基于EB-RRT*的无人机航迹规划算法研究

陈晋音 李玉玮 杜文耀
(浙江工业大学信息工程学院 杭州 310023)

摘 要 随着小型无人机的广泛应用,无人机的自动巡航能力变得至关重要。无人机航迹规划,是指其在已知环境地图信息下展开航迹规划以实现无碰撞地、平滑地从初始点到达目标点。针对现有算法依然存在收敛速度慢、遍历时间长和航迹曲线化无法满足实际无人机飞行条件等问题,首先提出了EB-RRT*(Efficient B-RRT*)算法,设计自适应避障提高算法收敛速度并减少内存占用;然后采用栅格分区的方法缩短附近节点的遍历时间;最后利用合理的降采样和三次贝塞尔插值算法对折点进行光滑化的处理,使算法最终生成相对平滑的航迹,为无人机实际飞行提供可行航迹规划方法。进行了多组不同环境复杂度的实验,并将该算法与其他算法进行对比,结果验证了所提算法的有效性。

关键词 RRT,无人机,航迹规划,收敛速度,航迹优化
中图法分类号 TP242 **文献标识码** A

UAV Navigation Algorithm Research Based on EB-RRT*

CHEN Jin-yin LI Yu-wei DU Wen-yao
(College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract With the wide application of UAV, automatic navigation capability of UAV becomes more important. UAV navigation algorithm was defined to plan a collision free and smooth path from start position to destination position in known maps. Aiming at three problems of current navigation algorithms including low convergence rate, long searching time and navigation path couldn't be applied to real UAV, EB-RRT* (Efficient B-RRT*) algorithm was proposed. A self-adaptive obstacle avoidance strategy was designed to speed up convergence rate of traditional navigation algorithm by reducing memory cost. Grid based segmentation mechanism was put forward to reduce searching time for path planning. Suitable down sampling and three times Bessel interpolation formula were adopted to smooth final path for practical UAV applications. Several simulation maps were used to testify the performances of proposed algorithms compared with other classic algorithms.

Keywords RRT, UAV, Navigation algorithm, Convergence rate, Path optimization

1 引言

航迹规划是无人机导航和机器人技术中的一个重要问题,其基本定义为:给定一个初始状态和目标状态,寻找一个可行航迹使得无人机无碰撞地从初始状态运行到目标状态。航迹规划具有广泛的应用场景:GPS导航、无人驾驶汽车、计算机动画、路由问题、制造行业的机械臂运动和生活以及工业领域的很多方面。因此,对航迹规划问题的研究成为近年来的一个研究热点。

根据感知能力,无人机航迹规划算法可以分为局部航迹规划和全局航迹规划。全局航迹规划就是在已知环境地图的情况下进行规划,预先知道环境的全局信息;而局部航迹规划只需要获得机器人感知范围内的环境信息,主要指障碍物信息,根据局部信息完成规划。全局航迹规划算法有很多,人工势场算法是典型的航迹规划算法[1],算法在环境中建立人工势场,障碍物和环境边界具有斥力,目标区域具有引力,无人机根据所受力向目标区域靠近。势场法不需要进行复杂的计算,只需要计算出环境的势场即可;然而,势场法在复杂环境中容易使飞行器陷入局部最小,并不适合在复杂环境以及狭窄的通道中进行规划。针对势场法的不足,有研究者提出了Dubins曲线算法以及细胞分裂算法[2]和Delaunay三角算法[3]等离散化搜索空间的算法,这些算法通过对障碍物或者环境空间进行建模的方法寻找最优航迹。同时,也有人将进化计算(如遗传算法[4]、粒子群算法[5])用于解决航迹规划问题,利用算法的进化操作和迭代过程找到最优航迹。然而,这类算法的计算开销特别大,算法在复杂的环境以及高维数的环境中需要大量的计算时间,无法直接应用在无人机的航迹规划应用中。

基于采样的航迹规划算法已被证明可以高效地解决航迹规划问题[6],概率路线图算法(PRM)[7]和快速扩展随机数算法(RRT)[8]是目前两种主要的采样算法。PRM算法随机在空间生成采样点,并且对这些点进行连接,最后通过图搜索算法找到初始状态到目标区域的航迹。与RPM算法相比,RRT算法采用树结构描述碰撞检测的次数,且树的航迹搜索

本文受浙江省自然科学基金(Y14F020092),国家自然科学青年基金(61502423),浙江省科技厅科研院专项(2016F50047)资助。
陈晋音(1982-),女,副教授,主要研究方向为无人机航迹规划、数据挖掘、智能计算等;李玉玮(1995-),男,主要研究方向为无人机航迹规划;杜文耀(1990-),男,硕士生,主要研究方向为无人机控制、航迹规划。

比图的航迹搜索更容易实现。然而，RRT 算法的收敛率太低，即需要通过大量的迭代才能找到最优航迹，并且随着迭代次数的增加，算法也需要大量的内存。因此，人们目前提出了很多针对 RRT 算法的变种算法以及改进算法。Nik A 通过将粒子滤波与 RRT 算法结合，提出了 PRRT 算法[9]用于局部航迹规划；Stephen R 将泰森多边形（Voronoi）引入树的生长中，以提高 RRT 找到可行解的速度[10]。其中应用最广并且效果最好的是 Sertac Karaman 提出的 RRT* 算法[11]。RRT* 算法在每次迭代后对新加入的节点及其邻近节点进行优化，这一优化操作改善了算法的收敛率，保证了算法的渐近最优性，从而使其广泛应用在航迹规划领域并衍生出一系列变种算法。A. H. Qureshi 为了加快 RRT* 算法的收敛速度[12]，在生成随机点的同时，将随机点、目标点和初始位置 3 个点构成的三角形的内心作为新的随机点加入树中，使随机点一定程度上偏向了目标点；M. Jordan[13]提出了利用两棵树生长寻找航迹的方法提高了算法的收敛率；X Zhang 等人[14]提出了基于自学习策略和混合偏差抽样方案来提高规划效率；K Baizid 等人[15]在无人机航迹规划的安全性上展开研究，提出优化策略 RRS；在此基础上，通过对这些算法进行优化来解决实际问题。J Xiong 等人[16]基于 RRT 解决分段装配路径规划任务；K Yang 等人采用有效的样条曲线参数化来代替系统动力学的数值积分，保证了路径曲率的连续性[17]。这些改进算法在实现无人机的航迹规划时，仍然存在以下问题：

