哈尔滨工业大学（深圳）

**《移动机器人导论》课程实验报告**

（2024-2025 秋季学期）

**课程名称 ：** 移动机器人导论

**题 目 ：** 差分驱动机器人运动控制

**班级学号** ： 210320111

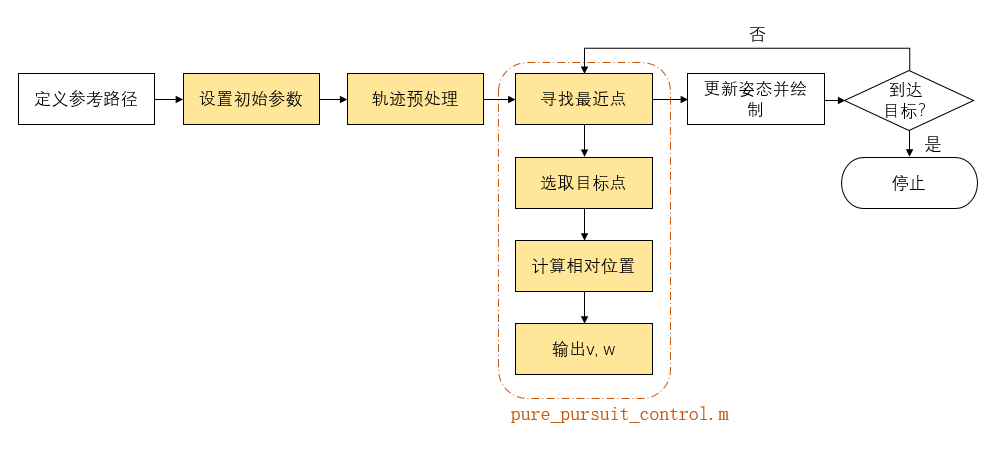
**学生姓名 ：** 吕家昊

2024年 10 月 28 日

**实验一 差分驱动轮式机器人的建模与运动控制**

**一、实验流程框图**

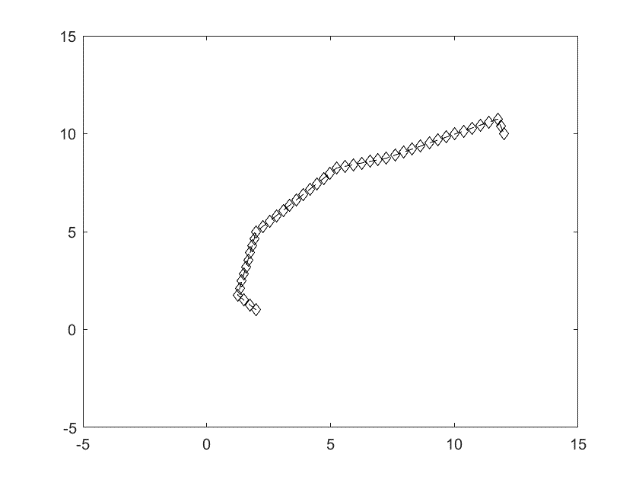
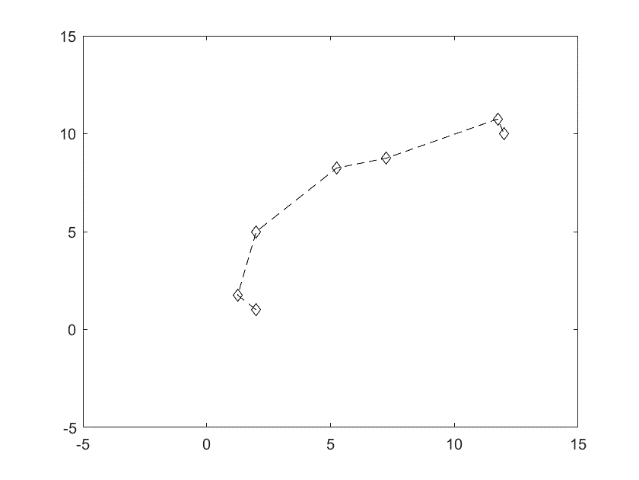
**代码逻辑框图**



