# Dynamic Programming

Manoj Kumar Reddy Manchala
*Department of Mechanical and Aerospace Engineering*
*University of California San Diego*
La Jolla, U.S.A.
mmanchala@ucsd.edu

Nikolay Atanasov
*Department of Electrical and Computer Engineering*
*University of California San Diego*
La Jolla, U.S.A.
natanasov@eng.ucsd.edu

***Abstract*—This report presents an implementation of two prominent path planning algorithms, A* and Rapidly-exploring Random Tree (RRT*), for motion planning in a continuous 3D environment. The objective is to find an optimal collision-free path from a start point to a goal point within a given map containing obstacles. We begin by discretizing the continuous space into a grid for the A* algorithm, leveraging its heuristic search capabilities to efficiently find the shortest path. Subsequently, we explore the sampling-based approach of RRT* using a Python motion planning library, which incrementally builds a tree to discover feasible paths. Both algorithms are tested on various complex environments, and their performance is evaluated in terms of computational efficiency, path optimality, and collision avoidance.**

***Index Terms*—Search based Planning, Sampling based planning, A*, RRT, Bi-directional RRT**

## I. Introduction

Motion planning is a fundamental problem in robotics, aiming to find a collision-free path for a robot to navigate from a start to a goal position within a defined environment. This project focuses on comparing the performance of search-based and sampling-based motion planning algorithms in simulating three-dimensional environments. The project involved implementing a collision checking algorithm, a search-based planning algorithm (e.g., weighted A*, jump point search), and either implementing a sampling-based planning algorithm (e.g., RRT) or utilizing existing libraries like OMPL or the Python motion planning library (rrt-algorithms). The report will discuss the technical approaches, compare the performance of the implemented algorithms based on specified criteria, and present the results obtained from navigating the robot through the provided three-dimensional environments.

To address these challenges, this project explores two prominent classes of motion planning algorithms:

1. **Search-Based Planning:** These algorithms systematically explore the search space, often using heuristics to guide the search towards promising regions. Examples include A*, weighted A*, and Jump Point Search. 2. **Sampling-Based Planning:** These algorithms construct a roadmap of the environment by randomly sampling configurations and connecting them if collision-free. Popular examples include Rapidly-exploring Random Trees (RRT), RRT*, and Probabilistic Roadmaps (PRM).

## II. Problem Formulation

Given a 3D environment with a defined boundary and several obstacles, the objective is to find the shortest collision-free path from a start point to a goal point. The path planning must account for the obstacles and ensure that the path does not intersect with any of them.

The 3-D motion planning problem can be formally defined as a deterministic shortest path problem in continuous space. Given:

- A 3-D environment $\mathcal{W} \subset \mathbb{R}^3$ (workspace)
- A set of obstacles $\mathcal{O} \subset \mathcal{W}$
- A start configuration $x_s \in \mathcal{W} \setminus \mathcal{O}$ (collision-free)
- A goal configuration $x_g \in \mathcal{W} \setminus \mathcal{O}$ (collision-free)

The objective is to find a continuous path $\pi : [0, 1] \to \mathcal{W} \setminus \mathcal{O}$ such that:

- $\pi(0) = x_s$ (starts at the start configuration)
- $\pi(1) = x_g$ (ends at the goal configuration)
- $\pi(t) \notin \mathcal{O}$ for all $t \in [0, 1]$ (collision-free)
- The path length $l(\pi) = \int_0^1 ||\dot{\pi}(t)||dt$ is minimized, where $\dot{\pi}(t)$ is the time derivative of the path

In this project, we make the following assumptions:

- The environment $\mathcal{W}$ is a rectangular prism
- The obstacles $\mathcal{O}$ are axis-aligned rectangular prisms (AABBs)
- The robot is modeled as a point in $\mathbb{R}^3$

These assumptions simplify the problem while still capturing the essential challenges of 3-D motion planning. The choice of AABBs for obstacles is common in practice due to their computational efficiency for collision checking.

### A. Mathematical Formulation

*a) Environment Representation:* The 3-D environment is defined by its boundaries and obstacles. The boundaries of the environment are given by:

$$\text{Boundary} = [xmin, ymin, zmin, xmax, ymax, zmax]$$

where $[xmin, ymin, zmin]$ and $[xmax, ymax, zmax]$ represent the minimum and maximum coordinates of the environment in the x, y, and z dimensions, respectively.

The obstacles within the environment are represented as axis-aligned bounding boxes (AABBs), defined by:

$$\text{Obstacle}_i = [xmin_i, ymin_i, zmin_i, xmax_i, ymax_i, zmax_i]$$

where $[xmin_i, ymin_i, zmin_i]$ and $[xmax_i, ymax_i, zmax_i]$ represent the minimum and maximum coordinates of the $i$-th obstacle in the x, y, and z dimensions, respectively.

*b) Start and Goal Positions:* The start position is defined as:

$$\text{Start} = [x_s, y_s, z_s]$$

The goal position is defined as:

$$\text{Goal} = [x_g, y_g, z_g]$$

*c) Path Representation:* A path $\mathbf{P}$ from the start to the goal is a continuous curve in the 3-D space:

$$\mathbf{P} : [0, 1] \to \mathbb{R}^3$$

such that $\mathbf{P}(0) = \text{Start}$ and $\mathbf{P}(1) = \text{Goal}$.

*d) Collision-Free Constraint:* The path $\mathbf{P}$ must be collision-free, i.e., it must not intersect with any obstacles. This can be expressed as:

$$\forall t \in [0, 1],\ \mathbf{P}(t) \notin \text{Obstacle}_i \quad \forall i$$

*e) Cost Function:* The cost function $J(\mathbf{P})$ to be minimized is the length of the path $\mathbf{P}$, given by:

$$J(\mathbf{P}) = \int_0^1 \left\| \frac{d\mathbf{P}(t)}{dt} \right\| dt$$

where $\|\cdot\|$ denotes the Euclidean norm.

*f) Optimization Problem:* The motion planning problem can be formulated as the following optimization problem:

$$\min_{\mathbf{P}} J(\mathbf{P})$$

subject to:

$$\mathbf{P}(0) = \text{Start}$$

$$\mathbf{P}(1) = \text{Goal}$$

$$\forall t \in [0, 1],\ \mathbf{P}(t) \notin \text{Obstacle}_i \quad \forall i$$

### *B. Discretization and Graph Representation*

To apply graph-based algorithms such as A* in a continuous space, the environment must be discretized into a grid. Each cell in the grid represents a node in the graph, and the edges represent possible transitions between nodes.

*a) Grid Representation:* The continuous space is discretized into a grid with a resolution $r$. Each cell in the grid is represented by:

$$\text{Cell}(i, j, k) = [x_{\min} + i \cdot r, y_{\min} + j \cdot r, z_{\min} + k \cdot r]$$

where $(i, j, k)$ are the indices of the cell.

*b) Cost Function in Discrete Space:* The cost function in the discretized space is the sum of the Euclidean distances between consecutive nodes along the path. For a path $\mathbf{P}$ represented as a sequence of nodes $\{n_0, n_1, \ldots, n_m\}$:

$$J(\mathbf{P}) = \sum_{i=0}^{m-1} \|n_{i+1} - n_i\|$$

where $\|n_{i+1} - n_i\|$ is the Euclidean distance between nodes $n_i$ and $n_{i+1}$.

*c) A* Algorithm:* The A* algorithm is used to find the shortest path in the discretized space. It employs a heuristic function $h(n)$ to estimate the cost from the current node $n$ to the goal node. The total cost $f(n)$ is given by:

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

where $g(n)$ is the cost from the start node to the current node $n$, and $h(n)$ is the heuristic estimated cost from the current node $n$ to the goal node.

*1) Heuristic Function:* The heuristic function $h(n)$ used in our implementation is the Euclidean distance:

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

### *C. Summary*

The deterministic shortest path problem in a continuous 3-D space involves finding a collision-free path from a start point to a goal point, minimizing the path length. The problem is mathematically formulated as an optimization problem with constraints, and the environment is discretized to apply graph-based algorithms like A* for practical implementation.

## III. Technical Approach

This project explores two distinct motion planning paradigms: a search-based A* algorithm that we developed and a sampling-based RRT algorithm using the 'rrt-algorithms' library.

The environment is represented as a 3D space with boundaries defined by:

$$\text{Boundary} = [xmin, ymin, zmin, xmax, ymax, zmax]$$

The obstacles are represented as axis-aligned bounding boxes (AABBs) defined by:

$$\text{Obstacle}_i = [xmin_i, ymin_i, zmin_i, xmax_i, ymax_i, zmax_i]$$

The start and goal positions are given by:

$$\text{Start} = [x_s, y_s, z_s]$$

$$\text{Goal} = [x_g, y_g, z_g]$$

### *A. Search-Based Planning: A**

A* is a widely used graph search algorithm that leverages heuristic information to efficiently find optimal or near-optimal paths. Our implementation tailors A* for 3-D environments with axis-aligned bounding box (AABB) obstacles.

*1) Discretization:* We discretize the continuous 3-D space $\mathcal{W}$ into a grid $\mathcal{G}$ with a user-defined resolution $r$. This transforms the problem into a search over discrete cells. The discretization function is defined as:

$$c = \text{discretize}(x) = \left\lfloor \frac{x - w_{\min}}{r} \right\rfloor$$

where $x \in \mathcal{W}$ is a continuous coordinate, $w_{\min}$ is the lower corner of the workspace, $r$ is the resolution and $c$ is the corresponding discrete cell coordinate.

