Project 2: Motion Planning

Chen-Yeh Lin
Department of Electrical and Computer Engineering
University of California, San Diego
PID: A69036794

***Abstract*—This project addresses the problem of three-dimensional motion planning for autonomous agents navigating environments defined by a rectangular outer boundary and a collection of axis-aligned obstacle blocks. We model the task as a deterministic shortest-path problem in continuous space, implement a robust line-segment–AABB collision checker, and develop two complementary planners: a heuristic search–based planner built on a weighted A* framework and a sampling–based planner using Rapidly Exploring Random Trees (RRT). We evaluated both methods in seven distinct map environments, comparing path quality, computational efficiency, and node expansions. Our results show that the search–based approach produces near–optimal trajectories in structured settings, while the sampling–based method achieves faster convergence in highly cluttered environments.**

I. Introduction

Motion planning is a fundamental challenge in robotics, requiring the computation of collision-free trajectories for agents operating in continuous, obstacle-filled spaces. Real-world applications, such as warehouse automation, aerial drone navigation, and search and rescue, demand fast and reliable planners that can cope with complex geometries and tight resource constraints. Traditional grid-based search techniques offer optimality guarantees but suffer from scalability issues, whereas sampling-based methods scale more gracefully at the expense of path quality.

In this project, we tackle 3D motion planning by first implementing a line-segment versus axis-aligned bounding-box (AABB) collision checker to ensure safe path validation. Building on this foundation, we develop a search-based planner using a weighted A* algorithm enhanced with heuristic weighting and efficient neighbor pruning. In parallel, we implement a RRT planner that leverages random sampling for rapid exploration. The robot is tested in seven different maps to comprehensively assess each planner's performance. We validate our approaches by analyzing trajectory optimality, planning time, and computational resources across these seven map scenarios.

II. Problem Statement

We consider a 3-D motion planning problem in a known static environment. Let the workspace be a rectangular boundary

$$\mathcal{B} = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}] \subset \mathbb{R}^3$$

and let there be m axis-aligned rectangular obstacle blocks

$$\mathcal{O}_i \;=\; [x_i^{\min}, x_i^{\max}] \times [y_i^{\min}, y_i^{\max}] \times [z_i^{\min}, z_i^{\max}], \quad i = 1, \ldots, m,$$

so that the free space is

$$\mathcal{X}_{\text{free}} = \big\{\, x \in \mathcal{B} \;\big|\; x \notin \bigcup_{i=1}^{m} \mathcal{O}_i \big\}. \tag{1}$$

Two given points start and goal

$$x_s,\; x_g \;\in\; \mathcal{X}_{\text{free}}. \tag{2}$$

We seek a continuous, collision-free path

$$f : [0, T] \;\longrightarrow\; \mathcal{X}_{\text{free}}, \qquad f(0) = x_s, \quad f(T) = x_g.$$

that minimizes total path cost under the cost function C. In its continuous form:

$$f^* \;=\; \underset{\substack{f:[0,T]\to\mathcal{X}_{\text{free}} \\ f(0)=x_s,\, f(T)=x_g}}{\arg\min} \int_0^T C\big(f(t), \dot{f}(t)\big)\, \mathrm{d}t. \tag{3}$$

Typically, we choose

$$C\big(f(t), \dot{f}(t)\big) = \big\|\dot{f}(t)\big\|,$$

so that

$$\int_0^T \big\|\dot{f}(t)\big\|\, \mathrm{d}t$$

equals the Euclidean arc-length of the path. Equivalently, in the discrete setting we represent a path by a sequence of waypoints $\{\, i_t\}_{t=0}^{T} \subset \mathcal{X}_{\text{free}}$ with

$$i_0 = x_s, \quad i_T = x_g,$$

and solve

$$i^* \;=\; \underset{\substack{\{i_t\}_{t=0}^{T}\subset\mathcal{X}_{\text{free}} \\ i_0=x_s,\, i_T=x_g}}{\arg\min} \sum_{t=0}^{T-1} C\big(i_t,\, i_{t+1}\big). \tag{4}$$

where

$$C\big(i_t,\, i_{t+1}\big) = \big\|\, i_{t+1} - i_t \,\big\|$$

is the Euclidean distance between successive waypoints.

