基于 RRT* 的智能车辆路径规划算法*

RRT* -based Path Planning Algorithm for Intelligent Vehicle

吴彬彬　罗　峰
(同济大学智能型新能源汽车协同创新中心,上海　201804)

摘　要: 以自动驾驶清扫车为实际应用背景,提出了一种改进的 RRT* 路径规划算法。该路径规划算法引入了车辆形状约束,并且采用目标偏向的采样策略,提高安全性的同时也保证了算法的实时性。另外对路径加入了最大曲率约束,并采用 B 样条曲线对路径进行平滑处理,使规划出的路径能够满足车辆运动学和动力学的约束。仿真和实车试验结果表明,该算法能够满足自动驾驶清扫车的实际应用。

关键词: 路径规划　快速扩展随机树星(RRT*)　B 样条曲线
DOI: 10.16413/j.cnki.issn.1007-080x.2017.10.003

Abstract: Aiming at the application of the autonomous driving sweeper, an improved rapidly-exploring random tree star (RRT*) path planning algorithm is proposed. The algorithm combines the constraints of the shape of the vehicle and the goal-biased sampling strategy, which not only improves the security but also guarantees the real time of the algorithm. In addition, a post-processing method based on the maximum curvature constraint and B-spline basic function is presented to guarantee the kinematic and dynamic constraints of the vehicle. Simulation experiments and real intelligent vehicle test verify the algorithm meet the need of the application of the sweeper.

Key words: path planning　rapidly-exploring random tree star (RRT*)　B-spline

0　引　言

自动驾驶智能车辆作为一个热点研究领域,综合了传感器技术、人工智能技术、控制技术等先进技术,是未来智能交通重要的组成部分。在智能车辆的相关技术中,路径规划是其核心技术之一。

自动驾驶汽车路径规划是指在给定环境的模型上,指定自动驾驶汽车起点和目标点之后,规划出一条能够满足车辆运动学和动力学约束,且不与障碍物发生碰撞的路径。目前,比较常见的路径规划算法主要分为基于地图的路径规划算法和基于采样的路径规划算法。

基于地图的搜索算法通常采用单元分解法或者道路图法建立环境模型。它通过搜索表示环境

* 基金项目: 上海市科学技术委员会项目"基于北斗导航的低速智能汽车关键技术研究",研究项目编号为 14DZ1104700。
作者简介: 吴彬彬　1991 年生,硕士研究生。主要研究方向为自动驾驶汽车决策、规划与控制。
罗　峰　1969 年生,博士,教授。主要研究方向为汽车电子。

信息的环境地图获得最终路径。Dijkstra 算法是求解最短路径的经典算法之一，它是典型的广度优先搜索算法[1]。启发式算法由 Dijkstra 算法改进而来，其显著的特点是在搜索过程中增加了启发函数。启发式搜索算法主要包括 A* 算法[2]、Anytime Repairing A*（ARA*）算法[3]、Lifelong Planning A*（LPA*）算法[4]等。研究表明，启发式搜索算法规划出的路径能够满足实时性和最优性的要求。但是这些算法大都没有考虑车辆非完整性约束的限制，对于自主车来说规划出的路径不一定可跟踪。

最为常见的基于采样的搜索算法是快速扩展随机树算法（rapidly-exploring random tree，RRT）[5]。RRT 算法由 LaValle 和 Kuffner 提出，它最初是用于解决含有运动学约束的路径规划问题。由于 RRT 算法在状态空间进行随机采样，搜索速度快，尤其在高维规划空间中搜索速度优势尤为明显，同时，RRT 算法还具备概率完备性的优点，因此，这种算法作为一种快速搜索方法在路径规划领域获得了广泛应用。

但是 RRT 算法本身也存在一些缺陷，由于在全局进行随机的均匀采样，且每次扩展时都要检验是否会与障碍物发生碰撞，会消耗较多的计算资源，使得收敛的速度较慢[6]；由于是随机采样的，导致生成的路径存在较多的弯折，不是最优的路径，且不能被智能车直接执行[7]；RRT 算法规划出的路径是随机的，处理动态环境时，存在一定难度。

