Lab2 Report

PRA0102 Group1

Task1: Collision detection

Point_to_cell:

This function determines the map cell the input “point” is currently occupying. The occupied cell’s location is calculated by the “point”’s location minus the ‘origin’’s position in pixels. Because the ‘origin’ is at the bottom left corner while the origin of the map is at top left corner, the occupied cell’s y position will need to be minused by the height of the map.

Points_to_robot_circle:

This function determines the cell that the robot will occupy in a series of motions. The function first converts the waypoints of the robot’s motions to cell location. Then, we use the “circle” function to calculate the cell that the circle(Robot) will occupy. That will be the total number of cells that the robot is occupying and their locations.

Task2: Simulate Trajectory

Trajectory_rollout:

This function computes the trajectory of the robot based on its linear and rotational velocity. The unicycle model is used to determine the robot’s trajectory between 2 points. The unicycle model is presented below.

$$\underbrace{\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}}_{\dot{\mathbf{q}}} = \underbrace{\begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}}_{\mathbf{G}(\mathbf{q})} \underbrace{\begin{bmatrix}v\\ \omega\end{bmatrix}}_{\mathbf{p}}$$

translational speed

rotational speed

generalized velocity

Robot_controller:

The robot_controller outputs the linear and rotational velocity of the robot from ‘node_i’ to ‘point_s’. It utilizes the unicycle model in ‘trajectory rollout’ to calculate the trajectory for every option that the robot has to move from node_i to point_s and check for the collision based on the cells that the robot occupies.

Simulate_trajectory:

It simulates the non-holonomic motion of the robot moving from node_i to point_s and produces the final trajectory of the robot between 2 points. It uses robot_controller to calculate the rotational and linear velocity and uses trajectory_rollout to produce the preprocessed trajectory. The robot_traj is then converted to the global_traj based on the accumulated distance and change in yaw angle(base_theta+robot_traj[2,:]).

Task3: RRT

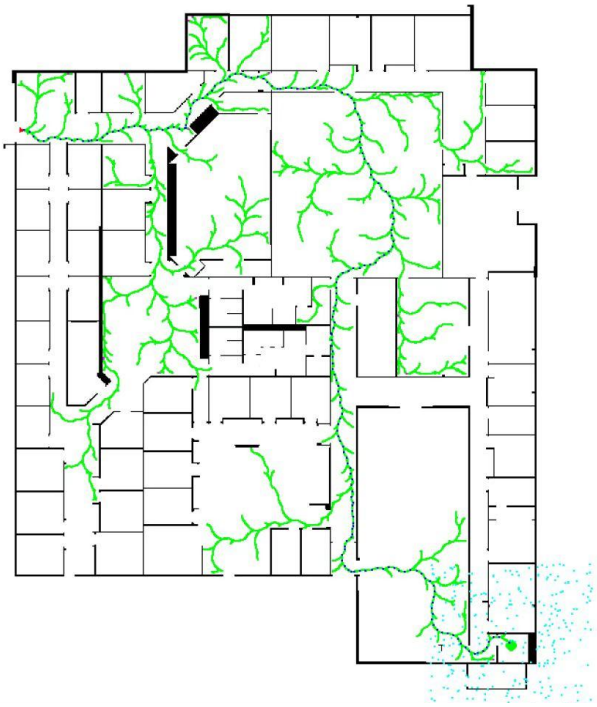

Fig.1 RRT result.

rrt_planning:

This algorithm expands random trajectories that the robot can move until the one trajectory that can reach the goal point. While the goal point is not reached, the trajectory tree will keep expanding until it reaches the maximum iterations. The sample points are uniformly distributed in the map in every iteration. The algorithm will try to find a trajectory to the closest point to the node added from the previous step. And if there is a valid trajectory(No collision), then the closest point will be added to be a node. Otherwise, the node will be marked as a dead node. The node will not be considered for other trajectories' expansion. We add some small modifications to the rrt to reduce the possibility for the correct trajectory to get stuck. For every 200 iterations, our algorithm will make the goal as the sampled point and see whether there is a trajectory directly to the goal. For every 205 and 210 iterations, the algorithm will sample the entry and the hallway of the room where the goal is located. This can check whether the robot can reach the entry for the room where the goal is located and the hallway near the goal respectively. When the robot is close to the goal, a higher density sample points in the area around the goal point will be produced to get through some tight maneuvers.

Task4: Rewiring the Existing Nodes

Connect_node_to_point:

This function uses the unicycle model to produce a trajectory in global coordinate that can connect one node to another.

Task5: Trajectory Costs

Cost_to_come:

This function calculates the cost of a trajectory. The cost is calculated by the accumulated Euclidean distance between the waypoints on the trajectory, namely, the length of the trajectory.

Task6: Updating Children

Update_children:

A recursive algorithm that will update the cost of all the connected nodes when a new node is added from its parent node and have a change in cost.

Task7: RRT Star

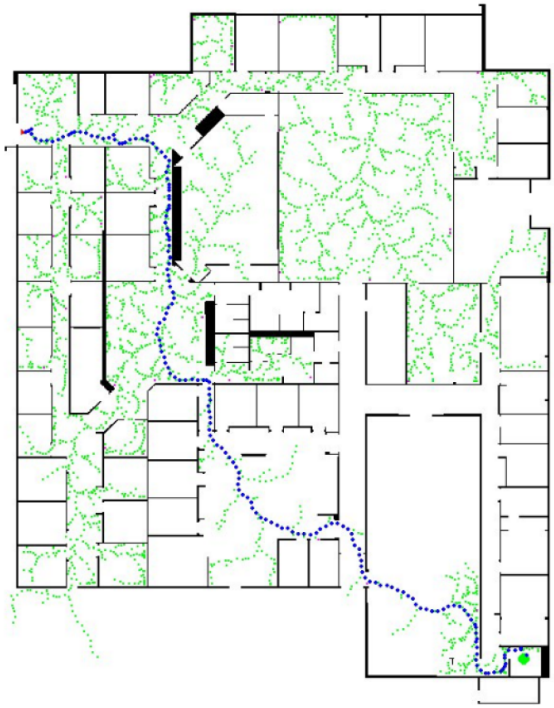

Fig.2 RRT star result

This algorithm will ideally generate the shortest path to the goal. The rapid-exploring tree will grow with the same logic as the normal RRT. However, the algorithm will first check the closest existing node to the new simulated node and connect them. Then it finds a nearby node that has less overall cost if it is rewired to the new node(In this case, the length of the trajectory), the that nearby node will be rewired to the new node and all the cost of all the nearby node's children will be updated.

Task8: Using Trajectory Rollout for Local Planning

trajectory_rollout:

The trajectory_rollout produces the trajectory waypoints based on the series of robot's linear and rotational velocity. The waypoints are calculated using the unicycle model and they are in the global coordinate.

Cost_to_go:

This function outputs the cost on the difference between waypoints' pose and closest pose in the trajectory. When the robot is far away from the current goal , the priority is to get closer to the current goal. When the robot is close to the current goal, the priority is to align with the waypoint heading. The cost is composited using 2 parts. The first part is the distance from the current position to the goal point using the euclidean distance. The second part is the difference between the orientation that the robot needs to be at the current goal point.

Follow_path:

The function will first use trajectory_rollout function to propagate the trajectory forward, and store the resulting waypoints in local_paths. Then, collision check will remove the paths that will collide with obstacles. The heuristic cost of the local paths are calculated and the path with the shortest path is selected to execute. When the robot collides with an obstacle, it will reverse in x direction to keep a distance from the obsiticle.