1.

For DFS, with noise $w_t = 0$.

State: $x_0 \in \mathcal{X}$, which is a finite space.

goal: $\min_{u_{0:T-1}} q(x_T) + \sum_{t=0}^{T-1} l_t(x_t, u_t)$

s.t. $x_{t+1} = f(x_t, u_t)$

$x_t \in \mathcal{X}$, $u_t \in \mathcal{U}$.

we can transfer it into a DSP problem:

State $\Leftrightarrow$ node. $\mathcal{V} := \left(\bigcup_{t=0}^{T} \{(t, x_t) \mid x_t \in \mathcal{X}\}\right) \cup \{\tau\}$

initial start point $s = (0, x_0)$

τ is an artifical terminal state.

cost $\Leftrightarrow$ edge.

$$\mathcal{C} = \left\{ \left((t, x_t), (t+1, x_{t+1}), c\right) \middle| c = \min_{\substack{u \in \mathcal{U}(x_t) \\ \text{s.t. } x_{t+1} = f(x_t, u_t)}} l_t(x_t, u) \right\} \cup \left\{ (T, x_T), \tau, q(x_T) \right\}$$

a. Thus, for Dijkstra's algorithm, its backward version is

Backward Dijkstra's Algorithm

$OPEN \leftarrow \{\tau\}$, $g_\tau = 0$, $g_i = \infty \ \forall i \in \mathcal{V} \setminus \{\tau\}$.

while OPEN is not empty do

  Remove $i = \operatorname{argmin}_{j \in OPEN} g_j$ from OPEN

  for $k \in$ Children(i)

    if $g_k > g_i + c_{k,i}$

      $g_k \leftarrow g_i + c_{k,i}$

      Parent$(k) \leftarrow i$

      if $k \neq s$ then

        OPEN.append(k)

It actually investigates the same states as Dynamic Programming(DP). For DP.

$$V_t(i) = \min_{j \in \mathcal{V} \setminus \{\tau\}} c_{i,j} + V_{t+1}(j)$$
$$\pi_t(i) = \operatorname{argmin}_{j \in \mathcal{V} \setminus \{\tau\}} c_{i,j} + V_{t+1}(j)$$

will still investigate all children of i to find out the minimum.

(b) Yes, because of the equivalence between DFS and DSP.

Here, a heuristic means a lower bound estimation for cost that takes from state x_t to terminal state x_τ.

Backward A^* Algorithm

OPEN $\leftarrow \{\tau\}$, CLOSED $= \{\}$. $\epsilon \geq 1$

$g_\tau = 0$, $g_i = \infty$ for $i \in \mathcal{V} \setminus \{\tau\}$

while $s \notin$ CLOSED do

  Remove $i = \operatorname{argmin}_{j \in \text{OPEN}} f_j = \operatorname{argmin}_{j \in \text{OPEN}} g_j + \epsilon h_j$ from OPEN

  CLOSED.append(i)

  for $k \in$ Children(i) and $k \notin$ CLOSED do

    if $g_k > g_i + c_{k,i}$ then

      $g_k \leftarrow g_i + c_{k,i}$

      Parent(k) $\leftarrow i$

      OPEN.append(k)

2.

(a) Because $h^{(1)}$ and $h^{(2)}$ are consistent, then

$$h(x_T) = \max\{h^{(1)}(x_T), h^{(2)}(x_T)\} = 0$$

Also.

$$\begin{aligned} h(x_i) &= \max\{h^{(1)}(x_i), h^{(2)}(x_i)\} \\ &\leq \max\{h^{(1)}(x_j) + cost(x_i, x_j), h^{(2)}(x_j) + cost(x_i, x_j)\} \\ &= cost(x_i, x_j) + \max\{h^{(1)}(x_j), h^{(2)}(x_j)\} \end{aligned}$$

Hence h is also consistent.

(b) If $h^{(1)}$, $h^{(2)}$ are heuristic, then

$$h(x_T) = h^{(1)}(x_T) + h^{(2)}(x_T) = 0.$$

Plus,

$$\begin{aligned} h(x_i) &= h^{(1)}(x_i) + h^{(2)}(x_i) \\ &\leq 2cost(x_i, x_j) + h^{(1)}(x_j) + h^{(2)}(x_j) \\ &= 2cost(x_i, x_j) + h(x_j) \end{aligned}$$

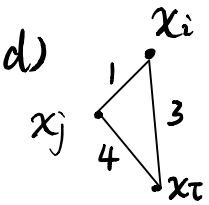
Let $\epsilon = 2$, then h is $2-$consistent.

(c) h is consistent, then

$$\begin{aligned} h(x_i) &\leq cost(x_i, x_T) + \cancel{h(x_T)}^{\,0} \\ &\leq dist(x_i, x_T) \end{aligned}$$

Thus h is also admissible.

(d)



If h is consistent then

$$h(x_i) \leq 1 + h(x_j)$$

$$h(x_j) \leq 4 + 0 = 4$$

If I set $h(x_i)=3$, $h(x_j)=1$, $h(x_T)=0$. then $h(x_i) > 1 + h(x_j)$

$h(x_j) \leq 4+0=4$

however $h(x_i) \leq dist(x_i, x_T)=3$

$h(x_j) \leq dist(x_j, x_T)=4$

Thus $h(x_i)=3$, $h(x_j)=1$, $h(x_T)=0$ is admissible but not consistent.

(e) Assume i is expanded and h is consistent, then $f_i = g_i + h_i \leq f_j \ \forall j \in OPEN$.

