扫地机器人模拟

一、 实验内容

现有的扫地机器人的路线规划方法一般有随机覆盖法，人工势场法，栅格法，模板模型法，人工智能法。

假设有一款扫地机器人，通过硬件可以模拟 GPS 卫星进行三点定位。该机器人还可以通过感应器避免碰撞周围的物体。

请为这种机器人设计扫地路线，并与随机覆盖法比较，并通过系统仿真评价你设计的方法的效率。

二、 实验目的

- 理解系统建模的过程，思路
- 理解算法设计的过程
- 学习可视化程序的方法

三、 组内成员

姓名	学号
张惠东	2017100101021
樊思明	2017100101013
罗秋林	2017100101012

四、 问题分析

问题的重点在于如何设计扫地路线能够更好的提高扫地机器人的扫地效率，对于本文来说，实现最大化效率即在满足清扫房间全区域覆盖的前提下，尽量满足扫地机器人移动的行进总路径最短。同时实现扫地机器人随机游走和路径规划两种方案的实现，对比总路径的大小，从而可以得到效率优劣。关键为以下几点：

1）如何抽象出机器人的关键属性。

2）需要将扫地区域离散化，怎么离散化。

3）怎样描述地图的属性（障碍物，是否被清扫过）

4）需要让仿真更加形象，怎么实现如增加图形显示扫地的过程。

五、 模型假设

将系统的有关模拟因素罗列如下:

1) 障碍物和房间的规模是扫地机器人的整数倍。

2) 清扫区域地面完全平整，只有凸出的障碍物存在。

3) 扫地机器人行进匀速。

4) 扫地机器人清扫方式为固定式前向吸尘，且清扫面积等于其占地面积。

5) 扫地机器人转向时只能转 90 度。

6) 扫地机器人在运行之前会得到房间的地图和自己的位置。

7) 假设机器人的初始位置在坐标原点处。

六、 模型建立与算法设计

1) 为了便于储存房间的信息，将房间栅格化，并且将栅格模型中的每一个栅格看成一个点。实际中的地图是连续，但在处理时看成离散的栅格。2) 将栅格模型抽象为标识矩阵，矩阵对应位置的标记表示栅格对应位置的状态，0 表示对应位置无障碍物，1 表示对应位置是障碍物。对房间地图分块随机生成障碍物。3) 由于栅格模型是离散的，我们可以进一步抽象出图模型，将栅格模型中可到达的点看作图的节点，相邻栅格看作可以相互到达的边，进而使用深度优先搜索算法实现图的全遍历，并记录下运动路径，和路径长度。4) 规定随机覆盖算法中机器人只能向四个方向运动且向每个方向运动的概率相同，记录运动路径和实现全清扫的路径长度。5) 可视化栅格模型和扫地机器人的运动过程，用黑色网格表示障碍物的位置，蓝色网格表示清扫一次。

七、 运行结果和结果分析

多次运行程序，发现深度优先搜索的路径长度约为 480，随机覆盖的路径长度约为 5000 11000，可以看出深度优先搜索的路径规划方法不仅步数少，而且具有很强的稳定性。

