Performance Comparison for Different Motion Planning Algorithms on 3D Environments

Jingpei Lu JIL360@UCSD.EDU
Department of Electrical & Computer Engineering
University of California, San Diego

1 Introduction

Motion planning in robotics is to find a sequence of valid configurations that moves the robot from the source to destination. For example, consider navigating a robot inside a grocery from a start point to the destination without hitting into people and obstacles. We want a motion planning algorithm to autonomously computing a valid and shortest path for achieving this goal. Motion planning has several robotics applications such as autonomous driving, as well as in other fields such as artificial intelligence.

In our problem, we are interested in the task of moving our agent from the start point to the end point in the given 3D environment. Therefore, our objective is to find a shortest and collision free path that navigate our agent from the start point to the goal. We are going to use both search-based algorithms and sampling-basted algorithms to tackle this problem. Specifically, we are going to explore A* algorithm, Learning Real-Time A* (LRTA*) algorithm, Rapidly Exploring Random Tree (RRT*) algorithm, and we are going to compare their performance in the different 3D environments.

2 Problem Formulation

2.1 Problem Setup

In this problem we have a set of different 3-D environments, and the configuration space C is described by a rectangular outer boundary and a set of rectangular obstacle blocks. Each rectangle is described by a 9 dimensional vector, specifying the coordinates of its lower left corner $(x_{min}, y_{min}, z_{min}) \in \mathfrak{R}^3$, right corner $(x_{max}, y_{max}, z_{max}) \in \mathfrak{R}^3$, RGB color $(R, G, B) \in \mathfrak{R}^3$. These form our obstacle space C_{obs}, and the free space C_{free} can be described as $C_{free} \coloneqq \{C \setminus C_{obs}\}$. We also know the start point $x_s \in \mathfrak{R}^3$ and the goal coordinates $x_\tau \in \mathfrak{R}^3$ for each of the available environments. Our objective is to navigate our robot from the start point to the goal point with an efficient and collision free path $Q \coloneqq (x_0, x_1, x_2, \dots, x_T)$, where $x_i \in \mathfrak{R}^3$ is a 3D point on the path. We also have the constraint that given the current robot position $x_t \in \mathfrak{R}^3$, our next robot move $x_{t+1} \in \mathfrak{R}^3$ should satisfy the condition that:

1. The moving distance should not be greater than 1, i.e., $||x_{t+1} - x_t|| \leq 1$
2. The robot remains collision free at x_{t+1}, i.e. $x_{t+1} \in C_{free}$
3. The next position x_{t+1} is produced within 2 seconds
4. The robot eventually reaches the goal, i.e. the last position of our path should be x_τ

2.2 Path Planning Problem

Given the Configuration space C, the obstacle space C_{obs}, the free space C_{free}, and the initial state $x_s \in C_{free}$, the goal state $x_\tau \in C_{free}$, we can find a set of paths $\mathbb{Q}_{s,\tau}$ that connect the initial state to the goal state. Each path $Q \in \mathbb{Q}_{s,\tau}$ is a continuous function $Q: [0,1] \to C$. Our objective is to find a path that is efficient and collision free. Our optimal path Q should satisfy the following criteria:

- It is a feasible path, i.e. $Q: [0,1] \to C_{free}$ and $Q(0) = x_s$, $Q(1) = x_\tau$

- It has the minimum cost, i.e. $J(Q^*) = \min_{Q \in \mathbb{Q}_{s,\tau}} J(Q)$
  where the cost function $J(\cdot)$ is defined as
  $$J(Q) = \sum_{t=0}^{T-1} c_{t,t+1} = \sum_{t=0}^{T-1} ||x_{t+1} - x_t|| \tag{1}$$
  which is the Euclidean distance between the two positions.

3 Technical Approach

3.1 Implementing the A* Algorithm

The A* algorithm is a search-based algorithm and it is the modification to the Label Correcting algorithm. It also uses the concept of label, such that g_i that keeps (an estimate of) the lowest cost from start node x_s to each visited node $x_i \in C$. It also keeps the concept of OPEN list which stores a set of nodes that can potentially be part of the shortest path to x_τ. However, the OPEN list is now implemented as a priority queue and the admission of the OPEN list is strengthened as:

$$g_i + c_{ij} + h_j < g_\tau \tag{2}$$

where h_j is a positive lower bound on the optimal cost to get from node x_j to x_τ, known as heuristic. And c_{ij} is the Euclidean distance between x_i and x_j. Here, we define the heuristic h_j to be the distance from the node j to the goal as