Suppose g_i can be improved again, i.e. g_i is larger than the least cost from s to i ($g_i > dist(s, i)$)

Then, there at least exists one state j on an optimal path from s to i, s.t. $j \in OPEN$, $j \notin CLOSED$, $f_i \leq f_j$

However.

$f_i = g_i + h_i > dist(s, i) + h_i = g_j + [dist(j, i) + h_i] \geq g_j + h_j = f_j$

contradicts with $f_i \leq f_j$

Hence g_i cannot be approved.

(f) h is neither consistent, nor admissible.

h is not consistent.

For example, $x_i \in \mathbb{R}^2$, Suppose $x_i = (6, 8)^T$, $x_T = (0, 0)^T$

then $h(x_i) = 8 + 0.4 \times 6 = 10.4$.

$cost(x_i, x_T) = 10$, $h(x_T)=0$

then $h(x_i) > h(x_T) + cost(x_i, x_T)$.

Therefore, h is not consistent.

h is not admissible.

Still suppose $x_i = (6, 8)^T$, $h(x_i) = 10.4$

Then $dist(x_i, x_t) = cost(x_i, x_t) = 10 < h(x_i)$

Thus h is not admissible.

3.

(a) We can formulate this problem as follows:

State space: $\mathcal{X} = [n] = \{1, 2, \cdots, n\}$, state $x_t \in \mathcal{X}$.

Control space: $\mathcal{U} = \{1, 0\}$, where "1" means we choose to recharge, "0" means we choose not to recharge, i.e. browsing & chatting using phone.

Control input $u_t \in \mathcal{U}$.

Motion model:

$$x_{t+1} \sim P_f(\cdot \mid x_t, u_t) = \begin{cases} x_t, & \text{if } u_t = 1, \text{ with prob. } 1-q \\ 1, & \text{if } u_t = 1, \text{ with prob. } q \\ j, & \text{if } u_t = 0, \text{ with prob. } P(x_t, j) \end{cases}$$

Reward/Cost: $\ell(x_t = i, u_t = 0) = -r(i)$, $i \in [n]$

$$\ell(x_t = i, u_t = 1) = \begin{cases} qC, & \text{with prob. } q \\ 0, & \text{with prob. } 1-q \end{cases}$$

Then value function:

$$V_t(x_t) = \mathbb{E}_{x_{t+1:T}}\left[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} \ell(x_\tau, u_\tau) \mid x_t\right]$$

As $T \to \infty$, it becomes a discounted Infinite-horizon problem:

$$\min_{u \in \mathcal{U}} V(x) = \min_{u \in \mathcal{U}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \ell(x_t, u_t) \mid x_0 = x\right]$$

$$x_{t+1} \sim P_f(\cdot \mid x_t, u_t) = \begin{cases} P_f(x_{t+1} \mid x_t, u_t = 0) = P(x_t, x_{t+1}) \\ P_f(x_t \mid x_t, u_t = 1) = 1-q \\ P_f(1 \mid x_t, u_t = 1) = q \end{cases}$$

$$x_t \sim \mathcal{X} = \{1, 2, 3, \cdots, n\}$$

$$u_t \sim \mathcal{U} = \{1, 0\}$$

Related Bellman equation:

$$V^*(x) = \min_{u \in \mathcal{U}(x)} \{\ell(x, u) + \gamma \mathbb{E}_{x' \sim P_f(\cdot \mid x, u)}[V(x')]\} \quad \forall x \in \mathcal{X}$$

where $\ell(x, u)$ is defined above.

It can also be written in a compacted version:

$$V^*(x) = \min_{u \in U(x)} \{-r(x)(1-u) + qc\, u + \gamma \sum_{x' \in X} P_f(x'|x,u)\, V^*(x')\}$$

$$= \min_{u \in U(x)} \begin{cases} -r(x) + \gamma \sum_{x'=l}^{n} P(x,x')\, V^*(x') & \text{, if } u = 0,\ l \in [n] \\ qc + r[qV^*(1) + (1-q)V^*(x)] & \text{, if } u = 1 \end{cases}$$

(b) Use Value Iteration (VI) to prove $V^*(i)$ increases in i:

① Assume $t=0$, $V_0(i) = 0$, then $t=1$,

$$V_1^*(i) = \min_{u \in \{1,0\}} \{-r(i), qc\} = -r(i), \text{ since } r(i) > 0.$$

According to that $r(i)$ is decreasing as i increases.

$V_1^*(i) = -r(i)$ is increasing as i increases.

② When $t > 0$, suppose $V_t^*(i)$ increases as i increases, then we want to prove $V_{t+1}^*(i)$ also has the same property. (Mathematical Induction).

Let $Q_t(i, u) = V_t(i)$,

then $V_{t+1}^*(i) = \min\{Q_{t+1}(i, u=1), Q_{t+1}(i, u=0)\}$.

We want to prove

$$V_{t+1}^*(i+1) = \min\{Q_{t+1}(i+1, u=1), Q_{t+1}(i+1, u=0)\} < V_{t+1}^*(i) \quad (*)$$

If we can prove

$$Q_{t+1}(i+1, u=1) > Q_{t+1}(i, u=1)$$

and $Q_{t+1}(i+1, u=0) > Q_{t+1}(i, u=0)$,

then (*) is correct.

