****

Motion planning in an Euclidian space with rectangular obstacles

****

****

Introduction

Motion planning is a computational process that aims to solve the problem of how to effectively move an object from one position to another without colliding with obstacles. It has many applications such as autonomous driving, robot navigation, robotic surgery, digital animation, and the study of molecular motion. The difficulty of this problem can vary a lot depending on the environment, and the obstacles, their geometry, whether they are static or dynamic, and how much we know about them (there can be some cases where the information about the obstacles is uncertain, or gained as the space of configurations is explored).

This paper focuses on a relatively simple case where the moving object is a point in a continuous three-dimensional space, and all the obstacles are static axis-aligned bounding boxes (AABBs). The trajectories are built from a sequence of line segments that lead from a starting point to a final point. Our aim is to find the shortest trajectory or path that goes to a goal point and that does not collide with any obstacle.
First, we implement an algorithm that checks collisions between line segments and the obstacles. Then we implement a deterministic search-based planning method using the weighted A* search algorithm. For this the space is discretized into a 3D grid. From this grid a 26-vertex-connected graph is built to search for a path towards the goal.
Finally we compare the results of this method with the rapidly exploring random tree (RRT) algorithm, which is probabilistic sampling-based method. The algorithms are tested on 7 different environments with different obstacle spaces.

This problem easily be extend to cases where the moving object is a non-rotating AABB by enlarging the size of the obstacles by half the respective dimensions of the rectangle via the Minkowski sum.

Problem statement

Consider a 3-dimensional euclidian space $\mathbb{R}^3$ where the points are specified using the cartesian coordinate system. The moving object is a point $p = (x, y, z)$. The configuration space for p is $C = \mathbb{R}^3$.
The free space C_{free}, and the obstacle space $C_{obs} = C \setminus C_{free}$ are defined by a set of $k \in \mathbb{N}$ AABBs: $B_0, \ldots, B_k$. One AABB, B_0, describes the boundary of the region in which p should stay, and any other AABB, $B_{i\geq 1}$, is a "block" describing a region of space in which p should not enter.

$$C_{free} = B_0 \setminus \bigcup_{i=1}^{k} B_k$$

Each AABB can be specified with two points: its "lower left" corner $(x_{min}, y_{min}, z_{min})$, and its "upper right" corner $(x_{max}, y_{max}, z_{max})$.
A point $(x, y, z) \in \mathbb{R}^3$ is inside an AABB if and only if:

$$\begin{cases} x_{min} \leq x_B \leq x_{max} \\ y_{min} \leq y_B \leq y_{max} \\ z_{min} \leq z_B \leq z_{max} \end{cases}$$

This means that the information about each block can be stored as a sequence of six floating-points

The trajectory can be defined by a sequence of $n \geq 2$ points in $\mathbb{R}^3$:

$$S = (p_1, p_2, \ldots, p_n)$$

More specifically, the trajectory is the 1-dimensional curve $\mathscr{C}$ defined by all the closed line segments between each pair of consecutive points in S.

$$\mathscr{C}_S = \bigcup_{i=1}^{n-1} [p_i, p_{i+1}]$$

Where $p_i = (x_i, y_i, z_i)$ and $p_{i+1} = (x_{i+1}, y_{i+1}, z_{i+1})$ and the segment $[p_i, p_{i+1}]$ is the set of points (x, y, z) verifying:

$$\begin{cases} x = x_i + t(x_{i+1} - x_i) \\ y = y_i + t(y_{i+1} - y_i) \\ z = z_i + t(z_{i+1} - x_i) \end{cases} , \forall t \in [0,1]$$

Note: the values (x, y, z) in a point correspond all three to the same $t \in [0,1]$.
The trajectory successfully goes from the start point s to the goal point τ without collision if:

- $p_1 = s$
- $p_n = \tau$
- $\mathscr{C}_S \subset C_{free}$ i.e. none of the segments of the trajectory intersect with any of the obstacles, in other words, there is no collision.

The euclidian norm $\| \ \|$ is used to define the length (or cost) l of a path $S = (p_1, p_2, \ldots, p_n)$:

$$l = \sum_{i=1}^{n-1} \| p_i - p_{i+1} \|$$

Where $\| \ \|$ is the euclidian norm.
Our aim is to find a non-colliding path to the goal that minimizes l as fast as possible. Since there can be cases were no path exist, the aim is also to be notified as quickly as possible if no path exists.

Technical Approach

First we implement a collision checker to verify that $\mathscr{C}_S \subset C_{free}$.
To check that there are not any collision we proceed in two main steps:

Step 1: check that the trajectory stays inside the B_0. Since any AABB is convex, this can be reduced to verifying that every point $p_i \in S$ is inside the B_0.

