第49卷 第7期
2021年7月

华南理工大学学报(自然科学版)
Journal of South China University of Technology
(Natural Science Edition)

Vol. 49 No. 7
July 2021

doi: 10.12141/j.issn.1000-565X.200769

基于双树 Quick-RRT* 算法的移动机器人路径规划

魏武 韩进 李艳杰† 高天啸

(华南理工大学 自动化科学与工程学院，广东 广州 510640)

摘 要： 最优快速拓展随机树（RRT*）是一种渐进最优的移动机器人路径规划方法，Quick-RRT* 缩短了 RRT* 的初始路径长度，提高了路径收敛速度。为进一步提高 Quick-RRT* 的收敛速度，文中提出了一种双树 Quick-RRT* 算法。首先，基于 Quick-RRT* 算法在起点和终点分别生成一棵随机树，起点树和终点树轮流生长，两棵树的连接采用贪婪法；然后，对提出的算法的概率完备性和渐进最优性进行理论分析，证明了算法的概率完备性和渐进最优性；最后，基于 Matlab 平台，在 3 种环境下采用双树 Quick-RRT* 与 RRT*、Quick-RRT* 和双向 RRT* 算法进行了对比仿真实验。结果表明，文中改进的算法不仅可以在更短的时间内找到初始路径和次优路径，而且初始路径更短。

关键词： 路径规划；移动机器人；Quick-RRT*；初始路径；收敛速度

中图分类号： TP242 **文章编号：** 1000-565X(2021)07-0051-08

近几十年来，路径规划问题的研究在机器人领域兴起并成为研究的热点[1-2]。路径规划问题就是在环境完全已知或部分已知的前提下寻找一条与障碍物无碰撞的路径。环境完全已知的路径规划问题为全局路径规划，环境部分已知的路径规划问题为局部路径规划。路径规划算法主要包括基于采样的方法[3-4]、人工势场法[5]、可视图法[6]、生物智能算法[7-9]和基于深度学习的方法[10]，其中基于采样的方法以概率路图法（PRM）[3]和快速扩展随机树（RRT）[4]为代表，生物智能算法的典型代表为遗传算法[7]和蚁群算法[8-9]。

基于采样的路径规划方法具有强大的搜索能力，将规划算法从几何学模型中分离[2]，在探索高维空间方面具有突出的表现。因此，众多学者展开了相关研究，其中 RRT 算法[4]引起了高度关注并得到了较为广泛的应用[10-12]。RRT 算法虽然具有概率完备性[13]和计算量小的优点，但 RRT 算法本身不考虑所获得的路径代价，故无法保证路径的质量[14]，即不具备渐进最优性的特性，而这一特性对于实际应用尤为重要。

为了加快 RRT 算法寻找路径的速度，文献[13]提出了双向 RRT 算法，与基本的 RRT 算法最大的不同在于，双向 RRT 算法同时存储从起点、终点处生成的两棵随机树而不是仅有一棵树。在计算资源充足的前提下，双向 RRT 算法的多树思想为加快收敛速度提供了一个新的思路。针对 RRT 算法不具备渐进最优性的缺点，大量学者基于 RRT 算法提出了相应的改进算法。文献[15]提出了具备渐进最优性的 RRT* 算法，其核心在于增添了选择最优父节点和剪枝两个优化过程，但此算法存在收敛速度慢的缺点。文献[16]从限定随机点采样范围的角度提出了 Informed-RRT* 算法，

收稿日期： 2020-12-17

基金项目： 广东省科技计划项目（2019A050520001）

Foundation item: Supported by the Science and Technology Planning Project of Guangdong Province (2019A050520001)

作者简介： 魏武（1970-），男，教授，博士生导师，主要从事机器人控制技术、智能控制技术、模式识别与人工智能研究。E-mail: weiwu@scut.edu.cn

† **通信作者：** 李艳杰（1991-），女，博士生，主要从事机器人路径规划及机器人控制研究。E-mail: 1073889317@qq.com

虽然其在一定程度上加快了 RRT* 算法的收敛速度，但其采样过程在很大程度上依赖于当前最优解，且当超椭球超出规划问题本身时，算法无法进行下去。文献[17]受多棵扩展树可加快收敛速度的启发而提出的双向 RRT* 算法，在保证渐进最优性的前提下很大程度上提高了收敛速度。文献[18]通过扩大 RRT* 算法两个优化过程的追溯范围，提出了 Quick-RRT* 算法，该算法为 RRT* 算法提供了一种新的树扩展框架。

基于多树结构的双向 RRT* 和 Quick-RRT* 算法的优势，本文提出了双树 Quick-RRT* 算法，并通过仿真实验分析了该算法的性能。

1 问题描述及背景知识

1.1 问题描述

路径规划空间常采用位形空间进行描述。依据是否与障碍物发生碰撞，整个位形空间 C 划分为自由位形 C_{free} 和碰撞位形 C_{obs}，其中 $C_{\text{obs}} = C \setminus C_{\text{free}}$。

路径规划问题是给定起点 v_{start} 和终点 v_{goal}（$v_{\text{start}}, v_{\text{goal}} \in C_{\text{free}}$）的前提下寻找一条从 v_{start} 到 v_{goal} 的无碰撞路径。