When $u=1$, based on VI:

$$V_{t+1}(i) = qc + \gamma[qV_t(1) + (1-q)V_t(i)]$$
$$V_{t+1}(i+1) = qc + \gamma[qV_t(1) + (1-q)V_t(i+1)].$$

By induction assumption, $V_t(i) < V_t(i+1)$, $1-q>0$

then $V_{t+1}(i+1) > V_{t+1}(i)$, i.e. $Q_{t+1}(i+1, u=1) > Q_{t+1}(i, u=1)$

When $u=0$, based on VI:

$$V_{t+1}(i) = -r(i) + \gamma \sum_{j=1}^{n} P(i,j) V_t(j).$$

we know $-r(i)$ is monotonously increasing over $i \in [n]$.

$$\sum_{j=1}^{n} P(i,j) V_t(j) = \underbrace{\left(\sum_{j=1}^{n} P(i,j)\right)}_{>0} V_t(1) + \sum_{j=2}^{n} \underbrace{\left(\sum_{l=j}^{n} P(i,l)\right)}_{\text{monotonously increasing}} \underbrace{(V_t(j) - V_t(j-1))}_{>0 \text{ (Induction assumption)}}$$

$\left(\sum_{j=1}^{n} P(i,j)\right)$: monotonously increasing (stochastic dominance)

Thus, $\sum_{j=1}^{n} P(i,j) V_t(j)$ is also monotonously increasing over $i \in [n]$.

Then $V_{t+1}(i)$ is monotonously increasing,

so $V_{t+1}(i) < V_{t+1}(i+1)$

Therefore, (*) holds and $V^*_{t+1}(i)$ increases in $i \in [n]$

Based on ① ②, the conclusion holds.

(c) $V^*(x)$

$$= \min_{u \in \mathcal{U}(x)} \begin{cases} -r(x) + \gamma \sum_{x'=1}^{n} P(x,x') V^*(x') & \text{if } u=0,\ l \in [n] \\ qc + \gamma [q V^*(1) + (1-q) V^*(x)] & \text{if } u=1 \end{cases}$$

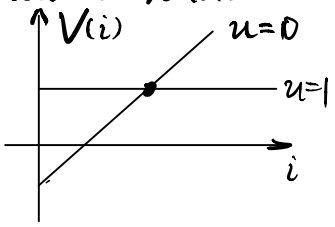
If $u=1$ is optimal, $V^*(i) = qc + \gamma [q V^*(1) + (1-q) V^*(i)]$,
then we can solve for $V^*(i)$, i.e. $V^*(i)$ is a constant. $V^*(i) = \dfrac{qc + \gamma q V^*(1)}{1 - \gamma(1-q)}$.

If $u=0$ is optimal policy,

$$V^*(i) = -r(i) + \gamma \sum_{j=1}^{n} P(i,j) V^*(j), \quad l \in [n]$$

is increasing in $i \in [n]$.

Thus, if $-r(i) + \gamma \sum_{j=1}^{n} P(i,j) V^*(j) - qc - \gamma [q V^*(1) + (1-q) V^*(i)]$ exists zero-point, then the thresh nature holds.



Otherwise, $t=1$ or n is the threshold.

Report for Problem 4

Yiran Xu

Collaborator: Shuo Xu, we discussed how to choose heuristic functions.

1 Introduction

Planning for motion is one of the most important areas of robotics. In order to make decision automatically, people find out various algorithms to plan as well as possible. Usually, the environment and robot itself can be formalized, e.g in the framework of Markov Decision Processes (MDP), which includes complex stochastic processes, just like in the real world.

Path planning is a basic planning problem. It requires robot, or an *agent*, to have the ability to find a rather short path to the goal. The algorithms for this problem start from MDP typically and there are two main families of methods for it. One is *Search-based* algorithms, e.g. A* algorithm and its variant versions; the other one is *Sampling-based* algorithms, such as Probabilistic Roadmap (PRM), Rapidly Exploring Random tree (RRT) and their modified algorithms.

Search-based methods are trying to explore the state space explicitly until reach the goal. One of the most famous algorithms is A* algorithm. A* searches the space relying on heuristic function, which is a lower bound on the optimal cost moving between nodes. Sometimes, in order to plan in large environments and to make decisions in limited time, some real-time A* algorithms are necessary. For large environments, agent-centered search brings us the most representative A*-based algorithms - *Learning Real-Time A** (LRTA*) and *Real-Time Adaptive A** (RTAA*). Those two methods both update heuristics for nodes during the search process. LRTA* updates heuristics by Dynamic Programming (DP) while RTAA* relies on the optimal cost f, which is faster than LRTA*. In this project, considering the time limit in every move, RTAA* is implemented as a representative of A*-based algorithms.

Search-based planning explores the space explicitly, and it usually discretizes the space. However, sampling-based planning conducts a search that probes the search the space in a sampling scheme. One of the mose popular techniques is RRT-based planning. It constructs a tree connecting the root and the goal by randomly sampling. Unlike PRM it does not need pre-processing. Its modified

versions include RRT-Connect, RRT*, etc. RRT-Connect takes the advantage of dealing with "narrow channel" while RRT* extends and smooths the tree simultaneously. In this project, given that there are several cases including narrow paths, therefore RRT-Connect is implemented.

2 Problem Formulation

The whole space can be defined as a Configuration Space (C-space). C-space will include all possible robot configurations, i.e. all possible positions in this case. Particularly, C-space can be divided into 2 parts.

- Free space C_{free}, containing all free positions.
- Occupied space C_{obs}, involving all obstacles in this case.

Usually, facing a large unknown environment, we assume unknown space is free.

For robot, we consider it as a 3D point and use MDP to model it.

