ATLANTIS PRESS

*Advances in Engineering Research, volume 120*

International Forum on Energy, Environment Science and Materials (IFEESM 2017)

# EB-RRT* based Navigation Algorithm for UAV

Dai Hongliang, Liu Qinglin

Zhejiang transportation research institute, Hangzhou Zhejiang 310023 China

**Keywords:** RRT; UAV; navigation algorithm; convergence speed

**Abstract:** With the wider application of unmanned aerial vehicle (UAV), automatic navigation capacity plays an important role. Navigation for UAV is the algorithm that automatically find out the obstacle-free, smoothing path from start position to target position. Current most navigation algorithms for UAV still have shortcomings including low convergence speed, long traversal time and smoothing challenges. EB-RRT* algorithm is proposed in this paper who has three outstanding strategies for UAV. Self-avoidance is adopted to improve convergence speed and less memory cost. Grid partitioning is applied to shorten the time of finding the nearby vertices. Smoothing turning point is introduced to improve the convergence rate of the algorithm and the smoothness of the final path. Finally, abundant simulations are carried out to testify the high performances of EB-RRT* compared with MB-RRT* and BRRT*.

## Introduction

Motion planning is the most important problem in UAVs and robotics. It can be defined as the process of finding a collision-free path for a UAV from its initial to goal point while avoiding collisions with any static obstacles or other agents present in its environment. It has gained popularity among researchers due to widespread applications such as in GPS navigation, UAV, computer animation, routing, manufacturing and many other aspects of daily life.According to the perceived ability, the navigation algorithms for UAV can be divided into local motion planning and global motion planning. Using global motion planning, it is need to know all the information of environment. But as for local motion planning, the information of environment in the perceptual range is enough. Artificial Potential Fields(APF) [1] is a well-known resolution complete algorithm. However, APF suffers from the problem of local minima and does not perform well in the environment with narrow passages. This discretization of search space makes the algorithm computationally expensive for higher dimensional spaces, that is why the application of such algorithms like Cell Decomposition methods[2], Delaunay Triangulations[3], Genetic algorithms[4] and Particle Swarm Optimization[5] are limited to low dimensional spaces only. Moreover the algorithms that combine the set of allowed motions with the graph search method thus generating state lattices also suffered from the undesirable effects of discretization. Hence to solve the higher dimensional planning problems, the sampling-based algorithms were introduced; the main advantage of sampling-based algorithms as compared to other state-of-the-art algorithms is avoidance of explicit construction of obstacle configuration space. These algorithms ensure probabilistic completeness which implies that as the number of iterations increases to infinity, the probability of finding a solution, if one exits, approaches one.

The sampling-based algorithms have proven to be computationally efficient solution to motion planning problems. Arguably, the most well-known sampling-based algorithms include Probabilistic Road Maps(PRM) [6] and Rapidly exploring Random Trees(RRT) [7]. However, PRMs tend to be inefficient when obstacle geometry is not known beforehand. Therefore, in order to derive efficient solutions for motion planning in the practical world, the Rapidly-exploring Random Trees(RRT) algorithms have been extensively explored. Various algorithms enhancing original RRT algorithm have been proposed. The Particle RRT algorithm[8] which explicitly considers uncertainty in its domain, similar to the operation of a particle filter is proposed by Nik A. Melchior and Reid Simmons. S.R. Lindemann introduces RRT-like planners based on exact Voronoi[9] diagram computation, as well as sampling-based algorithms which approximate their behavior. One of the most remarkable variant of RRT algorithm is RRT*[10], an algorithm which guarantees eventual convergence to an optimal path solution, unlike the original RRT algorithm. Just like the RRT algorithm, RRT* is able to generate an initial path towards the goal very quickly. It then continues to refine this initial path in successive iterations, eventually returning an optimal or near optimal path towards the goal as the number of iterations approach infinity. M Jordan presents a simple, computationally-efficient, two-tree variant of the RRT* algorithm[10] to improve convergence speed. Xu Zhang proposes an extension of RRT* based on a self-learning strategy and a hybrid-biased sampling scheme to improve the planning efficiency[12]. Rapidly-exploring random snakes (RRS) [13]proposed by K. Baizid is a combination of a modified deformable Active Contours Model and the RRT. On this basis, people use these algorithms to solve practical problems. A new method based on rapid-growing random trees (RRT) is used to solve the problem of segmented assembly path planning[14]. The SRRT guarantees continuity of curvature along the path satisfying any upper-bounded curvature constraints[15].

In this paper, EB-RRT* algorithm is proposed who has three outstanding strategies for UAV. Self-avoidance is adopted to improve convergence speed and less memory cost. Grid partitioning is applied to shorten the time of finding the nearby vertices. Smoothing turning point is introduced to improve the convergence rate of the algorithm and the smoothness of the final path. Finally, abundant simulations are carried out to testify the high performances of EB-RRT* compared with MB-RRT* and BRRT*.

## Related Work

### RRT

Rapidly exploring Random Trees(RRT) algorithm is proposed by S.M. LaValle and J.J. Kuffner and it has proven to be computationally efficient solution to motion planning problems. Firstly, sample in the obstacle-free space and get an independent and uniformly distributed random sample. Then, find the closest vertex to the sample. If the line segment between the node and the sample is in the obstacle-free space which means pass the collision test, the sample is inserted to the tree. It does not stops iteration until find a collision-free path from its initial to goal point while avoiding collisions with any static obstacles.

Random sampling makes the vertices of the tree cover the entire space with the increase of the iteration, which means the algorithm ensure probabilistic completeness. The algorithm do not need complex calculations and only need sampling, collision testing and connecting. However, RRT algorithm has low convergence speed and spends a lot of iterations to finding the optimal solution. Doing nothing after inserting the sample causes the algorithm need many iterations consuming a lot of time and memory.

### RRT*

Algorithm 1 is a slightly modified implementation of RRT*. The RRT* algorithm solves the problem that the finial path of RRT algorithm is not optimal. The RRT* algorithm preserves the probabilistic integrity of the RRT algorithm and has a faster convergence speed. Following are some of the processes employed by RRT*.