*2) Dediscretization:* In order to perform collision checking on the fly while extracting the valid neighbours, we need to convert these cell coordinates back to the workspace coordinates, which is performed as follows:

$$x = (c + 0.5) * r + m$$

where $x \in \mathcal{W}$ is a continuous coordinate, $w_{\text{min}}$ is the lower corner of the workspace, $r$ is the resolution and $c$ is the corresponding discrete cell coordinate.

*3) Heuristic Function:* We employ the Euclidean distance heuristic $h$ to estimate the remaining cost from a node $n$ to the goal $x_g$. This guides the search towards promising directions.

$$h(c) = ||\text{dediscretize}(c) - x_g||_2$$

where dediscretize$(n)$ converts the discrete cell coordinate back to continuous space.

*4) Priority Queue:* We maintain an open list of nodes using a priority queue, prioritizing nodes with lower estimated total cost $f(n) = g(n) + \epsilon h(n)$, where $g(n)$ is the cost-to-come from the start node to node $n$, and $\epsilon$ is a weighting factor.

### *B. Collision Checking Function*

The collision checking function determines if a line segment intersects with any of the obstacles. This is essential to ensure that the planned path is collision-free.

*1) Intersection with AABB:* Given an Axis-Aligned Bounding Box (AABB) defined by its minimum corner $A_{\text{min}} = (x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$ and maximum corner $A_{\text{max}} = (x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$, and a line segment defined by its start point $S = (x_s, y_s, z_s)$ and end point $E = (x_e, y_e, z_e)$, we want to determine if the line segment intersects with the AABB. The function to check if a line segment intersects with an AABB is defined as follows:

The intersection function calculates $t$ values for the intersection with the AABB planes using the following equations:

$$t_{\text{min}} = \frac{A_{\text{min}} - S}{E - S + \epsilon}$$
$$t_{\text{max}} = \frac{A_{\text{max}} - S}{E - S + \epsilon}$$

where $\epsilon = 1e - 6$ is a small value added to avoid division by zero.

To ensure $t_{\text{min}}$ is always the smaller value and $t_{\text{max}}$ is the larger value for each axis, we compute:

$$t_1 = \min(t_{\text{min}}, t_{\text{max}})$$
$$t_2 = \max(t_{\text{min}}, t_{\text{max}})$$

Next, we find the largest $t_{\text{min}}$ and smallest $t_{\text{max}}$ across all three axes:

$$t_{\text{enter}} = \max(t_1)$$
$$t_{\text{exit}} = \min(t_2)$$

Finally, we check if there is an intersection within the segment using the conditions:

$$t_{\text{enter}} \leq t_{\text{exit}} \quad \text{and} \quad t_{\text{exit}} \geq 0 \quad \text{and} \quad t_{\text{enter}} \leq 1$$

If these conditions are met, then there is an intersection, otherwise, there is no intersection.

### *C. Algorithm: A**

The A* algorithm is used for finding the shortest path in a graph. It employs a heuristic to guide the search towards the goal. The key components of the A* algorithm include the cost function and the heuristic function.

*1) Cost Function:* The cost function, $f(n)$, for node $n$ is defined as:

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

where:

- $g(n)$ is the cost from the start node to node $n$.
- $h(n)$ is the heuristic estimated cost from node $n$ to the goal node.

*2) A* Algorithm:* The A* algorithm can be summarized as follows:

**Algorithm 1** A* Search Algorithm

**Input:** Start node Start, Goal node Goal
**Output:** Path from Start to Goal

Initialize the open list with the start node
Initialize the closed list as empty

**while** open list is not empty **do**
  Remove the node with the lowest $f(n)$ from the open list
  **if** the removed node is the goal **then**
    **return** the path from start to goal
  **end if**
  **for** each neighbor of the current node **do**
    **if** neighbor is not traversable or in the closed list **then**
      **continue**
    **end if**
    Calculate $g(n)$ for the neighbor
    **if** neighbor is not in the open list or the new $g(n)$ is lower **then**
      Set the parent of the neighbor to the current node
      Calculate $f(n)$ for the neighbor
      **if** neighbor is not in the open list **then**
        Add the neighbor to the open list
      **end if**
    **end if**
  **end for**
  Add the current node to the closed list
**end while**

**return** failure (no path found) =0

### D. Rapidly-exploring Random Tree (RRT)

RRT is a sampling-based algorithm used for path planning in high-dimensional spaces. The key idea is to incrementally build a tree by randomly sampling the space and connecting the samples to the tree.

We employed the RRT algorithm from the 'rrt-algorithms' library for comparison. This algorithm efficiently explores high-dimensional spaces by incrementally building a tree of random samples connected if collision-free.

*a) RRT Algorithm:* The RRT algorithm can be summarized as follows:

**Algorithm 2** RRT Algorithm

**Input:** Start node, Goal node
**Output:** Path from start to goal

Initialize the tree with the start node
**for** each iteration **do**
  Sample a random node $x_{rand}$
  Find the nearest node $x_{near}$ in the tree to $x_{rand}$
  Create a new node $x_{new}$ by moving from $x_{near}$ towards $x_{rand}$
  **if** $x_{new}$ is in the free space **then**
    Add $x_{new}$ to the tree
    **if** $x_{new}$ is close to the goal **then**
      **return** the path from start to goal
    **end if**
  **end if**
**end for**

**return** failure (no path found) =0

### E. Properties

- (Sub)optimality: With $\epsilon = 1$, our A* implementation guarantees the optimal path in the discretized space. For $\epsilon > 1$ (weighted A*), it provides faster but potentially suboptimal solutions.
- Completeness: A* is complete; it will find a solution if one exists in the discretized space.
- Memory Efficiency: Memory usage grows with the size of the explored space, which can be a limitation for large environments.
- Time Efficiency: Efficiency is influenced by the heuristic quality and the branching factor (number of neighbors). The Euclidean distance heuristic effectively guides the search.

## IV. Results

### A. *A* - epsilon = 1.2*

*1) Visualization : A* - single cube & $\epsilon$ = 0.5:* The following represent the path planned by A* for the single cube environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 8.

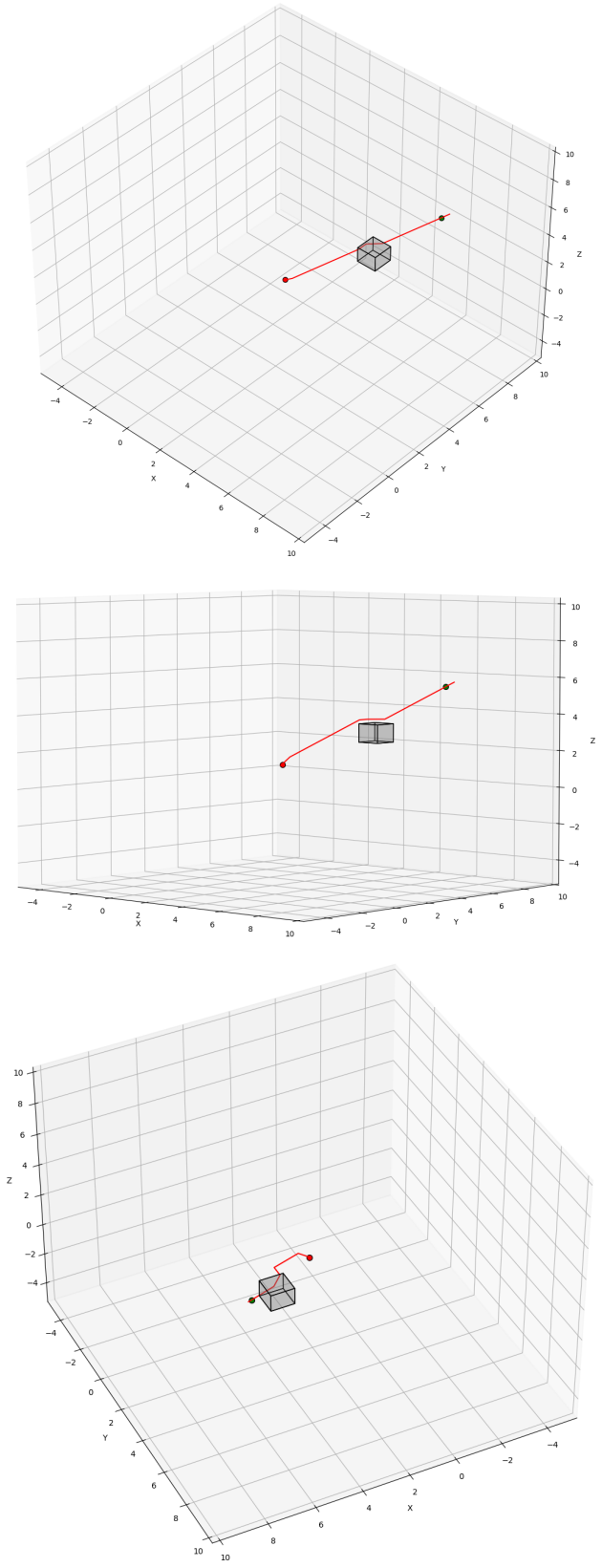


Fig. 1. Agent's path through the single cube environment

*2) **Visualization : A* - maze & $\epsilon$ = 0.5**:* The following represent the path planned by A* for the maze environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 77.