针对基本 RRT 算法的不足，国内外专家提出了一系列的改进措施。为了进一步提高搜索速度并保证算法的完备性，Kuffner 和 LaValle 提出了双向搜索树（Bi-RRT）：从初始位置和目标位置同时生成两棵树，待两棵树连接上时，便完成了路径的扩展，从而加快算法的收敛[8]。Urmson 和 Simmons 提出了一种启发式的功能，使得随机树的扩展具有偏向性来加快算法的收敛[9]。Karaman 和 Frazoll 提出了 RRT* 算法，可以得到一个渐近最优的路径[10]。为了减少规划出的路径的弯折，保证其可执行，Delsart 等人使用二次多项式来对路径进行优化，但是他们没有对障碍物进行考虑，且优化出的结果与理想目标相差较远[11]；杜明博等人提出基于最大曲率约束的方法，删除多余的节点，减少路径的弯折[12]；宋金泽等人使用 B 样条函数来拟合控制点生成平滑可跟踪的路径[13]。为了使 RRT 算法适用于动态环境，Karaman 等人提出固定路径的概念，取规划好的路径中接近车辆的一小段作为固定路径，车子沿着固定路径行驶，同时对固定路径之外的路径进行实时优化[14]；Lan 等人提出对规划好的路径进行障碍物检测，若发生碰撞，则对碰撞部分路径进行重规划使其适应于新环境[15]。

本文在基本 RRT* 算法的基础上，采用目标偏向采样策略，在碰撞检测时考虑了车身的形状，加入了最大曲率的约束，并采用 B 样条曲线对规划的路径进行平滑处理，从而生成曲率连续的平滑路径。通过仿真和实车试验验证了该算法的有效性。

1 问题描述

首先定义状态空间 $Q \subset R^n (n \in N)$ 表示给定的搜索空间的维度。被障碍物占用的地方用 $Q_{obs} \subset Q$ 表示，没有障碍物的空间用 $Q_{free} = Q / Q_{obs}$ 表示，$q_{init} \subset Q_{free}$ 表示起始点，$q_{goal} \subset Q_{free}$ 表示目标点。起点 q_{init} 和终点 q_{goal} 作为规划器的输入，其目标就是在起点 q_{init} 与终点 q_{goal} 之间寻找到一条不会与障碍区域发生碰撞的路径，同时希望路径的长度尽可能短，消耗的时间尽可能少。

2 改进 RRT* 算法

2.1 基本 RRT 算法

RRT 使用随机扩展的方法在搜索空间内从初始位置朝着目标位置来构建树。树的扩展过程就是一个反复迭代的过程。在一次迭代的过程中，从搜索空间随机选择一个点 q_{rand}，如果该点落在无障碍物区间的话，那么就会遍历 T 找到距离该随机点最近的节点 $q_{nearest}$。如果 q_{rand} 与 $q_{nearest}$

间的连线不会与障碍物发生碰撞，且 q_{rand} 与 $q_{nearest}$ 间的距离小于扩展步长，那么 q_{rand} 就作为叶节点 q_{new} 添加到随机扩展树上去；如果 q_{rand} 与 $q_{nearest}$ 间的距离大于扩展步长，那么在 q_{rand} 与 $q_{nearest}$ 的连线上，取距 $q_{nearest}$ 扩展步长长度的点作为 q_{new} 添加到随机扩展树上去，距离 q_{new} 最近的节点称作 q_{new} 的父节点。重复上述迭代过程，直到目标节点作为叶节点或者超过了最大的迭代次数时结束搜索。由目标节点，反向追溯到根节点，便得到了规划的路径。RRT 算法搜索过程如表 1 所示，节点扩展过程见图 1。

表 1　RRT 算法伪代码

RRT 算法
1. T←*InitializeTree*( );
2. T←*InsertNode*(∅, q_{init}, T);
3. *For* i=0 *to* i=N *do*
4. q_{rand}←*Sample*(i);
5. $q_{nearest}$←*Nearest*(T, q_{rand});
6. (q_{new}, U_{new}) ←*Steer*($q_{nearest}$, q_{rand});
7. *If Obstaclefree*(q_{new}) *then*
8. T←*InsertNode*(q_{min}, q_{new}, T);
9. *End if*
10. *End for*
11. *Return T*

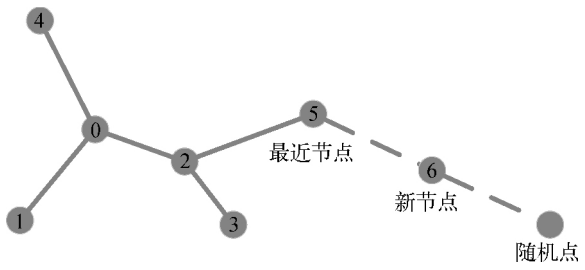

图 1　RRT 算法示意图

2.2　RRT* 算法

RRT* 算法的基本原理与 RRT 算法类似，只是在 RRT 算法的基础之上增加了两点改进，称之为临近空间的搜索和树的重新连接。临近空间的搜索是指在父节点的选择时，新节点会在事先定义好的半径为 r 的空间内，以该范围内的所有节点作为父节点，计算其到达新节点所消耗的距离，取距离最小的节点作为新节点的父节点。树的重新连接是指，以该新节点作为父节点，计算到达该范围内的其他节点的距离，若新的距离值小于原先到达该节点的距离值，那么则将新节点作为该节点的父节点重新连接。RRT* 算法搜索过程如表 2 所示。