Step 2: check that the trajectory does not intersect with any of the blocks $B_{i \geq 1}$. To do so we consider each block and each segment individually and check if they intersect. Please refer to the annex to see how this is done.

Using this collision checker we then implement an algorithm to compute a trajectory using the A*.

First we discretize the space in a 3-D cartesian grid with unit length $\delta > 0$. The way the parameter δ is chosen has a lot of importance and is discussed later. A 26-vertex-connected graph is then built such that for a vertex $\mathbf{p} = (x, y, z)^T$ in the grid, its neighbors are obtained by adding a vector $\mathbf{d}$ to $\mathbf{p}$, where $\mathbf{d}$ is an element of the set:

$$\mathscr{D} = \left\{(a, b, c)^T \mid a, b, c \in \{-\delta, 0, \delta\}\right\} \backslash (0,0,0)^T$$

The cardinal of $\mathscr{D}$ is 26, so there is no ambiguity on the definition the 26 neighbors of $\mathbf{p}$.
So the cost of an edge is either δ, $\sqrt{2}\,\delta$, or $\sqrt{3}\,\delta$.
The edges that collide with an obstacle are removed. So we do not necessarily en up with a 26-vertex-connected.
Without any obstacles, the distance between two points $p_1 = (x_1, y_1, z_1)$, and $p_2 = (x_2, y_2, z_2)$ in $\mathbb{R}^3$ is:

$$\text{dist}_{p_1,p_2} = \delta\,[\gamma\sqrt{3} + (\beta - \gamma)\sqrt{2} + (\alpha - \beta - \gamma)]$$

Where α, β, γ are each associated to one of values $\left|x_2 - x_1\right|, \left|y_2 - y_1\right|, \left|z_2 - z_1\right|$ respectively, such that $\gamma \leq \beta \leq \alpha$

Two heuristic functions h, and $\tilde{h}$ are tested for the A* algorithm. The first heuristic function is defined for any point p by:

$$h_p = \| p - \tau \|$$
$$\tilde{h_p} = \| p - \tau \|_{\infty} + 0.7 \| p - \tau \|_{-\infty}$$

Where $\| \ \|_{\infty}$ is the infinity norm,
and $\| \mathbf{x} \|_{-\infty} = \min_d \left|\mathbf{x}[d]\right|$.
Both heuristic are admissible. h is consistent, but not $\tilde{h}$.
We then run the A* algorithm with h or $\tilde{h}$, and each time a new vertex is explored, we check if that vertex can be connected to the goal point with one segment. If so the path to the goal has been found and we stop the search.

δ is an important parameter. If δ is too large there might be no way of reaching the goal point without hitting an obstacle. If δ is too small, then too many vertices have to be explored before the goal can be reached. The size of the graph grows exponentially, and so does the time complexity in some cases.
For example if we take δ = 2 and consider the monza or the maze map, since the smallest gap (between two obstacles) that needs to be passed across in order to reach the goal position is of size 1, there might not be any edge in the graph that

goes across this gap (except if two vertices of the graph happen to be perfectly aligned to allow an edge to go across).

More generally, for any fixed value of $\delta > 0$, a set of AABBs obstacles can be chosen to prevent the the edges from connecting the start to the goal.

For any finite set of AABBs obstacles, if there exists a non-colliding path going from the start to the goal, δ can be reduced so that the graphs allows edges to create a path from the start point to the goal.

Moreover the smaller δ is, the shorter the trajectory can be.

Therefore finding the right value of δ is a major challenge. To find the right value of δ we use a heuristic method:

$\delta = \alpha \parallel s - \tau \parallel$

Where $\parallel s - \tau \parallel$ is the euclidian norm between the start and the goal and $\alpha > 0$ is a parameter that can be adapted.

This way, if we enlarge or shrink the space by apply a homogeneous dilation, this should not change the final result.

We start with an initial value $\alpha = \frac{1}{20}$ and then reduce α by half if no path is found.

The A* search algorithm can be written:

```
# Initialization
s ← start, τ ← goal,
g(s) ← 0
f(s) ← 0
OPEN = {s}, CLOSED = {}
# Graph exploration
While OPEN is not empty, do:
    Remove i with smallest f(i) = g(i) + ϵh(i) from OPEN
    Insert i into CLOSED
    if the segment [i, τ] ∈ C_free, then:
        Parent(τ) ← i
        g(τ) ← g(i) + ‖ i − τ ‖
    else:
        For j ∈ Children(i)
            if j ∉ CLOSED and [i, j] ∈ C_free do:
                if g(j) > g(i) + c_ij then
                    g(j) ← g(i) + c_ij
                    Parent(j) ← i
                    f(j) ← g(j) + ϵh(j)
                if j ∈ OPEN then:
                    Reorder OPEN
                else:
                    Add j to OPEN
if no path is found, then:
    δ ← δ/2
    Restart until the time limit is reached
```

Where:

- c_{ij} is the length of the edge from i to its child j.
- g(i) is the distance from the start point to an explored vertex i.
- The time limit is set to 30 seconds.
- The parameter ϵ=1, can be increased to try and speed up the search.

