ROB 422 HW 3

Yulun Zhuang
yulunz@umich.edu

October 25, 2023

Questions

1. AI book, exercise 3.3

a. State space: all possible city pairs (i, j).
Successor function: the successors of (i, j) are all pairs (x, y) where city x is connected to city i and city y is connected to city j.
Actions: move to a neighboring successor.
Action cost: $\max(dist(i, x), dist(j, y))$.
Goal test: check if reach (i, i) for some i.

b. $D(i, j)/2$ is admissible, since for the best case, there is another city right in the middle between city i and j which has a cost $D(i, j)/2$ for state (i, j).

c. Yes, a map with two nodes connected by one link has no solution.

d. Yes, a map with two nodes connected by one link and one of them has a self loop has all solutions need one friend to visit that city twice.

2. AI book, exercise 4.1

a. Hill-climbing search

b. Breadth-first search

c. Hill-climbing but always choose the first feasible neighbor

d. Random-walk search

e. Random-walk search in the space of individual config dimensions

3. LaValle book, Chapter 4 exercise 5

The dimension of the C-space of the cylindrical rod is 6 if the rod can translate and rotate freely in $\mathbb{R}^3$. However, if the rotation about its central axis can not be detectable, the dimension reduces to **5**. The C-space can be represented as $\mathbb{R}^3 \times \mathbb{S}^2$. The translation part is $\mathbb{R}^3$, since it can translate freely in $\mathbb{R}^3$. And the rotation part is $\mathbb{S}^2$ because the rotation about one degree of freedom is undistinguished, and the resulting topology is $SO(3)/SO(2) = \mathbb{S}^2$.

4. LaValle book, Chapter 4 exercise 16

The topology of the resulting C-space is $SE(3)^5$ i.e. $(\mathbb{R}^3 \times SO(3))^5$ and its dimension is 30.

5. LaValle book, Chapter 5 exercise 18

Searching an implicit, high-resolution grid refers to the grid-based discretized planning. It requires to discretize the C-space into grid cells and each of them represents a possible configuration, and algorithms like A* can be used to navigate cells. High-resolution discretization can provide a more accurate representation of the space, but increase the search run-time and memory requirements a lot.

Growing search trees directly on the C-space without a grid refers to tree-based continuous planning. The tree starts with an initial configuration, and new configurations are added incrementally by connecting to existing ones. Algorithms like RRT can be used to grow the search tree. This method operates directly in the continuous C-space, allowing for more flexibility in representing the space.

Implementation

A*

a. 4-Connected variant

Computation time: 11.2534 s

Path cost: 12.4708

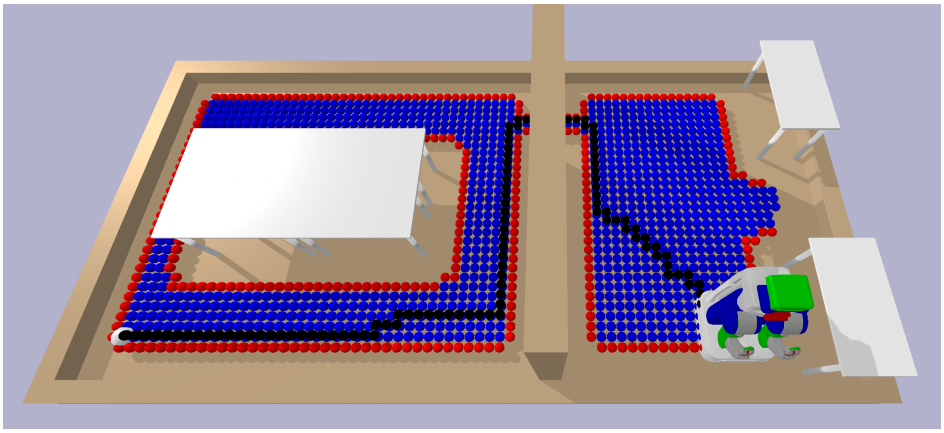

Figure 1: Visualization for explored configurations of 4-connected A* algorithm. The searched path is shown in black, the collision-free configurations are shown in blue and the colliding configurations are shown in red.