**Algorithm 1** $RRT*(x_{start}, x_{goal})$

$V \leftarrow x_{start}; E \leftarrow \varnothing; T \leftarrow (V, E); i \leftarrow 0$
**while** $i < N$ **do**
  $x_{rand} \leftarrow Sample(i)$
  $x_{nearst} \leftarrow NearstNode(x_{rand}, T)$
  $x_{new} \leftarrow Steer(x_{nearst}, x_{rand})$
  **if** $CollisionCheck(x_{nearst}, x_{new})$ **then**
    $T \leftarrow InsertNode(x_{new})$
    $X_{near} \leftarrow NearNodes(x_{new}, T)$
    $x_{parent} \leftarrow ChooseBestParent(x_{new}, X_{near}, T)$
    $OptimizeVertices(x_{parent}, x_{new}, T)$
  **end if**
**end while**

Fig.1 Fake code of RRT

*Sample*: the sample procedure returns an independent and uniformly distributed random sample from the obstacle-free space.

*Nearstnode*: given a vertex $x \in X_{free}$ and tree $T = (v, E)$, the function returns a vertex $x_{nearst}$ that is closest to $x$ in terms of a given distance function. In this paper, we will use Euclidean distance.

*Steer*: given two points $x_{nearst}$ and $x_{rand}$, the function returns a point $x_{new}$ such that $x_{new}$ is closer to $x_{nearst}$ than $x_{rand}$ is. Throughout the paper, the point $x_{new}$ returned by the function *Steer* will be such that $x_{new}$ minimizes $\|x_{new} - x_{rand}\|$ while at the same time maintaining $\|x_{new} - x_{nearst}\| = \mu$, for a prespecified $\mu$.

*Nearnodes*: given a vertex $x \in X_{free}$ and tree $T = (v, E)$, the function returns a set $V_1$ of vertices such that $V_1 = \{x_{near} \in v : d(x, x_{near} \le \gamma)$, where $\gamma = k(\log n / n)^{1/d}$, $k$ is a constant, $n$ is the number of iterations and $d$ is the dimension.

*Collisioncheck*: given two points $x_{new}$ and $x_{near}$, the function returns true if the line segment between $x_{new}$ and $x_{near}$ in $X_{free}$.

But, it still has a lot of problems. Because of the low convergence rate to the optimal solution, it spends many time to iterate especially working in complex maps such as channel and maze.

**B-RRT***

A.H. Qureshi proposed TG-RRT*[16] algorithm that has higher convergence rate than RRT* algorithm. B-RRT* uses a slight variation of greedy RRT-Connect heuristic for the connection of two trees. Two directional trees employing greedy connect heuristic for the connection of trees dose not ensure asymptotic optimality. The hybrid greedy connection heuristic of B-RRT* slows down its ability to converge to the optimal solution and also makes it computationally expensive. Following are some of the processes employed by B-RRT*.

**Algorithm 4** $B - RRT*(x_{start}, x_{goal})$

$E \leftarrow \varnothing; T_a \leftarrow (x_{start}, E); T_b \leftarrow (x_{goal}, E)$
$i \leftarrow 0; \theta_{best} \leftarrow \infty$
**while** $i < N$ **do**
  $x_{rand} \leftarrow Sample(i)$
  $x_{nearst} \leftarrow NearstNode(x_{rand}, T_a)$
  $x_{new} \leftarrow Steer(x_{nearst}, x_{rand})$
  **if** $CollisionCheck(x_{nearst}, x_{new})$ **then**
    $T_a \leftarrow InsertNode(x_{new})$
    $X_{near} \leftarrow NearNodes(x_{new}, T_a)$
    $x_{parent} \leftarrow ChooseBestParent(x_{new}, X_{near}, T_a)$
    $OptimizeVertices(x_{parent}, x_{new}, T_a)$
    $x_{mid} \leftarrow NearstNode(x_{new}, T_b)$
    $\theta' \leftarrow Connect(x_{new}, x_{mid}, \mu)$
    **if** $\theta' < \theta_{best}$ **then**
      $\theta_{best} = \theta'$
    **end if**
  **end if**
  $SwapTrees(T_a, T_b)$
**end while**

Fig.2 Fake code of B-RRT*

The algorithm using two directional trees need the extra function to connect them. The function calls the ***nearnodes*** function to find the closet vertex in the range of $k(log\ n / n)^{1/d}$ from another tree and then connect.

**MB-RRT***

MB-RRT* algorithm is proposed who has three outstanding strategies for UAV, include lazy sampling, self-adaptive step size and down sampling and curve fitting. Some of the processes employed by MB-RRT* are shown in Fig.3(a).

Lazy sampling is adopted to improve convergence speed and less memory cost. In the light of $StepSize = D_i / \mu_{Max} * \mu_{Min}$, self-adaptive step size algorithm is applied to solve navigation limitation near obstacles and improve initial solutions' quality and speed. Down sampling and curve fitting improve the smoothness of the final path.

## EB-RRT* algorithm

### Main idea

Although self-adaptive step size and lazy sampling shorten the sampling time and reduce sampling vertices effectively on the basis of B-RRT*, there are still some room for improvement.

We introduce EB-RRT* algorithm that adopts self-avoidance, grid partitioning and smoothing turning point for new vertex, near vertices and final path.

**Algorithm 6** $MB-RRT*(x_{start}, x_{goal})$

```
E ← ∅; T_a ← (x_start, E); T_b ← (x_goal, E)
i ← 0; θ_best ← ∞
while i < N do
  x_rand ← Sample(i)
  x_nearst ← NearstNode(x_rand, T_a)
  x_new ← AutoStepSteer(x_nearst, x_rand)
  if c(x_nearst + c(x_nearst, x_new)) > θ_best then
    Continue
  end if
  if CollisionCheck(x_nearst, x_new) then
    T_a ← InsertNode(x_new)
    X_near ← NearNodes(x_new, T_a)
    x_parent ← ChooseBestParent(x_new, X_near, T_a)
    OptimizeVertices(x_parent, x_new, T_a)
    x_mid ← NearstNode(x_new, T_b)
    θ' ← Connect(x_new, x_mid, μ)
    if θ' < θ_best then
      θ_best = θ'
    end if
  end if
  SwapTrees(T_a, T_b)
end while
θ_best ← DownSample(θ_best)
θ_best ← BezierCurve(θ_best)
```

