Lab II: Planning and Navigation

*RRT & RRT**

University of Toronto
ROB521

March 2nd, 2022

Alvin Pane [1004281118]
Isobel Lees [1003912484]
Thanos Lefas [1003927334]
Alaynah Malik [1003099032]

Introduction

The objective of this lab was to explore two global planners, namely RRT and RRT*, and a local controller, trajectory rollout. Given a map of a complex environment, containing multiple rooms and hallways, the above methods are meant to navigate a simulated robot from a start to a goal point. RRT and RRT* are meant to be optimal planners that generate a collision free path of waypoints from start to goal. The trajectory rollout controller is then to navigate the robot through this path by following the waypoints. This lab is divided into 3 components, Planning: RRT, Planning: RRT*, and Navigation: Trajectory Rollout.

Planning: RRT

The RRT algorithm is as follows. First, a point was randomly selected in real space (within specific bounds around the floorplan as we didn't want RRT to expand far away from the goal point). Second, the closest node to that point was found, as well as the straight-line connection (and distance) between the two. As trajectory_rollout only moved the robot forward for a certain time period, the straight line distance needed to be clipped to a maximum value (simply moving the point closer to the node). This was not so the robot actually reached the point in trajectory_rollout (it didn't), but because if the point was too far away robot_controller produced rotational velocities that didn't work over the shorter trajectory_rollout timeframe, and RRT basically only expanded in a straight line. The sampled point was also discarded if the node was closer than a minimum distance, as a time saving measure to avoid computing the trajectory and collisions for an already well explored space. Third, simulate_trajectory was used to move the robot from the node towards the point. Fourth, points_to_robot_circle was used to find all the pixels the robot would pass through while following the path generated by simulate_trajectory. Collision checking was then done for all the pixels. Finally, assuming no collisions, the end point of the path would be added as a node, with the closest node being its parent.

The legend for all figures used in this report is as follows:

- Red 'x' is a node
- Green circle is the goal
- Green line is the path to the goal
- Blue lines are possible trajectories