(1)算法的收敛速度还有很大的提升空间；

(2)算法的节点在障碍物附近地生长不具有灵活性；

(3)算法寻找附近节点集时需要进行大量的遍历操作，因此算法的运行时间较长；

(4)由于算法的航迹由树节点连接生成，最后生成的航迹不够平滑，难以满足无人机的实际应用需求。

针对这些问题，本文提出了一种 EB-RRT*（Efficient B-RRT*）算法，设计自适应避障来提高算法的收敛速度并减少内存占用；采用栅格分区的方法缩短附近节点的遍历时间；然后利用合理的降采样和三次贝塞尔插值算法对折点进行光滑化处理，使算法最终生成相对平滑的航迹，为无人机实际飞行提供可行航迹规划方法；最后通过相关实验验证了算法的有效性，并与其他算法进行比较来验证 EB-RRT* 的性能。

2　相关研究

2.1　RRT

S. LaValle 和 J. Kuffner 首先提出了 RRT 算法，且证明了 RRT 算法具有概率完备性，同时可以高效地解决航迹规划问题。RRT 算法首先在无障碍状态空间中进行随机采样，生成采样点后寻找树中距离采样点最近的节点，并对这两点之间的连线进行碰撞检测，判断连线中是否有障碍物，如果没有则将这个新的采样点加入树中；通过不断地迭代生长树直到树生长到目标区域，最后生成最优航迹。

由于 RRT 算法的采样是随机采样，使得算法产生的树节点随着迭代次数的增加覆盖到整个地图区域，保证了算法的概率完整性，而且算法只需要采样、碰撞检测、连接这 3 个过程，不需要进行过于复杂的计算。然而，RRT 算法的缺点也很明显：因为算法需要经过多次迭代才能找到最优解，所以算法的收敛率太低，且算法在插入随机采样点之后并没有其他修正操作，导致算法需要进行大量的迭代才能找到相对最优的航迹，这需要消耗大量的时间和内存。

2.2　RRT*

针对 RRT 算法收敛率低等问题，S. Karaman 和 E. Frazzoli 对 RRT 算法的连接过程进行了改进，提出了 RRT* 算法。与 RRT 算法的直接连接过程不同，RRT* 算法在添加节点的同时检测采样点周围的节点，判断是否在加入新节点后有更短的航迹，有则以新节点替换旧节点，从而对航迹进行了修正。

算法的主要流程如算法 1 所示。

算法 1　RRT*（$x_{star}t$，x_{goal}）

```
1. V ←x_start；E ← Ø；T← (V,E)；i ←0
2. while i<N do
3.    x_rand← Sample(i)
4.    x_nearst← NearstNode(x_rand，T)
5.    x_new← Steer(x_nearst，x_rand)
6.    if Collision Check(x_nearst，x_new) then
7.       T ← InsertNode(x_new)
8.       X_near← NearNodes(x_new，T)
9.       x_parent← ChooseBestParent(x_new，X_near，T)
10.         OptimizeVertices(x_parent，x_new，T)
11.   end if
12. end while
```

Sample：该函数在无障碍环境中随机生成一个状态点。

NearestNode：给定一个状态点 $x\in X_{free}$ 以及一个树 $T=(v,E)$，函数 NearestNode 在树 T 中搜索与点 x 的欧氏距离距离最近的节点 x_{nearst} 并返回。

Steer：给定两点 x_{nearst} 和 x_{rand}，函数 Steer 返回点 x_{new}，x_{new} 满足 $\| x_{new}-x_{nearst} \|=\mu$ 且使 $\| x_{new}-x_{rand} \|$ 最小，μ 为树生长的步长。

NearNodes：给定一个状态点 $x\in X_{free}$ 以及一个树 $T=(v,E)$，函数 NearNodes 返回一个与点 x 的欧氏距离小于 γ 的点集。其中，γ 的大小为 $k(\log n/n)^{1/d}$，k 为常数，n 为算法的迭代次数，d 为所在环境的维数。这一过程可以简化为：

$$V_1=\{x_{near}\in v:d(x,x_{near}\leqslant\gamma)$$

CollisionCheck：给定两个点 x_{new} 和 x_{near}，函数 CollisionCheck 对两点之间的连线航迹进行障碍检测，若两点之间无障碍则返回 True，否则返回 False。

算法 2　ChooseBestParent（x_{new}，X_{near}，T）

```
1. x′= x_nearst
2. for all x_near∈X_near do
3.    if Collision Check(x_near，x_new) then
4.       c′= c(x_start，x_near) + c(x_near，x_new)
5.       if c′<c(x_new) then
6.          x′= x_near
7.          c(x_new) = c′
8.       end if
9.    end if
10. end for
11. return x′
```

算法 3　OptimizeVertices（x_{parent}，X_{near}，T）

```
1. for all x_near∈X_near not x_parent do
2.    if Collision Check(x_near，x_new) and c(x_start，x_new)+c(x_new，x_near)<
      c(x_near) then
3.       x_p← Parent(x_near)
```

