机器人控制系统课程报告

扫地机器人路径规划算法设计

姓　　名：　刘瑾瑾

学　　号：　20201000128

班　　级：　231202

培 养 单 位：　自动化学院

二〇二三年六月

目录

第一章 引言

1.1 背景与意义

机器人是一种自动执行工作的装置，它既可以执行预先编排的程序，又可以接受人类指挥，或者按照人工智能技术制定的原则工作，它可以帮助或者替代人类进行工作。随着现代科学技术的发展，机器人与人类的关系变得越来越密切,可以说是深入到了人类生活、工作的方方面面。国际机器人联合会及其他国际标准化组织通常把机器人分为工业机器人和服务机器人两类,其中工业机器人指应用于工业领域的多自由度机器人或多关节机械手，而服务机器人则是除工业机器人之外、用于非制造业并服务于人类的各种先进机器人，包括:家用服务机器人和专业服务机器人，后者包括水下机器人、娱乐机器人、军用机器人、农业机器人等。

家用服务型机器人发展较快，有独立成体系的趋势，一方面，其综合了多种学科的理论基础及最前沿的科学技术,是一个非常值得深入研究和探讨的热门方向；另一方面，服务机器人能够替代人类完成繁杂而枯燥的工作，随着人们逐渐提高的生活水平和逐渐加快的生活节奏,家用服务型机器人的需求量变得越来越大。与工业机器人相比，家用服务机器人并不是工作于提前设定好的区域，也没有提前设定好的固定路线。所以，必须拥有探测周围环境和自我定位的能力,能否依据这些能力和获取的信息做到“随机应变”是衡量一个服务机器人是否智能的关键。

本文着重研究家庭服务机器人中的扫地机器人，即实现了自主化、智能化的吸尘器,扫地机器人的任务是在不依赖人工指导的情况下完成房间的全覆盖清扫。路径规划是移动机器人自主导航中最重要的问题。通过检测和避开障碍物得到从初始位置到目标位置的安全路径是任何导航技术中最重要的功能。一个好的路径规划算法能够有效减少移动机器人完成任务的时间，提高工作效率，能够达到节约资源与减少资金成本的目的。由于单一的路径规划算法只能实现扫地机器人在某一种工作环境中的路径规划需求。在大多数情况下，考虑到服务机器人工作环境的复杂性，既包含已知的静态信息又包含未知的动态信息。因此，使用多种路径规划算法相融合形成混合路径规划方法，从而提高扫地机

器人路径规划在复杂环境中的质量和效率。

1.2 设计的内容介绍

扫地机器人的根本任务是自主完成对整个房间的全覆盖清扫，而障碍物的躲避是完成任务的前提条件，在此基础上还需进行点对点的路径规划。本文主要对扫地机器人的路径规划问题进行研究，分别为以下三个问题：

（1）扫地机器人如何从完成清扫区域的终点移动到下一区域起点？

（2）扫地机器人如何躲避房间中存在的障碍物？

（3）扫地机器人如何实现对房间整体区域的全覆盖清扫？

上述三个问题分别对应于点对点路径规划算法、避障算法和全覆盖路径规划算法。假定实际房间布局与地图信息一致，并且在扫地机器人工作期间不会新增障碍物。本文先从扫地机器人的模型建立出发，明确扫地机器人的控制原理，然后设计点对点路径规划算法、避障算法和全覆盖路径规划算法，多种算法结合整体实现扫地机器人的路径规划。

其中，点对点路径规划算法选择基于采样的快速拓展随机树算法（Rapidly-exploring Random Trees,RRT）,该类算法不但适用于不确定环境的高维空间，而且能够很好的解决如人工势场法的局部极小问题。与其他避障方法相比，人工势场法基于简单的力学原理，将机器或机器人看作一个受力体，根据环境中的势场分布计算出合适的移动方向，能够实时地进行路径规划和避障，对于实时应用具有较高的效率，不需要建立复杂的地图或进行全局路径规划，可以根据当前环境的变化快速做出响应，故本文选用人工势场法进行避障规划，并通过改进优化其不可达和局部最优问题。对于扫地机器人来说，它的主要功能是实现室内的清扫，本文采用全覆盖的内螺旋路径规划，完成对室内的清扫，在存在障碍物的情况下，效果依然不错。

第二章 扫地机器人模型及路径规划算法

2.1 扫地机器人的运动模型

无论是路径规划还是避障，都需要控制机器人进行相应的运动姿态改变，所以运动控制是整个机器人的基础，下面对扫地机器人的运动模型进行介绍。

扫地机器人大部分采用圆饼形的设计，这样设计的优点是机器人在进行旋转动作时，不会像别的形状的机器人碰撞到家具等物品，而且当机器人撞到障碍物时，圆形的设计可以最大的缓解碰撞力来保护家具。扫地机器人的模型如图 2-1 所示，采用两个驱动轮和一个万向轮，两个驱动轮可以主动控制转速和方向，万向轮可以朝任意方向转动，起到辅助运动的作用。C 为扫地机器人的中心，D 表示扫地机器人的直径。

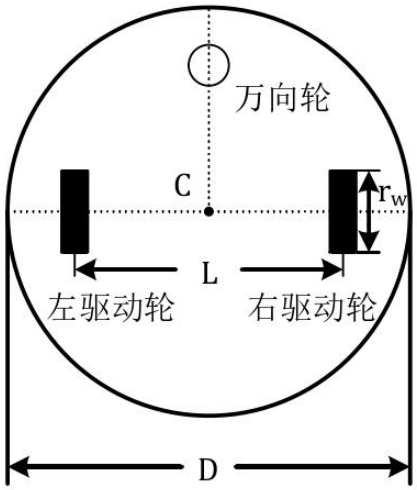


图 2- 1 扫地机器人模型

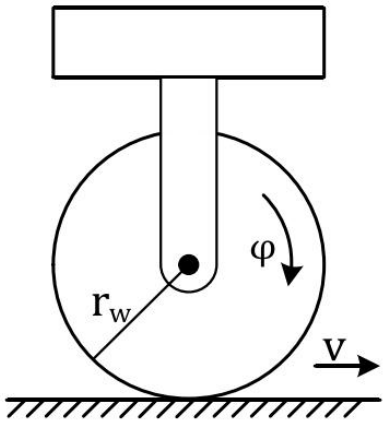


图 2- 2 驱动轮模型

扫地机器人的驱动轮示意图如图 2-2 所示，驱动轮的半径为 r_w，转动的角速度为φ，则驱动轮的速度 V 可以表示为:

$$V = r_w\varphi \tag{2.1}$$

若扫地机器人左右驱动轮的角速度为φ_L和φ_R，则左右驱动轮的速度分别为 $V_L = r_w\varphi_L$ 和 $V_R = r_w\varphi_R$。扫地机器人的运动模型如图所示，通过调节两个轮子的速度控制扫地机器人前进，后退，转弯等动作。

由于扫地机器人移动速度比较小，忽略横滑影响，则扫地机器人中心的速度

Vc 可以表示为:

$$V_C = \frac{V_R + V_L}{2} \tag{2.2}$$

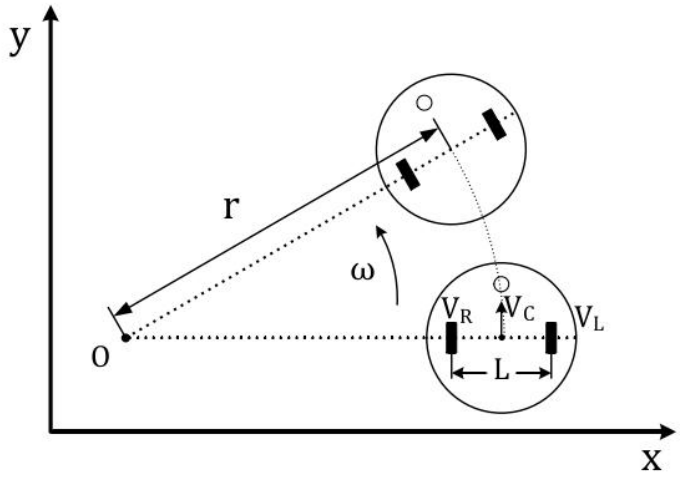


图 2- 3 扫地机器人运动模型

扫地机器人做半径为 r 的圆弧运动，O 为扫地机器人的瞬时转动中心，由刚体力学知识可知 O 与两个驱动轮的中心在一条直线上。L 为左右两轮的距离，左右两驱动轮关于 O 点运动的角速度相等，可以得到:

$$\frac{V_L}{r - \frac{L}{2}} = \frac{V_R}{r + \frac{L}{2}} \tag{2.3}$$

经过计算得到扫地机器人做圆弧运动的半径 r 为:

$$r = \frac{L(V_R + V_L)}{2(V_R - V_L)} \tag{2.4}$$

圆弧运动的角速度为:

$$\omega = \frac{V_L}{r - L/2} = \frac{V_R - V_L}{L} \tag{2.5}$$

所以根据不同的 V_L 和 V_R 可以控制机器人完成三种方式的运动：当 $V_L=V_R$ 时，r 无穷大，机器人沿直线运动；当 $V_L+V_R=0$ 时，r=0，机器人原地以角速度。旋转一定角度;当 $V_L \neq V_R$ 时，机器人做半径为 r 的圆弧运动，若 $V_L>V_R$ 机器人顺

时针运动，若 $V_L<V_R$ 机器人逆时针运动。。

表 2- 1 左右轮的速度与机器人运动方式的关系

运动方式	左右轮的速度
直线前进	$V_{L=}V_R>0$
直线后退	$V_{L=}V_R<0$
原地左转	$V_{L=-}V_R\ V_R>0$
原地右转	$V_{L=-}V_R\ V_L>0$
顺时针前进	$V_L\neq V_R\ V_L>V_R$
逆时针前进	$V_L\neq V_R\ V_L<V_R$

通过调节左右驱动轮电机的转速可以控制左右轮的速度，进而可以控制扫地机器人的运动方式，为机器人的避障控制打好基础。

2.2 近似单元分解——栅格法

栅格表示法是一种常用的环境建模方法，将环境划分为一个个规则的方格单元（即栅格）。栅格表示法可以通过近似单元分解的方式来简化环境的建模，以减少计算复杂度和内存开销。

在近似单元分解中，栅格表示法可以将环境中的复杂几何形状、障碍物和地形等信息近似地表示为一系列规则的栅格单元。每个栅格单元可以表示为一个状态，表示该单元的特性。通过将环境划分为栅格单元，可以将复杂的环境信息转化为一个二维数组或矩阵，便于算法的处理和运算。

