ME5413 Autonomous Mobile Robotics

Homework III Planning

Submitted by Homework Group 6

LI Peizhuo	Matrix Number
YU Zhuoyuan	Matrix Number
ZHONG Ningze	Matrix Number

AY2023/2024 (SEMESTER 2)

27th, Mar.,2024

TASK 1

In this chapter, we implement an A* planning algorithm and try different heuristic functions to improve search efficiency and path quality. Then we try to search from both ends, using the Bidirectional A* algorithm which decreases the total run time of the algorithm. Finally, we use the best-performed algorithm to calculate the distance between any two locations.

1. A* Algorithm

A* algorithm is a heuristic search algorithm commonly used in path search and graph search. It combines breadth-first search and heuristic evaluation functions to find the best path from the starting point to the endpoint.

1.1. Implementation Details

Our algorithm is divided into several parts of functions. The "is_walkable()" function determines whether a node is traversable by checking if there are any black pixels within a 0.3m range around the next node. "heuristic()" will be detailed in the next section. "a_star_search()" is the main function of implementing A* algorithm. For each node, A* algorithm calculates the aggregate cost (f-value) of its eight neighbor, by combining the actual and estimated costs:

$$f(n) = g(n) + h(n)$$

where g-value is the actual cost from the starting point to that node, and h-value is the estimated cost from that node to the destination node. Then select the node with the lowest integration cost from all the nodes that have not yet been explored as the next one to be explored. For a selected node, explore its neighbors and calculate their costs. If a shorter path to a node is found, update the path information for that node, and repeat until that goal node is found.

1.2. Attempts on Heuristic Function

Euclidean Distance is the straight-line distance between two points, that is, the length of the shortest path between two points in a two-dimensional plane. The Euclidean distance is a more accurate heuristic function but may consume more computational resources than the Manhattan distance in practice.

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

where (x_n, y_n) are the coordinates of the current node and (x_g, y_g) are the coordinates of the goal node.

Manhattan Distance is the right angular distance of two points on the grid, which is the total number of moves along the horizontal and vertical directions of the grid.

$$h(n) = |x_n - x_g| + |y_n - y_g|$$

Chebyshev Distance is the length of the diagonal path between two points. This heuristic function is often used on grid maps that allow diagonal movement, for which diagonal movement leads to a faster arrival at the target point.

$$h(n) = \max(|x_n - x_g|, |y_n - y_g|)$$

In this project, taking the path from the start to snacks as an example, the heuristic function

of Euclidean Distance performed best is 143.75m, and the path length is the shortest. The path using different heuristic functions is shown in Fig 1.

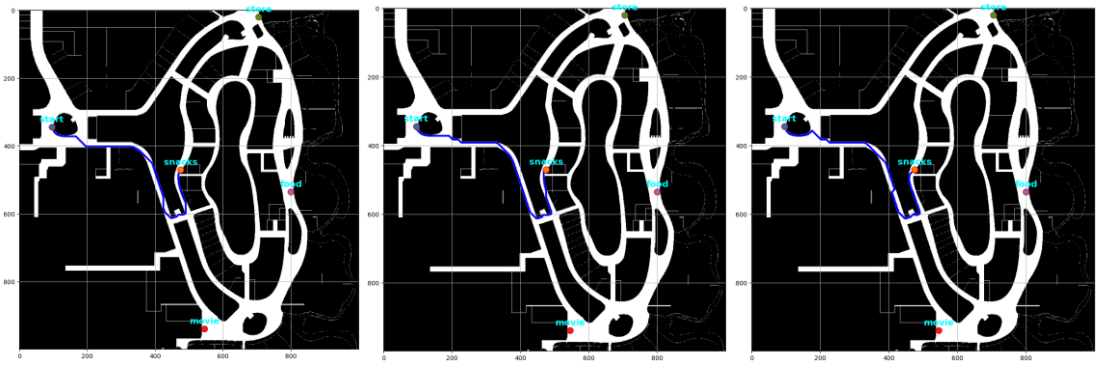

Fig 1 Total path distances of Euclidean Distance, Manhattan Distance and Chebyshev Distance are 143.75, 144.93 and 149.49m

1.3. Bidirectional A* Algorithm

The bidirectional A* algorithm is an optimized form of the A* algorithm, which starts searching from the starting point and the ending point at the same time until the two search paths meet. Compared with the traditional A algorithm, the bidirectional A algorithm can find the shortest path faster. By using this algorithm, our running time for the path from start to snacks has decreased **from 8s to 4s**, which is essentially half of the original time. Since the search results are basically consistent with the A* algorithm, they will not be displayed here due to the length of the content. For details, please refer to the code.

