同济大学

基于 RRT 系列算法的
机器人路径规划研究与仿真实验

课 程	机器人控制与自主系统
组员 1	邓立原
组员 2	关瑞琦
组员 3	崔晰然

2022 年 6 月

目 录

摘要

RRT 算法是一种基于采样的机器人路径规划算法。本文介绍了 RRT 算法及其衍生算法的基本概念和理论，比较了不同算法的异同。本文基于 Python 和 Gazebo 编写程序，通过仿真实验完成了对经典 RRT 算法和 RRT*、RRT-connect 等衍生算法的复现与验证，通过动画方式展现了不同算法的运行过程，并进行了性能分析。

本文以移动机器人未知环境探索和机械臂路径规划两个具体任务为例，介绍了 RRT 算法在两个任务上的应用。对于移动机器人未知环境探索任务，机器人可以利用 RRT 系列算法来确定前沿点的位置，并通过代价函数方式进行前沿点决策，使得机器人高效地完成未知环境的探索。本文利用 Gazebo 编写程序，展现了机器人在未知环境中的探索过程。对于机械臂路径规划任务，本文以 RobotAnno v6 机械臂为例，对机械臂进行了运动学建模，给出了基于 RRT 算法和插值算法的空间轨迹规划结果。

1. 路径规划及 RRT 算法基本概念

1.1 机器人路径规划基本概念

路径规划是机器人技术的重要分支。机器人路径规划指的是让目标机器人在规定的区域内找到一条从起点到终点的无碰撞安全路径。机器人路径规划通常伴随有优化目标，例如路径与障碍物的距离尽量远，同时路径的长度尽量短，或者能量尽可能少。因此，机器人路径规划实质上是一个有约束的优化问题。

随着机器人技术的不断成熟，机器人路径规划方法也越来越多。按算法原理分类，主要可以分为三类：基于搜索的算法、基于采样的算法、基于优化的算法、基于深度学习的算法等。其中，基于搜索的算法包括由 Dijkstra 发展而来的 A*[1] 及其改进算法 D*[2]等；基于采样的算法以 RRT[3]和 PRM[4]类算法为典型代表，在 RRT 算法基础上有许多衍生，例如 RRT-Connect[5]、RRT*[6]等；基于优化的算法通常有遗传算法、模拟退火算法、蚁群算法等；基于深度学习的方法在近年来兴起，具有代表性的工作为基于深度强化学习的无地图自主避障算法[7]。

本文将对基于采样的 RRT 算法及其变体 RRT-connect、RRT*等算法做详细的叙述与仿真实验分析。

1.2 RRT 算法基本概念

RRT 算法是快速扩展随机树（**R**apidly-exploring **R**andom **T**rees）算法英文首字母的缩写简称，是近十几年得到广泛发展与应用的基于采样的运动规划算法，其最早由美国爱荷华州立大学的 Steven M. LaValle 教授在 1998 年提出，后逐渐发展衍生为一类基于采样的算法。

RRT 算法是一种在多维空间中有效率的规划方法。经典 RRT 算法是通过一个初始点作为根结点，通过随机采样，增加叶子结点的方式，生成一个随机扩展树，当随机树中的叶子结点包含了目标点或进入了目标区域，便可以在随机树中找到一条由树结点组成的从初始点到目标点的路径。

RRT 是一种通过随机构建空间填充树来有效搜索非凸，高维空间的算法。树是从搜索空间中随机抽取的样本逐步构建的，并且本质上倾向于朝向大部分未探

测区域生长。由于它可以轻松处理障碍物和差分约束（非完整和动力学）的问题，并被广泛应用于自主机器人运动规划。

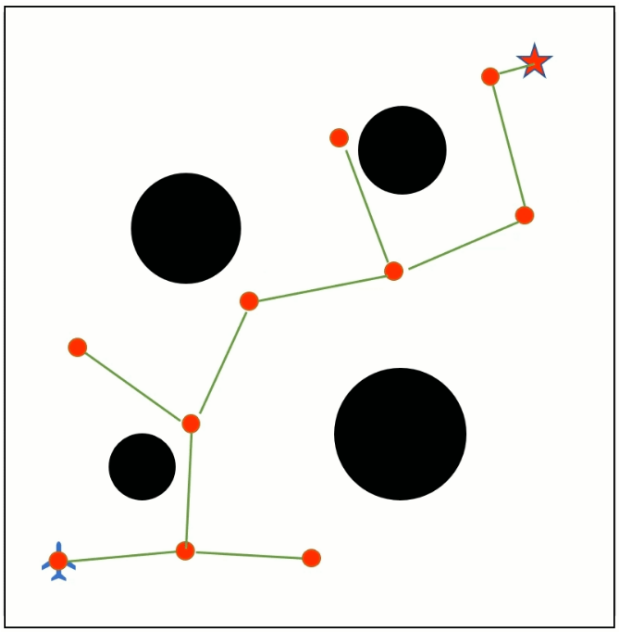


图 1.1 RRT 算法示意图

图 1.1 展示了一次 RRT 算法的路径规划结果。图中，飞机标记表示起点，五角星标记表示终点，黑色表示障碍物区域，白色表示无障碍区域，红色圆点表示 RRT 的结点，绿色线段表示 RRT 的树枝。RRT 从出起点出发，不断在无障碍区域中扩展子结点，直到有结点达到终点，即树中涵盖了一条从起点到终点的路径为止。连接这些结点的枝，即是 RRT 算法规划出的路径。

算法流程分为四步，包括：

（1）设定初始点x_{init}与目标点x_{goal}，以及设定状态采样空间（地图），初始化树$\mathcal{T}$；

（2）进行随机采样得到采样点x_{rand}，如果采样点x_{rand}在障碍物内，则重新随机采样；

（3）若不在障碍物内，计算该采样点x_{rand}与树$\mathcal{T}$中的所有结点之间的距离，得到离得最近的结点x_{near}，再从结点x_{near}以步长 StepSize 走向结点x_{rand}，生成一个新的结点x_{new}，若x_{near}与x_{new}的连线E_i经过障碍物，则重新随机采样；

（4）经过反复迭代，生成一个随机扩展树，当随机扩展树中的子结点进入了我们规定的目标区域，便可以在随机扩展树中找到一条由从初始点到目标点的路径Γ。

算法 1.1 展示了经典 RRT 算法的伪代码，与上述算法流程一致。

算法 1.1 经典 RRT 算法

输入： 地图$\mathcal{M}$，起点x_{init}，终点x_{goal}
输出： 一条从起点x_{init}到终点x_{goal}的路径Γ
$\mathcal{T}$.init()
for $i = 1$ **to** MaxIter
  x_{rand} = Sample($\mathcal{M}$)
  x_{near} = Near(x_{rand}, $\mathcal{T}$)
  x_{new} = Steer(x_{rand}, x_{near}, StepSize)
  E_i = Edge(x_{new}, x_{near})
  if CollisionFree($\mathcal{M}$, E_i)
    $\mathcal{T}$.AddNode(x_{new})
    $\mathcal{T}$.AddNode(E_i)
  if $x_{new} \approx x_{goal}$
    Success()

1.3 RRT 算法评价

1.3.1 RRT 算法的优点

RRT 算法是一个应用广泛、性能较佳的算法。

（1）速度较快

在一般的场景中，RRT 具有比 A*、D*等上世纪的传统算法更快的速度，对计算资源的消耗量少。

（2）概率完备

RRT 算法作为一个路径规划算法，具有概率完备性。当采样空间有限时，且路径规划问题有解时，RRT 算法可以在有限时间内求出可行解。

但在实际操作中，往往存在一些限制条件，例如总迭代步数，或总时间限制，这种情况下如果迭代次数较少或规划时间不够长，很有可能不能找出实际存在的可行路径。

（3）适应二维、三维空间

从 RRT 的原理（算法 1.1）中可以看出，该算法没有指定工作空间的维数，即该算法既能在 2D 场景中使用（地面机器人、平面机械臂），也能在 3D 场景中使用（无人机、三维机械臂）。

1.3.2 RRT 算法的缺点

RRT 算法也有一定的缺点。

（1）并非最优路径

从 1.1 中我们可以看出，显然 RRT 算法得到的路径不是最优解。

（2）特殊环境的适应性差

从图 1.2 中我们可以看出，若空间中存在狭小的通道，则 RRT 算法的探索性能较差，在规定的步数较小时，出现图左情况的概率要远高于图右情况。

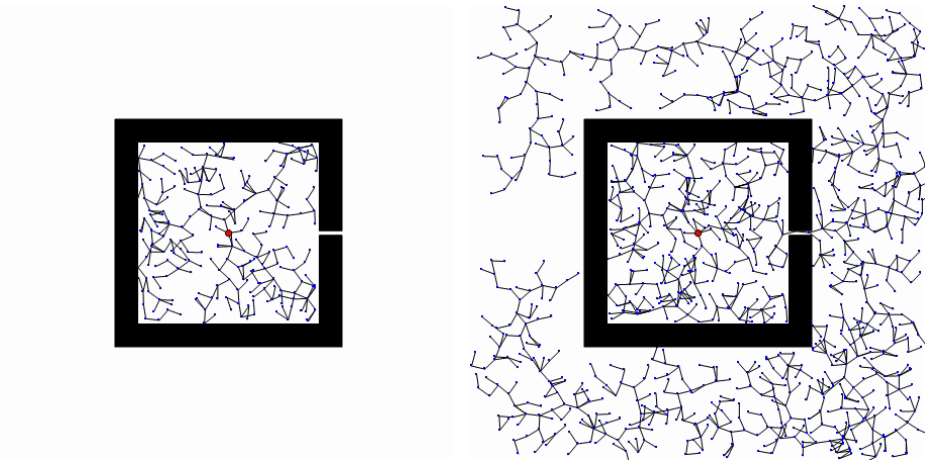


图 1.2 RRT 算法在狭小通道上性能较差

（3）纯采样的方式在空间较大时效率较低

纯基于采样，不依靠启发信息的 RRT 在探索较大规模的空间时效率低，这一点从 RRT 算法的随机采样特性中很容易看出。

2. 几种 RRT 衍生算法

针对经典 RRT 算法的缺点，有学者提出了一些衍生算法，例如基于概率 P 的 RRT、RRT-Connect[5]、RRT*[6]、Dynamic-RRT[8]等，这里我们分别进行介绍。

2.1 衍生一：基于概率 P 的 RRT 算法

为了加快随机树收敛到目标位置的速度，基于概率 P 的 RRT 算法在随机树扩展的步骤中引入一个概率 P，根据概率 P 的值来选择树的生长方向是随机生长还是朝向目标位置生长。

算法 2.1 基于概率 P 的 RRT 算法

输入： 地图$\mathcal{M}$，起点x_{init}，终点x_{goal}
输出： 一条从起点x_{init}到终点x_{goal}的路径Γ
$\mathcal{T}$.init()
for i = 1 **to** MaxIter
  x_{rand} = Sample($\mathcal{M}$, x_{goal}, P)
  x_{near} = Near(x_{rand}, $\mathcal{T}$)
  x_{new} = Steer(x_{rand}, x_{near}, StepSize)
  E_i = Edge(x_{new}, x_{near})
  if CollisionFree($\mathcal{M}$, E_i)
    $\mathcal{T}$.AddNode(x_{new})
    $\mathcal{T}$.AddNode(E_i)
  if $x_{new} \approx x_{goal}$
    Success()