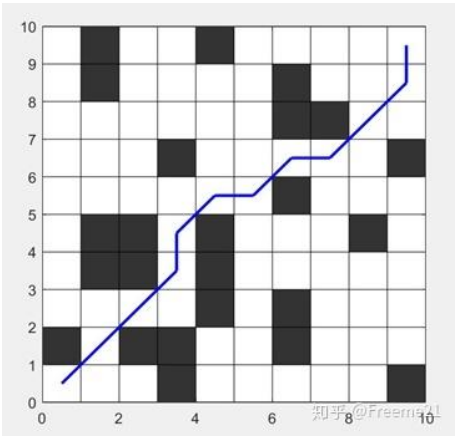


图 2- 4 栅格法示意图

先设定一定程度大小的空间作为栅格大小。假定障碍物在规划过程中是不变的，用黑色的部分表示(只要有障碍物及标识成黑色)。其他的部分用白色表示，表示可以自由航行。起点用 G 表示。终点用 S 表示。

近似单元分解的优势如下：（1）计算效率提高：栅格表示法将环境分解为规则的栅格单元，减少了计算复杂度。在栅格表示的基础上，可以使用基于栅格的算法进行路径规划和避障等任务，这些算法可以针对栅格单元之间的关系进行高效计算；（2）内存占用减少：栅格表示法可以将环境的复杂信息压缩为一个二维数组或矩阵，从而减少了存储环境信息所需的内存空间。这对于内存资源有限的嵌入式系统或移动机器人等应用非常有益。

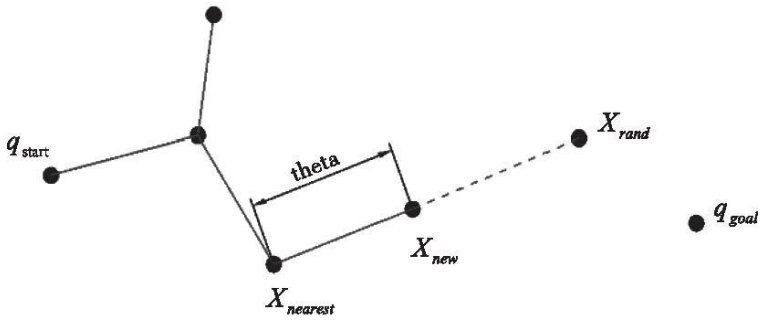
然而，近似单元分解也有一些限制：（1）精度损失：栅格表示法是一种离散化的方法，对环境的连续性和精细的几何形状可能存在一定的近似误差。这意味着，在某些情况下，栅格表示法可能无法准确地捕捉到环境的细节和真实的地形信息；（2）空间分辨率限制：栅格表示法的精度和空间分辨率受栅格大小的限制。较大的栅格可以减少计算复杂度，但可能导致精度损失和信息丢失。而较小的栅格可以提高精度，但会增加计算复杂度和内存开销。因此，在选择栅格大小时需要权衡计算效率和表示精度之间的平衡。

2.3 点对点路径规划——RRT 算法

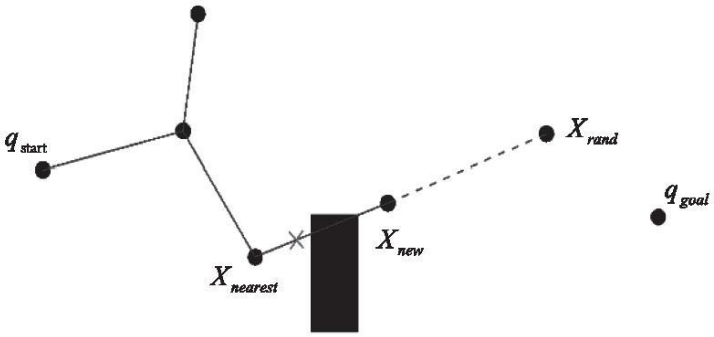
RRT（Rapidly-exploring Random Tree 快速扩展随机树）是一种常用的基于采样的路径规划方法，用于在连续状态空间中寻找路径。它以一个初始点作为根节点，通过随机采样增加叶子节点的方式，生成一个随机扩展树，当随机树中的叶子节点包含了目标点或进入了目标区域，便可以在随机树中找到一条由从初始点到目标点的路径。RRT 算法适用于机器人运动规划、自主导航和避障等领域。

RRT 原理如图 2-5 所示。在工作环境中定义路径规划的起点 q_{start} 与终点 q_{goal} 算法初始化根结点，以根结点开始生长，向四周进行探索，在地图中随机采样，采样点为 x_{rand}.生成公式如式(2)所示，其中 X,Y 为地图的边界尺寸。然后找到树中离 x_{rand} 最近的叶子结点 $x_{nearest}$.接下来以 theta 为步长，往 x_{rand} 方向进行扩展，扩展新的叶子结点为 x_{new}.公式如式(3)所示。对 $x_{nearest}$ 与 x_{new} 的连线进行碰撞检测，若连线不经过障碍物，则将 x_{new} 加入树中，进行下一轮的迭代，如图 2-5(a)所示;

若连线经过障碍物，则放弃本次迭代，重新选取随机点，如图 2-5(b)所示。



（a）随机扩展



（b）碰撞检测

图 2- 5 RRT 拓展过程

$$x_{rand} = (rand(0, X), rand(0, Y)) \tag{2.6}$$

$$x_{new} = x_{nearest} + theta \frac{(x_{rand} - x_{nearest})}{\|x_{rand} - x_{nearest}\|} \tag{2.7}$$

当树中的任意叶子结点与 q_{goal} 重合或 $x_{nearest}$ 与 x_{new} 的连线经过 q_{goal} 时完成搜索，随后算法在搜索树中寻找一条连接起点和终点的最短路径;若达到最大迭代次数也没有找到目标点，则搜索失败。

RRT 算法的基本步骤：

（1）初始化：将起始点作为树的根节点。构建一个包含起始点的树结构。

（2）采样：随机采样一个点（目标点）在整个空间中。

（3）最近邻点搜索：从树中找到距离采样点最近的节点作为最近邻点。

（4）扩展：从最近邻点向采样点延伸一步，生成一个新的节点。

（5）碰撞检测：检测新节点与障碍物是否发生碰撞。

（6）连接与添加：如果新节点与障碍物无碰撞，将其与最近邻点之间的路径添加到树中，并将新节点作为树的一个子节点。

（7）目标检测：检查新节点是否接近目标点。

（8）终止条件：如果满足目标条件，则路径生成完成；否则返回第 2 步。

通过重复执行上述步骤，RRT 算法会逐渐扩展树，直到找到一条可行路径连接起始点和目标点。由于随机采样和快速扩展的特性，RRT 算法能够在高维空间中快速探索，并且可以适应复杂的环境。

2.4 避障规划——人工势场法

2.4.1 传统人工势场

1986 年 Khatib 首先提出人工势场法，并将其应用在机器人避障领域。该方法的基本思想是在 d 障碍物周围构建障碍物斥力势场，在目标点周围构建引力势场，类似于物理学中的电磁场，如图 3.1 所示。被控对象在这两种势场组成的复合场中受到斥力作用和引力作用，斥力和引力的合力指引着被控对象的运动，搜索无碰的避障路径。

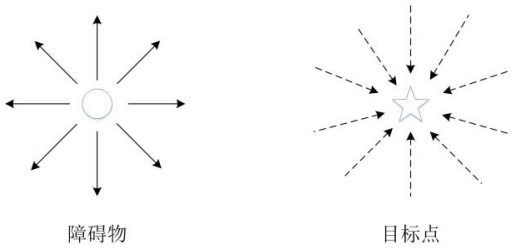


图 2-6 势场示意图

（1）引力势场

引力势场主要与机器人和目标点间的距离有关，距离越大，机器人所受的势能值就越大；距离越小，机器人所受的势能值则越小，所以引力势场的函数为：

$$U_{att}(q)=\frac{1}{2}\eta\rho^2(q,q_g) \tag{2.8}$$

其中η为正比例增益系数，$\rho(q,q_g)$为一个矢量，表示汽车的位置 q 和目标 点位置 q_g 之间的欧几里德距离 $|q-q_g|$，矢量方向是从汽车的位置指向目标点位置。

相应的引力 $F_{att}(X)$为引力场的负梯度：

$$F_{att}(X) = -\nabla U_{att}(X) = \eta\rho(q, q_g) \tag{2.9}$$

引力的方向指向目标点。

（2）斥力势场

决定障碍物斥力势场的因素是机器人与障碍物间的距离，当机器人未进入障碍物的影响范围时，其受到的势能值为零；在机器人进入障碍物的影响范围后，两者之间的距离越大，机器人受到的势能值就越小，距离越小，机器人受到的势能值就越大。

斥力势场的势场函数为：

$$U_{req}(X) = \begin{cases} \frac{1}{2}k(\frac{1}{\rho(q,q_0)} - \frac{1}{\rho_0})^2, 0 \le \rho(q,q_0) \le \rho_0 \\ 0 \qquad\qquad\qquad , \rho(q,q_0) \ge \rho_0 \end{cases} \tag{2.10}$$

其中 k 为正比例系数， $\rho(q, q_0)$ 为一矢量，方向为从障碍物指向机器人，大小为机器人与障碍物间的距离 $|q - q_0|$ ， ρ_0 为一常数，表示障碍物对机器人产生作用的最大距离。

相应的斥力为斥力场的负梯度，即：

$$F_{req}(X) = \begin{cases} k(\frac{1}{\rho(q,q_0)} - \frac{1}{\rho_0})\frac{1}{\rho^2(q,q_0)}\nabla\rho(q,q_0), 0 \le \rho(q,q_0) \le \rho_0 \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad , \rho(q,q_0) \ge \rho_0 \end{cases} \tag{2.11}$$

（3）合力势场

根据人工势场法原理，机器人受到由上两节建立的引力势场和障碍物斥力势场组成的复合场的作用，在机器人前往目标点的过程中很有可能同时受到多个障碍物的斥力场作用，即机器人所受到的斥力场作用是叠加的，机器人受到的势能值为一个引力场和多个障碍物斥力场的共同作用：

$$U(X) = U_{att}(X) + \sum_{i=0}^{m} U_{req}(X) \tag{2.12}$$

式中，m 为对汽车起作用的障碍物的个数。

汽车所受到的合力为：

$$F(X) = -\nabla U(X) = F_{att}(X) + F_{req}(X) \tag{2.13}$$

（4）传统人工势场法存在的问题

传统人工势场法存在目标不可达和局部最优的问题

① 目标不可达的原因

由于障碍物与目标点距离太近，当机器人到达目标点时，根据势场函数可知，目标点的引力降为零，而障碍物的斥力不为零，此时机器人虽到达目标点，但在斥力场的作用下不能停下来，从而导致目标不可达的问题。