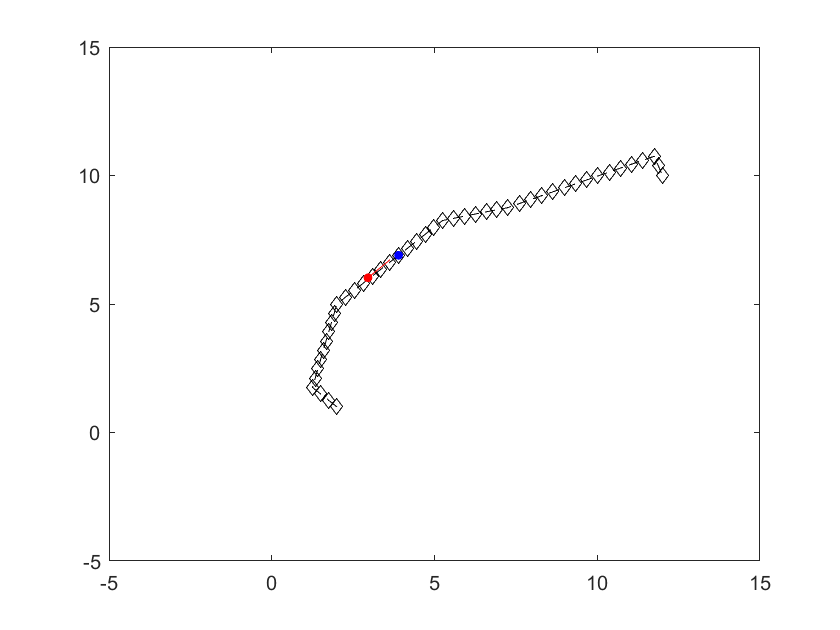
设置初始参数：与分别为倍率与视距常数，最小预瞄距离。

轨迹预处理：由于初始轨迹各路径点距离可能过大，因此需要在两个过远的路径点之间插入一些点，使得所有路径点的距离不超过。

即原始路径两点间距为时，等距插入个点。下图为插入路径点前后对比。



每一时刻，小车寻找离自身最近的路径点（下称“**当前路径点**”），并由当前路径点开始依次访问后续节点，直到出现首个距离当前位置大于的节点，或选取终点。下图蓝色路径点为**预瞄路径点**，红色为当前位置。



此后计算出小车坐标系下，预瞄路径点所在位置，取角速度。线速度使用单P控制器，即。

**二、实验程序代码**

主程序（该文件修改部分进行高亮）

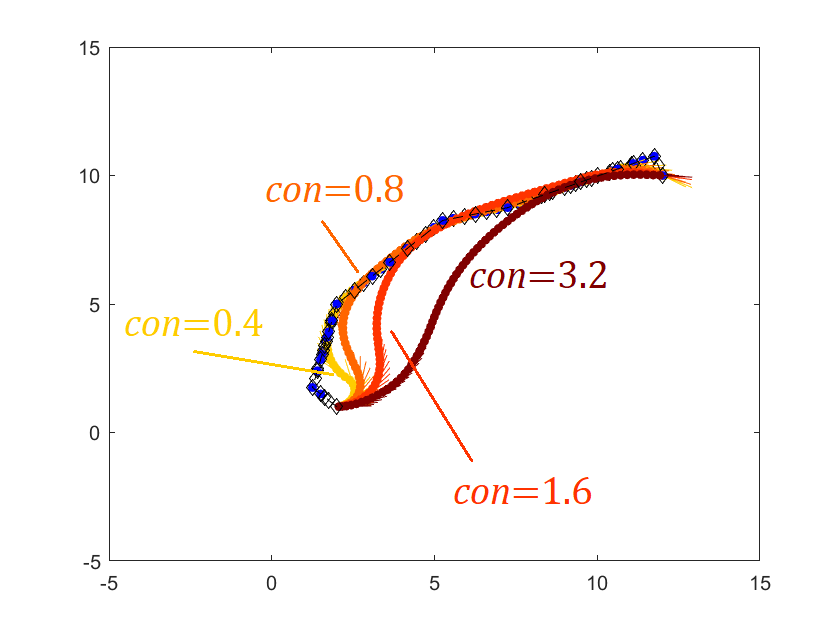
|  |
| --- |
| addpath(genpath('robotcore'))  clear;  clc;  % 参考路径3  pathx = 0:0.5:15;  pathy = sqrt(pathx) + sin(pathx);  path = [pathx', pathy'];  %% 相关参数定义  H = 1; % 车辆轴距，单位：m  goalRadius = 0.05; % 当车辆距离目标点小于此值时结束跟踪  sampleTime = 0.1; % 仿真时间，单位s  vizRate = rateControl(10); % 控制频率  % 设置机器人初始状态  robotInitialLocation = path(1,:); % 初始位置  initialOrientation = 0; % 初始航向角  robotCurrentPose = [robotInitialLocation initialOrientation]'; % [x; y; theta];  v0 = 0; % 初始线速度  w = 0; % 初始角速度  robotGoal = path(end,:); % 确定目标点  vr = 2; % 目标线速度  distanceToGoal = norm(robotInitialLocation - robotGoal); % 计算当前位置与目标点的距离  v = v0; % 初始速度  yaw = initialOrientation; % 初始航向角  % 确定车辆框架尺寸以最接近代表车辆  frameSize = H/0.5;  con = 0.4; % 视距常数  k = 0.4; % 倍率  % 轨迹预处理, 使得相邻轨迹点之间的距离不小于con  x\_tmp = [];  y\_tmp = [];  for i=1:size(path,1)-1  add\_points = floor(sqrt((path(i,1) - path(i+1,1)) ^ 2 + (path(i,2) - path(i+1,2)) ^ 2) / con) + 2;  x\_tmp = [x\_tmp, linspace(path(i,1), path(i+1,1), add\_points)];  y\_tmp = [y\_tmp, linspace(path(i,2), path(i+1,2), add\_points)];  end  path = [x\_tmp, path(size(path,1),1); y\_tmp, path(size(path,1),2)]';  while( distanceToGoal > goalRadius )  % 更新轨迹跟踪图  hold off    % 绘制参考轨迹  plot(path(:,1), path(:,2),"k--d")  hold all    % 计算控制器的输出，将其输入到机器人中  [v, w] = pure\_pursuit\_control(robotCurrentPose, path, v, vr, k, con); % 机器人系下v与w  % 使用控制输入得到机器人的速度  vel = derivative(robotCurrentPose, v, w, sampleTime);    % 更新当前姿态  robotCurrentPose = robotCurrentPose + vel\*sampleTime;    % 重新计算与目标点的距离  distanceToGoal = norm(robotCurrentPose(1:2) - robotGoal(:));    % 绘制机器人的运行轨迹  plotTrVec = [robotCurrentPose(1:2); 0];  plotRot = axang2quat([0 0 1 robotCurrentPose(3)]);  plotTransforms(plotTrVec', plotRot, "MeshFilePath", "groundvehicle.stl", "Parent", gca, "View","2D", "FrameSize", frameSize);  light;  % 设置图的横纵坐标  xlim([-5 15]);  ylim([-5 15]);    waitfor(vizRate); % 暂停代码执行，知道计时器vizRate出发，vizRate为采样频率    if distanceToGoal<0.1 % 到达目标后停止  break  end  end |