算法 2.1 与算法 1.1 的区别是采样的方式，见算法 2.1 的蓝色行。算法 1.1 完全是在地图的无障碍区域中随机采样，而算法 2.1 以启发式方式，以给定概率 P 朝目标方向扩展 RRT，以 1-P 概率随机采样，通常这个概率 P 值取 0.1~0.2。通过引入向目标生长的机制，该扩展算法加速路径搜索的收敛速度。

2.2 衍生二：RRT-Connect 算法

RRT-connect 算法又称双边 RRT 算法，是经典 RRT 算法的衍生算法，它以起始点x_{init}和终点x_{goal}为根结点，同时生长两棵快速扩展随机树来搜索状态空间，直到两棵树相交时获得一个路径规划的可行解。RRT-connect 算法比起经典 RRT 算法来说效率更高。在 RRT-connect 算法中，如何判断两棵树相交是主要需要解

决的问题。RRT-connect 算法的基本流程是两棵树$\mathcal{T}_a$和$\mathcal{T}_b$交替采样，然后判断两棵树是否相交。判断相交的方法是：每棵树将一次采样的x_{new}依次与另一棵树的全部有效结点计算距离，如果最小距离小于阈值则视为两棵树相交，即可以找到一组结点，使得两棵树联通。RRT-Connect 算法也可以使用启发式的方法来进行采样。即在交替采样时扩展完第一棵树的新结点x_{new}后，以这个点作为第二棵树的目标点，第二棵树朝着x_{new}方向扩展。算法 2.2 展示了 RRT-connect 算法的伪代码。

算法 2.2 RRT-connect 算法

输入： 地图$\mathcal{M}$，起点x_{init}，终点x_{goal}
输出： 一条从起点x_{init}到终点x_{goal}的路径Γ

$\mathcal{T}_a$.init()
$\mathcal{T}_b$.init()
for i = 1 **to** MaxIter
  x_{rand} = Sample($\mathcal{M}$) **or** HeuristicSample($\mathcal{M}$, P)
  x_{near} = Near(x_{rand}, $\mathcal{T}_a$)
  x_{new} = Steer(x_{rand}, x_{near}, StepSize)
  E_i = Edge(x_{new}, x_{near})
  if CollisionFree($\mathcal{M}$, E_i)
    $\mathcal{T}_a$.AddNode(x_{new})
    $\mathcal{T}_a$.AddNode(E_i)
  if Connect($\mathcal{T}_a$, $\mathcal{T}_b$)
    Success()
  Swap($\mathcal{T}_a$, $\mathcal{T}_b$)

对于狭小通道的情况，经典的 RRT 算法往往需要很长时间才能通过随机采样通过狭小通道，而 RRT-connect 由于是从两头同时往中间搜索，在遇到狭小通道时，能比 RRT 更快速高效地寻找到路径。该算法得到的路径如图 2.1 所示。

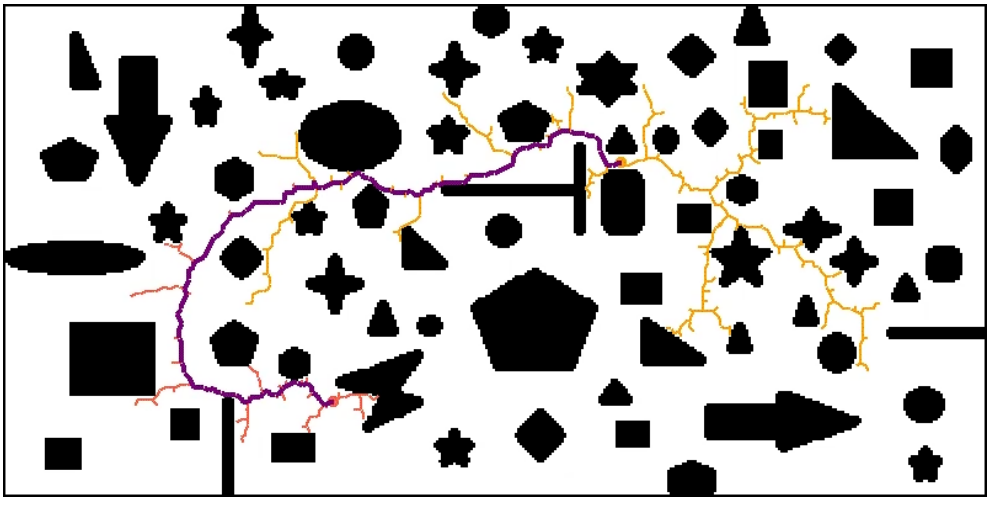


图 2.1 RRT-Connect 算法示意图

2.3 衍生三：RRT*算法

传统的 RRT 算法搜索得到的路径往往不是全局最优的，于是，RRT 算法的改进型——RRT*应运而生。RRT*算法流程与 RRT 算法流程基本相同，不同之处就在于最后加入将x_{new}加入搜索树$\mathcal{T}$后的父结点选择（ChooseParent）及重连（Rewire）过程，如算法 2.3 的绿色行所示。

算法 2.3 RRT*算法

输入： 地图$\mathcal{M}$，起点x_{init}，终点x_{goal}
输出： 一条从起点x_{init}到终点x_{goal}的路径Γ
$\mathcal{T}$.init()
for i = 1 **to** MaxIter
  x_{rand} = Sample($\mathcal{M}$)
  x_{near} = Near(x_{rand}, $\mathcal{T}$)
  x_{new} = Steer(x_{rand}, x_{near}, StepSize)
  E_i = Edge(x_{new}, x_{near})
  if CollisionFree($\mathcal{M}$, E_i)
    X_{near} = NearC($\mathcal{T}$, x_{new}, R)
    x_{min} = ChooseParent(X_{near}, x_{near}, x_{new})
    $\mathcal{T}$.AddNode(x_{new})
    $\mathcal{T}$.AddNode(E_i)
    $\mathcal{T}$.Rewire()
  if $x_{new} \approx x_{goal}$
    Success()

图 2.2 展示了 RRT*算法选择父结点及重连的过程。

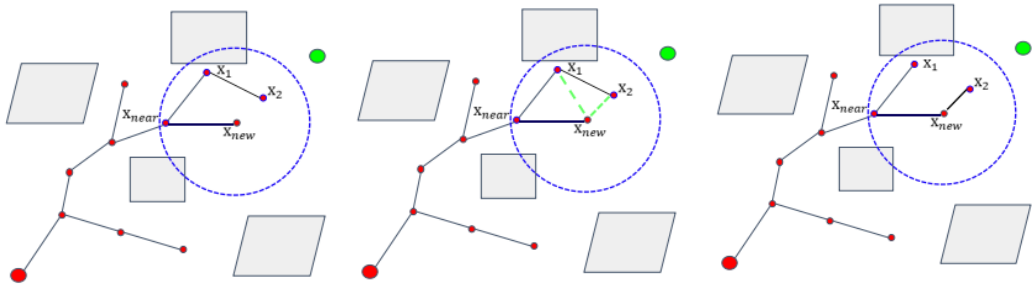
第一，在新加入结点后进行父结点选择。

RRT*中，我们在碰撞检测成功后，搜索以 R 为半径，以x_{new}结点为圆心范围内的搜索树上的相邻结点集合X_{near}，即算法 2.3 中的 NearC($\mathcal{T}$, x_{new}, R)函数，如图 2.2(a)所示。此处，X_{near}包含x_{near}、x_1、x_2三个结点。后裔依次比较以这个X_{near}中结点作为x_{new}的父结点的代价（一般为到起点x_{init}的路径长度），选取代价最小的结点作为x_{new}的父结点，即算法 2.3 中的 ChooseParent(X_{near}, x_{near}, x_{new})函数。

第二，对搜索树$\mathcal{T}$进行重连。

我们需要对X_{near}集合中的所有叶子结点到起始结点的代价进行判断，若存在更短路径，则重新修改叶子结点的父结点，从而完成重连。图 2.2(a)(b)中，x_2结点的父结点原本为x_1，由于x_2经过x_{new}到起始点的路径代价更小，因此，这里

修改x_2的父结点为x_{new}。同理，对其他相邻结点都进行这样的操作，便完成了重连。



(a) (b) (c)

图 2.2 RRT*算法示意图

RRT*算法在搜索到一条可通行路径后，并不会停止迭代，而是继续进行重连操作，直到迭代收敛或达到迭代次数上限，从而得到一条接近全局最优的可通行路径。当 RRT*找到一条可通行路径时，其采样函数依然在整个地图空间中进行均匀采样，然而进行重连等操作，这样方式致使 RRT*算法需要较大的计算量。

随着迭代轮数的增加，RRT*算法得到的可行路径向最优路径收敛，因此 RRT*算法是渐进优化的。

3. 仿真实验结果与分析

3.1 基于 Python 的仿真

我们利用 PathPlanning 仓库[9]，编写了 Python 程序，对经典 RRT 算法及其衍生算法基于概率 P 的 RRT 算法、RRT-connect 算法和 RRT*算法进行了仿真实验。我们设置了一个 30×50 的场景，并在其中放置了一些长方形及圆形障碍物。设置起始点为(2,2)，目标点为(49,24)。

（1） 经典 RRT 算法

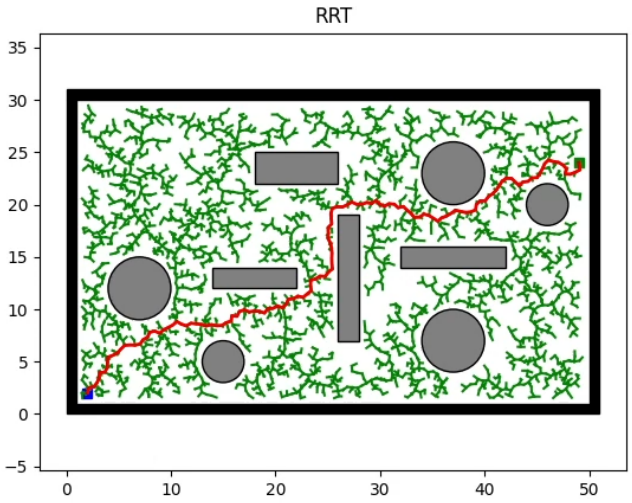


图 3.1 经典 RRT 算法规划结果

图 3.1 为经典 RRT 算法在实验场景下的运行结果，耗时 38s，可以看出搜索速度相对较慢。

通过 10 次实验，得到经典 RRT 算法在该场景下的平均耗时为 28s。

（2） 基于概率 P 的 RRT 算法

我们设置概率参数 P=0.05，P=0.2，对基于概率 P 的 RRT 算法进行仿真。

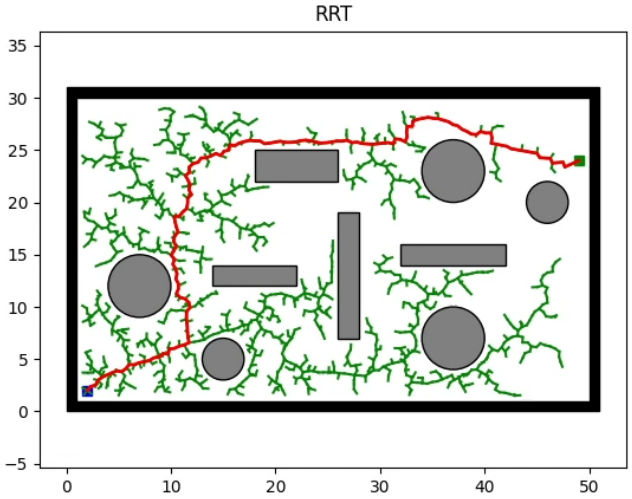


图 3.2 基于概率 P 的 RRT 算法（P=0.05）规划结果

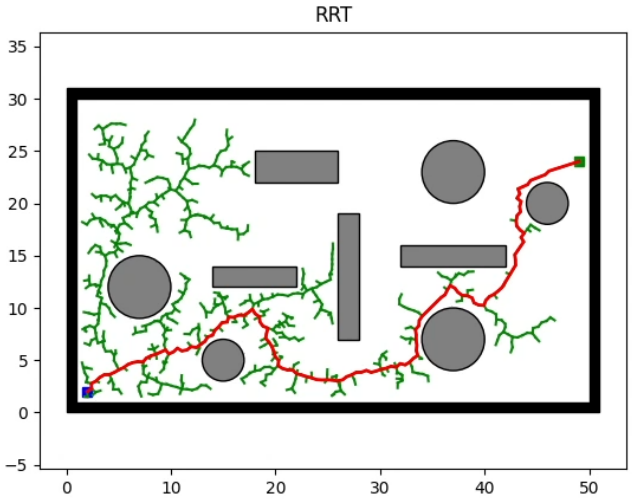


图 3.3 基于概率 P 的 RRT 算法（P=0.2）规划结果

图 3.2 为 P=0.05 时基于概率 P 的 RRT 算法在实验场景下的运行结果，耗时 22s，可以看出搜索速度较经典 RRT 算法有很大提高；图 3.3 为 P=0.2 时基于概率 P 的 RRT 算法在实验场景下的运行结果，耗时 3.5s；继续增大 P，运行耗时并不会继续减少，甚至增加。

