中国地质大学（武汉）

智能系统技术实践课程报告

学　　院：自动化学院

课　　程：智能系统技术实践

指导老师：王亚午

学　　号：20201000128

班　　级：231202

姓　　名：刘瑾瑾

2023 年 12 月 3 日

目录

第一章　实验目的

设计移动机器人路径规划和轨迹跟踪控制算法，使用 MATLAB 或 Simulink 进行代码化或图形化编程，在 Gazebo 环境下控制移动机器人，并实现其运动控制目标。

第二章　实验内容

(1) ROS 的基本原理；
(2) ROS 的通讯机制；
(3) Matlab/Simulink 对 ROS 编程的语法结构；
(4) 移动机器人路径规划算法设计；
(5) 移动机器人轨迹跟踪控制算法设计；
(6) Gazebo 环境下的移动机器人运动控制仿真实验。

第三章　实验原理

3.1　基本要求

(1) 使用 rosinit 命令创建 ROS 环境；
(2) 使用 inflate 命令对地图进行膨胀操作；
(3) 使用 PRM 算法为移动机器人规划可行运动路径；
(4) 为移动机器人创建 PurePursuit 控制器，用于轨迹跟踪控制；
(5) 创建 rospublisher 和 rossubscriber，分别用于向移动机器人发送速度控制指令和从机器人端接收位姿信息；
(6) 使用 rosshutdown 命令结束 ROS 交互。

3.2 ROS 的基本原理

ROS 是一个适用于机器人的开源的元操作系统。它提供了操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。在某些方面 ROS 相当于一种“机器人框架”。

ROS 的主要目标是 为机器人研究和开发提供代码复用的支持。ROS 是一个分布式的进程框架，这些进程被封装在易于被分享和发布的程序包和功能包集中。ROS 也支持一种类似于代码储存库的联合系统，这个系统也可以实现工程的协作及发布。这个设计可以使一个工程的开发和实现从文件系统到用户接口完全独立决策（不受 ROS 限制）。同时，所有的工程都可以被 ROS 的基础工具整合在一起。

点对点的分布式通信机制是 ROS 的核心，使用了基于 TCP/IP 的通信方式，实现模块间点对点的松耦合连接，可以执行若干种类型的通信，包括基于话题的异步数据流通信，基于服务的同步数据流通信，还有参数服务器上的数据存储等。

3.3 路径规划算法

路径规划是指在给定的环境中，找到从起点到目标点的最佳路径的过程。根据对环境信息的把握程度可把路径规划划分全局路径规划和局部路径规划。根其中，从获取障碍物信息是静态或是动态的角度看，全局路径规划属于静态规划(又称离线规划)，局部路径规划属于动态规划(又称在线规划)。

通常使用启发式方法如 A*，遗传算法、模拟退火和粒子群等等来进行路径优化选择，在本次实习中我们采用 PRM 和 RRT 两种方法规划路径，分别构成搜索树或者路线图。

3.3.1 概率路线图法（PRM）

PRM（Probabilistic Roadmap Method 概率路线图法）算法是基于采样算法的一种，很好的解决了在高维空间中构造出有效路径图的困难。该算法通过在构形空间中进行采样、对采样点进行碰撞检测、测试相邻采样点是否能够连接来表示路径图的连通性。此方法的一个优点是，其复杂度主要依赖于寻找路径的难度，跟整个规划场景的大小和构形空间的维数关系不大。然而当规划的路径需要通过密集的障碍物或者需要经过狭窄的通道时，PRM 方法的效率变的低下。

概率路线图 PRM 主要分为两个阶段，离线学习阶段中随机采样大量的机器人位姿点，为每个节点搜索邻居节点并建立连接，构建出路标地图；在线查询阶段中根据起始点，目标点，路标地图信息，采用启发式搜索算法从路标图搜索出一条可行路径。路线图算法可以有效避免对位姿空间中障碍物进行精确建模，能够有效解决复杂的运动动力学约束下的路径规划问题。