$$h_j = ||x_j - x_\tau|| \tag{3}$$

The problem for A* algorithm is that it can only be used for a known environment with the constructed graph, where the children of each node in the graph is known. To adopt the A* algorithm in our problem, we have to find a way to obtain the children for each node consistently. My approach is that we first discretize the environment to a 3D grid, and the length for each edge of small square is 0.5. Then, for each node $x_i = (x, y, z) \in \mathfrak{R}^3$, we define its children to be a set of nodes as

$$x_j \in X_c \coloneqq \{(x_c, y_c, z_c) | \, x_c \in \{x - 0.5, x, x + 0.5\}, y_c \in \{y - 0.5, y, y + 0.5\}, \\ z_c \in \{z - 0.5, z, z + 0.5\}\} \setminus \{x_i\} \tag{4}$$

By doing so, the children for each state are deterministic, and within the distance 1 from the parent node to satisfy the condition 1. Then, we can apply the A* algorithm to find the shortest path.

The implementation of the algorithm is shown below:

Algorithm 1: A* Algorithm

1: OPEN ← $\{x_s\}$, CLOSED ← {}
2: $g_s = 0$, $g_i = \infty$ for all $x_i \in C \setminus \{x_s\}$
3: while $x_\tau \notin$ CLOSED do:
4: remove x_i with smallest $f_i \coloneqq g_i + h_i$ from OPEN
5: insert x_i into CLOSED
6: for $x_j \in$ Children(x_i) and $x_j \notin$ CLOSED and Collisionfree(x_j, x_i) do:
7: if $g_i > (g_j + c_{ij})$ then:
8: $g_j \leftarrow (g_i + c_{ij})$
9: Parent(x_j) ← x_i
10: Insert x_j to OPEN

Figure 1: The A algorithm*

After the termination of the algorithm, we can find the shortest path by iteratively finding the parents from the goal x_τ to the start node x_s.

3.2 Implementing the LRTA* Algorithm

The LRTA* algorithm is an agent-centric search-based algorithm. The beauty of the agent-centric search is that we don't need to construct the graph or planning the path before the robot can move. We can plan path the while the robot is moving and iterate these two steps until arrive the goal. Therefore, it can be used on really large environments, where it is impossible to compute the path all the way to the goal.

The key idea of the LRTA* algorithm is that the heuristic needs to be updated over time. For each time, the robot repeatedly moves to the most promising adjacent cell $s \in \Re^3$ using

$$s = \arg \min_{x_j \in Children(x_i)} c_{sj} + h_j \tag{5}$$

And updating a heuristic:

$$h_s = \min_{x_j \in Children(s)} c_{sj} + h_j \tag{6}$$

The implementation of the LRTA* algorithm with a lookahead of one node is shown below:

Algorithm 2: LRTA* Algorithm

1: initialize the heuristic: $h \leftarrow h_0$
2: initialize the current state: $s \leftarrow x_s$
3: while $s \neq x_\tau$ do:
4: generate children one move away from state s
5: find the state s' with the lowest $f_j = g_j + h_j$ as in (5)
6: update h(s) to f(s′) if f(s′) is greater as in (6)
7: execute the action to get to s' as $s \leftarrow s'$
8: end while

Figure 2 : LRTA algorithm with a lookahead of one*

After the termination of the algorithm, the trajectory of the robots is the path from the start to goal.

3.3 Implementing the RRT* Algorithm

The RRT* algorithm is a sampling-based algorithm. Although the sampling-based algorithm may not be able to find the shortest path, its advantage is that it is efficient when dealing the high-dimensional planning problem as it is faster and requires less memory than search-based planning in many domains. The key idea of the sampling-based algorithm is to construct a roadmap from start point to end goal with sampled configurations. To implementing the sampling-based algorithm, we first need to define some primitive procedures:

