Motion Planning

Yahsiu Hsieh
Department of Electrical & Computer Engineering
University of California, San Diego
y4hsieh@eng.ucsd.edu

***Abstract*—This paper will focus on comparing the performance of search-based and sampling-based planning algorithms in 3-D Euclidean space.**

I. Introduction

Motion Planning is a term used in robotics is to find a sequence of valid configurations that moves the robot from the starting point to destination. It has several robotics applications, such as autonomy, automation, and robot design in CAD software, as well as applications in other fields, such as animating digital characters, video game, artificial intelligence, architectural design, robotic surgery, and the study of biological molecules.

This project will focus on comparing the performance of search-based and sampling-based planning algorithms in 3-D Euclidean space. The goal is to get a feasible path from starting point to goal point. The environment is a 3-D environments described by a rectangular outer boundary and a set of rectangular obstacle blocks.

The rest of paper is as follows. First we give the detailed formulations of Motion Planning problem in Section II. Technical approaches are introduced in Section III. And at last the experiment and results are presented in Section IV.

II. Problem Formulations

In this section, the feasible path planning problem are formalized.

Let $\mathbb{X} = (0,1)^d$ be the configuration space, where $d \in \mathbb{N}, d \geqslant 2$. Let $\mathbb{X}_{obs}$ be the obstacle region, such that $\mathbb{X} \backslash \mathbb{X}_{obs}$ is an open set, and denote the obstacle-free space as $\mathbb{X}_{free} = cl(\mathbb{X} \backslash \mathbb{X}_{obs})$, where $cl(\cdot)$ denotes the closure of a set. The initial condition x_{init} is an element of $\mathbb{X}_{free}$, and the goal region $\mathbb{X}_{goal}$ is an open subset of $\mathbb{X}_{free}$. A path planning problem is defined by a triplet $(\mathbb{X}_{free}, x_{init}, \mathbb{X}_{goal})$.

A function $\sigma : [0,1] \to \mathbb{R}^d$ of bounded variation is called a

- Path, if it is continuous;
- Collision-free path, if it is a path, and $\sigma(\tau) \in \mathbb{X}_{free}$, $\forall \tau \in [0,1]$;
- Feasible path if it is a collision-free path, $\sigma(0) = x_{init}$, and $\sigma(1) = cl(\mathbb{X}_{goal})$.

A feasible path planning problem is that, given a path planning problem $(\mathbb{X}_{free}, x_{init}, \mathbb{X}_{goal})$, find a feasible path $\sigma : [0,1] \to \mathbb{X}_{free}$ such that $\sigma(0) = x_{init}$ and $\sigma(1) \in cl(\mathbb{X}_{goal})$, if one exists. If no such path exists, report failure.

III. Technical Approaches

In this section we discuss about how search-based planning algorithms are designed in this project.

A. A* Algorithm

How we design our cost and heuristic function will greatly affect the efficiency and optimality of the A* algorithm. Therefore, a well designed cost and heuristic function is required in order to achieve this task. Here we will discuss how the cost function for each step and heuristic function h are designed, and the properties of A*.

For the step cost c_{ij}, $\forall i \neq \tau, j \in Children(i)$, we set it to the distance between node i and node j, which is 0.5. There are 6 children for each node i, namely front, behind, left, right, up, and down. These are enough for our algorithm to find a solution path if it exists, giving more children will result in higher time penalty.

As for the heuristic function h, we implement two versions of heuristic function, as described below:

$$h_i = \|x_\tau - x_i\|_{L1} \tag{1}$$

$$h_i = \|x_\tau - x_i\|_{L1} + 0.5 \times |x_{\tau,z} - x_{i,z}| \tag{2}$$

where x_τ is the goal node and x_i is the current node.

For all the maps, we can use Equation (1) as our heuristic function, but this heuristic function will lead to huge computation time penalty for specific maps such as monza. From Equation (2), we can see that we add the difference between the z-axis to our heuristic function. This version of heuristic function can be used in map like tower, monza, or maze. The effect of this is that it will greatly reduce our computation time since the search will have preference over those nodes who have same z-axis value as the goal node. Moreover, we implemented a weighted heuristic function which has $\epsilon = 3$ in order to achieve faster search.

For both our heuristic functions, we can see that they both satisfy admissible and $\epsilon-$consistent conditions. Since our A* algorithm uses a consistent heuristic, it is guaranteed to return an optimal path to τ. Conditions are listed below:

- Admissible: $h_i \leq dist(i, \tau)$, $\forall i \in \mathbb{V}$
- $\epsilon-$Consistent: $h_\tau = 0$ and $h_i \leq \epsilon \times c_{ij} + h_j$, $\forall i \neq \tau, j \in Children(i)$, $\epsilon \geq 1$

For the properties of A*, suppose we have $|\mathbb{V}|$ denotes number of nodes in the graph. We have the two following properties:

- Time complexity: $O(|\mathbb{V}|^2)$.
- Space complexity: $O(|\mathbb{V}|)$ since A* does provably minimum number of expansions.

Algorithm 1 illustrates the general idea of A* algorithm for this particular problem.