② 局部最优的原因

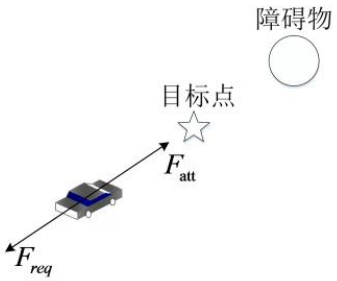


图 2- 7 局部最优原因之一

第一种就是图 2-7 所示，此时机器人受到的障碍物的斥力和目标点的引力之间的夹角很大，几乎在同一条直线上，就会出现图 b 中机器人在障碍物前陷入局部最优的问题。

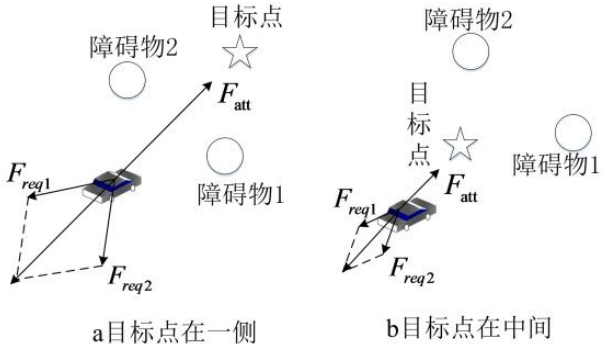


图 2- 8 局部最优原因之二

第二种就是如图 2-8 中两侧斥力与引力相等的情况，图 a 中，障碍物和机器人位于目标点的同一侧，此时，如果多个斥力场的合力与目标点的引力在同一条直线上，且大小相等、方向相反的话，就会陷入局部最优的问题，即 $F_{att} \leq$

$F_{req1}+F_{req2}$；在图 b 中，障碍物和机器人位于目标点的两侧，此时，也有可能出现多个斥力场的合力与目标点的引力相互抵消的情况。

③ 与障碍物碰撞的原因

在传统人工势场法中未考虑到障碍物的速度，所以，在算法根据采集到的信息进行路径规划后，障碍物也是以一定的速度向前运动了一段距离，这就有可能导致机器人与障碍物发生碰撞。

2.4.2 改进的人工势场函数

在传统人工势场法中，当机器人到达目标点时，引力为零而斥力不为零，就会导致目标不可达的问题；当所有障碍物斥力的合力方向与引力的方向相反，而此时机器人又未到达目标点，就有可能出现斥力和引力相等而陷入局部最优，为此在传统人工势场法的障碍物斥力场模型中加入调节因子ρ_g^n，使机器人只有到达目标点时，斥力和引力才同时减小到零，从而使局部最优和目标不可达的问题得到解决。

改进后的斥力场函数为：

$$U_{req}(X)=\begin{cases}\frac{1}{2}k(\frac{1}{\rho(q,q_0)}-\frac{1}{\rho_0})^2\rho_g^n, 0\le\rho(q,q_0)\le\rho_0\\ 0 \qquad\qquad\qquad ,\rho(q,q_0)\ge\rho_0\end{cases} \tag{2.14}$$

$\rho(q,q_0)$ 为机器人与障碍物间的距离，ρ_0 为障碍物的影响距离，ρ_g^n为机器人与目标点的距离，式中 n 为任意常数，经多次仿真实验得 n=2。

与传统人工势场法相比，改进的斥力场函数中加入了机器人与目标点间的距离，这样使机器人在驶向目标的过程中，受到的引力和斥力同时在一定程度上减小,且只有在机器人到达目标点时，引力和斥力才同时减小为零，即目标点成为势能值的最小点，从而使局部最优原因一和目标不可达的问题得到解决。当机器人未到达目标点时，相应的斥力为：

$$F_{reo}(X)=\begin{cases}(F_{reo1}+F_{reo2})\nabla\rho(q,q_0), 0\le\rho(q,q_0)\le\rho_0\\ 0 \qquad\qquad\qquad\qquad ,\rho(q,q_0)\ge\rho_0\end{cases} \tag{2.15}$$

$$\begin{cases}F_{reo1}=k(\frac{1}{\rho(q,q_0)}-\frac{1}{\rho_0})\frac{\rho_g^n}{\rho^2(q,q_0)}\\ F_{reo2}=\frac{n}{2}k(\frac{1}{\rho(q,q_0)}-\frac{1}{\rho_0})^2\rho_g^{n-1}\end{cases} \tag{2.16}$$

其中 F_{reo1} 的方向为从障碍物指向汽车，F_{reo2} 的方向为汽车指向目标点，如图 2-9 所示：

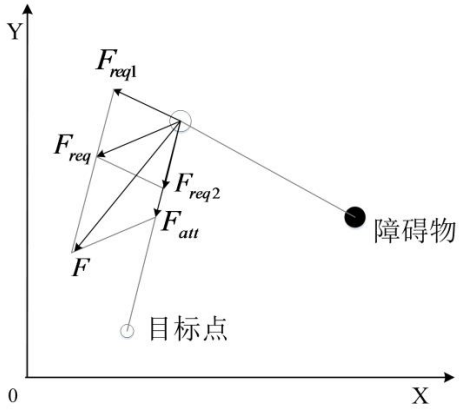


图 2- 9 机器人在改进后的势场中的受力

2.5 全覆盖路径规划——螺旋式路径规划

为了使扫地机器人能够完全遍历整个区域，基于单元分解法的全覆盖路径规划方法首先对整个区域进行子区域分解，然后解决子区域内的全覆盖路径规划方法，最后将所有的子区域有序的连接起来，从而得到一条全覆盖路径。

单元分解法分解的子区域不含有独立的障碍物，所以子区域的完全遍历方法比较简单，常用的方法为往复式路径和内螺旋路径。往复式路径的起点在子区域的一角，机器人在区域内做往复运动，终点在区域对角线或与起点在一条边上。内螺旋路径的起点也是子区域的一角，机器人沿区域的边界按顺时针或逆时针的方向不断向内做螺旋运动，直到覆盖整个子区域。

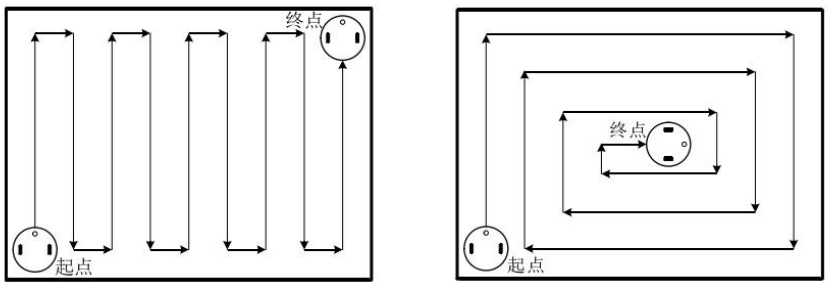


图 2- 10 子区域覆盖方法

螺旋式路径规划可以更高效地覆盖目标区域。它从中心点开始，以较小的半径和较大的旋转角度覆盖区域，然后逐渐增加半径并减小旋转角度，可以更快速地覆盖整个区域，减少覆盖路径的冗余。

相比于往复式路径规划，螺旋式路径规划通常具有较短的路径长度。通过逐渐增加半径和减小旋转角度，内螺旋式路径规划可以有效地减少路径的总长度，从而节省时间和资源。

螺旋式路径规划相对于往复式路径规划更简单且易于实现。它的规划逻辑较为直观，容易理解和实现，不需要复杂的环境感知和路径优化算法。螺旋式路径规划对于不规则的或具有障碍物的环境也比较适应。它可以自动调整半径和旋转角度，以适应环境中的障碍物，同时保持较高的覆盖效率。

由于本文只是对扫地机器人全覆盖路径规划的简单模拟，故选择螺旋式路径规划。

第三章 设计内容的 Matlab 程序仿真

3.1 基于栅格法的环境建模

为了能够精确表示地图信息、障碍物信息以及定位信息，需要将扫地机器人所处环境的高精度地图作为算法的基础。全局路径规划可以使用栅格地图，二维栅格地图使用占用栅格的形式存储环境中的静态障碍物信息，为全局路径规划提供信息。本文使用已知的二维栅格地图作为地图信息输入，在全局路径规划算法开始时初始化地图信息。

3.1.1 位形空间

对障碍物进行膨化处理，将障碍物全部膨胀略大于其半径的尺寸，得到扫地机器人的位形空间。在位形空间中，扫地机器人成为一个可移动的点，无需考虑姿态、体积和非完整运动学约束。膨胀后即可进行栅格化处理。

3.1.2 栅格尺寸选取

栅格的大小直接决定了地图在离散化过程中的误差大小，同时也决定了进行路径规划时计算的复杂程度以及对存储空间的要求高低。当栅格的尺寸比较小时，地图的误差就会减小，但所占的内存也会相应的增加，这就会导致搜索速度变慢；当栅格的尺寸比较大时，地图的误差就会增大，但搜索速度会的提升。综合分析，确定栅格地图的行列数：M=W/D，N=L/D。栅格尺寸与数量的合理选取也使对精度的影响降到最低。

3.1.3 栅格的标识

常用的有直角坐标系法和序号法。这里选择使用直角坐标系法，对处在工作环境中的左下角 W 设定为坐标原点，X 代表是水平位置，同时定义为向右为正，呈现越向右值越大的趋势。Y 类似，向上为正方向，同时随着向上方向的延伸值越大，每一个栅格都可以通过固定的 X 与 Y 来进行确定。所以直角坐标（X，Y）可以用来表示任意栅格，其坐标即为栅格的左下角坐标。

3.1.4 栅格的信息编码

由于栅格与数组存在对应的联系，因而矩阵进行行列分割能够实现栅格的存储。数组矩阵之中，栅格的坐标是结构体的索引，任何的栅格的信息都能够通过数组进行表示，其中结构体与地图中栅格进行对应。栅格的坐标值在直角坐标系中被 X 和 Y 表示，障碍物的选择标记被赋予 0 与 1 两个值，如果是 0，则意味该栅格为自由栅格，可以通行；如果是 1，则意味该栅格为障碍栅格，不能通行。

3.2 基于采样的 RRT 算法

3.2.1 MATLAB 程序的流程

（1）初始化参数：包括停止搜索的距离阈值（threshold）、RRT 树的最大节点数（maxNodes）、父节点到下一个节点的步长（neighborhood）、障碍物信息（obstacle）、机器人一次可前进的步长（step_size）等。

（2）定义 RRT 树的节点结构（rrt struct）：每个节点包含坐标（p）、父节点索引（iPrev）、累计代价（cost）和是否到达目标点的标志（goalReached）。