4. T ← DeleteEdge(x_p, x_{near})
5. T ← AddEdge(x_{new}, x_{near})
6. end if
7. end for

RRT* 算法解决了 RRT 算法生成航迹不是最优的问题，保留了 RRT 算法的概率完整性的特点并加快了收敛速度，即可以通过更少的迭代次数找到最优解。RRT* 算法仍然有很多待解决的问题：首先，RRT* 收敛到最优解的速率太慢，算法寻找最优解仍需进行大量的迭代过程，这也导致 RRT* 算法在迭代求解的过程中需要消耗大量的内存；同时，RRT* 算法在面对复杂的地图（如通道和迷宫）时需要进行大量的迭代才能找到最优解。

2.3 B-RRT*

针对 RRT* 算法的问题，不少学者对其进行了研究并提出了一系列解决方案。A. H. Qureshi 提出了 TG-RRT* 算法，这一算法较 RRT* 算法有更高的收敛率，但却在一定程度上损失了算法的概率完整性，且使树的生长具有一定偏向性，容易使算法陷入局部最优的情况。M. Jordan 和 A. Perez 提出的 B-RRT* 算法在提升了算法收敛率的同时，保留了算法的概率完整性以及渐近最优性。B-RRT* 算法在 RRT* 算法的基础上引入了双向树结构，即从初始点和目标点同时生长两棵树，在加入新的节点之后进行一次连接检测，判断两棵树能否相连。B-RRT* 算法的流程如算法 4 所示。

算法 4 B-RRT* (x_{start}, x_{goal})

1. E ←∅; T_a ← (x_{start}, E); T_b ← (x_{goal}, E)
2. i ← 0; θ_{best} ← ∞
3. while i < N do
4. x_{rand} ← Sample(i)
5. x_{nearst} ← NearstNode(x_{rand}, Ta)
6. x_{new} ← Steer(x_{nearst}, x_{rand})
7. if Collision Check(x_{nearst}, x_{new}) then
8. T_a ← InsertNode(x_{new})
9. X_{near} ← NearNodes(x_{new}, T_a)
10. x_{parent} ← ChooseBestParent(x_{new}, X_{near}, T_a)
11. OptimizeVertices(x_{parent}, x_{new}, T_a)
12. xmid ← NearstNode(x_{new}, T_b)
13. θ' ← Connect(x_{new}, x_{mid}, μ)
14. if $\theta' < \theta_{best}$ then
15. $\theta_{best} = \theta'$
16. end if
17. end if
18. SwapTrees(T_a, T_b)
19. end while

可以总结得到：算法采用双向树的结构需要额外的 Connect 函数对两棵树进行连接。Connect 函数应用了 RRT* 树寻找新节点时用的策略，调用了 NearNodes 函数在范围内对另一棵树的节点进行搜索计算，从邻近节点中搜索最优的连接节点并进行连接。

算法 5 Connect(x_{new}, x_{mid}, μ)

1. $\theta = \infty$
2. if c(x_{new}, x_{mid}) < μ then
3: θ = c(x_{start}, x_{new}) + c(x_{new}, x_{mid}) + c(x_{mid}, x_{goal})
4. end if
5. return θ

2.4 MB-RRT*

针对 B-RRT* 算法收敛速度较慢、固定步长和航迹平滑度不满足实际要求等问题，施晋、杜文耀和吴洋洋提出了 MB-RRT* 算法，其采用了懒惰采样、自适应步长、降采样和三次贝塞尔插值算法。算法的流程如算法 6 所示。

算法 6 MB -RRT* (x_{start}, x_{goal})

1. E ←∅; T_a ← (x_{start}, E); T_b ← (x_{goal}, E)
2. i ← 0; θ_{best} ← ∞
3. while i < N do
4. x_{rand} ← Sample(i)
5. x_{nearst} ← NearstNode(x_{rand}, T_a)
6. x_{new} ← AutoStepSteer(x_{nearst}, x_{rand})
7. if c(x_{nearst} + c(x_{nearst}, x_{new})) > θ_{best} then
8. Continue
9. end if
10. if CollisionCheck(x_{nearst}, x_{new}) then
11. T_a ← InsertNode(x_{new})
12. X_{near} ← NearNodes(x_{new}, T_a)
13. x_{parent} ← ChooseBestParent(x_{new}, X_{near}, T_a)
14. OptimizeVertices(x_{parent}, x_{new}, T_a)
15. x_{mid} ← NearstNode(x_{new}, T_b)
16. θ' ← Connect(x_{new}, x_{mid}, μ)
17. if $\theta' < \theta_{best}$ then
18. $\theta_{best} = \theta'$
19. end if
20. end if
21. SwapTrees(T_a, T_b)
22. end while
23. θ_{best} ← DownSample(θ_{best})
24. θ_{best} ← BezierCurve(θ_{best})

可以看到，懒惰采样的方法通过跳过对无用节点进行碰撞检测，寻找距离其最近的节点，存储其附近的节点集等步骤来减少内存占用；自适应步长算法根据 $StepSize = \frac{D_i}{\mu_{Max}} \mu_{Min}$ 计算出障碍物附近节点的生长步长，解决在障碍物附近生长的局限性问题；算法最后的降采样和三次贝塞尔插值实现了曲线拟合的功能。

3 基于 EB-RRT* 的航迹规划算法

3.1 主要思想

虽然 MB-RRT* 算法在 B-RRT* 的基础上通过引入自适应步长和懒惰采样等方法有效地减少了采样时间和采样点数，但是还可以在此基础上提高算法的运行效率。1）采用自适应步长虽然能在障碍物附近缩短生长步长，但仍然有大量的点因为碰撞检测而被抛弃，导致无法探索新的空间来提高算法速度；2）保留 RRT* 算法的概率完备性，使得算法在每次加入新节点后都需要对每个旧节点遍历一次，导致每次迭代时间不断增加；3）MB-RRT* 算法引入的贝塞尔曲线优化在狭长通道或者障碍物较密集的地图中容易产生碰撞。

