A 主函数

```
mapOriginal=im2bw(imread('map1.bmp')); % input map read from a bmp file. for new maps write the file name here %从 bmp 文件读取的输入映射。
对于新的地图，在这里写文件名
resolutionX=100;
resolutionY=100;
source=[10 10]; % source position in Y, X format   Y，X 格式的源位置
goal=[490 490]; % goal position in Y, X format  Y，X 格式的目标位置
conn=[1 1 1; % robot (marked as 2) can move up, left, right and down (all 1s), but not diagonally (all 0). you can increase/decrease the size of the matrix 机器人(标记为 2)
可以上、左、右、下移动(全为 1)，但不能沿对角线移动(全为 0)。您可以增加/减少矩阵的大小
  1 2 1;
  1 1 1];
%   conn=[1 1 1 1 1; % another option of conn conn 的另一个选择
%         1 1 1 1 1;
%         1 1 2 1 1;
%         1 1 1 1 1
%         1 1 1 1 1];

%   conn=[0 1 0; % another option of conn conn 的另一个选择
%         1 2 1;
%         0 1 0];

display=true; % display processing of nodes 节点的显示处理
```

%%%%% parameters end here %%%%% %%%%%参数在此处结束%%%%%

mapResized=imresize(mapOriginal,[resolutionX resolutionY]);

这行代码将原始地图图像 mapOriginal 缩放到指定的分辨率 resolutionX 和 resolutionY。这是必要的，因为 A*算法需要一个固定大小的地图来进行计算。B = imresize(A, m)

返回的图像 B 的长宽是图像 A 的长宽的 m 倍，即缩放图像。 m 大于 1， 则放大图像； m 小于 1， 缩小图像。

B = imresize(A, [numrows numcols])

numrows 和 numcols 分别指定目标图像的高度和宽度。

显而易见，由于这种格式允许图像缩放后长宽比例和源图像长宽比例不相同，因此所产生的图像有可能发生畸变。

[Y newmap] = imresize(X, map, scale)

[...] = imresize(..., method)

method 参数用于指定在改变图像尺寸时所使用的算法，可以为以下几种：

'nearest'： 这个参数也是默认的， 即改变图像尺寸时采用最近邻插值算法；

'bilinear'：采用双线性插值算法；

'bicubic'： 采用双三次插值算法，在 R2013a 版本里，默认为这种算法，所以不同版本可能有不同的默认参数，使用之前建议使用命令 help imresize 获得帮助信息，以帮助信息为准；

[...] = imresize(..., parameter, value,...)

map=mapResized; % grow boundary by a unit pixel 将边界增加一个单位像素

for i=1:size(mapResized,1)

for j=1:size(mapResized,2)for 循环遍历 mapResized 的每一个像素，如果一个像素是障碍物（在二值图像中用 0 表示），则代码会将这个障碍物的周围 8 个像素也标记为障碍物。

这样做的目的是为了确保机器人在路径搜索时不会太靠近障碍物，从而增加了路径搜索的安全性。

if mapResized(i,j)==0 在 if 语句中，代码检查每个障碍物像素周围的 8 个像素，并将它们也标记为障碍物（即设置为 0），这些操作确保了障碍物周围至少有一个像素的安全距离，这样机器人在规划路径时就不会考虑这些危险区域。通过这种方式，代码为机器人的路径搜索创建了一个更加安全的地图环境。

这样做的主要目的是为了确保机器人在实际应用中的安全性和可靠性。在现实世界的路径规划问题中，机器人需要避免与障碍物发生碰撞，而这通常要求在机器人路径和障碍物之间留出一定的安全距离。这个安全距离可以防止由于定位误差、机器人尺寸、动态变化等因素导致的潜在碰撞。

在路径规划算法中，如果不考虑这个安全距离，算法可能会规划出过于接近障碍物的路径，这在实际执行时可能会导致机器人碰撞到障碍物。通过在地图上扩展障碍物的边界，算法就会考虑这个安全距离，从而规划出更加安全的路径。