III. Technical Approach

In this section, the algorithm that solves the problem will be expressed. The algorithm could be stated into three parts, which are collision check, search-based planning algorithm, and sampling-based planning algorithm.

A. Collision Checking Algorithm

To test whether a continuous line segment

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,(\mathbf{p}_1 - \mathbf{p}_0), \quad t \in [0,1]$$

intersects an axis-aligned bounding box (AABB) defined by

$$[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}],$$

we implement the classic *Slab* method:

1) **Parametric representation.**
Let $\mathbf{v} = \mathbf{p}_1 - \mathbf{p}_0$. Along each axis $d \in \{x, y, z\}$, compute

$$t_{d,\text{enter}} = \frac{d_{\min} - p_{0,d}}{v_d}, \quad t_{d,\text{exit}} = \frac{d_{\max} - p_{0,d}}{v_d}.$$

If $v_d < 0$, swap so that $t_{d,\text{enter}} \le t_{d,\text{exit}}$.

2) **Handle parallelism.**
If $v_d = 0$, check

$$p_{0,d} \in [d_{\min}, d_{\max}];$$

otherwise return false.

3) **Compute global entry/exit times.**

$$t_{\min} = \max_d t_{d,\text{enter}}, \quad t_{\max} = \min_d t_{d,\text{exit}}.$$

4) **Intersection test.** Return true iff

$$t_{\min} \le t_{\max} \quad \text{and} \quad [t_{\min}, t_{\max}] \cap [0,1] \neq \emptyset,$$

else false.

B. Search-Based Planning Algorithm

We implement a *Weighted A** planner on the discretized 3-D grid. Let $\mathcal{V}$ be the set of grid cells, s the start cell, and τ the goal cell. We inflate the heuristic by a factor $\epsilon \ge 1$ to trade optimality for speed.

Heuristic. We use the Euclidean distance in $\mathbb{R}^3$,

$$h(i) = \left\|\mathbf{x}_i - \mathbf{x}_\tau\right\|_2,$$

which is admissible and consistent on our grid.

Algorithmic Steps.

1) *Initialization.* $\forall i \in \mathcal{V}: \; g(i) \leftarrow +\infty$, parent$(i) \leftarrow$ NULL. Set $g(s) = 0$, compute $f(s) = g(s) + \epsilon\, h(s)$, insert s into OPEN (a min-heap over f).

2) *Main Loop.* While OPEN is non-empty:
   a) Pop the node $i = \arg\min_{j \in \text{OPEN}} \big(g(j) + \epsilon\, h(j)\big)$.
   b) If $i = \tau$, stop and *reconstruct* the path via parent pointers.
   c) Move i to CLOSED.
   d) For each neighbor j of i in the 26-connected grid:

$$g_{\text{tent}} = g(i) + \|\,\mathbf{x}_i - \mathbf{x}_j\|_2.$$

If $g_{\text{tent}} < g(j)$ *and* the segment (i, j) is collision-free (slab AABB check), then

$$g(j) \leftarrow g_{\text{tent}}, \quad \text{parent}(j) \leftarrow i, \quad f(j) \leftarrow g(j) + \epsilon\, h(j),$$

and insert or decrease-key j in OPEN.