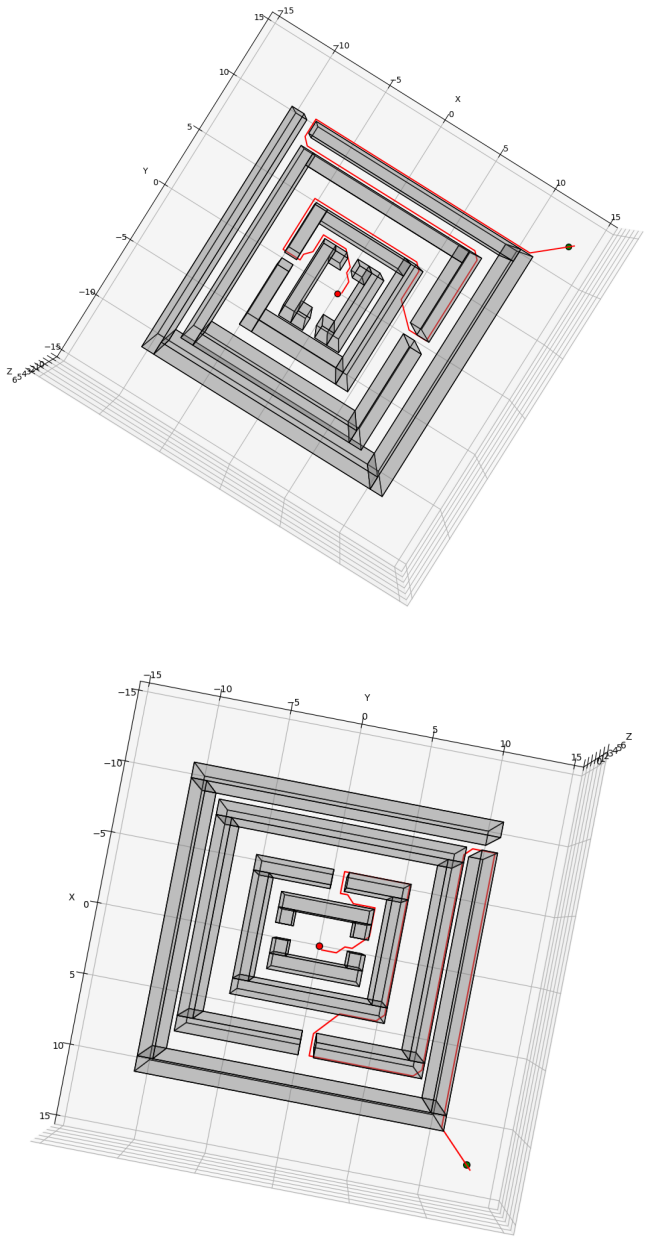

Fig. 2. Agent's path through the maze environment

*3) **Visualization : A* - flappy bird & $\epsilon$ = 0.5**:* The following represent the path planned by A* for the flappy bird environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 27.

*4) **Visualization : A* - monza & $\epsilon$ = 0.5**:* The following represent the path planned by A* for the monza environment from different angles. It can be seen that the agent is able to

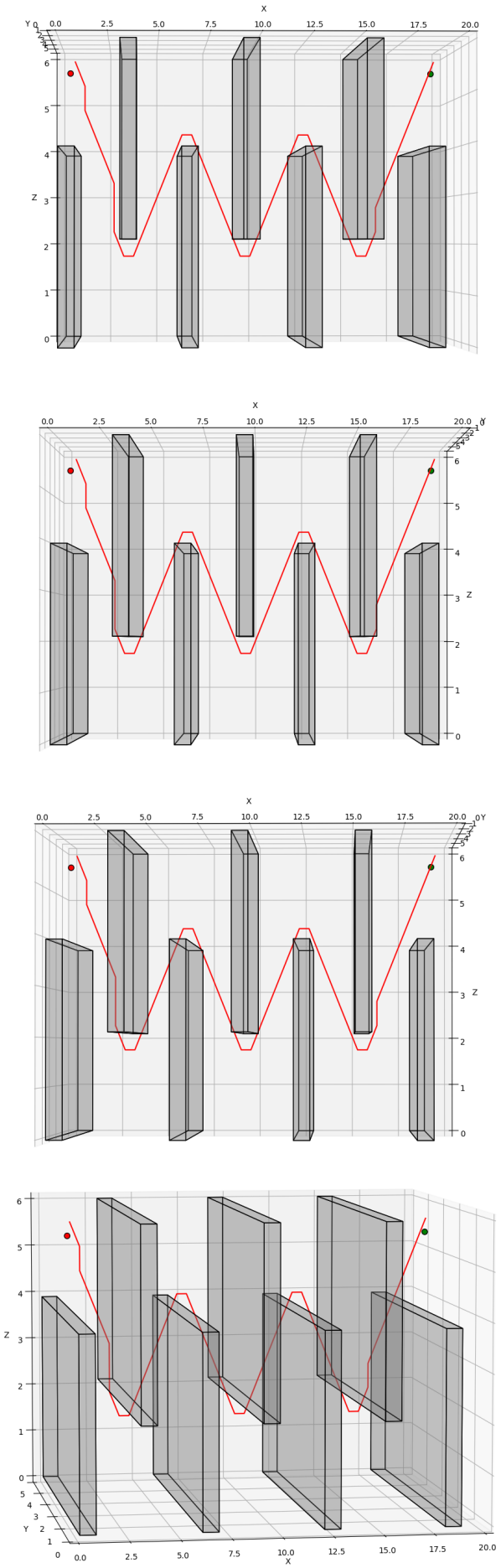

Fig. 3. Agent's path through the flappy bird environment

avoid all the obstacles and reach the goal succesfully. The path length is 76.

*5) **Visualization : A* - window & $\epsilon$ = 0.5**:* The following represent the path planned by A* for the window environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 26.

*6) **Visualization : A* - tower & $\epsilon$ = 0.5**:* The following represent the path planned by A* for the tower environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 30.

*7) **Visualization : A* - room & $\epsilon$ = 0.5**:* The following represent the path planned by A* for the room environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 12.

### *B. A* - epsilon = 1.2*

*1) **Visualization : A* - single cube & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the single cube environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 8.

*2) **Visualization : A* - maze & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the maze environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 77.

*3) **Visualization : A* - flappy bird & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the flappy bird environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 27.

*4) **Visualization : A* - monza & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the monza environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 76.

*5) **Visualization : A* - window & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the window environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 26.

*6) **Visualization : A* - tower & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the tower environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 30.

*7) **Visualization : A* - room & $\epsilon$ = 1.2**:* The following represent the path planned by A* for the room environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 12.

