EECS 465/ROB 422: Introduction to Algorithmic Robotics
Fall 2024
Homework Assignment #3

Jinjia Guo

---

1. Suppose two friends live in different cities on a map, such as the Romania map shown in Figure 3.2. On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city i to neighbor j is equal to the road distance $d(i, j)$ between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the first on his/her cell phone) before the next turn can begin. We want the two friends to meet as quickly as possible.
   (a) Write a detailed formulation for this search problem. (You will find it helpful to define some formal notation here.)
   (b) Let $D(i, j)$ be the straight-line distance between cities i and j. Which of the following heuristic functions are admissible? (i) $D(i, j)$; (ii) $2D(i, j)$; (iii) $D(i, j)/2$.
   (c) Are there completely connected maps for which no solution exists?
   (d) Are there maps in which all solutions require one friend to visit the same city twice?

(15 Points)

Ans:

(a) Ans. for (a)
The search problem is that there are two friends who are in different cities. They move to the neighboring cities of their current cities and eventually meet each other. The task goal is to simplify the time T spent on meeting each other. We define it as follows:
Problem definition Goal: In the shortest possible time T, two people (i.e. a and b) go to the same city i.
Definition: Define $d(i, j)$ as the straight-line distance between cities i and j.
State: The two people are located in cities (i, j), where i is the current city of a and j is the current city of b. Note that i and j can be equal, which means the same city.

Action: Everyone can choose to move to a neighboring city. When a moves from city i to city i', and b moves from city j to city j', the increase of T is:

$$T \leftarrow T + \max(\mathrm{d}(i, i'), \mathrm{d}(j, j'))$$

Where $d(i, i')$ represents the distance between cities i and i', and $d(j, j')$ represents the distance between cities j and j'. The moving time is determined by the larger distance between the two.

Solution: The solution is to construct two sequences, one for a and the other for b, in the following form:

$$i \to j \to \cdots \to k \to l \to x$$

$$a \to b \to \cdots \to c \to d \to x$$

The two sequences must eventually meet in the same city x. In addition, the two people cannot be in the same city at the same index except the last one during the entire sequence.

Time optimization:

Minimize the following function to reduce the total meeting time T. Assume that the travel distances of a and b are d_a and d_b respectively, that is:

$$d_a = [\mathrm{d}(i_1, i_2), \mathrm{d}(i_2, i_3), \dots]$$

$$d_b = [\mathrm{d}(j_1, j_2), \mathrm{d}(j_2, j_3), \dots]$$

We get the time increment of each position by calculating the maximum distance in each step:

$$\mathrm{d} = \max(\mathrm{d}(i_k, i_{k+1}), \mathrm{d}(j_k, j_{k+1}))$$

Therefore, the total time T can be expressed as:

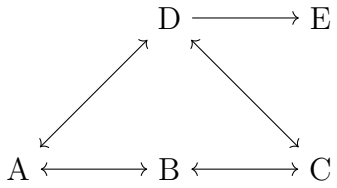
$$T = \sum_{k=1}^{n} \max(\mathrm{d}(i_k, i_{k+1}), \mathrm{d}(j_k, j_{k+1}))$$

The *max* is used here to ensure that the movement time of each step takes into account the slower of the two people, ensuring that the two people arrive at the next location synchronously, avoiding that one person arrives first and then has to wait for the other. Therefore, the total time of the whole process depends on the maximum movement time of each step.

(b) Ans. for (b)
The heuristic $d(i, j)$ is acceptable. This is a very natural thought since this is the minimum possible distance.
The heuristic $2/dotd(i, j)$ is unacceptable. the value returned by this heuristic may be larger than the actual cost of reaching the goal, which is unacceptable.
The heuristic $d(i, j)/2$ assumes that two people can walk directly towards each other at the same speed. This results in each person covering half the straight-line distance between their current positions. In this case, the two people move towards each other at the same speed at the same time, and they meet in the middle. This heuristic does not overestimate the actual cost. So it is acceptable.

(c) Ans. for (c)
In a fully connected map, every city is connected to every other city by a road. Since everyone can move to any neighboring city at any time, they always have a way to meet. Therefore, there is no unsolvable situation.