**Algorithm 8** $EBRRT*(x_{start}, x_{goal})$

```
E ← ∅; T_a ← (x_start, E); T_b ← (x_goal, E)
i ← 0; θ_best ← ∞
while i < N do
  x_rand ← Sample(i)
  x_nearst ← NearstNode(x_rand, T_a)
  x_new ← Steer(x_nearst, x_rand)
  if c(x_nearst + c(x_nearst, x_new)) > θ_best then
    Continue
  end if
  if !CollisionCheck(x_nearst, x_new) then
    x_nearst ← AddInform(x_nearst)
    d ← ChooseDirection(x_nearst)
    x_new = Toward(d)
  end if
  T_a ← InsertNode(x_new)
  X_near ← AreaNearNodes(x_new, T_a)
  x_parent ← ChooseBestParent(x_new, X_near, T_a)
  OptimizeVertices(x_parent, x_new, T_a)
  x_mid ← NearstNode(x_new, T_b)
  θ' ← Connect(x_new, x_mid, μ)
  if θ' < θ_best then
    θ_best = θ'
  end if
  SwapTrees(T_a, T_b)
end while
θ_best ← DownSample(θ_best)
θ_best ← SmoothPoint(θ_best)
```

Fig.3 Fake code of MB-RRT* and EB-RRT*

**Self-avoidance**

The EB-RRT* and MB-RRT* works in exactly the same manner as the original RRT* algorithm in its initial phases. It starts with sampling in the collision-free space and gets a random vertex $x_{rand}$, then searches for the closest vertex $x_{nearst}$ to it and grows forward the direction of $\overline{x_{nearst}x_{rand}}$. Self-adaptive step size adopted by MB-RRT* make the step size maintain $\mu_{Min} < \mu < \mu_{Max}$ near the obstacle, but can't avoid that a lot of vertices are abandoned because the step size is larger than the distance to the obstacle. It occurred in the maps that there are many obstacles frequently.

Following are growth processes of MB-RRT* and EB-RRT* near the obstacle. The possibility that the minimum step size is larger than the distance to the obstacle is shown in Fig.4(a). In the same situation, the surrounding environment is divided into 9 grid regions, labeled 1, 2, 3, 4, 5, 6, 7, 8, 9, respectively. The $x_{nearst}$ is in the grid regions of 5. The function *AddInform* return the information from other 8 regions and then distinguish them between the obstacle area and the non-obstacle area. The regions of 2 and 3 are obstacle areas and others are non-obstacle areas as shown in Fig.4(b). the function *ChooseDirection* samples in the non-obstacle areas as the new random vertex.

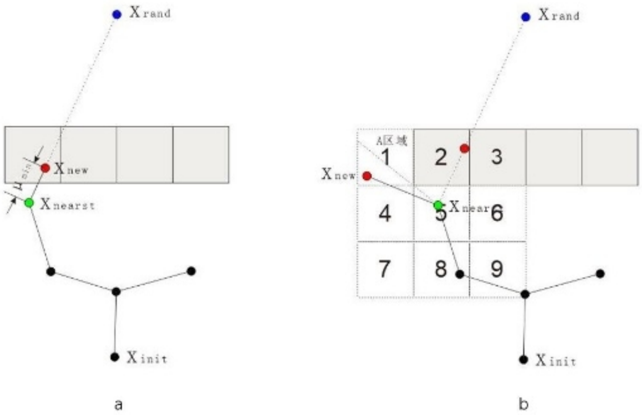



Fig.4 Growth processes of MB-RRT* and EB-RRT* near the obstacle in 2-D map

Obviously, this method also applies to complex 3-D environments that just the number of grid regions is increased to 27. However, we found that the time to distinguish non-obstacle areas with obstacle areas in complex 3-D environments using this self-avoidance method is longer. So, we use another method which is computationally small. First, find the nearest obstacle $x_{obs}$ from $x_{nearst}$ and calculate the distance $d_{obs}$. Similarly, calculate the distance $d_{parent}$ between $x_{parent}$ and $x_{nearst}$. Then, calculate the direction of $x_{new}$ based on:

$$\overrightarrow{x_{nearst}x_{new}} = \overrightarrow{x_{obs}x_{nearst}}/d_{obs} + \overrightarrow{x_{parent}x_{nearst}}/d_{parent}$$

the step size is selected $(d_{obs} + d_{parent})/2$. Fig.5 shows the growth processes of MB-RRT* and EB-RRT* in complex 3-D environments.

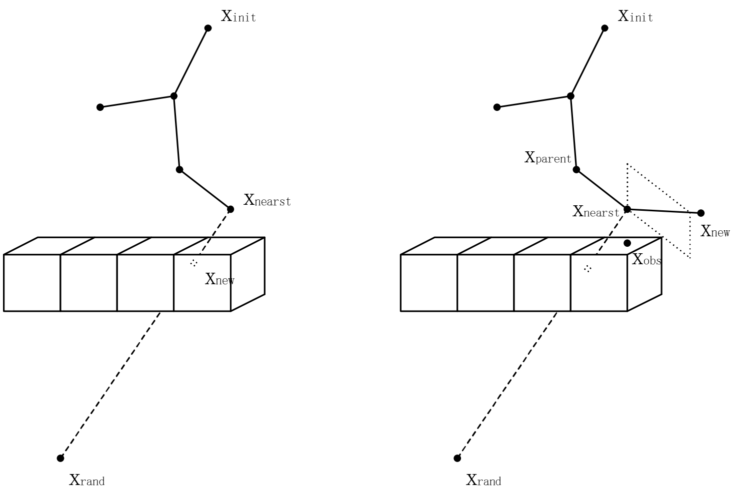



Fig.5 Growth processes of MB-RRT* and EB-RRT* near the obstacle in 3-D map

Self-avoidance make EB-RRT* algorithm explores larger range around obstacle and enhances the obstacle avoidance ability. So it can reduce the number of iterations and shorten running time effectively.

**Grid partitioning**

Each time a new vertex $x_{new}$ is inserted, it needs to find a set of $V_1 \in V$ where the distance between $x_{new}$ and the vertex is smaller than $\gamma$ through the function *NearNodes* and then determine whether the need for track correction for all vertices. During the process, it will traverse all the old vertices that have been inserted to the tree. But, with the expansion of the search space, the number of vertices and the traversal time is increasing.