Algorithm 1 A* Algorithm

```
OPEN ← {s}, CLOSED ← {}
ϵ = 3, g_s = 0, g_tau = 0
while OPEN is not EMPTY do
  Remove i with smallest f_i := g_i + ϵ × h_i from OPEN
  Insert i into CLOSED
  if ‖τ − i‖_L2 ≤ 1 then
    Parent(τ) ← i
    path ← extract(τ)
    return path
  end if
  for j ∈ Children(i) do
    if j ∉ CLOSED then
      g_j ← g_i + c_ij
      Parent(j) ← i
      OPEN ← OPEN ∪ j
      continue
    end if
    if g_j ≥ g_i + c_ij then
      Update priority of j
    end if
  end for
end while
```

B. RRT Algorithm

For the sampling-based algorithm, we chose to implement RRT algorithm. Since for this project we mainly focused on implementing search-based algorithm, we implemented RRT algorithm using the code on GitHub. In this section, we will discuss the parameters we specified.

We set our length of the search tree edge as 0.5 just like we did in A*, and we allowed each node has at most eight children. After getting a sample, we will check if that sample is able to be connected from the tree, and we will check if the path is in collision with resolution 0.1 to guarantee the resulting solution path is collision free.

Although the RRT algorithm ensures probabilistic completeness, it cannot guarantee finding the most optimal path. Therefore, in section IV, we can see that the resulting path is pretty ugly. For the time efficiency, the RRT performs better on average than the A* algorithm, we can see a more detailed comparison in section IV.

IV. Results

In this section I will present the comparison between our A* algorithm and the RRT algorithm, as well as other interesting details.

A. Time and Path Length

Here we provide the computational time and solution path length for each maps of two algorithms. From Table I we can see that, for A* algorithm, the resulting solution path has fewer path length comparing to those of RRT algorithm, the reason is that A* guarantee the optimality of the solution path while RRT does not. However, the average computational time of A* is much larger than RRT, except for monza map, which is harder for RRT to find a solution path.

TABLE I: Time and Path Length Comparison

Algorithms	A*		RRT	
Maps	Time(s)	Path Length	Time(s)	Path Length
Cube	0.027	13	0.035	16
Room	2.477	15	0.072	41
Maze	177.718	87	38.662	222
Monza	15.908	82	81.495	210
Tower	39.373	47	2.104	91
Window	0.235	33	0.281	59
Flappy Bird	441.157	38	0.577	72

B. Heuristic Function

In section III, we presented two versions of heuristic function for our A* algorithm. Here we will compare the time difference for three specific maps, namely monza, maze, and tower. From Table II we can clearly see that Equation 1 requires longer search time than Equation 2 does for those three maps.

TABLE II: Time and Path Length Comparison

Heuristic	Eq(1)	Eq(2)
Maps	Time(s)	Time(s)
Maze	888.590	177.718
Monza	397.103	15.908
Tower	39.373	1.825

C. Path visualization

Here we display the solution paths for each map generated by A* and RRT algorithm. For the RRT solution path, we can see two different colors, red and blue, the solution path is colored red and blue is the search tree. From the images we can observe that although the computational time for RRT is much faster than our A* algorithm, the resulting quality of paths of our A* algorithm is much better than those generated by RRT.

D. Conclusion

From the results above, we can see that our A* and RRT algorithm works well. However, I think that the solution can be improved in some ways. Below are some possible improvements:

- For solving more difficult maps such as monza or maze, A* algorithm may take longer than we expected. Maybe we could use Jump Point Search or other advanced A* algorithm for these maps, I believe it would reduce our computational time.

- For now we are assigning same cost for all moving directions, maybe we could try to assign different cost for different direction. For example higher cost for moving away from the goal node, maybe we could result in better computational time or better path.
- Since our testing environment is relatively small, we are not sure how our algorithm perform for larger map, for example 100x100. This is one thing we could try out.

In conclusion, the result of our motion planning algorithm is good, but could still be improved with the methods mentioned above.