- State space $\mathcal{X}$: $\mathbf{x}_t \in \mathcal{X} \subset \mathbb{R}^3$, which is the position of the robot in C.
- Control space $\mathcal{U}$: In this case, the control input $\mathbf{u}_t$ is where we decide to move at t. Then $\mathbf{u}_t = \mathbf{x}_{t+1}$.
- Motion Model: $\mathbf{x}_{t+1} = \mathbf{u}_t$.
- Cost: $l(\mathbf{x}_t, \mathbf{u}_t) = Path(\mathbf{x}_t, \mathbf{x}_{t+1})$. It is the length of path from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$.

Particularly, this problem is actually a real-time planning problem. In problem setting, we have to decide every move in 2 seconds, or the robot will move randomly, which is not what we wish. Therefore, the logic of whole processing is shown in Fig. 1.

After planning, we need to:
a) on a single planning episode, compare the quality of path in term of the effect of different parameters;
b) on the overall task, compare the ability to reach the goal and the length of the paths between search-based algorithm and sampling-based algorithm.

3 Technical Approach

3.1 Search-Based Planning

RTAA* algorithm is an agent-centered search algorithm. It is founded based on A* algorithm to expand the search area. The Algorithm 3.1 states the basic framework of RTAA* and Algorithm 3.1 is A* with *lookahead* times while-loop. The variables are listed as below.

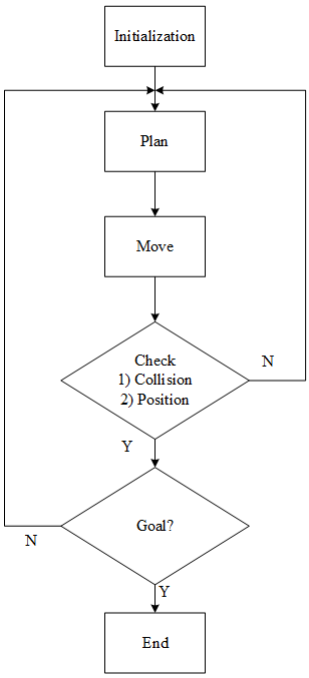


Figure 1: Overall Logic of Motion Planning

- $GOAL$: Goal state.
- $children(s)$: Children nodes of s. Children are adjacent nodes of s.
- $Parent(s)$: Parent node of s.
- $lookahead$: The times of expansion.
- h: Heuristic value.
- g: Label of cost.
- $\mathcal{V}$: A set of all nodes.

As we can see from Algorithm 3.1, RTAA* allows we search and move simultaneously. It also updates h-value to avoid the possibility that robot gets stuck in the corner. RTAA* also keeps the consistency and admissibility of heuristics. By this plan, we can somehow reach to our goal. Compared to LRTA*, RTAA* updates h-values faster which is suitable for this problem, i.e. plan and see more in limited time.

Algorithm 1 RTAA* Algorithm [1]

Result: $SUCCESS$ or $FAILURE$
while $GOAL \notin children(s_{curr})$ **do**
  $lookahead$:= the number of times for expansion
  A*($lookahead$)
  if $j^* = FALIURE$ **then**
    Return $FAILURE$
  end
  for $i \in CLOSED$ **do**
    $h_i = f_{j^*} - g_i$
  end
  Move to $j^* = argmin_{j \in OPEN}(g_j + h_j)$
  $s_{curr} = j^*$
end
Return $SUCCESS$

Algorithm 2 A* Algorithm

Result: j^*
$OPEN \leftarrow \{s\}, CLOSED \leftarrow \{\}$
$g_s = 0, g_i = \infty$ for all $i \in \mathcal{V} \backslash \{s\}$
for n = *1,...,lookahead* **do**
  if $OPEN$ *is empty* **then**
    Return $FAILURE$
  end
  Remove node i with the smallest $f_i = g_i + h_i$ from $OPEN$
  Insert i into $CLOSED$
  for $j \in Children(i)$ *and* $j \in CLOSED$ **do**
    if $g_j > g_i + Cost(i, j)$ **then**
      $g_j \leftarrow g_i + Cost(i, j)$
      $Parent(j) \leftarrow i$
      Insert j into $OPEN$
    end
  end
end
Return $j^* = argmin_{j \in OPEN}(g_j + h_j)$

3.2 Sampling-based Planning

Sampling-based planning is considering to establish a tree that connects the start point and the goal by randomly sampling scheme. The most popular one is RRT algorithm because it is simple, economical and with big possibility for extension. However, it also contains problems. For instance, *Bug Traps*. *Bug Traps* always happens when there exists narrow path on the way to goal. Due to the randomly sampling, we cannot guarantee that the tree grows quickly from the trap.

In order to improve the performance against *Bug Traps*, Bi-directional RRT is proposed. This way grows two trees, one root is the start point, the other one is the goal. Two trees will grow in two directions until they connect to each other. RRT-Connect is derived from this idea. The details of RRT-Connect is listed in Algorithm 3.2. Also, in order to search the tree more quickly, a data structure of tree is organized.

Some useful functions are listed as belows.

- SampleFree(): Returns i.i.d samples from free space C_{free}.
- Nearest(): Given a graph $G = (V, E)$ with $V \subset C$ and a point $x \in C$, returns a vertex $v \in V$ that is closed to x, i.e. Nearest((V, E), x) := $argmin_{v \in V} ||x - v||$.
- Steer(): Returns Steer$(x, y, \epsilon) := argmin_{z:||z-x|| \leq \epsilon} ||z - y||$.
- CollisionFree(): Given point $x, y \in C$, returns True if the line segment between x and y lies in C_{free} and False otherwise.
- Path(G_1, G_2): Returns the path between two graphs.
- Swap(G_1, G_2): Swaps the vertices and edges of two graphs.