算法流程包括采样、生成概率路图、搜索路径三个步骤：

（1）采样：在地图中随机撒点，剔除落在障碍物上的点；

（2）生成概率路图：根据点与点间的距离和是否存在直线通路将上步中得到的采样点进行连接；

（3）搜索路径：使用图搜索算法(如 Dijkstra 算法)在上步得到的路图中搜索出一条从起点到终点的最短路径。

3.3.2 快速拓展随机树算法（RRT）

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}。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}。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;若连线经过障碍物，则放弃本次迭代，重新选取随机点。

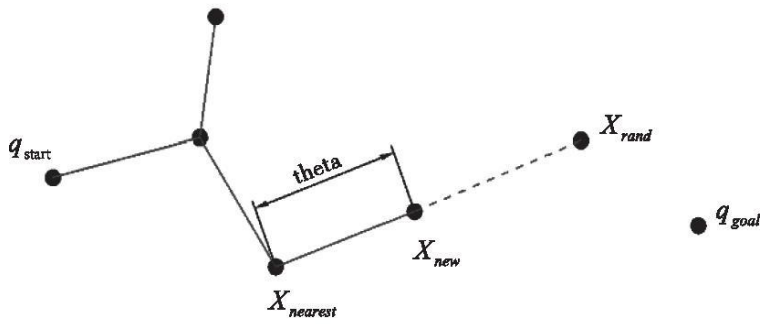

图 3- 1 RRT 原理

3.4 跟踪控制算法

3.4.1 纯跟踪路径跟踪算法（Pure Pursuit）

Pure Pursuit 是一种几何跟踪控制算法，也被称为纯跟踪控制算法。其思想是基于当前车辆的后轮中心位置（车辆质心），在参考路径上向l_d（称为前视距离）的距离匹配一个预瞄点，假设车辆后轮中心可以按照一定的转弯半径R行驶至该预瞄点，然后根据前视距离l_d、转弯半径R、车辆坐标系下预瞄点的朝向角α之间的几何关系来计算前轮转角δ。

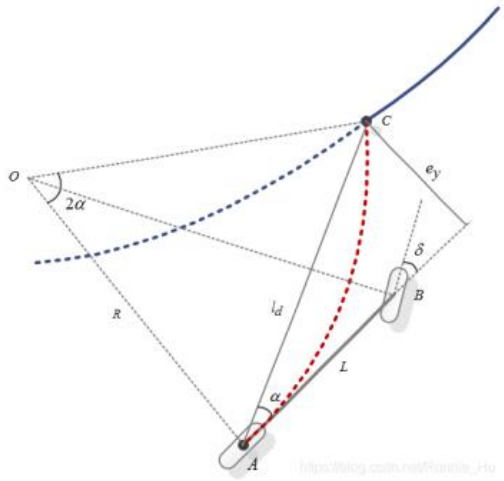

图 3- 2 Pure Pursuit 算法原理图

Pure Pursuit 并非常规的控制器，而是一种路径跟踪算法。其独特性在于，它仅依赖航向点的信息，能够明确所需的线速度及最大角速度。该算法具有简单、稳定和可靠的特点，因此在许多自动驾驶和智能车辆的应用中被广泛使用。

3.4.2 滑模控制算法（Sliding Mode Control）

滑模控制（Sliding Mode Control，SMC）是一种非线性控制方法，核心思想是建立一个滑模面，将被控系统拉取到滑模面上来，使系统沿着滑模面运动。滑模控制的一个优势是无视外部扰动和不确定参数，从而达到控制的目的。