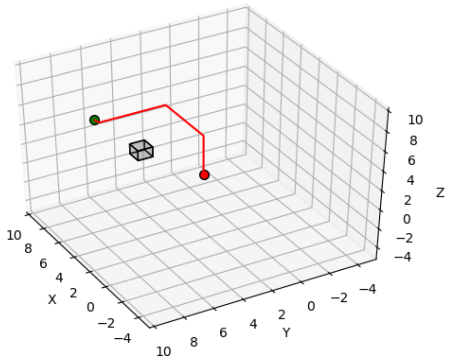

Fig. 1: Single Cube A*

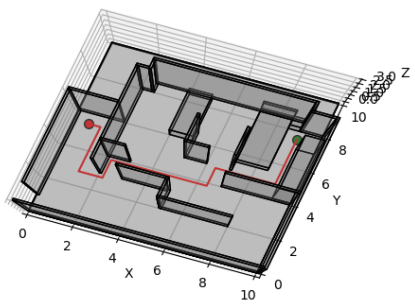

Fig. 2: Room A*

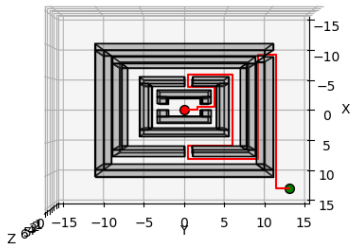

Fig. 3: Maze A*

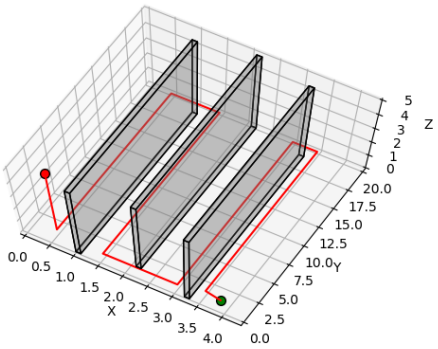

Fig. 4: Monza A*

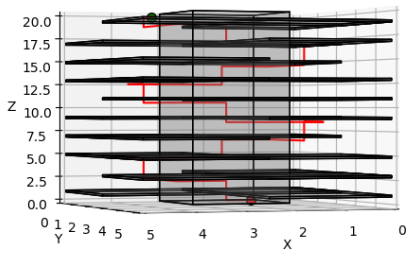

Fig. 5: Tower A*

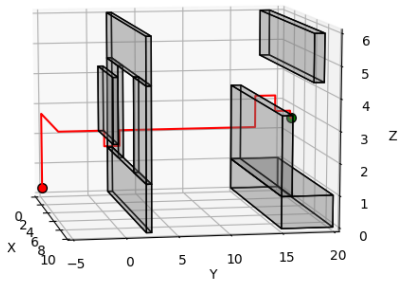

Fig. 6: Window A*

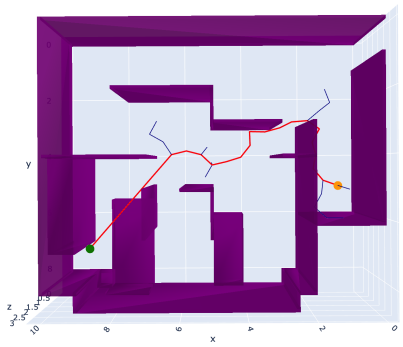

Fig. 9: Room RRT

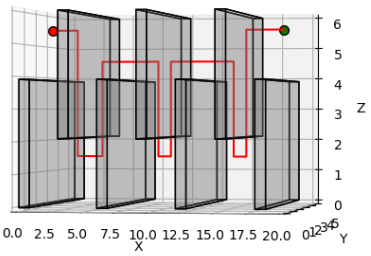

Fig. 7: Flappy Bird A*

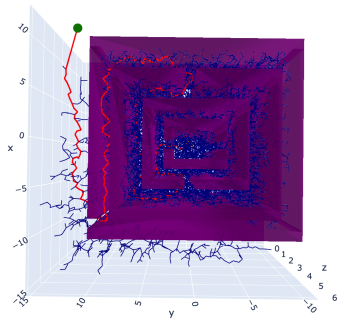

Fig. 10: Maze RRT

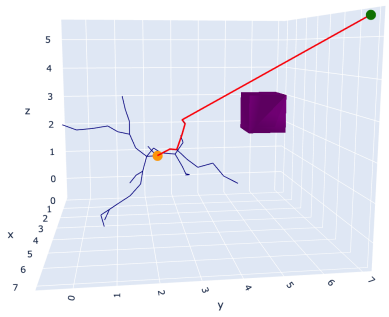

Fig. 8: Single Cube RRT

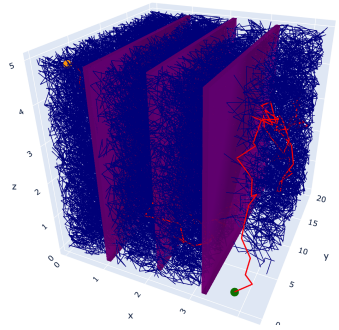

Fig. 11: Monza RRT

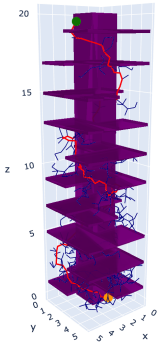

Fig. 12: Tower RRT

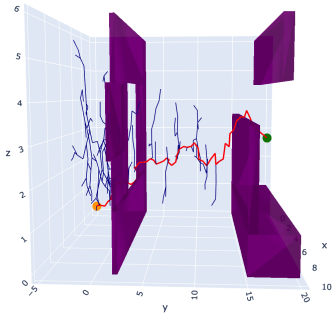

Fig. 13: Window RRT

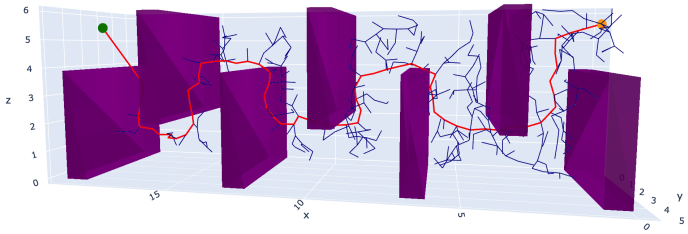

Fig. 14: Flappy Bird RRT