因此，本文提出 EB-RRT* 算法，分别针对新生长点、附近节点、曲线拟合等方面加入自适应避障、栅格分区、折点光滑等方法进行优化。

3.2 算法步骤

本文提出的 EB-RRT* 算法与 MB-RRT* 算法的流程图对比以及 EB-RRT* 算法的伪代码如下。

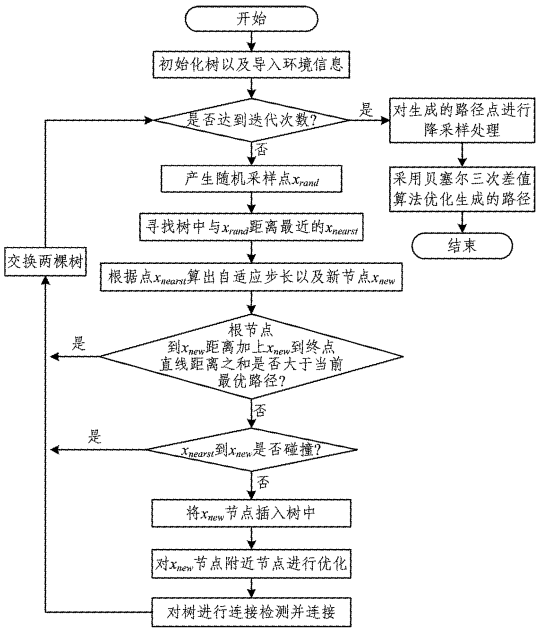


图 1　MB-RRT* 算法流程图

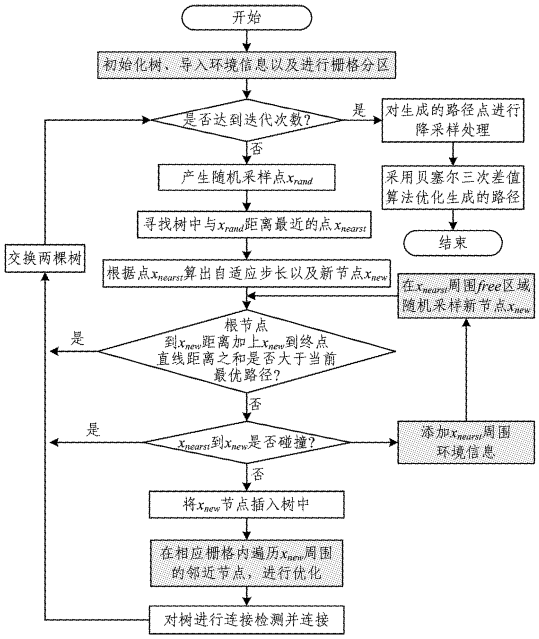


图 2　EB-RRT* 算法流程图

算法 7　EB-RRT* (x_{start}, x_{goal})

1. E ← ∅; T_a ← (x_{start}, E); T_b ← (x_{goal}, E)
2. i ← 0; θ_{best} ← ∞
3. while i < N do
4. x_{rand} ← Sample(i)
5. x_{nearst} ← NearstNode(x_{rand}, T_a)
6. x_{new} ← Steer(x_{nearst}, x_{rand})
7. if c(x_{nearst} + c(x_{nearst}, x_{new})) > θ_{best} then
8. Continue
9. end if
10. if !CollisionCheck(x_{nearst}, x_{new}) then
11. x_{obs} ← NearstObs(x_{nearst})
12. x_{new} ← Steer(x_{nearst}, x_{obs})
13. end if
14. T_a ← InsertNode(x_{new})
15. Xnear ← NearNodes(x_{new}, T_a)
16. x_{parent} ← ChooseBestParent(x_{new}, X_{near}, T_a)
17. OptimizeVertices(x_{parent}, x_{new}, T_a)
18. x_{mid} ← NearstNode(x_{new}, T_b)
19. θ' ← Connect(x_{new}, x_{mid}, μ)
20. if $\theta' < \theta_{best}$ then
21. $\theta_{best} = \theta'$
22. end if
23. SwapTrees(T_a, T_b)
24. end while
25. θ_{best} ← DownSample(θ_{best})
26. θ_{best} ← SmoothPoint(θ_{best})

从流程图和伪代码中可以看出，EB-RRT* 增加了没有通过障碍检测后的补救措施，而不是像 MB-RRT* 那样直接跳过本次迭代。另外，在搜索某一节点附近的点集时，不再对已插入的所有旧节点进行遍历，而是对所在区域及其附近区域内的节点进行遍历，有效减少了算法的运行时间。下面对加入的 3 个方法过程进行详细说明。

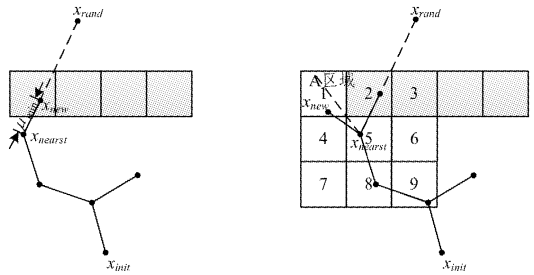
3.2.1　自适应避障

MB-RRT* 以及传统的 B-RRT* 等算法首先通过 Sample 函数产生一个随机点 x_{rand} 后，再通过 NearstNode 寻找树 T 旧节点中距离 x_{rand} 最近的节点 x_{nearst}，以矢量 $\overrightarrow{x_{nearst}x_{rand}}$ 为方向生长。