具体来说，这样做的几个原因包括：

机器人尺寸：机器人的实际尺寸可能比地图上的单个像素点大，因此需要在实际障碍物周围留出更多的空间以确保机器人能够安全通过。

定位精度：机器人的定位系统可能存在一定的误差，因此在障碍物周围留出额外的空间可以提供一个误差容忍区域，减少碰撞的风险。

动态障碍物：在动态环境中，障碍物可能会移动，留出安全距离可以提供额外的反应时间，使得机器人能够在障碍物移动到规划的路径上时做出调整。

算法简化：通过扩展障碍物边界，可以简化路径规划算法的实现，因为算法不需要处理复杂的障碍物形状和边缘情况。

传感器和动力学限制：机器人的传感器（如激光雷达、超声波传感器等）和动力系统（如驱动轮、转向系统等）可能有一定的限制，需要更宽的通道来保证机器人的稳定运行。

if 语句是用来处理地图上的障碍物像素，并将它们周围相邻的像素也标记为障碍物。这是通过将那些像素的值设置为 0 来实现的，因为在二值图像中，0 通常代表黑色或障碍物。

if i-1>=1, map(i-1,j)=0; end 这行代码检查当前像素上方（即 i-1 行）的像素是否在地图范围内（即 i-1 大于等于 1）。如果是，则将该上方像素的值设置为 0，将其标记为障碍物。

if j-1>=1, map(i,j-1)=0; end 这行代码检查当前像素左侧（即 j-1 列）的像素是否在地图范围内（即 j-1 大于等于 1）。如果是，则将该左侧像素的值设置为 0，将其标记为障碍物。

if i+1<=size(map,1), map(i+1,j)=0; end 这行代码检查当前像素下方（即 i+1 行）的像素是否在地图范围内（即 i+1 小于等于地图的行数）。如果是，则将该下方像素的值设置为 0，将其标记为障碍物。

if j+1<=size(map,2), map(i,j+1)=0; end 这行代码检查当前像素右侧（即 j+1 列）的像素是否在地图范围内（即 j+1 小于等于地图的列数）。如果是，则将该右侧像素的值设置为 0，将其标记为障碍物。

if i-1>=1 && j-1>=1, map(i-1,j-1)=0; end 这行代码检查当前像素左上方的像素是否在地图范围内（即 i-1 大于等于 1 且 j-1 大于等于 1）。如果是，则将该左上方像素的值设置为 0，将其标记为障碍物。

if i-1>=1 && j+1<=size(map,2), map(i-1,j+1)=0; end 这行代码检查当前像素右上方的像素是否在地图范围内（即 i-1 大于等于 1 且 j+1 小于等于地图的列数）。如果是，则将该右上方的像素的值设置为 0，将其标记为障碍物。

if i+1<=size(map,1) && j-1>=1, map(i+1,j-1)=0; end 这行代码检查当前像素左下方的像素是否在地图范围内（即 i+1 小于等于地图的行数且 j-1 大于等于 1）。如果是，则将该左下方像素的值设置为 0，将其标记为障碍物。

if i+1<=size(map,1) && j+1<=size(map,2), map(i+1,j+1)=0; end 这行代码检查当前像素右下方的像素是否在地图范围内（即 i+1 小于等于地图的行数且 j+1 小于等于地图的列数）。如果是，则将该右下方像素的值设置为 0，将其标记为障碍物。

```
        end
    end
end
source=double(int32((source.*[resolutionX
resolutionY])./size(mapOriginal)));
```

这行代码将起始点 source 的坐标从原始地图的分辨率转换为缩放后地图的

分辨率。首先，它将起始点的坐标乘以缩放因子（resolutionX 和 resolutionY），然后除以原始地图的大小（size(mapOriginal)），最后将结果转换为 double 类型。

goal=double(int32((goal.*[resolutionX resolutionY])./size(mapOriginal)));这行代码与上一行类似，但是用于目标点 goal 的坐标。它也将目标点的坐标从原始地图的分辨率转换为缩放后地图的分辨率。

if ~feasiblePoint(source,map), error('source lies on an obstacle or outside map'); end 这行代码检查起始点 source 是否在缩放后的地图上是一个可行的点，即它不在障碍物上也不在地图外面。如果 feasiblePoint 函数返回 false，表示起始点不可行，则使用 error 函数抛出一个错误。

if ~feasiblePoint(goal,map), error('goal lies on an obstacle or outside map'); end 这行代码检查目标点 goal 是否在缩放后的地图上是一个可行的点。如果 feasiblePoint 函数返回 false，表示目标点不可行，则使用 error 函数抛出一个错误。

if length(find(conn==2))~=1, error('no robot specified in connection matrix'); end 这行代码检查连接矩阵 conn 中是否恰好有一个值为 2 的元素，这代表机器人的位置。如果 find 函数返回的数组长度不是 1，则表示没有正确指定机器人位置，使用 error 函数抛出一个错误。

%structure of a node is taken as positionY, positionX, historic cost, heuristic cost, total cost, parent index in closed list (-1 for source) % 节点的结构被定义为 positionY, positionX, 历史成本, 启发式成本, 总成本, 闭列表中的父索引（对于源节点为-1）

