(y_vec)

        total_path_len = np.linalg.norm(x_vec - y_vec)
        print("     Path length at i = ", i, ": ", total path len)
        path_len_list.append(total_path_len)

        # Draw final path
        if show_final_plot:
            rrt_dubins.draw_graph()
            plt.plot([x for (x, y) in path], [y for (x, y) in path], '-r')
            plt.grid(True)
            plt.pause(0.001)  # Necessary for macs
            plt.show()

    # Save results to a CSV file
    np.savetxt('rrt_star.csv', path_len_list, delimiter=',')
```