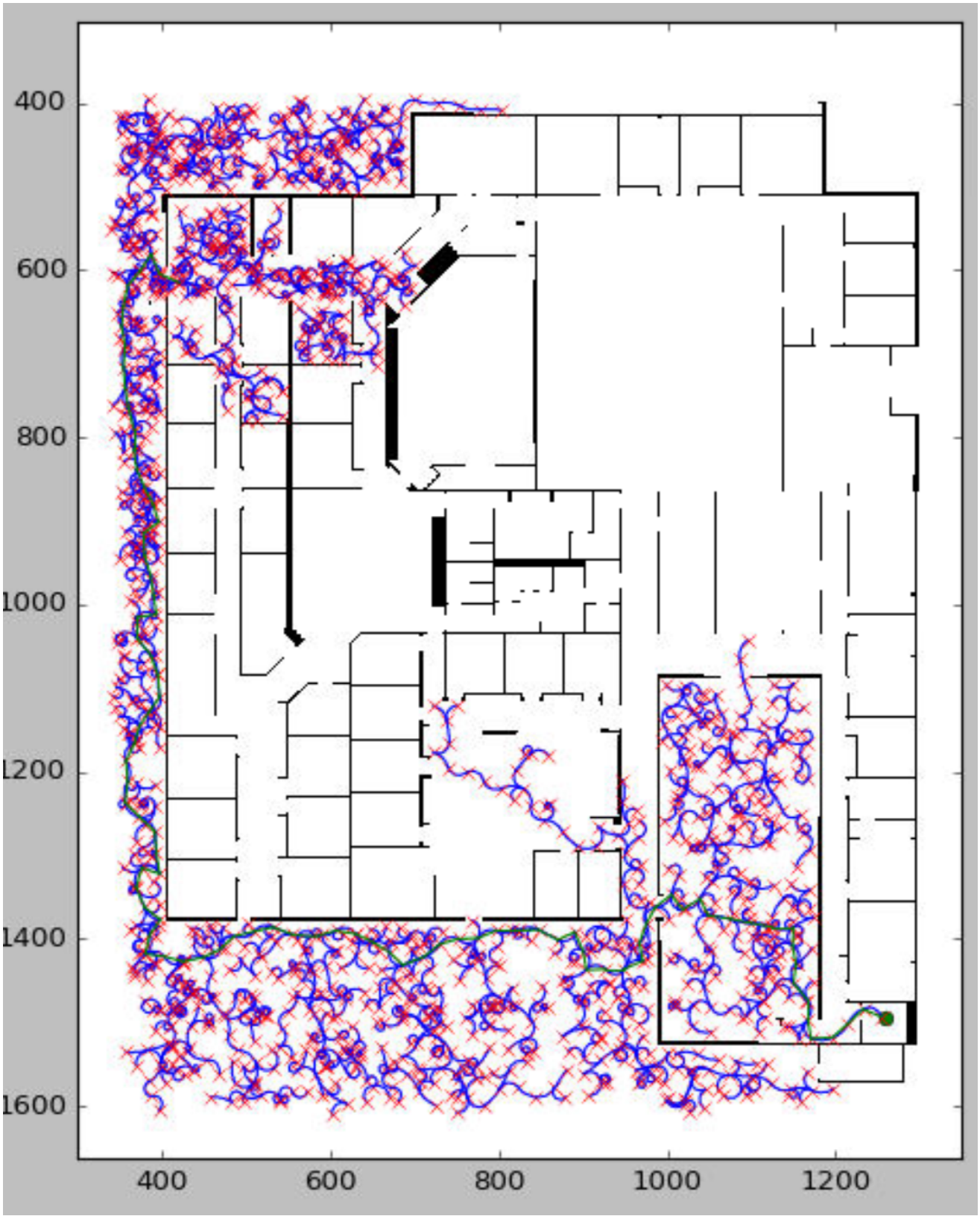

Fig.1 Successful Output of RRT With Given Goal

Planning: RRT*

RRT* is built off the original RRT algorithm, incorporating the ability to rewire connections in the RRT tree to create better paths. New nodes are randomly sampled as before, but for every new node, its surroundings (within a fixed distance) are checked. If there exists a connection with another node within this fixed-distance circle that lowers the cost to reach the newly sampled node, a rewiring is performed, setting that "cheaper" node to be the parent of the newest node. Then, all other nodes within the fixed-distance circle around the new node are checked, to see if they can make use of the new connection. If utilizing the new connection lowers the cost to reach one of the other nodes, the newest node becomes its parent and the cost of all its children is updated.

Three functions are added to make this work; connect_node_to_point, cost_to_come, and update_children. The first function, connect_node_to_point is used to generate a collision-free trajectory between two nodes in the tree, provided one exists. As the problem is over-constrained, we ignore the theta of the last position in the trajectory. The second function, cost_to_come, uses a straight-line distance approach to evaluate the distance (cost) of each path from the start node to the current node. More specifically, this is done by tallying the straight-line distances between all the nodes in the path leading from the start to the current node. Finally, the update_children function is used to propagate the new changes in cost to the child nodes once a parent node has its cost updated via the above method. Whenever a rewiring is performed, the cost associated with that node is changed, and thus the costs of its children are now different as well. This function ensures that the costs of all nodes remain consistent throughout rewiring.