通过 10 次实验，得到不同 P 值下基于概率 P 的 RRT 算法在该场景下的平均耗时，如表 3.1 所示。

表 3.1 不同 P 值下基于概率 P 的 RRT 算法在该场景下的平均耗时

P	平均耗时/s
0（经典 RRT 算法）	28
0.05	20
0.1	11
0.2	5.7
0.3	6.8
0.5	11.7
0.8	25.2

可以看出，随着 P 参数的增大，基于概率 P 的 RRT 算法的平均耗时呈现先减后增趋势。因此对于不同的场景，需要找到合适的 P 使得算法性能相对较佳。

（3） RRT-connect 算法

我们设置 RRT-connect 的贪婪参数 P=0.2，得到如图 3.4 所示的运行结果，本次实验耗时 1.2s。

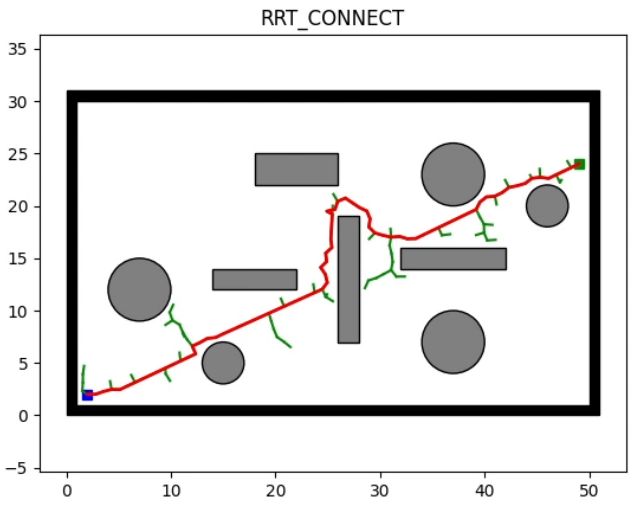


图 3.4 基于概率 P 的 RRT 算法（P=0.2）规划结果

通过 10 次实验，得到 RRT-connect 算法在该场景下的平均耗时为 1.38s。

（4） RRT*算法

3.5 和图 3.6 展示了实验环境下结点数量 N=1000 及 N=10000 时的 RRT*算法运行结果。从图中可以看出，N 较小时，RRT*算法便可以找到可行解，但该解并非最优解；随着 N 的增加，求得的可行解逐渐向最优解收敛。

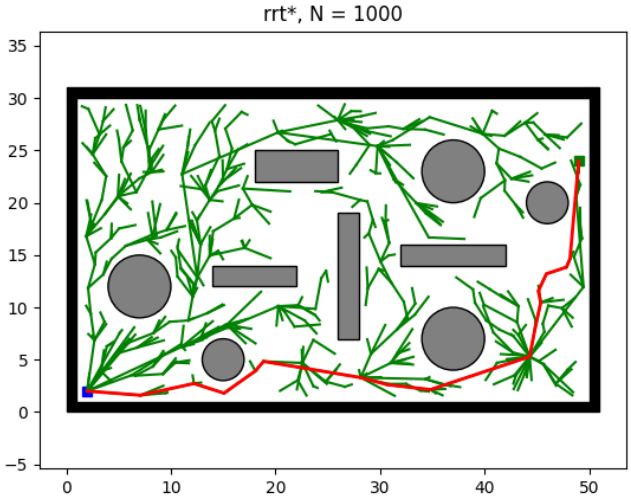


图 3.5 结点数量 N=1000 的 RRT*算法规划结果

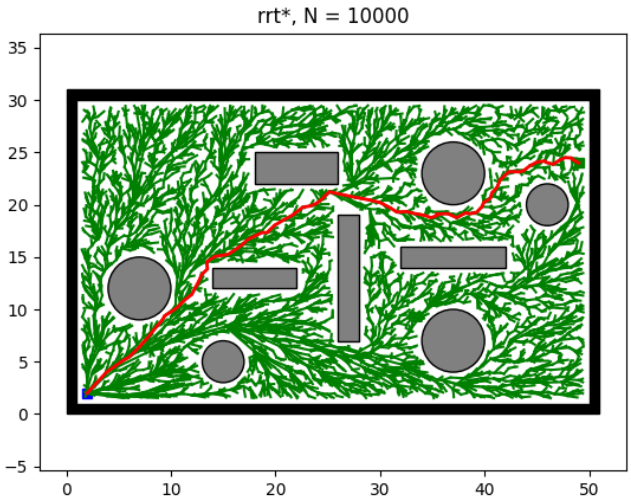


图 3.6 结点数量 N=10000 的 RRT*算法规划结果

通过各十次的实验，我们求得 N=1000 时的平均运行时间为 5s，N=10000 时的平均运行时间为 76s。因此为使 RRT*算法在解的最优性与求解速度中达到平衡，应设置较为合适的迭代轮数。

（5） Dynamic-RRT 算法

传统的 RRT 系列算法只可以在静态环境中求解出可行路径，若可行路径上出现了障碍物，则需要重新规划路径，Dynamic-RRT 使用重规划方法来解决这一问题。

图 3.7 展示了实验环境下 Dynamic-RRT 的路径规划结果。图中，蓝色路径为放置障碍物前的可行解，红色路径为放置障碍物后的可行解，绿色路径为放置障碍物前的 RRT 树路径，黄色路径为重规划的 RRT 树路径。从图中可以看出，放置障碍物后整课 RRT 树并不需要全部重新规划，新规划的路径需要依靠原规划的路径。

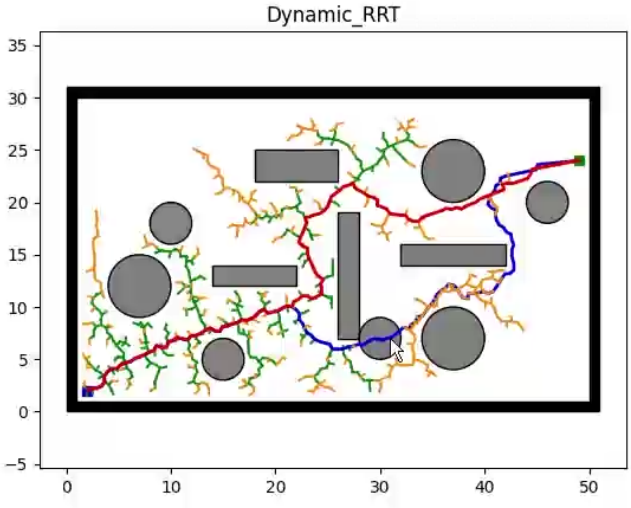


图 3.5 Dynamic-RRT 算法规划结果

（6） RRT 算法在 3D 环境中的仿真

RRT 算法不仅能用于 2D 环境中的路径规划，也能用于 3D 环境中的路径规划，图 3.8 展示了一个 3D 环境中使用 P=0.2 时基于概率 P 的 RRT 算法的路径规划结果。

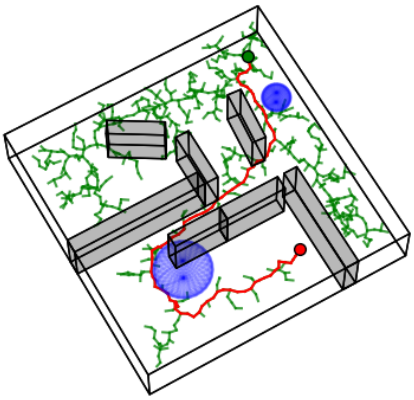


图 3.8 3D 环境下的 RRT 算法规划结果

3.2 基于 Gazebo 的仿真

我们在 Ubuntu 系统上，基于 ROS 编写了利用 RRT 及衍生算法无人车路径规划的程序（全局规划器插件）进行了仿真验证。图 3.9~3.11 中展示了 RRT*算法下的路径规划结果图 3.12 展示了带有路径优化的 RRT*算法的仿真结果，图 3.13 展示了 RRT-connect 算法的仿真结果，动态过程请见展示 PPT。

