7CCEMROB Robotics Systems 2022-2023 Coursework 2 - MOTION PLANNING

Haotian Li (21155223): is responsible for the task1 and task4

Nanjun Pan (21166698): is responsible for the task3

Xintian Liu (21160325): is responsible for the task2

Yajing Zhang (21167579): is responsible for the task3

Yu Su (21154771): is responsible for the task2

March 26, 2023

Contents

1 Abstract

This report contains literature review, task solving, discussion, and conclusion, on the aspect of motion planning, which fully demonstrates our learning and solving process on tasks in coursework 2.

2 Introduction

The following report is divided into three sections. In the "Background and Literature Review" section, the terminology, technology fields, and existing research of motion planning are introduced. Then, the required tasks and questions are solved following the route defined before in the section "Task Solving". After that, the comparison and discussion on different path planning algorithms are also listed in the section "Task Solving", which contains noteworthy attentions and innovative ideas. All work and result on tasks are presented logically and smoothly. Last but not least, the "conclusion" part analyzes the potential achievements and sums the above parts concise and comprehensive.

3 Background and Literature Review

Motion planning problems occur in different fields such as robots, assembly analysis, virtual prototyping, and computer animation, which involves searching the collision-free path connecting a given start and goal configuration in the system configuration space of one or more complex geometries while meeting the constraints imposed by various obstacles [1]. To deal with it, the path planning algorithms emerge as the times require.

Rapidly-Exploring Random Trees (RRT), which is a single query algorithm based on random sampling, constructs the connected graph by uniformly sampling the state space randomly, without explicit modeling of the free area of the state space, and the feasibility of the trajectory can be verified by collision detection. In addition, RRT-Connect introduces a two-tree expansion procedure based on RRT, which expands the random tree with the start point and goal point as the root nodes respectively, in order to make the searching faster [1].

Probabilistic Roadmap Method (PRM) is a path planning method based on random sampling strategy, which can solve the problem that effective paths are difficult to construct with most algorithms in high-dimensional space. The method proceeds in two phases: a learning phase and a query phase [2]. PRM motion planning techniques were applied to a number of challenging problems arising in a variety of field.

At the same time, these new applications served to point out some weaknesses in the PRM approach, so a number of PRM variants specifically targeted at this problem have been proposed. R. Bohlin and L. E. Kavraki (2000) proposed a new approach to probabilistic roadmap planners (PRMs). The overall theme of the algorithm, called Lazy PRM, is to minimize the number of collision checks performed during planning and hence minimize the running time of the planner [3].

A* (ASTAR) is created in 1968 to integrate formal methods like Dijkstra's Algorithm with heuristic methods like Best-First-Search [4]. While using Dijkstra's Algorithm, vertices in the graph are visited beginning from the starting point of the item. The closest unexplored vertex is then repeatedly searched, adding its vertices to the list of vertices to be investigated. At the initial place, it spreads outward until it reaches the destination. Similar to this, the Best-First-Search method operates with the exception that each vertex's distance from the target is estimated. It chooses the vertex closest to the objective rather than the one closest to the starting point. Dijkstra's Algorithm is guaranteed to find the shortest path from the starting point to the goal while the Best-First-Search is guaranteed to find paths very quickly [5].

4 Task Solving (Work & Result)

4.1 Task 1 (30%)

The corresponding pseudo-codes of RRT and RRTC are below:

Algorithm 1 EXTEND(τ, q)

```
1: q_near ← NEAREST_NEIGHBOR(q, τ);
2: if NEW_CONFIG(q, q_near, q_new) then
3:     τ.add_vertex(q_new);
4:     τ.add_edge(q_near, q_new);
5:     if q_new = q then
6:         return Reached;
7:     else
8:         return Advanced;
9: return Trapped;
```

Algorithm 2 BUILD_RRT(q_{init})

```
1: τ.init(q_init);
2: for k=1 to K do
3:     q_rand ← RANDOM_CONFIG();
4:     EXTEND(τ, q_rand); # Algorithm 1
5: return τ
```

