A 子函数

这个函数 checkPath 是用来检查从给定的节点 n 到新位置 newPos 的路径是否可行，即是否没有碰撞。如果路径是可行的，它将返回 true；否则，返回 false。以下是函数的逐行解释：

function feasible=checkPath(n,newPos,map)

这行代码定义了一个函数，该函数接受三个输入参数：n（当前节点坐标）、newPos（新位置坐标）和 map（地图矩阵）。

feasible=true

这行代码初始化一个布尔变量 feasible 为 true，表示假设路径是可行的。

dir=atan2(newPos(1)-n(1),newPos(2)-n(2))

这行代码计算从当前节点 n 到新位置 newPos 的方向角度。atan2 函数返回的是从正 X 轴到这个点的角度。

for r=0:0.5:sqrt(sum((n-newPos).^2))

这行代码开始一个循环，它会遍历从当前节点到新位置的路径上的所有点。r 是当前点到新位置的距离。

posCheck=n+r.*[sin(dir) cos(dir)]

这行代码计算当前点到新位置的路径上的一个点。它使用 sin 和 cos 函数来计算方向角度 dir 的正弦和余弦值，并将它们与当前点到新位置的距离 r 相乘，然后加上当前节点 n 的坐标。

if ~(feasiblePoint(ceil(posCheck),map) && feasiblePoint(floor(posCheck),map) && ...

feasiblePoint([ceil(posCheck(1)) floor(posCheck(2))],map)

```
&& feasiblePoint([floor(posCheck(1)) ceil(posCheck(2))],map))
        feasible=false;break;
    End
```

这行代码检查当前路径上的点是否可行。它使用了 feasiblePoint 函数来检查点是否在地图上且不是障碍物。如果任何点不可行，它将设置 feasible 为 false，并使用 break 语句退出循环。

```
if ~feasiblePoint(newPos,map), feasible=false; end
```

这行代码检查新位置 newPos 是否可行。如果不可行，它将设置 feasible 为 false。

End

这段代码定义了一个函数 feasiblePoint，用于检查一个给定的点是否在地图的边界内且不是障碍物。如果点是可行的，函数返回 true；否则，返回 false。

```
function feasible=feasiblePoint(point,map)
```

这行代码定义了一个函数，该函数接受两个输入参数：point（要检查的点的坐标）和 map（地图矩阵）。

```
feasible=true
```

这行代码初始化一个布尔变量 feasible 为 true，表示假设点是可行的。

```
% check if collission-free spot and inside maps
if  ~(point(1)>=1  &&  point(1)<=size(map,1)  &&  point(2)>=1  &&
point(2)<=size(map,2) && map(point(1),point(2))==1)
```

这行代码是一个条件语句，它检查点 point 是否在地图的边界内且不是障碍物。它首先检查点 point(1)是否在地图的行数 size(map,1)范围内（即是否大于

等于 1 且小于等于 size(map,1)），然后检查点 point(2)是否在地图的列数 size(map,2)范围内（即是否大于等于 1 且小于等于 size(map,2)）。最后，它检查地图矩阵 map 在点 point(1)和 point(2)处的值是否为 1，如果是，表示点不是障碍物。如果这些条件不满足，即点不在地图范围内或者在障碍物上，它将设置 feasible 为 false。

```
feasible=false
```

这行代码将 feasible 变量设置为 false，表示点是不可行的。

```
End
```

这段代码定义了一个函数 heuristic，用于计算从节点 X 到目标点 goal 的启发式成本。启发式成本通常基于某种估计，用于 A*算法中的启发式搜索。在这个例子中，它计算欧几里得距离作为启发式成本。

```
function h=heuristic(X,goal)
```

这行代码定义了一个函数，该函数接受两个输入参数：X（当前节点坐标）和 goal（目标节点坐标）。

```
h = sqrt(sum((X-goal).^2))
```

这行代码计算从节点 X 到目标点 goal 的欧几里得距离，并将其作为启发式成本。它首先计算 X 和 goal 之间的坐标差(X-goal)，然后将这个差值的平方求和，最后取平方根得到欧几里得距离。这个距离的平方根就是启发式成本 h。

这段代码定义了一个函数 historic，用于计算两个节点之间的历史成本。历史成本通常是在 A*算法中用来表示从起始点到当前节点的累积成本。在这个例子中，它计算从节点 a 到节点 b 的欧几里得距离，并将其作为历史成本。

```
function h=historic(a,b)
```

这行代码定义了一个函数，该函数接受两个输入参数：a（起始节点坐标）和 b（当前节点坐标）。

```
h = sqrt(sum((a-b).^2))
```

这行代码计算从起始节点 a 到当前节点 b 的欧几里得距离，并将其作为历史成本。它首先计算 a 和 b 之间的坐标差(a-b)，然后将这个差值的平方求和，最后取平方根得到欧几里得距离。这个距离的平方根就是历史成本 h。