1.2 RRT* 算法

RRT* 是一种单查询树状结构的搜索算法，从起点 v_{start} 初始化变量 Tree 后进入迭代过程，具体迭代过程如下:

(1) 使用随机采样函数 Sample() 产生随机采样点 v_{rand}。

(2) 产生新节点 v_{new}。先使用 Nearest(v_{rand}, Tree) 函数遍历 Tree 得到最近节点(v_{nearest})，然后 Steer($v_{\text{nearest}}, v_{\text{rand}}, \delta_{\text{step}}$) 函数以 v_{nearest} 为起点，朝着 v_{rand} 前进 δ_{step}，从而产生 v_{new}。

(3) 在 v_{nearest} 和 v_{new} 无碰撞的情况下，优化 v_{start} 到 v_{new} 的路径。Near(Tree, $v_{\text{new}}, r_{\text{near}}$) 函数以 v_{new} 为圆心、r_{near} 为半径找出邻域 V_{near}，ChooseParent($V_{\text{near}}, v_{\text{new}}$) 函数判断 v_{new} 父节点替换成 V_{near} 中的点后是否会缩短 v_{start} 到 v_{new} 的距离，如果缩短则进行替换操作，否则不操作。

(4) 优化 v_{start} 到 V_{near} 中点的路径。Rewire($V_{\text{near}}, v_{\text{new}}$) 函数判断 V_{near} 中每个节点是否可以将其父节点更新为 v_{new}，如果路径距离减小则进行父节点更新操作，否则不操作。优化过程使得 RRT* 算法具有渐进最优性[15]。RRT* 算法的具体实现过程如下:

```
{ Tree. init( v_start) ;
  for i  = 1 to K do
    v_rand ←Sample( i) ;
    v_nearest ←Nearest( v_rand ,Tree) ;
    v_new ←Steer( v_nearest ,v_rand ,δ_step) ;
    if CollisionFree( v_nearest ,v_new ,map)
      V_near ←Near( Tree ,v_new ,r_near) ;
      v_parent ←ChooseParent( V_near ,v_new) ;
      Tree←Link( v_parent ,v_new) ;
      Tree←Rewire( V_near ,v_new) ;
    end if
  end for
  returnTree; }
```

1.3 Quick-RRT* 算法

Quick RRT* 算法利用三角不等式定理对随机树结构进行优化。与 RRT* 算法相比，Quick-RRT* 算法扩大了 ChooseParent($V_{\text{near}}, v_{\text{new}}$) 和 Rewire($V_{\text{near}}, v_{\text{new}}$) 函数中 V_{near} 的范围，在保持时间复杂度和空间复杂度相同的情况下，获得了更优的初始路径和更快的收敛速度[18]。Quick-RRT* 算法的具体实现过程如下:

```
{ Tree. init( v_start) ;
  for i = 1 to K do
    v_rand ←Sample( i) ;
    v_nearest ←Nearest( v_rand ,Tree) ;
    v_new ←Steer( v_nearest ,v_rand ,δ_step) ;
    if CollisionFree( v_nearest ,v_new ,map)
      V_near ←Near( Tree ,v_new ,r_near) ;
      V_prtNear ←Ancestry( Tree ,V_near ,d_near) ;
      v_parent ←ChooseParent( V_near ∪ V_prtNear ,v_new) ;
      Tree←Link( v_parent ,v_new) ;
      V_prtNew ←Ancestry( Tree ,v_new ,d_new) ;
      Tree←Rewire( V_near ,v_new ∪ V_prtNew) ;
    end if
  end for
  return Tree; }
```

Quick RRT* 算法相较于 RRT* 算法，增加了 Ancestry(Tree, V, d) 函数，该函数的作用是追溯邻域 V 的父节点集合。d 为父节点层数，d 值越大，得到的随机树结构越优，但计算资源的需求会越强[18]。

2 双树 Quick-RRT* 算法

基于多树结构的双向 RRT* 算法在提高收敛速度上和 Quick-RRT* 可与多种启发方法结合的优势，

本文提出了双树 Quick-RRT 算法，其具体实现过程如下:

```
{ Tree1. init( vstart) ;
  Tree2. init( vgoal) ;
  for i = 1 to K do
   vrand←Sample( i) ;
   vnearest←Nearest( vrand ,Tree1) ;
   vnew←Steer( vnearest ,vrand ,δstep) ;
   if CollisionFree( vnearest ,vnew ,map)
      Vnear←Near( Tree1 ,vnew ,rnear) ;
      VprtNear←Ancestry( Tree1 ,Vnear ,dnear) ;
      vparent←ChooseParent( Vnear ∪ VprtNear ,vnew) ;
      Tree1←Link( vparent ,vnew) ;
      VprtNew←Ancestry( Tree1 ,vnew ,dnew) ;
      Tree1←Rewire( Vnear ,vnew ∪ VprtNew) ;
      Tree2←Connect( Tree2 ,vnew) ;
      if isConnected( Tree1 ,Tree2)
         Path = FillPath( Tree1 ,Tree2) ;
      end if
      Swap( Tree1 ,Tree2) ;
   end if
  end for
  return path; }
```