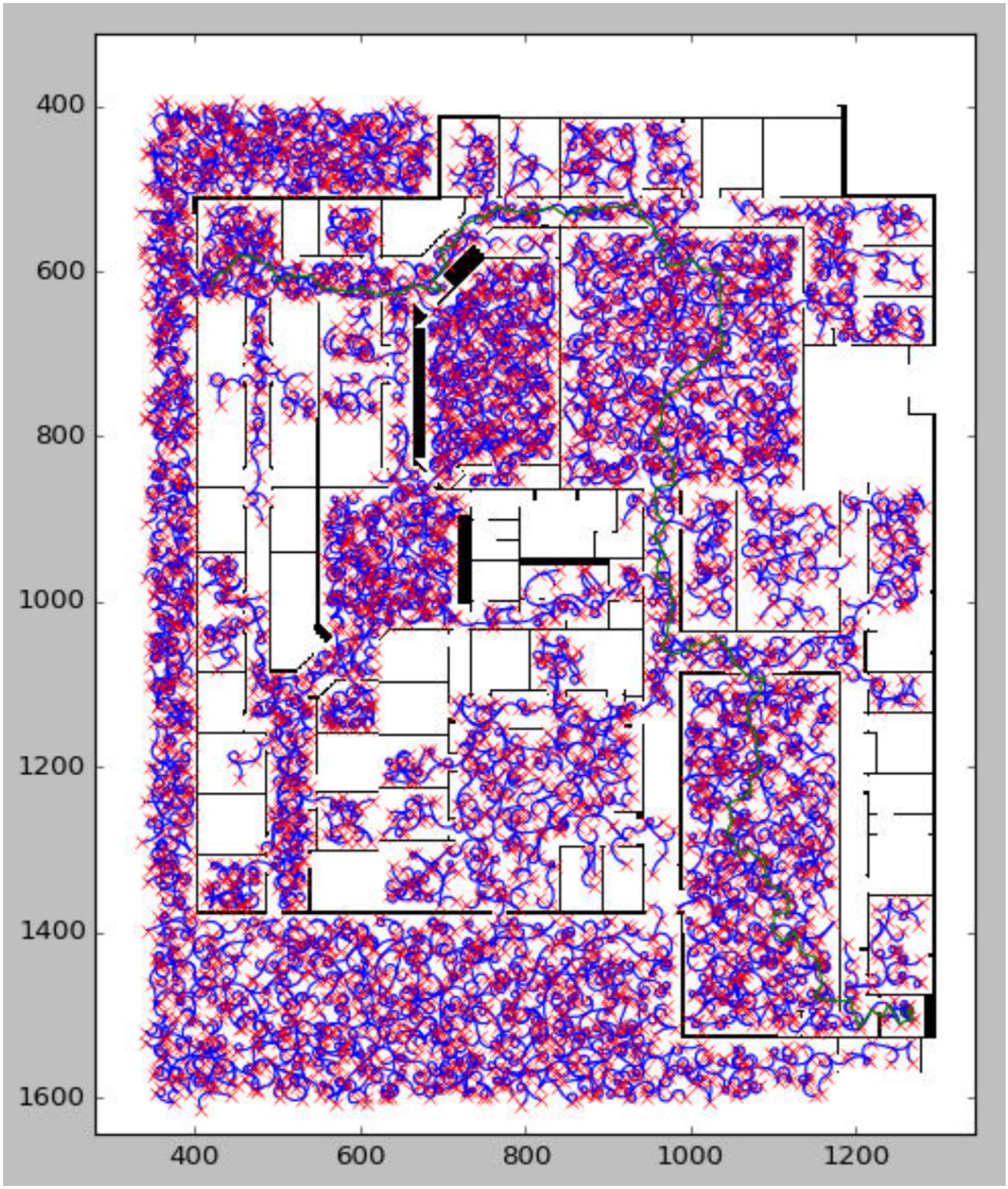

Fig.2 Successful Output of RRT* With Given Goal

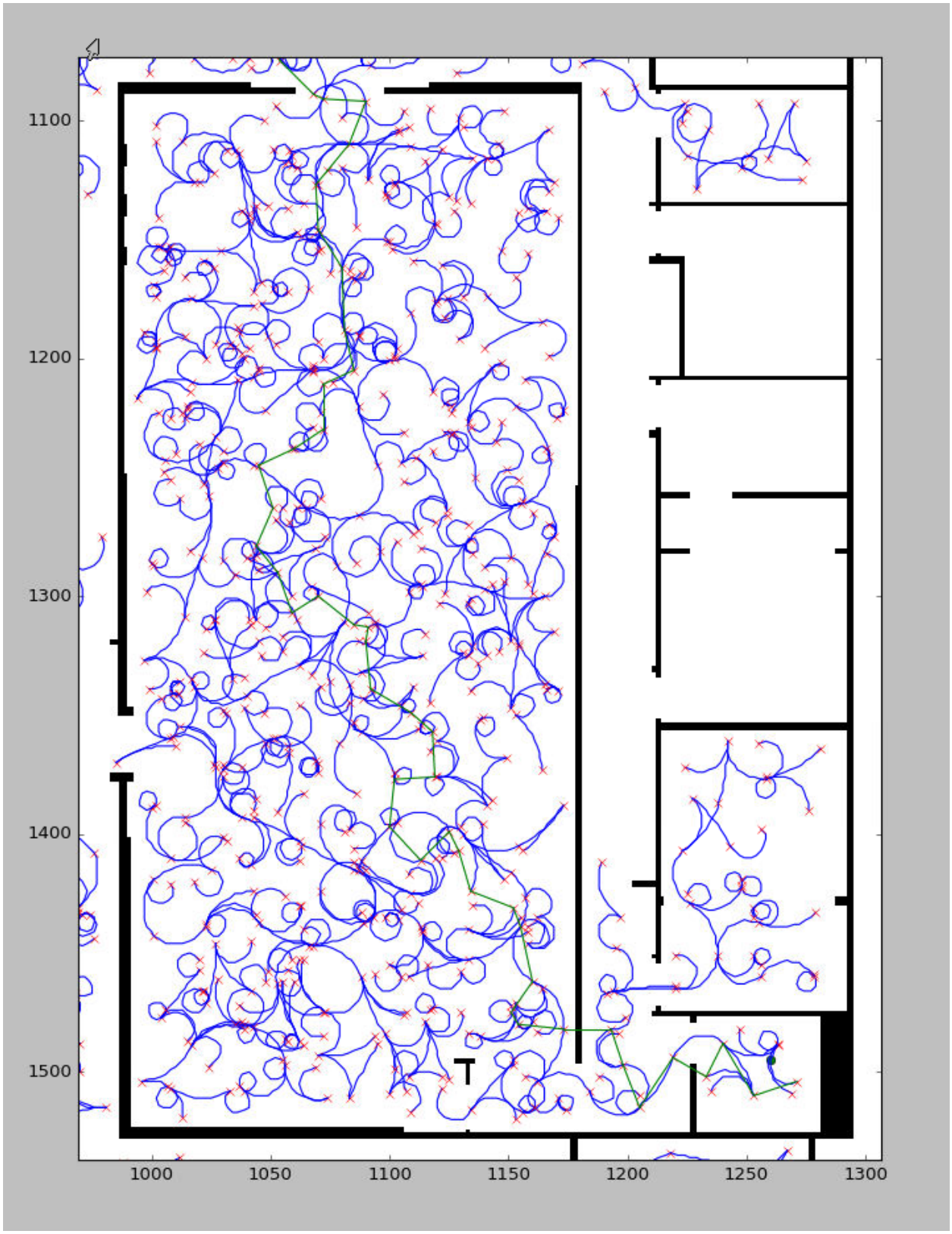

Fig. 3 Detailed View of RRT*'s Potential Trajectories Close to the Goal

Comparison of RRT and RRT*

One can see that the possible trajectories within the RRT* algorithm show a more fan-like structure thanks to the rewiring of the nodes. Also, the path is going through the rooms and hallways of the floor plan as opposed to the RRT algorithm which seemed to favor a path that stayed outside of the floorplan. This is likely as the RRT* algorithm managed to find a shorter path by going through the rooms and hallways instead of circling around the floorplan.

Navigation: Trajectory Rollout

The path-following aspect of this lab was completed using a trajectory rollout approach. Using the waypoints provided by the planner, the path-follower sets each waypoint as its current goal. Then, using a set list of combinations of translational and rotational velocities, trajectories corresponding to these combinations are simulated 1.5s into the future. This creates 43 possible trajectories that the robot can take, all stored in a variable titled local_paths. Each trajectory is then checked for a collision. The collision checking was done by checking every single point on the trajectory (specifically, a circle around every single point representing the space the robot would occupy) for a collision with an obstacle. Any trajectory options with a collision present were marked in a collision table. After all the trajectories had been iterated through, the entries in the collision table were removed from local paths. For all remaining valid options in local paths, a cost function was applied. The cost function we used was the average straight-line distance of all points in the trajectory to the goal. By considering all points in the trajectory, rather than just the end point, this allowed us to also score the curvature of each trajectory. At the end of this, the option with the lowest cost was selected, and its corresponding translational and rotational velocity was published to the robot.