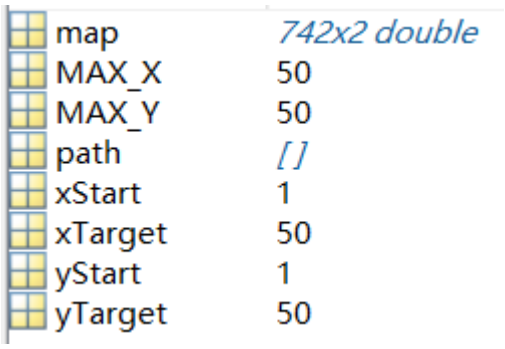
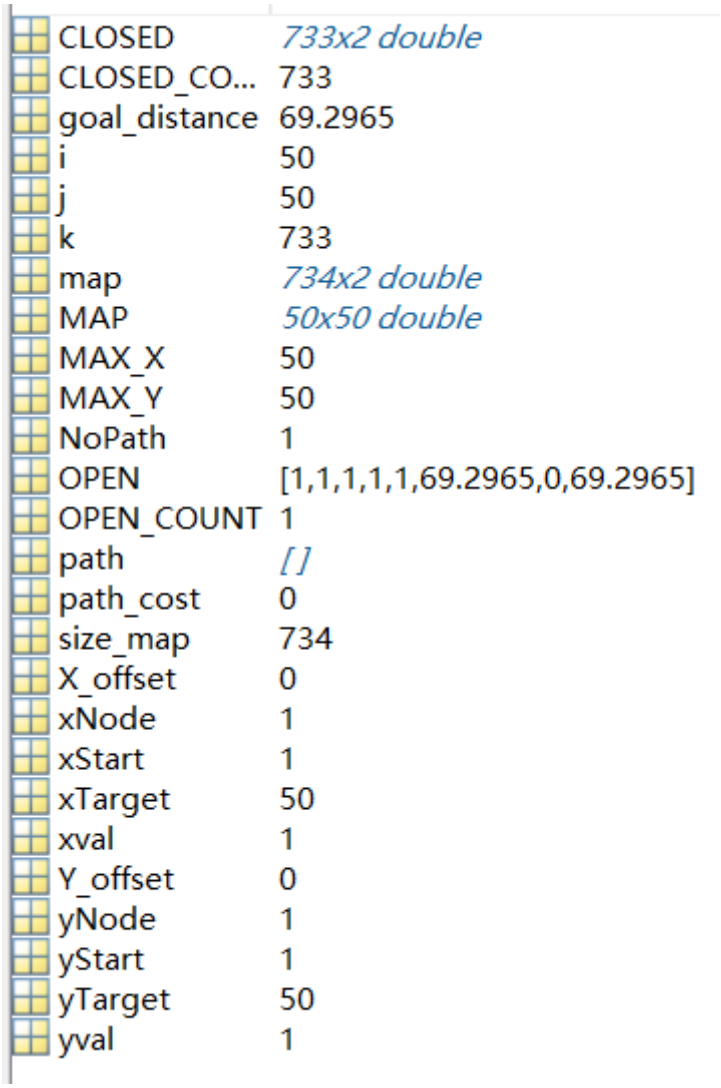
main.m

- 变量：
  - 

  - map ((n+2)*2 double): 用于储存起点，n个障碍物点和终点的坐标
- 算法流程：
  - 调用obstacle_map生成map
  - 调用A_star_search生成path **（todo）**
  - 可视化：visualize_map(map, path, [])

A_star_search.m框架

- 变量：
  - 

- MAP：栅格热图，起点，终点，障碍物点和free点上分别具有不同的值
- CLOSED：close list：用于存放障碍物和已经visit过的节点
- CLOSED_COUNT = len（CLOSED）
- OPEN：open list：用于存放expand到的点，其数据结构如下：

```
%OPEN LIST STRUCTURE
%-------------------------------------------------------------------------
%IS ON LIST 1/0 |X val |Y val |Parent X val |Parent Y val |h(n) |g(n)|f(n)|
%-------------------------------------------------------------------------
OPEN=[];
```

  - 其中第一位为flag位，当点被弹出open list后不用将其从open list中删除，只要将该flag位置为0即可，这样做的好处是可以记录所有expand过的节点
  - **从逻辑上讲，算法中的container等效于该OPEN中所有flag位=1的点的集合**
- OPEN_COUNT = len（OPEN）
- xStart，yStart存放起始点坐标
- xTarget，yTarget存放目标点坐标
- xNode，yNode存放当前visit点的坐标（流动）
- goal_distance：当前node到目标点的距离，即为Heuristic，可以通过修改distance.m采用不同的启发函数（默认为欧氏距离）
- path_cost：从父节点到该点走过的距离