起点树 $Tree_1$ 和终点树 $Tree_2$ 进行初始化后，进入迭代过程，具体迭代过程如下:

(1) 产生新节点，使用 Sample( ) 、Nearest(v_{rand} ,$Tree_1$) 函数产生 $v_{nearest}$，使用 Steer($v_{nearest}$,v_{rand} ,δ_{step}) 函数产生 v_{new}。

(2) 在 $v_{nearest}$ 和 v_{new} 无碰撞的情况下，优化 v_{start} 到 v_{new} 的路径。Near(Tree, v_{new} , r_{near}) 和 Ancestry(Tree ,V_{near} ,d_{near}) 两个函数找出 v_{new} 的潜在父节点，ChooseParent($V_{near} \cup V_{prtNear}$,v_{new}) 函数在潜在父节点中选择父节点，使得从 v_{start} 到 v_{new} 的路径距离最小，然后使用 Link(v_{parent} ,v_{new}) 函数连接 v_{new} 和 v_{parent} 这两个节点。

(3) 优化 v_{start} 到 V_{near} 中点的路径。使用 Ancestry(Tree ,v_{new} ,d_{new}) 函数找出 v_{new} 的父节点 $v_{prtNear}$，将 v_{new} 和 $v_{prtNear}$ 作为 $V_{prtNear}$ 中点的潜在父节点，使用 Rewire(V_{near} ,$v_{new} \cup V_{prtNew}$) 函数在潜在父节点中找出父节点，使得路径距离最小。

(4) 使用 Connect($Tree_2$,v_{new}) 函数连接 $Tree_2$ 与 v_{new}。先找出 $Tree_2$ 中与 v_{new} 最接近的节点 $v'_{nearest}$，然后 $v'_{nearest}$ 以 v_{new} 为目标不断前进 δ_{step}，直至遇见障碍物或者与 v_{new} 相连接。

(5) 使用 FillPath($Tree_1$,$Tree_2$) 函数拼接路径。在 $Tree_2$ 与 v_{new} 相连接的情况下，$Tree_1$ 从 v_{new} 追溯父节点直到 v_{start}，这是 $Tree_1$ 部分的路径，$Tree_2$ 采用相同的方法，从与 v_{new} 相连接的节点开始追溯父节点直到 v_{goal}，这是 $Tree_2$ 部分路径，将这两段路径进行拼接，得到可行路径。

(6) 使用 Swap($Tree_1$,$Tree_2$) 函数交换两棵树中的内容，因为上述迭代过程 $Tree_1$ 只增加了一个节点，$Tree_2$ 中增加了多个节点，所以需要交换两棵树内容，使得两棵树经过数次迭代后节点数量保持平衡。

Quick-RRT* 和双树的融合关键在于 Connect($Tree_2$,v_{new}) 和 Swap($Tree_1$,$Tree_2$) 函数。当 $Tree_1$ 完成 v_{new} 的拓展后，Connect($Tree_2$,v_{new}) 将 $Tree_2$ 与 v_{new} 进行直线连接，连接过程会向 $Tree_2$ 中增加多个点，且直线连接不需要优化，拓展只是增加一个点且需要优化路径，所以 Connect( ) 函数提高了算法的效率。Swap($Tree_1$,$Tree_2$) 将两棵树中的内容交换，该函数可以保证两棵树中点的数量大致平衡。

3　算法性能分析

3.1　概率完备性

对于双向 RRT 算法，随机树中的点 v 在 C_{free} 中服从均匀分布，且该算法具有概率完备性[13]。双树 Quick-RRT* 算法较于双向 RRT 算法，产生点的方法相同，区别在于优化了点之间的连接方式，故本算法中的点 v 在 C_{free} 中同样服从均匀分布。同样地，双树 Quick-RRT* 算法具有概率完备性。

3.2　渐进最优性

若算法 A 满足渐进最优性，第 k 次迭代路径长度为 c_k^A，最优路径长度为 c^*，则对于任意规划问题 (C_{free}, v_{start}, v_{goal})，有

$$P\left(\lim_{k\to+\infty}\sup\{c_k^A = c^*\}\right) = 1 \qquad (1)$$

文献[9]中指出，若算法满足概率完备性，且满足该文献中的假设 11 (代价函数的叠加性)、12 (代价函数的连续性) 和 13 (以 C_{free} 中的点 x 为圆心，ε ($\varepsilon \in \mathbf{R}^+$) 为半径的圆 (或超球体) $B_{\varepsilon,x} \subset C_{free}$)，则算法满足渐进最优性。

代价函数的叠加性和连续性确保了两个相似路径的代价也相似，假设 13 确保了最优轨迹附近有自由空间以允许收敛。概率完备性确保了当时间充足时，地图中存在 n ($n\to\infty$) 个点的连通图。当算法迭代时，连通图中的路径会不断缩短，代价函数的连续性确保了在路径缩短的同时，代价函数也会缩小，最终趋近于最优路径，也就是渐进最

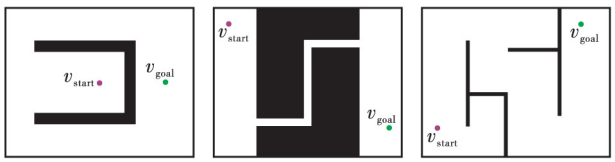
优性。

双树 Quick-RRT* 算法满足概率完备性，路径长度 l_{path} 满足假设 11、12，假设 13 是技术性假设，对地图的一种要求，与算法本身无关，故双树 Quick-RRT* 算法满足渐进最优性。

