HW2: Motion Planning

Shuo Xu
Department of Computer Science and Engineering
s3xu@ucsd.edu

Abstract

Acknowledgement

Special thanks to Ehsan Zobeid, Yu ShengDing, Yiran Xu, Zheng Liu for their hints on problem 3.

1 Problem 1

1.1 (a)

We firstly formulate the DFS problem.

$x_t \in \mathcal{X}$ is the state at time t. u_t is the control at time t. $x_{t+1} = f(x_t, u_t), t = 0, \ldots, T-1$ defines the state transition given control from time t to t + 1. $\ell_t(x_t, u_t)$ is the stage cost, and $\mathfrak{q}(x_T)$ is the terminal cost.

This is equivalent to a DSP problem. The states can be represented by a vertices set in the graph as

$$\mathcal{V} = (\bigcup_{t=0}^{T} \{(t, x_t) | x_t \in \mathcal{X}\}) \cup \{\tau\}$$

The cost can be defind as

$$\mathcal{C} := \left\{ ((t, x_t), (t+1, x_{t+1}), c) \,|c = \min_{u \in \mathcal{U}(x_t)} \quad \ell_t(x_t, u) \right\} \bigcup \{((T, x_T), \tau, \mathfrak{q}(x_T))\}$$

The optimization task can be solved by Dijkstra algorithm. Notice, the path is searched in the backward order, where we start from τ and ends at x_0.

In the worst case without early stopping, the time complexity of dynamic programming algorithm is $O(|\mathcal{V}|^3)$.

For Dijkstra algorithm, the time complexity depends on its implementation, the simplest implementation of Dijkstra's algorithm stores the vertex set Q as an ordinary linked list or array, and extract-minimum is simply a linear search through all vertices in $queue$. In this case, the running time is $O(|E| + |\mathcal{V}|^2) = O(|\mathcal{V}|^2)$.

Thus, the Dijkstra algorithm runs more efficiently compared to DP algorithm. However, in terms of the number of investigated nodes, Dijkstra algorithm is no worse than DP algorithm with an early stopping. It saves time by avoiding investigating non-neighboured nodes at each time step.

1.2 (b)

Yes.

Algorithm 1 Dijkstra algorithm

procedure DIJKSTRA(start=τ, end=x_0, distance)
  $parents \leftarrow$ {node : node for node in graph}
  $queue \leftarrow$all nodes in graph
  while $queue$ is not empty **do**
    $prev\_node \leftarrow$ node in queue with smallest distance to start node
    for each neighbour n of prev_node **do**
      new_dis = distance[start][prev_node] + distance[prev_node][n]
      if new_dis < distance[start][n] **then**
        $distance[start][n] \leftarrow new\_dis$
        $parents[n] \leftarrow prev\_node$
  $shortest\_path \leftarrow [target]$
  while start != target **do**
    $target \leftarrow parents[target]$
    $shortest\_path.append(target)$
  return $shortest\_path$

The heuristic function $h((t, x_t))$ means the lower bound on the optimal cost to get from start node x_0 to x_t.

Let's assume $h((t, x_t)) = t$. We can develop the weighted A^* algorithm as

Algorithm 2 A^* algorithm

procedure A^*(start=τ, end=x_0)
  $OPEN \leftarrow \{s\}, CLOSED \leftarrow \{\}, \eta \geq 1$
  $g_s = 0, g_i = \infty$ for all $i \in \mathcal{V} \backslash \{s\}$
  while $\tau \notin CLOSED$ **do**
    Remove i with smallest $f_i := g_i + \epsilon h_i$
    Insert i into CLOSED
    for $j \in$ Children (i) and $j \notin CLOSED$ **do**
      if $g_j > (g_i + c_{ij})$ **then**
        $g_j \leftarrow (g_i + c_{ij})$
        Parent $(j) \leftarrow i$
        Insert j into OPEN
  $shortest\_path \leftarrow [target]$
  while start != target **do**
    $target \leftarrow parents[target]$
    $shortest\_path.append(target)$
  return $shortest\_path$