For RRT-family algorithms, however, it is unwise to move before the roadmap is done. Hence, the strategy for moving is let the robot stay at the start point before the roadmap is completed. Once the map completed, let the robot move based on the constraint conditions.

3.3 Collision Checker

In last section, CollisionFree() is used for check if the segment of two points will intersect with blocks.

The line in 3D parameterized by two points is

$$\begin{aligned} \mathbf{l} &= t\mathbf{X}_1 + (1-t)\mathbf{X}_2 \\ &= (t(x_1 - x_2) + x_2, t(y_1 - y_2) + y_2, t(z_1 - z_2) + z_2)^T, \end{aligned} \tag{1}$$

where $\mathbf{l}$, $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^3, 0 \leq t \leq 1$ The plane's function can be expressed by

$$Ax + By + Cz + D = 0, \tag{2}$$

where $A, B, C, D, x, y, z \in \mathbb{R}$. $(A, B, C)^T$ is actually the normal vector of the plane, it can be computed by taking outer production of two lines in the plane. After that, plug any point in the plane into Eq. 2 we can compute D. Then plug Eq. 1 into Eq. 2 we can solve for t:

$$t = -\frac{Ax_2 + By_2 + Cz_2 + D}{A(x_1 - x_2) + B(y_1 - y_2) + C(z_1 - z_2)} \tag{3}$$

Algorithm 3 RRT-Connect

Result: $(V_a, E_a), (V_b, E_b)$
$V_a \leftarrow x_s, E_a \leftarrow \emptyset, V_b \leftarrow x_\tau, E_b \leftarrow \emptyset$
while *True* **do**
  $x_{rand} \leftarrow$ SAMPLEFREE()
  if *not* EXTEND((V_a, E_a),x_{rand}) = TRAPPED **then**
    if CONNECT((V_b, E_b), x_{new}) = REACHED **then**
      Return PATH((V_a, E_a),(V_b, E_b))
    end
  end
  SWAP((V_a, E_a),(V_b, E_b))
end
function EXTEND((V, E),x)
$x_{nearest} \leftarrow$ NEAREST((V,E), x)
$x_{new} \leftarrow$ STEER($x_{nearest}$, x, ϵ)
if COLLISIONFREE($x_{nearest}$, x) **then**
  $V \leftarrow \{x_{new}\}$, $E \leftarrow \{(x_{nearest}, x_{new}),(x_{new}, x_{nearest})\}$
  if $x_{new} = x$ **then**
    Return REACHED **else Return** ADVANCED
  end
end
function CONNECT((V, E),x)
repeat status $\leftarrow$ EXTEND((V,E),x) **until** status $\neq$ ADVANCED
Return status

Then substitute Eq. 3 back into (1) we can get the intersected point.
Finally, by checking whether the point solved above is inside the planes of the block, we can know whether the robot collicides or not.

4 Results

4.1 Experiment Description

There are totally 7 maps to test our algorithms. Those maps contain boundaries and blocks. We must make sure in every move, the robot will not move beyond the boundaries or collide with blocks. Also, the robot cannot move beyond 1 at each step, i.e. $||\mathbf{x}_{t+1} - \mathbf{x}_t||_2 \leq 1$. Finally, the next move should be decided in 2 seconds, or the robot will move randomly. Our goal is to let the robot reach the goal by search-based planning and sampling-based planning, respectively.

After the experiment, we need to:
a) on a single planning episode, compare the quality of path in term of the effect of different parameters;
b) on the overall task, compare the ability to reach the goal and the length of the paths between search-based algorithm and sampling-based algorithm.

4.2 Results and Discussion

The qualitative results of 7 maps are shown in Fig. 3 and Fig. 4. Both algorithms can reach the goal for all maps.

For quantitative analysis, we should examine two aspects.

First, we need to compare the performance within search-based planning and sampling-based planning in term of different parameters, respectively. The main hyper-parameters for RTTA* include: the metrics of the discrete grid (scale), the number of expansion (N) and heuristic functions, where scale is the ratio to 1 in the space, e.g. if scale = 0.5, then each grid in the space is $0.5 \times 0.5 \times 0.5$. The hyper-parameter for RRT-Connect is steer radius ϵ.

The results are listed in the Table 1[1] and Table 2. The order from 1 to 7 represents 7 maps: Single Cube, Maze, Flappy Bird, Monza, Window, Tower, Room. The results of RRT-Connect are average lengths of paths, based on 5 experiments with different random seeds (np.random.seed(1) to np.random.seed(5)).