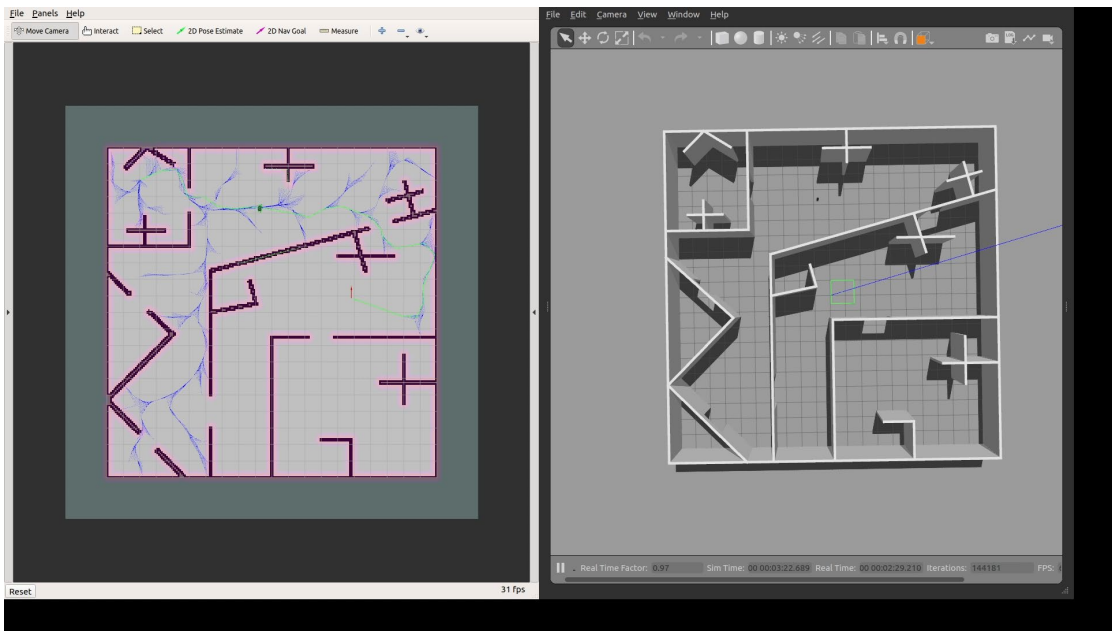


图 3.9 RRT*算法在 Gazebo 中的仿真：目标点 1

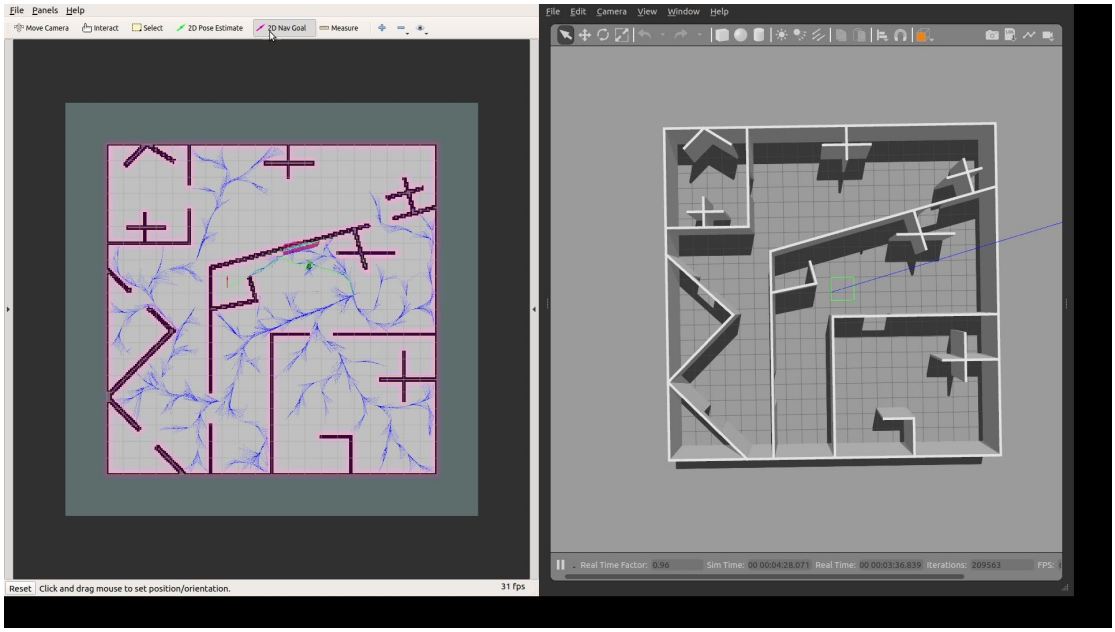


图 3.10 RRT*算法在 Gazebo 中的仿真：目标点 2

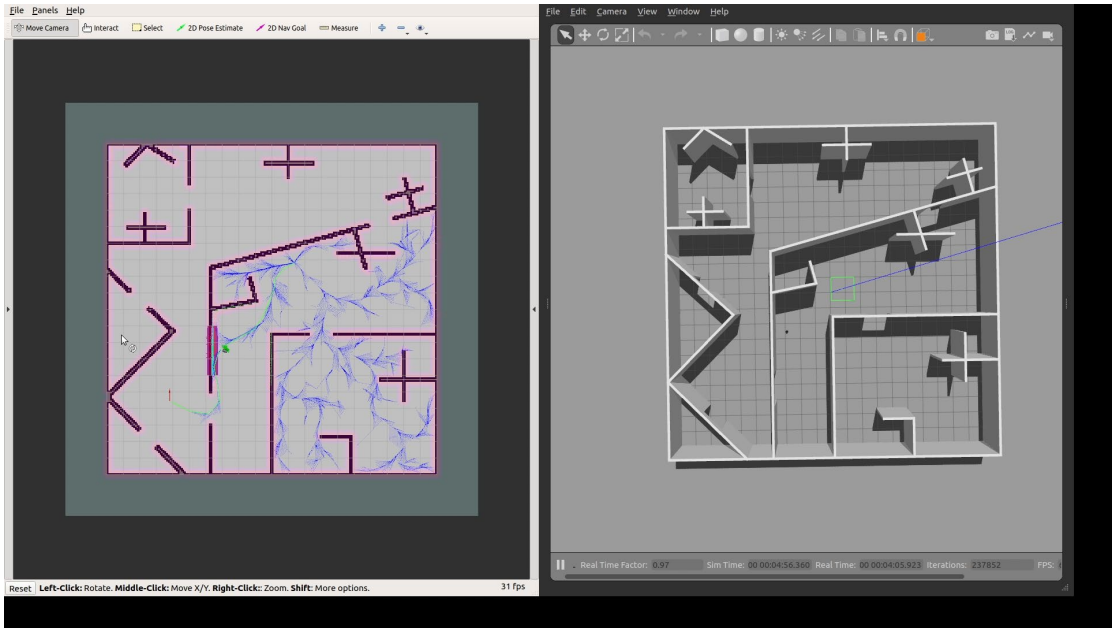


图 3.11 RRT*算法在 Gazebo 中的仿真：目标点 3

从图 3.9~3.11 中我们可以看出，机器人可以通过 RRT*算法来找到连接起点和目标点的可行路径，且很显然该路径并不是最优的，因为路径中存在较多弯曲。因此，机器人按 RRT 路径前往目标点时需要耗费较多能量。

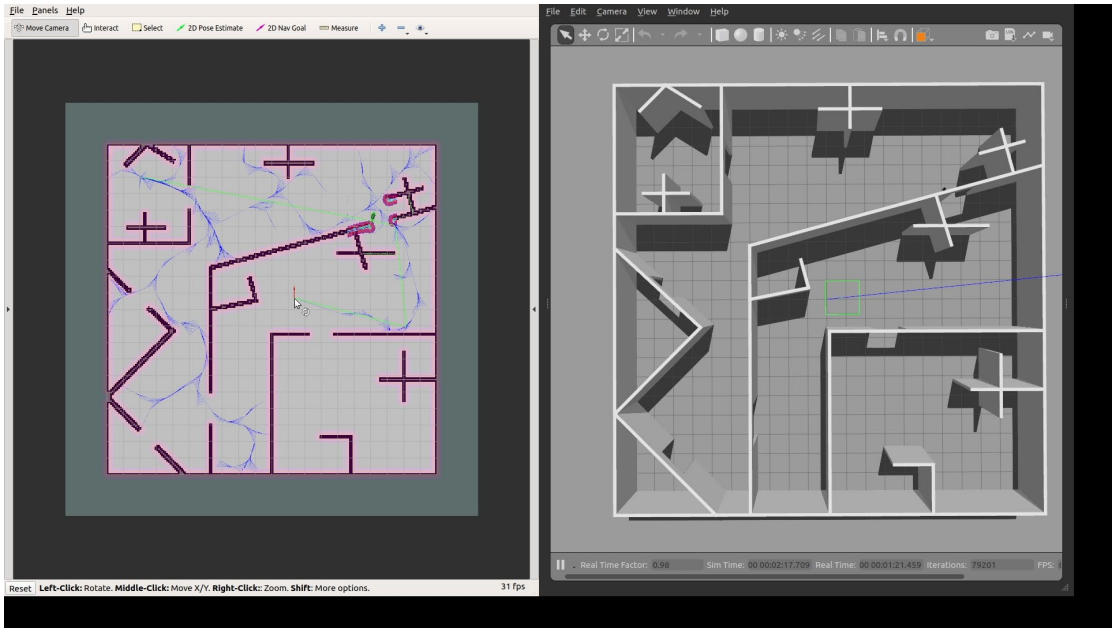


图 3.12 带有路径优化的 RRT*算法在 Gazebo 中的仿真

为减少机器人的能量消耗，我们对机器人进行路径优化。路径优化的方法是，按贪心算法找到无障碍区域内首尾均为 RRT 结点的最长的线段，即“化曲为直”，避免过多的拐弯，使路径相对更优。图 3.12 展示了该方法的路径规划结果。

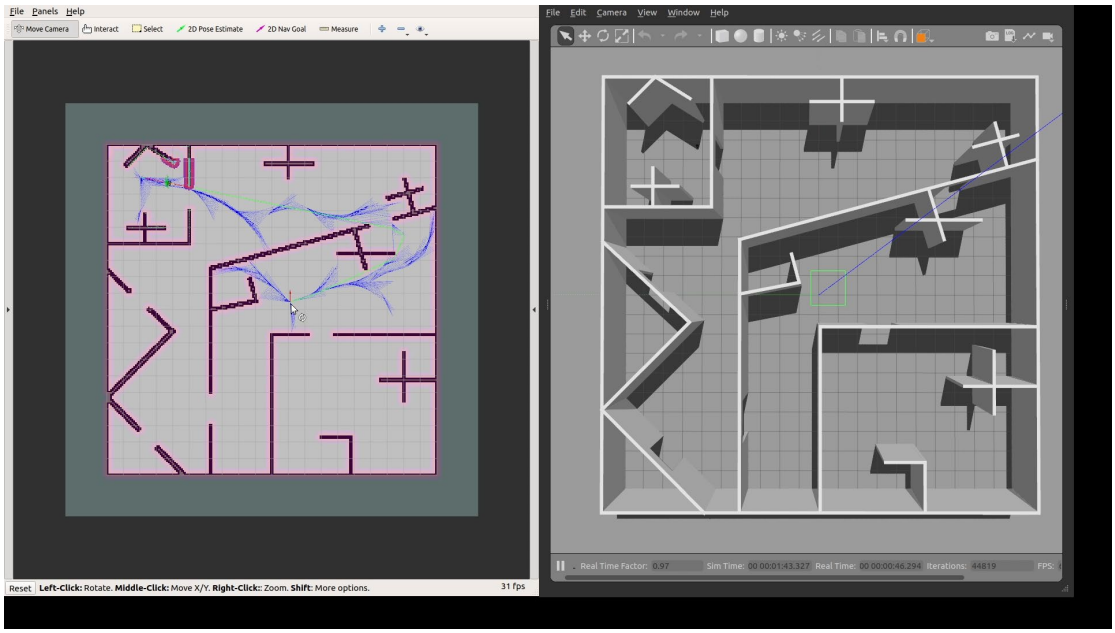


图 3.13 RRT-connect 算法在 Gazebo 中的仿真

RRT-connect 算法通过建立两颗 RRT 来寻找可行路径，具有更高的效率，图 3.13 很好地表明出 RRT-connect 算法的高效性。

4. RRT 方法的其他应用

RRT 方法不仅可以用于移动机器人在已知地图下的路径规划，还可以用在很多其他方面。这里我们展示了 RRT 算法在另外两个任务上的应用，包括在机器人未知环境探索上的应用和 RRT 方法在机械臂路径规划上的应用。

4.1 RRT 方法在机器人未知环境探索上的应用

Umari 等人[10]在 2017 年提出了可以将 RRT 算法应用在机器人探索任务上，与基于图像处理的方法相比，探索效率更高，所需代价更低。本节中，将对该任务和相应算法做具体介绍，并展示仿真结果。

4.1.1 机器人未知环境探索任务简介

自主探索（有的文献中也称其为主动 SLAM）是一种移动机器人在任务过程中动态地发现周围环境并进行实时建图的方法，适用于测绘、搜救和巡检等领域，对构建无人系统来说具有重大意义。

