LABORATORY II
Planning and Navigation

ROB521 Mobile Robotics and Perception

Mar. 6, 2023

Tianyimeng Fang 1004871025

Qihan Gao 1004846517

Mulan Wu 1004962542

Jonathan Spraggett 1003958737

2.3 Part A Deliverable Summary

1. A picture with a successful output from RRT, given goal in green.

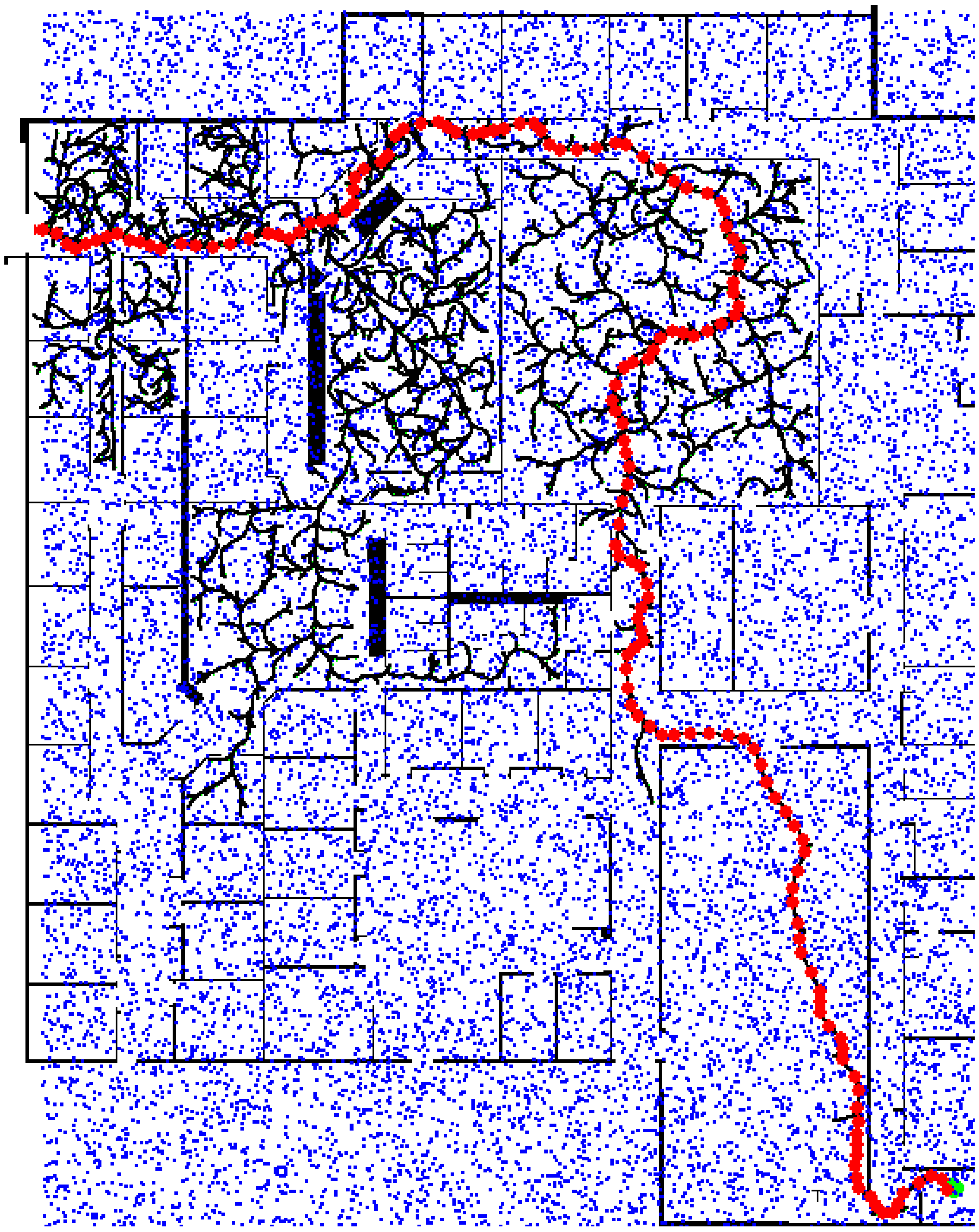

Figure 1. Successful output from RRT

2. A picture with a successful output from RRT*, given goal in green.

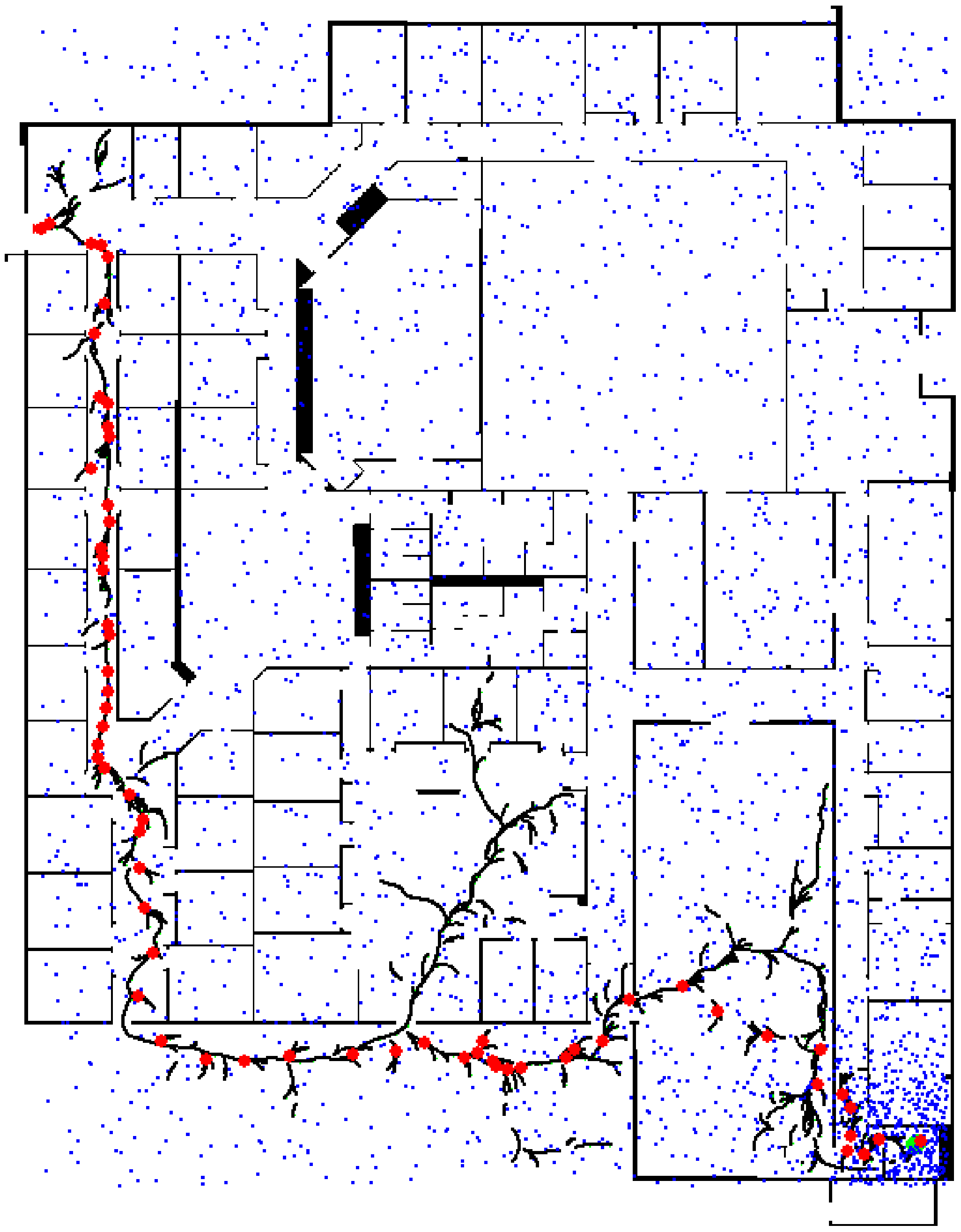

Figure 2. Successful output from RRT*

3. (/3) A video with a successful run of trajectory rollout on your path from RRT or RRT*. If you were not able to successfully solve RRT or RRT*, you must show trajectory rollout working on our provided path (path_complete.npy).

lab2-rrtstar.mp4

4. A brief explanation of algorithms

The initial step is to generate an occupancy map using the given map. This is accomplished by the point_to_cell() function, which converts [x,y] points in the map to corresponding cell indices in the occupancy map. The points_to_robot_circle() function is also used to convert [x,y] points to robot map footprints for later collision detection.

To compute a feasible trajectory, the simulate_trajectory() function takes a point and a node as inputs and utilizes the robot_controller and trajectory_rollout functions. The controller function calculates the velocities required to move the robot from node to node using a feedback controller with a tuning factor alpha of 0.4. The trajectory_rollout() function provides the robot's positions at 1-second intervals with timesteps of 0.1 seconds.

The algorithm known as RRT (Rapidly-Exploring Random Trees) is responsible for determining the path to the desired goal state. This is accomplished by constructing a tree structure composed of randomly selected points within the robot's operating space shown by the rrt_planning(). To begin the process, the algorithm randomly selects a point within the map and searches for the nearest node in its proximity. From there, the simulate_trajectory() function generates the trajectory towards the point. The trajectory is then checked for duplicates