表 2　RRT* 算法伪代码

RRT* 算法
1. T←*InitializeTree*( );
2. T←*InsertNode*(∅, q_{init}, T);
3. *For* i=0 *to* i=N *do*
4. q_{rand}←*Sample*(i);
5. $q_{nearest}$←*Nearest*(T, q_{rand});
6. (q_{new}, U_{new}) ←*Steer*($q_{nearest}$, q_{rand});
7. *If Obstaclefree*(q_{new}) *then*
8. q_{near}←*Near*(T, q_{new}, |T|)
9. q_{min}←*Chooseparent*(q_{near}, $q_{nearest}$, q_{new});
10. T←*InsertNode*(q_{min}, q_{new}, T);
11. T←*Rewire*(T, q_{near}, q_{min}, T);
12. *End if*
13. *End for*
14. *Return T*

图 2 生成了一个随机节点，其距离节点 6 的距离为 2，相比于扩展树上的其他节点，距离节点 6 的距离最近，且与节点 6 之间的连接不会与障

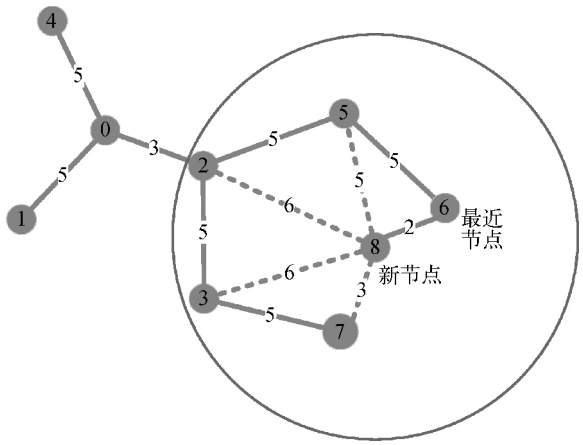

图 2　RRT* 算法新节点的添加

碍物发生碰撞。故节点 8 作为新节点添加到扩展树上面。

RRT* 算法相比于 RRT 算法，在父节点的选择时，增加了临近空间的搜索，以新节点 8 为圆心，r 为半径，画出一个圆，从该圆范围内的节点中，搜索出从起点至节点 8 距离最近的节点。如图 3 所示，该圆范围内包含节点 2、3、5、6、7，节点 8 的初始扩展路径为 0→2→5→6→8，该路径的长度为 15。该圆范围内还存在通过其他节点扩展至节点 8 的路径，分别为 0→2→8，0→2→5→8，0→2→3→8，0→2→3→7→8，以上四条路径的长度分别为 9，13，14，16。可以发现路径 0→2→8 至节点 8 的路径最短，故对节点 8 重新连接，不再以节点 6 为父节点，而是以节点 2 作为父节点。

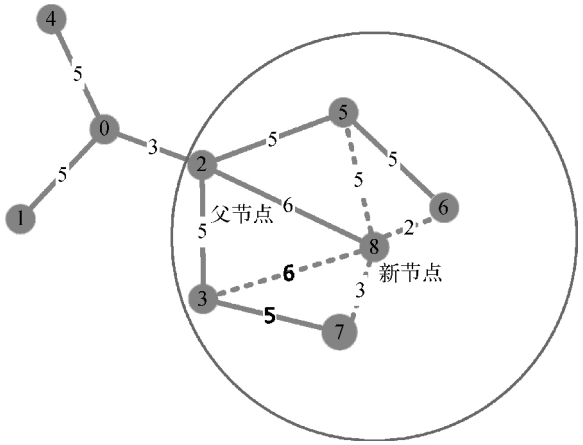

图 3　RRT* 算法临近空间搜索

RRT* 算法相比于 RRT 算法，还增加了树的重新连接的特点。在确定了节点 8 的父节点后，圆内的其他节点 3、5、6、7 出现了以节点 8 为父节点的可能性。将节点 3、5、6、7 分别以节点 8 作为父节点，计算其路径长度，可以发现节点 6 以节点 8 为父节点时，其路径长度为 11，小于以节点 5 为父节点的路径长度 13；同理节点 7 以节点 8 为父节点时，其路径长度为 12，小于以节点 3 为父节点的路径长度 13。故将节点 8 作为节点 6、7 的父节点对扩展树进行重新连接。