4 仿真实验

为了分析本文提出的算法的性能，设计了 3 种环境进行仿真分析，考虑到移动机器人的自身半径，对障碍物进行了膨化处理，并将机器人看做一个质点。仿真实验平台及配置为：Matlab R2018b，64 位 Windows 10，处理器 Intel (R) Core (TM) i7-7500U，CPU 主频 2.7 GHz，内存 12 GB。

仿真地图环境如图 1 所示，所有环境的地图规模均为 [184,872]，以地图左上角为坐标原点，最优路径长度为 l_{opt}，次优路径长度 l_{subOpt} 满足 $l_{opt} < l_{subOpt} \leqslant 1.05 l_{opt}$，路径的长度单位是像素，发现可行路径时间为 t_{find}，发现次优路径时间为 $t_{5\%}$。



(a) U形障碍物环境 (b) 狭长通道环境 (c) 简易迷宫环境

图 1 仿真实验环境

Fig. 1 Simulation experiment environments

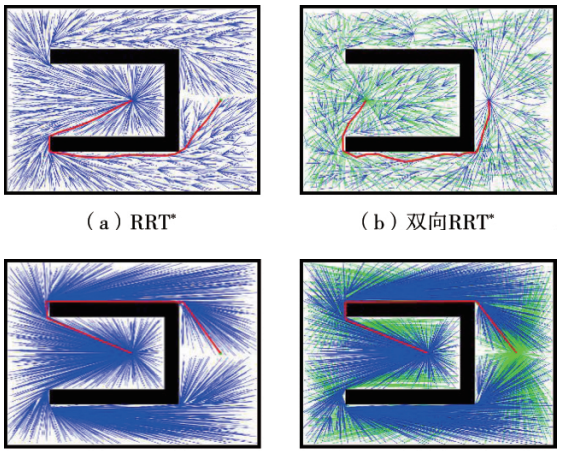
采用本文提出的双树 Quick-RRT* 算法与 RRT*、Quick-RRT* 和双向 RRT* 算法进行对比仿真实验，其中 RRT* 和 Quick RRT* 是单树算法，双向 RRT* 和双树 Quick-RRT* 是双树算法，算法中设置的变量及参数如下：步长 $\delta_{step} = 30$ 像素，产生 V_{near} 的半径 $r_{near} = 80$ 像素，Quick-RRT* 和双树 Quick-RRT* 算法中选择父节点的深度 $d_{near} = d_{new} = 1$。

使用以下 5 个参数对上述 4 种算法进行比较：初始路径的长度 (l_{init})、t_{find}、$t_{5\%}$，算法发现路径的成功率 (R_{suc}) 和路径长度 (l_{path})。前 3 个参数取各算法运行 100 次后的均值；R_{suc} 和时间呈正相关，4 种算法都具备概率完备性，当时间充足时，R_{suc} 都会达到 100%。本文采用以下方法计算 R_{suc}：算法运行 100 次，记录每次的 t_{find}，计算 t_{find} 从 0 s 到指定时刻的分布数量，就是该时刻的成功率。l_{path} 用以显示算法的路径收敛效果。由于算法具有随机性，在相同条件下多次运行同一算法，t_{find} 都会不同，这就导致算法在刚开始运行时没有发现可行路径，或者只有少数次数产生了 l_{path}，这些少数次数无法代表 100 次运行的结果，所以需要 R_{suc} 作为判断依据。本文按以下方法统计 l_{path}：算法运行 100 次，每次运行都会定时记录 l_{path} 的值，故当 100 次运行完毕，每一个时刻都会有 100 次有效或者无效的数据，无效数据指该时刻没有产生 l_{path} (此时 $l_{path} = 0$)，当该时刻有 60 次以上的有效数据才计算该算法的 l_{path} 均值。采用此种方法计算时，因算法之间的时间差异较大，故不同算法间的时间跨度会较大。

上文说明的数据是分为两批实验记录的，因为同时记录所有数据难度较大。l_{path} 是算法运行 100 次定时记录的，l_{init}、t_{find}、$t_{5\%}$ 和 R_{suc} 是另外运行 100 次记录的。l_{init} 和 l_{path} 的起始值有差异，主要受两个因素的影响，一是数据不在同一批实验中获得，二是 l_{init} 不受 R_{suc} 的影响，而 l_{path} 受 R_{suc} 的影响。

4.1 U 形障碍物环境

U 形障碍物环境中的 v_{start} 设置为 [92,436]，v_{goal} 设置为 [1 000,436]。4 种算法的仿真结果如图 2 所示。



(a) RRT* (b) 双向RRT*

(c) Quick-RRT* (d) 双树Quick-RRT*

图 2 4 种算法在 U 形障碍物环境下的仿真结果

Fig. 2 Simulation results of 4 algorithms in U-shaped obstacle environment

从图 2 可知，在 U 形障碍物环境下，双树 Quick-RRT* 算法的性能最佳，路径完全贴合 U 形障碍物边缘。4 种算法的 l_{first}、t_{find} 和 $t_{5\%}$ 如表 1 所示，相同时间的成功率和路径长度如图 3 所示。