虽然 MB-RRT* 采用自适应步长的方法使得在障碍物附近生长的步长保持在范围 $\mu_{Min} < \mu < \mu_{Max}$，但若 x_{rand} 在障碍物周围 μ_{Min} 范围内，则仍然会导致生成的新节点 x_{new} 进入障碍物范围而被舍弃，使该次迭代未插入新的节点。这种情况在狭长通道以及障碍物比较密集的空间下发生的概率较高。

本文提出的 EB-RRT* 算法采用的自适应避障方法能够避免这种不必要的迭代次数的发生，使生长树每次都能探索到新的空间，增大了两棵树连接的可能性，提高了算法找到可行路径的效率。

图 3 给出 MB-RRT* 和 EB-RRT* 在靠近障碍物的情况下的生长过程。



(a) MB-RRT* 在障碍附近的生长情况　　(b) EB-RRT* 在障碍附近的生长情况

图 3

图 3(a)为 MB-RRT* 的自适应步长的生长方法，可以看到在距离障碍物的长度小于最短步长 μ_{Min} 时，仍然会导致新节点 x_{new} 处于障碍物范围内而被舍弃。图 3(b)为 EB-RRT* 采用自适应避障方法，在原 x_{new} 处于障碍物范围内时，在节点 x_{nearst} 处将周围环境划分为 9 个区域，分别标记为 1－9，自身所在点为区域 5，通过函数 AddInform 对其他 8 个区域进行信息收集，区分为障碍区和非障碍区，图中 2 和 3 为障碍区，1，4，6，7，8，9 为非障碍区；函数 ChooseDirection 在剩余的 6 个非障碍区内随机取一点作为新 x_{new}，再在 x_{nearst} 和 x_{new} 之间进行障碍检测，防止在 A 区域内的点作为新 x_{new}。

与引入自适应步长方法的 MB-RRT* 算法相比，采用了

自适应避障方法后的 EB-RRT* 算法扩大了障碍物周围的探索范围，增强了避障的能力，因此有效地减少了找到初次可行路径的迭代次数和运行时间。

3.2.2 栅格分区

继承了 RRT* 概率完备性的 B-RRT* 和 MB-RRT* 算法，在双向树生长过程中，每次插入一个新的节点 x_{new} 后，需要通过函数 NearNodes 寻找一个与其欧氏距离小于 γ 的点集 $V_1 \in V$，再对该点集中的所有点判断是否需要进行航迹修正。在这一过程中，每次寻找点集都需要对生长树 T 中的所有旧节点进行遍历。然而，随着搜索范围的扩大，插入的节点也不断增多，遍历的时间也会呈线性增长。

针对这个问题，EB-RRT* 引入的栅格分区方法能够在保留概率完备性的基础上极大地缩短遍历时间，以此来提高算法的运行效率。

为了减少加载地图信息后初始化需要的准备时间，只对二维地图进行简单的栅格划分，不考虑障碍信息，由 Ahmad Abbadi 和 Vaclav Prenosil 提出的细胞分解算法(Cell Decomposition Algorithms)[18]可将整个地图空间分为障碍区和非障碍区。图 4 将宽为 W、高为 H 的地图空间划分为 $m * n$ 个宽为 W_{grid}、高为 H_{grid} 的区域，其中：

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

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

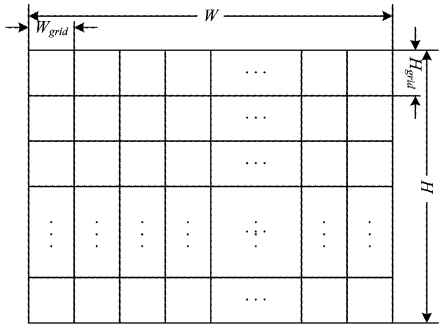


图 4 栅格分区

在插入新节点 x_{new} 后，只需要遍历 x_{new} 所在区域及其周围 8 个区域内的所有旧节点即可。为了保证与其欧氏距离小于 γ 的所有点均在这 9 个区域内，须满足：

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

又因为

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

所以

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

只需要满足

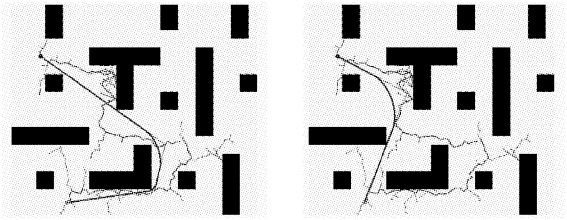
$$W_{grid} \geqslant k(\log 2/2)^{1/d}$$

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

实际中，为了考虑内存的开销并便于操作，可以将 W_{grid} 和 H_{grid} 设为较大值，使得 m 和 n 恰好为整数值。

3.2.3 折点光滑

MB-RRT* 算法对最后生成的最优航迹轨迹点进行降采样的过程虽然能够使最后的轨迹点尽量少，但是存在图 5(a)的这种情况，图中黑色块状部分为障碍区，细折线部分为以 x_{start} 为根节点的起点生长树 T_{start} 和以 x_{goal} 为根节点的终点生长树 T_{goal}，粗折线为降采样前后的轨迹。通过图 5(a)和图 5(b)的对比可以看到，在被删除的轨迹点中存在一点 x_i，其与终点 x_{goal} 相连且能够通过碰撞检测，而不需要经过 $x_{i+1}, \cdots, x_{goal-1}$，且图 5(b)中的最终轨迹远小于图 5(a)中的航迹。