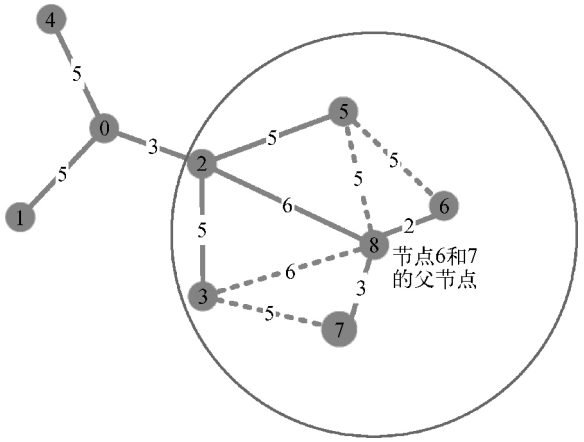

图 4　RRT* 算法树的重新连接

2.3　静态环境下的路径规划

2.3.1　车辆形状约束

在 RRT 算法的路径规划过程中，将本车看成运动的质点，忽略该车辆的形状参数。但是实际上，因为本车的大小和障碍物的大小处于同一个数量级，因此忽略本车的形状参数可能会导致当沿着规划的路径行驶时与障碍物发生碰撞。

为了满足车辆的形状约束，即在路径规划过程中要判断车辆是否会和障碍物发生碰撞，因此在对新节点进行碰撞检测时，需要对运动车辆的四条边界也进行碰撞检测。

如图 5 所示，在最近节点处，车的位置为 $A_1B_1C_1D_1$，在新节点处，车的位置为 $A_2B_2C_2D_2$。若将本车作为质点的话，在进行碰撞检测时，最近节点与新节点的连线不会与障碍物发生碰撞，但是若将车辆形状考虑进去的话，就会与障碍物发生碰撞。所以在进行碰撞检测时，需要考虑车

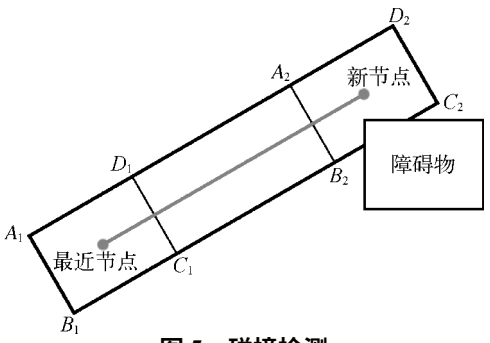

图 5　碰撞检测

辆形状约束，对多边形 $A_1B_1C_2D_2$ 进行碰撞检测。

2.3.2 目标偏向采样策略

基本 RRT 算法采用的是对整个状态空间进行均匀的随机搜索，这样的搜索方式便于向全局空间进行扩展，但是会导致其在不必要的空间进行采样，从而浪费大量的计算资源，降低算法的收敛速度。为了克服这一缺陷，本文采用了目标偏向采样策略。在进行随机采样时，每进行固定次数的随机采样之后，便把目标点取为采样点。具体采样过程见表 3。目标偏向采样策略既保持了算法的随机特性，又加快了算法的收敛速度。将目标偏向采样伪代码代替 RRT^* 算法伪代码的第三、第四行，便为偏向 RRT^* 算法伪代码。

表 3 目标偏向采样伪代码

目标偏向采样
1. For $i=0$ to $i=N$ *do*
2. If $i=n+b$, $n+2b$, $n+3b$ … then
3. $q_{rand}=q_{goal}$;
4. Else
5. $q_{rand}=\mathrm{Sample}(i)$

2.3.3 最大曲率约束

对于智能车的实际情况，其前轮转角 $|\varphi|\leqslant\varphi_{max}$（一般在 30°~40°之间，这里取 $\varphi_{max}=35°$），若要使智能车能够跟踪规划出的路径，必须使得路径的曲率半径大于智能车的最小转弯半径。