Q=[source 0 heuristic(source,goal) 0+heuristic(source,goal) -1]; % the processing queue of A* algorihtm, open list A*算法的处理队列，开列表 这行代码初始化 A*算法的开列表 Q。它将起始点 source 作为第一个节点添加到列表中，并设置其历史成本为 0（因为它是起始点），启发式成本为 heuristic(source,goal)（从起始点到目标的估计成本），总成本为启发式成本

（因为历史成本为 0），父索引为-1（表示没有父节点）。

closed=ones(size(map)); % the closed list taken as a hash map. 1=not visited, 0=visited 闭列表作为哈希表。1=未访问，0=已访问 这行代码初始化 A*算法的闭列表 closed。它创建一个与地图大小相同的矩阵，并用 1 填充，表示所有点都未访问过。

closedList=[]; % the closed list taken as a list 闭列表作为列表 这行代码初始化一个用于存储闭列表的空数组 closedList。这个列表将用于存储被访问过的节点。

pathFound=false;这行代码初始化一个布尔变量 pathFound，用于指示是否找到了路径。初始设置为 false，表示尚未找到路径。

A*算法的主体部分，它通过一个循环来查找从起始点到目标的路径

tic;这行代码开始计时，tic 函数是 MATLAB 中的计时函数，它会在循环结束时自动调用 toc 函数来计算循环的总运行时间。

counter=0;这行代码初始化一个计数器变量 counter，用于记录循环执行的次数。

colormap(gray(256));这行代码设置颜色映射为灰度，以便在显示图像时使用。

while size(Q,1)>0 这是一个循环，只要开列表 Q 中的元素数量大于 0，循环就会继续执行。

[A, I]=min(Q,[],1);

n=Q(I(5),:); % smallest cost element to process

Q=[Q(1:I(5)-1,:);Q(I(5)+1:end,:)]; % delete element under processing 这行代码从开列表 Q 中找到具有最小成本的元素，并将其从列表中删除。min 函数返回最小成本元素的位置 I，然后使用 Q(I(5),:)获取该元素，并将其从 Q 中移除。

if n(1)==goal(1) && n(2)==goal(2) % goal test

pathFound=true;break;

end 这行代码检查当前处理的节点是否是目标点。如果是，则设置 pathFound 为 true，并使用 break 语句退出循环。

[rx,ry,rv]=find(conn==2); % robot position at the connection matrix

[mx,my,mv]=find(conn==1); % array of possible moves 这行代码使用 find 函数从连接矩阵 conn 中找到机器人的位置（值为 2）和所有可能的移动方向（值为 1）

A*算法中的关键部分，它负责扩展当前节点 n 的邻近节点，并决定是否将它们添加到开列表 Q 中。

for mxi=1:size(mx,1) %iterate through all moves 这行代码开始一个循环，它会遍历所有可能的移动方向，这些方向存储在数组 mx、my 和 mv 中，这些数组是在之前的代码中通过 find 函数从连接矩阵 conn 中获取的。

newPos=[n(1)+mx(mxi)-rx n(2)+my(mxi)-ry]; % possible new node 这行代码计算从当前节点 n 移动到新位置 newPos 的坐标。它通过将当前节点的坐标与可能的移动方向相加来得到新位置的坐标。

if checkPath(n(1:2),newPos,map) %if path from n to newPos is collission-free 这行代码检查从当前节点 n 到新位置 newPos 的路径是否安全，即没有障碍物。如果 checkPath 函数返回 true，表示路径是安全的，代码将继续执行。

if closed(newPos(1),newPos(2))~=0 % not already in closed 这行代码检查新位置 newPos 是否已经在闭列表 closed 中。如果 newPos 的坐标值在 closed 中不为 0（通常表示未访问），那么新位置没有被访问过，代码将继续执行。

historicCost=n(3)+historic(n(1:2),newPos);

heuristicCost=heuristic(newPos,goal);

totalCost=historicCost+heuristicCost;这行代码计算新位置的历史成本 historicCost，它是从起始点到当前节点的成本加上从当前节点到新位置的成本。它还计算新位置的启发式成本 heuristicCost，这是从新位置到目标的估计成本。然后，它计算总成本 totalCost，这是历史成本和启发式成本的总和。

add=true; % not already in queue with better cost 这行代码设置一个标志 add 为 true，表示新位置应该被添加到开列表 Q 中，因为它不在闭列表中，且具有更好的成本。

if length(find((Q(:,1)==newPos(1)) .* (Q(:,2)==newPos(2))))>=1 这行代码检查开列表 Q 中是否已经存在具有相同坐标的新位置。如果存在，它将使用 find 函数找到该位置，并将其坐标存储在变量 I 中。