滑膜控制的一般步骤包括：

（1）系统建模： 首先，对系统进行建模，了解系统的动态特性，确定系统的状态方程和输出方程。

（2）选择滑模面： 选择一个合适的滑模面，它通常是一个超平面，可以使系统的状态在这个面上快速收敛。滑模面的选择需要考虑系统的特性以及控制设计的目标。

（3）引入滑模控制器： 根据选择的滑模面，引入一个控制器来使系统的状态快速滑动到滑模面上。控制器的设计通常包括根据滑模面的误差调整控制输入。

（4）稳定性分析： 对引入滑模控制器后的系统进行稳定性分析，确保系统在滑模面上稳定。

在本次实习中，滑模控制是通过计算滑模面 s 和基于该滑模面的控制输入 u 来实现的。滑模面计算为 $s = \sin(\theta) + \mu \cdot \text{norm}(\text{robotCurrentPose} - \text{robotGoal})$ 其中θ是机器人当前位置到目标位置的方向，μ是滑模面的设计参数。控制输入 u 的计算为 $u = -\alpha \cdot \text{sigmoid}(s)$或者 $u = -\alpha \cdot s \cdot \text{sign}(s)$，其中$\alpha$是控制增益。

第四章　实验结果

4.1　膨胀地图

膨胀地图是在原始地图的基础上考虑机器人的实际大小，而对障碍物进行膨胀（增加半径），起到从位形空间到工作空间的映射作用，有助于规划和执行路径时更好地避开障碍物，确保机器人在运动过程中不会与障碍物碰撞。