Algorithm 1 Weighted A* Algorithm

OPEN $\leftarrow \{s\}$, CLOSED $\leftarrow \emptyset$
for all $i \in \mathcal{V} \setminus \{s\}$ **do**
  $g_i \leftarrow \infty$
end for
$g_s \leftarrow 0$
while $\tau \notin$ CLOSED **do**
  Remove node i with smallest

$$f_i \;=\; g_i + \epsilon\, h_i$$

  from OPEN
  OPEN $\leftarrow$ OPEN $\setminus \{i\}$
  CLOSED $\leftarrow$ CLOSED $\cup \{i\}$
  for all $j \in \text{Children}(i)$ **and** $j \notin$ CLOSED **do**
    if $g_j > g_i + c_{ij}$ **then**
      $g_j \leftarrow g_i + c_{ij}$
      Parent$(j) \leftarrow i$
      if $j \in$ OPEN **then**
        Update priority of j (key $= g_j + \epsilon\, h_j$)
      else
        OPEN $\leftarrow$ OPEN $\cup \{j\}$
      end if
    end if
  end for
end while=0

Implementation Properties:

- *Suboptimality bound:* returns a path of cost $\le \epsilon \cdot C^*$, where C^* is optimal.
- *Completeness:* guaranteed to find a solution if one exists (finite graph, $\epsilon < \infty$).
- *Time efficiency:* inflation $\epsilon > 1$ typically reduces the number of expansions by roughly ϵ, accelerating large-map searches.
- *Memory efficiency:* stores only $\mathcal{O}(N)$ states in OPEN/CLOSED, where N is the number of visited cells.

C. Sampling–Based Planning Algorithm

We implement the classic Rapidly-Exploring Random Tree (RRT) in continuous 3-D space. Let $\mathcal{T} = (V, E)$ be our search tree, initialized with $V = \{x_s\}$ and $E = \emptyset$.

Hyperparameters.

$$\text{max\_iterations} = 10^6, \quad \delta = 0.5, \quad p_{\text{goal}} = 0.05.$$

Algorithmic Steps.

1) *Sampling.* With probability p_{goal}, set $x_{\text{rand}} = x_\tau$; otherwise sample x_{rand} uniformly within the environment's bounding box.
2) *Nearest Neighbor.* Find
$$v_{\text{nearest}} = \arg\min_{v \in V} \|v - x_{\text{rand}}\|_2.$$
3) *Steer.* Compute
$$x_{\text{new}} = \begin{cases} x_{\text{rand}}, \|x_{\text{rand}} - v_{\text{nearest}}\|_2 \le \delta, \\ v_{\text{nearest}} + \delta \dfrac{x_{\text{rand}} - v_{\text{nearest}}}{\|x_{\text{rand}} - v_{\text{nearest}}\|_2}, \quad \text{otherwise.} \end{cases}$$
4) *Collision Checking.* If the segment $(v_{\text{nearest}}, x_{\text{new}})$ is collision-free (slab AABB test), add x_{new} to V and $(v_{\text{nearest}}, x_{\text{new}})$ to E.
5) *Goal Test.* If $\|x_{\text{new}} - x_\tau\|_2 \le \delta$ and the direct connection to x_τ is collision-free, insert x_τ and its edge, then *reconstruct* the path back to x_s.
6) *Termination.* Repeat until a path is found or the iteration count reaches max_iterations.

Algorithm 2 Rapidly-Exploring Random Tree (RRT)

Require: start configuration x_s, number of iterations n
Ensure: graph $G = (V, E)$ approximating the reachable space

$V \leftarrow \{x_s\}$; $E \leftarrow \emptyset$
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \text{SampleFree}()$
  $x_{\text{nearest}} \leftarrow \text{Nearest}((V, E), x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \text{Steer}_\epsilon(x_{\text{nearest}}, x_{\text{rand}})$
  if $\text{CollisionFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$
    $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$
  end if
end for
return $G = (V, E)$ =0

Implementation Properties:

- *Optimality:* No deterministic optimality guarantee—RRT finds a feasible path quickly but is typically suboptimal (use RRT* for asymptotic optimality).
- *Probabilistically complete*: as max_iterations $\to \infty$, the probability of finding a path (if one exists) approaches 1.
- *Time efficiency:* Brute-force nearest-neighbor search costs $O(|V|)$ each iteration (worst-case $O(n^2)$ for n = max_iterations), but can be reduced to $O(n \log n)$ with a k-d tree.
- *Memory efficiency:* Stores $O(n)$ nodes and edges, where $n \le$ max_iterations.