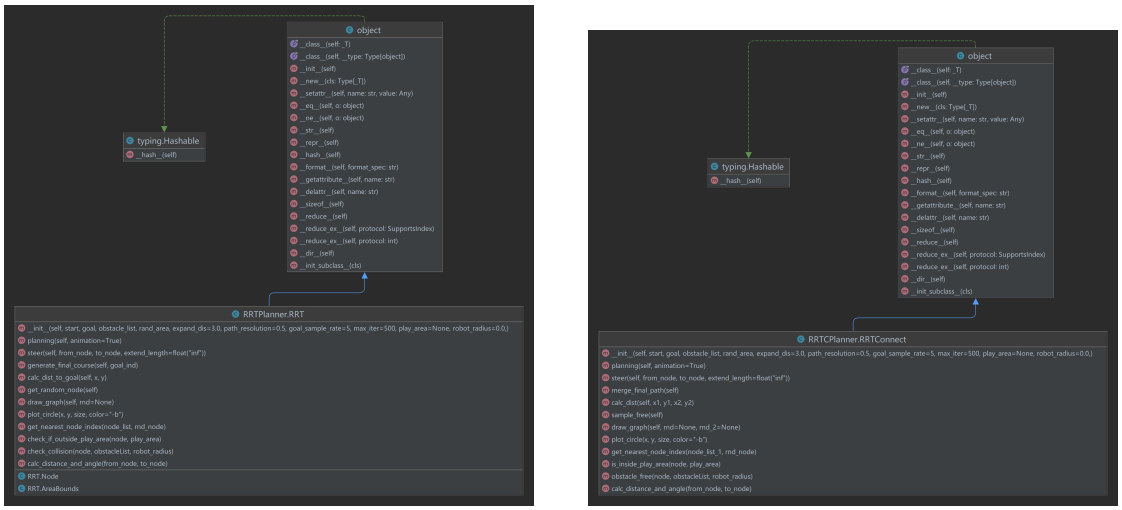
Algorithm 3 CONNECT(τ, q)

repeat S $\leftarrow$ EXTEND(τ, q); # Algorithm 1
until not (S = 'Advanced')
return S;

Algorithm 4 RRT_CONNECT_PLANNER(q_{init}, q_{goal})

τ_a.init(q_{init}); τ_b.init(q_{goal});
for k=1 to K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    if not EXTEND(τ_a, q_{rand}) = 'Trapped' **then** # Algorithm 1
        if CONNECT(τ_b, q_{new}) = 'Reached' **then** # Algorithm 3
            return PATH(τ_a, τ_b);
    SWAP(τ_a, τ_b);
return Failure

The UML class diagrams which illustrate the code structure of RRT and RRTC are shown below, in Figure 1:



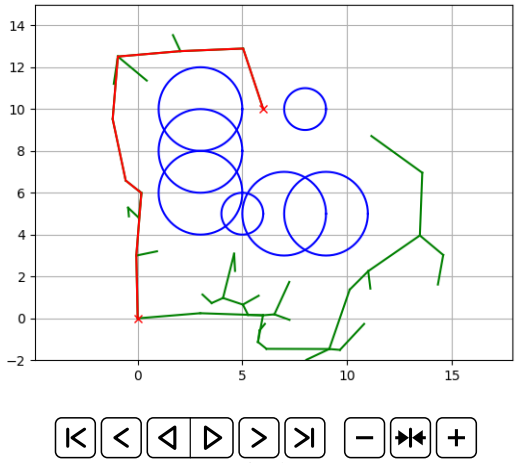
(a) The code structure of RRTPlanner

(b) The code structure of RRTCPlanner

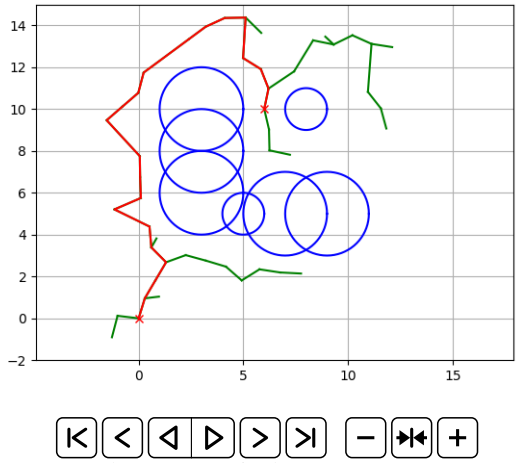
Figure 1: The code structure (RRT and its variants)

The RRT-Connect creates two node lists instead of one in RRT, balances the growth of two trees in the function "planning", and merges them in the function "merge_final_path".