pure\_pursuit\_control.m

|  |
| --- |
| function [v, w] = pure\_pursuit\_control(robotCurrentPose, path, v, vr, k, con)  % 寻找离小车最近的点  now\_index = 0;  now\_dist = 999999;  for i=1:size(path,1)  if sqrt((path(i,1) - robotCurrentPose(1)) ^ 2 + (path(i,2) - robotCurrentPose(2)) ^ 2) < now\_dist  now\_dist = sqrt((path(i,1) - robotCurrentPose(1)) ^ 2 + (path(i,2) - robotCurrentPose(2)) ^ 2);  now\_index = i;  end  end  % 选取目标点  tar\_index = size(path,1);  vis = v \* k + con; % 预瞄距离  for i=now\_index+1:size(path,1)  if sqrt((path(i,1) - robotCurrentPose(1)) ^ 2 + (path(i,2) - robotCurrentPose(2)) ^ 2) > vis  tar\_index = i;  break  end  end  scatter(path(tar\_index,1), path(tar\_index,2), 20, 'b', 'filled'); % 绘制目标  % 计算相对位置  if tar\_index ~= size(path,1)  point\_pose = atan2(path(tar\_index+1,2) - path(tar\_index,2), path(tar\_index+1,1) - path(tar\_index,1));  else  point\_pose = atan2(path(tar\_index,2) - path(tar\_index-1,2), path(tar\_index,1) - path(tar\_index-1,1));  end  delPose = [path(tar\_index, 1:2)'; point\_pose] - robotCurrentPose;  delPose = [ cos(robotCurrentPose(3)), sin(robotCurrentPose(3)), 0;  -sin(robotCurrentPose(3)), cos(robotCurrentPose(3)), 0;  0, 0, 1] \* delPose;  if delPose(3) > pi  delPose(3) = delPose(3) - 2\*pi;  elseif delPose(3) < -pi  delPose(3) = delPose(3) + 2\*pi;  end  delPose  v = v + 0.3\*(vr - v); % 使用单P控制器实现线速度的控制  w = 2 \* v \* delPose(2) / (delPose(1)^2 + delPose(2)^2);  end |