(d) Ans. for (d)
Suppose there is a simple map consisting of 5 cities, where cities A, B, C, and D form a loop, and city E is a dead end extending outward from city D. The map structure is as follows:



Two people a and b are located in different cities, and their goal is to meet as soon as possible. The assumption is as follows:

- Friend a starts in city A, and friend b starts in city E. - Friend a can move along the loop $A \to B \to C \to D \to E$. - Since E is a dead end, the only path available to friend b is to return to D. A feasible solution: 1. Friend a starts from A and moves along $B \to C \to D \to E$. 2. Friend b starts from E and returns to D, but may have to enter E again or wait in D because friend a has not arrived yet. In this case, friend b must go back and forth between D and E, thus visiting the same city multiple times, until the two can move synchronously and eventually meet. In summary: Due to the dead end E in the map, friend b's path is restricted, he can only move from E to D, and may need to go back and forth between D and E to wait for friend

a to arrive. Therefore, in this dead end map, the solution requires a friend to visit the same city repeatedly.

2. Give the name of the algorithm that results from each of the following special cases:?
   (a) Local beam search with $k = 1$.
   (b) Local beam search with one initial state and no limit on the number of states retained.
   (c) Simulated annealing with $T = 0$ at all times (and omitting the termination test).
   (d) Simulated annealing with $T = \infty$ at all times.
   (e) Genetic algorithm with population size $N = 1$.

(10 points)

Ans:

(a) Hill climbing algorithm

(b) Breadth-first search

(c) Greedy algorithm

(d) Random walk algorithm

(e) Hill climbing algorithm

3. What is the dimension of the C-space for a cylindrical rod that can translate and rotate in $\mathbb{R}^3$? If the rod is rotated about its central axis, it is assumed that the rod's position and orientation are not changed in any detectable way. Express the C-space of the rod in terms of a Cartesian product of simpler spaces (such as $\mathbb{S}^1$, $\mathbb{S}^2$, $\mathbb{R}^n$, P^2, etc.). What is your reasoning? (10 points)

Ans:

(a) **Translational Degrees of Freedom** Obviously, $\mathbb{R}^3$
Rotational Degrees of Freedom The rod rotates about its own central axis, which provides 1 degree of freedom.$\mathbb{S}^1$ The remaining two rotations actually form a sphere in space.$\mathbb{S}^2$

Total C-space and Dimension

$$C = \mathbb{R}^3 \times \mathbb{S}^1 \times \mathbb{S}^2$$

(b) **Translational Degrees of Freedom** Obviously, $\mathbb{R}^3$
Rotational Degrees of Freedom Since the rod can only roll about its central axis, there is just 1 rotational degree of freedom. The rod cannot pitch (change its vertical angle) or yaw (rotate horizontally), so we do not consider those degrees of freedom.So $\mathbb{S}^1$
Total C-space and Dimension

$$C = \mathbb{R}^3 \times \mathbb{S}^1$$

4. Suppose five polyhedral bodies float freely in a 3D world. They are each capable of rotating and translating. If these are treated as 'one" composite robot, what is the topology of the resulting C-space (assume that the bodies are not attached to each other)? What is its dimension? (10 points)

Ans:
For one single polyhedral body:

$$C = \mathbb{R}^3 \times \mathrm{SO}^3$$

Because we have five polyhedral bodies:

$$C = \left(\mathbb{R}^3 \times \mathrm{SO}(3)\right)^5 = \mathbb{R}^{15} \times \mathrm{SO}^{15}$$

The total dimension of the C-space is:

$$dim(C) = 15 + 15 = 30$$

5. Explain the difference between searching an implicit, high-resolution grid and growing search trees directly on the C-space without a grid. (10 points)

Ans:
Searching an implicit, high-resolution grid: By discretizing the C space into a high-resolution grid, the search is performed using a grid-based neighborhood relationship. It is suitable for low-dimensional spaces, but in high dimensions,

due to the exponential growth of the grid, the computation and memory consumption are very large. However, the shortest path can be found. Like A^* and $Dijkstra$

Growing search trees directly on the C-space without a grid: It directly samples in the continuous C space, without discretizing the C space, and generates the search tree by randomly sampling points, which is suitable for high-dimensional continuous space. Like RRT.