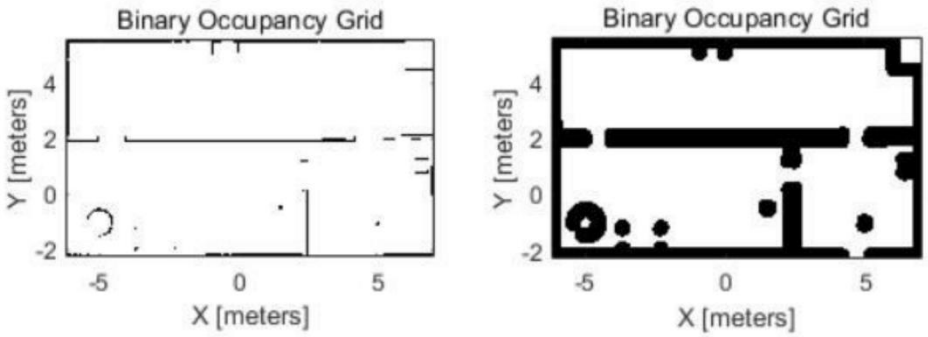

图 4- 1 膨胀地图

4.2　路径规划

4.2.1　PRM 路径规划

使用例程中给定的 PRM 算法进行路径规划，纯跟踪控制算法 Pure Pursuit 进行路径跟踪。控制效果如下：

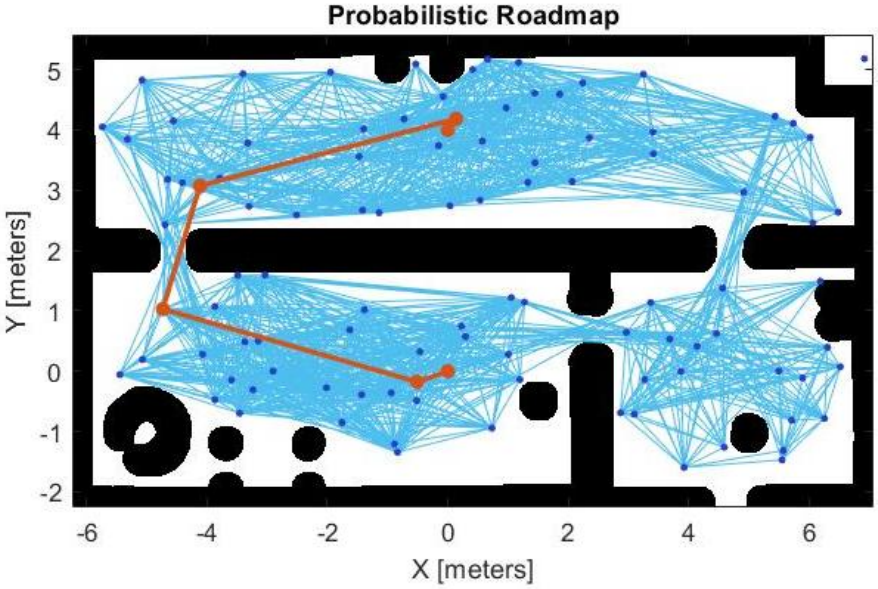

图 4- 2 PRM 路径规划结果 1

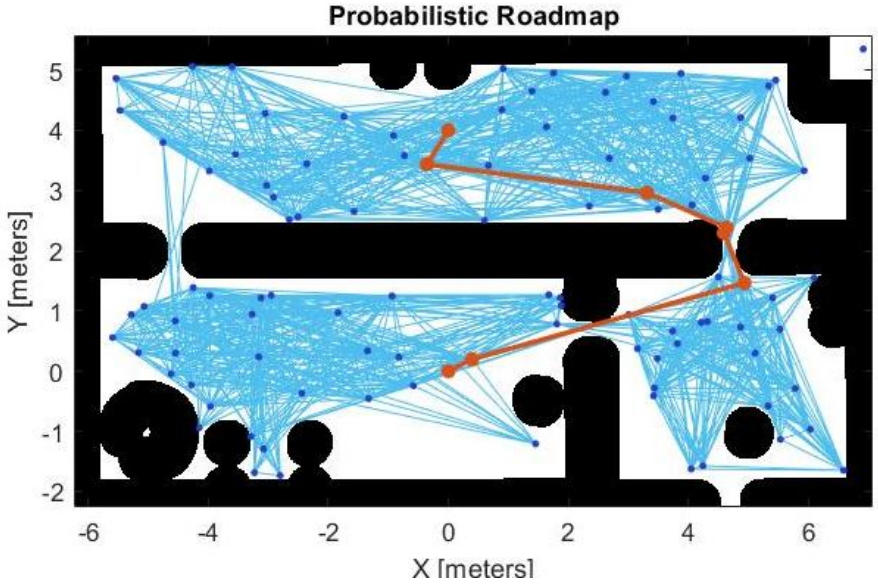

图 4- 3 PRM 路径规划结果 2

图 4-2 和图 4-3 的路径规划结果不同，PRM 在自由空间中进行随机采样，从而得到一组采样点，会得到不同的采样点分布，从而影响了生成的图结构，在连接采样点时，根据碰撞检测的结果选择可以连接的边，由于不同的碰撞检测结果可能在不同的运行中出现，因此可能选择不同的连接，从而导致不同的路径。