**三、实验结果及分析**

取不同值时运动轨迹如下图，小车模型已省略。（始终取）



**四、实验结论**

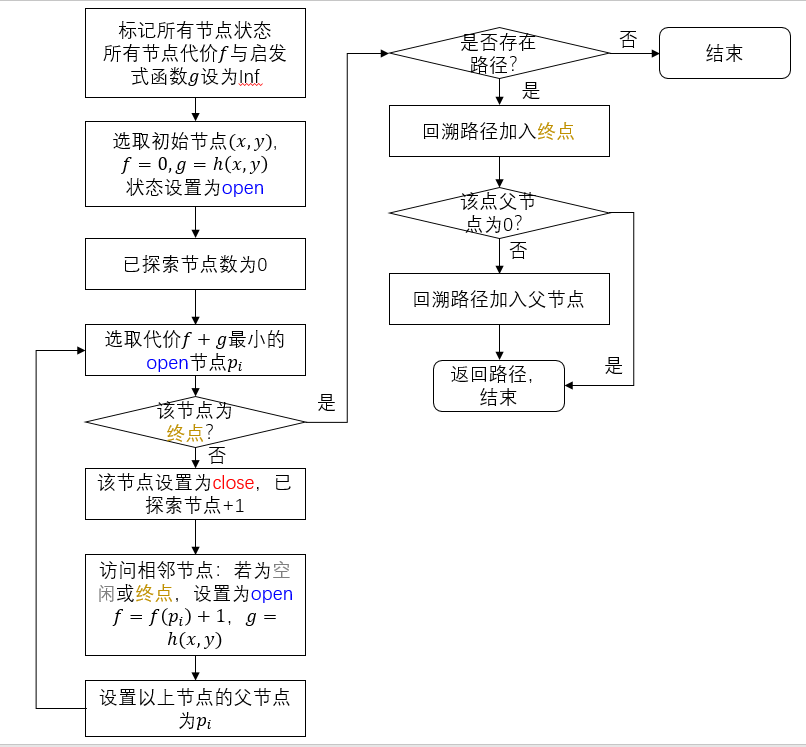
采用纯跟踪算法控制小车时，关于世界坐标系下的系统稳定。

当更大时，小车跟踪精度降低，但跟踪速度快；更小时，小车跟踪精度高，但速度较慢。

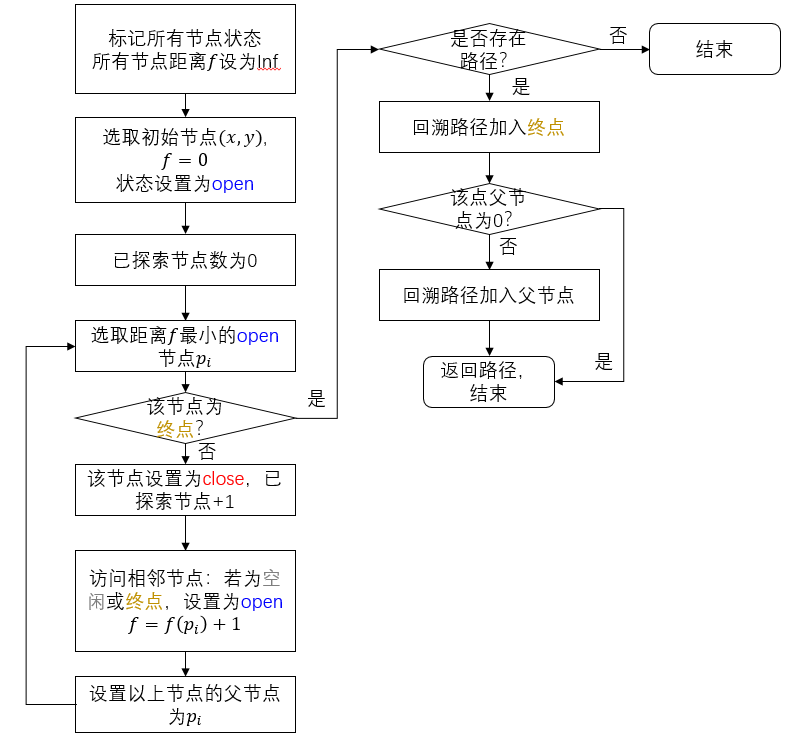
**实验二 差分驱动轮式机器人的路径规划（一）**

**一、实验流程框图**

A\*算法：



Dijkstra算法：



**二、实验程序代码**

主程序

|  |
| --- |
| clc  clear  close all  addpath(genpath('robotcore'))  % 栅格地图的大小  mapSize = [30, 30];  % 障碍物占据概率（0-1之间的值）  obstacleProbability = 0.2;  % 生成随机栅格地图  gridMap = rand(mapSize) < obstacleProbability;  % 设置起点和终点  start\_coords = [2, 5];  dest\_coords = [25, 28];  gridMap = logical(gridMap); % 转换为逻辑数组  % 调用A\*算法输入路径  [route, numExpanded,I,J] = AStarGrid (gridMap, start\_coords, dest\_coords);  % 调用Dijkstra算法输入路径  % [route, numExpanded,I,J] = DijkstraGrid (gridMap, start\_coords, dest\_coords);  path=[I;J]'; % 生成参考路径  %% 此处与实验一代码一致，使用纯跟踪算法对规划好的参考路径进行跟踪  % 相关参数定义  H = 1; % 车辆轴距，单位：m  goalRadius = 0.05; % 当车辆距离目标点小于此值时结束跟踪  sampleTime = 0.05; % 仿真时间，单位s  vizRate = rateControl(10); % 控制频率  % 设置机器人初始状态  robotInitialLocation = path(1,:); % 初始位置  initialOrientation = pi/2; % 初始航向角  robotCurrentPose = [robotInitialLocation initialOrientation]';  idx\_target = 1;  v0 = 5; % 初始线速度  w = 0; % 初始角速度  robotGoal = path(end,:); % 确定目标点  vr = 5; % 目标线速度  distanceToGoal = norm(robotInitialLocation - robotGoal); % 计算当前位置与目标点的距离  v = v0; % 初始速度  yaw = initialOrientation; % 初始航向角  % 确定车辆框架尺寸以最接近代表车辆，以便于在图上显示  frameSize = H/0.5;  while( distanceToGoal > goalRadius )    % 计算控制器的输出，将其输入到机器人中  [v, w] = pure\_pursuit\_control(robotCurrentPose, path, v, vr);  % 使用控制输入得到机器人的速度  vel = derivative(robotCurrentPose, v, w, sampleTime);    % 更新当前姿态  robotCurrentPose = robotCurrentPose + vel\*sampleTime;    % 重新计算与目标点的距离  distanceToGoal = norm(robotCurrentPose(1:2) - robotGoal(:));  % 更新轨迹跟踪图  hold off  % show(map); % 使栅格地图保留在轨迹跟踪图中  map1 = imrotate(gridMap,90);  image(0.5, 0.5, gridMap);  hold all  % 绘制参考轨迹  plot(path(:,1), path(:,2),"k--d")  hold all    % 绘制机器人的运行轨迹  plotTrVec = [robotCurrentPose(1:2); 0];  plotRot = axang2quat([0 0 1 robotCurrentPose(3)]);  plotTransforms(plotTrVec', plotRot, "MeshFilePath", "groundvehicle.stl", "Parent", gca, "View","2D", "FrameSize", frameSize);  light;  % 设置图的横纵坐标  xlim([0 27]);  ylim([0 26]);    waitfor(vizRate); % 暂停代码执行，知道计时器vizRate出发，vizRate为采样频率  end |