6. Implementation of A^* algorithm

   (a) To grade your work, we will run the command python3 astar template.py in a folder where we have extracted your source code. We will not run any other command or any modifications of this command. When this command is run, your code should plan using variant (b) and execute a trajectory for the robot and we should see the robot following this trajectory in the viewer.
   Your code should output the solution in under 10 minutes.

   For each variant record the the computation time to find a path and the path cost (using the action cost function defined above) in your pdf.

   (b) Which variant performs better in terms of computation time? Which variant performs better in terms of path cost? Explain why in the pdf.

Ans:

(a) see *astartemplate.py* in the zip file for the code.
Here we use $dx = 0.1$, $dy = 0.1$, $d\theta = \frac{\pi}{2}$
For "4-connected" neighbors:

```
/home/rob422/anaconda3/envs/rob422/bin/python /home/rob422/PycharmProjects/ROB442-EECS465/Rob422_HW/HW3/astar_template.py
pybullet build time: Sep  3 2024 12:51:03
MESA: error: ZINK: failed to choose pdev
glx: failed to create drisw screen
ven = Mesa
ven = Mesa
Path cost:  12.470796326794868
Planner run time:  6.654031753540039
Executing trajectory
Finished
```

Figure 1: The cost and run time of "4-connected" neighbors A^*

And we have the multiple configurations image:

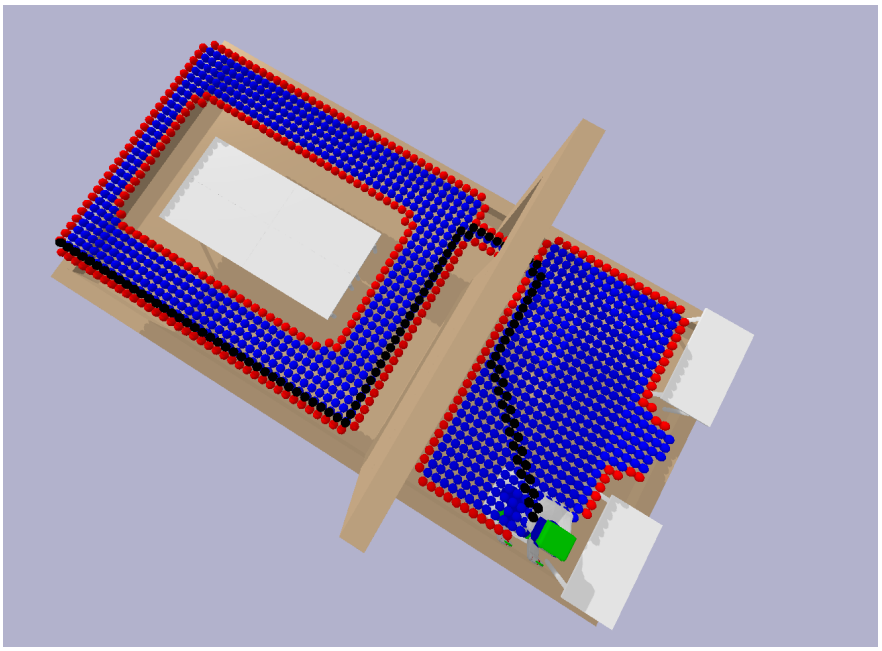


Figure 2: Multiple configurations of "4-connected" neighbors

For "8-connected" neighbors:

```
/home/rob422/anaconda3/envs/rob422/bin/python /home/rob422/PycharmProjects/ROB442-EECS465/Rob422_HW/HW3/astar_template.py
pybullet build time: Sep  3 2024 12:51:03
MESA: error: ZINK: failed to choose pdev
glx: failed to create drisw screen
ven = Mesa
ven = Mesa
Path cost:  10.69552629433298
Planner run time:  4.623113393783569
Executing trajectory
Finished
```

Figure 3: The cost and run time of "8-connected" neighbors A^*

And we have the multiple configurations image:

