ECE276B HW2 Project Report

- ## Introduction

Motion planning, also known as the piano mover's problem, aims to find a sequence of valid configurations that moves the robot from source to destination. It is of great importance in the field of robotics for providing collision-free (and sometimes optimal) movements that robot can follow. In this project, a 3D configuration space with boundaries and obstacles is given, and we are trying to figure out a valid path (if there exists one) for any source and destination of our zero-volume robot. Two typical search-based and sample-based algorithms are implemented to solve this problem:

1. **Real-Time Adaptive A* (RTAA) Algorithm:**
   An agent-centered search algorithm based on A* algorithm, which has strict constraint on the search horizon of every timestamp.
2. **N-Directional Rapidly-Exploring Random Tree (NRT) Algorithm:**
   A variant of Rapidly-Exploring Random Tree (RRT) algorithm, which has in total n trees grow simultaneously to explore the configuration space.

This report describes these two approaches and compare their performance on specific searching tasks.

- ## Problem Formulation

Given the following world configuration:

C_{free} – Free Space

C_{obs} – Obstacle Space

C_S – Start Space

C_T – Target Space

Formulate a deterministic finite state problem with $T := |C_{free}|-1$ stages:

State space: $X_0 := \{C_s\}$, $X_T := \{C_T\}$, $X_t := C_{free} \backslash \{C_T\}$ for t = 1,…,T-1

Control space: $U_{T-1} := \{C_T\}$, and $U_t := C_{free} \backslash \{C_T\}$ for t = 0,…,T-2

Motion model: $x_{t+1} = u_t$ for $u_t \in U_t$, t = 0,..,T-1

Stage cost: $l_t(x_t, u_t) = c_{xt,ut}$ for t = 0,…,T-1

Terminal cost: $q(C_T) := 0$

Our goal is to construct an optimal control sequence (path) $u_{0:T-1}$ such that:

$$\min_{u_{0:T-1}} \; q(x_T) + \sum_{t=0}^{T-1} \ell_t(x_t, u_t)$$
$$\text{s.t. } x_{t+1} = f(x_t, u_t), \; t = 0, \ldots, T-1$$
$$x_t \in \mathcal{X}, \; u_t \in \mathcal{U}(x_t),$$

And our solution should satisfy the following constraints:

1. The robot moves a small distance at each time step, i.e. $||x_{t+1}-x_t|| \leq 1$;

2. The path between x_t and x_{t+1} should be collision-free;
3. The next position x_{t+1} is produced within 2 seconds.

The solution is evaluated based on its path length (optimality) and planning time (efficiency).

- ## Technical Approach

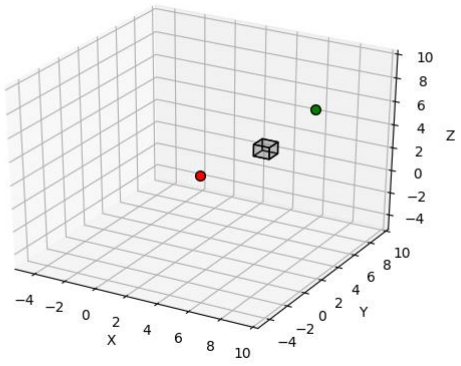


Fig. 1

Fig. 1 demonstrates a simple case of world configuration, red dot is the current robot position while the green dot is the target position. The grey rectangle is the obstacle that we should avoid while moving the robot towards the target.

1. Real-Time Adaptive A* Algorithm

Map Discretization

Since the A* is a search-based planning algorithm, it is necessary to discretize the continuous world space to avoid over-exhaustive search. We discretize the world space according to the following criterion:

a. Set a maximum resolution w1 to be 0.5
b. Search through all the walls(obstacle), find the minimum wall width w2 (along x, y and z direction)
c. Choose the minimum between w1 and w2 to be our map resolution and discretize the world

After this, we map the start and target coordinate to corresponding grid coordinate, and now we can directly search through the discretized grid, which significantly reduce the search space and thus the searching is more efficient.

Collision Checking

After discretization, for a certain grid (i,j), we can check if the physical coordinate of the center of this grid is on the surface or within any obstacle, if so, this is considered an invalid grid and will never be expanded during A*, otherwise this grid is in the free space.

Since our map resolution is smaller than or equal to the minimum width among all obstacles, it will never happen that two adjacent grid that cross over an obstacle are both considered free, in other worlds, our optimal path will never cross over any obstacle.