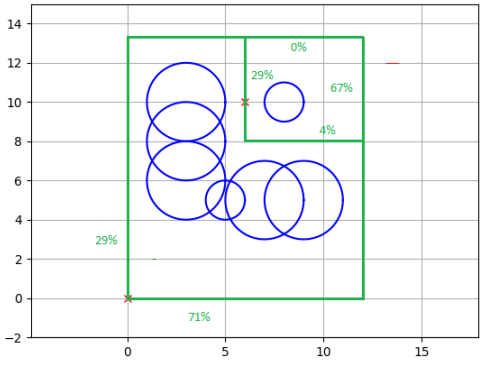
Since RRT and RRTC are single query algorithms, there is various kind of RRTtree in the final state, some typical cases are in Figure 2 below. To better classify them, we allocate the final state of RRTtree and RRTCtree (in the case of "path found") into four classes, detected by two factors: 1. the first direction after leaving of the start point - go up or right; 2. the behaviour to the farthest obstacles(8, 10, 1) - bypass or not. The Figure 2 also demonstrates the appearance percentage of each kind of the final state while the Figure 3 illustrates the most common case of the final state of RRTtree and RRTCtree (*Note: the size of the dataset is 100*).



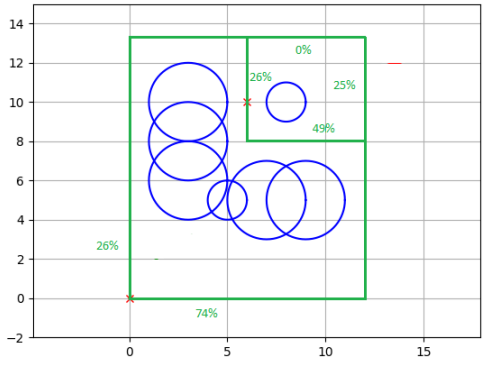
(a) Some typical cases (RRT)



(b) Some typical cases (RRTC)

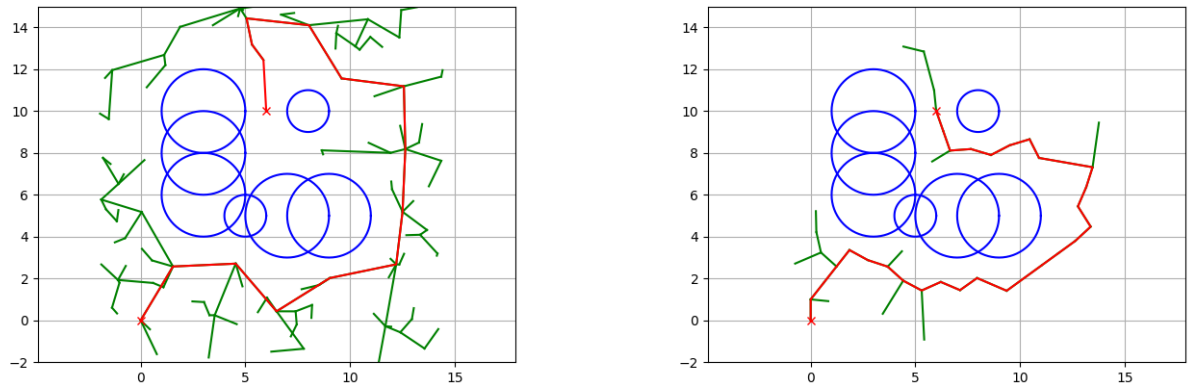


(c) The appearance percentage (RRT)



(d) The appearance percentage (RRTC)

Figure 2: Some typical cases and the appearance percentage (RRT and its variants)



(a) The most common case (RRT)

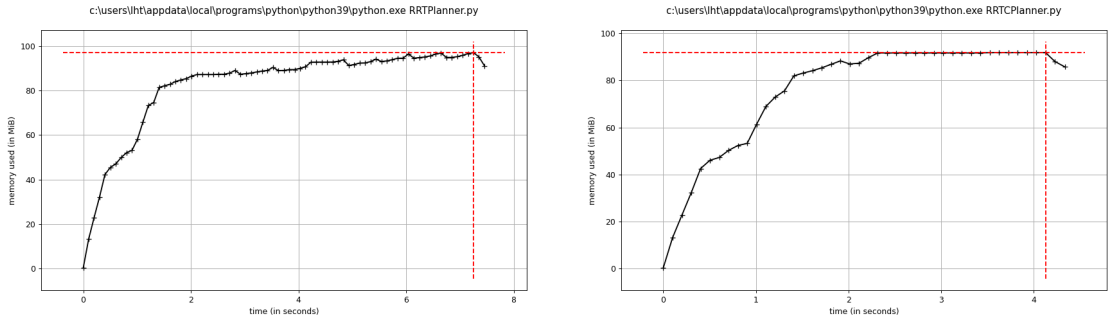
(b) The most common case (RRTC)

Figure 3: The most common case (RRT and its variants)

To evaluate the performance, the Table 1 shows the average time cost and path length (total 100 times) and the Figure 4 presents the memory usage in one run.