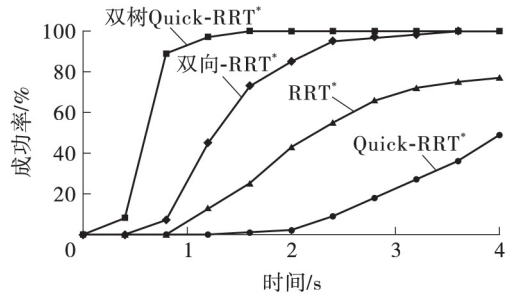
从表 1 可知，双树 Quick-RRT* 算法的性能表现最优，相较于 Quick-RRT* 算法，t_{find} 缩短了 89.8%，l_{init} 却更短，$t_{5\%}$ 缩短了 68.0%；相较于 RRT* 和双向 RRT* 算法，t_{find} 分别缩短了 79.1% 和

57.0%，$t_{5\%}$ 分别缩短了 83.0% 和 77.7%。可以得出，双树 Quick-RRT* 算法大大缩短了 t_{find} 和 $t_{5\%}$，产生的路径更优。

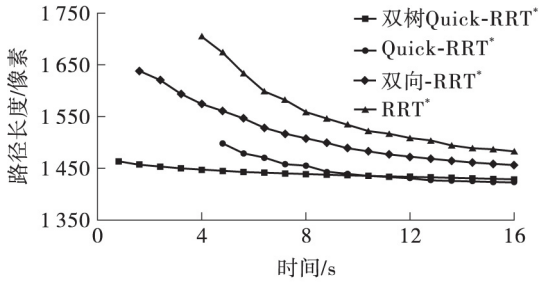
表 1　4 种算法在 U 形障碍物环境中的性能比较

Table 1　Comparison of performance among 4 algorithms in U-shaped obstacle environment

算法	l_{init} /像素	t_{find} /s	$t_{5\%}$ /s
RRT*	1699.825	2.874	18.308
Quick-RRT*	1528.137	5.887	9.727
双向 RRT*	1737.176	1.396	13.960
双树 Quick-RRT*	1510.305	0.600	3.115



（a）发现路径的成功率



（b）路径长度

图 3　U 形障碍物环境下 4 种算法的路径长度和发现路径的成功率

Fig. 3　Path length and success rate of path-finding of 4 algorithms in U-shaped obstacle environment

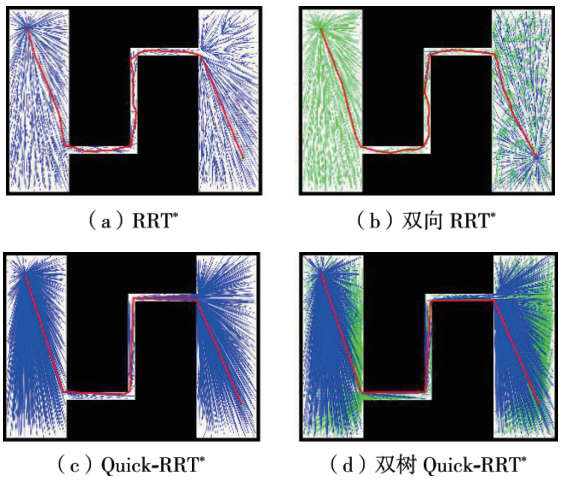
图 3（a）更加具体地展示了 t_{find} 的时间区间分布，双树 Quick-RRT* 算法全部成功所需时间最少，曲线上升最快，双向 RRT* 算法次之，Quick-RRT* 所需时间最长。可以发现，双树结构算法比单树结构算法到达 $R_{\text{suc}}=100$ 所需的时间更短。

从图 3（b）可知：双树 Quick-RRT* 算法出现数据的时间最早，说明在 100 次实验中，有 60 次以上的 l_{path} 所需时间最短，该曲线最为平缓，说明算法不需要花费过多的时间优化路径；双向 RRT* 和 RRT* 算法虽然产生数据的时间也比较早，但曲线前后差值大，说明算法需要更多的时间优化路径；Quick-RRT* 算法有 60 次以上的 l_{path} 所需时间最多，曲线前后差值也比双树 Quick-RRT* 算法大。

综合上述分析可知：Quick-RRT* 算法发现路径需要的时间最长；相较于其他 4 种算法，双树 Quick-RRT* 算法的 t_{find} 最短，相同时间发现路径的成功率最高，路径收敛速度最快。

4.2　狭长通道环境

狭长通道环境中，v_{start} 设置为 [100，100]，v_{goal} 设置为 [100，700]。4 种算法在该环境下的仿真结果如图 4 所示。



（a）RRT*　（b）双向 RRT*　（c）Quick-RRT*　（d）双树 Quick-RRT*

图 4　4 种算法在狭长通道环境下的仿真结果

Fig. 4　Simulation results of 4 algorithms in narrow channel environment

从图 4 可知，Quick-RRT* 和双树 Quick-RRT* 算法的性能明显优于 RRT* 和双向 RRT* 算法。4 种算法的 l_{init}、t_{find} 和 $t_{5\%}$ 参数如表 2 所示，相同时间的成功率和路径长度如图 5 所示。