AStarGrid.m

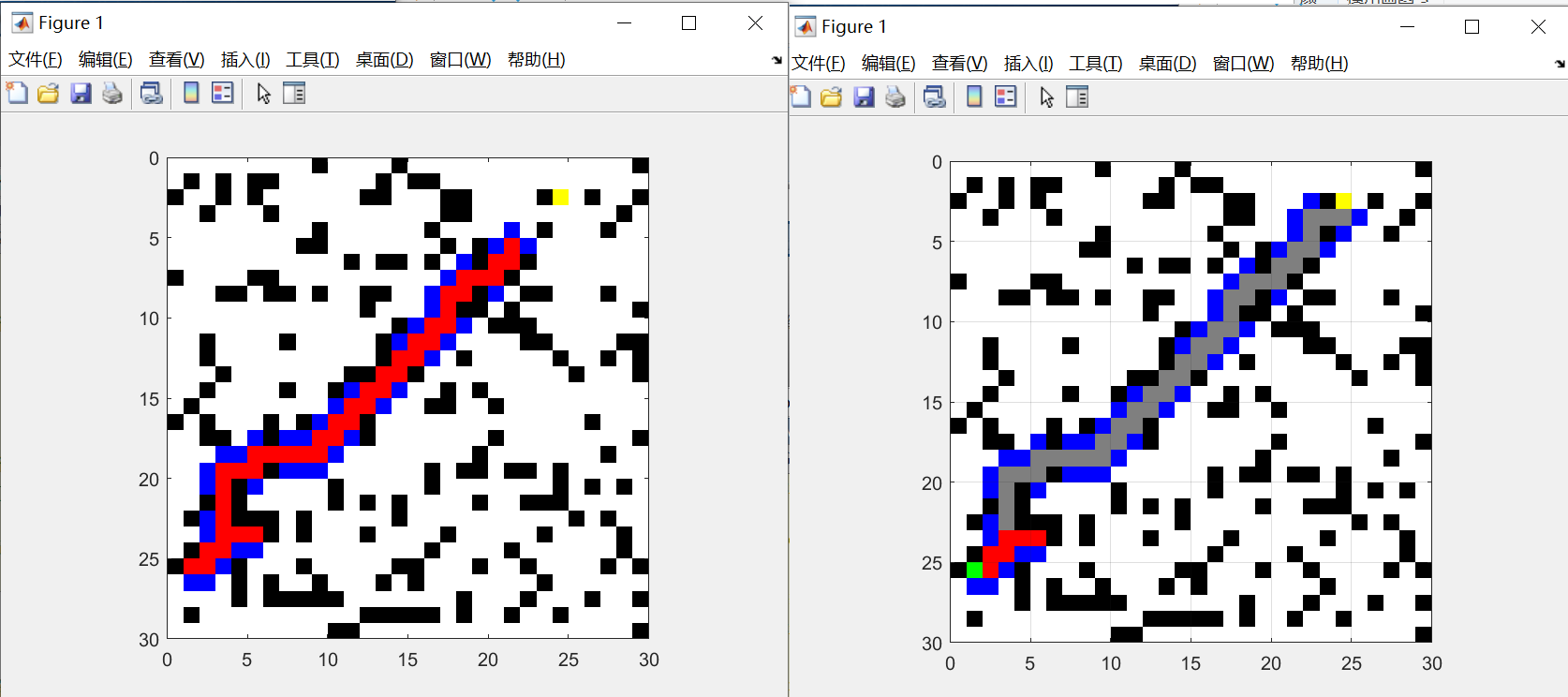
|  |
| --- |
| function [route,numExpanded,I,J] = AStarGrid (input\_map, start\_coords, dest\_coords)  cmap = [1 1 1; ... 白色，表示空闲的格子  0 0 0; ... 黑色，表示障碍物  1 0 0; ... 红色，表示已访问的格子  0 0 1; ... 蓝色，表示正在考虑的格子  0 1 0; ... 绿色，表示起点  1 1 0; ... 黄色，表示终点  0.5 0.5 0.5]; %灰色，表示路径上的格子  colormap(cmap);  % 是否将每次迭代地图进行可视化展示  drawMapEveryTime = true;  [nrows, ncols] = size(input\_map);  % map - 一个二维数组，用于跟踪每个格子的状态  map = zeros(nrows,ncols);  map(~input\_map) = 1; % 标记空闲格子  map(input\_map) = 2; % 标记障碍物格子  % 产生起点和终点的线性索引  start\_node = sub2ind(size(map), start\_coords(1), start\_coords(2));  dest\_node = sub2ind(size(map), dest\_coords(1), dest\_coords(2));  map(start\_node) = 5; % 标记起点  map(dest\_node) = 6; % 标记终点  parent = zeros(nrows,ncols); % 用来记录每个节点的父节点  [X, Y] = meshgrid (1:ncols, 1:nrows); % 函数可以将输入的向量扩展为二维网格  xd = dest\_coords(1); % 目标节点的 x 坐标  yd = dest\_coords(2); % 目标节点的 y 坐标  % 用欧氏距离距离作为启发式函数  H = sqrt((X - xd).\*(X-xd) + (Y - yd).\*(Y-yd));  H = H';  % 初始化成本评价函数，将代价设置为正无穷表示还没探索过这些节点  f = Inf(nrows, ncols);  g = Inf(nrows, ncols);  g(start\_node) = 0;  f(start\_node) = H(start\_node);  % 记录已经被探索的节点数  numExpanded = 0;  % Main Loop  while true  if (drawMapEveryTime)  map1=imrotate(map,90);  image(0.5, 0.5, map1);  axis image; % 设置坐标轴的宽高比为1，使地图绘制时不发生形变  drawnow; % 立即更新图形窗口  end  % 寻找f最小的点，并返回最小值min\_f和对应的索引current  [min\_f, current] = min(f(:) + abs(map(:) - 4) \* 10000); % 只会访问open(map=4)的点  % 判断条件是否满足其中之一，说明已找最短路径或无法到达终点  % 跳出循环，结束路径搜索  if ((current == dest\_node) || isinf(min\_f))  break;  end  % 更新地图  map(current) = 3; % 