(a)MB-RRT* 降采样效果 (b)EB-RRT* 降采样效果

图 5

因此，EB-RRT* 算法采用的降采样算法可以在减少轨迹点的基础上适当缩减初始轨迹的长度，达到优化效果。两者的伪代码如算法 7 和算法 8 所示。EB-RRT* 采用的降采样算法是在 MB-RRT* 的基础上增加了对初始轨迹点的遍历，以确定是否存在点 x_i，使得其与起点或者终点相连能够通过障碍检测来判断是否删除多余轨迹点。

MB-RRT* 采用的降采样算法如算法 8 所示。

算法 8 DownSample(θ_{best})

```
1. span = 2
2. for span<θbest. size() do
3.    flag = false
4.    i = 0
5.    for span + i <θbest. size() do
6.       i = i + 1
7.       if CollisionCheck(θbest[i],θbest[i + span]) then
8.          t = 1
9.          for t<span do
10.            t = t + 1
11.            θbest. erase(i + 1)
12.         end for
13.         flag = true
14.      end if
15.   end for
16.   if flag == false then
17.      span = span + 1
18.   end if
19. end for
20. return θbest
```

EB-RRT* 采用的降采样算法如算法 9 所示。

算法 9 DownSample(θ_{best})

```
1. for i< θbest. size() do
2.    if CollisionCheck(θbest[start],θbest[i]) then
3.       for x∈(1,i−1) do
4.          best. erase(x)
5.       end for
6.    end if
7.    if CollisionCheck(θbest[i],θbest[goal]) then
8.       for x∈(i+1,goal) do
9.          θbest. erase(x)
10.      end for
11.   end if
12.   end for
13. span = 2
14. for span<θbest. size() do
15.   flag = false
16.   i = 0
```

```
for span + i < θbest.size() do
    i = i + 1
    if CollisionCheck(θbest[i], θbest[i + span]) then
        t = 1
        for t < span do
            t = t + 1
            θbest.erase(i + 1)
        end for
        flag = true
    end if
end for
if flag == false then
    span = span + 1
end if
end for
return θbest
```

为了使经过降采样后的折线轨迹在转折点处变得光滑，不同于 MB-RRT* 算法中选取控制点的方式，EB-RRT* 采用贝塞尔曲线拟合。图 6 给出在转折点处三阶贝塞尔曲线两个端点和两个控制点的选取过程，最后拟合成三阶贝塞尔曲线。其中，X_i 为某一转折点，线段 $\overline{P_0X_i}=\overline{X_iP_3})=d$，$\overline{P_1X_i}=\overline{X_iP_2})=\alpha d$。再根据三次贝塞尔曲线方程：

$$B(t)=P_0(1-t)^3+3P_1t(1-t)^2+3P_2t^2(1-t)+P_3t^3,\ 0<t<1$$

计算出曲线上的点，即可生成曲线。

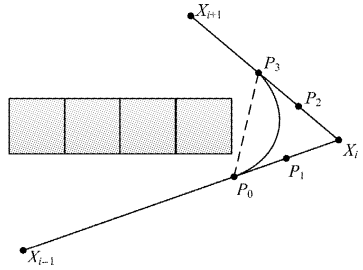


图 6　曲线取点拟合过程

可见，d 越大，该转折点处的路径长度就越短，同时又要满足路径的可行性，因此只需要线段 $\overline{P_0\ P_3}$ 通过碰撞检测，那么生成的曲线也不会碰到障碍物，保证了轨迹的安全性和可行性。

4　EB-RRT* 算法的性能分析

4.1　概率完备性

在任意环境中，如果算法在航迹可行解存在的情况下，运行迭代次数趋于无穷大时能够找到可行解，那么称该算法具有概率完备性（Probability Completeness）。MB-RRT* 算法保留了 RRT* 算法的概率完备性。本文提出的 EB-RRT* 算法采用的自适应避障和栅格分区方法只是通过加快障碍物周围的探索速度和寻找 x_{new} 节点附近点集的遍历速度来提高算法的运行效率，并不对算法的概率完备性产生影响。因此，算法依旧保留着 *RRT** 算法的概率完备性。

4.2　渐进最优性

渐近最优（Asymptotic Optimality）在航迹规划问题中的定义为：令 C^* 为当前环境下航迹规划的最优解，Y_n^{ALG} 为算法 ALG 在迭代到 n 次之后产生的最优航迹解的长度，算法 ALG 满足渐近最优性当：

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

已经知道，SertacKaraman 和 Emilio Frazzoli 证明了 RRT* 算法具有渐近最优性，同时 Matthew Jordan 和 Alejandro Perez 证明双向树版本的 B-RRT* 算法具有渐近最优性。MB-RRT* 算法采用与 B-RRT* 相同的连接过程，和 RRT* 算法加入新节点的过程类似，即上文提到的 Connect 函数，其算法的单棵树的生长过程又继承 RRT* 算法，同时引入了自适应步长，而 Sertac Karaman 和 Emilio Frazzoli 提出 RRT* 算法的渐近最优性适用于步长 $\eta>0$ 的任意步长，因此 MB-RRT* 算法也继承了 RRT* 算法的渐进最优性。同样的，本文提出的 EB-RRT* 算法的连接过程与 B-RRT* 也类似，也满足步长 $\eta>0$ 的条件，所以也具有相同的渐近最优性。

4.3　时间复杂度

EB-RRT* 和 MB-RRT* 算法每次迭代需要执行的 Sample，CollisionCheck，OptimizeVertices，Connect 等函数的运行时间与迭代次数没有关系。MB-RRT* 中 AutoStepSteer 函数需要对数时间 $\Omega(\log\ n)$ 运行，而 EB-RRT* 中沿用的 RRT* 和 B-RRT* 中的 Steer 函数需要的时间为常数值。EB-RRT* 中只有在 x_{new} 未通过障碍检测后才会调用 AddInform，ChooseDirection 和 Toward，AddInform 和 ChooseDirection函数需要的时间为常数，Toward 函数中多调用了一次 Steer 和 CollisionCheck 函数，也为常数。另外，EB-RRT* 中的 AreaNearNodes 函数所用的时间大约为 NearNodes 函数的 $1/(m*n)$。