在自主探索任务中，机器人配备有激光雷达、相机等传感器，用于实时构建环境的地图，同时机器人具备移动能力，机器人在移动的过程中不断感知环境，直至建立好整个环境的地图。

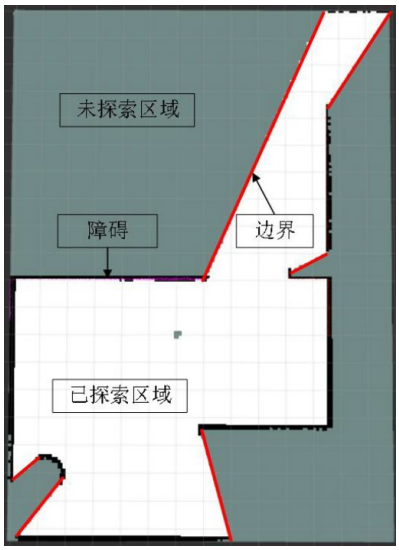


图 4.1 机器人环境探索示意图

图 4.1 展示了机器人未知环境探索的一些概念。图中，白色区域为机器人已探索区域，也称已建图区域，绿色部分为机器人为探索区域，也称未知区域，黑

色部分为障碍物，例如墙壁、桌椅等。红色部分最为重要，我们称为“前沿点（Frontiers）”，表示机器人已知区域和未知区域的界限。机器人在前沿点中选择一个作为目标，前往该点，以此来扩大已知区域。地图更新后，前沿点的位置也会随之变化，随后机器人前往新的边界点，继续扩大地图，不断重复这个过程，以达到完整建图的目的。

机器人探索任务中有两大重点，一是如何计算出前沿点的位置，二是前往哪个前沿点，即前沿点决策问题。这两大重点决定了的机器人探索效率和效果的好坏。

传统方法中一般依靠图像处理的方式来计算前沿点的位置，但图像处理的方式计算量较大，在杂乱的环境中容易出现误判，导致探索效率低。

对于前沿点决策，较早期的一般直接让机器人前往离自身最近的前沿点，新世纪以来一般使用代价函数法，使用基于 RRT 的方法在 2017 年[10]被提出。此外，还有更加新的方法，例如近两年被提出的人工势场法[11]和强化学习法[12]。

4.1.2 RRT 方法在该任务上的应用

该文献中提出[10]，使用两颗 RRT 树来进行前沿点检测，如图 4.2 所示。图中，黄色圆圈表示机器人目前所在位置，红色线条代表 Local RRT，蓝色线条代表 Global RRT，绿色方块代表 RRT 检测到的前沿点。

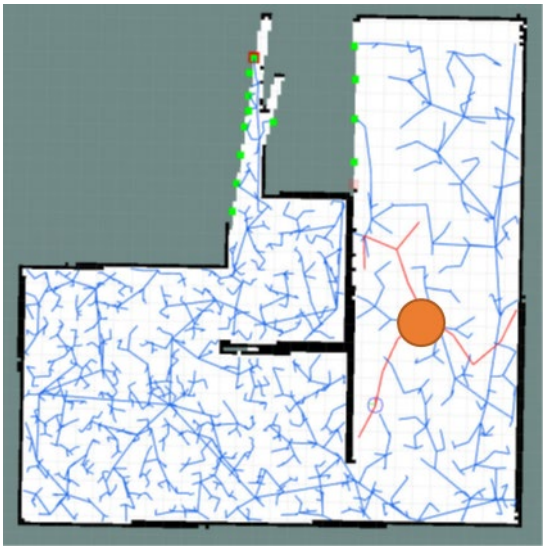


图 4.2 利用 RRT 来进行前沿点检测

下面，我们分别来介绍 Global RRT 和 Local RRT 的作用，两者相互配合用于确定前沿点，整体架构如图 4.3 所示。

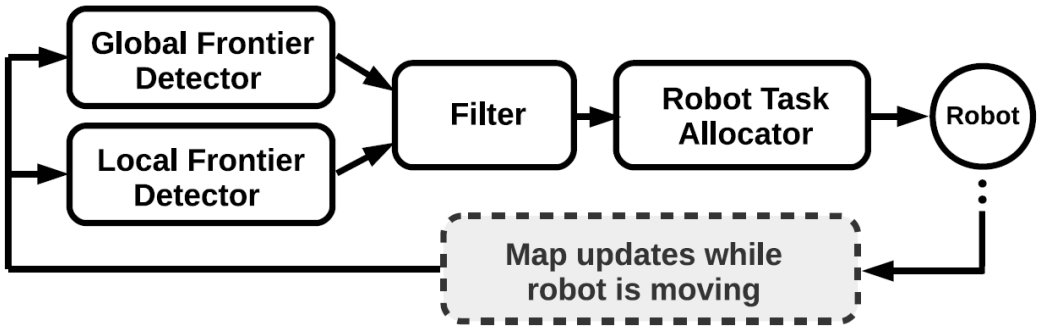


图 4.3 RRT 探索算法架构图

Local RRT 和经典 RRT 算法相似，不同的是它并不存在目标点。当树枝生长到了未知区域，那么当前点就可以认为是前沿点。若探测到了前沿点，则将前沿点输入滤波器，随后 Local RRT 被重置，从机器人的当前位置开始重新生长。

Global RRT 和 Local RRT 的区别是其生命周期，机器人在起始位置开始构建 Global RRT，随着机器人的运动和地图的扩大，Global RRT 也随之扩大，Global RRT 找到前沿点后并不会被删除，而是继续生长。

两棵 RRT 树的相互配合，使得机器人可以找到其附近的前沿点，并不会遗漏地图中的一些小角落。

由于有两个检测器的存在，前沿点的检测会非常容易且频繁，过多的点可能是非常接近或是重合的，这时候滤波器模块对前沿点进行聚类，只储存一个中心点坐标，并且滤波器模块还会过滤掉不可达的无效点以及已经探索的前沿点。

该论文[10]使用代价函数法来对机器人分配任务，即计算每个候选目标点的代价函数，并令机器人前往代价最小的那一个。

4.1.3 实验结果及分析

我们基于 ROS 和 Gazebo 以仿真方式复现了该文献[10]的工作，在如图 4.4 所示的仿真环境里进行一次探索任务。机器人以环境的中心为起点，开始自动建图，直到建完整个环境的代价地图。其中，机器人 SLAM 方法使用了 ROS 中 TurtleBot 自带的 Gmapping 方法。

