CS1207: Assignment4

Path Design for Autonomous Driving

TA: *Yao Xu*

Credit: *AI 3603 Artificial Intelligence: Principles and Techniques, 2022Fall*

Path planning is a significant algorithm in autonomous driving and many other fields, in this lab-assignment, you should implement the *A* algorithm* to find the optimal path for a car to drive from the start point to the end point.

Generally, you should learn the idea of *heuristic* and concatenate *Dijkstra* algorithm and *greedy* algorithm to implement the *A* algorithm* in `python` language and visualize your results. You may spend a lot of time on improving the efficiency and making the path smooth.

Please complete the assignment **individually**.

1 Introduction

In this assignment, you will develop a path planning framework for a service robot in the unknown environment using *A* algorithm*. Suppose an autonomous robot `DR20` exploring in an unknown scene. The global map is **unavailable** for the robot at the beginning of exploration, but a laser scanner mounted on robot's body is utilized to scan the obstacles around the robot. The robot can gradually build the map as it explores, util it reaches the goal, which is shown as the gray block.

This assignment will be implemented in the CoppeliaSim simulator. A quick-start guide is provided for CoppeliaSim in the file list. For more details, please refer to `https://www.coppeliarobotics.com` and `https://www.coppeliarobotics.com/helpFiles/index.html`.

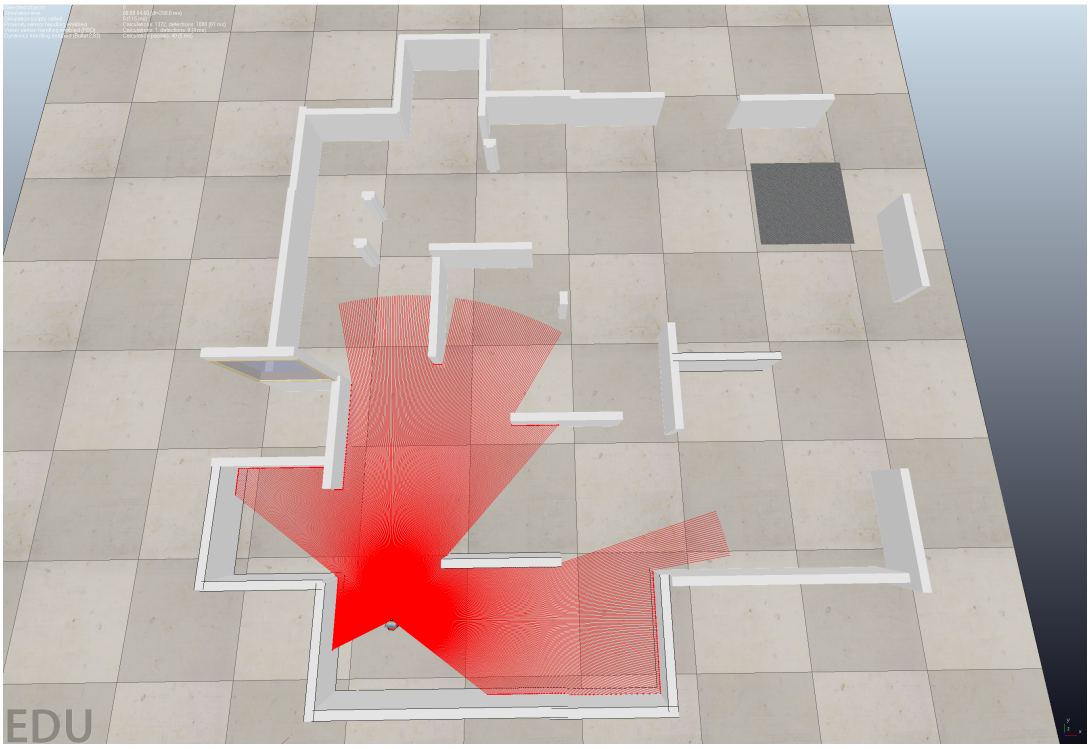

图 1-1 robot in the unknown environment

Provided File List

1. `CoppeliaSim Installer`: please ask TA for the file;
2. `Assignment.pdf`: this file, including requirements of this assignment;
3. `DR20-API`: API for DR20 in CoppeliaSim.
4. `Search.ttt`: the scene file in CoppeliaSim;
5. `Example.py`: an example code for CoppeliaSim and the APIs;
6. `Task_1.py`: the code for basic A* algorithm to be completed;
7. `Task_2.py`: the code for improved A* algorithm to be completed;
8. `Task_3.py`: the code for self-driving planning algorithm to be completed;