Performance Evaluation	Average time cost	Average Path Length
RRT	6.44s	31.1345
RRTC	1.54s	30.1784

Table 1: The performance evaluation (RRT and its variants)



(a) The memory usage (RRT)

(b) The memory usage (RRTC)

Figure 4: The memory usage (RRT and its variants)

4.2 Task 2 (30%)

The corresponding pseudo-code is below:

Algorithm 5 Roadmap Construction Algorithm

Input:
n: number of nodes to put in the roadmap
k: number of closest neighbors to examine for each configuration
Output:
A roadmap G = (V, E)

```
1:  G.init();
2:  while |V| <n do
3:      repeat
4:      q ← RANDOM_CONFIG();
5:      until q is collision-free
6:      V ← V ∪ q
7:  for all q ∈ V do
8:      N_q ← the k closest neighbors of q chosen from V according to dist
9:      for all q' ∈ N_q do
10:         if (q, q') ∉ E and Δ (q, q') ≠ NIL then
11:             E ← E ∪ {(q, q')}
```

Firstly, the PRM initialize two sets, where V denotes the set of random points and E denotes the set of paths. And then it sample one collision-free point at a time at random and insert this collision-free point to V and repeat n times. A probabilistic road map is then generated. Next for each point q in V, k neighbourhood points are selected based on a certain range of distances. and for each domain point q', if q and q' do not yet form a path, they are connected to form a path, followed by collision detection, and if there is no collision, the path is retained. After constructing the roadmap according to the above steps, the shortest path from the starting point to the end point is searched in the roadmap using Dijkstra's algorithm. **Since there are two plannings, the code structure is differ from others.**

The UML class diagram which illustrates the code structure is shown below, in Figure 5:

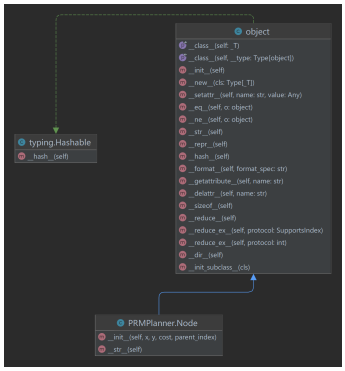


Figure 5: The code structure of PRMPlanner

Although PRM is a single query algorithm, there is a limited kind of PRmap in the final state since the Dijkstra always choose the local shortest path. The Figure 6 is the most common case of the final state of PRmap (*Note: the size of the dataset is 10*).

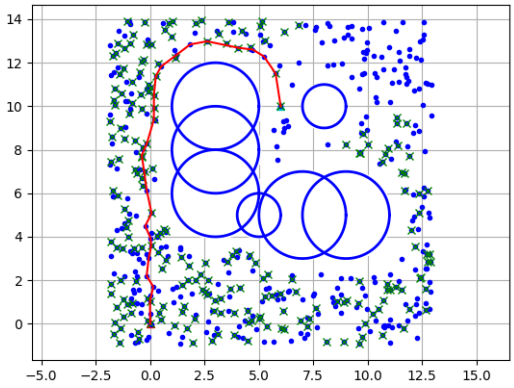


Figure 6: The most common case (PRM)

To evaluate the performance, the Table 2 shows the average time cost and path length (total 10 times) and the Figure 7 presents the memory usage in one run.

Performance Evaluation	Average time cost	Average Path Length
PRM (n=500)	21.30s	22.3317
PRM (n=1500)	42.07s	20.9249

Table 2: The performance evaluation (PRM)

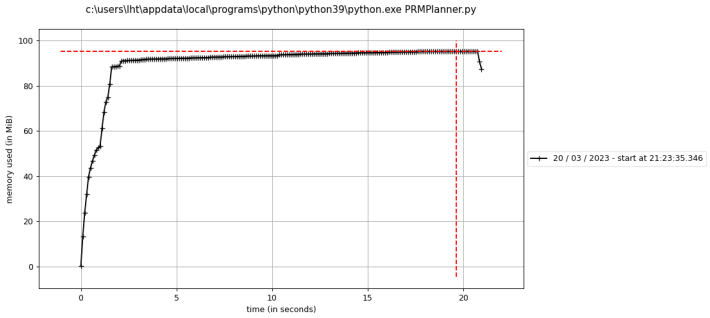


Figure 7: The memory used (PRM)