The cell decomposition algorithm[18] proposed by Ahmad Abbadi and Vaclav Prenosil can divide the entire map space into obstacle and non-obstacle areas. But for shortening the loading time required to initialize the map information, we divide the 3-D map into grids, regardless of the obstacle information. The map with length $L$, width $W$ and height $H$ is divided into $l*m*n$ grids. The length of the grid is $L$, the width is $W_{grid}$ and the height is $H_{grid}$.

So

$$L_{grid} = L / l$$

$$W_{grid} = W / m$$

$$H_{grid} = H / n$$

After inserting the new vertex $x_{new}$, it only needs to traverse the old vertices in the area where the vertex is located and no more than 7 areas around it. In order to ensure that all the vertices whose Euclidean distance is less than $\gamma$ are within these 8 areas,

$$min\{L_{grid}, W_{grid}, H_{grid}\} > 2 * max(\gamma)$$

Because

$$\gamma = k(log\ n / n)^{1/d}$$

$$\gamma \le k(log\ 2 / 2)^{1/d}$$

So that

$$L_{grid} \ge 2 * k(log\ 2 / 2)^{1/d}$$

$$W_{grid} \ge 2 * k(log\ 2 / 2)^{1/d}$$

$$H_{grid} \ge 2 * k(log\ 2 / 2)^{1/d}$$

**Smoothing turning point**

Smoothing turning point breakpoints also is including down sampling and curve fitting. Down sampling of MB-RRT* makes the final path point as little as possible. But, this situation will occur in Fig6(a). There is a vertex $x_i$ that it can be connected to the goal point without collision and do not need to go through $x_{i+1},\cdots,x_{goal-1}$. The length of the final path is shorter than before as shown in Fig.6(b).

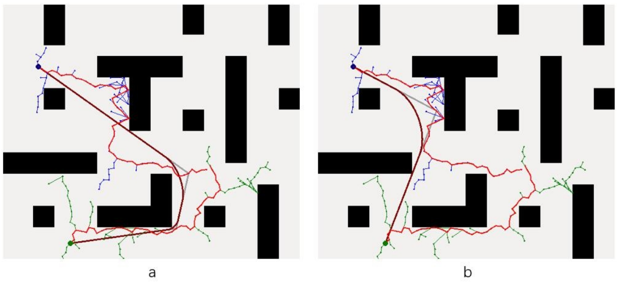



Fig.6 Down sampling of MB-RRT* and EB-RRT*

Following are the fake codes of down sampling as shown in Fig.7. Based on MB-RRT*, the down sampling algorithm used by EB-RRT* has traversed to determine whether the existence of $x_i$ that it is connected with the starting point or goal point without collision before deleting vertices.

**Algorithm 9** $DownSample(\theta_{best})$

```
for i < θ_best.size() do
  if CollisionCheck(θ_best[start], θ_best[i]) then
    for x ∈ (1, i − 1) do
      θ_best.erase(x)
    end for
  end if
  if CollisionCheck(θ_best[i], θ_best[goal]) then
    for x ∈ (i + 1, goal) do
      θ_best.erase(x)
    end for
  end if
end for
span = 2
for span < θ_best.size() do
  flag = false
  i = 0
  for span + i < θ_best.size() do
    i = i + 1
    if CollisionCheck(θ_best[i], θ_best[i + span]) then
      t = 1
      for t < span do
        t = t + 1
        θ_best.erase(i + 1)
      end for
      flag = true
    end if
  end for
  if flag == false then
    span = span + 1
  end if
end for
return θ_best
```

Fig.7 Fake code of down sampling of EB-RRT*

Curve fitting of EB-RRT* has own way to select two endpoints and two control point to calculate the Bezier curve. Fig.8 shows the process and the blue curve is the third-order Bezier curve. $X_i$ is the turning point, the line $\overline{P_0X_i} = \overline{X_iP_3} = d$ and $\overline{P_1X_i} = \overline{X_iP_2} = \alpha d$. Then the points on the curve are calculated according to cubic Bezier curve equation

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3$$

Where

$t(0 < t < 1)$

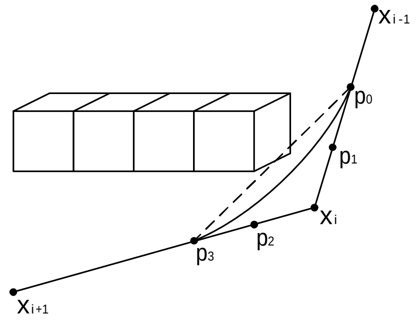


Fig.8 Curve fitting

## Analysis

### Probabilistic completeness

In any configuration space. An algorithm is said to be probabilistically complete if the probability of finding a path solution, if ones exist, approaches one as the number of samples taken from the configuration space reaches infinity. It is known that RRT is a probabilistically complete algorithm, as its optimal variant RRT*. The MB-RRT* algorithm preserves the probabilistic completeness of the RRT* algorithm. Since our proposed EB-RRT* algorithm performs the random sampling function exactly like the aforementioned algorithms and is merely a efficient version of MB- RRT*, it can be reasonably proffered that it also inherits the probabilistic completeness property of MB-RRT*.

### Asymptotic optimality

Asymptotic optimality is defined as follows: let $c^*$ be the optimal solution of the motion planning, $Y$ is the optimal path length by $\mathrm{ALG}$ algorithm after $\mathrm{n}$ iterations, the algorithm should satisfy the following equation:

$$P\left(\left\{\lim_{n\to\infty} Y_n^{ALG} = c^*\right\}\right) = 1$$

It is known that RRT* and MB-RRT* ensure optimality when the number of iterations are increased to infinity. Since there is no extra connection heuristic required for connection of the two trees and the two trees are generated exactly as the tree generated in the original RRT* algorithm, it can be reasonably proposed that the EB-RRT* algorithm inherits the asymptotic optimality property of MB-RRT*.