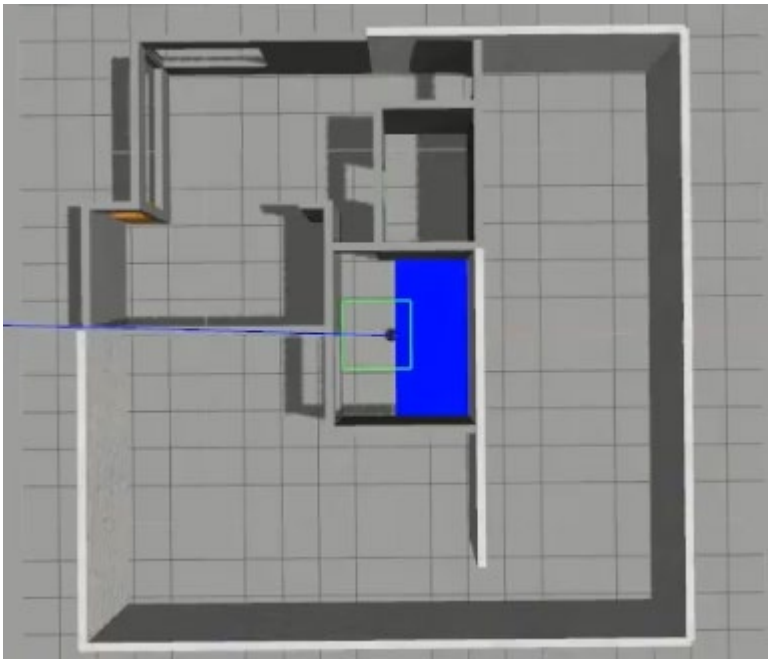


图 4.4 探索任务仿真环境

图 4.5(a)是完成探索任务过程中的一次截图，4.5(b)展示了一次机器人探索任务的结果，完整的仿真探索流程请见 PPT 中的视频。

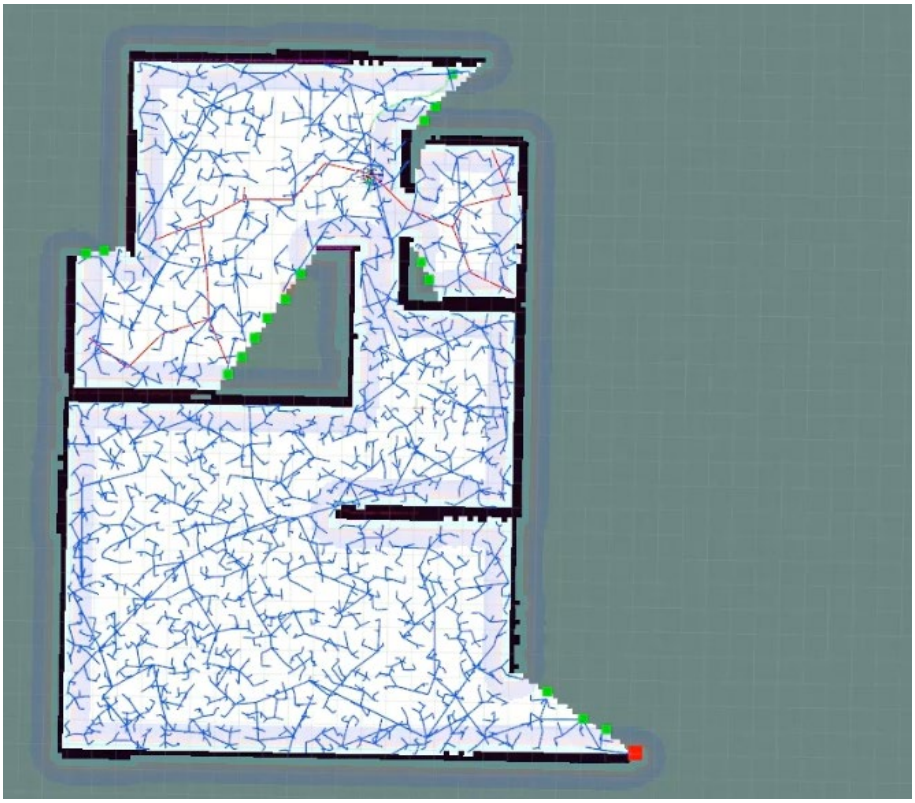


图 4.4(b) 机器人利用 RRT 算法进行未知环境探索过程

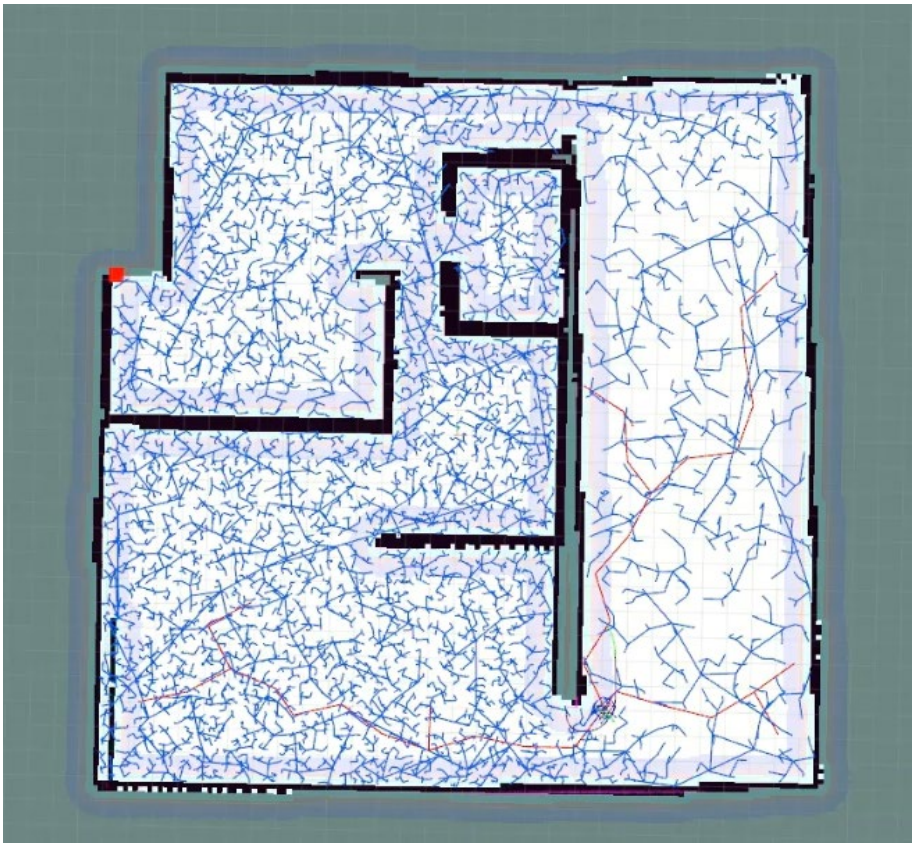


图 4.4(b) 利用 RRT 算法进行未知环境探索结果

从图 4.4 中可以看出，机器人可以利用 RRT 来寻找前沿点，并自主完成未知环境的探索任务。

4.2 RRT 方法在机械臂路径规划上的应用

4.2.1 机械臂运动学建模

本文采用 RobotAnno v6 机器人进行建模分析，RobotAnno v6 是高精度、多功能的消费级桌面机器人，是一款关节型串联机器人，总共有 6 个旋转轴（RRRRRR），其外形如图 4.6(a)所示。该机器人的特点是结构坚固，且抗扰动性强，采用多轴联动插补控制。

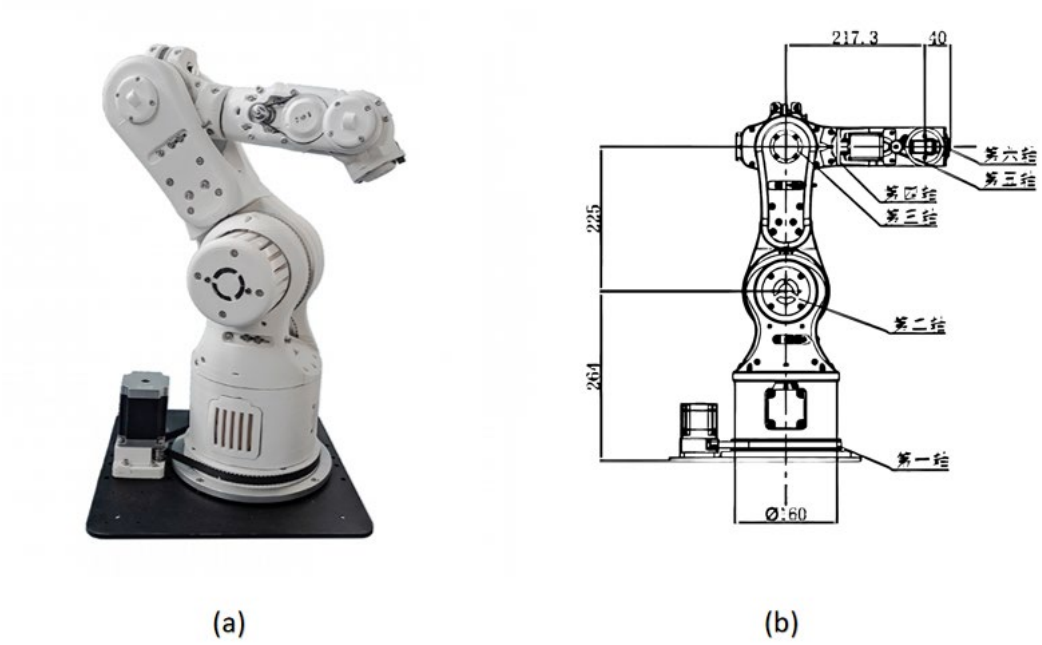


图 4.6 RobotAnno v6 机器人

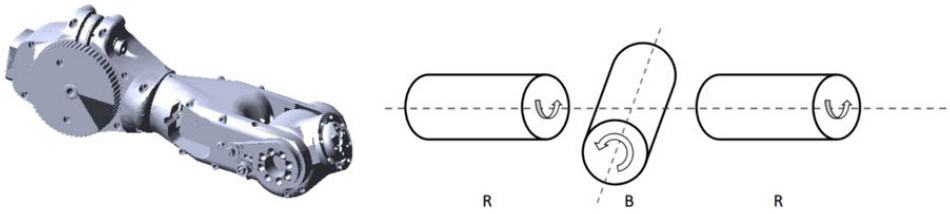


图 4.7 RBR 型腕部结构

如图 4.6(b)所示，RobotAnno v6 机器人采用仿人型的设计，包含了三个大关节：腰（waist）、肩（shoulder）和肘（elbow），分别对应三个较长的连杆：机体（body）、主臂（upperarm）和前臂（forearm），这一部分主要决定了机器人末端位置；在前臂末端还有三个较小的腕部关节，主要影响机器人末端姿态。RobotAnno v6 机器人腕部采用 RBR 型的机械结构，如图 4.7 所示，RBR 型腕部关节的三个关节轴线交于一点，符合 Piper 准则，可以求得运动学的封闭解。

运动学建模就是建立各关节坐标系，然后通过关节坐标系间的位姿变换关系来描述机器人运动。本文采用 MDH 参数法建模。MDH 参数法建模主要有两个步骤：一是根据关节轴向建立各关节坐标系，然后根据关节坐标系确定 MDH 参数。局部连杆间变换总共通过 4 个参数来描述，分别为连杆扭角 α_i，连杆长度

a_i，关节转角为θ_i，连杆偏移d_i，MDH 建模方法的局部连杆坐标关系如图 4.8 所示。

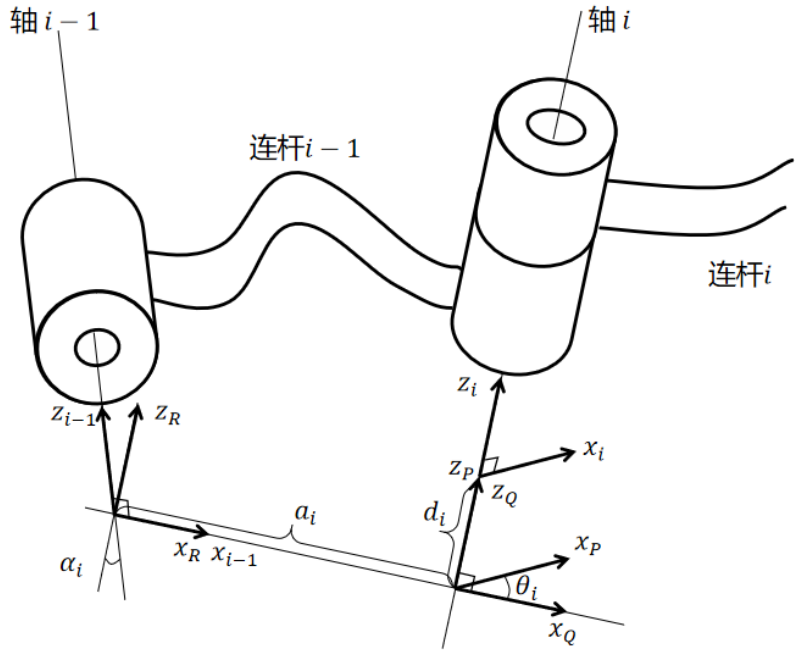


图 4.8 局部连杆关系与 DH 参数建模

MDH 参数法建立坐标系的步骤如下：

(1) 规定连杆i的驱动轴为关节i的z_i轴。

(2) 将z_i与z_{i+1}的公垂线作为x_i轴，方向由z_i关节指向z_{i+1}关节。

(3) 将z_i与x_i的交点作为坐标系原点O_i。

(4) y_i轴根据右手法则确定。

建立了坐标系，则 4 个 MDH 参数也随之确立，则关节坐标系$\{i-1\}$与$\{i\}$间的变换可以通过 4 步完成：

1. 绕x_{i-1}旋转α_i使z_{i-1}、z_i平行；
2. 沿x_{i-1}平移a_i使z_{i-1}、z_i重合；
3. 绕z_i旋转θ_i使x_{i-1}、x_i平行；
4. 沿z_i平移d_i使x_{i-1}、x_i重合。

表 4.1 RobotAnno 机器人 MDH 参数

i	$\alpha_i(rad)$	$a_i(m)$	$\theta_i(rad)$	$d_i(m)$
1	0	0	0	0.284
2	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$	0
3	0	0.225	0	0
4	$\frac{\pi}{2}$	0	π	0.2289
5	$\frac{\pi}{2}$	0	π	0
6	$\frac{\pi}{2}$	0	0	0.055

因此，连杆$i-1$两端关节坐标系间的齐次变换通式为

$$ {}_{i}^{i-1}T = Rot(x,\alpha_i)Trans(x,a_i)Rot(z,\theta_i)Trans(z,d_i) $$

$$ {}_{i}^{i-1}T = \begin{bmatrix} \mathrm{c}\theta_i & -\mathrm{s}\theta_i & 0 & a_i \\ \mathrm{c}\alpha_i\,\mathrm{s}\theta_i & \mathrm{c}\alpha_i\,\mathrm{c}\theta_i & -\mathrm{s}\alpha_i & -d_i\,\mathrm{s}\alpha_i \\ \mathrm{s}\alpha_i\,\mathrm{s}\theta_i & \mathrm{s}\alpha_i\,\mathrm{c}\theta_i & \mathrm{c}\alpha_i & d_i\,\mathrm{c}\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

通过上述步骤，可对 RobotAnno v6 串联机器人进行建模，其对应的 MDH 参数如表 4.1 所示。

那么从机器人基座到末端执行器的变换的齐次矩阵为

$$ {}_{tool}^{0}T = {}_{1}^{0}T\,{}_{2}^{1}T\cdots{}_{n}^{n-1}T\,{}_{tool}^{n}T $$

一般情况下，定义$\{0\}$为基坐标系，$\{tool\}$为固接在末端执行器处的工具坐标系。

4.2.2 机械臂轨迹规划

在机械臂操作任务中，根据任务需要规划出一条保持特定位姿的机械臂运动轨迹。规划流程如下：首先采用改进 RRT 算法规划出一条三维空间内的可行路径。指定机械臂在每一点的姿态，再根据机械臂在路径点的位姿进行逆运动学解