2 Problem 2

2.1 (a)

$$\begin{aligned} h\left(\mathrm{x}_\tau\right) &= \max \left\{ h^{(1)}(x_\tau), h^{(2)}(x_\tau) \right\} \\ &= \max \{0, 0\} \\ &= 0 \end{aligned}$$

$$
\begin{aligned}
h(x_i) &= \max\left\{h^{(1)}(x_i), h^{(2)}(x_i)\right\} \\
&\leq \max\left\{c\left(\mathbf{x}_i, \mathbf{x}_j\right) + h^{(1)}\left(\mathbf{x}_j\right), c\left(\mathbf{x}_i, \mathbf{x}_j\right) + h^{(2)}\left(\mathbf{x}_j\right)\right\} \\
&= c\left(\mathbf{x}_i, \mathbf{x}_j\right) + \max\left\{h^{(1)}\left(\mathbf{x}_j\right), h^{(2)}\left(\mathbf{x}_j\right)\right\} \\
&= c\left(\mathbf{x}_i, \mathbf{x}_j\right) + h(x_j)
\end{aligned}
$$

Thus h is consistent.

2.2 (b)

$$
\begin{aligned}
h\left(\mathrm{x}_\tau\right) &= h^{(1)}(x_\tau) + h^{(2)}(x_\tau) \\
&= 0 + 0 \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
h\left(\mathrm{x}_i\right) &= h^{(1)}\left(\mathrm{x}_i\right) + h^{(2)}\left(\mathrm{x}_i\right) \\
&\leq c\left(\mathrm{x}_i, \mathrm{x}_j\right) + h^{(1)}\left(\mathrm{x}_j\right) + c\left(\mathrm{x}_i, \mathrm{x}_j\right) + h^{(2)}\left(\mathrm{x}_j\right) \\
&\leq 2c\left(\mathrm{x}_i, \mathrm{x}_j\right) + h(x_j)
\end{aligned}
$$

Let $\epsilon = 2$. Thus h is ϵ -consistent.

2.3 (c)

Suppose the shortest path from any node i to terminal node τ is $P_{shortest} = \{p_1 = x_i, p_2, \cdots, p_m = x_\tau\}$.

$$
\begin{aligned}
h\left(\mathbf{x}_i\right) &= h\left(\mathbf{p}_1\right) \\
&\leq c\left(\mathbf{p}_1, \mathbf{p}_2\right) + h\left(\mathbf{p}_2\right) \\
&\leq c\left(\mathbf{p}_1, \mathbf{p}_2\right) + c\left(\mathbf{p}_2, \mathbf{p}_3\right) + h\left(\mathbf{p}_3\right) \\
&\leq \sum_{i=1}^{m-1} c\left(\mathbf{p}_i, \mathbf{p}_{i+1}\right) + h\left(\mathbf{x}_\tau\right) \\
&= \sum_{i=1}^{m-1} c\left(\mathbf{p}_i, \mathbf{p}_{i+1}\right) \\
&= \textbf{dist}\left(\mathrm{x}_i, \mathrm{x}_\tau\right)
\end{aligned}
$$

Thus, if h is consistent, then it is also admissible.

2.4 (d)

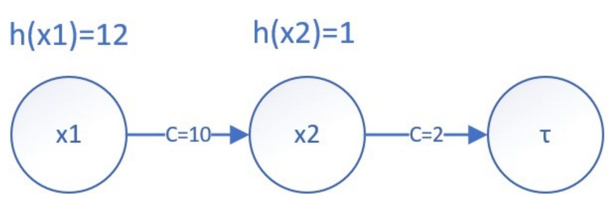

Figure 1: Admissible heuristic h that is not consistent

In the graph above, It admissible because $h(x_1) \leq dist(x_1, \tau) = 12$ and $h(x_2) \leq dist(x_2, \tau) = 2$.

However, it's not consistent because $h(x_1) = 12 > c(x_1, x_2) + h(x_2) = 11$.

2.5 (e)

Let $c_{ij} > 0$ for $i, j \in \mathcal{V}$ and h be a consistent heuristic. Assume all previously expanded states in CLOSED have correct g-values. Let the next state to expand be i with $f_i := g_i + h_i \leq f_j$ for all $j \in OPEN$. Suppose that g_i is incorrect, i.e. larger than the actual least cost from s to i. Then, there must be at least one state j on an optimal path from s to i such that $j \in OPEN$ but $j \notin CLOSED$ so that $f_j \geq f_i$. But this leads to a contradiction:

$$f_i = g_i + h_i > V^*_{s,i} + h_i = g_j + V^*_{ji} + h_i \geq g_j + h_j = f_j$$

So if the heuristic function used in A^* is consistent, then A^* will not re-open nodes.

2.6 (f)

It's not consistent not admissible.

Proof

Suppose $x_1 = 0.001, x_2 = -1, x_\tau = 0$.