Table 1: Results of the Length of Paths, RTAA*[1]

Scale	N	Heuristics	1	2	3	4	5	6	7
0.5	1	l_1-norm	8.3251	>30000	1147.9030	-	34.2987	502.8719	47.4808
0.5	1	l_2-norm	8.3249	30346.52	337.8665	31420.3	31.2151	**62.7782**	62.3480
0.5	1	inf-norm	8.4817	>30000	45.0372	>30000	55.7462	127.1924	70.0246
0.5	1	$-inf$-norm	614.20	**807.3675**	1477.7721	**358.9594**	1339.039	2208.0512	311.8856
0.5	1	Octal	8.4382	>30000	46.7426	>30000	30.1222	136.1442	45.3640
0.5	5	l_1-norm	**8.2094**	>30000	960.6027	-	35.1483	387.1070	49.8720
0.5	5	l_2-norm	8.8355	24678.06	174.1030	>30000	30.2787	64.2726	92.1315
0.5	5	inf-norm	11.1668	>30000	44.8714	>30000	42.0403	147.0370	92.6885
0.5	5	$-inf$-norm	32.0913	910.7741	145.3975	715.8195	209.9345	882.5951	191.1925
0.5	5	Octal	11.1668	>30000	47.1024	>30000	31.4323	109.0633	39.9640
0.9	1	l_1-norm	8.3804	>30000	-	-	31.6701	-	47.2648
0.9	1	l_2-norm	8.3804	-	303.2074	>30000	32.0823	72.8194	**30.0611**
0.9	1	inf-norm	8.6940	>30000	**42.0333**	>30000	44.4817	118.5230	118.7198
0.9	1	$-inf$-norm	8.6940	1258.29	-	964.4501	1976.598	-	210.2364
0.9	1	Octal	8.6940	>30000	43.3095	>30000	31.9444	103.0686	37.7740
0.9	5	l_1-norm	8.6551	>30000	-	-	**28.9392**	-	44.7347
0.9	5	l_2-norm	9.9669	-	188.8487	>30000	35.7853	72.3276	39.0079
0.9	5	inf-norm	12.8614	>30000	155.4739	>30000	63.8958	87.8827	76.0573
0.9	5	$-inf$-norm	335.10	1597.18	-	1054.238	224.8918	-	442.4529
0.9	5	Octal	15.1058	>30000	140.7012	>30000	45.3569	97.2870	41.8979

[1]"-" means failure and > means running time is too long but the length is greater than some number.

Table 2: Results of the Length of Paths, RRT-Connect

Steer Radius	1	2	3	4	5	6	7
$\epsilon = 0.3$	**8.2375**	131.3546	42.4986	**101.9076**	28.2900	45.1499	23.0702
$\epsilon = 0.5$	8.4220	132.1002	46.9855	102.1912	29.0867	47.5105	21.4823
$\epsilon = 0.7$	8.4834	**125.2153**	**36.6691**	104.7145	**28.2002**	**41.3808**	**18.5029**
$\epsilon = 1.0$	9.1798	131.2050	41.5117	102.5778	30.7766	46.3960	23.0702
$\epsilon = 2.0$	10.8367	125.7608	45.5462	105.3413	30.5433	49.7928	19.8283

From Table 1, we can see the final lengths are pretty different over the heuristic function. In some cases, some lengths are too long under the guidance of several heuristic functions. Heuristic function is a very important hyper-parameters for RTAA* algorithm. Take 2 (Maze) and 4 (Monza) for example, those two maps both have narrow channels. If we choose improper heuristics, the robot will have a big chance getting stuck in the corners. This is because we waste most of the time to search in the z-axis. However, for those two maps, the most significant search should focus on x axis and y axis (See Fig. 3(b) and Fig. 3(d)). It is useless to move along the z-axis. Given this, it is intuitive to consider using -inf-norm, which takes the minimum absolute value of the vector. By using -inf-norm, the oscillation along the z-axis becomes little as Fig. 2 shows. Therefore the final length and the time for searching will decrease significantly.

For scale, i.e. the size of the discrete grids, it will somehow have impact on the final length, but it will not change it significantly. However, in some cases, if the scale is big but N is not big enough, the robot will get stuck in several points (See - in Table 1). This is because the algorithm has found out all discrete points in nearby area. To improve this, we need to increase N. Plus, for some narrow setting (e.g. Maze), using a big scale is not recommended because they will be recognized as collision. This happens in Maze as scale = 0.9.

For N, i.e. the times of expansion, it is supposed to be pretty large to expand more areas technically. However, due to the limit of PC, if $N > 25$ in my case, the planning time will excess 2 seconds. Thus here we only discuss the situations that N is not big. When N is small, the effect of it becomes trivial since 3D space is really large. Hence, larger N, e.g. $N > 50$ is needed for 3D space search.

From Table 2, we can see the average lengths over different steer radius do not vary a lot. The minimum lengths also distribute in different steer radii. However, increasing the steer radius will speed up the whole processing in general because the area for connecting is bigger if we use a larger radius.

After examining the effect of parameters, we will compare the performance between RTAA* and RRT-Connect. However, due to different strategies, comparing the final length of the paths seems unwise since RTAA* is a real-time