算得到机械臂的关节角，最后在关节空间进行三次多项式插值以得到平滑的机械臂轨迹。轨迹规划流程如图 4.9 所示。

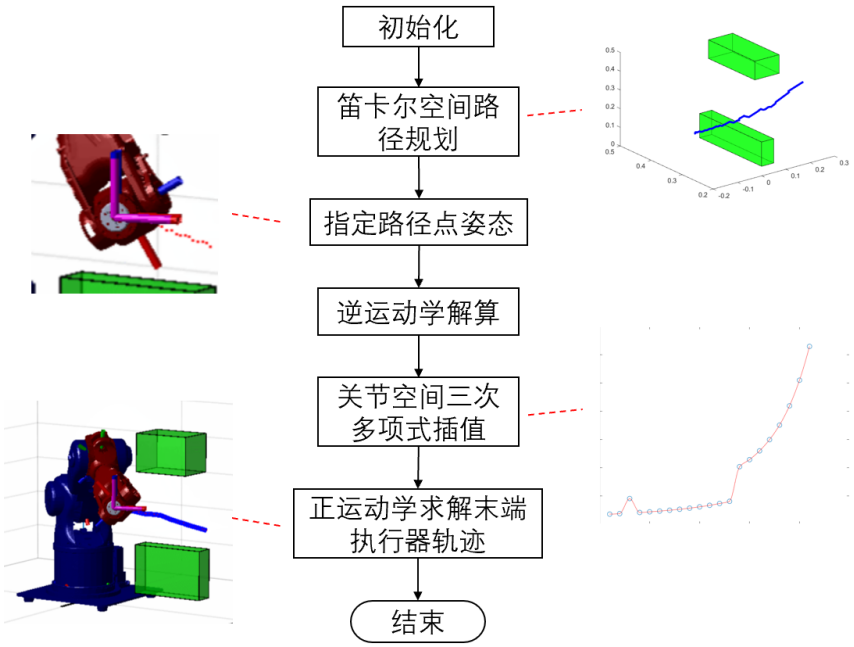


图 4.9 机械臂轨迹规划流程

4.2.3 基于 RRT 算法的任务空间路径规划

在机械臂搬运任务中，基本的要求是保证机械臂和操作人员的安全，令机械臂具有躲避障碍的能力，能够规划出一条从起点到终点的可行路径。本文采用 RRT 算法进行路径规划，并针对机械臂操作空间的特点对其进行了改进。

首先采用基于概率的 RRT 算法。在三维的工作空间中基本 RRT 算法搜索效率低，计算量大。基于概率的 RRT 算法可以保证路径向终点的趋向，选择适当的概率使得路径不会陷入局部最小值。

RRT 的特点是规划得到的路径不是最优解，采用 RRT*算法的思想对基于概率的 RRT 算法进行改进，可以使得路径优化，让机械臂运动轨迹更平滑，减小关节角度剧烈变化的概率，增加运动精度，维护机械结构。

4.2.4 基于抛物线插值的关节空间轨迹规划

关节空间的轨迹规划首先将在任务空间的期望的路径点通过逆运动学计算

得到期望的关节位置，然后在关节空间内给每个关节找到一个经过中间点到达目标终点的光滑函数，同时使得每个关节到达中间点和终点的时间是相同的，这样便可保证机械手工具能够到达期望的直角坐标位置。关节空间轨迹规划可以保证经过路径点，且可以避免机构的奇异点问题。

常用的插值方法有三次多项式插值，五次多项式插值，线性插值等。本文采用抛物线连接的线性函数插值，也称梯形速度插值。线性插值计算简单，但会使得关节的运动速度在起点和终点处不连续，也意味着需要产生无穷大的加速度，这显然是不希望的。因此可以考虑在起点和终点处用抛物线与直线连结起来，在抛物线段内，使用恒定的加速度来平滑地改变速度，从而使整个轨迹的位置和速度也是连续的。