4.3 Task 3 (20%)

The corresponding pseudo-code is below:

Algorithm 6 A* Construction Algorithm

Input:
O: the priority queue
C: the nodes that have been visited.
Output:
A global shortest path from start to goal

repeat
Pick n_{best} from O such that f(n_{best}) $\leq$ f(n), $\forall$ n $\in$ O
Remove n_{best} from O and C
if $n_{best} = q_{goal}$ **then**
    EXIT
Expand n_{best}:
for $\forall$ x $\in$ Start(n_{best}) that are not in C **do**
    if x $\notin$ O **then**
        O $\leftarrow$ O $\cup$ {x}
    elseif g(n_{best}) + c(n_{best}, x) $<$ g(x) then
        Update x's back-pointer to point n_{best}
until O is empty

First put the starting point in the set O. If O is not empty, repeat the following work:

1. Find the point n in O with the least cost leading to the target point as the current point;

2. Delete the current point n from O and add it to C;

1) If the current point n is the target point, then exit the loop and set the found path flag;

2) If the current point n is not the target point, traverse each neighboring point of point n. If the neighboring point x is not blocked by an obstacle and is not in C, the process is as follows:

a) If the node x is not in O, calculate the cost from node x to the target node, take node n as the parent node of node x, and add node x to O;

b) If the node x is in O, then compare the cost of node x and node n. If the cost of node n is smaller, then take node n as the parent node of point x. And recalculate the cost of point x leading to the target point, and reorder O;

After completing the above steps, start from the target point, search for the parent node of each node in turn, until the starting point is found. Return the final target path.

The UML class diagram which illustrates the code structure is shown below, in Figure 8:

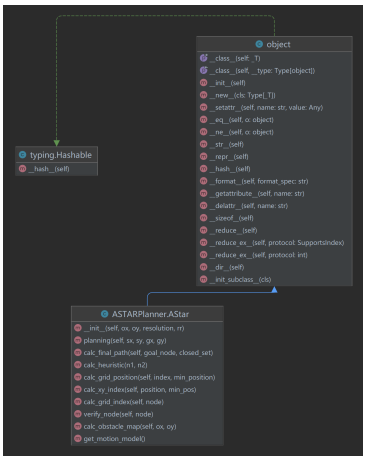


Figure 8: The code structure of ASTARPlanner

Astar is not a single query algorithm, there is only one kind of the final state since the astar always choose the global shortest path. The Figure 9 is the most common case of the final state (*Note: the size*

of the dataset is 10).

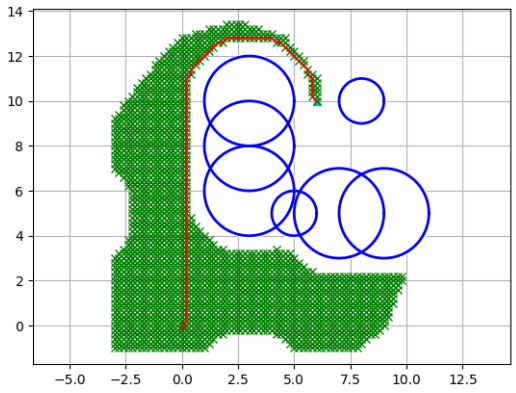


Figure 9: The most common case (ASTAR)

To evaluate the performance, the Table 3 shows the average time cost and path length (total 10 times) and the Figure 10 presents the memory usage in one run.

Performance Evaluation	**Average time cost**	**Average Path Length**
A*	37.10s	19.4912

Table 3: The performance evaluation (PRM)

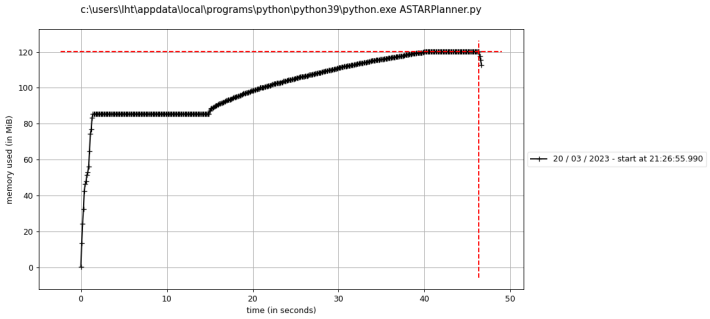


Figure 10: The memory used (ASTAR)

4.4 Task 4 (20%)

RRT and its variants: compared to the single tree expansion RRT algorithm, RRT-Connect introduces dual tree expansion, which reduces the blindness of search space to increase the search speed. After considering all algorithms, they have the following features:

- **Pros:** relatively low time and space complexities.