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

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

其中，P 为旧 x_{new} 节点在障碍物中的概率。

因此，存在一个常数 ϕ，使得

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

5　仿真实验

二维环境下的航迹规划演示与优化在 Ubuntu 系统下进行，并利用 Qt 完成算法在二维环境下的可视化演示。表 1 列出本实验使用的硬件配置。

表 1　实验硬件

类型	参数
处理器	Intel(R)Core(TM)i3-2310M 2.10GHz*4
系统版本	Ubuntu 14.04LTS
内存	5.7GB

5.1　实验地图

二维环境下的实验环境大小为 800*600，通过在环境中放置不同的障碍物生成了 4 幅难度不同的地图，并进行了算法验证。地图的相关参数如表 2 所列。

表 2　实验地图的参数

地图 \ 参数	起点坐标	终点坐标	障碍物数量	障碍物占空比
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

5.2 实验结果与分析

本节介绍 EB-RRT* 算法、MB-RRT* 算法和 B-RRT* 算法在不同地图环境下的运行情况。为了使对比效果更明显，将 MB-RRT* 与 B-RRT 算法的运行结果图进行展示对比，其中图 7－图 12 给出两算法初次找到可行解的节点分布图，图(a)为 EB-RRT* 算法，图(b)为 MB-RRT 算法。在测试 EB-RRT* 算法过程中，W_{grid} 取值为 100，H_{grid} 取值为 100，所以将 800 * 600 的地图环境划分为 48 个区域来缩短遍历时间。

图中黑色块状部分为障碍区，细折线部分为以 x_{start} 为根节点的起点生长树 T_{start} 和以 x_{goal} 为根节点的终点生长树 T_{goal}，粗折线为降采样前后的轨迹。

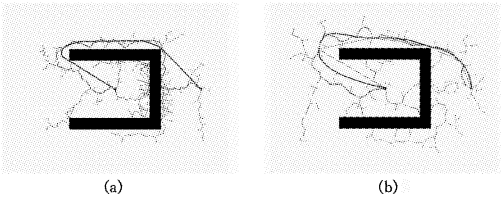


图 7 Map1 中初次找到可行解的运行结果

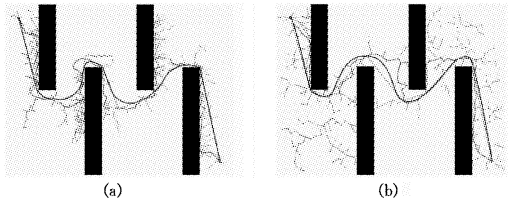


图 8 Map2 中初次找到可行解的运行结果

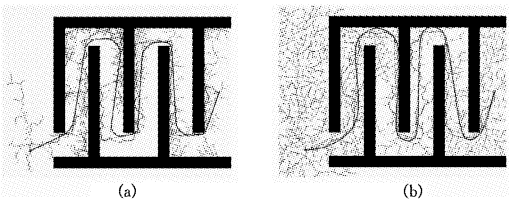


图 9 Map3 中初次找到可行解的运行结果

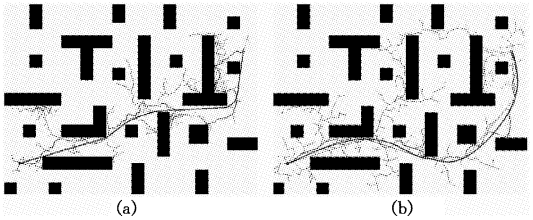


图 10 Map4 中初次找到可行解的运行结果

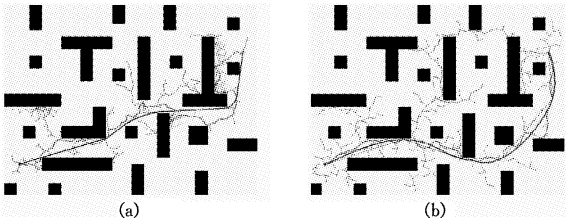


图 11 Map5 中初次找到可行解的运行结果

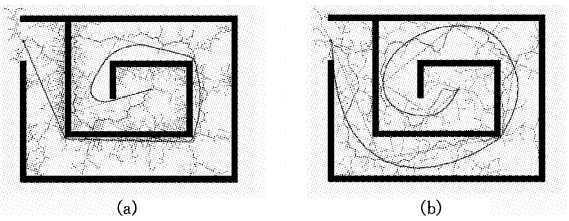


图 12 Map6 中初次找到可行解的运行结果

通过 EB-RRT* 和 MB-RRT* 算法在 Map1－Map4 的地图环境下的运行结果对比图，可以看到 EB-RRT* 在障碍物物附近生长得比较密集，这说明在大量的迭代次数中，障碍物附近的点被选中为 x_{nearst}；MB-RRT* 因未通过障碍检测而舍弃，白白消耗了寻找 x_{nearst} 的开销，而 EB-RRT* 采用的自适应避障方法，使得该次迭代能够插入新的节点，提高了找到可行解的效率；EB-RRT* 在最后的曲线拟合时只对转折点进行光滑处理，有效地解决了拟合后的曲线实际不可行的问题。

表 3 Map1－Map6 上的实验结果

指标	算法	Map1	Map2	Map3	Map4	Map5	Map6
初次找到可行解的迭代数	B-RRT*	428	769	4486	444	2254	1508
	MB-RRT*	227	686	3385	357	1954	1372
	EB-RRT*	253	277	762	231	368	837
初次找到可行解的路径长度	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
初次找到可行解的运行时间	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
迭代 1000 次所得解	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
迭代 2000 次所得解	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
迭代 3000 次所得解	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
迭代 4000 次所得解	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