4. 2. 2 RRT 路径规划

在 PRM 算法的基础上，我们使用 RRT 算法进行路径规划。设置 RRT 算法

中 MaxConnectionDistance 参数分别位 0.25，0.5，0.75 和 1.0，得到下面的实验结果：

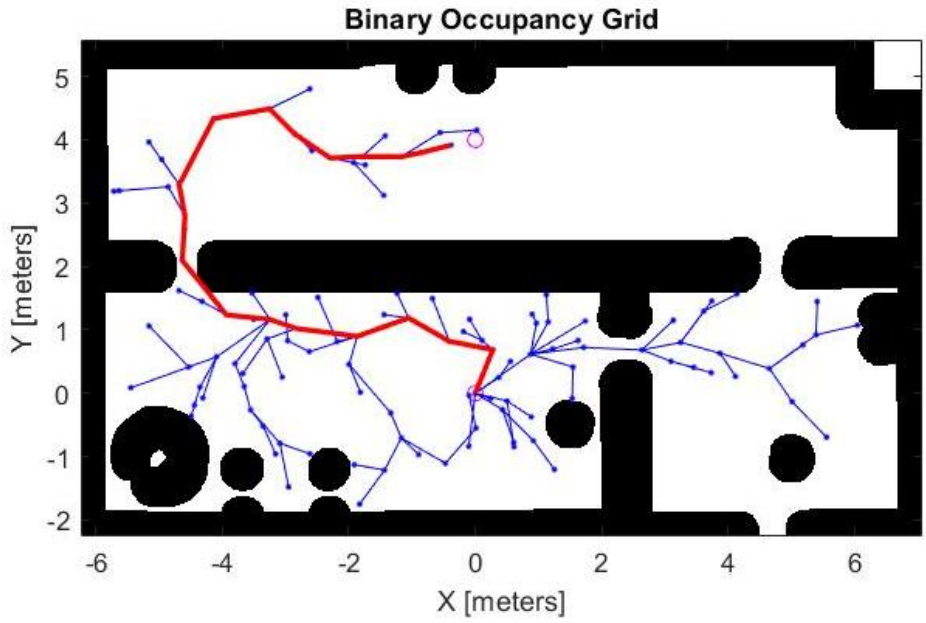

图 4- 4 RRT 算法路径规划结果（MaxConnectionDistance=1.0）

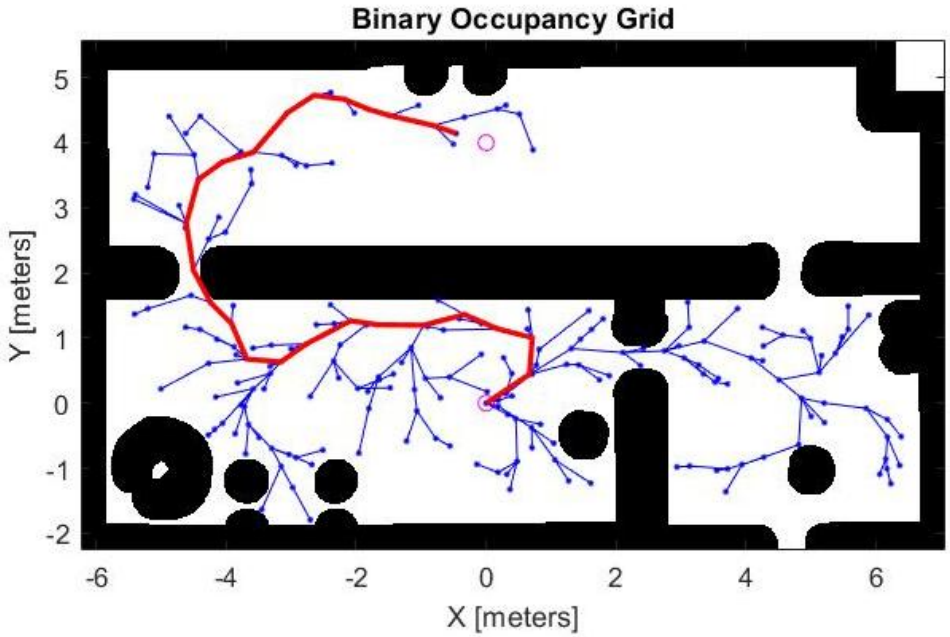

图 4- 5 RRT 算法路径规划结果（MaxConnectionDistance=0.75）

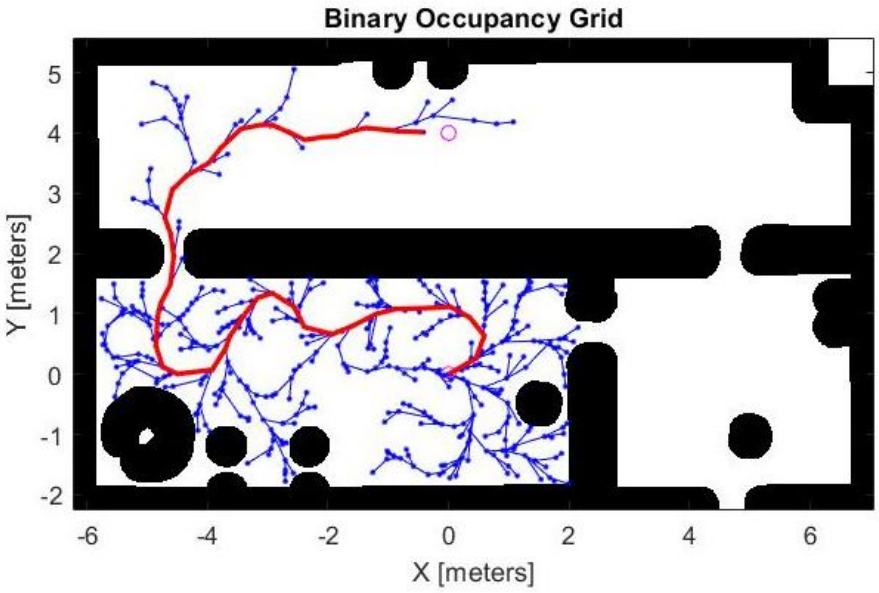

图 4- 6 RRT 算法路径规划结果（MaxConnectionDistance=0.50）

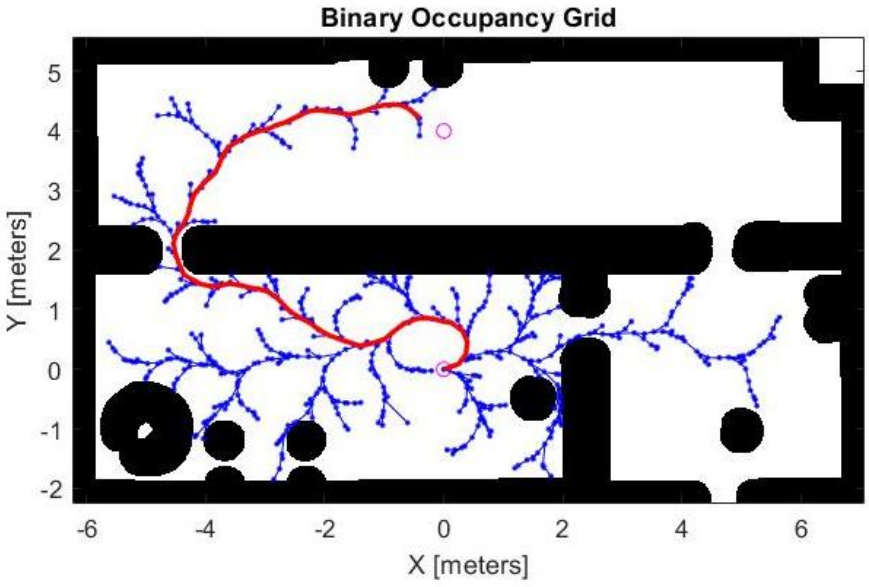

图 4- 7 RRT 算法路径规划结果（MaxConnectionDistance=0.25）

从图 4-2 和图 4-3 可以看出，PRM 和 RRT 两种路径规划方法的不同。PRM 和 RRT 在构建连通图和探索空间方面存在显著差异。PRM 采用随机采样节点的方式，即在地图上均匀地随机放置点，并去除与障碍物重叠的点，然后将所有可视点进行连接，构建出连通图。而 RRT 通过生长方式逐渐探索空间，以初始点作为根节点，并随机采样增加叶子节点，形成随机扩展树，并在此树上寻找最优路径。简言之，PRM 注重随机采样，而 RRT 则强调逐步生长和探索。