I=find((Q(:,1)==newPos(1)) .* (Q(:,2)==newPos(2)));

if Q(I,5)<totalCost, add=false; 这行代码检查开列表 Q 中具有相同坐标的新位置的成本是否小于新位置的总成本。如果成本较低,则设置 add 为 false，表示新位置不应该被添加到开列表中，因为它不会带来更好的路径。

else Q=[Q(1:I-1,:);Q(I+1:end,:);];add=true; 如果新位置的成本不小于开列表中已有位置的成本，它将从开列表中移除这个位置，并将新位置添加到列表的末尾。

end

end

if add 这行代码检查 add 标志是否为 true

Q=[Q;newPos historicCost heuristicCost totalCost size(closedList,1)+1]; % add new nodes in queue 如果 add 标志为 true，表示新位置应该被添加到开列表 Q 中，代码将新位置的坐标、历史成本、启发式成本、总成本以及它在闭列表中的索引添加到开列表 Q 的末尾。

end

end

end

end

closed(n(1),n(2))=0;closedList=[closedList;n]; % update closed lists 这行代码更新闭列表 closed，将当前节点 n 标记为已访问（设置为 0），并将当前节点添加到闭列表 closedList 中。

if display

image((map==0).*0 + ((closed==0).*(map==1)).*125 + ((closed==1).*(map==1)).*255);

counter=counter+1;

M(counter)=getframe;

end 这行代码检查是否启用了显示功能。如果启用了显示，它会显示当前的地图，其中未访问的节点显示为浅灰色，已访问的节点显示为深灰色。同时，它会增加计数器 counter 的值，并获取当前帧，用于动画显示。

end 这行代码结束 while 循环，当开列表 Q 中的元素数量减少到 0 时，循环结束。

```
if ~pathFound
    error('no path found')
```

end 这行代码检查是否找到了路径。如果 pathFound 为 false，表示没有找到路径，则使用 error 函数抛出一个错误。

```
if display
    disp('click/press any key');
    waitforbuttonpress;
End
```

这行代码检查是否启用了显示功能。如果启用了显示，它会等待用户点击或按键来结束显示。

这段代码是 A*算法找到路径后的处理部分，它从闭列表中提取路径，并计算路径的长度，然后显示处理时间和路径。

```
path=[n(1:2)]; %retrieve path from parent information
```

这行代码从闭列表 closedList 中提取当前节点的坐标作为路径的第一个节点。它使用父节点索引 n(6)来访问闭列表中的对应节点，并提取其坐标 n(1:2)。

```
prev=n(6);
```

这行代码将父节点索引 n(6)赋值给变量 prev，这个索引指向闭列表中的父节点。

```
while prev>0
```

这行代码开始一个循环，它会遍历闭列表 closedList 中的所有节点，直到找到源节点为止。

```
path=[closedList(prev,1:2);path];
```

这行代码将闭列表中父节点的坐标添加到路径的末尾。它使用父节点索引 prev 来访问闭列表中的对应节点，并提取其坐标 closedList(prev,1:2)。

```
prev=closedList(prev,6);
```

这行代码更新父节点索引 prev，使其指向闭列表中的下一个父节点。

End

这行代码结束循环，当 prev 变为 0 时，循环结束，这意味着路径到达了源节点。

path=[(path(:,1)*size(mapOriginal,1))/resolutionX

(path(:,2)*size(mapOriginal,2))/resolutionY];

这行代码将路径的坐标从缩放后的地图分辨率转换回原始地图的分辨率。它通过将路径的每个坐标乘以缩放因子（size(mapOriginal,1) 和 size(mapOriginal,2)）并除以分辨率（resolutionX 和 resolutionY）来实现。

pathLength=0;fori=1:length(path)-1,

pathLength=pathLength+historic(path(i,:),path(i+1,:)); end

这行代码计算路径的长度。它通过遍历路径中的每个相邻节点对，并使用 historic 函数计算它们之间的距离，然后累加这些距离来得到路径的总长度。

fprintf('processing time=%d \nPath Length=%d \n\n',

toc,pathLength);

这行代码使用 fprintf 函数在控制台输出处理时间和路径长度。它使用 toc 函数计算从 tic 开始的处理时间，并将这两个值格式化输出。

imshow(mapOriginal);这行代码显示原始地图图像。

rectangle('position',[1 1 size(mapOriginal)-1],'edgecolor','k')

这行代码在原始地图上绘制一个矩形，表示地图的边界。

line(path(:,2),path(:,1));

这行代码在原始地图上绘制路径。它使用 path 数组中的 Y 坐标作为 x 值，X 坐标作为 y 值来绘制路径。