（3）开始路径规划功能函数：创建 RRT 树，并进行迭代搜索。

（4）迭代搜索过程：在每次循环中，执行以下步骤：

a. 随机采样一个点。

b. 找到现有节点中距离采样点最近的节点。

c. 沿着最近节点到采样点的方向，按照步长前进，得到新的节点。

d. 对新节点进行障碍物判断和父节点选择。

e. 将新节点插入到 RRT 树中，并进行路径优化。

f. 判断新节点是否满足到达目标点的条件（norm(new_node-p_goal) == param.threshold），如果满足，则标记该节点为到达目标点。

g. 更新循环计数器。

（5）根据得到的 RRT 树，计算最终路径并绘制：

a. 绘制 RRT 树的节点之间的连线。

b. 找到到达目标点的节点中代价最小的节点，从该节点通过回溯找到起始节点，形成最短路径。

c. 绘制最短路径。

d. 返回路径的代价和运行时间。

具体代码见附录，RRT 的伪代码如下：

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

3.2.2 MATLAB 仿真结果及分析

（1）地图中无障碍物：

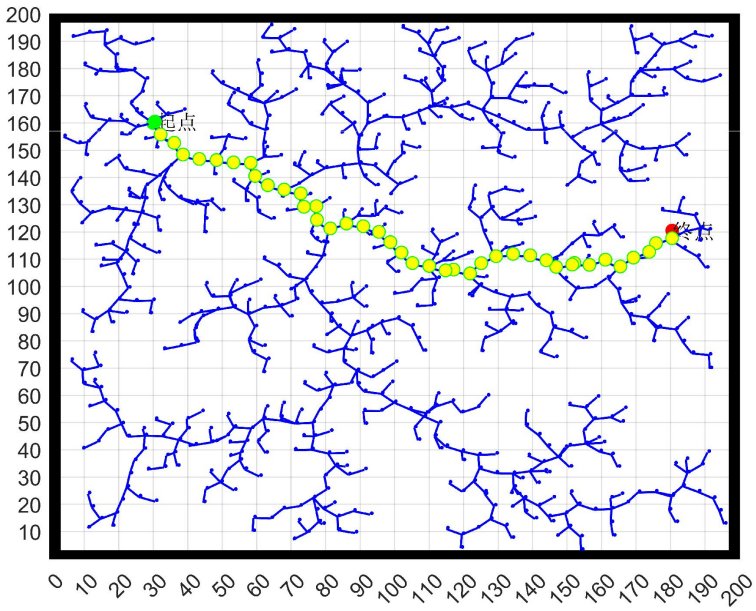


图 3- 1 无障碍物的 RRT 路径规划结果

由图可知，RRT 算法并不是最优路径规划算法，而是一种概率完备的、用于搜索可行路径的算法。理论上，从起点（绿色）到终点（红色）的最优路径应该为直线，但通过 RRT 得到的路线是曲折的线，具有一定的随机性。

（2）地图中有障碍物：

在栅格地图中，使用黑色矩形表示障碍物，从起点到终点的路径规划结果如图 3-2，RRT 算法找到了可行路径。

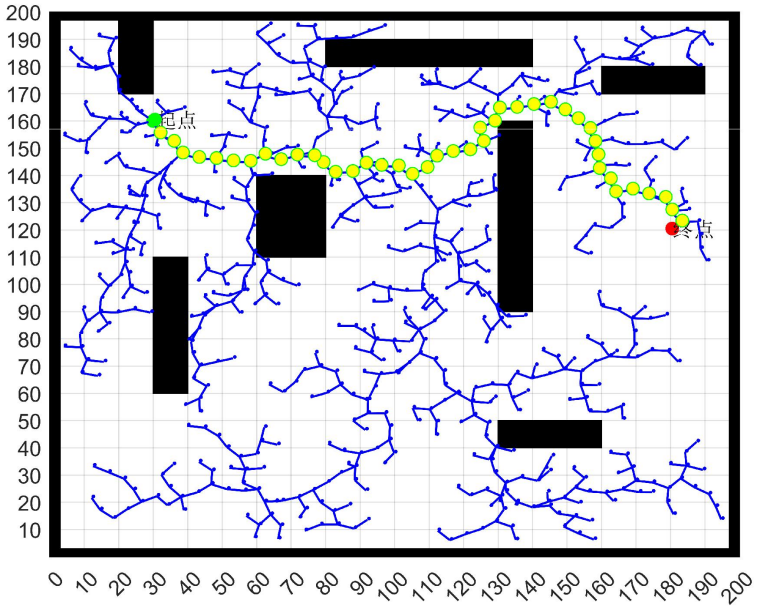


图 3- 2 有障碍物的 RRT 路径规划结果

3.3 改进的人工势场法

3.3.1 MATLAB 程序的流程

斥力场和引力场的建立，机器人当前受到的合力作用。而该程序的实现逻辑大致为：以起点为圆心 R 为半径画圆，并取得圆上八个均布的点，分别计算各个点前进的代价（引力与斥力之和），并选取代价最小的点作为下一个起点。程序的流程如下：

（1）起点、终点 、障碍物、迭代次数、取点半径等参数的设定

（2）以起点为中心，作半径为 r 的圆，从圆上取八个均布的点

（3）分别计算八个点的前进“代价”—— 终点对其的引力和所有障碍物对其的斥力之和。传统人工势场法合力计算参考式 2.13，改进人工势场法合力计算参考式 2.15 和 2.16。

（4）取“代价”最小的点的坐标，结合现有起点，计算得到新的起点，然后重复上述内容。

（5）当发现一个点距离终点很近或者迭代的次数计算完程序停止。

3.3.2 MATLAB 仿真结果及分析

起点使用绿色表示，终点使用红色表示，障碍物使用红色的椭圆形表示，障碍物大小相同，仿真结果如下：

（1）目标不可达问题

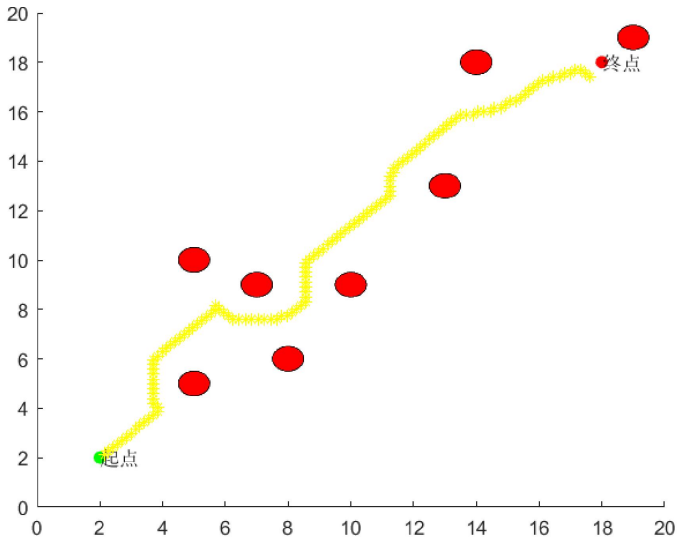


图 3- 3 传统人工势场法生成避障路径

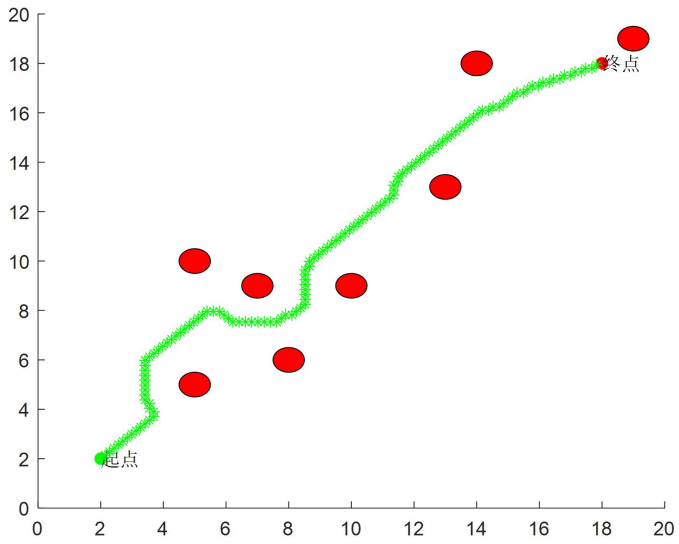


图 3- 4 改进人工势场法生成避障路径

在目标点后存在障碍物时，由于机器人越靠近目标点引力越接近于 0，而此时斥力还比较大，故传统人工势场法出现不可达的情况，而改进的人工势场法由于距离修正因子的作用，接近目标点时引力和斥力都为零，故可以到达终点。

（2）局部最优问题

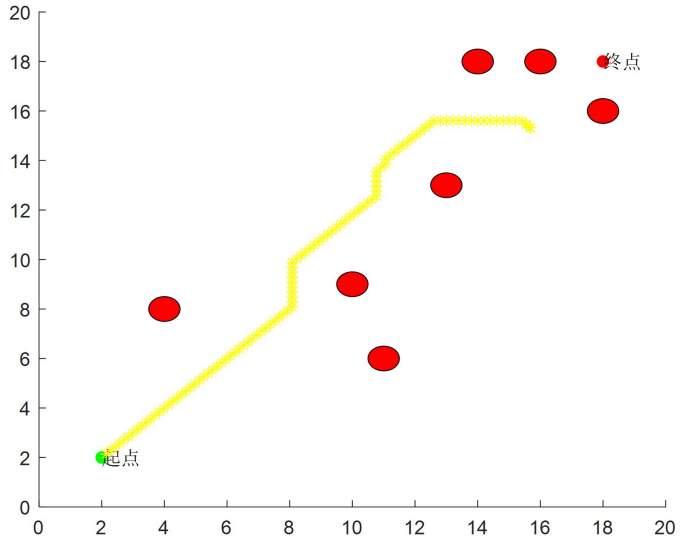


图 3- 5 传统人工势场法路径规划

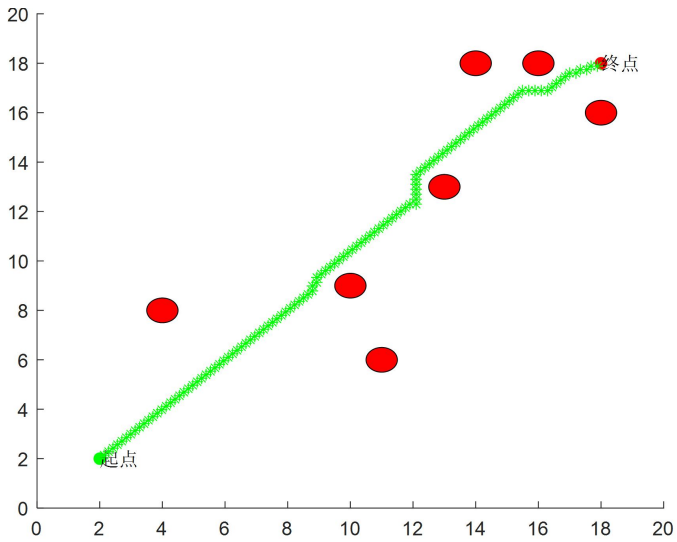


图 3- 6 改进人工势场法路径规划结果

传统人工势场法存在陷入局部最小的问题，如图 3-5，此时引力和斥力相等，使得扫地机器人卡住，不能移动，而引入距离因子和修正人工势场法使斥力在一定程度上减小，故可以到达目标点。但是，引入距离修正因子只能部分解决