将当前节点标记为已访问状态  numExpanded = numExpanded + 1;    % 计算当前节点的行列坐标  [i, j] = ind2sub(size(f), current);  % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  % 两行星号\*之间编写代码，实现以下功能：  % 访问当前节点的每个相邻节点  % 根据访问结果更新地图、f和g和父节点表  % 访问节点  if i > 1  if map(i-1, j) == 1 || map(i-1, j) == 6  map(i-1, j) = 4;  parent(i-1, j) = sub2ind(size(map), i, j);  g(i-1, j) = g(i, j) + 0.1;  end  end  if j > 1  if map(i, j-1) == 1 || map(i, j-1) == 6  map(i, j-1) = 4;  parent(i, j-1) = sub2ind(size(map), i, j);  g(i, j-1) = g(i, j) + 0.1;  end  end  if i < nrows  if map(i+1, j) == 1 || map(i+1, j) == 6  map(i+1, j) = 4;  parent(i+1, j) = sub2ind(size(map), i, j);  g(i+1, j) = g(i, j) + 0.1;  end  end  if j < ncols  if map(i, j+1) == 1 || map(i, j+1) == 6  map(i, j+1) = 4;  parent(i, j+1) = sub2ind(size(map), i, j);  g(i, j+1) = g(i, j) + 0.1;  end  end  f = g + H;  %\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  end  %% 构建从起点到终点的路径，并进行可视化  if (isinf(f(dest\_node))) % 判断是否存在路径  route = [];  else  % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  % 两行星号\*之间编写代码，实现以下功能：  % 构建从终点到起点回溯路径  route = [];  I = [];  J = [];  now = dest\_node;  while now ~= 0 % 循环在起点停止  route = [route now];  I = [I X(now)];  J = [J Y(now)];  now = parent(now);  end  numExpanded = numExpanded - 1; % 长度去除终点  I = I - 0.5;  J = J - 0.5;    % 重绘起点与终点  map(start\_node) = 5;  map(dest\_node) = 6;    % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  % 用于可视化地图和路径的代码片段  for k = 2:length(route) - 1  map(route(k)) = 7;  pause(0.04);  map1 = imrotate(map,90);  image(0.5, 0.5, map1);  grid on;  axis image;  end  end  end |