选取其中一次运行结果如图 1，图 2 所示（图 1 为路径规划，图 2 为随机覆盖。

路径规划总长度为 479，随机覆盖总长度为 7156。

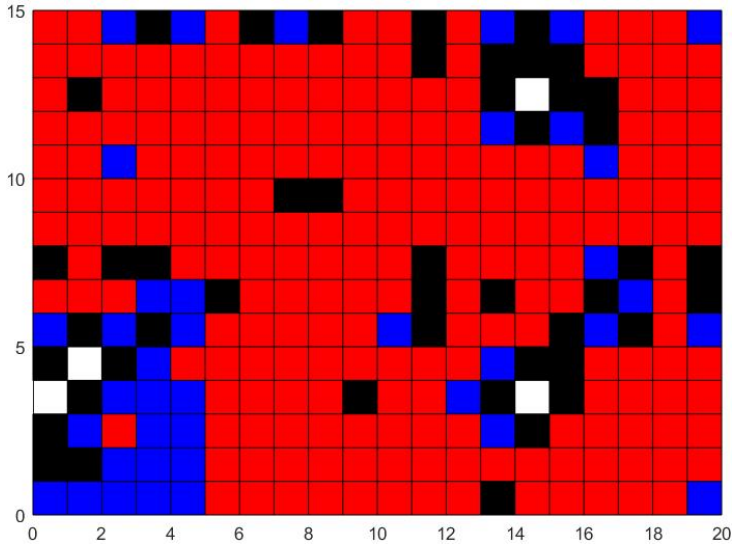

图 1　路径规划结果图

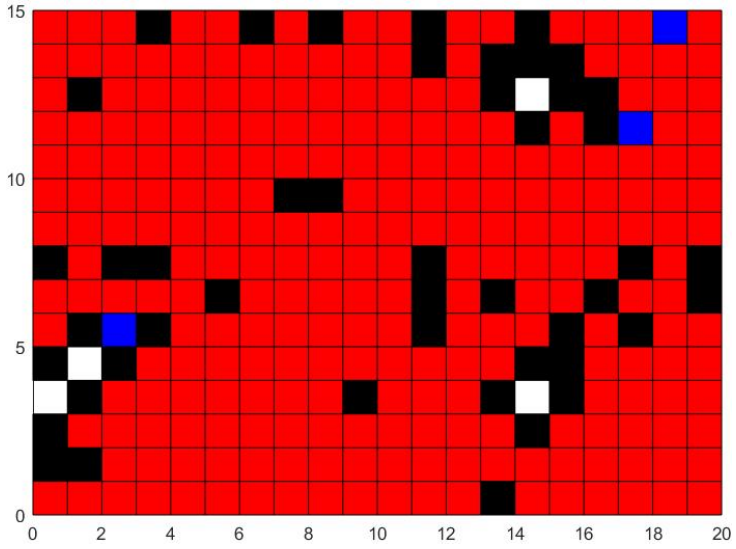

图 2　随机覆盖结果图

附录 A　扫地机器人模拟算法–matlab 源程序

```
%% 扫地机器人模拟，主函数
function [step_num_plan, step_num_random] = robot_sweeper()
% 返回深度优先搜索实现全覆盖的运行次数
%将栅格模型的每一个栅格看成一个点
%实际中栅格模型是连续的，在计算机处理时看作离散的

%将栅格模型抽象为标识矩阵，矩阵对应位置的标记表示栅格对应位置的状态
%0表示对应位置无障碍物，1表示对应位置是障碍物

%参数设置
size = 1; %扫地机器人的尺寸
```

```
map_x = 20; %房间的尺寸
map_y = 15;

%生成地图网格对应的标识矩阵
tag = zeros(map_x, map_y);
%随机生成障碍物，也是通过矩阵生成
Tag = barrier_generate(tag);
Tag(1, 1) = 0;
%构建图的邻接压缩表
b = graph_convert(Tag);
[re, node_num] = DFS(b, Tag);
step_num_plan = length(re);
result_display(re, Tag);
%进行随机碰撞过程展示
step_num_random = random_display(Tag, node_num);
%% 生成标记矩阵
function Tag = barrier_generate(Tag)
x_num = 3;
y_num = 2;
[map_x, map_y] = size(Tag);
x_1 = round(map_x / x_num); %仅在地图不是太小的时候成立
x_2 = round(map_x * 2 / x_num);
y_1 = round(map_y / y_num);
x_area = [0 x_1 x_2 map_x];
y_area = [0 y_1 map_y];
for i = 1:length(x_area) - 1
    for j = 1:length(y_area) - 1
        temp_x = x_area(i+1) - x_area(i);
        temp_y = y_area(j+1) - y_area(j);
        obs_1 = round(rand(temp_x, temp_y));
        obs_2 = round(rand(temp_x, temp_y));
        obs = conjunction(obs_1, obs_2); %取两次随机结果的合取保证有通路
        Tag([x_area(i)+1:x_area(i+1)], [y_area(j)+1:y_area(j+1)]) = obs;
    end
end
%% 生成障碍物，是barrier_generate的子函数
function obs = conjunction(obs_1, obs_2)
[map_x, map_y] = size(obs_1);
for i = 1:map_x
    for j = i:map_y
        if obs_1(i,j) == 1 & obs_2(i,j) == 1
            obs(i,j) = 1;
        else obs(i,j) = 0;
        end
    end
end
%% 深度优先搜索
```

```
function [re, node_num] = DFS(b, Tag)
%传入图的压缩表，传出深度优先搜索的路径
%压缩表中最大值就是邻接矩阵的宽与高
m = max(b(:));
%从邻接压缩表构造图的矩阵表示
A = compresstable2matrix(b, Tag);
node_num = 0;
for i = 1:length(A)
    if isempty(find(A(i,:) == 1)) ~= 1
        node_num = node_num + 1;
    end
end
count = 1;
top = 1;
stack(top) = 1; %将第一个节点入栈
flag = 1; %标记某个节点是否访问过了
re = [];
% node_num
while top ~= 0 %判断堆栈是否为空
    pre_len = length(stack); %搜寻下一个节点前的堆栈长度
    i = stack(top); %取堆栈顶节点
    for j = 1:m
        if A(i,j) == 1 && isempty(find(flag == j,1)) %如果节点相连并且没有访问过
            top = top+1; %扩展堆栈
            stack(top) = j; %新节点入栈
            flag = [flag j]; %对新节点进行标记