Then

$$\begin{gathered} h(x_1) = 0.001 + 0.4 \times 0.001 = 0.0014, \\ h(x_2) = -1 - 0.4 \times 1 = -1.4, \\ c(x_1, x_2) = (0.001 + 1)^2 = 1.002001 \end{gathered}$$

Thus,

$$h(x_1) = 0.0014 \geq h(x_2) + c(x_1, x_2) = -0.397999$$

So the heuristic function is not consistent.

Since

$$h(x_1) = 0.0014 > dist(x_1, x_\tau) = 0.001,$$

the heuristic function is also not admissible.

3 Problem 3

3.1 (a)

The problem can be formulated as following.

Suppose the cell phone's battery x can take on the value from $\mathcal{X} = 1, 2, \ldots, n$. The choice of whether to charge or not given current state is denoted as $\pi(x) \in \mathcal{U} = \{0, 1\}$, where 1 means recharge and 0 means not.

The transition model is defined as below.

$$p_f\left(x_{t+1} | x_t, \pi\left(x_t\right)\right) = \begin{cases} q & x_{t+1} = 1, \pi(x_t) = 1 \\ 1 - q & x_{t+1} = x_t, \pi(x_t) = 1 \\ P(i, j) & x_t = i, x_{t+1} = j, \pi(x_t) = 0 \\ 0 & else \end{cases}$$

where $P(i, j)$ describes the probability of transitioning from state i to state j.

The stage cost is defined as

$$\begin{aligned} \ell(x_t, \pi(x_t)) &= \pi(x_t)[qc + (1 - q) \times 0] + (1 - \pi(x_t))[-\sum_{x_{t+1}} P(x_t, x_{t+1}) r(x_t)] \\ &= \pi(x_t) qc - (1 - \pi(x_t)) \sum_{x_{t+1}} P(x_t, x_{t+1}) r(x_t) \\ &= \pi(x_t) qc - (1 - \pi(x_t)) r(x_t) \sum_{x_{t+1}} P(x_t, x_{t+1}) \\ &= \pi(x_t) qc - (1 - \pi(x_t)) r(x_t) \end{aligned}$$

The discounting problem is formulated as

$$V^*(x) = \min_\pi V^\pi(x) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \ell\left(x_t, \pi\left(x_t\right)\right) | x_0 = x\right]$$
$$s.t. \quad x_{t+1} \sim p_f\left(\cdot | x_t, \pi\left(x_t\right)\right)$$
$$x_t \in \mathcal{X}$$
$$\pi\left(x_t\right) \in \mathcal{U}$$

Substitute the stage cost and transition model into the discounting equation into the Bellman equation, we have

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

3.2 (b)

The claim can be proved using induction. I will try to prove that in value iteration, V_{t+1} is increasing if V_t is increasing.

Since value iteration will get the optimal value which is irrelavant to the initalization of $V_0(x)$, we can assign specific numbers to $V_0(i)$ that is increasing in i.

Suppose we have all the $V_t(i)$ that is increasing in i, then we will prove $V_{t+1}(i)$ is also increasing in i. According to the value iteration algorithm, we have

$$V_{t+1}(x) = \begin{cases} -r(x) + \gamma \sum_{x' \in \mathcal{X}} P(x, x')V_t(x') & u = 0 \\ qc + \gamma[qV_t(1) + (1-q)V_t(x)] & u = 1 \end{cases}$$

If $a < b$ and $c < d$, it's guaranteed that $min(a, b) < min(c, d)$, so we only need to prove that $Q_t(i, 0) \geq Q_t(i-1, 0)$ and $Q_t(i, 1) < Q_t(i-1, 1)$, then we will get $V_t(i) \geq V_t(i-1)$ because $V_t(i) = min[Q_t(i, 0), Q_t(i, 1)]$.

$$V_{t+1}(x) = \min_{u \in \mathcal{U}(x)}\left[\ell(x, u) + \gamma \sum_{x' \in \mathcal{X}} p_f\left(x' | x, u\right) V_t\left(x'\right)\right], \quad \forall x \in \mathcal{X}$$

. This means we only need to prove that $V_{t+1}(x)$ is increasing when $u = 0$ and $u = 1$, separately.

When $u = 1$, since $(1-q) \geq 0$, $\gamma > 0$ and $V_t(x)$ is increasing, $V_{t+1}(x)$ is also monotonously not decreasing.