另外，从图 4-4、图 4-5、图 4-6 和图 4-7 可以看出步长对于 RRT 算法构建

的生成树的影响。较大的最大连接距离允许更远探索，使得算法更快地覆盖整个可行空间，但是能导致树在某些区域过于稀疏，而在其他区域过于密集；小的最大连接距离可以导致更细致的局部路径，有助于生成更优质的路径，但也容易陷入局部最优解，增大计算开销。调整 RRT 的最大连接距离从 1.0 改为 0.75、0.5 和 0.25，可以使路径更加平滑，便于机器人进行转向。路径变得更加平滑，有利于机器人进行转向，但过小可能导致路径更加弯曲、路径长度变大，陷入局部最优解。

4.3 滑模控制

将跟踪控制算法改为滑模控制算法，路径规划算法位 PRM。在滑膜控制中，设计滑模面计算为 $s = \sin(\theta) + \mu \cdot \mathrm{norm}(\mathrm{robotCurrentPose} - \mathrm{robotGoal})$ 其中θ是机器人当前位置到目标位置的方向，μ是滑模面的设计参数。控制输入 u 的计算为 $u = -\alpha \cdot \mathrm{sigmoid}(s)$或者 $u = -\alpha \cdot s \cdot \mathrm{sign}(s)$，其中$\alpha$是控制增益，结果如下：