The time complexity of the algorithm depends on the number of vertices that need to be explored in order to reach the goal. It grows exponentially as δ is reduced.

Since the space is discretized the solution provided for the path is not optimal except if there are no obstacles between the starting point and the goal. Moreover since the free space is an open set, there will always be a path that goes closer to the boundaries of the obstacles and could potentially be shorter, so there is no shortest path in general.

If we consider the graph, the A* algorithm should return the shortest path that can be built with that grid. But since we use a weighted A* algorithm the path found is not necessarily the shortest in the graph.

Also the algorithm is not complete, because when the algorithm terminates, there is no way of telling (without information about the environment) if it stopped because the time limit was reached or because no path exist.

If we use an infinite time limit the algorithm should end up giving a path to the goal if one exists, but it will run forever if there exists no path to the goal.

We then use a motion planning library using the rapidly-exploring random tree algorithm to try and build a path from the start to the goal.

This algorithm explores the free space by building a tree of vertices rooted at the start vertex and expanding in random directions.

This is done by taking a random point in the free space. Some parameters limit the distance between this sampled point and the closest vertex in the tree (in our case the maximum distance is 4). If the segment between this sampled point and the the closest vertex in the tree does not intersect with the obstacle space, then the sampled point becomes a vertex of the tree connected to that closest vertex. Each time a point is added to the tree, the algorithm checks with a small probability ($p = 0.1$ in our case) if that point can be connected to the goal.

The total number of points sampled is specified at the beginning ($N = 20{,}000$ in our case). If N is too small, the tree might not grow large enough to be able to reach the goal.

Results

We try the for seven environments.
Please see to the plots of the solution in annex.

	Weighted A*			RRT		
Map	Length	Nodes	Time	Length	Nodes	Time
single cube	8	8	0.005	18	25	0.025
maze	76	15656	27	134	14206	10.8
flappy bird	28	417	0.61	42	320	0.29
monza	74	13611	21	105	12313	8.9
window	26	155	0.35	32	64	0.067
tower	33	700	4.6	42	602	0.56
room	12	160	0.69	17	100	0.10

Table 1: Comparison of the lengths of the computed paths, the total number of nodes explored and the execution time between the A* heuristic and the RRT algorithm.

We notice that the weighted A* algorithm tends to give shorter paths, but takes more time to compute the paths, whereas the RRT algorithm returns longer paths, but is much more time efficient since it explores less nodes.

We try different values of δ, and notice that indeed, for too large values of δ no path can be built to the goal, and for too small values of δ, the running time can explode.

There doesn't seem to be any major difference between the heuristic functions h and $\tilde{h}$, appart that $\tilde{h}$ allows the path to be returned a little faster in general.

Conclusion

We have successfully managed to find a path in all seven environments. However the time efficiency is not always the best in each case.

An approach to try and find a shorter path could be using visibility graphs.
An interesting thing to try also could be to use a moving (symmetric) object with volume instead of a point and use Minkowski sum to enlarge the obstacles.

References

- https://github.com/motion-planning/rrt-algorithms
- **********
- *Wikipedia* "Motion planning" page
- *Wikipedia* "Rapidly-exploring random tree" page

Annex 1 - Description of the collision checking algorithm

Let $[p_1, p_2]$ be a segment with $p_1 = (x_1, y_1, z_1)$, and $p_2 = (x_2, y_2, z_2)$.
Let B be an AABB block defined by $(x^B_{min}, y^B_{min}, z^B_{min}, x^B_{max}, y^B_{max}, z^B_{max})$.
To check that if any point of the segment $[p_1, p_2]$ is in B we use equation of a segment and check that it does not intersect with the B, proceed in three steps, along the three axis (x, y, and z):

Step 1 along the x-axis:

$x_{min} = \min(x_1, x_2)$
$x_{max} = \max(x_1, x_2)$
$I_x = [x_{min}, x_{max}] \cap [x^B_{min}, x^B_{max}]$
If $I_x = \emptyset$ there is no intersection.
Otherwise I_x can be written $I_x = [x^{I_x}_{min}, x^{I_x}_{max}]$ with

$$x^{I_x}_{min} = \max(x_{min}, x^B_{min})$$
$$x^{I_x}_{max} = \min(x_{max}, x^B_{max})$$