When $u = 0$, we have

$$V_{t+1}(x) = -r(x) + \gamma \sum_{x' \in \mathcal{X}} P\left(x, x'\right) V_t\left(x'\right)$$

Since $-r(x)$ is increasing in i, we only need to prove $\sum_{x' \in \mathcal{X}} P\left(x, x'\right) V_t\left(x'\right)$ is also increasing. We abbreviated $P\left(x, x'\right)$ as $P_{x,x'}$

$$\sum_{x' \in \mathcal{X}} P_{x,x'} V_t\left(x'\right) = \left(\sum_{x'=1}^{n} P_{xx'}\right) V_t(1) + \sum_{x'=2}^{n} \left(\sum_{l=x'}^{n} P_{xl}\right) \left(V_t(x') - V_t(x'-1)\right)$$

Since $V_t(x') - V_t(x'-1) > 0$, and $\sum_{l=x'}^{n} P_{xl}$ is increasing in x, it's obvious $\sum_{x'=2}^{n} \left(\sum_{l=x'}^{n} P_{xl}\right) \left(V_t\left(x'\right) - V_t\left(x'-1\right)\right)$ also increase in i. So $\sum_{x' \in \mathcal{X}} P_{x,x'} V_t\left(x'\right)$ is increasing in x.

Thus, we've proved $V_{t+1}(x)$ is also increasing in i.

By induction, we've proved that $V_t(x)$ is increasing in i.

3.3 (c)

According to

$$V^{*}(x)=\min_{u\in\mathcal{U}(x)}\begin{cases} qc+\gamma\left(qV^{*}(1)+(1-q)V^{*}(x)\right) & u=1\\ -r(x)+\gamma\sum_{x'\in\mathcal{X}}P\left(x,x'\right)V^{*}\left(x'\right) & u=0\end{cases}$$

we noticed that when $u=1$, $V^{*}(x)=qc+\gamma\left(qV^{*}(1)+(1-q)V^{*}(x)\right)$, thus

$$V^{*}(x)=\frac{1}{1-\gamma+\gamma q}(qc+\gamma qv^{*}(1))$$

which is a constant irrelevant with x.

Thus we have

$$Q^{*}(x,u)=\begin{cases}-r(x)+\gamma\sum_{x'\in\mathcal{X}}P\left(x,x'\right)V^{*}\left(x'\right) & u=0\\ \frac{1}{1-\gamma+\gamma q}(qc+\gamma qv^{*}(1)) & x_{t+1}=x_{t},\pi(x_{t})=1\end{cases}$$

Thus,

$$\begin{aligned}\Delta_{x}&=Q^{*}(x,0)-Q^{*}(x,1)\\&=-r(x)+\gamma\sum_{x'\in\mathcal{X}}P\left(x,x'\right)V^{*}\left(x'\right)-1-\gamma+\gamma q(qc+\gamma qv^{*}(1))\end{aligned}$$

In the last section, we've already proved that $-r(x)+\gamma\sum_{x'\in\mathcal{X}}P\left(x,x'\right)V^{*}\left(x'\right)$ is an increasing function, and since the rest of the difference above is constant, Δ_{x} is also a increasing function.

Thus, Δ_{x} would have at most one zero point. Suppose the zero point is t, when $x<t, Q^{*}(x,0)\leq Q^{*}(x,1)$, and when $x>t, Q^{*}(x,0)\geq Q^{*}(x,1)$. Thus it's proved to be threshold. If the zero point does not exist, we can simply incorporate this case as $t=-\infty$ if $u=0$ and $t=\infty$ if $u=1$.

4 Problem 4

4.1 Introduction

The objective of this planning problem is to fi

nd a feasible (and cost-efficient) path in a 3D space with box obstacles from the current confi

guration of the robot to its goal confi

guration.

The robot is defined as a point without volume. The cost is chosen from distance functions, like euclidean distance, infinite norm, Manhattan distance, etc.

To guide the robot from start point to the goal, I implemented 2 search algorithms based on different strategies: search-based RTAA* and sample based Bi-directional RRT.

4.2 Problem Formulation

Firstly, we define a cuboid set as $Cub=\{min,max|min,max\in R^{3},min\leq max\}$, where $min\leq max$ means the coordinate of min on all axis is smaller than that of max.

- 3D space: $\mathcal{S}\subseteq R^{3}$.
- Boundary: a cuboid $\mathcal{B}\in Cub$.
- Obstacles: $\mathcal{O}\subseteq Cub$, since all the obstacles in the space are box-shaped.
- Start point, goal point: $start, goal\in R^{3}$.