- Sample: returns independent and identical distributed (iid) samples from configuration space C
- SampleFree: returns iid samples from C_{free}
- Nearest: given a graph G = (V, E) with V ⊂ C and a point x ∈ C, returns a vertex v ∈ V that is closest to x:

$$\text{Nearest}((V, E), x) := \arg \min_{v \in V} ||x - v||$$

- Near: given a graph G = (V, E) with V ⊂ C, a point x ∈ C, and r > 0, returns the vertices in V that are within a distance r from x:

$$Near\big((V,E),x,r\big) \coloneqq \{v \in V \mid \|x-v\| \le r\}$$

- Steer: given points x, y ∈ C and $\epsilon > 0$, returns a point z ∈ C that minimizes $\|z-y\|$ while remaining within ϵ from x:
$$Steer_\epsilon(x,y) \coloneqq \underset{z:\|z-x\|\le\epsilon}{\operatorname{argmin}} \|z-y\|$$
- CollisionFree: given points x, y ∈ C, returns True if the line segment between x and y lies in C_{free} and False otherwise

Instead of constructing a roadmap from every node to every node in configuration space, the roadmap of RRT* is a tree constructed from random samples with root x_s. The tree is grown until it contains a path to x_τ. RRT* are well-suited for single-shot planning between a single pair of x_s and x_τ.

To implementing RRT*, we also need to define two cost function:

- $Cost(x)$ is implemented the same as the label in A* algorithm, such that it keeps (an estimate of) the lowest cost from start node x_s to each visited node x
- $Cost(x_i, x_j)$ is the Euclidean distance between two nodes, i.e. $Cost\big(x_i, x_j\big) \coloneqq \|x_i - x_j\|$

For my implementation, I set r and ϵ to be 1 to ensure the condition 1 is satisfied. The implementation of the RRT* algorithm is on the figure below:

Algorithm 3: RRT* Algorithm

1: $V \leftarrow \{x_s\}, E \leftarrow \{\}$
2: while $x_\tau \notin V$ do:
3: $\quad x_{rand} \leftarrow SampleFree()$
4: $\quad x_{nearest} \leftarrow Nearest((V,E), x_{rand})$
5: $\quad x_{new} \leftarrow Steer(x_{nearest}\ ,\ x_{rand})$
6: $\quad$ if $CollisionFree(x_{nearest}, x_{new})$ then:
7: $\qquad X_{near} \leftarrow Near((V,E), x_{new}, \min\{r,\epsilon\})$
8: $\qquad V \leftarrow V \cup \{x_{new}\}$
9: $\qquad c_{min} \leftarrow Cost(x_{nearest}) + Cost(x_{nearest}, x_{new})$
10: $\qquad$ for $x_{near} \in X_{near}$ do:
11: $\qquad\quad$ if $CollisionFree(x_{near}, x_{new})$ then:
12: $\qquad\qquad$ if $Cost(x_{near}) + Cost(x_{near}, X_{new}) < c_{min}$ then:
13: $\qquad\qquad\quad x_{min} \leftarrow x_{near}$
14: $\qquad\qquad\quad c_{min} \leftarrow Cost(x_{near}) + Cost(x_{near}, x_{new})$
15: $\qquad E \leftarrow E \cup \{(x_{min}, x_{new})\}$
16: $\qquad$ for $x_{near} \in X_{near}$ do:
17: $\qquad\quad$ if $CollisionFree(x_{near}, x_{new})$ then:
18: $\qquad\qquad$ if $Cost(x_{new}) + Cost(x_{near}, X_{new}) < Cost(x_{near})$:
19: $\qquad\qquad\quad x_{parent} \leftarrow Parent(x_{near})$
20: $\qquad\qquad\quad E \leftarrow \big(E \setminus \big\{(x_{parent}, x_{near})\big\}\big) \cup$
$\{(x_{new}, x_{near})\}$
21: return $G = (V,E)$

Figure 3: the RRT algorithm*

After the termination of the algorithm, we can find the shortest path by iteratively finding the parents from the goal x_τ to the start node x_s.

4 Result

4.1 Performance Comparison

The runtime performance

	Single_cube	Maze	Flappy_bird	Monza	Window	Tower	Room
A*	8s	680s	76s	40s	138s	67s	5s
LRTA*	1s	>10000s	15s	1010s	2s	12s	10s
RRT*	2s	1477s	14s	9958s	2s	14s	3s

Number of moves needed for the resulting path

	Single_cube	Maze	Flappy_bird	Monza	Window	Tower	Room
A*	12	150	41	151	46	48	21
LRTA*	18	>200000	2962	168053	244	1060	1280
RRT*	13	125	49	101	39	48	20

The length of each move is less than 1.