Algorithm Detail

This is the logic flow of RTAA* algorithm:

1: $s \leftarrow C_s$, $\varepsilon \geq 1$, expand_size = n
2: while $C_T \notin$ CLOSED do
3: OPEN $\leftarrow$ {s}, CLOSED $\leftarrow$ {}, $g_s = 0$, $g_i = \infty$ for all $i \in C_{free} \setminus \{s\}$
4: for t = 1 : expand_size do
5: remove i with smallest $f_i := g_i + \varepsilon h_i$ from OPEN
6: insert i into CLOSED
7: for $j \in$ Children(i) and $j \notin$ CLOSED do
8: if $g_j > (g_i + C_{ij})$ then
9: $g_j \leftarrow (g_i + C_{ij})$
10: parent(j) $\leftarrow$ i
11: insert j into OPEN
12: remove i with smallest $f_{i^*} := g_i + \varepsilon h_i$ from OPEN
13: move s to i along the found path
14: update heuristic of all nodes i in CLOSED list by $h_i = f_{j^*} - g_i$

- The children of a grid node i is defined to be the adjacent 26 grid nodes of i.
- By choosing the appropriate expand_size, we can limit the planning time of every time step to be less than 2 seconds, while not compromising the optimality of A* algorithm too much.
- The RTAA* algorithm basically is a series of A* algorithm which makes use of the updated heuristic information obtained from previous iterations, at each time step the expand size of A* is strictly limited to catering the planning time constraints, after every A*, the node is moved to the current most promising location.

Time Complexity

Since we have to maintain the OPEN list as a priority queue, the time complexity of RTAA* algorithm is bounded by O(nlogn)

Performance Evaluation

The performance of this RTAA* varies with the following parameters:

1. Heuristic weight factor ε
2. Expand size n
3. Map resolution r

These are discussed in detail in the Results part.

2. N-Directional Rapidly-Exploring Random Tree Algorithm

Map Representation

Unlike the RTAA* algorithm, for our NRT algorithm, we would like to directly sample from the continuous free space to make our node tree as sparse as possible, therefore, we directly use the physical coordinate representations of our map.

Collision Checking

Without map discretization, there is probability that:

1. Our sampled point is on the surface or inside the obstacle;
2. The path between two sampled points collide with some obstacle.

And the following are the ways we deal with these two problems respectively:

1. Check the sampled point, if it is on the surface or inside any obstacle, resample until a valid point is obtained;
2. Discretize the line segment between two nodes with a small resolution (0.1), check if all the points on this line segments are valid points. If so, this path is valid, otherwise it is not a valid path (go through obstacles).

Algorithm Detail

This is the logic flow of NRT algorithm:

```
1: tree_list ← {start tree, end tree, n extra trees}
2: while the start tree and end tree are not connected do
3:         sample a point i in free space
4:         for tree j in tree_list do
5:                 among all nodes in tree j, find the closest node m to i
6:                 if path(m,i) is valid
7:                         connect m to i
8:                 else
9:                         extend m to the furthest valid point p along direction to i
10:        merge all trees that i directly connected to
11:        if rewiring is True
12:                for all node j within distance r to i
13:                        if path(i,j) is valid
14:                                connect i, j
15: run dijkstra to find the shortest path from start to target
```

- The minimum number of n extra trees is 0, which means only the start and end tree to expand simultaneously, i.e. bidirectional RRT.
- The cost between two tree nodes are maintained by adjacency matrix.
- Normally, the NRT algorithm can construct the tree and find a valid path within 2 seconds, however, for the cases that the environment is very complicated (e.g. maze), we need more trees expanding simultaneously, which significantly increase

the computational time. Therefore, we add a timer for the NRT algorithm, which will let the robot stay where it was if the optimal path has not been found within 2 seconds.

- Theoretically, the path found by RRT based algorithms are very sub-optimal due to the random sampling nature, therefore, we add a rewire step to reconnect the new node to all nodes that are within distance r to itself, i.e. to partially correct our trees. After rewiring the optimal path length is more acceptable.