b. 8-Connected variant

Computation time: 7.5935 s

Path cost: 10.6955

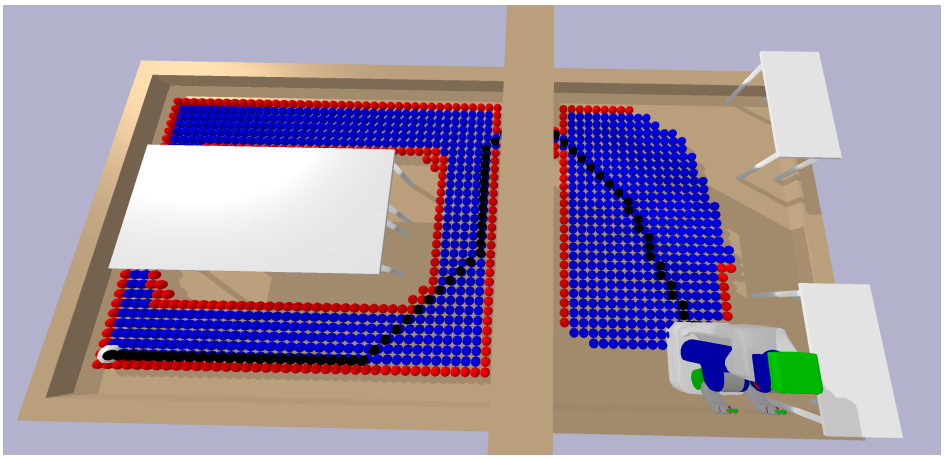

Figure 2: Visualization for explored configurations of 8-connected A* algorithm. The searched path is shown in black, the collision-free configurations are shown in blue and the colliding configurations are shown in red.

c. Discussion

A* Variant	4-Connected	8-Connected
Computation Time (s)	11.2534	7.5935
Path Cost	12.4708	10.6955
# Expend Directions	6	26
# Iterations	9026	7088
Open List Length	3596	7321
Closed List Length	6628	4431

Table 1: Comparison of runtime statistic between two variants of A* algorithm. Run times are recorded on a laptop with i7-12700H CPU.

The 8-connected variant performs better in terms of both computation time and path cost as shown in Table 1.

It's obvious that the 8-connected variant need explore more nodes (11752) than the other (10224) since for each node it will explore more neighbor nodes around it, and it can utilize the diagonal direction of each config to find a path with lower accumulated action costs (10.6955) i.e. g cost than the other (12.4708) as well.

However, due to the fact that collision checking is far more computational expensive than other run time costs, the implementation can be optimized so that the variant with fewer iterations will run faster than the other.

It runs faster because fewer iterations are required and the most time expensive operation is processed only once in each iterations. There are two main sources for the run time expense, the first and dominant one is the collision checking process and the second one is the lookup of the closed list. They are necessary conditions to check for each explored nodes, but there are also two slots to choose for checking these conditions. One is right after getting the lowest cost node from the Open List and the other is when a neighbor node is explored. As long as both of them are checked once in total for both slots, the optimality of the searched path is reserved. Thus, to trade off the computation time over the number of explored nodes, check both collision and closed list at the first slot and check only closed list at the second slot will result in the fewer iterations while keep the number of explored nodes at a reasonable amounts (the more checks, the fewer nodes will be explored), so that makes the 8-connected variant faster.

Rapidly-Exploring Random Tree (RRT)

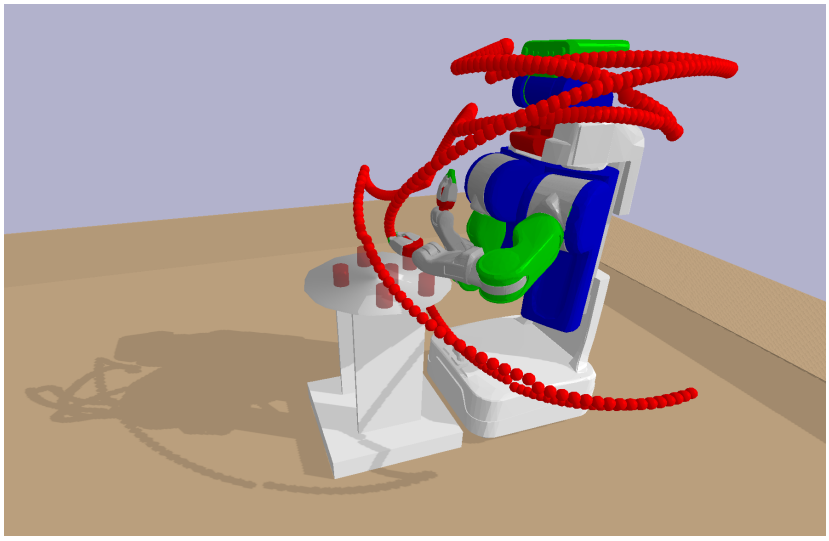

Figure 3: Visualization for searched path of the left end-effector in red.