### Computational complexity

When calling the function of *sample*, *collisioncheck*, *optimaizeVertices* and *connect*, the running time does not depend on the number of iterations. The function *AutoStepSteer* in MB-RRT* spends $\Omega(\log n)$ to run. The function Steer used in EB-RRT* requires constant time like both in RRT* and EB-RRT*. Only when the new vertex $x_{new}$ fails the collision test, EB-RRT* will call the function of *AddInform*, *ChooseDirection* and *Toward*, all of which requires constant time. And the function of *AreaNearNodes* takes approximately $1/(m*n)$ of the function *NearNodes*.

$$\lim_{n\to\infty} \Omega_n^{MB-RRT*} = (1+2P)\Omega(\log n)$$

$$\lim_{n\to\infty} \Omega_n^{EB-RRT*} = (2+P/(m*n))\Omega(\log n)$$

$P$ is the probability of the old vertex $x_{new}$ in the obstacle. So there is a constant $\phi$ and an equation

$$\lim_{n\to\infty} \frac{\Omega_n^{EB-RRT*}}{\Omega_n^{MB-RRT*}} \leq \phi$$

## Simulation

This 2-D simulation is performed on the QT software in the Ubuntu system and the 3-D is on the ROS platform. Table 1 shows the hardware configuration used in this lab.

Table 1 Experimental hardware

| Type | Parameter |
|---|---|
| Processor | Intel(R)Core(TM)i3-2310M 2.10GHz*4 |
| System version | Ubuntu 14.04LTS |
| RAM | 5.7G |

## Experimental map

There are 6 2-D maps with different difficulty by placing different obstacles and 3 3-D maps for verifying the algorithm. The experimental environment of the 2-D is 800*600. Following are the tables of experiment map parameters.

Table 2 Experiment 2-D map parameters

| Parameter / Map | Start coordinate | Goal coordinate | Number of obstructions | Duty cycle |
|---|---|---|---|---|
| Map1 | (400,300) | (700,300) | 1 | 84/1200 |
| Map2 | (50,50) | (750,550) | 4 | 204/1200 |
| Map3 | (100,500) | (750,300) | 2 | 158/1200 |
| Map4 | (50,500) | (750,150) | 23 | 247/1200 |
| Map5 | (150,70) | (150,560) | 6 | 311/1200 |
| Map6 | (70,120) | (470,270) | 1 | 175/1200 |

Table 3 Experiment 3-D map parameters

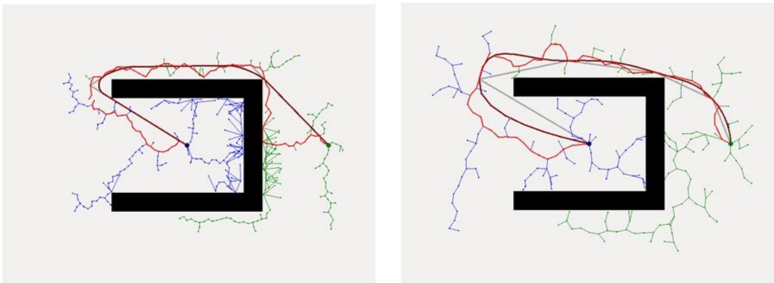
| Parameter / Map | Start coordinate/m | Goal coordinate/m | Number of obstructions | Size of space/$m^3$ | Number of grids |
|---|---|---|---|---|---|
| Map1 | (6,4,4) | (14,22,6) | 17078 | 20*25*10 | 8 |
| Map2 | (1,1,1) | (7,4,0.2) | 64090 | 9*9*2.5 | 45 |
| Map3 | (2,9,3) | (14,2,1) | 59798 | 15*15*6 | 50 |

## Conclusions and analysis

### 2-D map

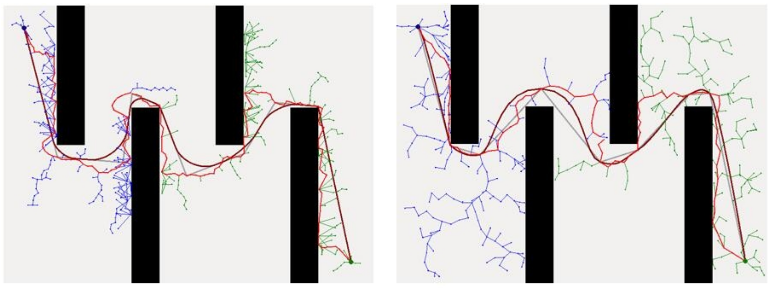
Fig.9 to Fig14 show the solution for the first time in 2-D maps, the left of which is EB-RRT* and the right is MB-RRT*. When running the EB-RRT*, the $W_{grid}$ and $H_{grid}$ are both 100, so that the map is divided into 48 grids to shorten traversal time.

The black part of the figure is the obstacle area, the blue part is starting point $x_{start}$ and the tree $T_{start}$ whose root is it, the green part is goal point $x_{goal}$ and the tree $T_{goal}$, the red part is the initial path of the current optimal solution, the gray line is the path after down-sampling, and the dark red curve is the final path.