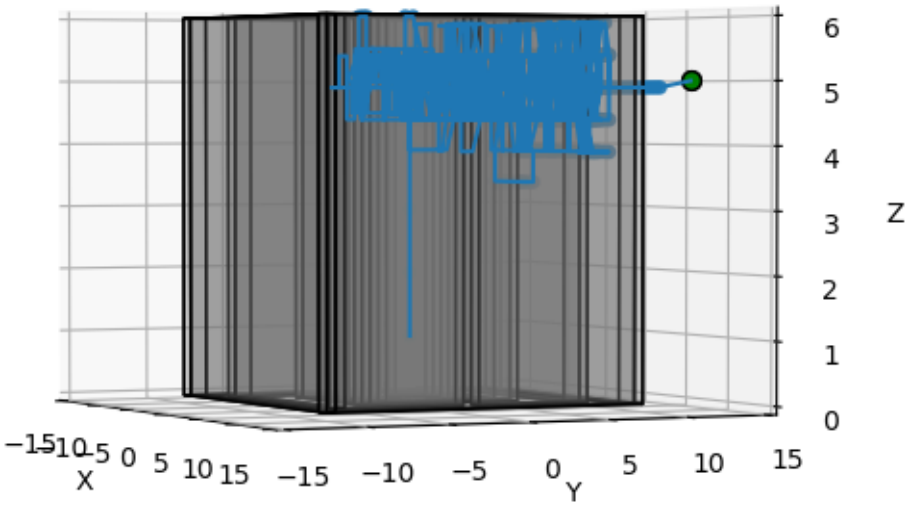
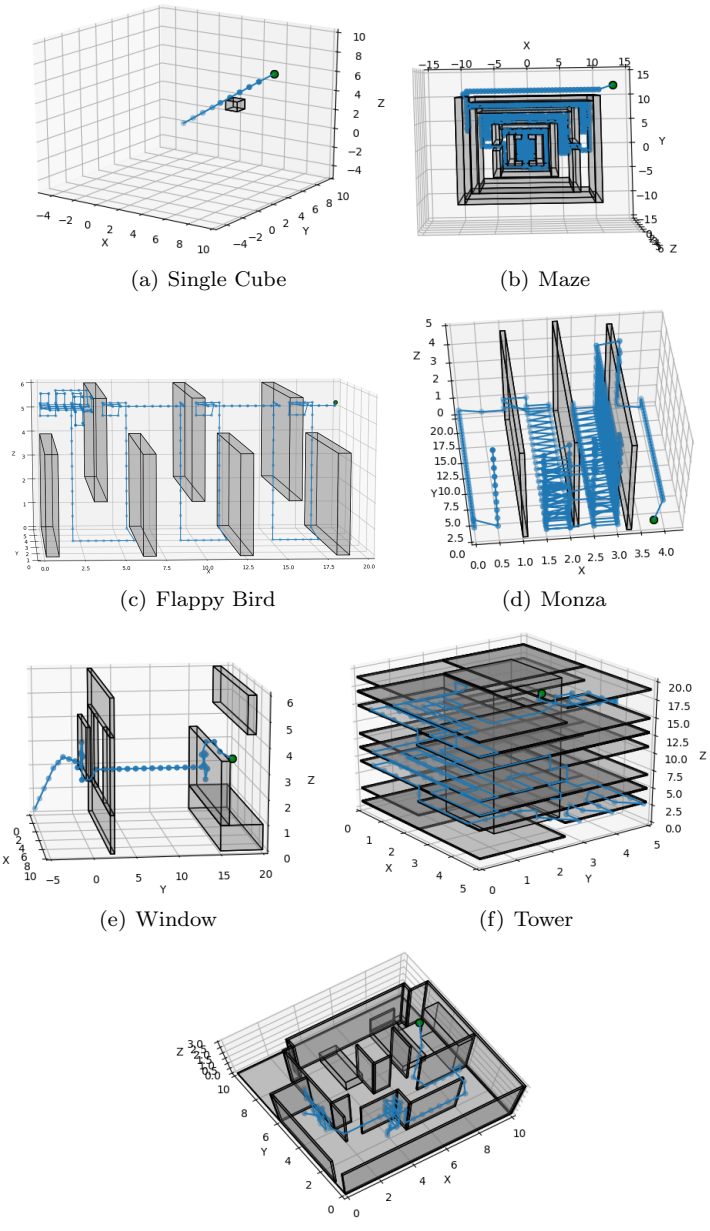


Figure 2: Maze RTAA* with $-inf$-norm, oscillation is around the goal's z-value

algorithm. Thus, we will focus on the ability to reach the goal. The average time of RRT-Connect is much longer than of RTAA* if the map includes many obstacles and tortuous paths (e.g. Maze and Monza). However, for other cases, RRT-Connect performs slightly better than RTAA* in the running time because this cases are much simple. If we can find a good heuristic function, RTAA* perhaps will be faster than RRT in complex maps. In conclusions, if the map is simple, without too many obstacles or twist paths, and time for planning is enough, RRT-Connect is a better choice. Nevertheless, we need to choose RTAA* for saving time but sacrifice the length.

In general, RTAA* is suitable for large unknown environment and it can plan in limited time, but it might push the robot into worse situations if we don't have enough information of the environment. Also, we need to consider different heuristic functions for different maps. Finally, in order to use larger N, some parts of the algorithm can be improved. For instance, CollisionFree could be a time-consuming part, now it calculates all the intersections of line segment with planes of the obstacles. We could speed up this by using interpolation instead.

RRT-Connect is good at planning in simple maps and it is able to generate shorter path than RTAA*. It can also deal with some *Bug Traps* situations. However, it is not suitable for online search. Plus, its dependence on random sampling cannot always yield the optimal solution. In order to improve it, we can introduce smoothing techniques.