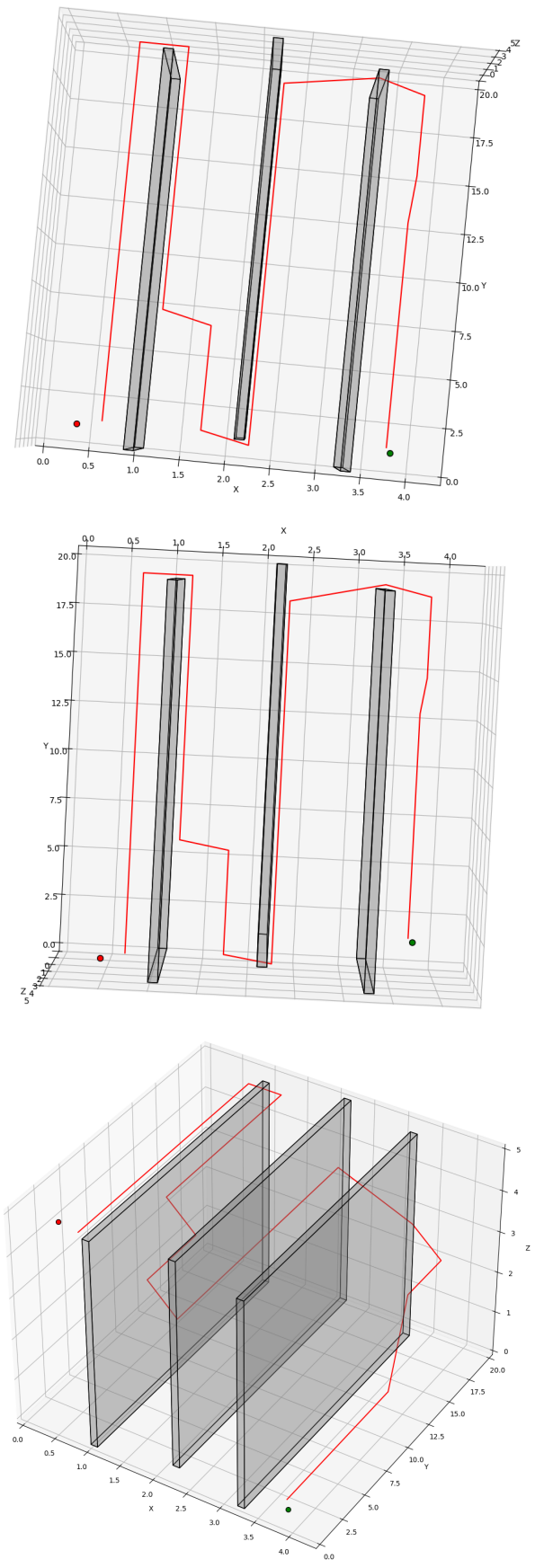

Fig. 4. Agent's path through the monza environment

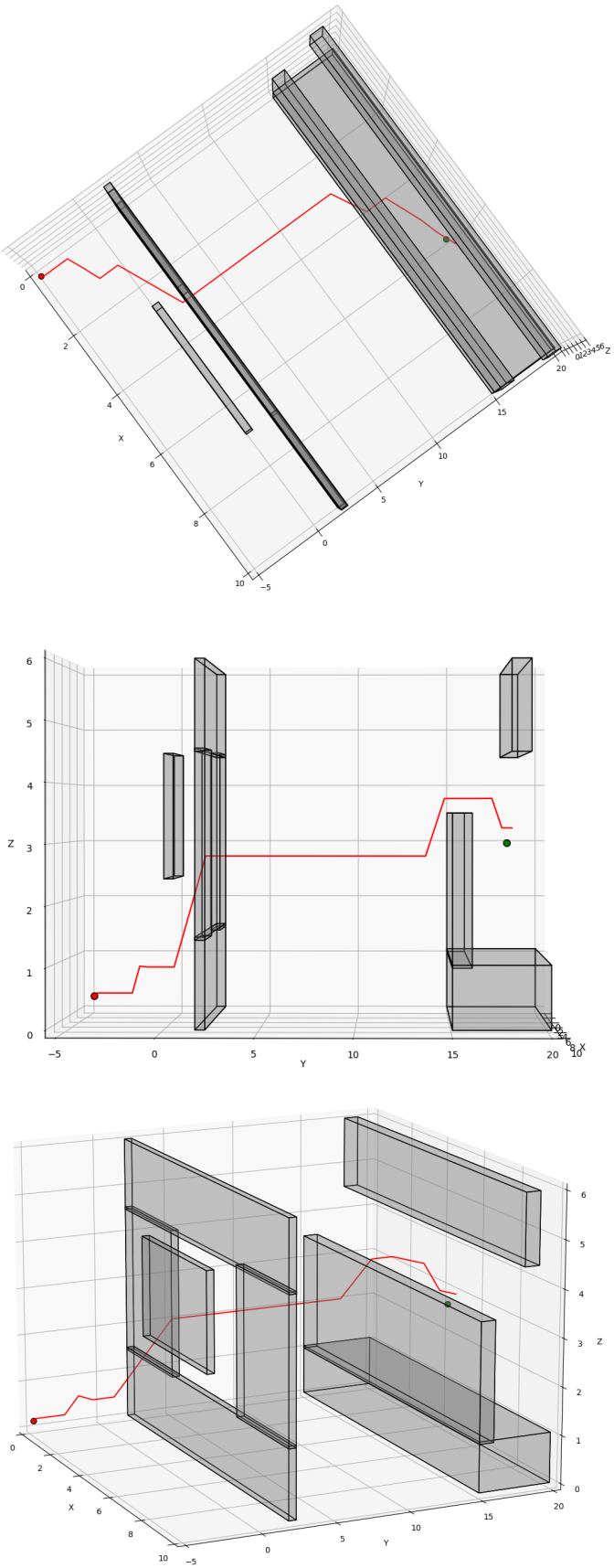

Fig. 5. Agent's path through the window environment

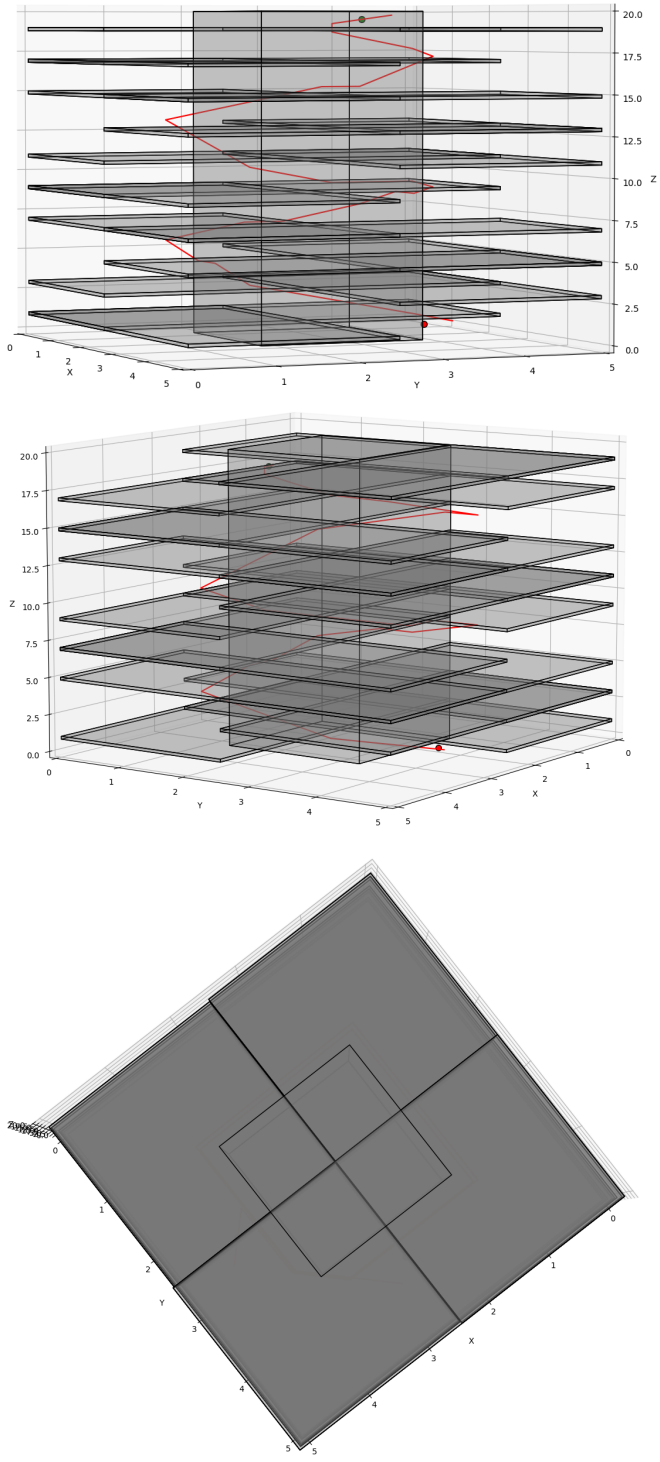

Fig. 6. Agent's path through the tower environment

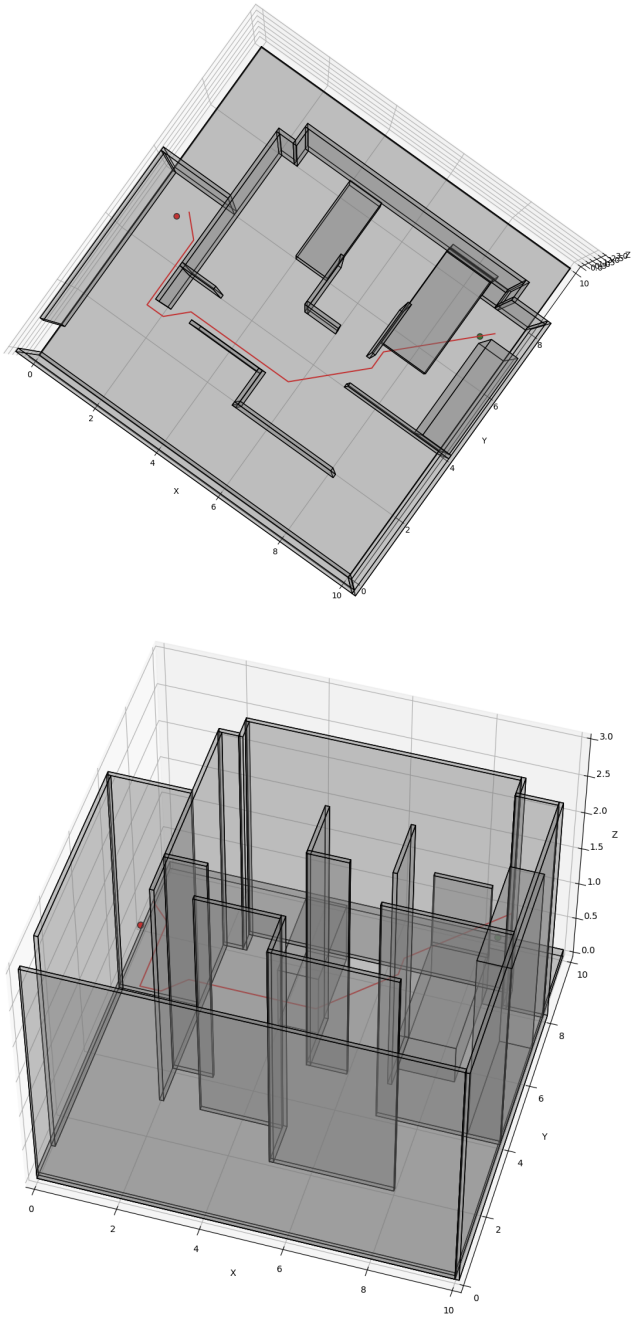

Fig. 7. Agent's path through the room environment

### C. Discussion - A*

As epsilon increases in the heuristic, the path will become less optimal(which can been seen form the images) but the computation time reduces.

### D. RRT

*1)* ***Visualization : RRT - single cube*** *:* The following represent the path planned by RRT for the single cube environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 8.