            re = [re;i j]; %将边存入结果
            count = count + 1;
            break
        end
    end
    %如果堆栈长度没有增加，则节点开始出栈
    if length(stack) == pre_len
        if count == node_num
            break
        end
        if top ~= 1
            re = [re;stack(top) stack(top-1)];
            stack(top) = [];
            top = top-1;
        else
            %           re = [re;stack(top) 1];
            stack(top) = [];
            top = top-1;
        end
    end
end
```

```
% count
%% 图的压缩矩阵与邻接矩阵的转换，是DFS的子函数
function A = compresstable2matrix(b, Tag)
%label是和Tag同维度的矩阵，可以达到的位置是0，不能达到的位置是1
[n, ~] = size(b);
[map_x, map_y] = size(Tag);
m = max(b(:));
A = zeros(m,m);
for i = 1:n
    A(b(i,1),b(i,2)) = 1;
    A(b(i,2),b(i,1)) = 1;
end
%% 将标记矩阵转换为图的压缩矩阵
function b = graph_convert(Tag)
[map_x, map_y] = size(Tag);
b = [];
%遍历每行每列找到零元素所在位置，将相邻的非零元素连起来
for i = 1:map_x
    index = find(Tag(i,:) == 0);
    for j = 1:length(index) - 1
        if ((index(j+1) - index(j) == 1))
            % i表示点所在的行数，index(j)表示点所在的列数
            b = [b;[index(j)+(i-1)*map_y index(j+1)+(i-1)*map_y]];
        else
            continue
        end
    end
end
for i = 1:map_y
    index = find(Tag(:,i) == 0);
    for j = 1:length(index) - 1
        if (index(j+1) - index(j) == 1)
            % i表示点所在的列数，index(j)表示点所在的行数
            b = [b;[i+(index(j)-1)*map_y i+(index(j+1)-1)*map_y]];
        else continue
        end
    end
end
b = sortrows(b, 1);
%% 规划路径结果展示
function result_display(re, Tag)
[map_x, map_y] = size(Tag);
figure(1)
[xData, yData] = netplot(Tag);

%生成栅格编号和地图位置之间的换算矩阵
conversion_matrix = reshape((1:map_x * map_y), map_y, map_x);
```

```
conversion_matrix = conversion_matrix';

%模拟机器人的运动路径，扫过的格子用蓝色表示，重复清扫的格子用红色表示
xx = size(re, 1);
x_temp = [0 1 1 0];
y_temp = [0 0 1 1];
patch('xData', x_temp, 'yData', y_temp, 'FaceColor', 'b');
pause_time = 0.05; %运行速率，可以调整
pause(pause_time)
sign = zeros(map_x, map_y);
sign(1, 1) = 1;
for i = 1:xx
    %   temp1 = re(i, :);
    temp2 = re(i, :);
    [xco, yco] = find(conversion_matrix == temp2(1));
    [xco1, yco1] = find(conversion_matrix == temp2(2));
    x_temp = [0 1 1 0] + xco -1;
    y_temp = [0 0 1 1] + yco -1;
    x1_temp = [0 1 1 0] + xco1 -1;
    y1_temp = [0 0 1 1] + yco1 -1;
    if sign(xco1, yco1) == 0 %表示按深度优先顺序走下去
        sign(xco1, yco1) = 1;