DijkstraGrid.m

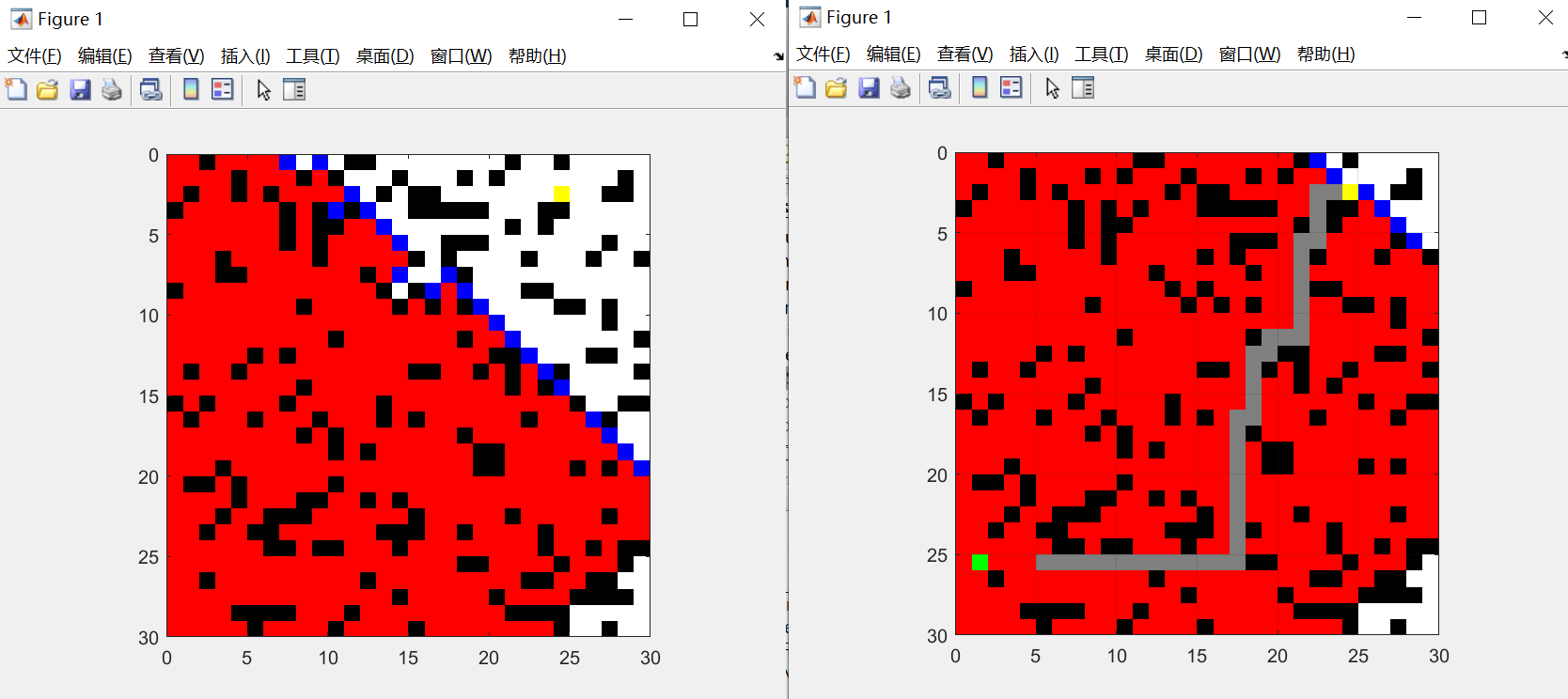
|  |
| --- |
| function [route,numExpanded,I,J] = DijkstraGrid (input\_map, start\_coords, dest\_coords)  cmap = [1 1 1; ... 白色，表示空闲的格子  0 0 0; ... 黑色，表示障碍物  1 0 0; ... 红色，表示已访问的格子  0 0 1; ... 蓝色，表示正在考虑的格子  0 1 0; ... 绿色，表示起点  1 1 0; ... 黄色，表示终点  0.5 0.5 0.5]; %灰色，表示路径上的格子  colormap(cmap);  % 是否将每次迭代地图进行可视化展示  drawMapEveryTime = true;  [nrows, ncols] = size(input\_map);  % map - 一个二维数组，用于跟踪每个格子的状态  map = zeros(nrows,ncols);  map(~input\_map) = 1; % 标记空闲格子  map(input\_map) = 2; % 标记障碍物格子  % 产生起点和终点的线性索引  start\_node = sub2ind(size(map), start\_coords(1), start\_coords(2));  dest\_node = sub2ind(size(map), dest\_coords(1), dest\_coords(2));  map(start\_node) = 5; % 标记起点  map(dest\_node) = 6; % 标记终点  % 初始化距离数组，除起点外所有节点的距离都为无穷大  distanceFromStart = Inf(nrows, ncols);  % 存储每个网格单元格的父节点索引  parent = zeros(nrows,ncols);  [X, Y] = meshgrid (1:ncols, 1:nrows); % 函数可以将输入的向量扩展为二维网格  % 初始化距离数组，除起点外所有节点的距离都为无穷大  distanceFromStart(start\_node) = 0;  % 将起点节点的距离设置为0  numExpanded = 0;  % Main Loop  while true  if (drawMapEveryTime)  map1=imrotate(map,90);  image(0.5, 0.5, map1);  axis image; % 设置坐标轴的宽高比为1，使地图绘制时不发生形变  drawnow; % 立即更新图形窗口  end    % 寻找最小距离点，并返回最小值min\_dist和对应的索引current  [min\_dist, current] = min(distanceFromStart(:) + abs(map(:) - 4) \* 10000); % 只会访问open(map=4)的点  % 判断条件是否满足其中之一，说明已找最短路径或无法到达终点  % 跳出循环，结束路径搜索  if ((current == dest\_node) || isinf(min\_dist))  break;  end    % 更新地图  map(current) = 3; % 将当前节点标记为已访问状态    numExpanded = numExpanded + 1;  % 计算当前节点的行列坐标  [i, j] = ind2sub(size(distanceFromStart), current);  % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  % 两行星号\*之间编写代码，实现以下功能：  % 访问当前节点的每个相邻节点  % 根据访问结果更新地图、f和g和父节点表  if i > 1  if map(i-1, j) == 1 || map(i-1, j) == 6  map(i-1, j) = 4;  parent(i-1, j) = sub2ind(size(map), i, j);  distanceFromStart(i-1, j) = distanceFromStart(i, j) + 1;  end  end  if j > 1  if map(i, j-1) == 1 || map(i, j-1) == 6  map(i, j-1) = 4;  parent(i, j-1) = sub2ind(size(map), i, j);  distanceFromStart(i, j-1) = distanceFromStart(i, j) + 1;  end  end  if i < nrows  if map(i+1, j) == 1 || map(i+1, j) == 6  map(i+1, j) = 4;  parent(i+1, j) = sub2ind(size(map), i, j);  distanceFromStart(i+1, j) = distanceFromStart(i, j) + 1;  end  end  if j < ncols  if map(i, j+1) == 1 || map(i, j+1) == 6  map(i, j+1) = 4;  parent(i, j+1) = sub2ind(size(map), i, j);  distanceFromStart(i, j+1) = distanceFromStart(i, j) + 1;  end  end  % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  end  %% 构建从起点到终点的路径，并进行可视化  if (isinf(distanceFromStart(dest\_node))) % 判断是否存在路径  route = [];  else  % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  % 两行星号\*之间编写代码，实现以下功能：  % 构建从终点到起点回溯路径  route = [];  I = [];  J = [];  now = dest\_node;  while now ~= 0 % 循环在起点停止  route = [route now];  I = [I X(now)];  J = [J Y(now)];  now = parent(now);  end  numExpanded = numExpanded - 1; % 长度去除终点  I = I - 0.5;  J = J - 0.5;    % 重绘起点与终点  map(start\_node) = 5;  map(dest\_node) = 6;  % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  % 用于可视化地图和路径的代码片段  for k = 2:length(route) - 1  map(route(k)) = 7;  pause(0.1);  map1 = imrotate(map,90);  image(0.5, 0.5, map1);  grid on;  axis image;  end  end  end |

