# ROB521 Laboratory 2

**RRT**

Our RRT algorithm is composed of three major components: random point sampling, trajectory rollout towards random point, and collision detection. In random point sampling, points within a predefined bounding box is sampled randomly. We gradually decrease the bounding box size as the tree expands towards the goal, to sample more densely the region around the goal and speedup planning. In trajectory rollout, we find the control that would move the robot from a given pose to the sampled point. If the trajectory is collision free, we take the last point in the trajectory and add it as a new node if it is collision free and distinct from existing nodes.

**RRT\***

Our RRT\* algorithm is an extension to the RRT algorithm described above, with a modified connection step and an additional rewiring step. Instead of connecting the new node to the closest node, we use Bezier curves to check if the new node can be reached from neighboring node poses. Among all the viable neighboring node poses, we check the cost to come for the new node when wired to the neighboring nodes. The connection is established between the new node and neighboring node that yields the lowest cost to come. The cost to come is calculated based on the total trajectory length. In the rewiring step, we attempt to connect new node to neighboring nodes. If the connection results in a lower cost to come for the neighboring nodes, the neighbor node is rewired to the new node, and the cost to come reduction is propagated to all the children of the neighboring node.

**RRT Result RRT\* Result**

A picture containing graphical user interface

Description automatically generated A picture containing graphical user interface

Description automatically generated

**Tracking Controller**

Our local planner roughly used the same template that was provided to us in the handout and skeleton code, where we selected a set of possible controls, rolled out each one, checked for collision, and then picked the best control out of a set of heuristics. To improve runtime, we lowered the time horizon for the rollout and increased the timestep slightly. Rather than checking for collisions on every point on each trajectory, we also stopped checking as soon as we found a collision for a given control. Our heuristic penalized position error exclusively when far from the goal, then penalized heading error once within the vicinity. Lastly, we added an additional control to the list that allowed the robot to slowly back up, which we found to be helpful when making small adjustments close to certain waypoints.