IV. Results

In this section, we compare the performance of our search-based planner (weighted A*) and sampling-based planner (RRT) on the seven test environments provided.

TABLE I: Path Length Comparison

Environment	A* ($\epsilon = 1$)	A* ($\epsilon = 5$)	RRT
Single Cube	8.12	9.20	10.52
Maze	85.73	92.86	111.95
Flappy Bird	30.74	32.00	45.41
Pillars	30.41	30.70	45.37
Window	28.21	29.92	34.34
Tower	34.87	37.82	41.80
Room	19.07	21.56	22.53

TABLE II: Planning Time Comparison (seconds)

Environment	A* ($\epsilon = 1$)	A* ($\epsilon = 5$)	RRT
Single Cube	0.00210	0.00209	0.00361
Maze	2.38972	2.33748	511.83538
Flappy Bird	0.29267	0.05735	1.33716
Pillars	0.85290	0.03930	1.31746
Window	1.03647	0.03092	1.33194
Tower	0.52314	0.09211	2.41289
Room	0.18256	0.09472	0.08273

TABLE III: Number of Nodes Considered

Environment	A* ($\epsilon = 1$)	A* ($\epsilon = 5$)	RRT
Single Cube	5	5	62
Maze	2240	2170	14757
Flappy Bird	411	71	1290
Pillars	535	22	1165
Window	930	26	1429
Tower	474	78	1442
Room	125	66	225

1) Discussion: As shown in Tables I–III, the two planner classes exhibit markedly different trade-offs in path quality, computational effort, and sensitivity to parameters.

1) *Quality of computed paths.* A$^*(\varepsilon = 1)$ yields optimal or near-optimal paths. Weighted A$^*(\varepsilon = 5)$ increases path length modestly ($\approx 5\%$–10% over $\varepsilon = 1$) but remains substantially shorter than RRT; for example, in the Maze environment path lengths

grow from 85.73 to 92.86 for $\varepsilon = 5$, versus 111.95 for RRT.

2) *Number of considered nodes.* Inflating the heuristic factor dramatically reduces expansions: in the Pillars environment, $\mathrm{A}^*(1)$ considers 535 nodes, whereas $\mathrm{A}^*(5)$ examines only 22. By contrast, RRT requires 1165 nodes.
3) *Effect of parameters and planning time.* Increasing ε trades bounded suboptimality (path cost $\leq \varepsilon\, C^*$) for significant speedups. RRT's performance depends heavily on sampling strategy—uniform sampling struggles in narrow passages, leading to excessive collision checks and long runtimes (e.g. 511s in Maze).
4) *Interesting observations.* In highly cluttered environments, deterministic search with an admissible heuristic vastly outperforms random sampling. In open spaces (e.g. Room), RRT can marginally beat weighted A^* in raw runtime (0.083s vs. 0.095s) but produces $\approx 20\%$–30% longer paths. Moreover, A^* guarantees consistency and a user-specified suboptimality bound, whereas RRT offers only probabilistic completeness.