Time Complexity

Suppose the total number of nodes we have at the end of tree construction step is n, the time complexity is bounded by $O(n^2)$. (since we have to consider the relationship between new node and all existing nodes)

Performance Evaluation

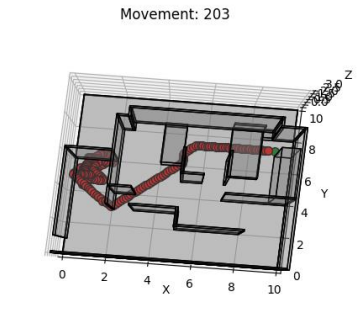
The performance of this RTAA* varies with the following parameters:

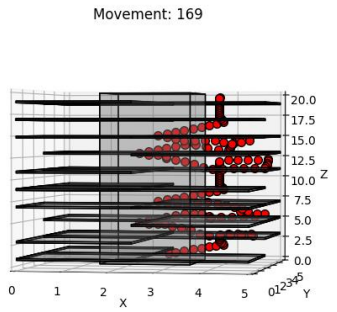
1. The complexity of map
2. Number of trees to expand simultaneously
3. Whether do the rewiring step

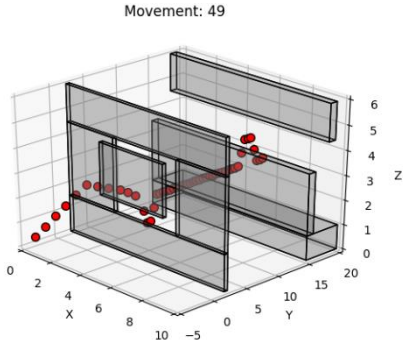
These are discussed in detail in the Results part.

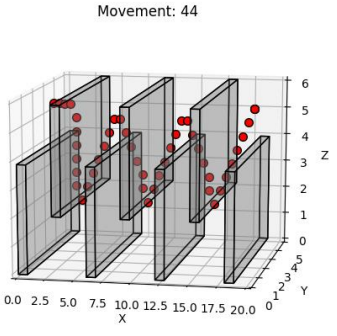
• Results

Sample Path obtained using RTAA*

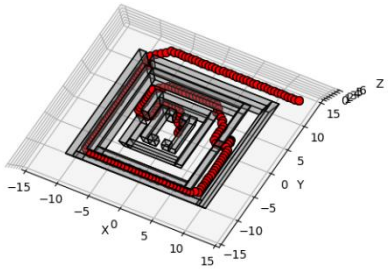




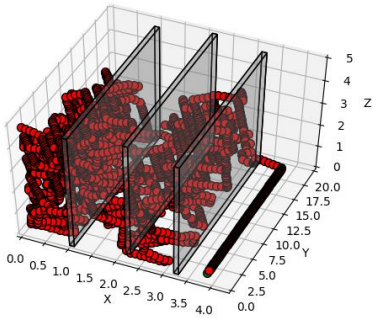




Movement: 202

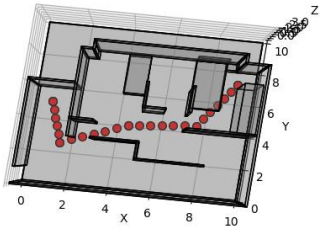


Movement: 3442

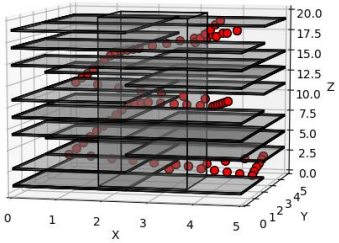


Sample Path obtained using NRT

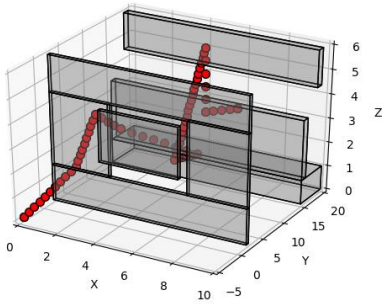
Movement: 29



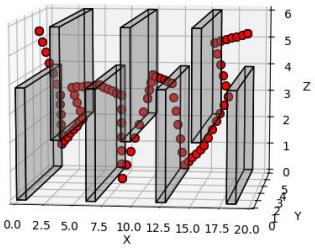
Movement: 96



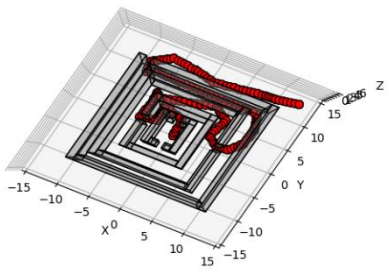
Movement: 76



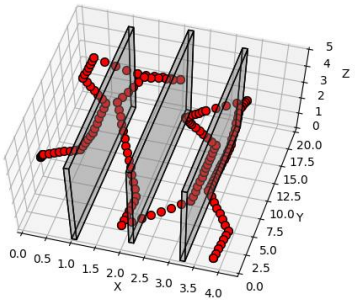
Movement: 97



Movement: 205



Movement: 176



RTAA* VS NRT

Video demo of RTAA* and NRT algorithm (also included in the zip file):

https://drive.google.com/file/d/1Qwghn3Q0DNI7jhY_ScJLxSVfH3Zbm9i_/view?usp=sharing