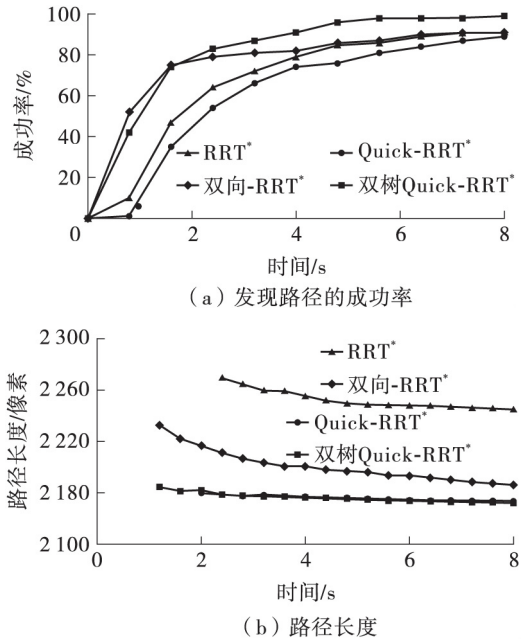
表 2　4 种算法在狭长通道环境中的性能比较

Table 2　Comparison of performance among 4 algorithms in narrow channel environment

算法	l_{init} /像素	t_{find} /s	$t_{5\%}$ /s
RRT*	2270.970	5.6650	9.9720
Quick-RRT*	2166.421	5.1196	5.1197
双向 RRT*	2259.641	4.0470	4.0970
双树 Quick-RRT*	2179.217	1.8450	1.8480

因为狭长通道空间有限，限制了路径优化的空间，使得 l_{init} 与 l_{subOpt} 的差距减小，路径优化时间减少，例如图 5（b）中曲线比图 3（b）中曲线更加平缓，表 2 中 t_{find} 和 $t_{5\%}$ 的差值减小，Quick-RRT* 需要增加一位小数以显示值的不同。双树 Quick-RRT* 算法相较于 Quick-RRT*，l_{init} 虽略长，但 t_{find} 缩短了 64.0%，$t_{5\%}$ 缩短了 63.9%，而相较于 RRT* 和双向 RRT* 算法，t_{find} 分别缩短了 67.4% 和

54.4%，$t_{5\%}$ 分别缩短了 81.5% 和 54.9%。从表 2 可知：双树 Quick-RRT* 算法在保证路径质量的前提下，大幅度提高了算法的搜索路径效率。



（a）发现路径的成功率

（b）路径长度

图 5 狭长通道环境下 4 种算法的的路径长度和发现路径的成功率

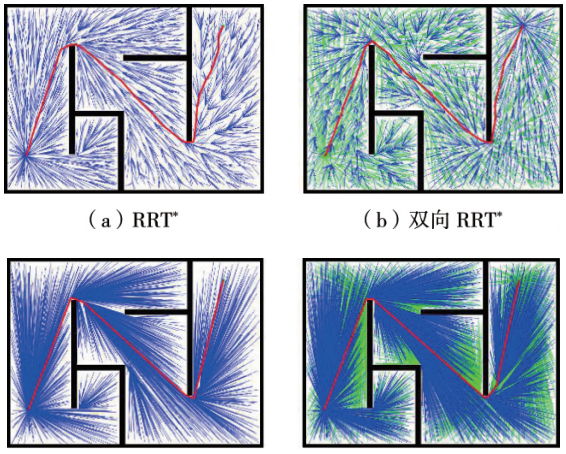
Fig. 5 Path length and success rate of path-finding of 4 algorithms in narrow channel environment

狭长通道空间的限制，使得算法的有效采样点范围大大缩小，加快了算法找到有效路径的效率，也减小了算法之间成功率的差异。图 5（a）中曲线之间的差异比图 3（a）中小，RRT* 和双向 RRT* 算法的曲线在后期几乎重合，说明狭长通道环境减小了单树结构算法和双树结构算法之间的效率差异。双树 Quick-RRT* 和 Quick-RRT* 算法的曲线依旧有明显的差异，说明本文提出的算法在效率上仍有明显的优势。

图 5（b）中双树 Quick-RRT* 算法在 2 s 时的路径长度因为有新产生的 l_{path} 加入而有微小的上升，而不是路径没有收敛。双树 Quick-RRT* 与 Quick-RRT* 算法的曲线几乎重合，但前者的时间缩短了 40%。双树 Quick-RRT* 与双向 RRT* 算法的曲线相比，开始时间相同，但后者的路径长度更长。相较于 RRT* 算法，双树 Quick-RRT* 算法的搜索路径和路径收敛效率高，产生的路径短。综合上述分析可知：即使在狭长空间这种有严重空间限制的环境中，双树 Quick-RRT* 算法使用相同的时间产生的路径长度最短。

4.3 简易迷宫环境

简易迷宫环境中，v_{start} 设置为 [100,700]，v_{goal} 设置为 [1 000,100]，4 种算法在简易迷宫环境下的仿真结果如图 6 所示，l_{init}、t_{find} 和 $t_{5\%}$ 如表 3 所示，相同时间的成功率和路径长度如图 7 所示。