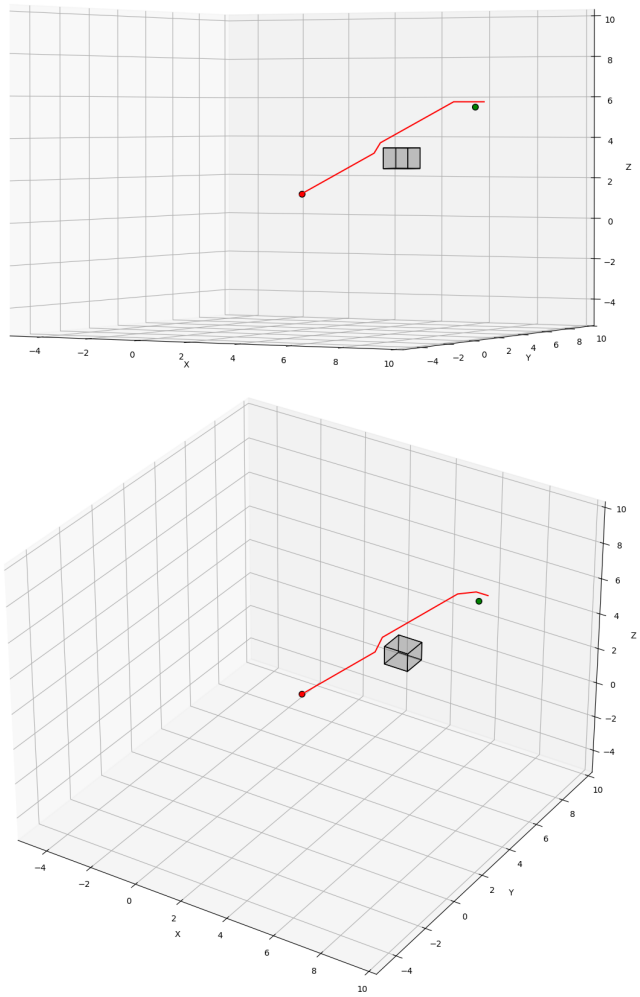

Fig. 8. Agent's path through the single cube environment - sub optimal

*2)* ***Visualization : Bidirectional - RRT - maze****:* Bidirectional RRT has been used for the maze environment as sampling through the environment takes a lot of time. The following represent the path planned by Bidirectional RRT for the maze environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 74.

*3)* ***Visualization : RRT - flappy bird****:* The following represent the path planned by RRT for the flappy bird environment from different angles. It can be seen that the agent is able to avoid the cube obstacle and reach the goal succesfully. The path length is 34.

*4)* ***Visualization : Bidirectional RRT - monza****:* The following represent the path planned by Bidirectional RRT for the monza environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 77.

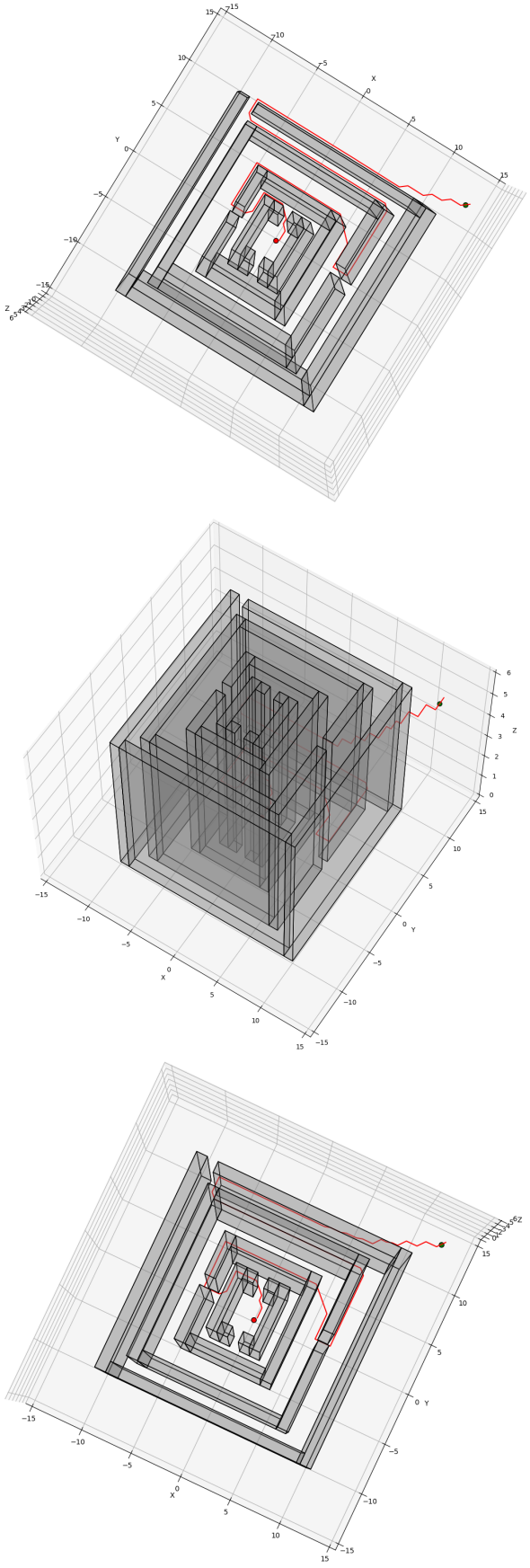

Fig. 9. Agent's path through the maze environment - sub optimal

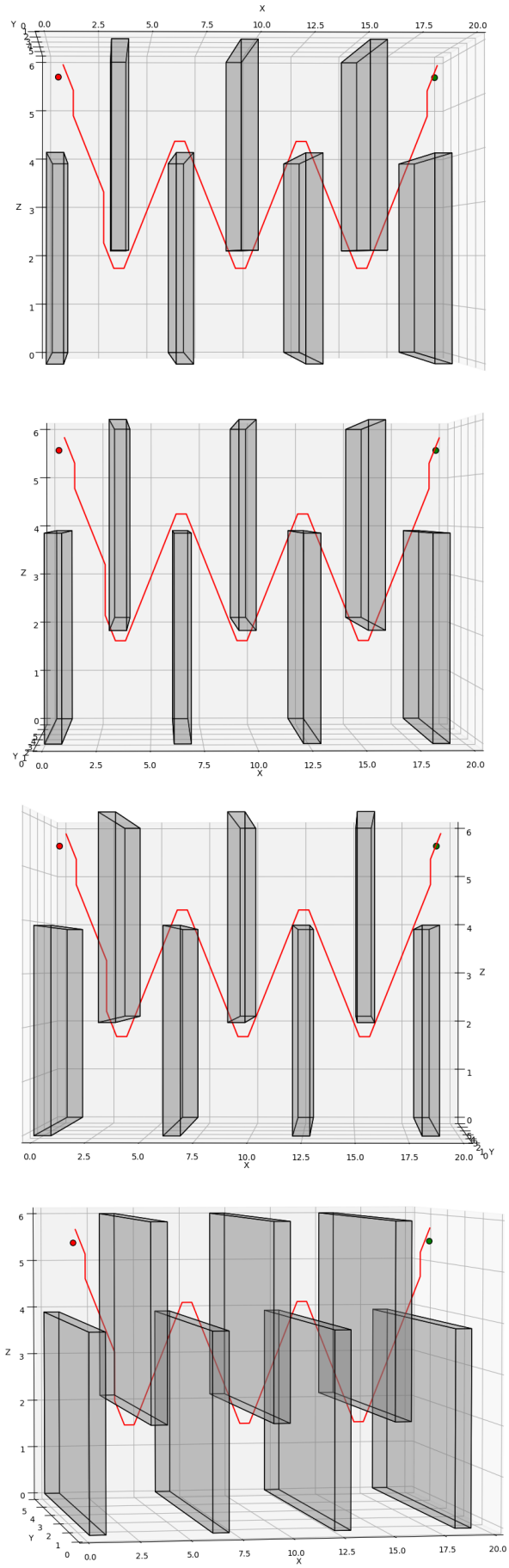

Fig. 10. Agent's path through the flappy bird environment - sub optimal

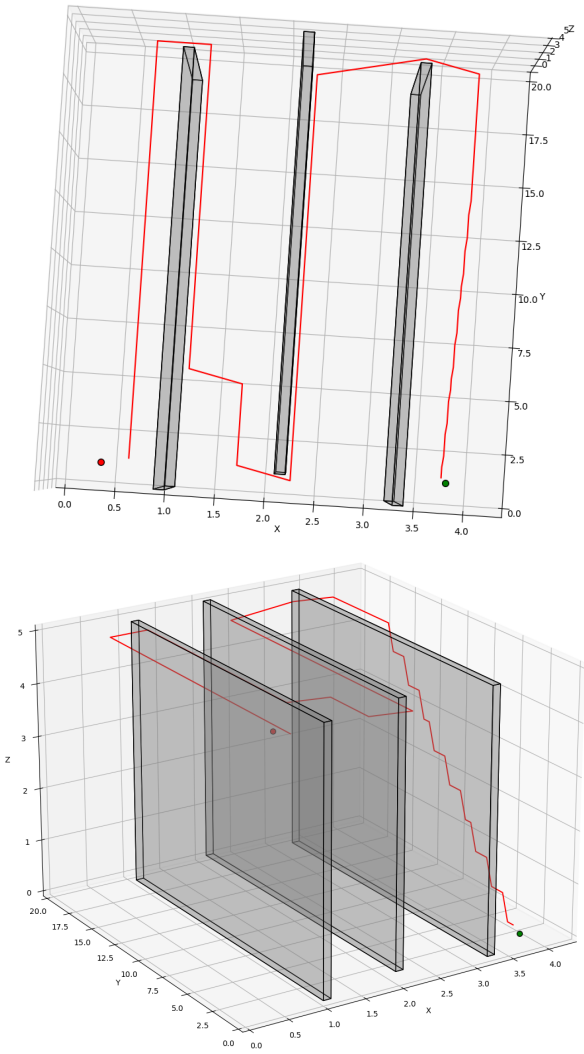

Fig. 11. Agent's path through the monza environment - sub optimal

*5) **Visualization : RRT - window**:* The following represent the path planned by RRT for the window environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 28.