4.2 Resulting Path for A*

The resulting path for A* algorithm

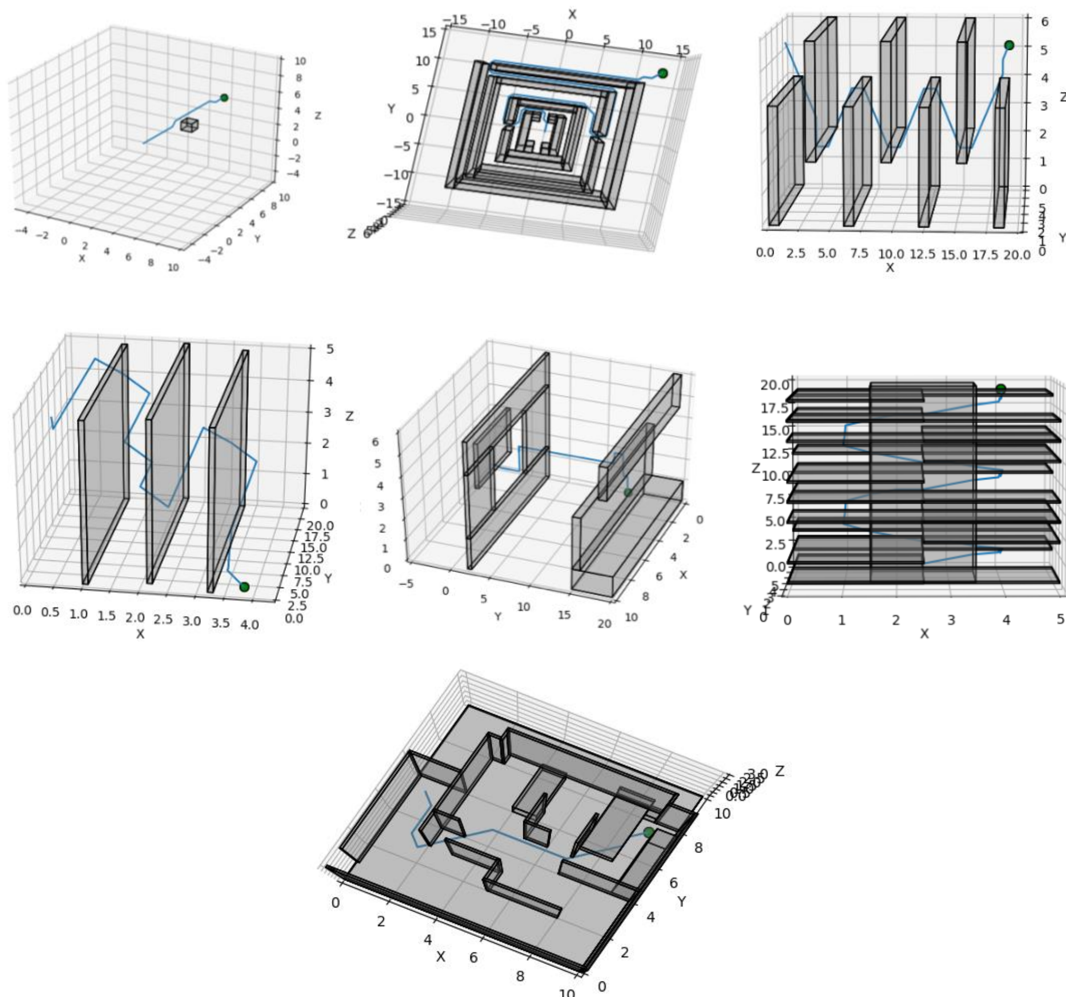

Figure 4

Figure 4 shows the resulting path from 7 different 3D environments obtained using A* algorithm. The environments with the order from left to right and from top to bottom are “Single_cube”, “Maze”, “Flappy_bird”, “Monza”, “Window”, “Tower”, “Room”.