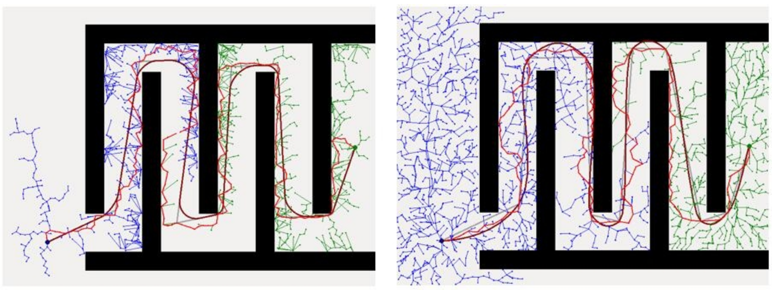
a b

Fig.9 Performance in Map1



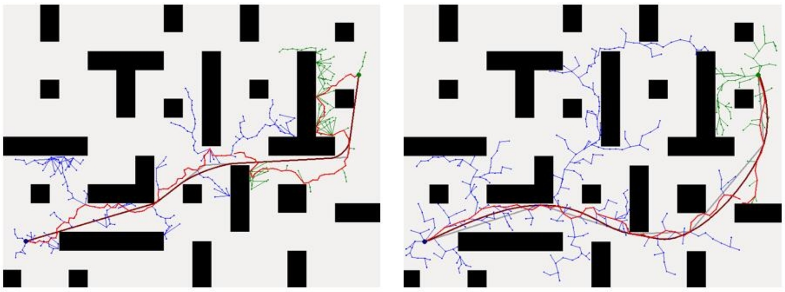
a b

Fig.10 Performance in Map2



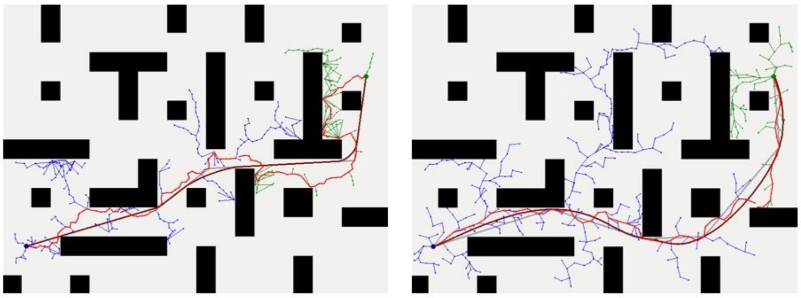
a b

Fig.11 Performance in Map3



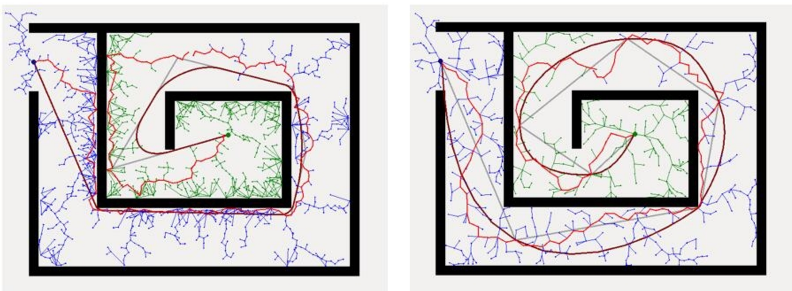
a b

Fig.12 Performance in Map4



a b

Fig.13 Performance in Map5



a b

Fig.14 Performance in Map6

EB-RRT* growth in the vicinity of obstacles is relatively intensive from the Fig.9 to 14, which means that during a lot of iterations, a lot of vertices are abandoned because the step size is larger than the distance to the obstacle in MB-RRT* and the overhead of searching for $x_{nearst}$ is meaningless. However, self-avoidance used by EB-RRT* can guarantee that a new vertex can be inserted to a certain extent that improves the efficiency of finding feasible solutions. Smoothing turning point effectively solves the problem that the final curve is not feasible.

Table 4 depicts the exact data of EB-RRT*, MB-RRT* and B-RRT* running on the Map1 to Map6 in 2-D maps.

Table 4 Experimental results for computing optimal path solution in 2-D maps

| Index | ALG | Map1 | Map2 | Map3 | Map4 | Map5 | Map6 |
|---|---|---|---|---|---|---|---|
| Iterative number | B-RRT* | 428 | 769 | 4486 | 444 | 2254 | 1508 |
| | MB-RRT* | 227 | 686 | 3385 | 357 | 1954 | 1372 |
| | EB-RRT* | 253 | 277 | 762 | 231 | 368 | 837 |
| Path length | B-RRT* | 937.909 | 1585.740 | 2253.670 | 1074.460 | 1409.880 | 2247.960 |
| | MB-RRT* | 872.330 | 1405.990 | 1917.230 | 995.762 | 1194.630 | 1803.050 |
| | EB-RRT* | 797.748 | 1446.190 | 1981.190 | 957.998 | 1177.300 | 1828.210 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Time/s | B-RRT* | 0.232896 | 0.376302 | 8.940450 | 0.178626 | 1.311880 | 1.543880 |
| | MB-RRT* | 0.079537 | 0.249050 | 4.827820 | 0.116184 | 1.182490 | 0.719947 |
| | EB-RRT* | 0.092506 | 0.106442 | 0.494353 | 0.072950 | 0.114690 | 0.682838 |
| Time/s for 1000iterations | B-RRT* | 949.847 | 1408.51 | | 1012.3 | | |
| | MB-RRT* | 1183.75 | 1405.97 | | 910.462 | | |
| | EB-RRT* | 997.991 | 1403.11 | 2379.57 | 935.454 | 1201.5 | 1828.21 |
| Time/s for 2000iterations | B-RRT* | 926.793 | 1408.51 | | 1010.51 | 1634.46 | 1813.99 |
| | MB-RRT* | 1157.45 | 1405.97 | | 910.462 | 1194.63 | 1658.58 |
| | EB-RRT* | 994.669 | 1403.11 | 2379.57 | 908.876 | 1201.5 | 1746.8 |
| Time/s for 3000iterations | B-RRT* | 925.313 | 1408.51 | | 1010.51 | 1609.03 | 1808.14 |
| | MB-RRT* | 1127.27 | 1405.97 | | 910.462 | 1194.63 | 1649.67 |
| | EB-RRT* | 985.066 | 1403.11 | 2379.57 | 908.876 | 1201.5 | 1573.09 |
| Time/s for 4000iterations | B-RRT* | 920.801 | 1407.71 | | 1010.51 | 1044.45 | 1805.16 |
| | MB-RRT* | 1126.62 | 1405.97 | 2268.35 | 910.462 | 1080.66 | 1649.67 |
| | EB-RRT* | 984.182 | 1403.11 | 2379.57 | 908.876 | 1201.5 | 1567.64 |

Fig.15 to 17 are the histograms of the iterative number, path length and time for the first feasible solution according to the Table 4. It is obviously that the number of iterations and the time of EB-RRT*are much smaller than those of MB-RRT* and B-RRT*.

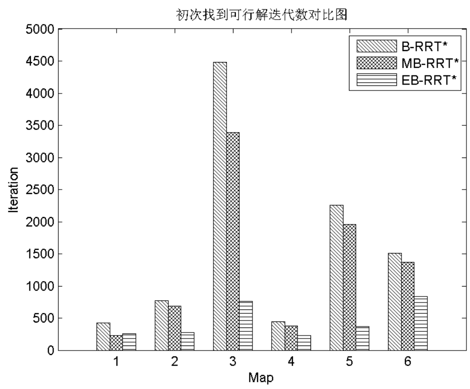
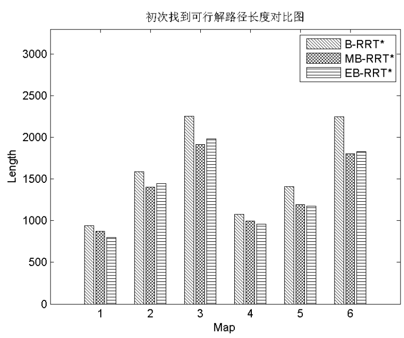