2. Difficulties and Solutions

Problem 1: The agent moves into narrow gaps as shown in Fig 2, but although these gaps are white pixels, they are not viable paths.

Fig 2 Agent moves into a narrow gap

Solution: This is due to the neglect of the agent's collision volume. Therefore, considering the agent's radius as 0.3m in the calculation can solve the problem.

Problem 2: There is a small defect at some turning, which seems to have gone further around as shown in Fig 3.

Fig 3 Detour at a turning

Solution: This problem could be solved by using the heuristic function with dynamical

adjustment of the weights.

3. Result

Here are the results of shortest distances using different heuristic functions.

Table 1 the shortest distances of Manhattan Distance between each pair of locations

	start	snacks	store	movie	food
start	0	144.19	161.84	217.26	226.80
snacks	144.18	0	122.31	110.07	140.39
store	161.84	122.31	0	217.26	110.92
movie	179.49	110.15	217.26	0	122.90
food	226.88	140.39	111.00	122.90	0

Table 2 the shortest distances of Euclidean Distance between each pair of locations

	start	snacks	store	movie	food
start	0	143.75	155.61	179.15	225.40
snacks	143.75	0	115.44	108.35	135.02
store	155.61	115.44	0	210.14	111.12
movie	179.15	108.35	210.14	0	113.83
food	225.40	135.02	111.12	113.83	0

TASK 2

In this section, we will address the Traveling Salesman Problem(TSP) using the distances between different nodes obtained from the previous section. Specifically, we aim to find the shortest path that starts at 'start', traverses through all the nodes once, and returns to 'start'.

1. Load the data via Networkx

First, record the data shown in Table 2 in the format of an adjacency matrix, then use networkx to transform it into a multidigraph as depicted in Figure 4. A multidigraph can store the weights of edges as well as accommodate bidirectional routes, thereby facilitating the search for adjacent nodes to each node and the calculation of distances. This makes it well-suited for TSP.

Fig 4 Multidigraph of five nodes

2. Brute-Force Approach

After loading the data, the first algorithm we employ is the brute force method, which involves enumerating all eligible paths and selecting the shortest one. The advantage of this algorithm is that it guarantees the result is the shortest possible, which facilitates the tuning of

parameters for other algorithms we may use subsequently.

The shortest path obtained from the data in Table 2 is as follows:

`start` → `store` → `food` → `movie` → `snacks` → `start`

The total distance according to this path is 632.66m.

3. Simulated Annealing Algorithm

The Simulated Annealing algorithm is inspired by the principle of annealing in solid-state physics. It is a probability-based algorithm that heats a solid to a sufficiently high temperature and then allows it to cool slowly. As the temperature rises during heating, the particles within the solid become disordered, increasing the internal energy. As the solid gradually cools, the particles move towards an ordered state, reaching equilibrium at each temperature level. Finally, at room temperature, the system reaches its ground state, and the internal energy is minimized.

Simulated Annealing is actually a type of greedy algorithm, but it introduces randomness into the search process. It accepts a solution that is worse than the current one with a certain probability. Therefore, it has the potential to escape local optima and reach a global optimum.

Thus, we used the data from Table 2 to perform calculations, and adjusted the initial and final temperatures based on the results of the brute force algorithm. The final outcome was the same as that obtained by the brute force method.

4. Result

The final shortest route is shown in Fig 5, and the total distance is 632.66m.