(left is NRT, right is RTAA*)

Parameters used in this demonstration:

NRT	n = 100	rewiring = False	r = 3
RTAA*	step_size = 0.5	expand_size = 500	epsilon = 5

Here are some data from the above demo:

	NRT			RTAA*		
	Distance	Nodes	Time	Distance	Nodes	Time
Maze	188.30	371	53	135.23	243	32
Flappy bird	42.86	186	24	29.49	45	5
Monza	85.46	215	25	458.69	3442	540
Window	62.65	66	8	28.69	49	6
Tower	85.94	239	33	57.47	233	42
Room	18.97	74	10	30.78	233	41

(UNIT: Distance – m Time – s)

Compare the performance of these two algorithms on different tasks, we can conclude that:

1. For cases that are simple or highly symmetric (e.g. Maze, Flappy bird, Window, Tower), RTAA* tends to spend less time to find the optimal path than NRT does, and the path get by RTAA* is shorter than that obtained by NRT;
2. However, for some difficult or highly unsymmetric cases (e.g. Monza, Room, Tower), RTAA* tends to spend more time than NRT, and the number of nodes visited and the distance traveled using RTAA* is larger than that of using NRT.
3. One noticeable example is Monza, that RTAA* spend considerable time planning and the distance traveled in unreasonably longer. This is because the obstacles in this case has very thin width, making our map discretization to be very inefficient, since we have way more grids than other cases. Once the searching becomes exhaustive, our planning becomes inefficient. What's worse, this case is very unsymmetric, resulting in that our algorithm has to search through every possible state to find the possible solution.
4. Therefore, these two algorithms have their own pros and cons. As summarized by the following table:

RTAA*	NRT
Works better for symmetric cases	Works better for unsymmetric cases
Path is usually quite optimal	Path is always highly sub-optimal

Planning for each step is quick	Has to construct the whole tree before planning
More target oriented.	Searches through complicated spaces more quickly

Hyper-parameters comparison

RTAA* (tested on Maze)

Epsilon / Expand_size	1			5			10		
	D	N	T	D	N	T	D	N	T
100	1315	2252	182	105	179	15	146	248	22
500	334	591	70	135	243	22	93	164	15
1000	250	451	53	109	202	19	154	276	27

Epsilon – heuristic weight factor

Expand_size – number of nodes to expand for evert iteration of RTAA*

D – total distance traveled, unit: m

N – Number of nodes visited

T – total time used, unit: s

Analysis:

1. For complicated cases like Maze, it is better to use lager heuristic factor, i.e. it is better to trust more on heuristics, but this is not absolute, if the heuristic factor gets too large, the result may compromise.
2. Generally speaking, it is better to expand more nodes during one step of RTAA*, although this might increase the total planning time, the probability of finding target node within limited time is also increased.

NRT (tested on Tower)

Rewiring / N	R = 0 (No rewiring)			R = 2			R = 4		
	D	N	T	D	N	T	D	N	T
10	60	218	22	36	122	13	36	100	11
50	91	321	27	35	122	13	33	97	23
100	74	242	21	49	165	16	34	92	19

N – Number of extra trees to expand simultaneously

R – Rewiring distance

D – total distance traveled, unit: m

N – Number of nodes visited

T – total time used, unit: s

Analysis:

1. Clearly, with increasing rewiring distance, our path become more optimal. However, more rewiring distance may also increase the planning time significantly. Since we have to consider more nodes to rewire, and the for the extreme case, we have to consider all other nodes, and the time complexity becomes $O(n^2)$
2. With more nodes to expand simultaneously, the possibility that a path is found increases but the computational time also increases, these two factors work together to determine the total planning time. Therefore, we should choose our N wisely. For simple cases like flappy bird, we may have less trees to expand, for complicated case like maze, we would like more trees to expand simultaneously to increase the possibility of construction of a valid path within constraint time.