Fig.15 Iterative number for the first feasible solution



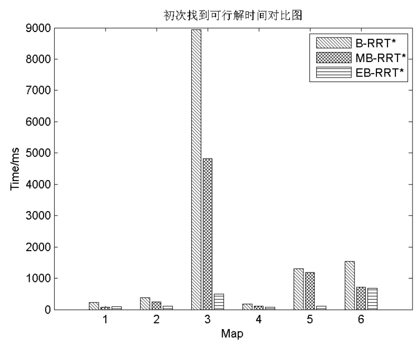




Fig.16 Path length for the first feasible solution Fig.17 Time for the first feasible solution

**3-D map**

Fig.18 to Fig.20 show the solution for the first time in 3-D maps, the left of which is EB-RRT* and the right is MB-RRT*. The white part of the figure is the obstacle area, the black part is the non-obstacle area, the blue part is starting point $x_{start}$ and the tree $T_{start}$ whose root is it, the orange part is goal point $x_{goal}$ and the tree $T_{goal}$, the yellow line is the path after down-sampling, and the red curve is the final path.

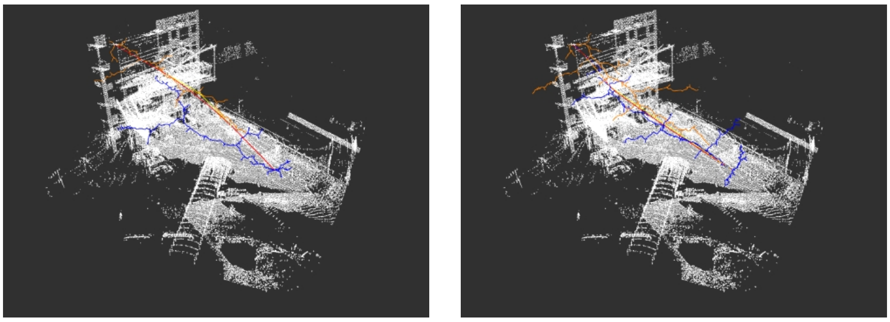


Fig.18 Performance in Map1

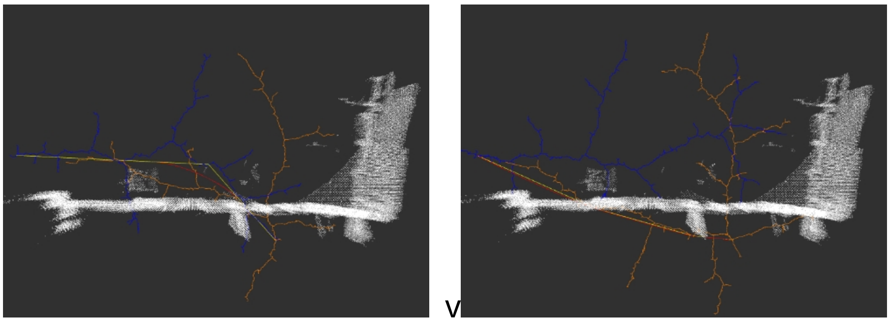


Fig.19 Performance in Map1

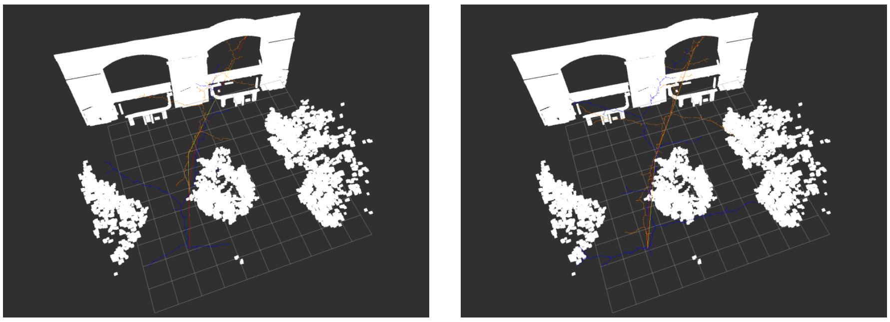


Fig.20 Performance in Map1

Because of the self-avoidance, the number of nodes of EB-RRT* is less than MB-RRT* from the performances in Map1 to Map3.

Table 5 depicts the exact data of EB-RRT* and MB-RRT* running on the Map1 to Map3 in 3-D maps.

Table 5 Experimental results for computing optimal path solution in 3-D maps

| Map | ALG | Index | First time | | | |
|---|---|---|---|---|---|---|
| Map1 | MB-RRT* | Iteration | 299 | 500 | 1000 | 1500 |
| | | Time/s | 0.349922 | 0.628029 | 2.486304 | 5.063244 |
| | | Path length/cm | 2010.445135 | 2005.03937 | 2003.30776 | 2000.097335 |
| | EB-RRT* | Iteration | 212 | 500 | 1000 | 1500 |
| | | Time/s | 0.225663 | 0.517994 | 1.368203 | 2.932502 |
| | | Path length/cm | 2004.131785 | 2004.028015 | 2001.173385 | 2000.866655 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Map2 | MB-RRT* | Iteration | 671 | 1000 | 1500 | 2000 |
| | | Time/s | 1.399038 | 2.303930 | 5.518727 | 7.430588 |
| | | Path length/cm | 733.553761 | 727.844566 | 716.675316 | 712.924559 |
| | EB-RRT* | Iteration | 599 | 1000 | 1500 | 2000 |
| | | time/s | 2.399083 | 3.379559 | 4.216219 | 6.428026 |
| | | Path length/cm | 717.090288 | 712.392299 | 705.622081 | 701.508011 |
| Map3 | MB-RRT* | Iteration | 1142 | 2000 | 2500 | 3000 |
| | | Time/s | 5.070364 | 13.512511 | 19.525062 | 26.755054 |
| | | Path length/cm | 1417.855103 | 1412.680359 | 1411.090919 | 1410.016504 |
| | EB-RRT* | Iteration | 778 | 2000 | 2500 | 3000 |
| | | Time/s | 13.876751 | 17.081206 | 20.590168 | 24.491691 |
| | | Path length/cm | 1417.230194 | 1416.904486 | 1413.268311 | 1409.403931 |

In this three 3-D maps, EB-RRT* and MB-RRT*'s data of path length are almost the same and realistic. Although the time for the first feasible solution of EB-RRT* is longer than MB-RRT*, the iteration is smaller. Assuming that the time required for MB-RRT* is $\Omega(\log n)$, then the time for EB-RRT* is $(\Omega(\log n)+A)/(l*m*n)$, $A$ is a constant which is the time to find the nearest obstacle and calculate the coordinate of $x_{new}$. So, as the number of iterations increases, EB-RRT* will take less time than MB-RRT*. The data in Table 5 also demonstrate it.