因此在获得规划好的路径后，从路径起点开始，逐一计算两相邻树枝的夹角 α，需要满足 $\alpha\geqslant\pi-\varphi_{max}$，即 $\alpha_{min}=\pi-\varphi_{max}$。当其中相邻树枝段间夹角小于 α_{min} 时，调整下一树枝段的节点，从而使尖锐的夹角变得平缓，以便进行 B 样条拟合时生成的轨迹曲率不超过最大曲率的约束。如图 6 所示，规划出的路径段由节点 z_1、z_2、z_3、z_4、z_5 和 z_6 组成。当 z_2、z_3 和 z_4 间的夹角 $\alpha<\pi-\varphi_{max}$ 时，需要基于 α_{min} 插入一个节点 z_{insert}，对这样的尖锐夹角进行处理，使得 $\angle z_2z_{insert}z_4=\alpha_{min}$。若 $z_{insert}z_4$ 不与障碍物发生碰撞，则将 z_{insert} 替换 z_3 加入路径中，若 $z_{insert}z_4$ 会与障碍物发生碰撞，则令 $z_{insert}z_3$ 的中点为新的 z_{insert} 和取 z_3z_4 的中点为 $z_{insert2}$ 进行障碍物判断，直到路径段与障碍物无碰撞。将 z_{insert} 和 $z_{insert2}$ 替换 z_3 加入路径中，最终得到相对更为平缓的路径点序列。

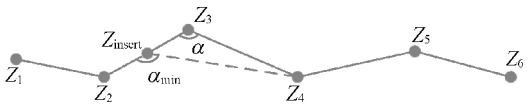

图 6 最大曲率约束

两相邻树枝段的夹角 α 可以由 z_2、z_3 和 z_4 的坐标得到：

$$K_1=\frac{y_{z2}-y_{z3}}{x_{z2}-x_{z3}}$$

$$K_2=\frac{y_{z4}-y_{z3}}{x_{z4}-x_{z3}}$$

$$\alpha=\arctan\left(\frac{K_1-K_2}{1+K_1K_2}\right)>\pi-\varphi_{min}$$

2.3.4 平滑处理

B 样条曲线具有连续性和局部性等优点，同时，B 样条曲线规划路径可以满足对局部路径进行修改而不需要改变整个路径形状的特性，使得 B 样条曲线在路径规划中的应用比较广泛。因此本文利用它的这些优点对路径点进行曲率连续拟合，以生成智能车辆可执行的平滑路径。

给定 $n+1$ 个控制点 $P_i(i=0,1,2,\cdots,n)$ 的坐标，n 次 B 样条曲线段的参数表达式为

$$P(t)=\sum_{i=0}^{n}P_iF_{i,n}(t),\ t\in[0,1]$$

式中，$F_{i,n}(t)$ 为 n 次 B 样条基函数，其形式为

$$F_{i,n}(t)=\frac{1}{n!}\sum_{j=0}^{n-i}(-1)^jC_{n+1}^{j}(t+n-i-j)^n$$

其中，$C_{n+1}^{j}=\dfrac{(n+1)!}{j!\ (n+1-j)!}$。

2.4 动态环境下的路径规划

当环境发生改变，且对之前规划好的路径产生影响，则需要对当前路径进行重新规划。目前，进行路径重新规划的方法主要有两种：第一种是，丢弃之前规划好的路径，从头开始重新规划。在环境经常发生变化的场景，这种方法会消耗很多的计算资源，难以满足实时性的要求。另一种方法是，对之前规划好的路径进行局部修改，使得整个路径满足新的环境。

本文采用第二种方法。首先利用静态环境下的规划方法，规划出一条路径，在环境感知传感器数据更新的每个周期里，根据环境信息判断规划好的路径是否会与障碍物发生碰撞。若不发生碰撞，则沿着规划好的路径行驶，然后在下一个周期继续判断；若障碍物会与规划好的路径发生碰撞，则判断是哪段路径会与障碍物发生碰撞，找到这段路径的起始点以及终止点，删除该起始点与终止点之间的路径，并以该起始点为根节点，终止点为目标节点，重新使用 RRT* 算法进行扩展。进行重新规划时，生成的扩展点尽量靠近受影响的路段，这样能够在受影响路段附近快速生成可行驶的路径。

表 4 重规划算法伪代码

重规划算法
Input: $\sigma=\{x_0, x_1, x_2, \cdots, x_n, x_{n+1}\}$, $T=(V, E)$, Q_{obs};
1. *For* $i=1$ *to* $i=n+1$ *do*
2. *If Obstaclefree*(x_i, x_{i+1}) *then*
3. *mark* x_{i+1} *as VALID*;
4. *Else*
5. *mark* x_{i+1} *as INVALID*;
6. *End if*
7. *End for*
8. *If* σ *does not have any INVALID nodes then*
9. *return* σ;
10. *Else*
11. *find the first INVALID vertice* x_a;
12. *choose the parent vertice of* x_a *as* q_{init};
13. *find the last INVALID vertice* x_b;
14. *choose the chirdren vertice of* x_b *as* q_{goal};
15. *delete all the INVALID nodes from the tree*;
16. *apply RRT* to regrow the tree from* q_{init} *to* q_{goal};
17. *End if*
18. *Return a new path* σ_{new}