We discretize $\mathcal{S}$ into discrete points denoted as $\mathcal{G}\subset\mathcal{S}$. The original planning task is converted into a shortest path problem, which is defined as below.

- Node: $\mathcal{V} = \{v | v \in \mathcal{G}, v \notin \mathcal{O}\}$.
- Start node: $s \in \mathcal{V}$.
- Goal node: $\tau \in \mathcal{V}$.
- Cost:
$$C(v_i, v_{i+1}) = \begin{cases} ||v_i - v_{i+1}||_2 & \text{if } v_i \text{ and } v_{i+1} \text{ are neighboured} \\ \infty & \text{else} \end{cases}$$
- Path: an ordered list $Q := (v_1, v_2, \ldots, v_q)$ of nodes $v_k \in \mathcal{V}$.
- Set of all paths from $s \in \mathcal{V}$ to $\tau \in \mathcal{V}$: $\mathbb{Q}_{s,\tau}$.
- Path Length: sum of the arc lengths over the path: $J^Q = \sum_{t=1}^{q-1} c_{t,t+1}$.

The objective is to find a path $Q^* = \underset{Q \in \mathbb{Q}_{s,\tau}}{\arg\min} J^Q$ that has the smallest length from node $s \in \mathcal{V}$ to node $\tau \in \mathcal{V}$.

4.3 Technical Approach

One difference between search-based algorithm and sample-based algorithm is in the discretization approach. In the search-based case, the grid is aligned with axis, the neighboured nodes are defined as the immediate-neighboured grid points and the immediate-neighboured diagonal points, because we support both straight move and diagonal move. In the sampling-based algorithm, the nodes are no more constrained to be grid points, but randomly sampled in the valid configuation space.

4.3.1 RTAA*

The time complexity of A* depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: O(bd), where b is the branching factor (the average number of successors per state). RTAA* guarantees that the goal is reached in a finite number of steps.

A^* is complete and will always find a solution if one exists provided $d(x, y) > \varepsilon > 0$ for fixed ε. The most admissible and consistent heuristics. RTAA* guarantee admissible and consistent heuristics. And the heuristics is monotonically increasing.

A* is also optimally efficient for any heuristic h, meaning that no optimal algorithm employing the same heuristic will expand fewer nodes than A*, except when there are multiple partial solutions where h exactly predicts the cost of the optimal path.

The RTAA* algorithm3 is based on the A* algorithm. The main idea is that the adaptive A* makes the heuristics more informed after each A* search in order to speed up future A* searches. Compared to original A*, which may perform myopically at local minima, RTAA* **updates the heuristic over time by repeatedly move to the most promising adjacent cell using and updating a heuristic**. The heuristic updates make h more informed while ensuring it remains admissible and consistent.

Moreover, the robot is guaranteed to reach the goal in a finite number of steps if

- All edge costs are bounded from below: $c_{ij} \geq \Delta > 0$
- The graph is finite size and there exists a finite-cost path to the goal
- All actions are reversible ensures that we do not get stuck in a local min

We now formulate the main idea behind Adaptive A*. Here we define some variables that will appear in the peudocode. Notice that the Variables annotated with [A*] are updated during the call to astar() (= line 4 in the pseudo code), which performs a (forward) A* search guided by the current heuristics from the current state of the agent toward the goal states until a goal state is about to be expanded or $lookahead > 0$ states have been expanded.

constants and functions

- S: set of states of the search task, a set of states

- GOAL: set of goal states, a set of states
- A(): sets of actions, a set of actions for every state
- succ(): successor function, a state for every state-action pair
- lookahead: number of states to expand at most, an integer larger than zero
- movements: number of actions to execute at most, an integer larger than zero
- scurr: current state of the agent, a state [USER]
- c: current action costs, a float for every state-action pair [USER]
- h: current (consistent) heuristics, a float for every state [USER]
- g: g-values, a float for every state [A*]
- CLOSED: closed list of A* (= all expanded states), a set of states [A*]
- $\bar{s}$: state that A* was about to expand when it terminated, a state [A*]