(a) Single Cube

(b) Maze

(c) Flappy Bird

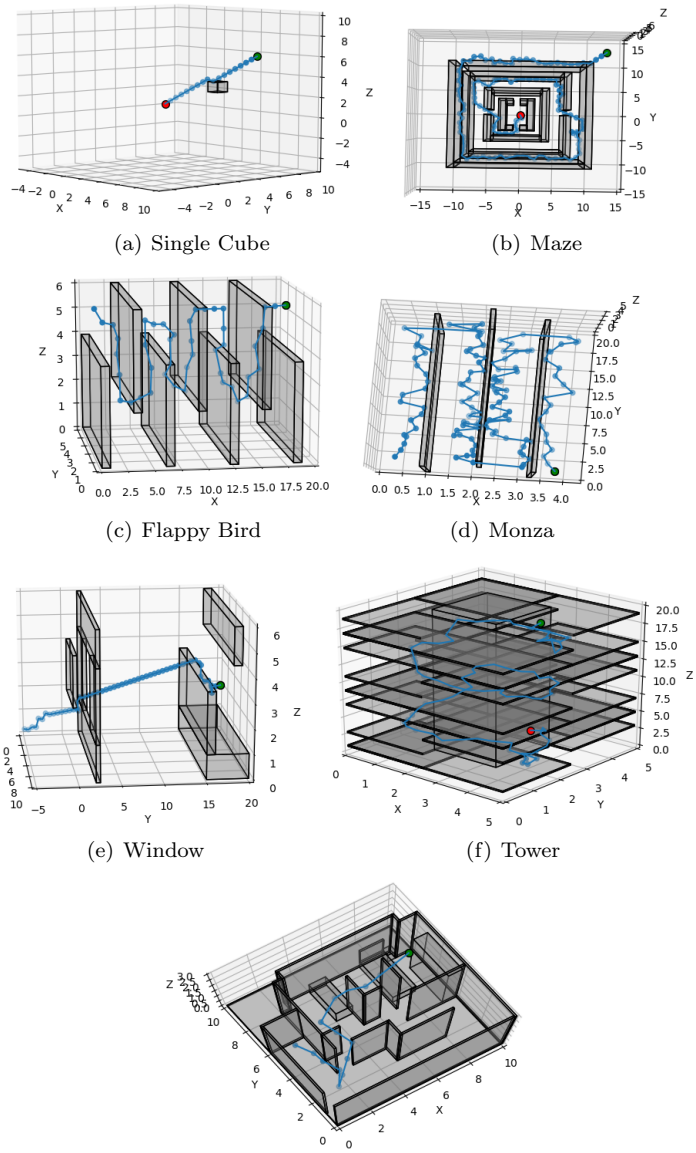
(d) Monza

(e) Window

(f) Tower

(g) Room

Figure 3: Qualitative Results, RTAA*



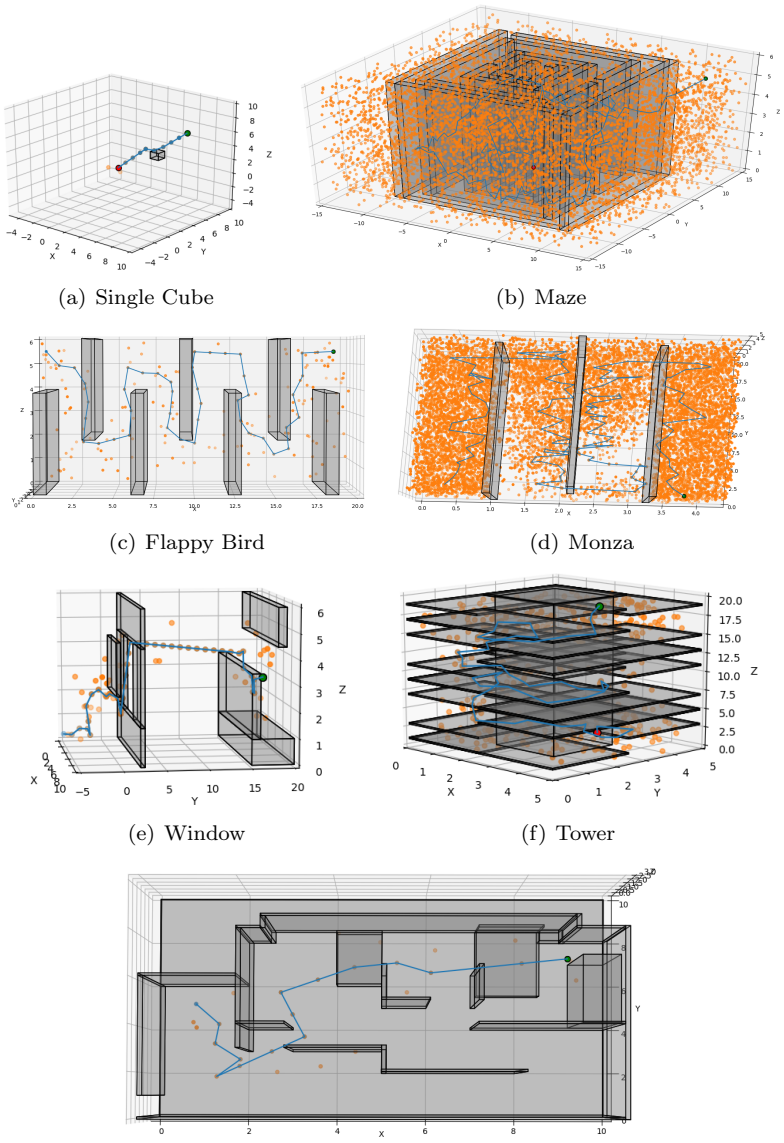
(a) Single Cube (b) Maze

(c) Flappy Bird (d) Monza

(e) Window (f) Tower

(g) Room

Figure 4: Qualitative Results, RRT-Connect



(a) Single Cube

(b) Maze

(c) Flappy Bird

(d) Monza

(e) Window

(f) Tower

(g) Room

Figure 5: Qualitative Results, RRT-Connect, with nodes (Orange)

References

[1] Koenig, S., & Likhachev, M. (2006, May). Real-time adaptive A. In Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (pp. 281-288). ACM.