3 实验与分析

3.1 仿真实验

3.1.1 静态环境

仿真环境为 MATLAB2015b，硬件平台为 Win10+Corei5-6200U@2.30GHz+8GB RAM。静态仿真时，在 100 m×100 m 的环境中进行，起始点位置(50，0)，目标点位置(50，100)。

表 5 静止环境下各算法规划结果对比

算 法	扩展时间/s	节点数	路径长度/m
RRT	12.872 5	820.8	136.426
RRT*	224.507 5	833.8	112.622 2
改进 RRT*	1.886	56.4	107.602 6

图 7 为 RRT 算法规划出的路径(本文所有图中涉及到的单位均为 m)。由表 5 中数据可知，RRT 算法进行 20 次路径规划，平均耗时 12.872 5 s，平均扩展节点数 820.8 个，路径平均长度 136.426 m。从图 7 中可以看出，规划出的路径存在较多的弯折，同时路径的长度较长。

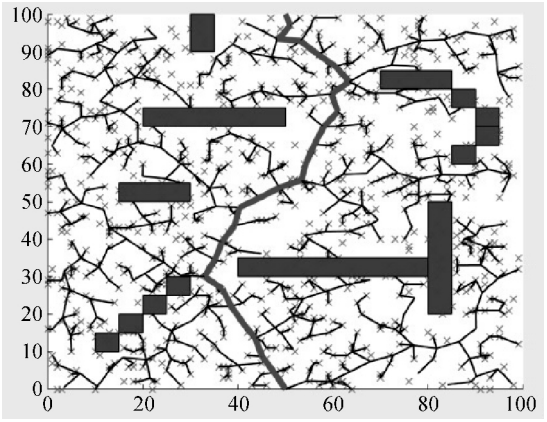

图 7 RRT 算法路径规划结果

图 8 为 RRT* 算法规划出的路径。由表 5 中数据可知，RRT* 算法进行 20 次路径规划，平均耗时 224.507 5 s，平均扩展节点数 833.8 个，路径平

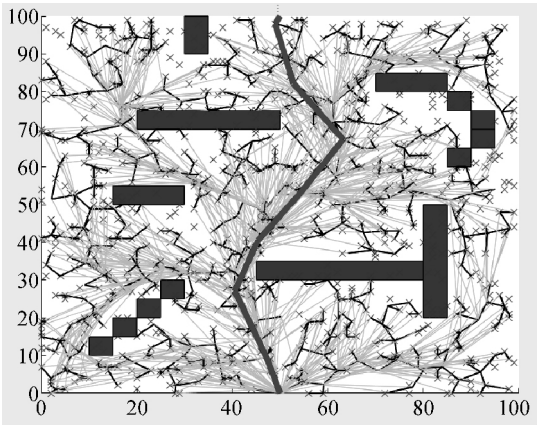

图 8　RRT* 算法路径规划结果

均长度 112.622 2 m。从图 8 中可以看出，RRT* 算法规划出的路径相比于 RRT 算法弯折更少，路径长度明显缩短。但是由于 RRT* 算法对于节点的重新连接，使得规划时间大大延长，并不能在实车上直接应用。

图 9 为改进 RRT* 算法规划出的路径。由表 5 中数据可知，改进 RRT* 算法进行 20 次路径规划，平均耗时 1.886 s，平均扩展节点数 56.4 个，路径平均长度 107.602 6 m。改进 RRT* 算法规划出的路径弯折较少，经过 B 样条优化后，成为车辆可执行的平滑轨迹。同时规划时间短，生成路径也很短，接近最优的路径。

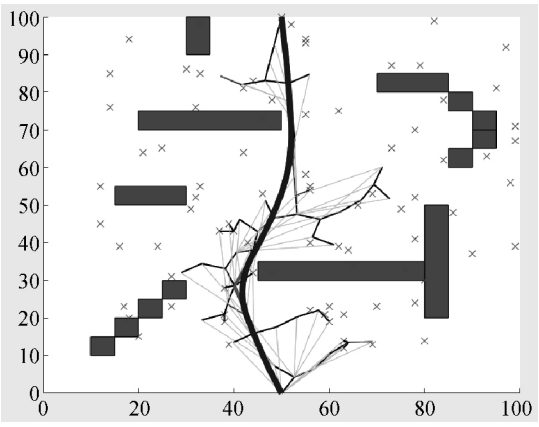

图 9　改进 RRT* 算法路径规划结果

3.1.2　动态环境

动态环境场景如图 10 所示。在试验车辆 A 前方有一辆汽车 B，路径规划模块会规划出一条超车路径，在超车的过程中，检测到车辆 B 前方有一辆汽车 C，如图 11 所示。此时进行路径的重新规划，规划出一条避免与汽车 C 发生碰撞的路径。