Algorithm 3 $RTAA^*$ algorithm

procedure $RTAA^*$(start=x_0, end=τ)
  while $s_{\text{curr}} \notin GOAL$ **do**
    lookahead := any desired integer greater than zero
    astar()
    if $\bar{s} = FAILURE$ **then return** $FAILURE$
    for all $s \in CLOSED$ **do**
      $h[s] := g[\bar{s}] + h[\bar{s}] - g[s]$
    movements := any desired integer greater than zero
    while $S_{\text{curr}} \neq \overline{S}$ AND movements > 0 **do**
      $a :=$ the action in $A\left(s_{\text{curr}}\right)$ on the cost-minimal trajectory from s_{curr} to $\bar{s}$
      $s_{\text{curr}} := \operatorname{succ}\left(s_{\text{curr}}, a\right)$
      movements := movements -1
  return $SUCCESS$

4.3.2 Bi-directional RRT

The main idea of Bi-directional RRT is that a tree is constructed from random samples with root xs. The tree is grown until it contains a path to x_τ. RRTs are well-suited for single-shot planning between a single pair of x_s and x_τ (single query). Bi-direction RRT expand two trees at the same time, which can accelerate the search of the tree.

RRT and RRT-Connect are probabilistically complete, which means the probability that a feasible path will be found if one exists, approaches 1 exponentially as the number of samples approaches in

finity.

But RRT is not optimal. The probability that RRT converges to an optimal solution, as the number of samples approaches in

nity, is zero under reasonable technical assumptions.

Algorithm 4 $Bi - directionalRRT^*$ algorithm

procedure $Bi_R RT^*$(start=x_0, end=τ)
  while $s_{\text{curr}} \notin GOAL$ **do**
    $V_a \leftarrow \{x_s\}; E_a \leftarrow \emptyset; V_b \leftarrow \{x_\tau\}; E_b \leftarrow \emptyset$
    for $i = 1 \ldots n$ **do**
      $X_{\text{rand}} \leftarrow$ SAMPLEFREE ()
      $x_{\text{ nearest }} \leftarrow$ NEAREST $((V_a, E_a), x_{\text{rand}})$
      $x_c \leftarrow$ STEER $(x_{\text{nearest}}, x_{\text{rand}})$
      if $x_c \neq x_{\text{nearest}}$ **then**
        $V_a \leftarrow V_a \cup \{x_c\}; E_a \leftarrow \{(x_{nearest}, x_C), (x_c, x_{nearest})\}$
        $x'_{\text{nearest}} \leftarrow$ N EAREST $((V_b, E_b), x_c)$
        $x'_c \leftarrow$ STEER $(x'_{\text{nearest}}, x_C)$
        if $x'_c \neq x'_{\text{nearest}}$ **then** $V_b \leftarrow V_b \cup \{x'_c\}; E_b \leftarrow \{(x'_{nearest}, x'_c), (x'_c, x'_{nearest})\}$
        if $x'_c = x_c$ **then**
          Return SOLUTION
        if $|V_b| < |V_a|$ **then**
          $Swap((V_a, E_a), (V_b, E_b))$

4.4 Results

Here, we compare the performance of RRT and A*.

RRT

- Sparse exploration requires little memory and computation
- Solutions can be highly sub-optimal and require post-processing (path smoothing) which may be difficult

Weighted A*

- Systematic exploration may require a lot of memory and computation
- Returns a path with (sub-)optimality guarantees

In the table, we observed that

- Bi-directional RRT usually generates shorter path than RTAA*.
- Bi-directional RRT is not good at Monza and Maze, because the points are enclosed in mostly surrounded space with only a little exit on the wall. But RTAA* handles the case better.

In the RTAA* implementation, I used dynamically adjusted step size. The step size is large at first and when it approaches the goal, the step size will become smaller. In practice, this do accelerate the searching process.

In terms of the smoothness of the path generated by two algorithms, RTAA* is much more straight, while RRT is more zigzaging. This may be improved by performing smoothing techniques.

In term of the search efficiency, RRT is less efficient by searching lots of useless space. While RTAA* with proper heuristic function could generated a path that has a tendency towards the goal.

I also tried out a few different heuristics, and the conclusion is that under most case, the euclidean distance performs well. But in the flappy bird and Monza have better performance when using infinite distance.

Table 1: Performance comparison

Algorithm + scene	path length	number of moves	planning time
Bi-RRT + single cube	7.346	13	4
Bi-RRT + flappy bird	34.967	57	52
Bi-RRT + tower	39.04	64	172
Bi-RRT + room	13.127	22	12
Bi-RRT + window	33.812	55	16
Bi-RRT + monza	75.813	124	2241
Bi-RRT + maze	112.303	179	2565
RTAA* + single cube	1.37	49	0
RTAA* + flappy bird	9.92	685	0
RTAA* + tower	18.33	1042	0
RTAA* + room	9.66	546	0
RTAA* + window	5.09	375	0
RTAA* + monza	162.74	367	0
RTAA* + maze	344.24	1118	0

Here are screen shots of visualized path. Notice, the red line in the RTAA* figures are the real time trajectory, while ones in the RRT figures are the RRT trees, only the blue lines are the trajectory of robots.