*6) **Visualization : RRT - tower**:* The following represent the path planned by RRT for the tower environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 42.

*7) **Visualization : Bidirectional RRT - room** :* The following represent the path planned by Bidirectional RRT for the room environment from different angles. It can be seen that the agent is able to avoid all the obstacles and reach the goal succesfully. The path length is 10.

### *E. Discussion - RRT*

Since RRT is a sampling based method, it can get stuck easily in complex environments and might need lot of samples

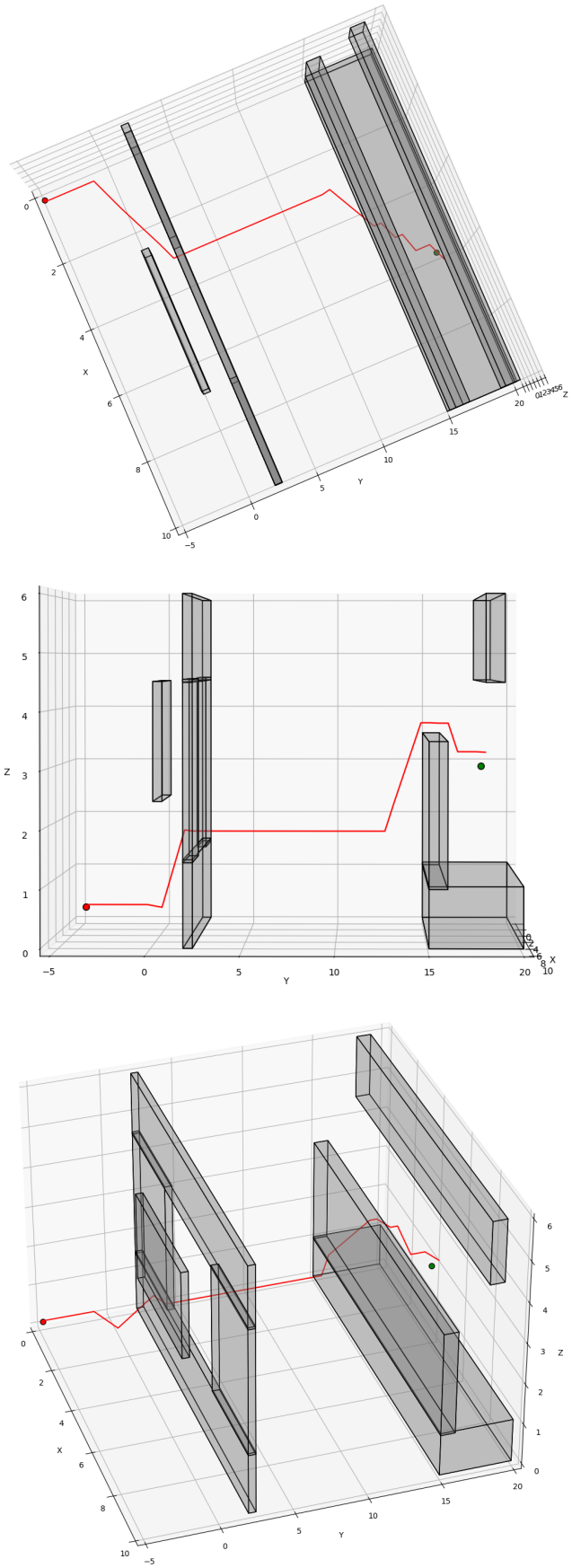

Fig. 12. Agent's path through the window environment - sub optimal

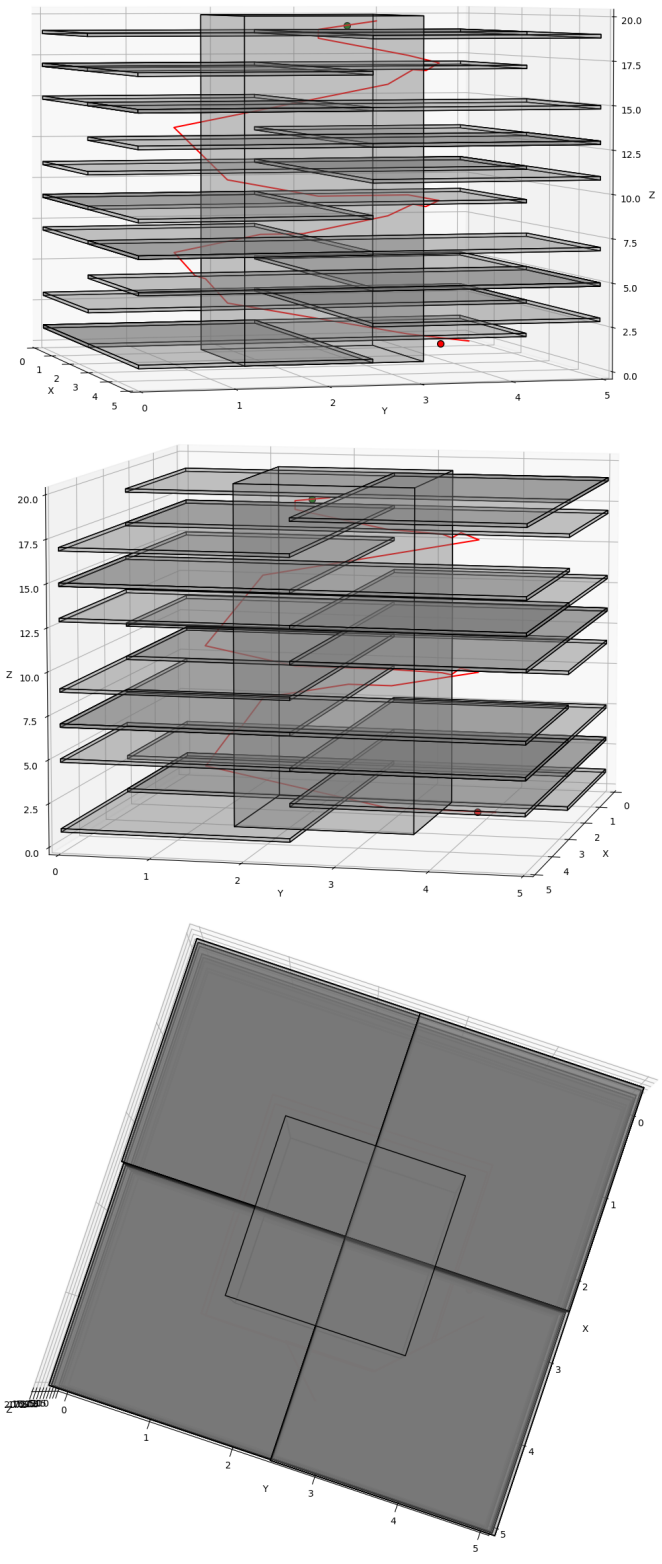

Fig. 13. Agent's path through the tower environment - sub optimal

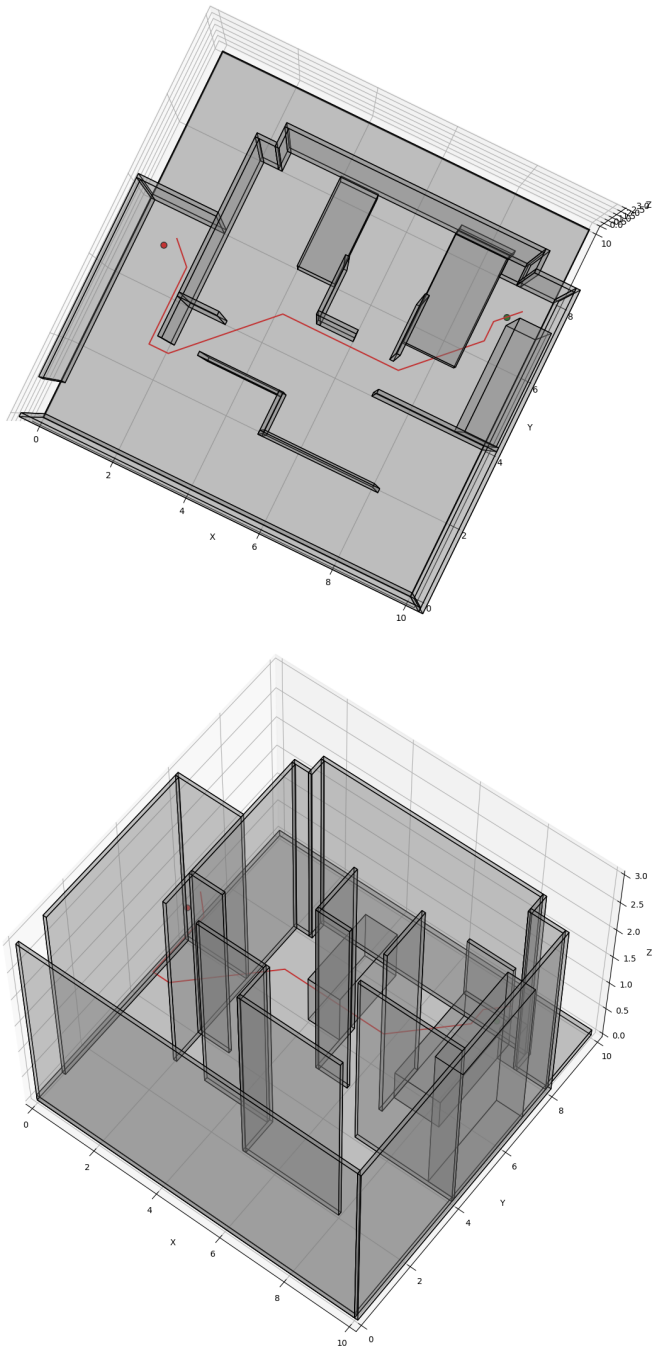

Fig. 14. Agent's path through the room environment - sub optimal

to connect to the goal. In such cases like 'maze', 'monza' and 'room' environments, Bidirectional RRT is used for better performance, which resulted in significant boost in computational time.