**三、实验结果及分析**

A\*算法：



Dijkstra算法：

****

**四、实验结论**

由于相邻点之间的距离均相等，因此Dijkstra算法会依次访问整个区域内可到达的所有节点，耗时更长，但能获得最优解。

A\*算法通过引入启发式函数，使得访问方向能“朝向终点”，明显提高了路径规划速度。

**实验三 差分驱动轮式机器人的全局路径规划（二）**

**一、实验过程**

1.开启底盘与激光雷达，控制机器人在室内进行移动并建图。

2.加载上一步建图，启动路径规划终端。

直接打开终端时，rviz中不显示地图，或显示不正确的地图，这是因为上一步中保存的地图路径与加载地图的路径不同。打开navigation\_imu.launch，将地图放至文件提到的路径中，此时地图正常加载。

3.小幅度移动机器人，在rviz中观察机器人可能的位置，逐渐收敛后标注目前起点位置与方向。

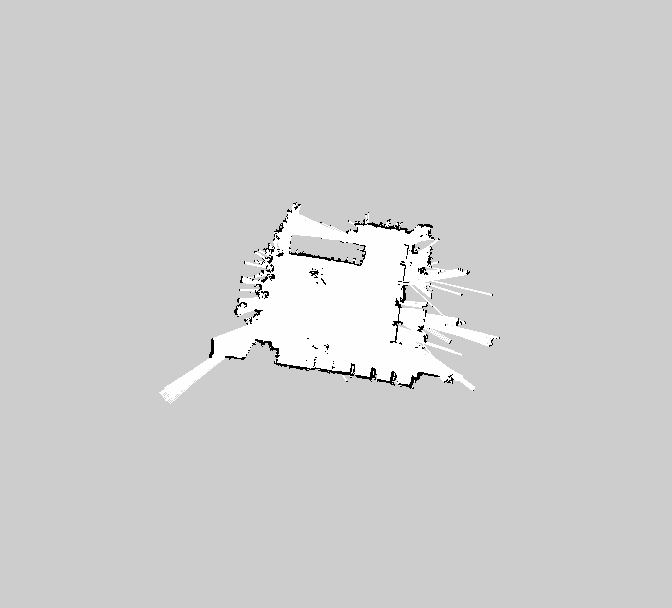
此步骤机器人移动速度不能过大，否则地图上的粒子可能发散。

4.标注路径目标点，机器人开始轨迹规划与移动。

5.修改global planner参数，重复步骤3、4。

**二、实验结果及分析**

建图效果



**三、实验结论**

lethal\_cost初始为255。当将该值调整为128后，机器人运动会更为“保守”，即与障碍物会保持更远的距离。

**实验四 差分驱动轮式机器人的局部路径规划**

**一、实验过程**

1.机器人定位，与实验三步骤3相同。

2.标注目标点，机器人开始运动。

3.修改teb\_local\_planner参数，重复步骤1、2。

**二、实验结果及分析**

weight\_obstacle（优化过程中，和障碍物最小距离的权重）初始为50。减小weight\_obstacle至20后，机器人会距离障碍（另一台静止的机器人）更近的位置进行转向，能规划更优的路径。

当将该值调整至300，机器人无法通过障碍并到达目标点。移走障碍后机器人可到达目标。

yaw\_goal\_tolerance（目标角度偏移容忍度）增大后，机器人到达目标点停止时的方向，与规划时目标方向会存在偏差。