Step 2 Check along the y-axis:

We only consider the values of y in $[y^{I_x}_{min}, y^{I_x}_{max}]$corresponding values of x in I_x. There are two cases:

If $x_1 - x_2 = 0$, then

$$y^{I_x}_{min} = \min(y_1, y_2)$$
$$y^{I_x}_{max} = \max(y_1, y_2)$$

Otherwise there exists a unique $t^{I_x}_{min} \in [0,1]$ such that

$$\begin{cases} x^{I_x}_{min} = x_1 + t^{Ix}_{min}(x_2 - x_1) \\ y^{I_x}_{min} = x_1 + t^{Ix}_{min}(y_2 - y_1) \end{cases}$$

$$\begin{cases} t^{Ix}_{min} = \frac{x^{I_x}_{min} - x_1}{x_2 - x_1} \\ y^{I_x}_{min} = x_1 + \frac{x^{I_x}_{min} - x_1}{x_2 - x_1}(y_2 - y_1) \end{cases}$$

In the same way, we can find $y^{I_x}_{max}$.

Then we define
$I_y = [y^{I_x}_{min}, y^{I_x}_{max}] \cap [y^B_{min}, y^B_{max}]$
If $I_y = \emptyset$, there is no intersection.
Otherwise I_y can be written $I_x = [y^{I_y}_{min}, y^{I_y}_{max}]$ with

$$y^{I_y}_{min} = \max(y^{I_x}_{min}, y^B_{min})$$
$$y^{I_y}_{max} = \min(y^{I_x}_{max}, y^B_{max})$$

Step 3 Along the z-axis:

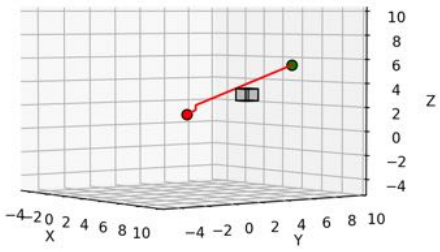
We only consider the values of z in $[z^I_{min}, z^I_{max}]$ corresponding values of y in I_y and x in I_x. There are four cases:

- If $x_1 - x_2 = 0$ and $y_1 - y_2 = 0$
- If $x_1 - x_2 = 0$ and $y_1 \neq y_2$
- If $y_1 - y_2 = 0$ and $x_1 \neq x_2$
- Otherwise $x_1 \neq x_2$ and $y_1 \neq y_2$

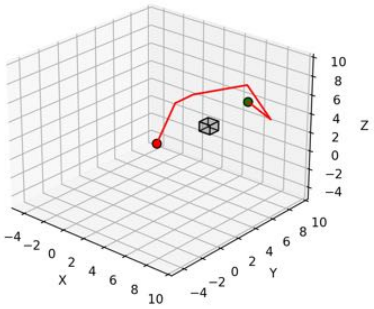
We proceed the same way as in step 2 and verify that for all points in $[p_1, p_2]$, the x, y, and z-components are not all "simultaneously" in B