amongst the existing nodes and assessed for any potential collisions using the check_if_duplicate() and check_collision() functions. Assuming that both conditions are satisfied, the point is added to the tree and the robot is directed toward the point. The algorithm verifies if the goal has been reached. Lastly, the nodes are retrieved in reverse order to output the final path.

The RRT* algorithm is an extension of the RRT algorithm that enhances its functionality by rewriting the existing nodes based on the cost computation shown by the function rrt_star_planning(). Similar to the RRT algorithm, the RRT* algorithm begins by selecting a sample point and searching for the nearest node. The resulting trajectory is then evaluated for any potential duplicates or collisions. With N set to 5, the trajectory is subsequently compared with trajectories from the other four closest nodes based on their cost-to-come. If the cost is lower than any of the other nodes, then the node is rewired accordingly. The second rewiring step involves evaluating whether any existing trajectories can be optimized after incorporating the new sample point. The algorithm checks the four closest nodes for potential rewiring. If the cost-to-come is lower via the trajectory passing through the sample point, then the node is disconnected from the parent node and reconnected to the new point. Once the tree has been updated, the path is retrieved if the goal state is reached.

With the path obtained by RRT or RRT*, a local planner described in follow_path() is used to publish velocity commands. The trajectory_rollout() function is utilized again to generate paths that are checked for potential collisions. The trajectory with the lowest cost-to-come is selected as the optimal one. This approach enables the robot to complete its tasks and reach the goal state successfully.

3.3 Part B Deliverable Summary

1.A video with a successful run of trajectory rollout on your path from RRT or RRT* on the new Myhal map.

myhal_rrstar.mp4

2. Successful open-loop demonstration of RRT* on the Myhal map in real-world testing

- Already been marked by TA during the lab session.