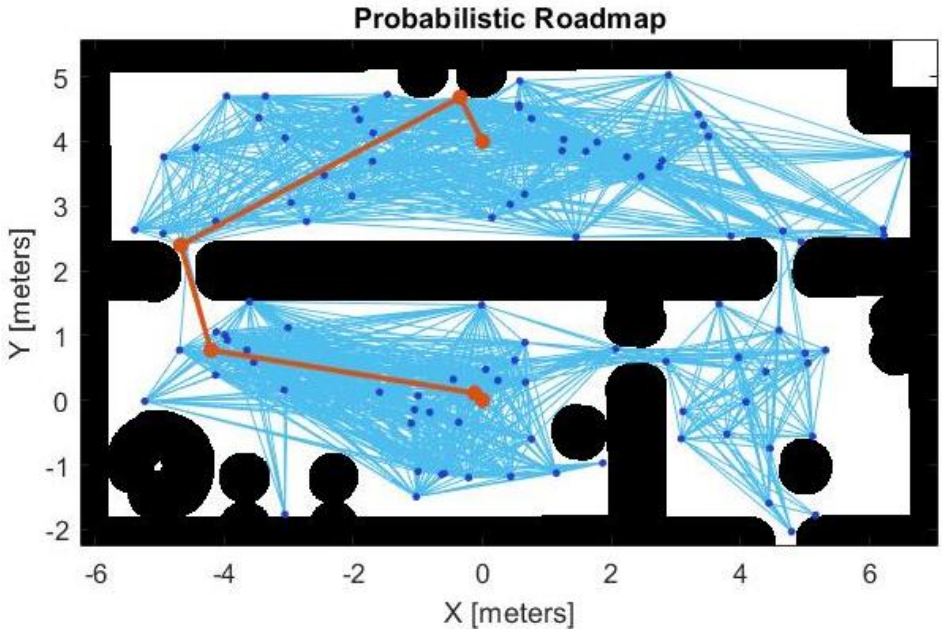

图 4- 6 滑模控制路线规划

由于纯跟踪算法和滑模控制算法仅为跟踪算法，具体效果在路径规划中并不能体现出，具体运行见验收视频。

第五章　总结

本次实习主要研究了基于 Matlab/Simulink 的 ROS 移动机器人运动控制问题。本次实习主要是为了让我们学习移动机器人路径规划和轨迹跟踪控制算法，以及在 Gazebo 环境下控制移动机器人，完成对其的运动控制。在实习中还学到了 ROS 的基本原理以及 ROS 的通讯机制，初步了解了如何使用虚拟机，在 Gazebo 环境下进行移动机器人的运动控制仿真实验。

通过本次智能系统技术实践课程，我学习了解了 ROS 机器人的基本原理，同时也对 PRM、RRT、Pure Pursuit 和 SMC 等算法有了较深的理解，对于移动机器人的路径规划和轨迹规划有了进一步理解，巩固了《机器人控制系统》课程学习的相关知识。

当然，本次实习还存在许多不足的地方，如没有使用 Simulink 进行仿真验证等。在移动机器人控制中，还可以采用人工势场法等方法，或者采用自适应控制或者预测控制，这些工作有待未来进一步探索。

在实践过程中遇到了很多问题与困难，但都经过自己的思考、与同学相互交流、上网查找资料逐一解决了，在发现问题、解决问题的过程中我也受益匪浅，对我今后学习机器人相关的知识打下了基础，十分感谢王老师悉心的指导。