表 3 列出 EB-RRT*，MB-RRT* 和 B-RRT* 在地图环境 Map1－Map6 上运行的精确数据，初次找到的可行迭代数据为每幅地图分别运行 10 次算法取平均值所得。可以看到，EB-RRT* 在加入自适应避障和栅格分区方法后初次找到可

行解的时间远少于 MB-RRT* 和 B-RRT* 所用的时间，并且最后所得的路径长度与 MB-RRT* 的结果相近。已知 MB-RRT* 得到的是高质量的可行解，而 EB-RRT* 算法能够在更短的时间内得到结果，因此效率比较高。

图 13—图 15 为根据表 3 数据中 3 种算法初次找到可行解的迭代数、路径长度和时间的柱状图。很明显可以看出，EB-RRT* 的迭代数和时间远远小于 MB-RRT* 和 B-RRT* 的运行结果；而且，EB-RRT* 最后生成的路径长度与 MB-RRT* 相近，并没有因为时间的减少而得到较差的可行解。以上说明了 EB-RRT* 的运行效率远高于 MB-RRT* 和 B-RRT* 。

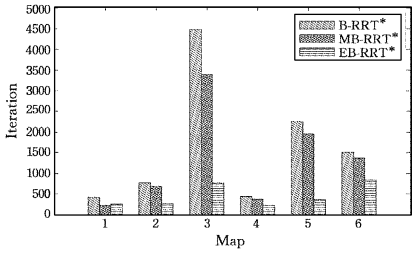


图 13　初次找到可行解的迭代数对比图

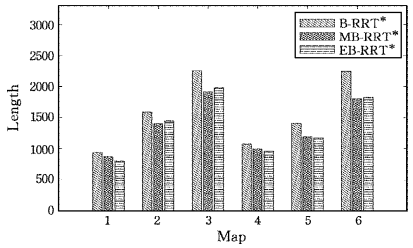


图 14　初次找到可行解的路径长度对比图

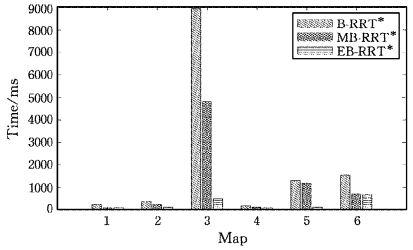


图 15　初次找到可行解的时间对比图

结束语　本文设计了一种无人机航迹规划算法 EB-RRT* ，提出自适应避障以提高算法的收敛速度并减少内存占用；采用栅格分区的方法缩短附近节点的遍历时间；最后利用合理的降采样和三次贝塞尔插值算法对折点进行光滑化的处理，使算法最终生成相对平滑的航迹，为无人机实际飞行提供可行航迹规划方法。未来可以在加入动力学约束后再对路径进行规划和优化以及在三维地图上进行航迹规划算法的研究。

参 考 文 献

[1] VALAVANIS K P. Advances in unmanned aerial vehicles[M]// State of Art and Road to Autonomy. 2007.

[2] DE A,CAVES J. Human-automation collaborative RRT for UAV mission path planning[M]// Massachusetts Institute of Technology,2010.

[3] TRIHARMINTO H H,PRABUWONO A S. UAV Dynamic Path Planning for Intercepting of a Moving Target: A Review [J]. Communications in Computer and Information Science, 2013,376:206-219.

[4] HOLLAND J H. 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] KAVRAKI L,SVESTKA P. Probabilistic roadmaps for path planning in high-dimensional configuration spaces [J]. IEEE Transactions on Robotics & Automations, 1996, 12(4): 566-580.

[7] LAVALLE S M,KUFFNER J J. Randomized Kinodynamic Planning[J]. IEEE International Conference on Robotics & Automation,1999,1(5):473-479.

[8] MELCHIOR N A, SIMMONS R. Particle RRT for Path Planning with Uncertainty[C]// IEEE International Conference on Robotics & Automation. 2007:1617-1624.

[9] LINDEMANN S R,LAVALLE S M. Incrementally reducing dispersion by increasing voronoi bias in rrts[J]. IEEE International Conference on Robotics & Automation,2004,4(4):3251-3257.

[10] KARAMAN S,FRAZZOLI E. Incremental sampling-based algorithms for optimal motion planning[J]. International Journal of Robotics Research,2010,30(7):2011.

[11] QURESHI A H,MUMTAZ S,IQBAL K F,et al. Triangular geometry based optimal motion planning using RRT* -motion planner[C]// IEEE International Workshop on Advanced Motion Control. 2014:380-385.

[12] KUFFNER J J,LAVALLE S M. RRT-connect: An efficient approach to single-query path planning[C]// IEEE International Conference on Robotics and Automation. 2000:995-1001.

[13] JORDAN M, PEREZ A. Optimal Bidirectional Rapidly-Exploring Random Trees: MIT-CSAIL-TR-2013-021[R]. 2013.

[14] ZHANG X,LÜTTEKE F,ZIEGLER C,et al. Self-learning RRT* Algorithm for Mobile Robot Motion Planning in Complex Environments[M]// Intelligent Autonomous Systems 13,2016:57-69.

[15] BAIZID K,CHELLALI R,LUZA R,et al. RRS: Rapidly-Exploring Random Snakes a New Method for Mobile Robot Path Planning[M]// Intelligent Autonomous Systems 13. Springer International Publishing,2014:291-305.

[16] 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.

[17] YANG K. An efficient Spline-based RRT path planner for nonholonomic robots in cluttered environments[C]// International Conference on Unmanned Aircraft Systems. 2013:288-297.

[18] 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.