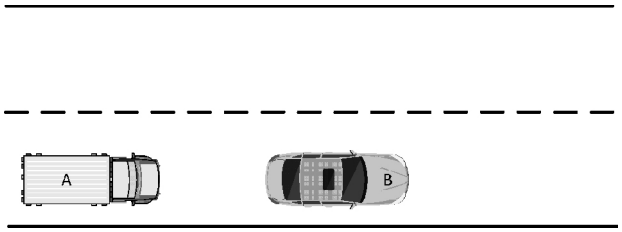

图 10　动态环境示意图 1

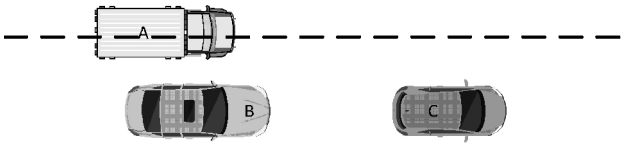

图 11　动态环境示意图 2

动态仿真时，在 10 m×50 m 的环境中进行，起始点位置(0，2.5)，目标点位置(50，2.5)。规划出路径如图 12 所示的曲线。

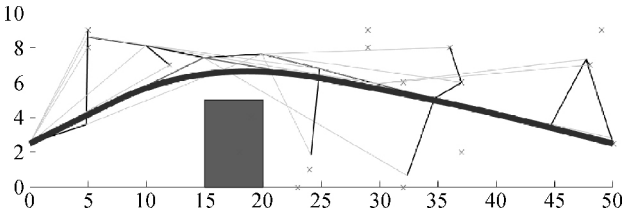

图 12　动态环境路径规划结果

规划出图 12 所示的路径之后，车辆沿着规划好的路径行驶。然后车辆检测到前方路径上出现一个障碍物，如图 13 所示。此时需要对规划好的路径进行重新规划。

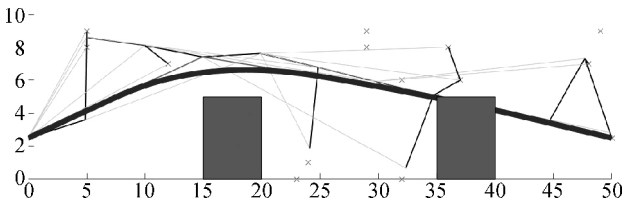

图 13　动态环境检测到新障碍物

重新规划的过程中，车依然会沿着之前的路径行驶，待规划出可以避开第二个障碍物的第二段路径后，如图 14 所示，车辆沿着第二段路径行驶。在该动态环境下，20 次规划结果如表 6 所示，重规划平均耗时 85.3 ms，小于环境感知信息更新周期 100 ms，该时间满足低速清扫车的实际使用需求。

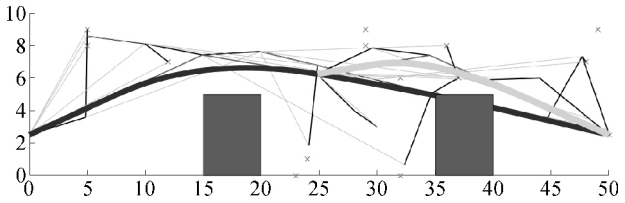

图 14　动态环境路径重新规划

表 6　动态环境下改进 RRT* 算法仿真结果

算　法	扩展时间/s	重规划时间/s
改进 RRT*	0.877	0.085 3

3.2　实车试验

实车试验车辆如图 15 所示。该车为一辆自动驾驶电动清扫车，对电动清扫车进行线控制动和线控转向的改造，同时进行智能化改造。加装的环境感知传感器包括：前向单目摄像头 DELPHI IFV250、前向毫米波雷达 DELPHI ESR、后向毫米波雷达 24G-RF01 LU WANG，以及车身周围的超声波传感器 SAIC ULTRASOUND。清扫车环境感知传感器探测范围如图 16 所示。