4.5 RTAA*

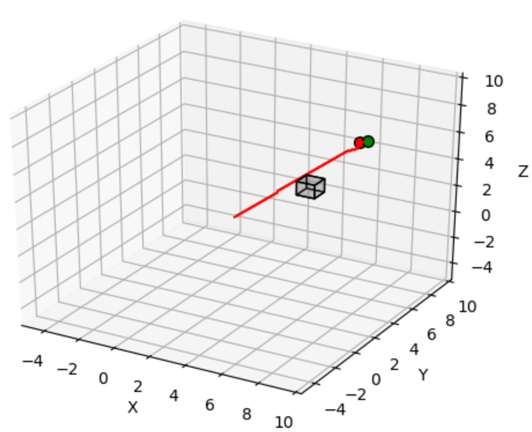

Figure 2: RTAA-singlecube

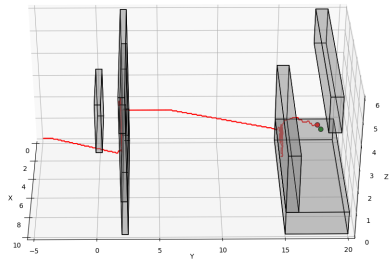

Figure 3: RTAA-window

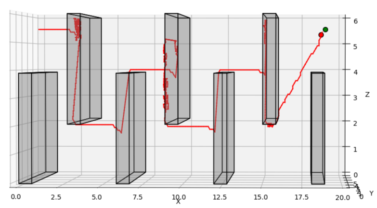

Figure 4: RTAA-flappy

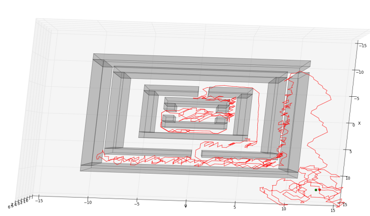

Figure 5: RTAA-maze

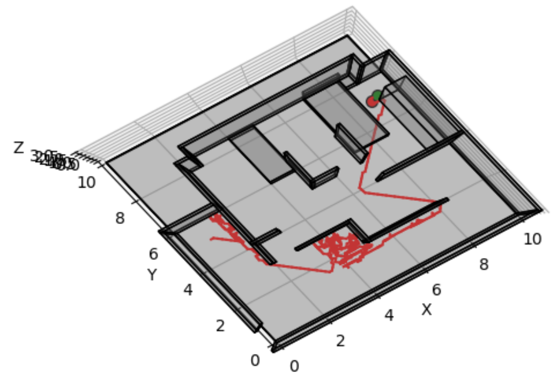

Figure 6: RTAA-room

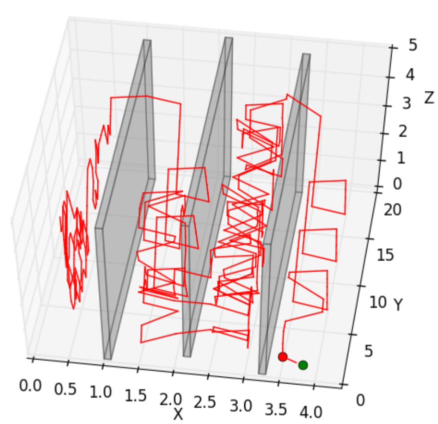

Figure 7: RTAA-monza

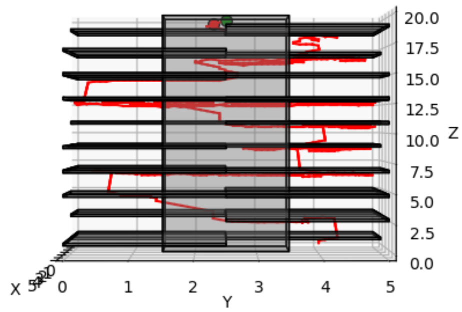

Figure 8: RTAA-tower

4.6 Bi-Directional RRT

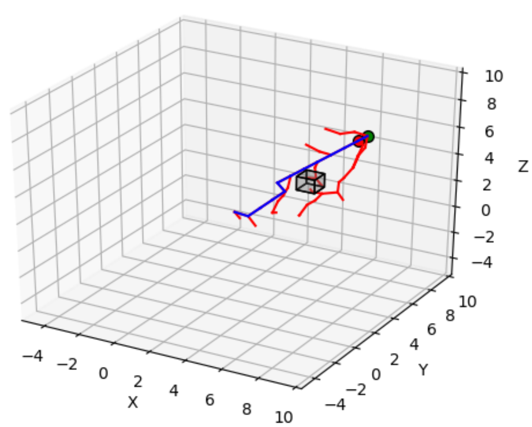

Figure 9: RRT-singlecube

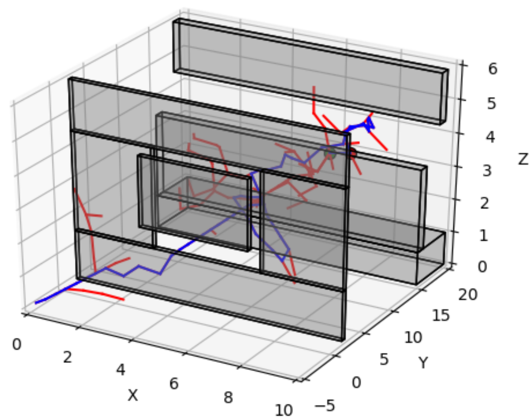

Figure 10: RRT-window

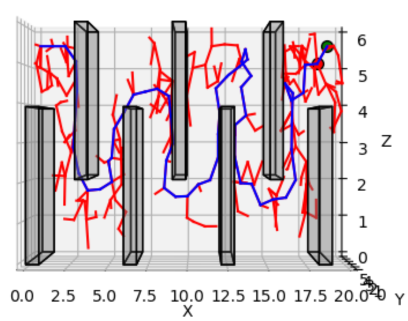