Annex 2: Comparison of the paths computed by the weighted A* and RRT algorithms

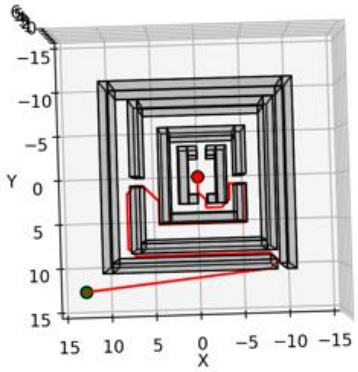
Single cube - A*



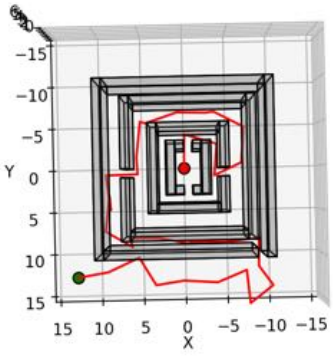
Single cube - RRT



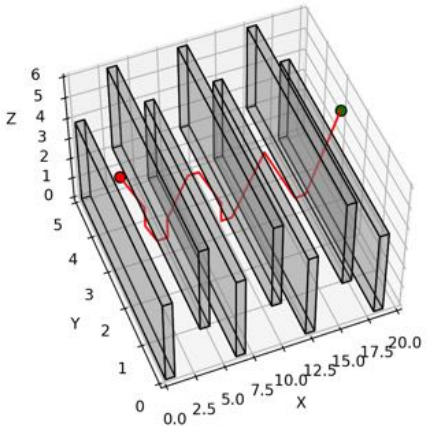
Maze - A*



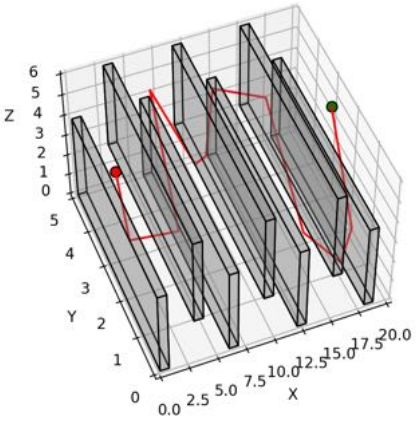
Maze - RRT



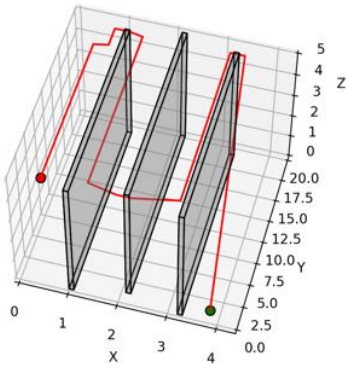
Flappy bird - A*



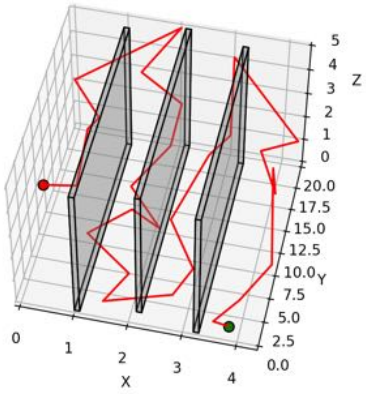
Flappy bird - RRT



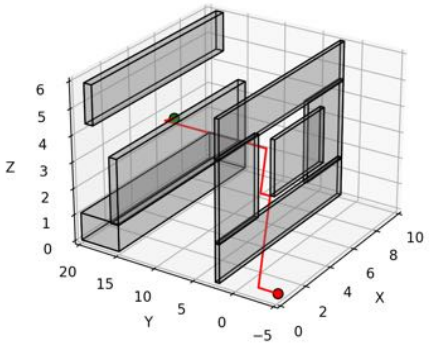
Monza - A*



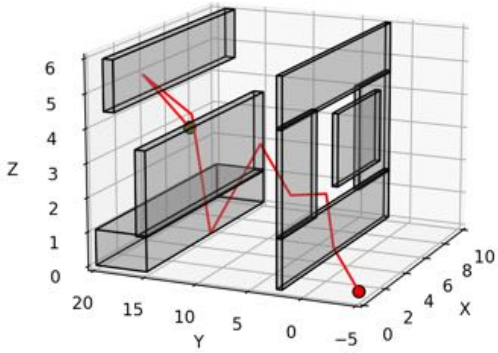
Monza - RRT



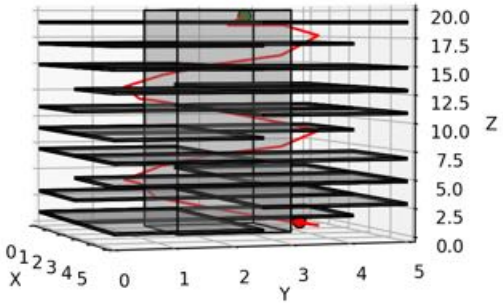
Window - A*



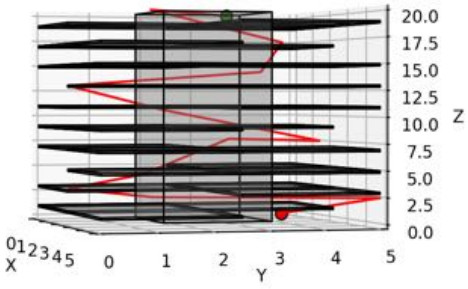
Window - RRT



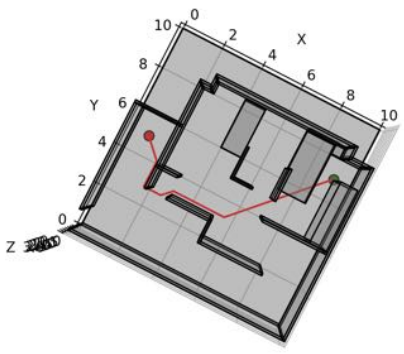
Tower - A*



Tower - RRT



Room - A*



Room - RRT