- 算法流程：
  - 对2D grid map进行初始化：

```
%%
%This part is about map/obstacle/and other settings
    %pre-process the grid map, add offset
    size_map = size(map,1);
    Y_offset = 0;
    X_offset = 0;

    %Define the 2D grid map array.
    %Obstacle=-1, Target = 0, Start=1, Free=2
    MAP=2*(ones(MAX_X,MAX_Y));  % 先全部设为free

    %Initialize MAP with location of the target
    xval=floor(map(size_map, 1)) + X_offset;
    yval=floor(map(size_map, 2)) + Y_offset;    % 取map的最后一个元素，即目标点
    xTarget=xval;
    yTarget=yval;    % 暂存目标点坐标
    MAP(xval,yval)=0;  % 目标点设为0

    %Initialize MAP with location of the obstacle
    for i = 2: size_map-1
        xval=floor(map(i, 1)) + X_offset;
        yval=floor(map(i, 2)) + Y_offset;
        MAP(xval,yval)=-1;    % 障碍物点设为-1
    end

    %Initialize MAP with location of the start point
    xval=floor(map(1, 1)) + X_offset;
    yval=floor(map(1, 2)) + Y_offset;    % 取map的第一个元素，即起点
    xStart=xval;
    yStart=yval;  % 暂存起点坐标
    MAP(xval,yval)=1;    % 起点设为1
```

  - 定义OPEN LIST：

```
%OPEN LIST STRUCTURE
%-------------------------------------------------------------------------
%IS ON LIST 1/0 |X val |Y val |Parent X val |Parent Y val |h(n) |g(n)|f(n)|
%-------------------------------------------------------------------------
OPEN=[];
```

- 定义CLOSE LIST：

```
%CLOSED LIST STRUCTURE
%--------------
%X val | Y val |
%--------------
% CLOSED=zeros(MAX_VAL,2);
CLOSED=[];
```

- 将所有障碍物置入CLOSE LIST：

```
k=1;%Dummy counter
for i=1:MAX_X
    for j=1:MAX_Y
        if(MAP(i,j) == -1)
            CLOSED(k,1)=i;
            CLOSED(k,2)=j;
            k=k+1;
        end
    end
end
CLOSED_COUNT=size(CLOSED,1);
```

- **注意：CLOSE LIST和OPEN LIST每次update后要及时update他们的COUNT**
- 初始化：visit起始点

```
%set the starting node as the first node (一些初始化)
xNode=xStart;
yNode=yStart;   % 当前访问节点
OPEN_COUNT=1;   % open list的长度
goal_distance=distance(xNode,yNode,xTarget,yTarget);   % 当前点到目标点的欧拉距离（即h(n)的值）
path_cost=0;   % 从父节点到该点走过的距离
% 将当前节点加入open list，open list中每个节点的数据结构为
% （bool(是否在close list中),该点x,该点y,父节点x,父节点y,g值,h值,f=g+h）
OPEN(OPEN_COUNT,:)=insert_open(xNode,yNode,xNode,yNode,goal_distance,path_cost);
OPEN(OPEN_COUNT,1)=0; % 该点已经visit，即可将flag位置为0，同时将其加入close list
% 将该点加入close list
CLOSED_COUNT=CLOSED_COUNT+1;
CLOSED(CLOSED_COUNT,1)=xNode;
CLOSED(CLOSED_COUNT,2)=yNode;
OPEN(OPEN_COUNT,1)=1; % 初始化一个包含初始点的container
% 路径的序号
NoPath=1;
```

可用封装工具

- dist = distance(x1,y1,x2,y2)：计算两点的欧氏距离
- n_index = node_index(OPEN,xval,yval)：根据坐标xval，yval在OPEN list中找到该点的索引
- new_row = insert_open(xval,yval,parent_xval,parent_yval,hn,gn)：扩展OPEN list

```
new_row=[1,8];
new_row(1,1)=1;
new_row(1,2)=xval;
new_row(1,3)=yval;
new_row(1,4)=parent_xval;
new_row(1,5)=parent_yval;
new_row(1,6)=hn;
new_row(1,7)=gn;
new_row(1,8)=hn+gn;
```

    - flag位默认置为1
    - f由h和g自动计算得出
- i_min = min_fn(OPEN,OPEN_COUNT,xTarget,yTarget)：找到container中f值最小的node
- exp_array=expand_array(node_x,node_y,gn,xTarget,yTarget,CLOSED,MAX_X,MAX_Y)：根据当前节点找到所有可扩展结点
    - 输出的数据格式：

```
%EXPANDED ARRAY FORMAT
%--------------------------------
%|X val |Y val ||h(n) |g(n)|f(n)|
%--------------------------------
```

    - ps：这里的gn是从当前节点扩展到该结点的gn和

```
exp_array(exp_count,4) = gn+distance(node_x,node_y,s_x,s_y);
```

A_star_search.m实现（todo）

- 逻辑根据伪代码进行即可
- 开始循环前初始化一个含有起点的container
- 开始循环：
    1. 退出条件：若container为空，退出循环
    2. 从container中弹出具有最小f的节点
        - 弹出的同时visit该节点
        - 已访问则将OPEN的flag位置为0
        - 将该点加入closelist
    3. 退出条件：若visit到目标点，退出循环
    4. expand节点，对所有expand到的neighbors检查是否已经在OPEN中
        - 若已经在，则比较g值，维护最小的g值
        - 若还不在，则把该neighbor加入OPEN中，g值存为现有的g值
    5. 下一循环
- 生成path：找到target后当前访问的node即为终点，沿终点逐一沿父节点回溯直到找到起始点