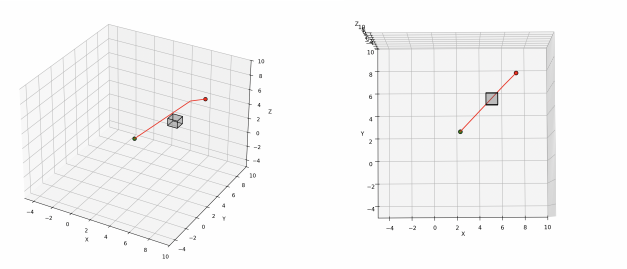

Fig. 1: Single Cube A*-1

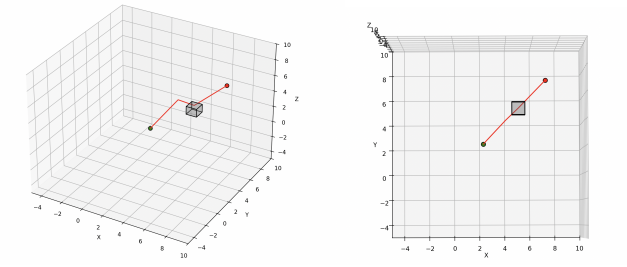

Fig. 2: Single Cube A*-5

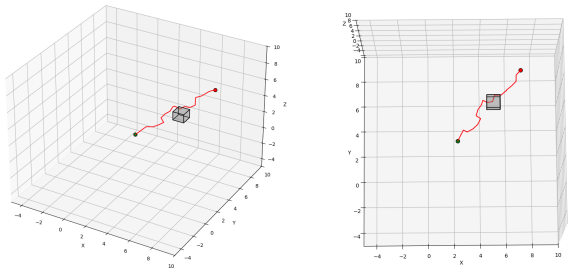

Fig. 3: Single Cube RRT

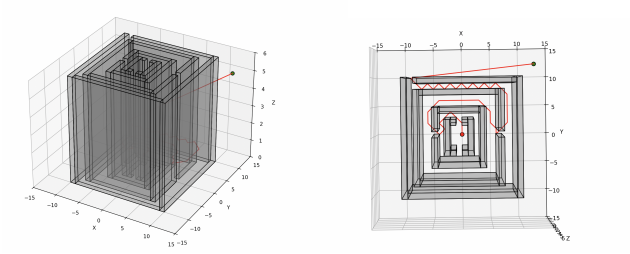

Fig. 4: Maze A*-1

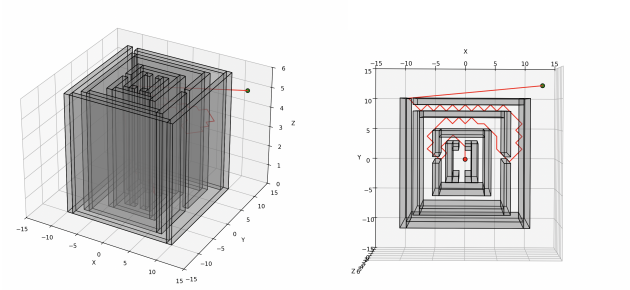

Fig. 5: Maze A*-5

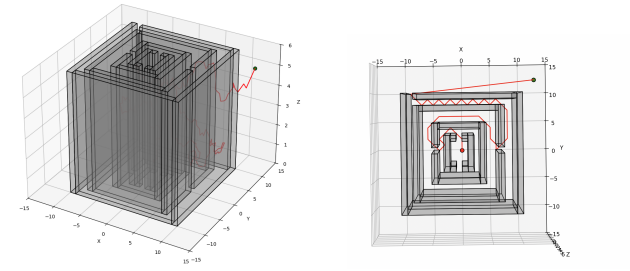

Fig. 6: Maze RRT

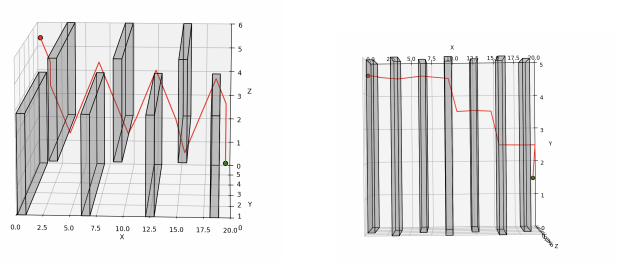

Fig. 7: Flappy Bird A*-1