（a）RRT*　（b）双向 RRT*

（c）Quick-RRT*　（d）双树 Quick-RRT*

图 6 4 种算法在简易迷宫环境下的仿真结果

Fig. 6 Simulation results of 4 algorithms in simple maze environment

从图 6 中可知，在简易迷宫环境下，Quick-RRT* 和双树 Quick-RRT* 算法的性能优于 RRT* 和双向 RRT* 算法。

表 3 4 种算法在简易迷宫环境下的性能比较

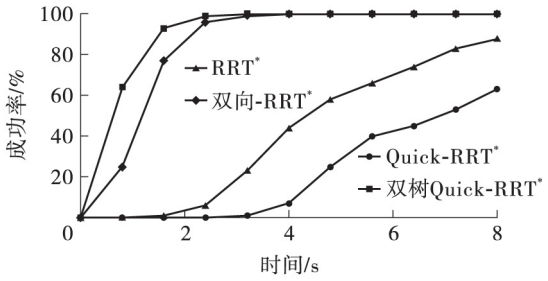
Table 3 Comparison of performance among 4 algorithms in simple maze environment

算法	l_{init}/像素	t_{find}/s	$t_{5\%}$/s
RRT*	2 159.856	5.334	23.806
Quick-RRT*	1 944.616	7.789	9.916
双向 RRT*	2 100.728	1.223	13.919
双树 Quick-RRT*	1 991.625	0.799	4.377

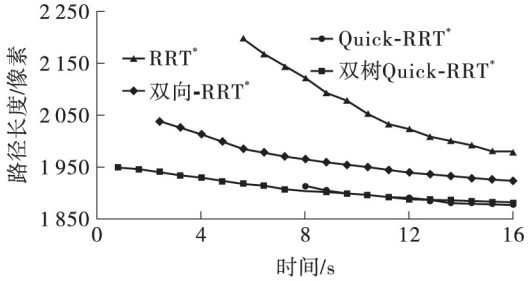
从表 3 可知，双树 Quick-RRT* 算法的参数值仍然是 4 种算法中最小的，相较于 RRT*、Quick-RRT* 和双向 RRT* 算法，t_{find} 分别缩短了 85.0%、89.7% 和 34.7%，$t_{5\%}$ 分别缩短了 81.6%、55.9% 和 68.6%，说明本文提出的算法明显缩短了 t_{find} 和 $t_{5\%}$。

从图 7（a）可知，双树 Quick-RRT* 算法的 R_{suc} 依旧是最快到达 100%，双向 RRT* 算法次之，Quick-RRT* 算法最慢，说明双树 Quick-RRT* 算法在较为复杂的环境下仍然大幅度缩短了 t_{find}。从图 7（b）可知：双树 Quick-RRT* 和 Quick-RRT* 算法的曲线几乎重合，但前者出现数据的时间是后

者的 1/10，大幅度缩短了时间；与另外两种算法相比，双树 Quick-RRT* 算法的搜索路径效率和路径长度都具有优势。



（a）发现路径的成功率



（b）路径长度

图 7　简易迷宫环境下 4 种算法的路径长度和发现路径的成功率

Fig. 7　Path length and success rate of path-finding of 4 algorithms in simple maze environment

综合上述分析可知：在简易迷宫环境中，相较于其他 3 种算法，双树 Quick-RRT* 算法的 t_{find} 仍旧是最小，路径收敛效果最好。

5　结语

为进一步提高 Quick-RRT* 算法的收敛速度，本文提出了一种双树 Quick-RRT* 算法，该算法使用 Quick-RRT* 算法建立起点树和终点树，并分析了该算法的概率完备性和渐进最优性。在 3 种不同环境下的仿真实验结果结明：相较于其他 3 种算法，本文算法提高了寻路效率和质量，t_{find} 缩短了约 69%，$t_{5\%}$ 缩短了约 70%，l_{init} 缩短了约 5%。

本文的研究对象是二维环境中移动机器人，未来可以考虑将双树 Quick-RRT* 算法应用于多维度关节空间的移动机械臂的路径规划问题。

参考文献：

[1]　杨俊成，李淑霞，蔡增玉. 路径规划算法的研究与发展 [J]. 控制工程，2017，24(7)：1473-1480.
YANG Jun-cheng，LI Shu-xia，CAI Zeng-yu. Research and development of path planning algorithm [J]. Control Engineering of China，2017，24(7)：1473-1480.

[2]　LAVALLE S M. Planning algorithms [M]. Cambridge：Cambridge University Press，2006：185-186.

[3]　KAVRAKI L E，SVESTKA P，LATOMBE J C，et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces [J]. IEEE Transactions on Robotics and Automation，1996，12(4)：566-580.

[4]　LAVALLE S M. Rapidly-exploring random trees：a new tool for path planning [R]. Ames：Iowa State University，1998.

[5]　KHATIB O. Real-time obstacle avoidance for manipulators and mobile robots [C] //Proceedings of 1985 IEEE International Conference on Robotics and Automation. St. Louis：IEEE，1985：500-505.

[6]　LOZANO-PEREZ T，WESLEY M A. An algorithm for planning collision-free paths among polyhedral obstacles [J]. Communications of the ACM，1979，22(10)：560-570.