局部最优问题，且距离修正因子的参数需要调试。

3.4 全覆盖螺旋式路径规划

3.4.1 MATLAB 程序的流程

（1）绘制初始地图：

a.创建一个 22x20 的全零矩阵 map 表示地图。

b.定义障碍物的坐标 a，其中 a 是一个包含多个二维数组的单元格数组，每个二 c.维数组表示一个障碍物的坐标。

d.将障碍物的坐标在 map 上标记为 1，即将相应位置的元素设为 1。

（2）生成起点和终点：

a.定义起点坐标 q_start 和终点坐标 q_goal。

b.在绘制的地图上，用绿色圆点表示起点，用红色圆点表示终点。

（3）路径规划：

a.初始化变量和绘图设置。

b.在每个循环迭代中，尝试向右、向上、向左、向下移动，检查移动后的位置是否可行。

c.如果可行，更新当前位置并在绘图中标记移动的路径。

d.根据移动方向的优先级，判断下一步应该移动的方向。

e.重复上述步骤，直到无法再移动到新位置为止。

（4）绘制路径：

a.在绘制的地图上，用蓝色实心圆点表示规划得到的路径点。

b.在绘制的地图上，用蓝色实线连接路径上的相邻点，形成路径线。

（5）显示地图和路径：

a.设置图形的显示范围和网格线。

b.绘制地图、起点、终点和路径。

c.在绘制的图像中添加网格线并显示。

3.4.2 MATLAB 仿真结果及分析

（1）无障碍物全覆盖内螺旋路径规划：

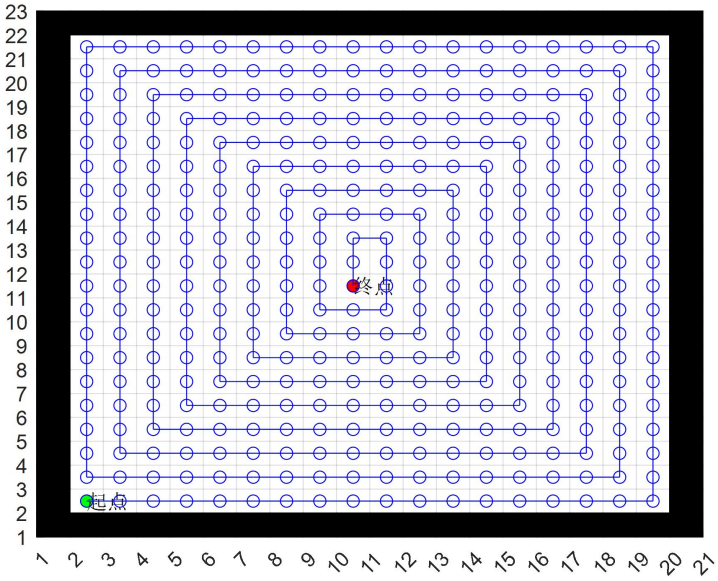


图 3-7 无障碍物全覆盖内螺旋路径规划

（2）有障碍物全覆盖内螺旋路径规划：

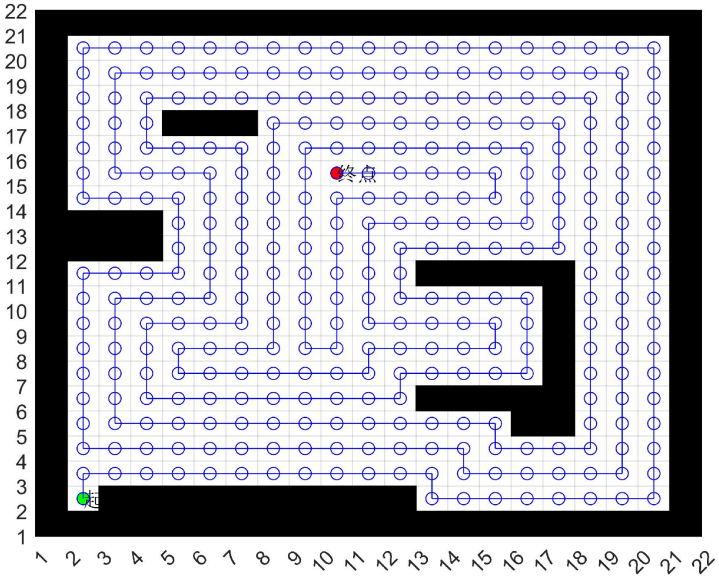


图 3-8 有障碍物全覆盖内螺旋路径规划

不论对于有障碍物的室内环境，还是无障碍物的室内环境，内螺旋算法都可是实现室内环境的覆盖，达到良好的效果，但也可能存在重复或者覆盖率低等情况。另外，本文考虑的算法比较简单，对于复杂的室内环境或者移动障碍物没有进行深入研究，可能需要使用智能算法或其他算法进一步优化。

第四章 结论

本文从扫地机器人路径规划的三个问题出发，通过 MATLAB 简略模拟实际室内环境，完成对点对点路径规划算法、避障算法和全覆盖路径规划算法的仿真。其中，点对点路径规划算法选择 RRT 算法，避障算法选择人工势场法，全覆盖算法选择内螺旋路径规划。通过实际仿真，我对这三种算法的原理有了更深刻的理解，对于《机器人控制系统》课程上学习到的专业知识有了进一步的理解和认识。

此外，经过课堂作业的 MATLAB 仿真和课程实验的操作，我对机器人的运动学和动力学、轨迹规划等基本理解和掌握，对于 MATLAB 的 Robotics Toolbox 可以熟练使用。

当然，本次课程设计也存在不足，RRT 算法只能找到可行路径，却不一定能够找到最优路径，最优路径可以使用 A*算法、Dijkstra 或者智能算法寻找；虽然通过添加距离修正因子优化了人工势场法目标不可达和局部最小化问题，但是局部最小化问题并没有得到根本解决；全覆盖内螺旋算法实现的比较简单，会存在覆盖率不足或者重复等问题。并且，本次课程设计扫地机器人只实现了路径规划，并没有实现整体的设计，还存在很多不足，还有许多问题需要解决。

最后，感谢郑老师的辛苦付出，在机器人课程上我学习到了很多东西，也激发了我对于机器人的兴趣，相信以后的学习和工作中一定会运用所学习到的专业知识。

参考文献

[1]姚相滢,许伦辉,林世城.基于改进 RRT 算法的智能车辆路径规划研究[J].计算机仿真,2023,40(04):165-169+525.

[2]朱伟达.基于改进型人工势场法的车辆避障路径规划研究[D].江苏大学,2017.

[3]张驰,于丹. 多移动机器人全覆盖路径规划研究[C]//中国自动化学会.2022 中国自动化大会论文集.2022:291-296.DOI:10.26914/c.cnkihy.2022.053837.

[4]张岩. 基于改进蚁群和人工势场法的服务机器人路径规划研究[D].天津工业大学,2020.DOI:10.27357/d.cnki.gtgyu.2020.000180.

[5]钱金伟,戴晓强,高宏博,朱延栓.基于内螺旋搜索的生物激励遍历路径规划算法.计算机仿真,2021,38(09):339-343+394.

[6]荀燕琴,田竹梅,任国凤,鹿嘉航.基于遗传算法的智能扫地机器人路径规划研究[7].高师理科学刊,2020,40(03):56-60.

[8]王庆禄,吴冯国,郑成辰,李辉.基于优化人工势场法的无人机航迹规划[J].系统工程与电子技术,2023,45(05):1461-1468.

[9]任云天. 扫地机器人避障全覆盖路径规划算法的研究与设计[D].华中科技大学,2019.

[10]方夏章. 扫地机器人单目视觉障碍物探测及路径规划的设计与实现[D].中国科学技术大学,2018.

[11]肖友伦. 扫地机器人地毯识别与路径规划研究[D].湖南大学,2021.DOI:10.27135/d.cnki.ghudu.2021.000956.

[12]罗正男. 基于 Graft-RRT 的机器人路径规划研究[D].江西理工大学,2022.DOI:10.27176/d.cnki.gnfyc.2022.000556.