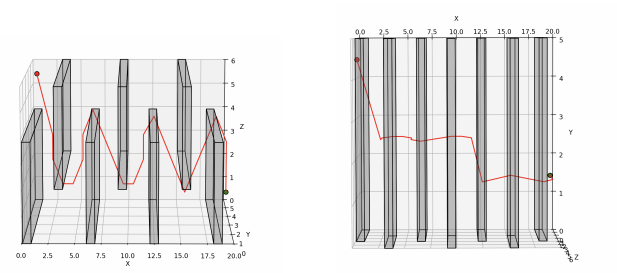

Fig. 8: Flappy Bird A*-5

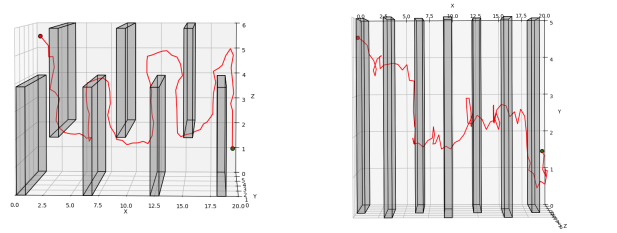

Fig. 9: Flappy Bird RRT

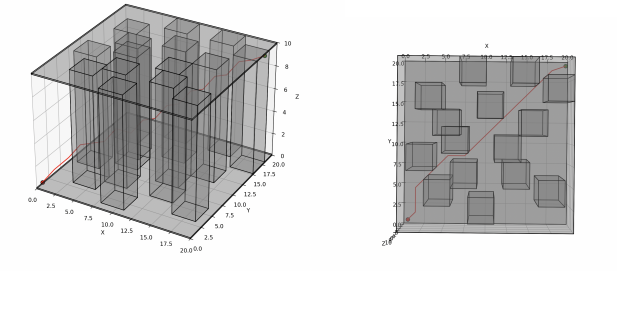

Fig. 10: Pillars A*-1

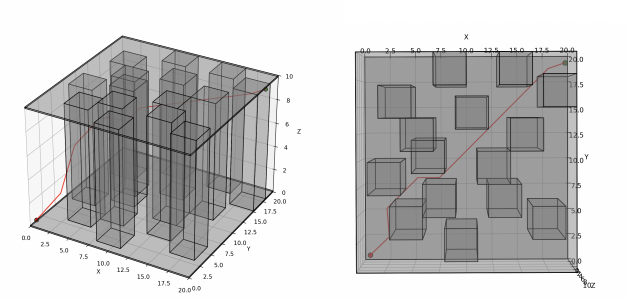

Fig. 11: Pillars A*-5

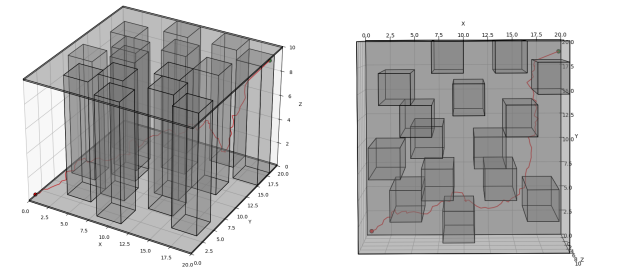

Fig. 12: Pillars RRT

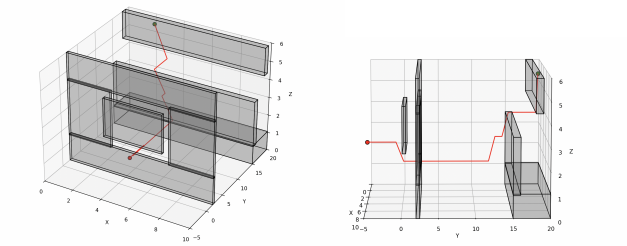

Fig. 13: Window A*-1

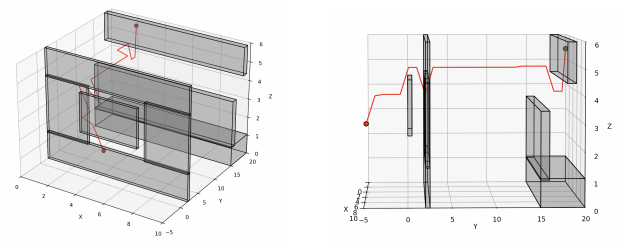