        patch('xData', x1_temp, 'yData', y1_temp, 'FaceColor', 'b');
        pause(pause_time)
    else %表示回溯过程
        patch('xData', x1_temp, 'yData', y1_temp, 'FaceColor', 'r');
        pause(pause_time)
    end
end

%% 随机碰撞结果展示
function step_num_random = random_display(Tag, node_num)
%机器人从坐标原点开始清扫，机器人不具有记忆功能
%默认清扫方向是向右，向上，向左，向下
%最大运行次数为深度优先搜索规划路径的运行次数
step_num_random = 0;
sweep_order = [1 0;0 1;-1 0;0 -1];%分别表示向右，向上，向左，向下
figure(2)
[xData, yData] = netplot(Tag);
sign = Tag;
%未清扫的区域为白色，清扫过一次的区域为蓝色，重复清扫过的区域为红色
[map_x, map_y] = size(Tag);
xx = [0 1 1 0];
yy = [0 0 1 1];
patch('xData', xx, 'yData', yy, 'FaceColor', 'b');
pause_time = 0.01;
pause(pause_time)
```

```
sign(1, 1) = 1;
index = [1 1];
% for i = 1:max_step
step_num_random = 1;
count = 1;
while count < node_num
    for j = 1:4
        time = ceil(rand * 4);
        dir = sweep_order(time, :);
        if index(1)+dir(1)<1 || index(1)+dir(1)>map_x ||...
                index(2)+dir(2)<1 || index(2)+dir(2)>map_y
            continue
        end
        if Tag(index(1)+dir(1), index(2)+dir(2)) == 1
            continue
        end
        index = index + dir;
        x_temp = [0 1 1 0] +index(1) - 1;
        y_temp = [0 0 1 1] + index(2) - 1;
        if sign(index(1), index(2)) == 0
            patch('xData', x_temp, 'yData', y_temp, 'FaceColor', 'b');
            pause(pause_time)
            sign(index(1), index(2)) = 1;
            step_num_random = step_num_random + 1;
            count = count + 1;
            break
        else
            patch('xData', x_temp, 'yData', y_temp, 'FaceColor', 'r');
            pause(pause_time)
            step_num_random = step_num_random + 1;
            break
        end
    end
end


%% 绘制地图以及障碍物的位置
function [xData, yData] = netplot(Tag)
[map_x, map_y] = size(Tag);
x = 0:map_x;
y = 0:map_y;
x1 = [x(1) x(end)]';
y1 = [y(1) y(end)]';
%产生所有网格线的数据xData,yData
x2 = repmat(x1, 1, length(y)-2);
x3 = repmat(x(2) : x(end-1), 2, 1);
xData = [x2 x3];
```

```
y2 = repmat(y1, 1, length(x)-2);
y3 = repmat(y(2):y(end-1), 2, 1);
yData = [y3 y2];
h = line(xData, yData);
box on;
set(h, 'Color', 'k');
%绘出障碍物的位置，用黑色区域表示
[co_x, co_y] = find(Tag == 1);
% [co_x co_y]
for i = 1:length(co_x)
    x = [0 1 1 0] + co_x(i) -1;
    y = [0 0 1 1] + co_y(i) -1;
    patch('xData', x, 'yData', y, 'FaceColor', 'k');
end
```