图 15　自动驾驶清扫车前方传感器布置

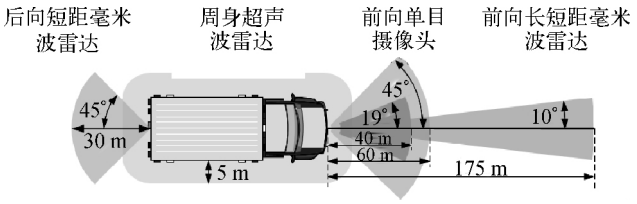

图 16　清扫车环境感知传感器范围

进行实车试验时，当清扫车 A 检测到前方障碍车辆 B 时，规划出如图 17 所示的路径，清扫车 A 沿着该路径行驶。当清扫车 A 行驶至障碍车 B 左侧时，传感器检测到障碍物车辆 C，发现若沿着当前规划好的路径继续行驶的话，会与 C 发生碰撞。此时对碰撞部分重新规划，20 次实车试验规划时间如表 7 所示，规划出如图 18 所示的第二段路径，清扫车沿着第二段路径行驶，避免了与障碍物的碰撞。

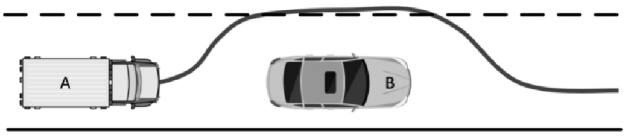

图 17　实车试验路径规划结果

表 7　动态环境下改进 RRT* 算法实车试验结果

算　法	扩展时间/s	重规划时间/s
改进 RRT*	0.926	0.087 1

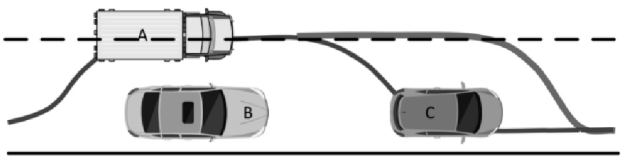

图 18　实车试验路径重新规划

4　结束语

本文在 RRT* 算法的基础之上，提出若干改进措施，包括加入车辆形状的约束、目标偏向采样策略、最大曲率约束以及基于 B 样条曲线的平

滑处理。仿真和试验结果均表明，该改进的 RRT* 算法能够规划出满足车辆运动学和动力学约束的路径，且实时性较好，具有一定的理论价值和工程应用价值。

参 考 文 献

[1] McKEEVER S D. Path Planning for an Autonomous Vehicle [M]. Master Thesis, MIT, 2000.

[2] HUYN N, DECHTER R, PEARL J. Probabilistic analysis of the complexity of A* [J]. Artificial Intelligence, 1980, 15(3): 241-254.

[3] LIKHACHEV M, FERGUSON D, GORDON G, et al. Anytime dynamic A*: an anytime, replanning algorithm [C]. Fifteenth International Conference on Automated Planning & Scheduling, 2005: 262-271.

[4] KOENIG S, LIKHACHEV M. Incremental A* [J]. Proceedings of the Neural Information Processing Systems, 2001, 57(9): 1539-1546.

[5] LAVALLE S M. Rapidly-exploring random trees: a new tool path planning [R]. Ames, USA: Iowa State University, 1998: 293-308.

[6] CHRISTOS K, MOHAMMED Q, CHEN W H, et al. Real-time motion planning methods for autonomous onroad driving: state-of-the-art and future research directions [J]. Transportations Research Part C, 2015, 60: 416-442.

[7] MURPHY L, NEWMAN P. Risky planning: path planning over cost maps with a probabilistically bounded speed-accuracy tradeoff [C]. IEEE International Conference on Robotics and Automation, 2011, 19(6): 3727-3732.

[8] LaVALLE S M, KUFFNER J J. Rapidly-exploring random trees: Progress and prospects [C]. Algorithmic & Computational Robotics New Directions, 2000: 293-308.

[9] URMSON C, SIMMONS R G. Approaches for heuristically biasing RRT growth [C]. IEEE/RSJ International Conference on Intelligent Robots & Systems, 2003, 2: 1178-1183.

[10] KARAMAN S, FRAZZOLI E. Incremental sampling-based algorithms for optimal motion planning [J]. International Journal of Robotics Research, 2012, 30(7): 846-894.

[11] DELSART V, FRAICHARD T, MARTINEZ L. Real-time trajectory generation for car-like vehicles navigating dynamic environments [C]. IEEE International Conference on Robotic and Automation, 2009: 3401-3406.

[12] 杜明博，梅涛，陈佳佳，等. 复杂环境下基于 RRT 的智能车辆运动规划算法 [J]. 机器人，2015，7：443-450.

[13] 宋金泽，戴斌，单恩忠，等. 一种改进的 RRT 路径规划算法 [J]. 电子学报，2010，2：225-228.

[14] KARAMAN S, WALTER M R, PEREZ A, et al. Anytime motion planning using the RRT* [C]. IEEE International Conference on Robotics and Automation (ICRA), 2011, 47(10): 1478-1483.

[15] LAN X, CAIRANO S D. Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT* [C]. Control Conference, 2015: 2360-2365.