第六章　附录

```
clear;close all;clc;
%% 创建 ROS 环境
rosshutdown;
rosinit('192.168.114.129');
%% 路径规划
load("officemap.mat");
robotRadius=0.25;
mapInflated=copy(map);
%膨胀障碍物，得到机器人的位形空间
inflate(mapInflated,(robotRadius+0.05));
start = [0, 0, 0];%设置起点
goal = [0, 4, 0];%设置终点
bounds = [mapInflated.XWorldLimits; mapInflated.YWorldLimits; [-pi
pi]];
 ss = stateSpaceDubins(bounds);
 ss.MinTurningRadius = 0.4; %最小转弯半径
 stateValidator = validatorOccupancyMap(ss);
 stateValidator.Map = mapInflated;
 stateValidator.ValidationDistance = 0.05;
 planner = plannerRRT(ss, stateValidator);
 planner.MaxConnectionDistance = 1.0;
 planner.MaxIterations = 30000;
 rng(0,'twister') %随机数生成
[pthObj, solnInfo] = plan(planner, start, goal); %使用 RRT*路径规划器
规划车辆路径
 %作图显示路径规划结果
 figure ( );
 show(mapInflated);%显示地图
 hold on;
 plot(start(1), start(2), 'mo',goal(1), goal(2), 'mo');%显示起点终点
 hold on;
```

```
    plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), 'b.-');%显示生成树
    hold on;
    plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWidth',2)%显示路径
    interpolate(pthObj,300) % 插值
    %% 定义机器人的当前位姿
    robotCurrentLocation = [start(1)+0.02, start(2)-0.05];    %定义机器人的初始位置(设置误差)
    robotGoal = [goal(1), goal(2)];                %定义机器人的目标位置
    initialOrientation = 0+0.01;                %定义机器人的初始朝向(设置误差)
    robotCurrentPose = [robotCurrentLocation initialOrientation];   %定义机器人的当前位姿
    %% 创建控制器（纯追踪算法）
    controller = controllerPurePursuit;      % 创建算法
    controller.Waypoints = pthObj.States(:,1:2);            % 设置目标路径的组成点
    controller.DesiredLinearVelocity = 0.3;
    controller.MaxAngularVelocity = 3;
    controller.LookaheadDistance = 0.5;
    goalRadius=0.05;
    distanceToGoal=norm(robotCurrentLocation - robotGoal);
    %% 定义 Publisher 和 Subscriber
    velcomd=rospublisher('/mobile_base/commands/velocity');
    %velcomd=rospublisher('/cmd_vel');
    vel=rosmessage(velcomd);
    odomsub=rossubscriber('/odom');
    %% 运动控制
    while( distanceToGoal > goalRadius )
    [v1,omega1] = step(controller, robotCurrentPose); % 根据当前 Pose，计算速度和转角
    vel.Linear.X=v1;
    vel.Angular.Z=omega1;
    send(velcomd,vel); % 驱动机器人
```

```
odom=receive(odomsub,1); % 收集位姿信息
robotCurrentPose=MyreadPose(odom);
robotCurrentLocation = robotCurrentPose(1:2);  % 获取驱动后 Pose
distanceToGoal = norm(robotCurrentLocation - robotGoal);  % 计算偏差
waitfor(rateControl(20));                                 % 延迟等待
end
  % %滑模控制
  % 初始化控制器参数
  lambda = 1; % 滑模面的设计参数
  alpha = 1; % 控制增益
  while( distanceToGoal > goalRadius )
  % [v1,omega1] = step(controller, robotCurrentPose); % 根据当前 Pose，
计算速度和转角
  theta=
atan2(endLocation(2)-robotCurrentPose(2),endLocation(1)-robotCurren
tPose(1))
  % 计算滑模面
  s = sin(theta) + lambda * norm(robotCurrentPose - robotGoal);
  % 计算控制输入
  u = -alpha * sign(s);
  % 根据控制输入计算速度
  v = u;
  % 计算角速度
  omega = (1/v) * u;
  vel.Linear.X=v;
  vel.Angular.Z=omega;
  send(velcomd,vel); % 驱动机器人
  odom=receive(odomsub,1); % 收集位姿信息
  robotCurrentPose=MyreadPose(odom);
  robotCurrentLocation = robotCurrentPose(1:2); % 获取驱动后 Pose
  distanceToGoal = norm(robotCurrentLocation - robotGoal); % 算偏差
  waitfor(rateControl(20)); % 延迟等待
  end
```