在指定时间后，限定不同的加速时间，最大加速度，以及最大角速度，会得到不同的插值实现方式，需要根据机械机构的能力选择，如图 4.10 所示。

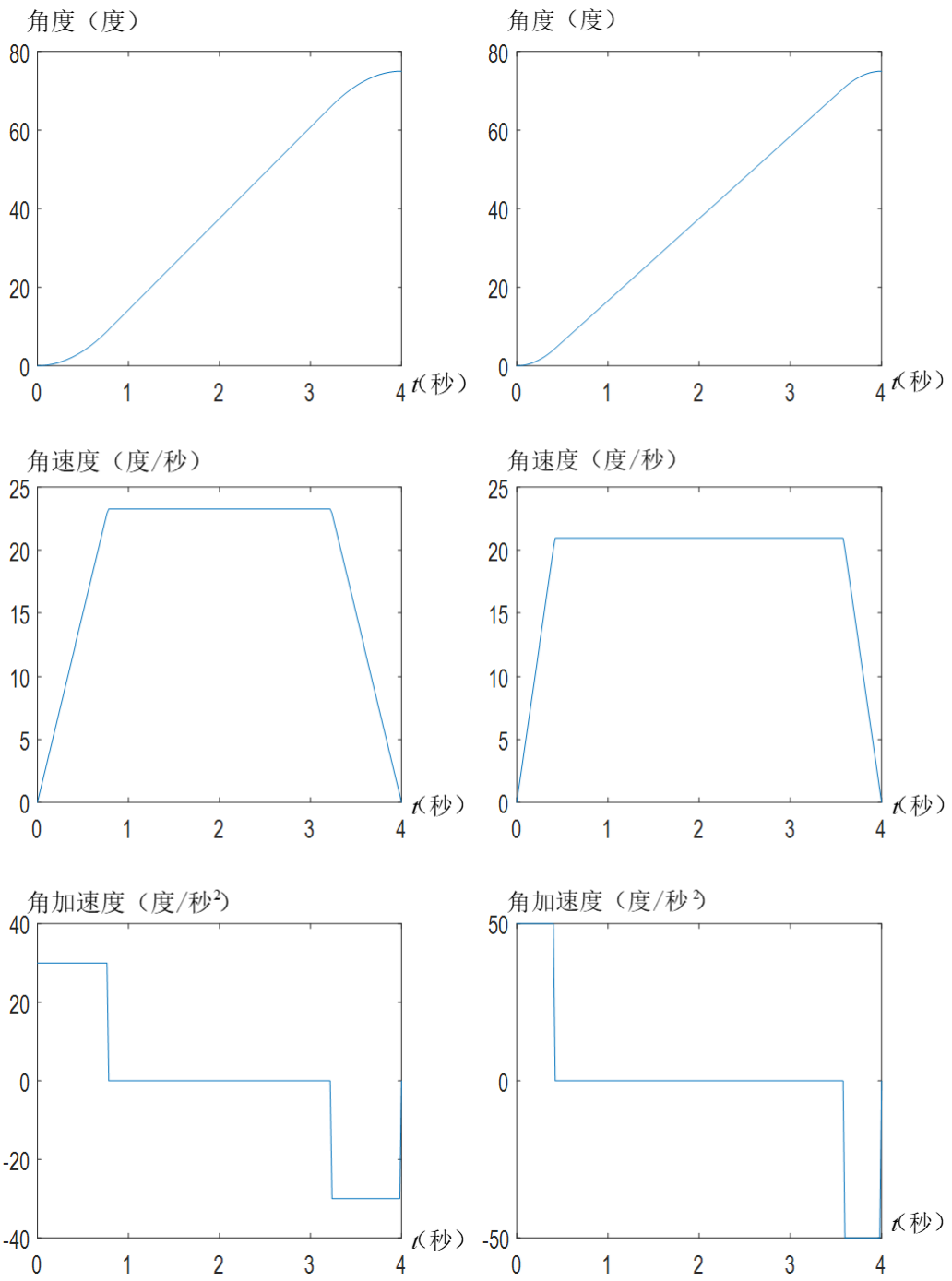


图 4.10 利用抛物线段连接的线性插值规划出的运动轨迹

4.2.5 机械臂轨迹规划实验

（1） RRT 路径规划实验

基于概率的 RRT 与 RRT*在三维空间寻径过程的随机树如图 4.11 所示。

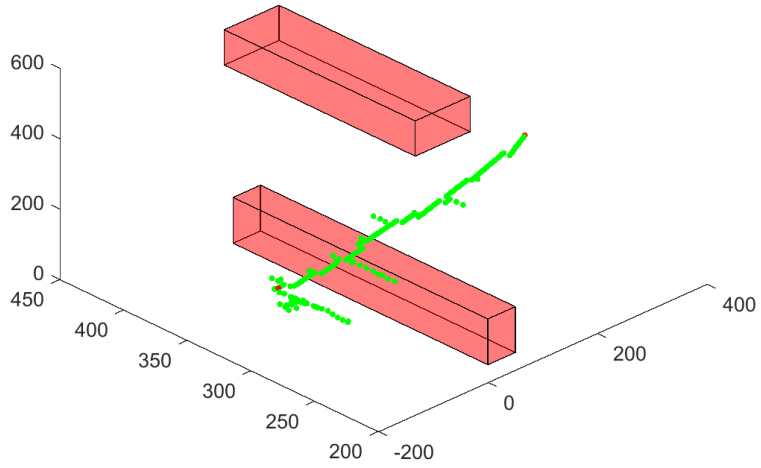


图 4.11(a) RRT 随机树

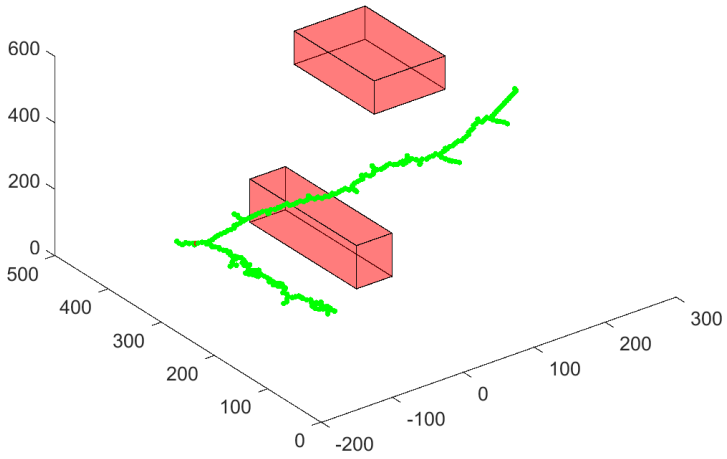


图 4.11(b) RRT*随机树

在相同起点和终点，相同障碍物的条件下，分别使用基于概率的 RRT 算法和基于概率的 RRT*算法，进行路径规划，得到的平均路径点点数为：

表 4.3 平均路径点点数

p-RRT	117
p-RRT*	15

改进 RRT 算法得到的机械臂工作空间内末端执行器路径如图 4.12 所示。

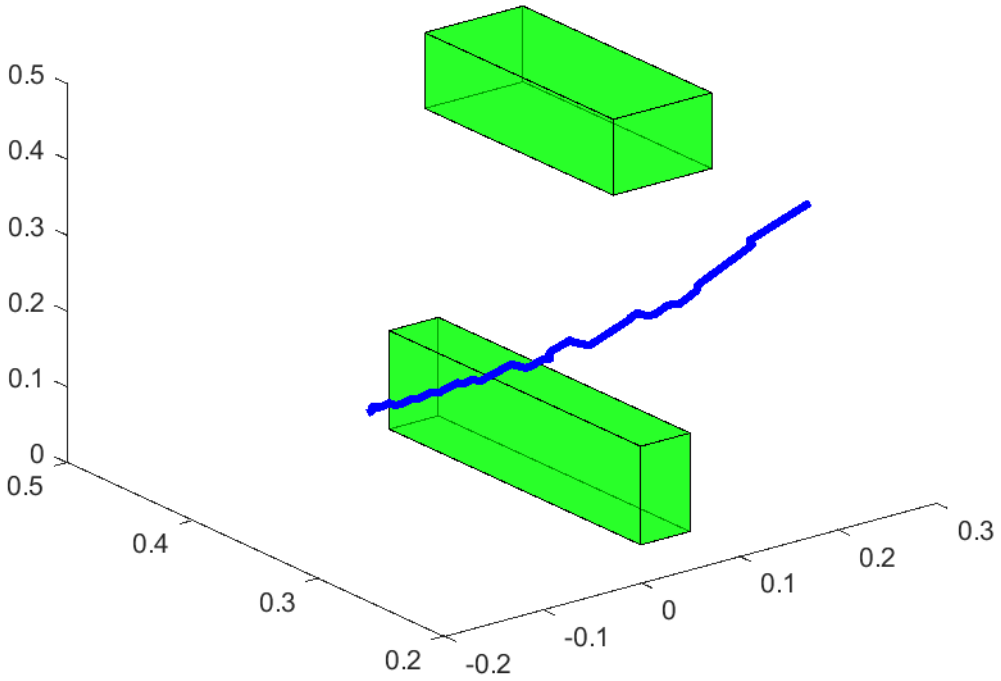


图 4.12 根据改进 RRT 算法的末端执行器路径

（2） 关节空间插值实验

对 RRT 算法得到的路径点指定位姿后，即可根据逆运动学得到机械臂的关节构型，此时需要对关节角进行插值以保证路径的顺滑。梯形速度插值后关节空间值如图 4.13 所示。

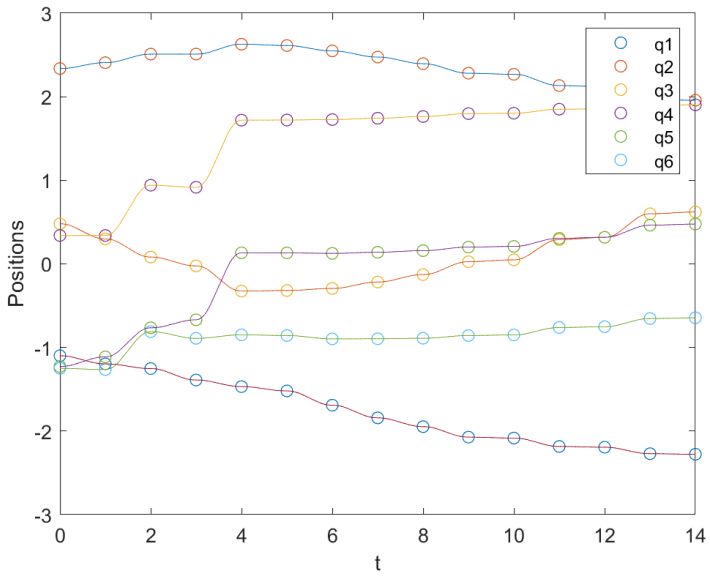


图 4.13(a) 关节角 q 插值前后

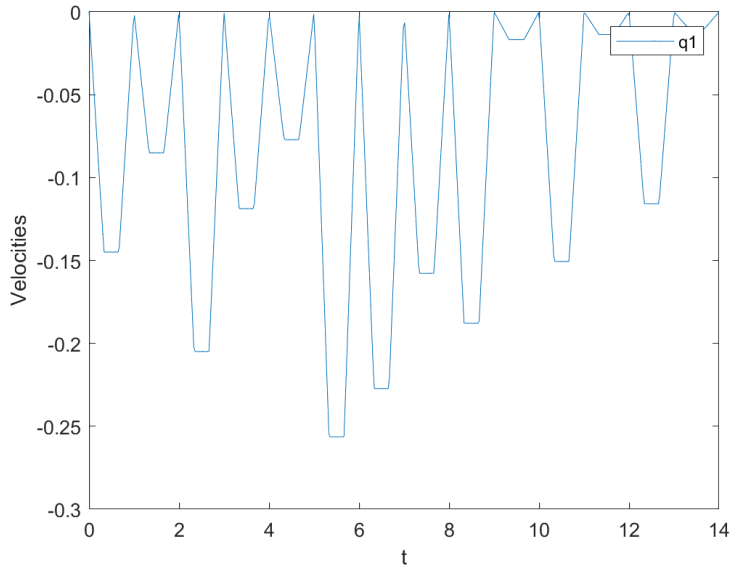


图 4.13(b) 关节角速度 $\dot{q}$ 插值后

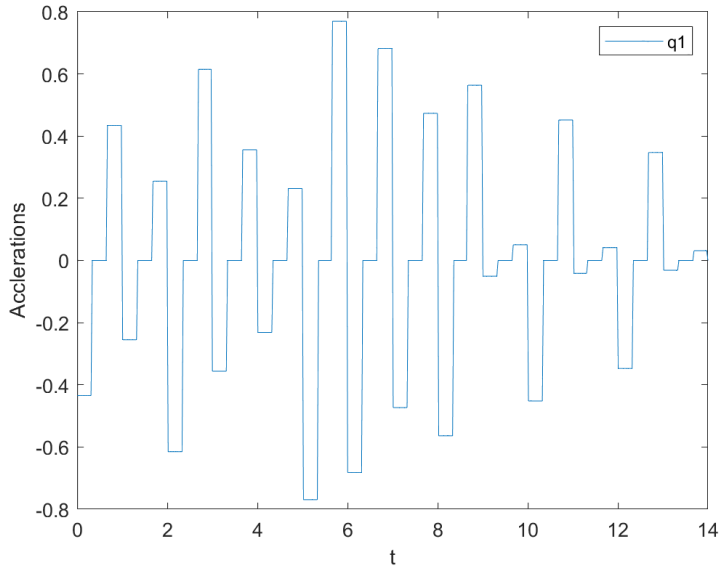


图 4.13(c) 角加速度 $\ddot{q}$ 插值后

从实验结果可以看出，在每两个插值点之间速度是对称的，加速度会有一个经过零点并且绝对值相反的变化。

（3） 机械臂轨迹规划实验

在机械臂运动过程中，指定末端执行器保持姿态

$$ {}^{0}_{6}R = \begin{bmatrix} & -1 & \\ 1 & & \\ & & 1 \end{bmatrix} $$

最终结果见 PPT 动态图，图 4.14 为截图。对比 p-RRT 与 p-RRT*算法，机械臂沿后者运动波动更小，结果更好。

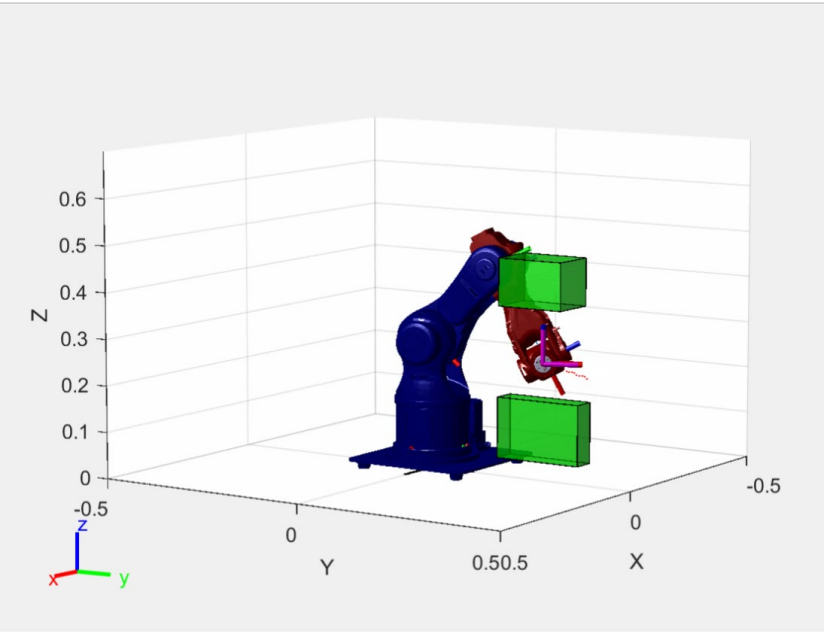


图 4.14 机械臂作业

由实验结果看出，机械臂较好地实现了轨迹规划。

5. 总结与展望

5.1 总结

本文对经典 RRT 算法及其衍生算法进行了理论介绍，并在 Python 和 Gazebo 对这些算法进行了仿真验证，对比了不同算法的区别和特性。

本文以机器人未知环境探索任务和机械臂轨迹规划任务为例，介绍了 RRT 算法在这些任务上的应用，给出了仿真结果与分析。

5.2 展望

RRT 算法提出的时间较早，随着研究工作的开展目前出现了一些更新的、更高效的 RRT 衍生算法，例如、Closed-Loop RRT*[13]、Informed RRT*[14]等，以及对 RRT 算法改动较大的基于采样的路径规划算法 ABIT*[15]、AIT*[16]等。

还可以探究 RRT 系列算法在其他任务上的应用，例如无人机搜救、多智能体协同作业等方面。

参考文献

[1] Hart P E, Nilsson N J, Raphael B. A formal basis for the heuristic determination of minimum cost paths[J]. IEEE transactions on Systems Science and Cybernetics, 1968, 4(2): 100-107.

[2] Stentz A. Optimal and efficient path planning for partially known environments[M]//Intelligent unmanned ground vehicles. Springer, Boston, MA, 1997: 203-220.

[3] LaValle S M. Rapidly-exploring random trees: A new tool for path planning[J]. 1998.

[4] Kavraki L E, Kolountzakis M N, Latombe J C. Analysis of probabilistic roadmaps for path planning[J]. IEEE Transactions on Robotics and automation, 1998, 14(1): 166-171.

[5] Kuffner J J, LaValle S M. RRT-connect: An efficient approach to single-query path planning[C]//Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). IEEE, 2000, 2: 995-1001.

[6] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning[J]. The international journal of robotics research, 2011, 30(7): 846-894.

[7] Niroui F, Zhang K, Kashino Z, et al. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments[J]. IEEE Robotics and Automation Letters, 2019, 4(2): 610-617.

[8] Ferguson D, Kalra N, Stentz A. Replanning with rrts[C]//Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. IEEE, 2006: 1243-1248.

[9] https://github.com/zhm-real/PathPlanning

[10] Umari H, Mukhopadhyay S. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees[C]//2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017: 1396-1402.

[11] Yu J, Tong J, Xu Y, et al. SMMR-explore: SubMap-based multi-robot exploration system with multi-robot multi-target potential field exploration method[C]//2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021: 8779-8785.

[12] Niroui F, Zhang K, Kashino Z, et al. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments[J]. IEEE Robotics and Automation Letters, 2019, 4(2): 610-617.

[13] Kuwata Y, Teo J, Fiore G, et al. Real-time motion planning with applications to autonomous urban driving[J]. IEEE Transactions on control systems technology, 2009, 17(5): 1105-1118.

[14] 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]//2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2014: 2997-3004.

[15] Strub M P, Gammell J D. Advanced BIT*(ABIT*): Sampling-based planning with advanced graph-search techniques[C]//2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020: 130-136.

[16] Strub M P, Gammell J D. Adaptively Informed Trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics[C]//2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020: 3191-3198.