Submission File List

1. 3 `python` scripts: **SEPARATELY** named as `Task_1.py`, `Task_2.py` and `Task_3.py`;
2. 3 video demos: **SEPARATELY** named as `Task_1.mp4`, `Task_2.mp4` and `Task_3.mp4`,

   You need to speed it up (if neccessary) **WITHIN** 10 sec;
3. `RunningTime.md` or `.txt`: report your running time of the robot from start to termial;
4. `<StudentID>-<StudentName>-Report.pdf`: a *well-organizaed* report,

   TA will check your report to see whether you have understood the algorithm and your implementation is correct.

2 Preliminaries

2.1 Basic Grammar of `Python`

You should install python (better with version $\geq$ `3.7.0`) and learn the basic grammar of `python` yourself.

2.2 `Dijkstra` Algorithm

`Dijkstra` algorithm is a classical algorithm for finding the shortest path between two nodes in a graph. It is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

However, consider the shortest-path tree, we will find it so 'width' that it will waste time to search for useless paths because we know where the terminal is.

So we need to add some *heuristic* to guide the search.

2.3 Heuristic Idea for Algorithm Design, A^* Algorithm

Heuristic

In computer science, artificial intelligence, and mathematical optimization, *Heuristic* is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed.

Intuitively, since we know where the terminal is, we can directly use *Greedy* algorithm which will directly lead the robot to the terminal.

Of course, as we all know, *Greedy* is not always correct and in this setting we have to consider the **obstacles**.

So we can concatenate *Dijkstra* algorithm and *Greedy* algorithm to implement the *A* algorithm.*

A* Algorithm

Let $g(v)$ denotes the distance calculated by *Dijkstra*, $h(v)$ denotes the distance calculated by *Greedy*, then we have the sense that:

- $g(v)$ is the distance from the start node to node v;
- $h(v)$ is the distance from node v to the terminal;

Then we estimate the 'total distance' by $f(v) = g(v) + h(v)$, and we will choose the node with the smallest $f(v)$ to expand.

And this is exactly the *A* algorithm.*

3 Task1: Basic A* Algorithm [50pts]

In Task1, you will implement basic A* algorithm for the autonomous robot DR20. The coordinate of the goal is known for the robot but the global map is unavailable at the beginning. You can control the robot move **forward**, **backward**, **left** and **right**. As long as the robot enters the gray block, the mission is considered successful.

4 Task2: Improved A* Algorithm [10pts]

Now, you can ALSO control the robot move towards **upper left**, **upper right**, **bottom left**, **bottom right**.

Besides, you should also consider **distance** between the robot and the obstacles to avoid real-world collision and the **cost** of turning steering.

Perhaps, there is a trade-off that you can determine how to deal with it yourself.

5 Task3: Self-Driving Planning [40pts]

In this task, you should think about the concerns of real-world drivers.

You can improve your algorithm in the following perspectives:

1. [Required, 10pts] *Smooth* path design;
2. [Required, 10pts] *Optimization* of path planning speed;
3. [Optional, 20pts] Other *creative* ideas.

You should implement the required parts and you should also implement one or more ideas in the optional part for more points!

6 Grade Policy

1. **Basic A* Algorithm**, 50pts: You can easily get 50pts as long as you can successfully implement the basic A* algorithm;
2. **Improved A* Algorithm**, 10pts: TA will grade your work by the visualized path, not by running or other factors;
3. **Self-Driving Planning**, 40pts: Points will differ here considering the path, running time and other issues.

Besides, your report, on-time submission and daily communication with TA will give you ±20pts added or deducted.

7 Suggested Time-line

- **Tue., Week1 to Tue., Week2**: Basic A* Algorithm;
- **Tue., Week2 to Fri., Week2**: Improved A* Algorithm;
- **Tue., Week3 to Tue., Week4**: Optimization of Self-Driving Planning;
- **Thu., Week4 to Fri., Week4**: Demo recording.

Meanwhile, please schedule your **report writing**.