[7]　张好剑，苏婷婷，吴少泓，等. 基于改进遗传算法的并联机器人分拣路径优化 [J]. 华南理工大学学报(自然科学版)，2017，45(10)：93-99.
ZHANG Hao-jian，SU Ting-ting，WU Shao-hong，et al. Sorting route optimization of parallel robot based on improved genetic algorithm [J]. Journal of South China University of Technology (Natural Science Edition)，2017，45(10)：93-99.

[8]　吴玉香，王超. 一种改进的移动机器人三维路径规划方法 [J]. 华南理工大学学报（自然科学版），2016，44(9)：53-60.
WU Yu-xiang，WANG Chao. An improved three-dimensional path planning method of mobile robot [J]. Journal of South China University of Technology (Natural Science Edition)，2016，44(9)：53-60.

[9]　胡耀民，刘伟铭. 基于改进型蚁群算法的最优路径问题求解 [J]. 华南理工大学学报（自然科学版），2010，38(10)：105-110.
HU Yao-ming，LIU Wei-ming. Solving of optimal path problem based on improved ant colony algorithm [J]. Journal of South China University of Technology (Natural Science Edition)，2010，38(10)：105-110.

[10]　洪晓斌，魏新勇，黄烨笙，等. 融合图像识别和 VFH+的无人艇局部路径规划方法 [J]. 华南理工大学学报（自然科学版），2019，47(10)：24-33.
HONG Xiaobin，WEI Xinyong，HUANG Yesheng，et al. Local path planning method for unmanned surface vehicle based on image recognition and VFH+ [J]. Journal of South China University of Technology (Na-

tural Science Edition), 2019, 47(10): 24-33.

[11] 蔡文涛，邓屹，张静，等. 基于改进RRT算法的机械臂路径规划[J]. 传感器与微系统，2019，38(5): 121-124.
CAI Wentao, DENG Yi, ZHANG Jing, et al. Manipulator path planning based on improved RRT algorithm [J]. Transducer and Microsystem Technologies, 2019, 38(5): 121-124.

[12] 成浩浩，杨森，齐晓慧. 基于改进RRT算法的四旋翼无人机航迹规划[J]. 计算机工程与设计，2018，39(12): 3705-3711.
CHENG Hao-hao, YANG Sen, QI Xiao-hui. Dynamic trajectory planning of quadrotors UAV based on improved RRT algorithm [J]. Computer Engineering and Design, 2018, 39(12): 3705-3711.

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

[14] 康亮，赵春霞，郭剑辉. 基于模糊滚动RRT算法的移动机器人路径规划[J]. 南京理工大学学报(自然科学版)，2010，34(5): 642-648.
KANG Liang, ZHAO Chun-xia, GUO Jian-hui. Path planning based on fuzzy rolling rapidly-exploring random tree for mobile robot [J]. Journal of Nanjing University of Science and Technology (Natural Science), 2010, 34(5): 642-648.

[15] KARAMEN S, FRAZZOLI E. Sampling-based algorithms for optimal motion planning [J]. The International Journal of Robotics Research, 2011, 30(7): 846-894.

[16] GAMMELL J D, SRINIVASA S S, BARFOOT T D. Informed RRT*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic [C] // Proceedings of 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. Chicago: IEEE, 2014: 2997-3004.

[17] KLEMM S, OBERLÄNDER J, HERMANN A, et al. RRT*-connect: faster, asymptotically optimal motion planning [C] // Proceedings of 2015 IEEE Conference on Robotics and Biomimetics. Zhuhai: IEEE, 2015: 1670-1677.

[18] JEONG I B, LEE S J, KIM J H. Quick-RRT*: triangular inequality-based implementation of RRT* with improved initial solution and convergence rate [J]. Expert Systems with Applications, 2019, 123: 82-90.

[19] KARAMAN S, FRAZZOLI E. Incremental sampling-based algorithms for optimal motion planning [C] // Robotics: Science and Systems VI. Zaragoza: [s. n.], 2010: 34/1-8.

Path Planning of Mobile Robots Based on Dual-Tree Quick-RRT* Algorithm

WEI Wu HAN Jin LI Yanjie GAO Tianxiao

(School of Automation Science and Engineering, South China University of Technology, Guangzhou 510640, Guangdong, China)

Abstract: The optimal rapid expansion randomized tree (RRT*) is an asymptotically optimal path planning method for mobile robots. Quick-RRT* reduces the initial path length of RRT* and increases the path convergence speed. In order to further improve the convergence speed of Quick-RRT*, this paper proposed a dual-tree Quick-RRT* (Quick-RRT*-Connect) algorithm. Firstly, two random trees were generated at the start and end points respectively based on the Quick-RRT* algorithm. Two trees grew in turn and they were connected with greedy method. Then, the probability completeness and asymptotic optimality of the proposed algorithm were analyzed and testified. Finally, based on the Matlab platform, Quick-RRT*-Connect was compared with RRT*, Quick-RRT* and RRT*-Connect in three environments. The results show that the improved algorithm can not only find initial path and sub-optimal path in a shorter time, but also reduce the initial path length.

Key words: path planning; mobile robot; Quick-RRT*; initial path; convergence speed