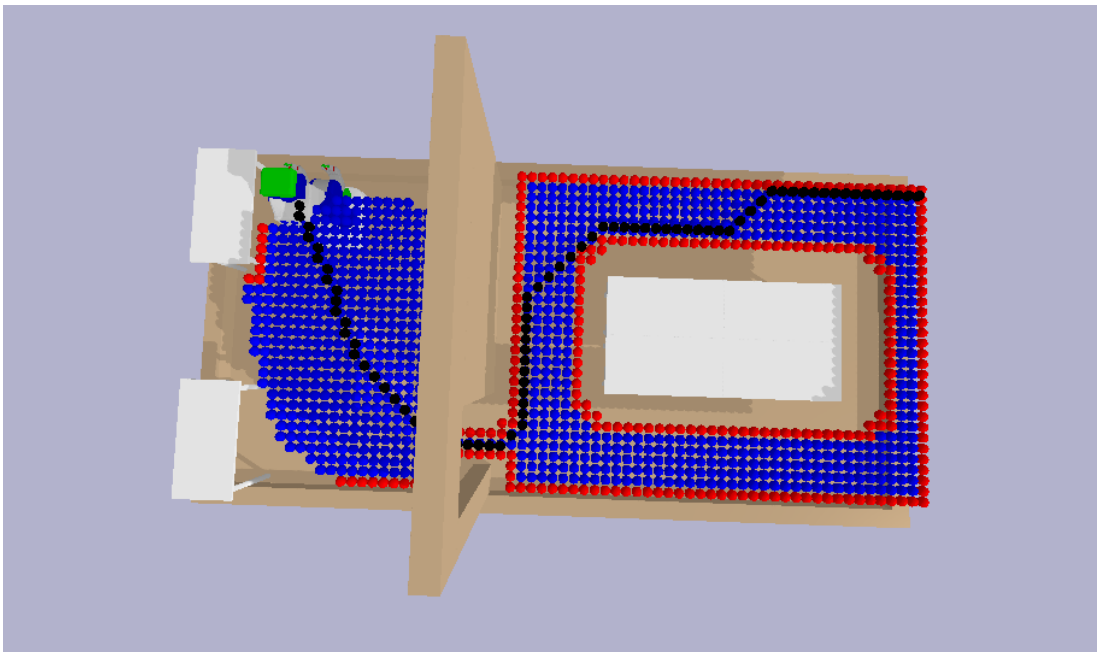


Figure 4: Multiple configurations of "8-connected" neighbors

(b) The running time of A* for 8-connected neighbors is 4.62 seconds, while the running time of A* for 4-connected neighbors is 6.65 seconds. The path cost of 8-connected neighbors is also smaller, it is 10.69s while the path cost of 4-connected neighbors is 12.47s.
Generally speaking, 4-connected is often faster in terms of computational time because fewer neighbors are explored. However, since 4-connected cannot move diagonally, it often goes to unnecessary nodes, resulting in increased costs.
In my code, although 8-connected neighbors increase the number of candidate neighbors for each node because the path is straighter and shorter, A* finds the endpoint in a shorter time, so the overall time consumption is reduced. 4-connected neighbors can only explore four directions, the path becomes longer, and more nodes need to be explored, resulting in a longer running time. In these two scenarios, 8-connected neighbors perform better, reducing both time overhead and path cost.

7. Implementation of RRT-Connect algorithm

   (a) Draw the position of the left end-effector of the robot for every configuration along the path in red in the viewer (see demo.py for how to get this position). You should see that the points along the path are no more than a few centime-

ters apart. Include a screenshot showing the path you computed in your pdf. Implement the shortcut smoothing algorithm to shorten the path. Use 150 iterations. Draw the original path of the end-effector computed by the RRT in red and the shortcut-smoothed path of the end-effector in blue in the viewer. Include a screenshot showing the two paths in your pdf.

(b) Execute your path and verify that the arm indeed reaches the goal (doesn't hit obstacles).

Ans:

(a) This is the screenshot showing the path computed.