Figure 11: RRT-flappy

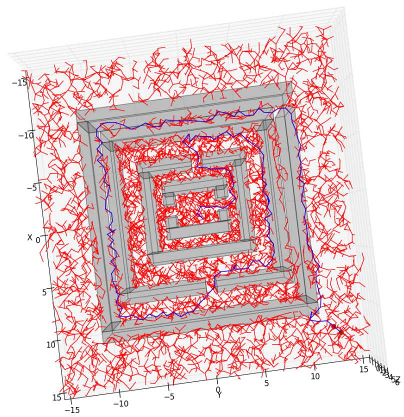

Figure 12: RRT-maze

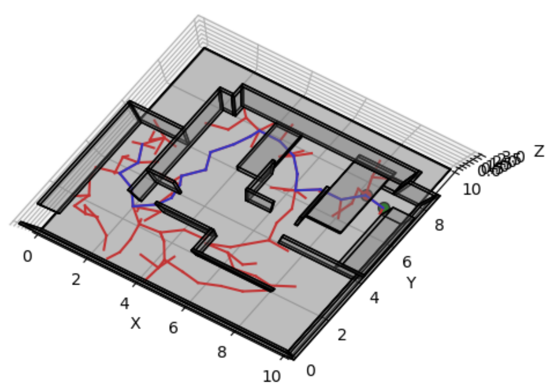

Figure 13: RRT-room

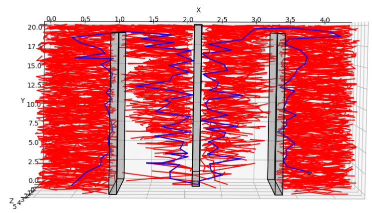

Figure 14: RRT-monza

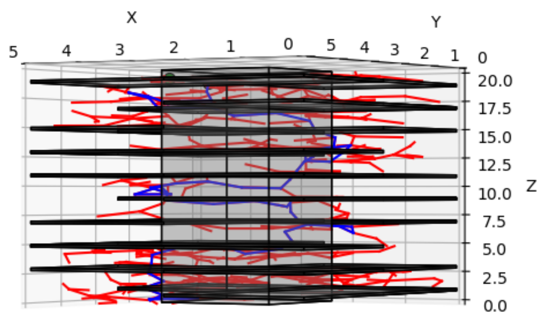

Figure 15: RRT-tower

The collision detection is inspired by the ray-box intersection algorithm[1].

References

[1] Williams, Amy, et al. ”An efficient and robust ray-box intersection algorithm.” Journal of graphics tools 10.1 (2005): 49-54.

[2] Koenig, Sven, and Maxim Likhachev. ”Real-time adaptive A.” Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. ACM, 2006.