The line charts of time and path length in the case of the same number of iterations are shown in Fig.21 to Fig.23 according to the Table 5.

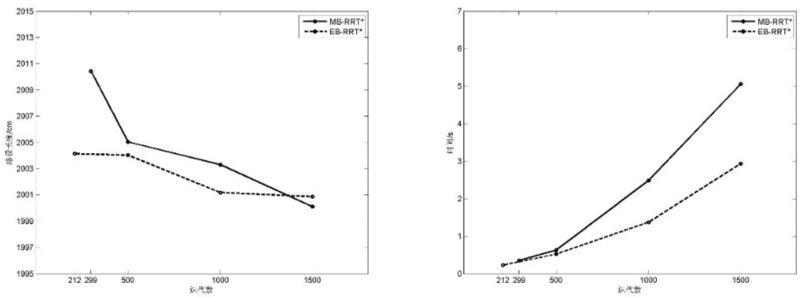



Fig.21 Time and path length in Map1

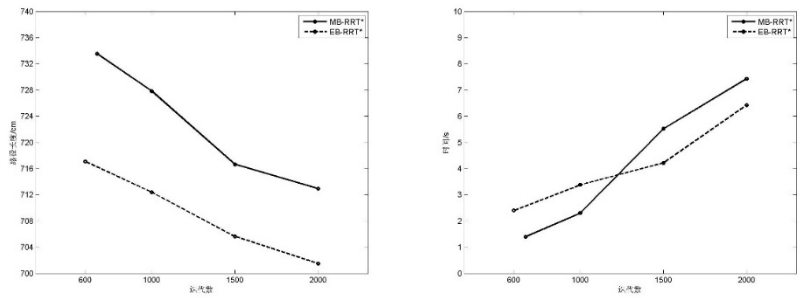



Fig.22 Time and path length in Map2

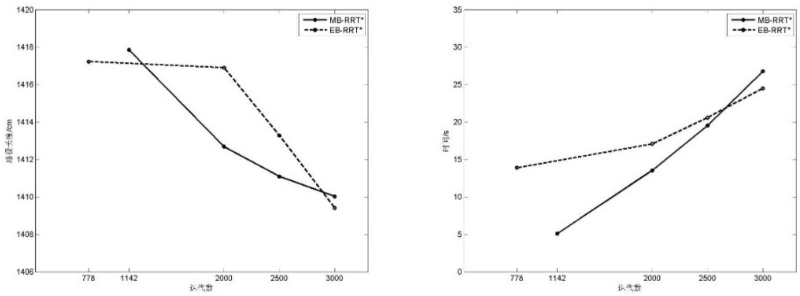


Fig.23 Time and path length in Map3

**Conclusions and future work**

UAV automatic navigation capacity is an essential function with wider application of UAV, so this paper presents a detailed comparative analysis of performance of our proposed EB-RRT* algorithm with the existing algorithms MB-RRT* and B-RRT*. Three novel strategies are brought up for speeding up convergence rate and navigation accuracy for UAV. First of all, self-avoidance is adopted to improve convergence speed and less memory cost. And then grid partitioning is applied to shorten the time of finding the nearby vertices. Finally smoothing turning point improves the convergence rate of the algorithm and the smoothness of the final path. Hence, we anticipate employing EB-RRT* for online motion planning of animated characters in complex 3-D environments.

**Acknowledgement**

The authors would like to thank you for the support of foundation research project of Zhejiang province for research institute titled Bridge quality and security detection research and applications based on UAV（2016F50047）. This work was supported by a grant from the National Natural Science Foundation of China (No. 61502423), Zhejiang Provincial Natural Science Foundation (Y14F020092).

References

[1] KP Valavanis. Advance in unmanned aerial vehicles[M]. State of art and road to autonomy, 2007.
[2] A De, J Caves. Human-automation collaborative RRT for UAV mission path planning[M]. Massachusetts Institute of Technology, 2010.
[3] HH Triharminto., AS Prabuwono. UAV Dynamic Path Planning for Intercepting of a Moving Target: A Review[J]. Communications in Computer and Information Science, 2013, 376:206-219.
[4] JH Holland. Adaptation in natural and artificial systems[M]. MIT Press, 1992.
[5] Roberge V., Tarbouchi M., Labonte G. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real Time UAV Path Planning[J]. IEEE Transactions on Industrial Informatics, 2013, 9(1):132-141.
[6] L Kavraki., P Svestka. Probabilistic roadmaps for path planning in high-dimensional configuration spaces[J]. IEEE Transactions on Robotics & Automations, 1996, 12(4):566-580.
[7] SM Lavalle., JJ Kuffner. Randomized Kinodynamic Planning[J]. IEEE International Conference on Robotics & Automation, 1999, 1(5):473-479.
[8] SR Lindemann., SM LaValle. Incrementally reducing dispersion by increasing voronoi bias in

rrts[J]. IEEE International Conference on Robotics & Automation, 2004, 4(4):3251-3257.
[9] Xiong J, Hu Y, Wu B, et al. Minimum-cost rapid-growing random trees for segmented assembly path planning[J]. The International Journal of Advanced Manufacturing Technology, 2015, 77(5):1043-1055.
[10] Yang K. An efficient Spline-based RRT path planner for non-holonomic robots in cluttered environments[C]// International Conference on Unmanned Aircraft Systems. 2013:288-297.
[11] AH Qureshi., S Mumtaz., KF Iqbal., Y Ayaz. Triangular geometry based optimal motion planning using RRT*-motion planner[J]. IEEE International Workshop on Advanced Motion Control, 2014, 380-385.
[12] Jordan M, Perez A. Optimal Bidirectional Rapidly-Exploring Random Trees[J]. 2013.
[13] Abbadi A, Prenosil V. Collided Path Replanning in Dynamic Environments Using RRT and Cell Decomposition Algorithms[M]// Modelling and Simulation for Autonomous Systems. Springer International Publishing, 2015:131-143.