附录

1.RRT 算法代码

```
function []=rrt_run

    % 画矩形块：[x,y,a,b]起始点(x,y)
    function[] = rect(x,y,l,b)
      hold on
      rectangle('Position',[x,y,l,b],'FaceColor','k')
    end

    % 画圆的函数
    function circle(x,y,r)
        ang=0:0.01:2*pi;
        xp=r*cos(ang);
        yp=r*sin(ang);
        plot(x+xp,y+yp, '.r');
    end

    % 画图，及一些输入参数的设置
figure;
axis([0,200,0,200])
set(gca, 'XTick', 0:10:200)
set(gca, 'YTick', 10:10:200)
grid ON
hold on
%边框
rect(0,0,3,200);
rect(0,0,200,3);
rect(197,0,3,200);
rect(0,197,200,3);
% 画矩形块充当障碍物
rect(20,170,10,30);
rect(80,180,60,10);
rect(160,170,30,10);
rect(60,110,20,30);
rect(130,90,10,70);
rect(30,60,10,50);
rect(130,40,30,10);
```

```
37.
38.  p_start = [30;160];     % 起点，目标点设定
39.  p_goal = [180;120];
40.
41.  rob.x = 30;       % 机器人所在起点坐标
42.  rob.y = 160;
43.
44.  % 初始化参数
45.  param.obstacles =[20,170,10,30; 80,180,60,10;160,170,30,10;
46.      60,110,20,30;130,90,10,70;30,60,10,50;130,40,30,10;
47.      0,0,3,200;0,0,200,3;197,0,3,200;10,197,200,3];    % 对应矩形块
48.  % param.obstacles =[0,0,3,200;0,0,200,3;197,0,3,200;10,197,200,3];
       % 对应矩形块
49.  param.threshold = 2;
50.  param.maxNodes = 800;
51.  param.step_size = 5;        % 机器人每次行进步数
52.  param.neighbourhood = 5;    % 寻找子节点的距离
53.  param.random_seed = 40;
54.
55.  % plot(rob.x,rob.y,'.r')
56.  plot(rob.x,rob.y,'go','MarkerFaceColor','g');
57.  hold on;
58.  plot(p_goal(1)+.5,p_goal(2)+.5,'ro','MarkerFaceColor','r');
59.  hold on;
60.  text(rob.x+.5,rob.y+.5,'起点');
61.  text(p_goal(1)+.5,p_goal(2)+.5,'终点');
62.
63.  % 进行搜图并出结果
64.  result = PlanPathRRTstar(param, p_start, p_goal)
65.  plot(rob.x+.5,rob.y+.5,'go','MarkerFaceColor','g');
66.  end
67.  function result = PlanPathRRTstar(param, p_start, p_goal)
68.  % rrt 树的四个参数
69.  field1 = 'p';
70.  field2 = 'iPrev';  %  借助该参数，可将整个路径搜索出来。
71.  field3 = 'cost';
72.  field4 = 'goalReached';
73.
74.  rng(param.random_seed);    % 用指定的 randomseed 初始化随机数生成器
```

```
tic;        % tic 开始计时，常与 toc 配合使用，toc 停止计时  （算法运行的时间计时）
start();    % 执行路径规划功能函数

    function start()
        % s = struct(field1,value1,...,fieldN,valueN) 创建一个包含多个字段的结构体数组
        rrt(1) = struct(field1, p_start, field2, 0, field3, 0, field4, 0);
        N = param.maxNodes; % 迭代次数 iterations
        j = 1;

%         while endcondition>param.threshold %&& j<=N
%         每走一次循环，j++，一共循环 N 次
        while j<=N
            % 1）随机采样一个点
            sample_node = getSample();
%             plot(sample_node(1), sample_node(2), '.g');
%             text(sample_node(1), sample_node(2), strcat('random',num2str(j)))
            % 2）找到现有节点中，距离该采样点最近的点
            nearest_node_ind = findNearest(rrt, sample_node);
%             plot(rrt(nearest_node_ind).p(1), rrt(nearest_node_ind).p(2), '.g');
%             text(rrt(nearest_node_ind).p(1), rrt(nearest_node_ind).p(2), strcat('nearest', num2str(j)));
            % 3）沿最近点到采样点方向，按照步长前进，得到新的节点
            new_node = steering(rrt(nearest_node_ind).p, sample_node);
            if (isObstacleFree(new_node)==1)        % 4.1） 新节点的障碍物检测
%                 plot(new_node(1), new_node(2), '.g');
%                 text(new_node(1), new_node(2)+3, strcat('steered: new node', num2str(j)))
                % 获取新节点附近可到达的节点索引  neighbourhood
                neighbors_ind = getNeighbors(rrt, new_node);
                if(~isempty(neighbors_ind))         % 4.2） 判断附近可达到的节点 存在与否 （找该新节点的父节点）
                    % 存在— 从根节点为新子节点选择成本最低的父节点  【难理解 1】
```

```
                    parent_node_ind = chooseParent(rrt, neighbors_ind, nearest_node_ind,new_node);
%                    plot(rrt(parent_node_ind).p(1), rrt(parent_node_ind).p(2), '.g');
%                    text(rrt(parent_node_ind).p(1), rrt(parent_node_ind).p(2)+3, strcat('parent', num2str(j)));
                else
                    % 不存在— 选取距离该采样点最近的点为父节点
                    parent_node_ind = nearest_node_ind;
                end
                % 5）将新节点插入到 rrt 搜索树中
                rrt = insertNode(rrt, parent_node_ind, new_node);
                if (~isempty(neighbors_ind))    % 存在可到达节点时，进行重连操作  优化路径  【难理解 2】
                    rrt = reWire(rrt, neighbors_ind, parent_node_ind, length(rrt));
                end
                % 满足距离阈值条件，记该节点为到达目标 （但依旧会持续嵌套搜索）
                if norm(new_node-p_goal) == param.threshold
                    rrt = setReachGoal(rrt);
                end
            end
            j = j + 1;
        end
        setPath(rrt);   % 上述 while 结束后，绘制寻找的路径
%         text1 = strcat('Total number of generated nodes:', num2str(j-1))
%         text1 = strcat('Total number of nodes in tree:', length(rrt))
    end

%% 一系列功能函数（start 中调用）

    % 在 rrt 树中标记该节点为“到达目标点”（第四个参数）  ok
    function rrt=setReachGoal(rrt)
        rrt(end).goalReached = 1;
    end

    % 绘制出 rrt 中各个节点的关系，并标识出最终得到的路径  ok
```

```
136.     function setPath(rrt)
137.         % 1）绘制 rrt 树 ：各个父节点与子节点之间的连线 【父节点→子节点】
138.         for i = 1: length(rrt)-1
139.             p1 = rrt(i).p;    % 遍历 rrt 中各个节点
140.             rob.x = p1(1); rob.y=p1(2);
141.             plot(rob.x+.5,rob.y+.5,'.b')  % 绘制各个节点
142.             child_ind = find([rrt.iPrev]==i);  % 寻找 rrt 中父节点索引为 i 的一堆子节点
143.             for j = 1: length(child_ind)        % 遍历上述一堆子节点
144.                 p2 = rrt(child_ind(j)).p;       % 找到并绘制 上述父节点到子节点 的连线
145.                 pause(0.01);              % 程序暂停一会再继续运行 -- 体现出路径搜索的过程
146.                 plot([p1(1),p2(1)], [p1(2),p2(2)], 'b', 'LineWidth', 1);
147.             end
148.         end
149.         % 2）找到最短路径，并绘制
150.         [cost,i] = getFinalResult(rrt);
151.         result.cost = cost;
152.         result.rrt = [rrt.p];  % result 输出的节点坐标
153.         % 绘制最终得到的最短路径
154.         while i ~= 0
155.             p11 = rrt(i).p;
156.             plot(p11(1)+.5,p11(2)+.5,'go','MarkerFaceColor','y');
157.             i = rrt(i).iPrev;
158.             if i ~= 0
159.                 p22 = rrt(i).p;  % 依次画出该节点，其父节点，其父节点的父节点（从终点画到起点）
160.                 plot(p22(1)+.5,p22(2)+.5,'go','MarkerFaceColor','y');
161.                 pause(0.02);              % 程序暂停一会再继续运行 -- 体现出路径搜索的过程
162. %                 plot([p11(1),p22(1)],[p11(2),p22(2)], 'b', 'LineWidth', 3);
163.             end
164.         end
165.         result.time_taken = toc;      % 返回程序总的运行时间
166.     end
167.
```

```
168.
169.      % 找最终路径 （在 setPath 中调用）                ok
170.      % 参      数：rrt 树
171.      % 返 回 值：代价 cost 最接近终点的最优节点索引
172.      % 寻找方法：寻找“到达目标点”的节点中，代价最小的节点
173.      % 由该代价最小的节点，rrt 树中的第二个参数 iPrev，可再找到其父节点，
    由此可得到一条代价最小的路径
174.      function [value,min_node_ind] = getFinalResult(rrt)
175.          % 找到所有“到目标点”的节点索引    rrt 第四个参数
176.          goal_ind = find([rrt.goalReached]==1);
177.          if ~(isempty(goal_ind))    % 判断是否达到目标点（goal 是否存在）
178.              disp('Goal has been reached!');
179.              rrt_goal = rrt(goal_ind);  % 储存“到目标点”节点
180.              value = min([rrt_goal.cost]);  % 取得上述节点中 cost 最
    小值
181.              min_node_ind = find([rrt.cost]==value);  % 找到 cost
    最小值节点的索引
182.              if length(min_node_ind)>1     % 如果 cost 最小值对应的索
    引有多个，取第一个即可
183.                  min_node_ind = min_node_ind(1);
184.              end
185.          else    % goal_ind 不存在
186.              disp('Goal has not been reached!');
187.              % 计算 rrt 中各个节点到目标点的距离，求出最小的一个节点，充
    当“终点”
188.              for i =1:length(rrt)
189.                  norm_rrt(i) = norm(p_goal-rrt(i).p);
190.              end
191.              [value,min_node_ind]= min(norm_rrt);    % 求取其中距离
    目标点最近的节点
192.              value = rrt(min_node_ind).cost;          % 以求得的该点
    信息作为返回值
193.          end
194.      end
195.
196.
197.      % 新节点的障碍物判断函数 （针对此程序的矩形障碍物判断）     ok
198.      % 参      数：新节点的坐标信息
199.      % 返 回 值：  0-有障碍物    1-无障碍物
```

```
200.      % 判断方法：节点坐标是否在矩形障碍物四个端点的约束内（可更改该函数，实现障碍物的不同变换）
201.      % param.obstacles =[130,70,20,60; 70,135,60,20;];
202.      function free=isObstacleFree(node_free)
203.          free = 1;
204.          for i = 1: length(param.obstacles(:,1))
205.              obstacle = param.obstacles(i,:);    % 取一个矩形障碍物的信息
206.              op1 = [obstacle(1), obstacle(2)];   % 根据信息求得矩形的四个端点坐标
207.              op2 = [op1(1)+obstacle(3), op1(2)];
208.              op3 = [op2(1), op1(2) + obstacle(4)];
209.              op4 = [op1(1), op3(2)];
210.
211.              nx = node_free(1);      % 取得待判断节点的 xy 坐标
212.              ny = node_free(2);
213.
214.              % 判断节点是否在障碍物范围内
215.              if ((nx>=op1(1) && nx<=op2(1)) && (ny>=op1(2) && ny<=op4(2)))
216.                  free = 0;
217.              end
218.          end
219.      end
220.
221.      % 【step_size】     ok
222.      % 沿最近点到采样点方向，按照机器人步长前进，得到新的节点
223.      % 参      数：  离采样点最近的节点   采样点
224.      % 返  回  值：  新节点的坐标
225.      function new_node=steering(nearest_node, random_node)
226.          dist = norm(random_node-nearest_node);    % 两点距离
227.          ratio_distance = param.step_size/dist;
228.          % 计算新节点的 xy 坐标   （这里也可使用其他方式进行计算）
229.          x = (1-ratio_distance).* nearest_node(1)+ratio_distance .* random_node(1);
230.          y = (1-ratio_distance).* nearest_node(2)+ratio_distance .* random_node(2);
231.
232.          new_node = [x;y];
233.      end
```

```
234.
235.      % 范围内的可到达的节点重新连接，以得到更优的路径
236.      % 参      数：  rrt 树     临近节点索引     父节点索引     新节点的索引
    --rrt 节点总数（前面传入）
237.      % 返  回  值：  更新  新节点附近的临近节点的  父节点  【建立了更短的路径
    联系】
238.      function rrt = reWire(rrt, neighbors, parent, new)
239.          for i=1:length(neighbors)       % 遍历每个可达到的临近节点
240.              cost = rrt(new).cost + norm(rrt(neighbors(i)).p - rrt
    (new).p);    % 求以新节点作为父节点时，临近节点的代价
241.
242.              if (cost<rrt(neighbors(i)).cost)  % 如果  上述新代价  小
    于  临近节点现有代价
243. %                 if norm(rrt(new).p-rrt(neighbors(i)).p)<param.s
    tep_size
244. % %                     plot(rrt(neighbors(i)).p(1), rrt(neighbor
    s(i)).p(2), '.b');
245. %                     rrt(neighbors(i)).p = steering(rrt(new).p,
    rrt(neighbors(i)).p);
246. %                 end
247. %                 plot(rrt(neighbors(i)).p(1), rrt(neighbors(i)).
    p(2), '.m');
248.                  rrt(neighbors(i)).iPrev = new;     % 将  新节点作
    为  该临近节点的  父节点
249.                  rrt(neighbors(i)).cost = cost;
250.              end
251.          end
252.      end
253.
254.      % 将新的节点插入 rrt 树末尾        ok
255.      % 参      数：  rrt   父节点索引   新节点坐标
256.      % 返  回  值：  更新 rrt
257.      % 其中第三个参数 cost = 父节点代价+新节点到父节点的代价
258.      function rrt = insertNode(rrt, parent, new_node)
259.          rrt(end+1) = struct(field1, new_node, field2, parent, fie
    ld3, rrt(parent).cost + norm(rrt(parent).p-new_node), field4, 0);
260.      end
261.
262.      % 从根节点为新子节点选择成本最低的父节点      ok   父节点？
263.      % 参      数：  rrt   临近的节点   最近的节点   新节点
```

```
264.      % 返 回 值:  父节点索引
265.      function parent = chooseParent(rrt, neighbors, nearest, new_n
    ode)
266.          min_cost = getCostFromRoot(rrt, nearest, new_node);    %
    求以最近的节点为父节点时，新节点的代价
267.          parent = nearest;                %  暂取最近的节点，作为父节点
268.          for i=1:length(neighbors)     % neighbors - 下一步可到达的
    节点
269.              cost = getCostFromRoot(rrt, neighbors(i), new_node);
      % 求以可到达的节点为父节点时，新节点的代价
270.              if (cost<min_cost)     % 最终取代价最小的那个节点，作为父
    节点
271.                  min_cost = cost;
272.                  parent = neighbors(i);
273.              end
274.          end
275.      end
276.
277.      % 父节点的 cost + 子节点到父节点的距离代价     ok
278.      % 参     数:  rrt   父节点索引   子节点
279.      % 返 回 值:  子节点的代价
280.      function cost = getCostFromRoot(rrt, parent, child_node)

281.          cost =  rrt(parent).cost + norm(child_node - rrt(parent).p)
    ;
282.      end
283.
284.      % 【neighbourhood】          ok
285.      % 获取指定节点周围可以到达的节点索引
286.      % 参     数:  rrt   node -new_node（上面调用时传的参数）
287.      % 返 回 值:  保存可到达节点的索引的 neighbors
288.      function neighbors = getNeighbors(rrt, node)
289.          neighbors = [];
290.          for i = 1:length(rrt)     % 遍历 rrt 树中所有的节点
291.              dist = norm(rrt(i).p-node);       % 计算 rrt 中各个节点到
    node 的距离代价
292.              if (dist<=param.neighbourhood)    % 找到 距离代价＜单次
    行进步长的节点索引
293.                  neighbors = [neighbors i];    % 使用 neighbors 将节
    点索引保存
```

```
            end
        end
    end

    % 在坐标图上生成随机的采样点          ok
    function node = getSample()
        x = 0;
        y = 0;
        a = 0;
        b = 200;
        node = [x;y];
        node(1) = (b-a) * rand(1) + a;
        node(2) = (b-a) * rand(1) + a;
    end

    % 寻找 rrt 树中距离采样点最近的节点    ok
    function indx = findNearest(rrt, n)
        mindist = norm(rrt(1).p - n);
        indx = 1;
        for i = 2:length(rrt)
            dist = norm(rrt(i).p - n);
            if (dist<mindist)
               mindist = dist;
               indx = i;
            end
        end
    end

end
```