Also RRT generally performs worse than A*, since it is randomly sampling the paths rather than moving towards the goal. The same can be seen from the images and increased path length.

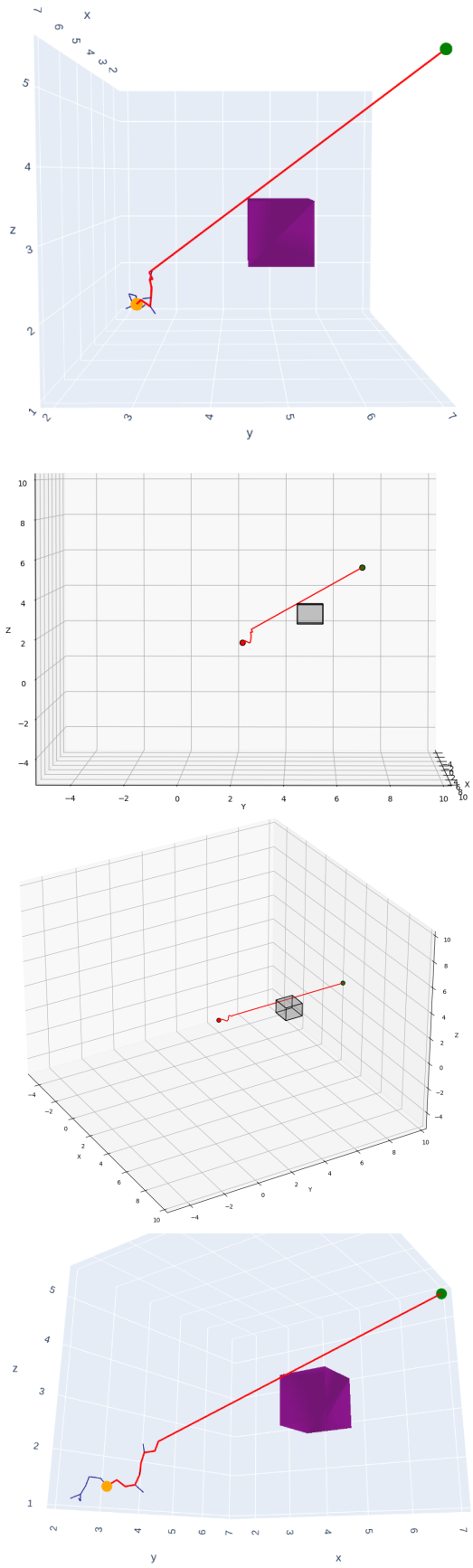

Fig. 15. Agent's path through the single cube environment

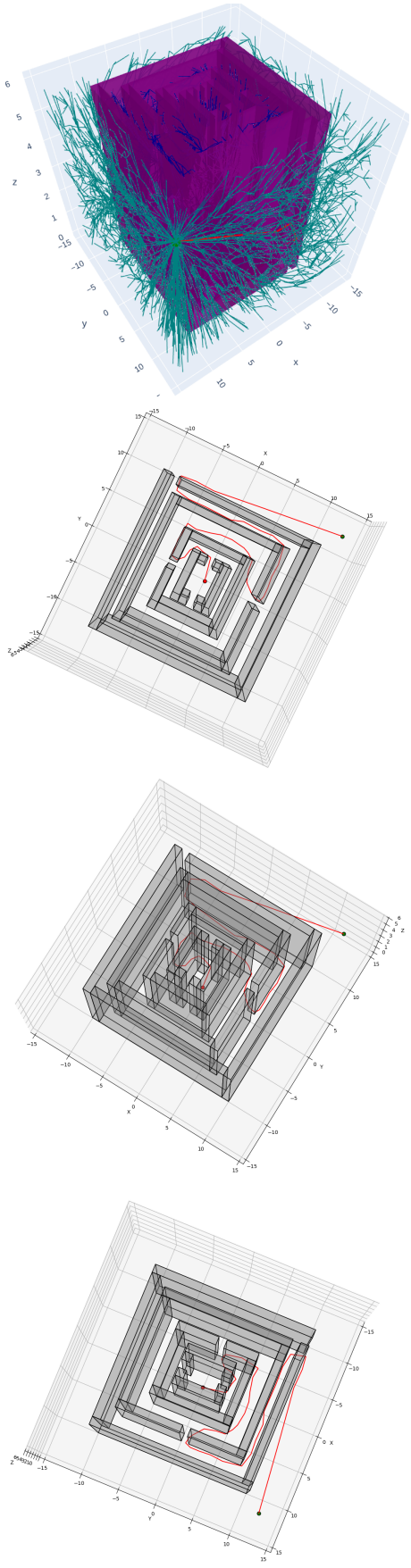

Fig. 16. Agent's path through the maze environment

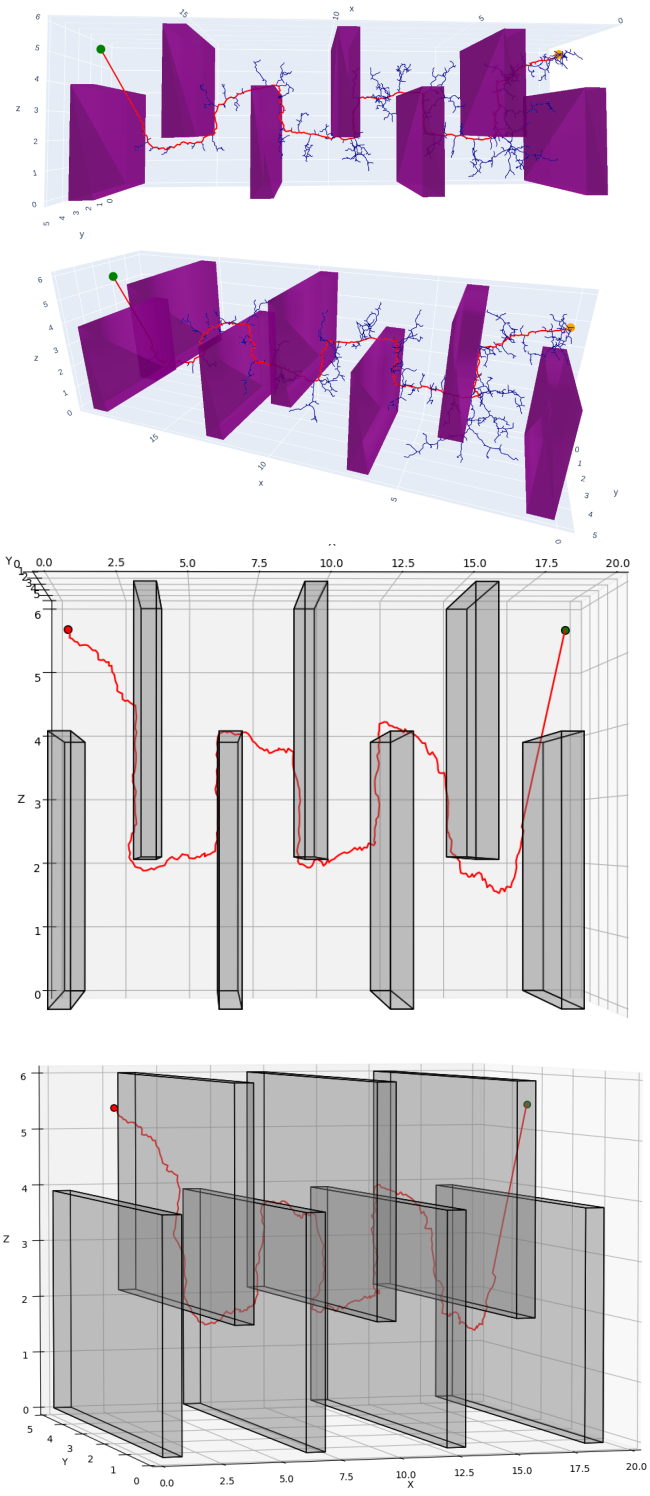

Fig. 17. Agent's path through the flappy bird environment