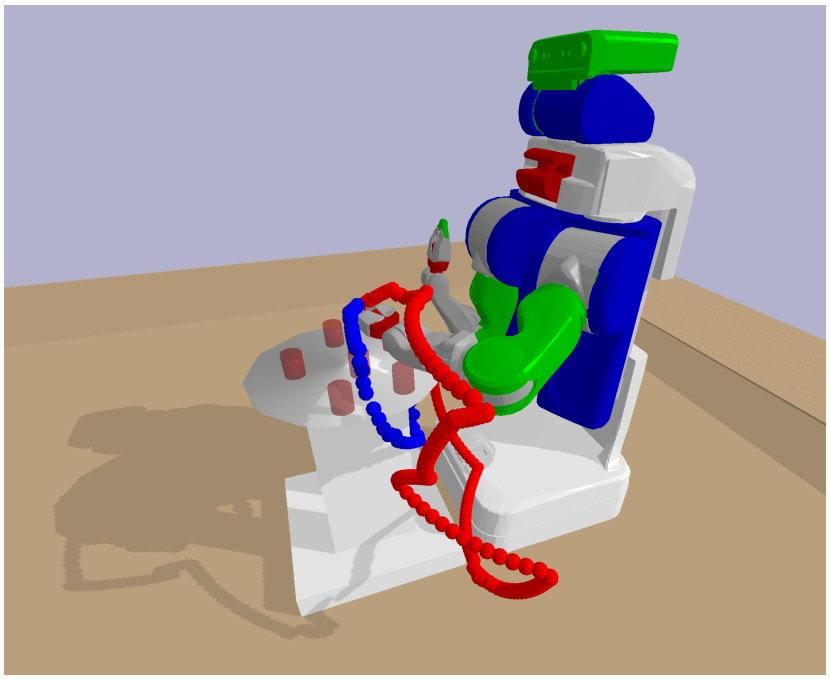

Figure 4: Visualization for searched path of the left end-effector in red and the shortcut-smoothed path in blue.

A Fun Facts

This appendix aims to discuss some fun facts and interesting characteristics of both A* and RRT algorithm I explored during implementation.

A*

a. Search with heuristic cost only, but get a much faster suboptimal result?

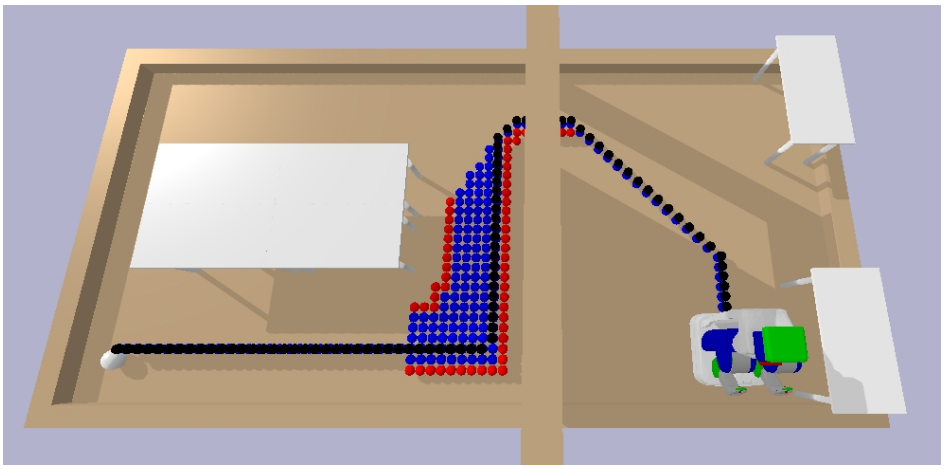

Figure 5: A* with heuristic cost only.

This is only true if the $h(n)$ is very close the true cost and not an lower bound estimator (admissible), otherwise this is a disaster since the algorithm will just ignore the true cost of traveling from one node to the other (g cost) and result in a "imagined" optimal path. Indeed, the problem setup for this homework falls into the true case.

b. Enqueue any feasible neighbors and ignore nodes with lower g cost in open list, but get same result?

This is true for problems I have solved so far. The mechanism behind is just let the priority queue to sift up the node with lower cost and let the closed list to handle those duplicated configs. The drawback of this is that much more unnecessary nodes are enqueued so typically requires more iterations and slows down the run time.

RRT

a. Fun thing to try. Apply RRT to the first scenario?

It works perfectly fine (Figure 6), as expected, even though all config dimensions are treated as linear (no special distance wrap e.g. superior arc).

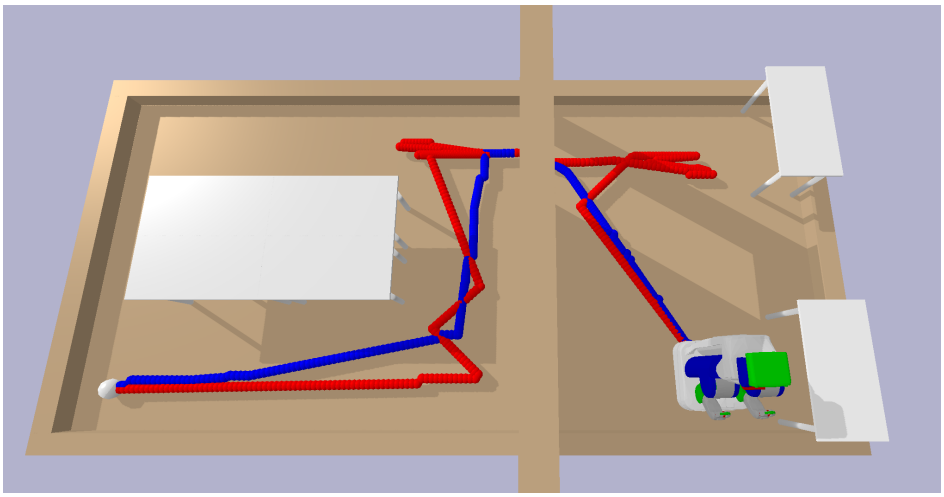

Figure 6: Use RRT for path finding.