2.人工势场法代码

```
clc
clear
close all

figure(1);
axis([0 20 0 20]);    % 地图 20x20
begin=[2;2];          % 起点
over=[18;18];         % 终点
```

```
9.   obstacle=[4 10 11 13 14 6 8 8;8 9 6 13 18 8 6 4];  % 障碍物 x;y 坐
     标
10.  % obstacle=[ 10 12 6 8;12 10 8 6];
11.
12.  % 绘制起点、终点、障碍物
13.  hold on;
14.  plot(begin(1),begin(2),'go','MarkerFaceColor','g');
15.  plot(over(1),over(2),'ro','MarkerFaceColor','r');
16.  text(begin(1),begin(2),'起点');
17.  text(over(1),over(2),'终点');
18.  plot(obstacle(1,:),obstacle(2,:),'ob');
19.  title('传统人工势场法生成避障路径')
20.   for i=1:size(obstacle,2)       % 在个障碍物点处，绘制椭
     圆。  'Curvature' 矩形的曲率
21.      rectangle('Position',[obstacle(1,i)-0.5,obstacle(2,i)-0.5,1,1]
     ,'Curvature',[1,1],'FaceColor','r');
22.   end
23.  scatter(obstacle(1,:),obstacle(2,:),'r')
24.  % point= path_plan(begin,over,obstacle);  % 计算并绘制出路径
25.  title('改进的人工势场法生成避障路径')
26.  point=path_plan_new1(begin,over,obstacle);
27.
28.  function [ point ] = path_plan(begin,over,obstacle)
29.
30.  iters=1;       % 迭代次数
31.  curr=begin;    % 起点坐标
32.  testR=0.4;     % 测试 8 点的圆的半径为 0.5
33.
34.  while (norm(curr-over)>0.2) &&  (iters<=2000)     % 未到终点&迭代次
     数不足
35.
36.  %     attr=attractive(curr,over);
37.  %     repu=repulsion(curr,obstacle);
38.      %curoutput=computP(curr,over,obstacle);
39.      %计算当前点附近半径为 0.2 的 8 个点的势能，然后让当前点的势能减去 8 个
     点的势能取差值最大的，确定这个
40.      %方向，就是下一步迭代的点
41.
42.      point(:,iters)=curr;     % point 为函数返回值，储存每次遍历得到的
     新起点 curr
```

```
43.
44.       %先求这八个点的坐标
45.       for i=1:8    % 求 以 curr 为起点，testR 为半径的圆上的八个均匀分布的
    点
46.           testPoint(:,i)=[testR*sin((i-1)*pi/4)+curr(1);testR*cos((
    i-1)*pi/4)+curr(2)];
47.           testOut(:,i)=computP(testPoint(:,i),over,obstacle);    %
    计算上述各个点的所受合力
48.       end
49.       [temp num]=min(testOut); % 找出这八个点中，代价最小的点
50.
51.       %迭代的距离为 0.1
52.       curr=(curr+testPoint(:,num))/2;  % 将上述求得点，迭代到 curr
    上。（求取的 curr 与 testPoint 的中点）
53.       plot(curr(1),curr(2),'*y');       % 绘制得到的 新的起点 curr
54.       pause(0.01);              % 程序暂停一会再继续运行 -- 体现出路径搜
    索的过程
55.       iters=iters+1;            % 迭代次数+1
56.   end
57.   end
58.
59.
60.    % 计算周围几个点的势能（代价）
61.    % 参数：当前起点   终点   障碍物    的坐标
62.    function [ output ] = computP( curr,over,obstacle )
63.
64.   % 几个用于计算的相关参数
65.   k_att=1;
66.   repu=0;
67.   k_rep=200;
68.   Q_star=3;      %。障碍物的斥力作用半径
69.
70.   % 计算终点对当前点的引力
71.   % tips：这个数值越小，说明当前点离终点越近
72.   attr=1/2*k_att*(norm(curr-over))^2;      % 引力计算公式
73.
74.   % 计算所有障碍物对当前点斥力合
75.   % tips：斥力合越小，说明当前点遇到障碍物的概率越小
76.   for i=1:size(obstacle,2)
```

```
77.       if norm(curr-obstacle(:,i))<=Q_star     % 障碍物到当前点距离在阈值内，考虑其斥力影响
78.           repu=repu+1/2*k_rep*(1/norm(curr-obstacle(:,i))-1/Q_star)^2;     % 斥力计算公式
79.           % ps:  当 curr 和 obstacle 坐标重合时，是否会出现 repu 计算卡死？ 是否需要对该条件进行设置
80.       else        % 障碍物到当前点距离在阈值外，忽略斥力影响
81.           repu=repu+0;
82.       end
83.   end
84.
85.   output=attr+repu;    % 引力+斥力   这个数值越小，越适合当作下一个起点
86.
87.    end
88.
89.   function [ point ] = path_plan_new1(begin,over,obstacle)
90.
91.   iters=1;        % 迭代次数
92.   curr=begin;    % 起点坐标
93.   testR=0.4;     % 测试 8 点的圆的半径为 0.5
94.
95.   while (norm(curr-over)>0.2) &&  (iters<=2000)     % 未到终点&迭代次数不足
96.
97.   %      attr=attractive(curr,over);
98.   %      repu=repulsion(curr,obstacle);
99.       %curoutput=computP(curr,over,obstacle);
100.      %计算当前点附近半径为 0.2 的 8 个点的势能，然后让当前点的势能减去 8 个点的势能取差值最大的，确定这个
101.      %方向，就是下一步迭代的点
102.
103.      point(:,iters)=curr;     % point 为函数返回值，储存每次遍历得到的新起点 curr
104.
105.      %先求这八个点的坐标
106.      for i=1:8    % 求 以 curr 为起点，testR 为半径的圆上的八个均匀分布的点
107.          testPoint(:,i)=[testR*sin((i-1)*pi/4)+curr(1);testR*cos((i-1)*pi/4)+curr(2)];
```

```
        testOut(:,i)=computP(testPoint(:,i),over,obstacle);    % 计算上述各个点的所受合力
    end
    [temp num]=min(testOut); % 找出这八个点中，代价最小的点
    %迭代的距离为 0.1
    curr=(curr+testPoint(:,num))/2;  % 将上述求得点，迭代到 curr 上。（求取的 curr 与 testPoint 的中点）
    plot(curr(1),curr(2),'*g');       % 绘制得到的 新的起点 curr
    pause(0.01);              % 程序暂停一会再继续运行 -- 体现出路径搜索的过程
    iters=iters+1;            % 迭代次数+1
end
end


 % 计算周围几个点的势能（代价）
 % 参数：当前起点   终点   障碍物    的坐标
 function [ output ] = computP( curr,over,obstacle)

att = 45;%引力增益系数
req = 10;%斥力增益系数
p0 = 3;%障碍物产生影响的最大距离，当障碍与移动目标之间距离大于 Po 时，斥力为 0。
n=length(obstacle(1,:));%障碍物个数

    %% 引力计算
    V_att = (over-curr)';%路径点到目标点的向量
    r_att = sqrt(V_att(1)^2 + V_att(2)^2);%路径点到目标点的欧氏距离
    P_att = 0.5*att * (r_att)^2;%引力

    %% 斥力计算
    %改进的人工势场法，将斥力分散一部分到引力方向。通过添加随机扰动 r_att^n 实现，r_att 为路径点到目标点的欧氏距离，本文 n 取 2。
    V_req = zeros(n,2);
    r_req=zeros(n,1);
    for j =1:n
        V_req(j,:) = [obstacle(1,j) - curr(1,:), obstacle(2,j) - curr(2,:)];%路径点到各个障碍物的向量
        r_req(j,:) = sqrt(V_req(j,1)* V_req(j,1) + V_req(j,2)* V_req(j,2));%路径点到各个障碍物的欧氏距离
```

```
    end
    P_req = 0;
    for k = 1:n
        if r_req(k,:) <= p0
            P_req1 = req * (1 / r_req(k,:) - 1 / p0) * r_att^2 / r_req(k,:)^2;%斥力分量 1：障碍物指向路径点的斥力
            P_req2 = req * (1 / r_req(k,:) - 1 / p0)^2 * r_att;%斥力分量 2：路径点指向目标点的分引力
%             P_reqk = P_req1 / r_req(k,:) * V_req(k,:) + P_req2 / r_att * V_att;%合力分散到 x,y 方向
            P_reqk = P_req1 / r_req(k,:) * V_req(k,:) + P_req2 / r_att * V_att;
            P_req = P_req + norm(P_reqk);%斥力
        end
    end
    %% 合力计算
    output = P_att + P_req;
end
```