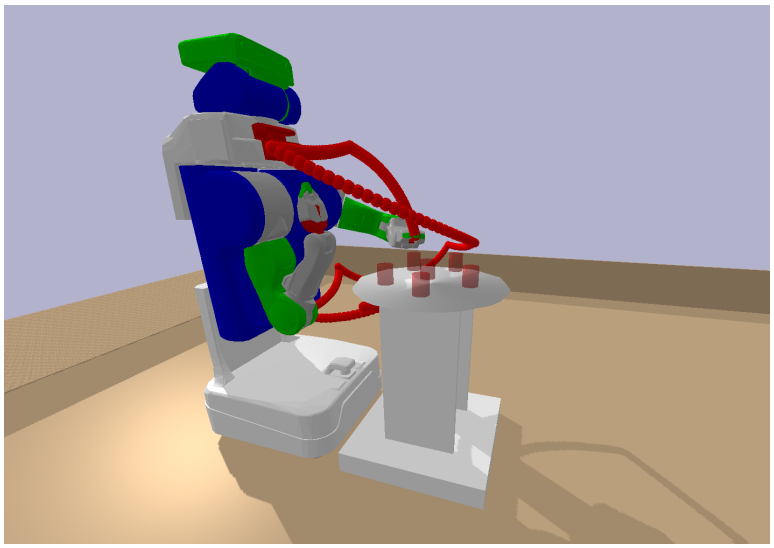


Figure 5: The screenshot showing the path computed by RRT-Connect algorithm

This is the pic of the two paths viewed in the GUI(the original path of the end-effector computed by the RRT in red and the shortcut-smoothed path of the end-effector in blue)

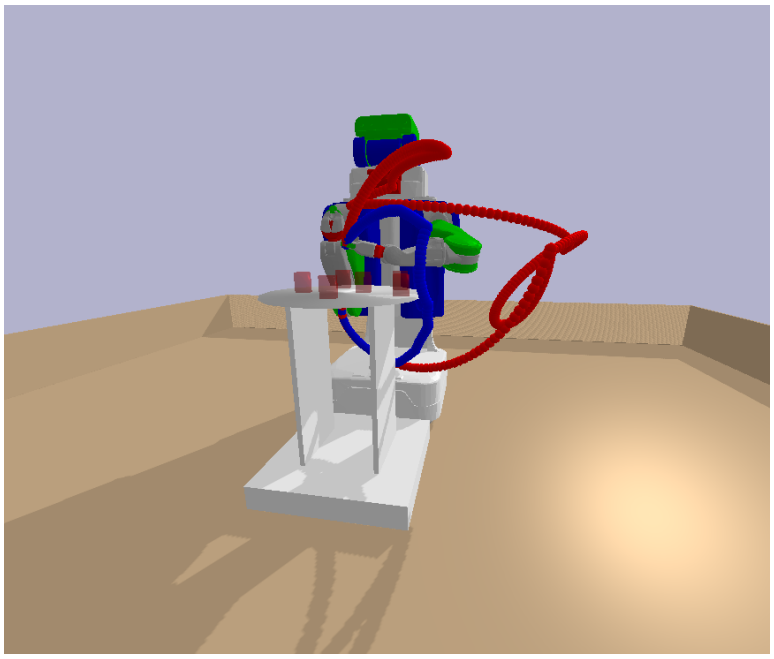


Figure 6: The screenshot showing the path computed by the original path and shortcut-smoothed path

(b) Here, we zoom in and see if the arm indeed reaches the goal

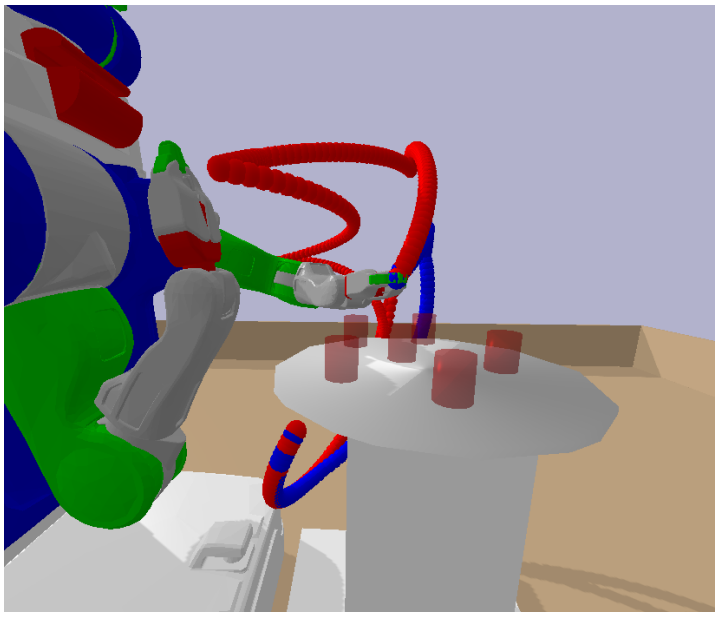


Figure 7: The final arm position