4.3 Resulting Path for LRTA*

The resulting path for LRTA* algorithm

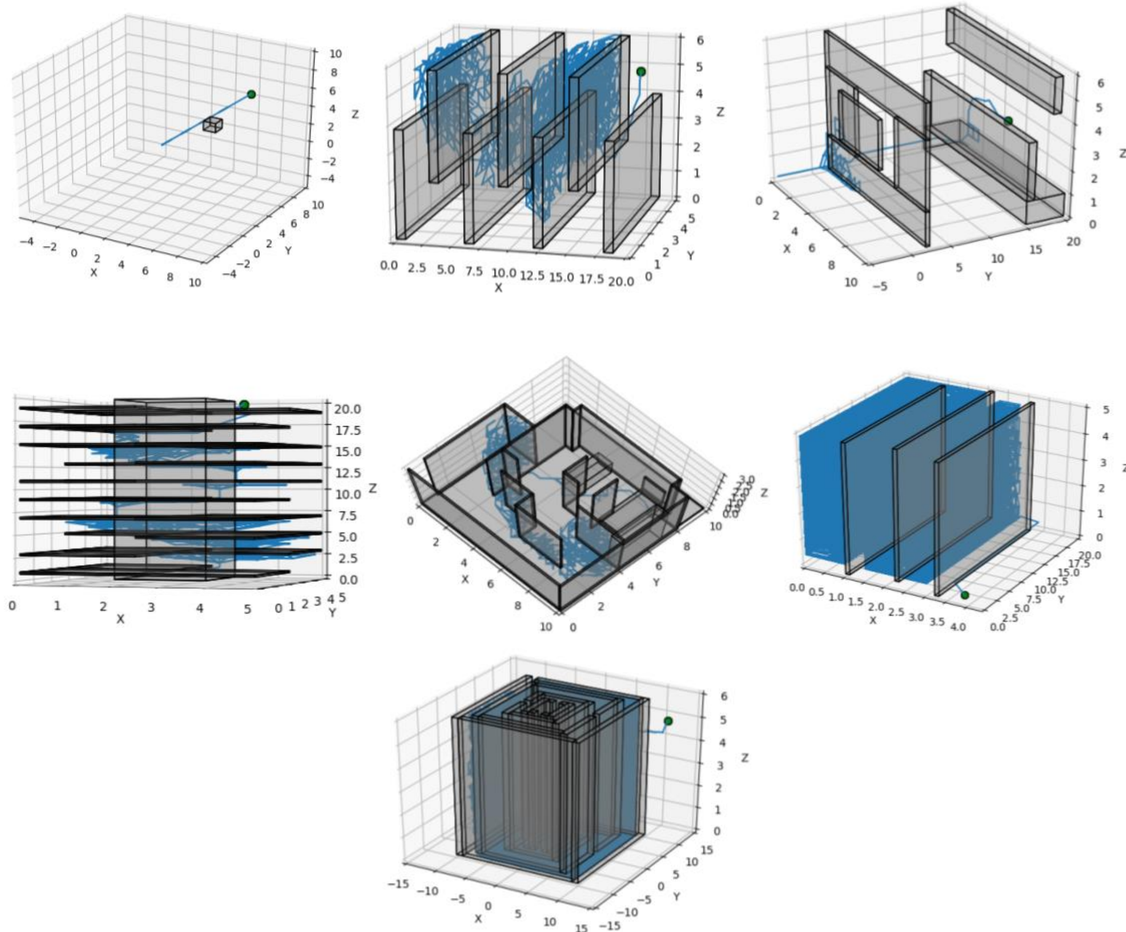

Figure 5

Figure 5 shows the resulting path from 7 different 3D environments obtained using LRTA* algorithm. The environments with the order from left to right and from top to bottom are “Single_cube”, “Flappy_bird”, “Window”, “Tower”, “Room”, “Monza”, “Maze”.