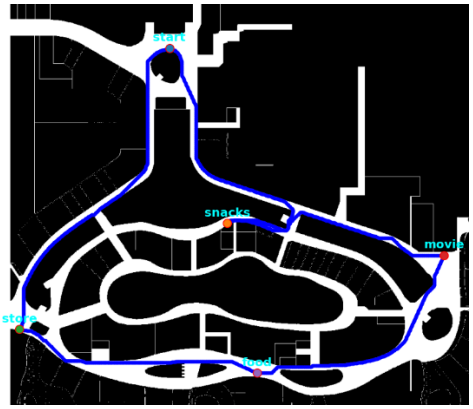

Fig 5 The Final Shortest Route

5. Comparison

5.1 Running Speed

Regarding the execution speed, the Simulated Annealing algorithm is far superior to the brute force algorithm, as the latter requires traversing all possible paths. In more complex networks, the number of such cases can be quite large. Therefore, the brute force approach is more suitable for networks with lower complexity. For situations with higher complexity, it is recommended to use heuristic algorithms such as Simulated Annealing.

5.2 Accuracy

In terms of accuracy, as previously mentioned, the brute force algorithm can guarantee that the result obtained is the global optimum. However, this is not the case with the Simulated Annealing algorithm; the outcome it provides is a local optimum, which may not necessarily be the global optimum. Therefore, if the requirement is to obtain the global optimum, the brute force algorithm would be more accurate compared to other algorithms.

TASK 3

GitHub repository: https://github.com/TWpoint/ME5413.git

In this section, we try to use Nonlinear Model Predictive Control (NMPC) to track the trajectory. NMPC is an advanced control method that combines Model Predictive Control (MPC) and nonlinear system control technology.

In this method, the algorithm accepts the current state and trajectory instructions of the robot and makes predictions based on the nonlinear dynamics model of the system to generate speed instructions, so that the robot can move along the expected trajectory. Compared with traditional PID control, NMPC can better handle nonlinear systems and nonlinear constraints, to achieve more accurate control in complex environments.

1. Method

In order to implement this algorithm, based on the open-source mpc_ros repository by Geonhee Lee on GitHub, we made modifications to the input and output topics to enable communication control in our environment. This approach accepts the current state of the robot (direction, velocity, position) and path commands, generating velocity commands (linear velocity in the x-direction, yaw) to control the robot.

The method begins by setting the constraints of the problem. These constraints include initial state constraints and system dynamic model constraints. Initial state constraints ensure that the vehicle starts from its current state. System dynamic model constraints, based on the vehicle's dynamic model, ensure that the vehicle moves according to the model at each time step.

During the solving process, this method utilizes a series of cost functions and employs the IPOPT library for solving. IPOPT iteratively optimizes until it finds the optimal solution satisfying the constraints or reaches the maximum iteration count.

In addition, we also found that the mpc_ros method cannot effectively track the speed, so we clip the linear velocity output with the target velocity. We found that this clip method does not significantly reduce the tracking accuracy of position and heading, but it can significantly improve the tracking effect of the method on speed.

2. Result

We conducted tests on the modified mpc_ros implementation. As shown in Figure X, the method demonstrates effective tracking during operation. However, during the startup phase, it may require a certain distance to converge to the desired trajectory. Additionally, we observed that the method's performance diminishes during large angle turns, often necessitating a larger turning radius. Further optimization of control parameters and trajectory planning strategies may enhance the system's performance and robustness.

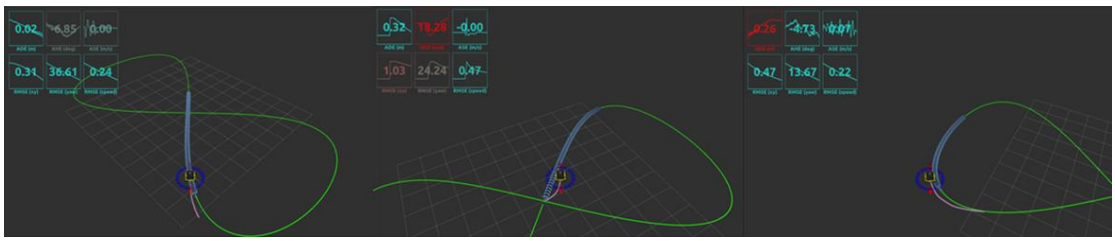

Figure 6 MPC Tracking Results: Left (Operation), Middle (Start), Right (Large Angle Turn)