Fig. 14: Window A*-5

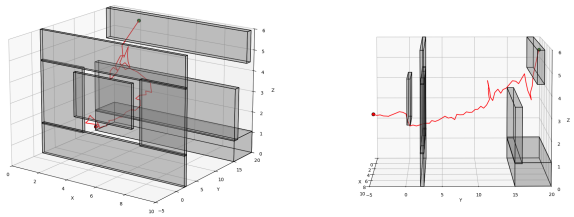

Fig. 15: Window RRT

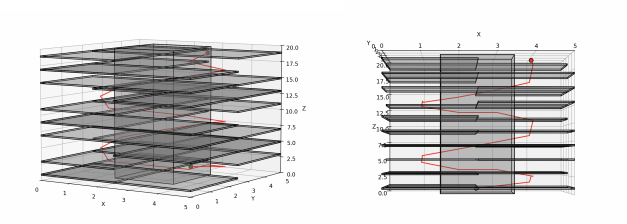

Fig. 16: Tower A*-1

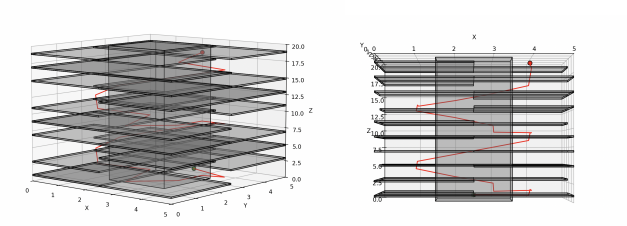

Fig. 17: Tower A*-5

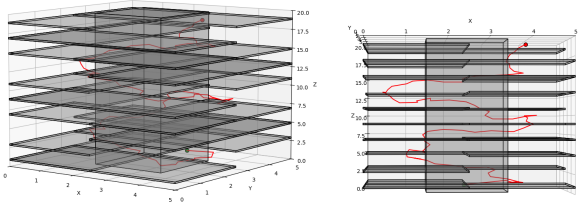

Fig. 18: Tower RRT

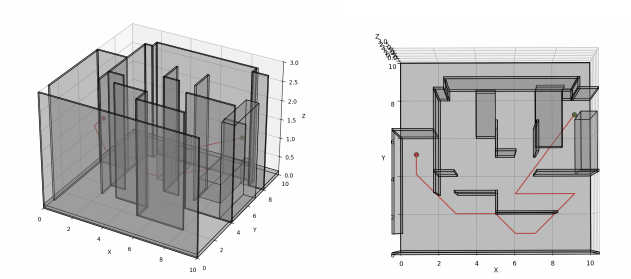

Fig. 19: Room A*-1

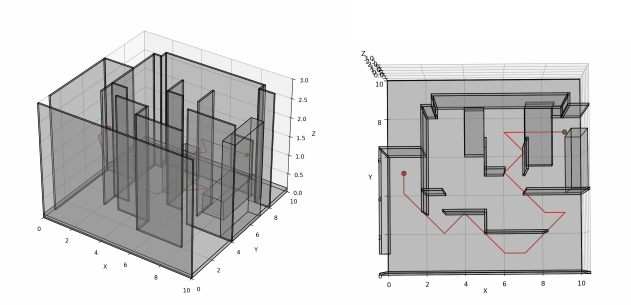

Fig. 20: Room A*-5

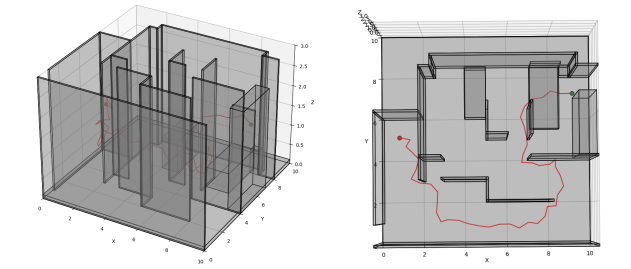

Fig. 21: Room RRT