- **Cons:** its result is usually not the optimal path.

PRM: compared to RRT and its invariants, PRM introduces the roadmap instead of a tree. The involved Dijkstra's algorithm generates more calculations on finding the final path since it doesn't similar to RRT, which stops as long as a single path is found, but finishes 500 "iterations" and returns the local shortest path. However, the Dijkstra's algorithm usually is worse than A* since it doesn't have heuristic step. So, if the number of random points increases from 500 to 1500, which should be the same scope to A* since the grid size is set to 0.2, the running time would no longer be the advantage of PRM (average running time: 42s).

- **Pros:** relatively low space complexity and local optimal path.

- **Cons:** it takes more time than RRT and RRTC but its result is still not globally optimal.

A*: compared to the above algorithms, A* reduces the randomness, and introduces the heuristic step, which indicates the distance to the next goal point in the grid space. As a result, the direction of exploration is facing the goal point, the calculation overhead is mainly decided by the grid size, and the space complexity would larger than before to store more temporary information. In the case statement, the goal point is right behind the obstacles, so the running time would also increase due to the unkind of case settings. However, the result is always optimal.

- **Pros:** its result is always the global optimal path, if exists.

- **Cons:** relatively high time and space complexities.

Note: the total running time include the pausing time while plotting. Obviously, the PRM and A require more steps while plotting, so we reduce the pausing time from 10ms to 1ms. Without pausing, the calculation time difference of finding final path between the above algorithms is very little since the configuration space isn't complex enough.*

5 General Conclusion

This coursework is a practical application of the theoretical knowledge of motion planning. The students are able to implement the corresponding algorithm and performance testing, as well as generate the pseudo-code. It also improves our critical thinking skills during the analysis in evaluation and comparison steps, which enhances our understanding of these path planning algorithms.

6 Reference

[1] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, USA, 2000, pp. 995-1001 vol.2, doi: 10.1109/ROBOT.2000.844730.

[2] L. E. Kavraki, P. Svestka, J. C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566-580, Aug. 1996, doi: 10.1109/70.508439.

[3] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, USA, 2000, pp. 521-528 vol.1, doi: 10.1109/ROBOT.2000.844107.

[4] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science andCybernetics, vol. 4, no. 2, pp. 100–107, 1968, doi: https://doi.org/10.1109/tssc.1968.300136.

[5] A. Patel, "Introduction to A*," Stanford.edu, 2019. http://theory.stanford.edu/ amitp/GameProgramming/AStarComparison.html

7 Appendix (Individual Page of the Group Representative)

Before describing my personal contribution, learned experience, and achievement, I want to clearly state that all members in my group are indispensable, and the absence of any one would not have made the report as great as it is now.

I first complete the RRTC based on the RRT, then design the remaining methodology and hold a meeting to share it with other team members, analyze the pseudocode and code structure, set up a to-do list for team members, and provide a reference website called Python Robotics[1] and code to ensure that the assigned tasks can be better completed, preventing future rework due to inconsistent formats and other issues. At the same time, I propose some possible indicators for evaluating performance and received positive feedback.

As soon as they complete the algorithm, I start adding code for the test indicators, and testing RRT and RRTC. After the test was completed, I join the work of the two algorithms. After delivering those two sets of code, I add the test code, export statistical results, and make preliminary comparisons to obtain preliminary conclusions. After handing them the complete code and preliminary conclusions, we will browse the corresponding references together, refine our conclusions, and compile them for me to complete the documentation. Finally, the LaTeX layout report is available at (link only readable):

https://www.overleaf.com/read/tqzkqfqbgwfg

And some procedural files have been uploaded to the Github:

https://github.com/lihaotian1102/KCL_ROB_CW2

[1]Python Robotics: Robotics in Python, includes documentation and coding of path planning algorithms, available at: https://github.com/AtsushiSakai/PythonRobotics.