3.全覆盖内螺旋规划代码

```
clc
clear
close all

%% 绘制初始地图
map=zeros(22,20);
% % 障碍物坐标，两行，上面为 y，下面为 x
a{1}=[ones(1,size(11:20,2))*30,ones(1,size(11:20,2))*31;11:20,11:20];
a{2}=[ones(1,size(10:12,2))*19,ones(1,size(10:12,2))*20;10:12,10:12];
a{3}=[ones(1,size(21:25,2))*21,ones(1,size(21:25,2))*26;21:25,21:25];
a{4}=[22:26;ones(1,size(22:26,2))*25];
a{5}=[9,10,11,12,13,14,15;29,29,29,29,29,29,29];
a{6}=[15,15,15;13,14,15];

for i=1:size(a,2)
    Aa=a{i};
```

```
        for j=1:size(Aa,2)
            map(Aa(1,j),Aa(2,j))=1;
        end
    end

    map(1:10,:)=[];
    map(:,1:8)=[];
    map(1,:)=ones(1,length(map(1,:)))*1;
    map(end,:)=ones(1,size(map,2))*1;
    map(:,1)=ones(length(map(:,1)),1)*1;
    map(:,end)=ones(length(map(:,end)),1)*1;
    map(17,16)=1;
    map(17,17)=1;
    %% 无障碍物
    % map(:,1)=ones(length(map(:,1)),1)*1;
    % map(:,end)=ones(length(map(:,1)),1)*1;
    % map(1,:)=ones(length(map(1,:)),1)*1;
    % map(end,:)=ones(length(map(1,:)),1)*1;

    q_start=[2,2];
    MAX=rot90(map,3);
    MAX_X=size(MAX,1);
    MAX_Y=size(MAX,2);
    axis([1 MAX_X+1, 1 MAX_Y+1])
    set(gca,'xtick',1:1:MAX_X+1,'ytick',1:1:MAX_Y+1,'GridLineStyle','
-',...
        'xGrid','on','yGrid','on')
    grid on;
    hold on;

    n=0;  % 障碍的数量
    for j=1:MAX_Y
        for i=1:MAX_X
            if (MAX(i,j)==1)
                plot(i+.5,j+.5,'ks','MarkerFaceColor','b');
                n=n+1;
            end
        end
    end
    %% 生成起点和终点
```

```
q_goal=[10,15];
% q_goal=[10,11];
plot(q_start(1)+.5,q_start(2)+.5,'go','MarkerFaceColor','g');
plot(q_goal(1)+.5,q_goal(2)+.5,'ro','MarkerFaceColor','r');
text(q_start(1)+.5,q_start(2)+.5,'起点');
text(q_goal(1)+.5,q_goal(2)+.5,'终点');
MAX=rot90(map,3); %%%设置 0,1 摆放的图像与存入的数组不一样
MAX_X=size(MAX,1); %%% 获取列数，即 x 轴长度
MAX_Y=size(MAX,2); %%% 获取行数，即 y 轴长度
axis([1 MAX_X+1, 1 MAX_Y+1]) %%%设置 x，y 轴上下限
set(gca,'xtick',1:1:MAX_X+1,'ytick',1:1:MAX_Y+1,'GridLineStyle','-',...
    'xGrid','on','yGrid','on')
grid on;  %%% 在画图的时候添加网格线
hold on; %%% 当前轴及图像保持而不被刷新，准备接受此后将绘制的图形，多图共存
n=0;%障碍的数量
for j=1:MAX_Y
    for i=1:MAX_X
        if (MAX(i,j)==1)
            %%plot(i+.5,j+.5,'ks','MarkerFaceColor','b'); 原来是红点圆表示
            fill([i,i+1,i+1,i],[j,j,j+1,j+1],'k'); %%%改成 用黑方块来表示障碍物
        end
    end
end
io=1;
Q=[2;2];
MAX(Q(1,end),Q(2,end))=1;
plot(Q(1,:)+.5,Q(2,:)+.5,'ob')
hold on
pause(0.01)
io1=0;io2=0;io3=0;io4=0;%1、2、3、4 对应判断是否能右、上、左、下走，逆时针
while io==1
    io=0;
```

```
88.       if MAX(Q(1,end)+1,Q(2,end))==0&&io1==0&&io2==0&&io3==0&&io4==
    0%对初始点单独寻找一个方向
89.           Qa=[Q(1,end)+1,Q(2,end)];
90.           MAX(Q(1,end)+1,Q(2,end))=1;%将走过的点填充成障碍物，以下同理
91.           Q=[Q,Qa'];
92.           plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
93.           hold on
94.           plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,
    2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
95.           hold on
96.           pause(0.01)
97.           io=1;
98.           io1=1;
99.       elseif MAX(Q(1,end),Q(2,end)+1)==0&&io1==0&&io2==0&&io3==0&&i
    o4==0
100.          Qa=[Q(1,end),Q(2,end)+1];
101.          MAX(Q(1,end),Q(2,end)+1)=1;
102.          Q=[Q,Qa'];
103.          plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
104.          hold on
105.          plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,
    2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
106.          hold on
107.          pause(0.01)
108.          io=1;
109.          io2=1;
110.      elseif MAX(Q(1,end)-1,Q(2,end))==0&&io1==0&&io2==0&&io3==0&&i
    o4==0
111.          Qa=[Q(1,end)-1,Q(2,end)];
112.          MAX(Q(1,end)-1,Q(2,end))=1;
113.          Q=[Q,Qa'];
114.          plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
115.          hold on
116.          plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,
    2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
117.          hold on
118.          pause(0.01)
119.          io=1;
120.          io3=1;
```

```
    elseif MAX(Q(1,end),Q(2,end)-1)==0&&io1==0&&io2==0&&io3==0&&io4==0
        Qa=[Q(1,end),Q(2,end)-1];
        MAX(Q(1,end),Q(2,end)-1)=1;
        Q=[Q,Qa'];
        plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
        hold on
        plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
        hold on
        pause(0.01)
        io=1;
        io4=1;
    end
    %io1 是向右，具体是在向右的过程中能向下就向下，然后向右不了了往上走；如果向右和向上都不行了就跳出只能向左
   %io2~io4 同理
    while io1==1
        io1=0;
        if MAX(Q(1,end),Q(2,end)-1)==0
            io4=1;
            io=1;
            break
        else
            if MAX(Q(1,end)+1,Q(2,end))==0
                Qa=[Q(1,end)+1,Q(2,end)];
                MAX(Q(1,end)+1,Q(2,end))=1;
                Q=[Q,Qa'];
                plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                hold on
                plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                hold on
                pause(0.01)
                io1=1;
                io=1;
            else
                if MAX(Q(1,end),Q(2,end)+1)==0
                    Qa=[Q(1,end),Q(2,end)+1];
                    MAX(Q(1,end),Q(2,end)+1)=1;
```

```
                    Q=[Q,Qa'];
                    plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                    hold on
                    plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                    hold on
                    pause(0.01)
                    io1=1;
                    io=1;
                end
            end
        end
    end

    while io2==1
        io2=0;
        if MAX(Q(1,end)+1,Q(2,end))==0
            io1=1;
            io=1;
            break
        else
            if MAX(Q(1,end),Q(2,end)+1)==0
                Qa=[Q(1,end),Q(2,end)+1];
                MAX(Q(1,end),Q(2,end)+1)=1;
                Q=[Q,Qa'];
                plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                hold on
                plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                hold on
                pause(0.01)
                io2=1;
                io=1;
            else
                if MAX(Q(1,end)-1,Q(2,end))==0
                    Qa=[Q(1,end)-1,Q(2,end)];
                    MAX(Q(1,end)-1,Q(2,end))=1;
                    Q=[Q,Qa'];
                    plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                    hold on
```

```
                    plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                    hold on
                    pause(0.01)
                    io2=1;
                    io=1;
                end
            end
        end
    end

    while io3==1
        io3=0;
        if MAX(Q(1,end),Q(2,end)+1)==0
            io2=1;
            io=1;
            break
        else
            if MAX(Q(1,end)-1,Q(2,end))==0
                Qa=[Q(1,end)-1,Q(2,end)];
                MAX(Q(1,end)-1,Q(2,end))=1;
                Q=[Q,Qa'];
                plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                hold on
                plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                hold on
                pause(0.01)
                io3=1;
                io=1;
            else
                if MAX(Q(1,end),Q(2,end)-1)==0
                    Qa=[Q(1,end),Q(2,end)-1];
                    MAX(Q(1,end),Q(2,end)-1)=1;
                    Q=[Q,Qa'];
                    plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                    hold on
                    plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                    hold on
```

```
                    pause(0.01)
                    io3=1;
                    io=1;
                end
            end
        end
    end

    while io4==1
        io4=0;
        if MAX(Q(1,end)-1,Q(2,end))==0
            io3=1;
            io=1;
            break
        else
            if MAX(Q(1,end),Q(2,end)-1)==0
                Qa=[Q(1,end),Q(2,end)-1];
                MAX(Q(1,end),Q(2,end)-1)=1;
                Q=[Q,Qa'];
                plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                hold on
                plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                hold on
                pause(0.01)
                io4=1;
                io=1;
            else
                if MAX(Q(1,end)+1,Q(2,end))==0
                    Qa=[Q(1,end)+1,Q(2,end)];
                    MAX(Q(1,end)+1,Q(2,end))=1;
                    Q=[Q,Qa'];
                    plot(Q(1,end)+.5,Q(2,end)+.5,'ob')
                    hold on
                    plot([Q(1,size(Q,2))+.5,Q(1,size(Q,2)-1)+.5],[Q(2,size(Q,2))+.5,Q(2,size(Q,2)-1)+.5],'-b')
                    hold on
                    pause(0.01)
                    io4=1;
                    io=1;
```

```
                    end
                end
            end
        end
    end
end
disp('搜索完成')
```