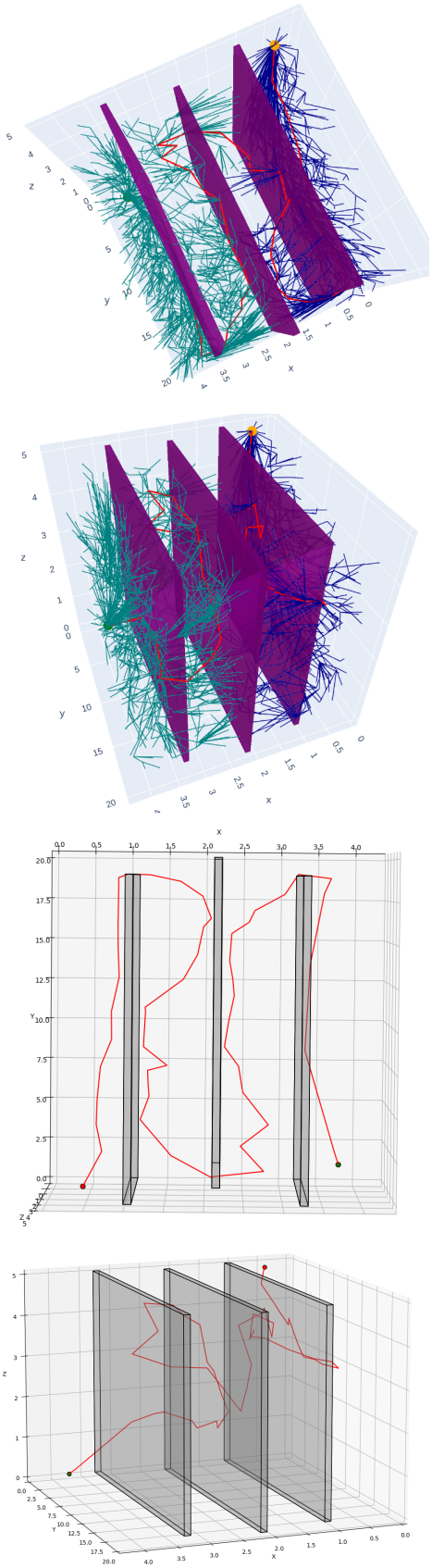

Fig. 18. Agent's path through the monza environment

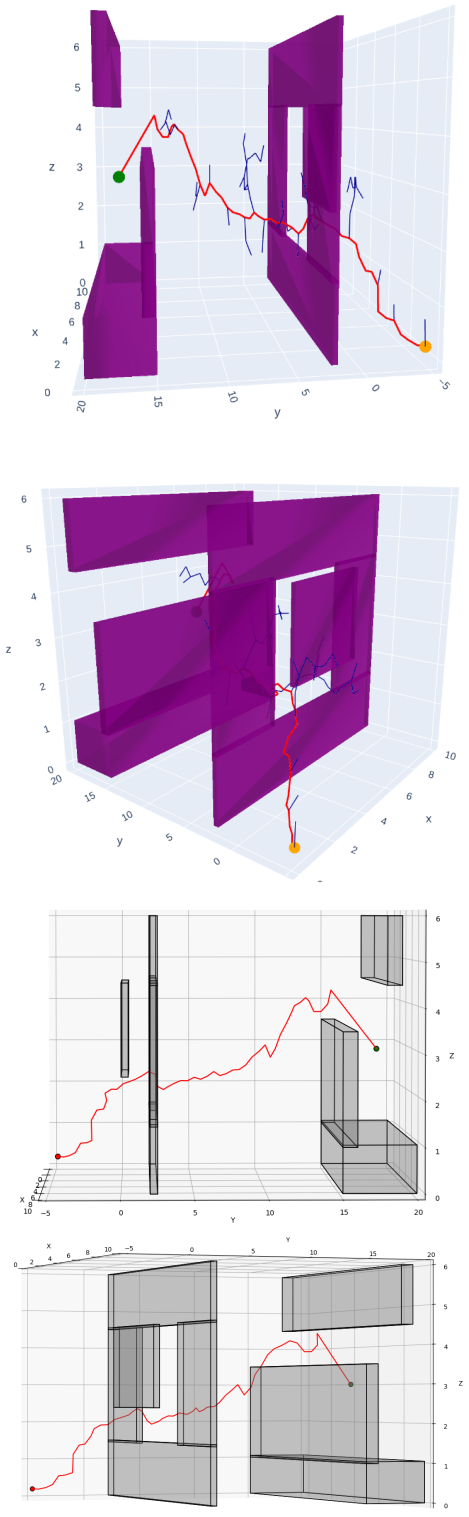

Fig. 19. Agent's path through the window environment

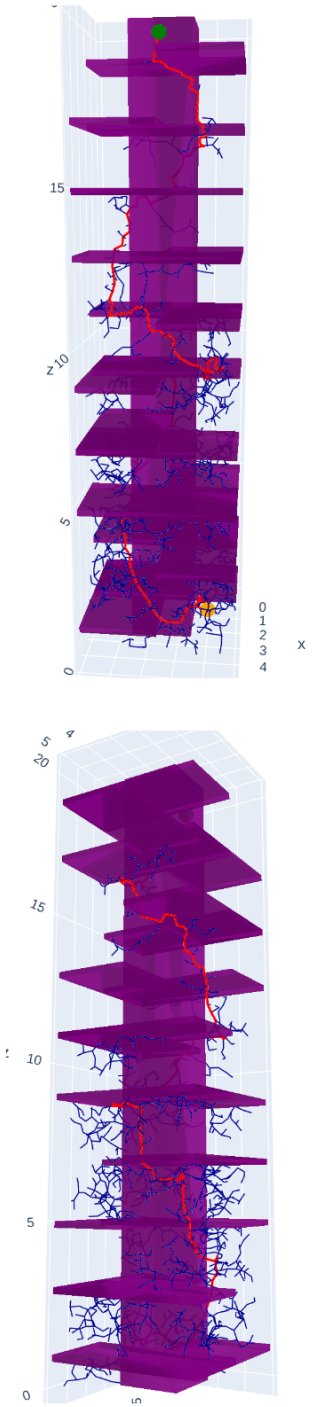

Fig. 20. Agent's path through the tower environment

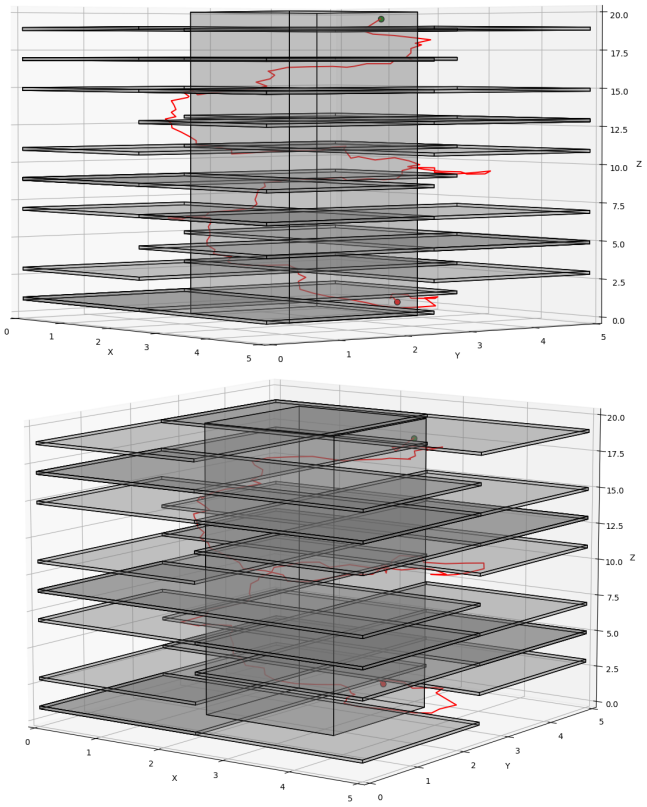


Fig. 21. Agent's path through the tower environment

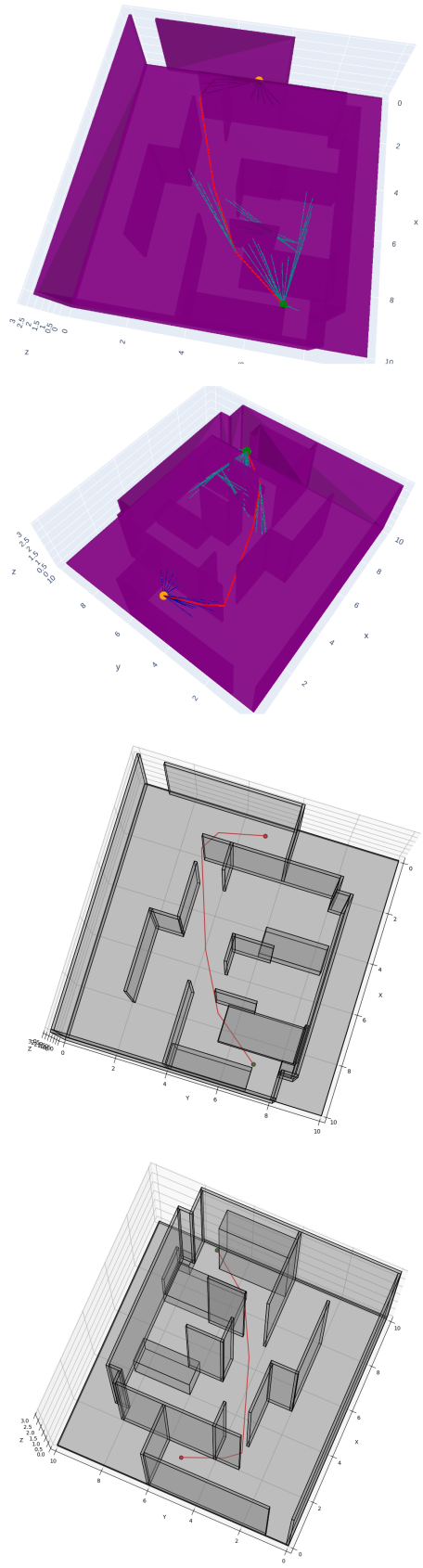

Fig. 22. Agent's path through the room environment