4.4 Resulting Path of RRT*

The resulting path for RRT* algorithm

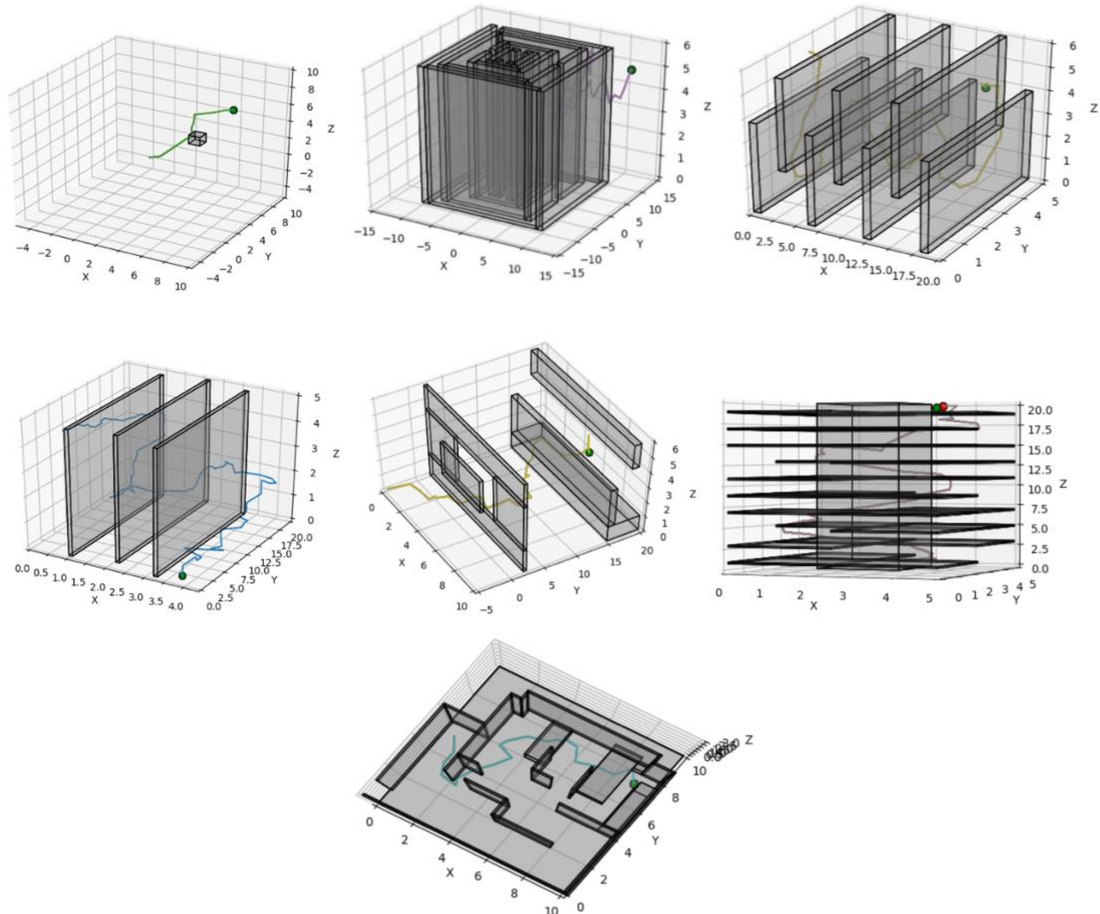

Figure 6

Figure 6 shows the resulting path from 7 different 3D environments obtained using RRT* algorithm. The environments with the order from left to right and from top to bottom are "Single_cube", "Maze", "Flappy_bird", "Monza", "Window", "Tower", "Room".

4.5 Discussion

From the result, we can see that not a single algorithm can beat others in all the cases. All of them have some advantages and disadvantages. Therefore, for different environments, their performance can be significantly affected.

The runtime of A* is very much depend on the size of the environment, and less affected by the complexity compared to other algorithms. The A* is the fastest one on "Maze" and "Monza". For these two environments, the heuristic can't help with the planning, which makes the planning problem more difficult. The A* does pretty good job on these because it is not affected by the heuristic by putting the visited nodes in the CLOSED node. However, if an environment is very large, even if it is simple, A* still needs some time on the planning, which make it worse than the LRTA* and RRT*. In terms of the resulting path, the A* can always generate the shortest path based on how we discretize the environment.

The runtime of LRTA* is significantly affected by the complexity of the environment, and less affected by the size compared to other algorithms. The LRTA* is the fastest on "Single_cube", "Window", and "Tower". These two environments are not very complicated, and the heuristic

provides very helpful information during the planning. The LRTA* is very sensitive to the heuristic, and it can be misguided by the heuristic very often. Therefore, even if an environment is small, if can also be struck because of the nonhelpful heuristic. However, as long as the heuristic is helpful, LRTA* can have a good performance even if the environment is large. Another advantage of the LRTA* is that it is agent-centric, so we don't need to know the entire environment in advance and do the path planning beforehand. However, there are lots of back-forward steps on the resulting path which makes the path not optimal.

The runtime of RRT* is affected by both the size and the complexity of the environment. The RRT* performs the best on "Flappy_bird" and "Room", because these environments have a good balance between the complexity and size. For these two environments, the heuristic sometimes helps but not always help for the planning. RRT* mitigates the effect of the heuristic by generating the random sample in different direction. However, heuristic helps to grow the tree by provides the information of the direction. The size of the environments also has an effect because RRT* needs to sample more configurations for the larger environment, and it avoid sampling everywhere with the help of the heuristic. Therefore, I think RRT* has a good balance between the use of heuristic and the randomness. In terms of the resulting path, it is always the shortest based on the sampled configurations.