# 第3章 机器人被动式轨迹规划

## 3.1 引言

传统的被动路径规划主要基于安全性，路径长短，轨迹平滑这三方面优化路径**错误!未找到引用源。**，这三项在主动视觉路径规划中也是需要考虑的重要因素。这些被动的路径规划思想为主动视觉路径规划提供了基础框架。在被动式导航中，忽视了生成路径中各处感知质量对定位与建图的影响，导致建图算法的精度下降甚至失去定位。

在本文中，主动视觉路径规划基于A\*算法，该算法已经开源整合进入ROS的导航功能包。下面将对这一被动式的导航算法进行介绍，对它与视觉SLAM算法结合的自主探索过程不考虑感知质量而带来的问题做出分析。

## 3.2 机器人路径规划框架

机器人的路径规划一般采取全局规划和本地规划相结合的方式。两种规划器区别在于获取传感器的信息的时间空间范围不同。图3-1展示了ROS机器人操作系统中典型的机器人路径规划框架。

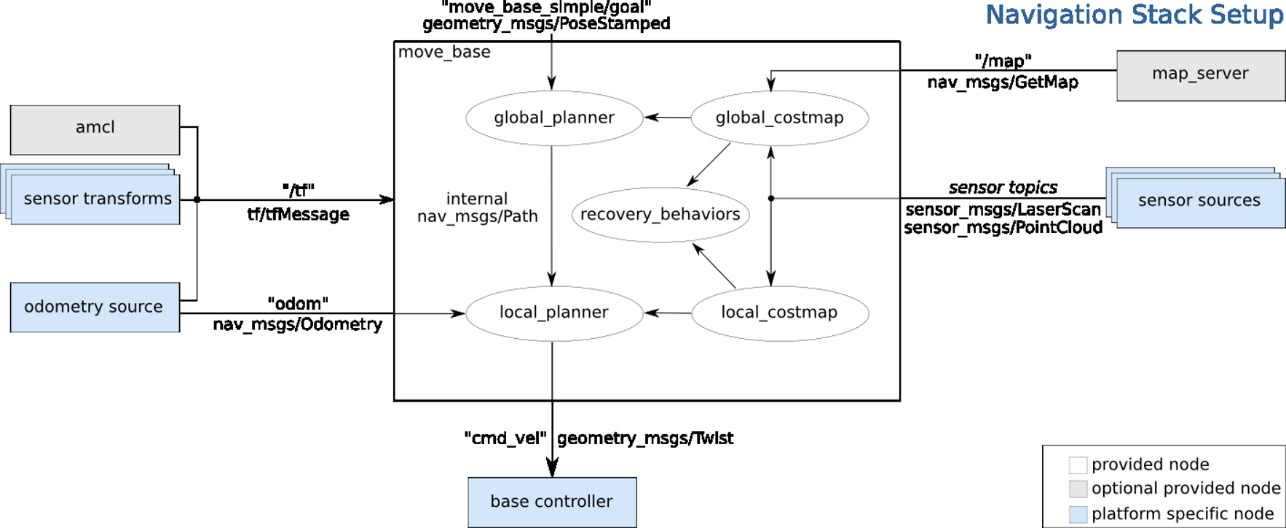


图 3-1机器人导航框架

其中白色模块表示算法路径规划中提供的节点，灰色模块表示可选的节点，淡蓝色模块表示移动机器人需要提供的部分。在进行路径规划时，需要输入的信息包括传感器感知信息。用户可以选择是否输入先验地图和机器人需要的定位节点生成的里程计。其中定位节点和地图服务器节点是可选的。RTAB-map也提供了用于定位的接口用于机器人的路径规划。

在路径规划的过程中，首先获取目标点，根据地图服务器提供的先验地图信息生成全局代价地图。全局规划器基于先验地图在目标和机器人当前位置之间生成全局路径。路径的生成不依赖于实时采集的环境信息，只利用已经建立好的地图寻找相对最优路径，所以无法处理机器人导航过程中的突发情况，比如遇到突然出现的不存在于先验地图上的障碍物，或者存在由于机器人机械结构问题无法按照全局路径通过的路段等。为了让机器人在导航过程中更加灵活的躲避障碍物，局部路径规划器基于局部代价地图对全局路径规划进行局部优化。由于局部代价地图是不断被更新的，因此可以通过局部代价地图进行局部路径生成。在给定的有限范围内，机器人按照局部路径躲避先验地图中没有被添加进去的不可达区域。虽然有局部路径的覆盖，但是机器人的总体行进路线应尽量贴近于给定的全局路径。在行进过程中，机器人通过搭载的相机或者雷达等装置不断感知环境信息，生成局部代价地图。行走的路人、突然驶过的车辆，以及其他在此区域工作的移动机器人均会被感知到，建立在局部地图中。这样的分时进行的路径规划完成后，节点输出实际的移动机器人速度信息（一般以话题的形式发布）。在速度信息发布之后，控制器进行解算并使底盘执行该运动。如果机器人陷入到障碍物区域无法脱离，将会通过一系列的恢复动作，逐步清除地图中机器人周围的障碍，并重新规划路径尝试移动离开此区域。

### 3.2.1代价地图（costmap）

RTAB-map生成的二维地图是无法直接被路径规划节点使用的。地图服务器提供给轨迹规划算法的地图信息为静态地图，静态地图在路径规划过程中存在以下两点问题：

1、静态地图只提供了建立地图时的障碍物信息，对于动态可变的障碍物信息无法试试更新到静态地图中。只依赖静态地图无法躲避导航过程中出现的未知障碍物。

2、静态地图仅表达地图中的障碍物确切的区域，使用静态地图进行路径规划时，难以控制机器人躲避障碍物的余量，机器人由于惯性，轮胎打滑，路面颠簸，机械故障等突发情况可能脱离规划出的路径导致碰撞。

为了解决以上两点问题，在静态地图的基础上添加辅助的地图信息。比如通过获取传感器信息实时更新的局部地图，对静态地图中的障碍物区域进行放大等。通过叠加辅助信息产生的具有实时更新功能且信息更丰富的地图称为代价地图**错误!未找到引用源。**。一个可视化的代价地图演示如图3-2所示。代价地图分为全局和局部两个尺度范围，分别用于两种规划器。这两张地图的构成基本相同，只是地图生成的区域有所不同。

向静态地图中添加辅助信息生成代价地图的过程通过叠加图层实现，一般常用的地图图层有：

Static Map Layer: 静态地图图层。该图层即由SLAM建立的静态地图，是静态不变的地图图层。该图层一般在机器人进行路径规划之前就已经存在内存当中。在进行全局路径规划时，规划器需要使用与传感器无关的先验地图信息生成全局路径，该地图由静态地图提供。（在机器人导航过程中同时进行slam建图时，允许更新静态地图）

Obstacles Layer:障碍物层。移动机器人在运行过程中，不断通过激光雷达等传感器获取周围障碍物信息，对于存在障碍物的区域，将地图中的栅格标记为占用。

Voxels Layer:与障碍物层的作用相同，区别在于该层使用三维传感器信息生成，可以反应多个高度上的障碍物信息。

Inflation Layer:膨胀层，对前面生成的图层中的致命障碍物区域进行膨胀，即在障碍物周围添加缓冲区。通过添加膨胀层，使得移动机器人在导航过程中不仅仅与障碍物避免发生碰撞，同时也不会与障碍物过于贴近。

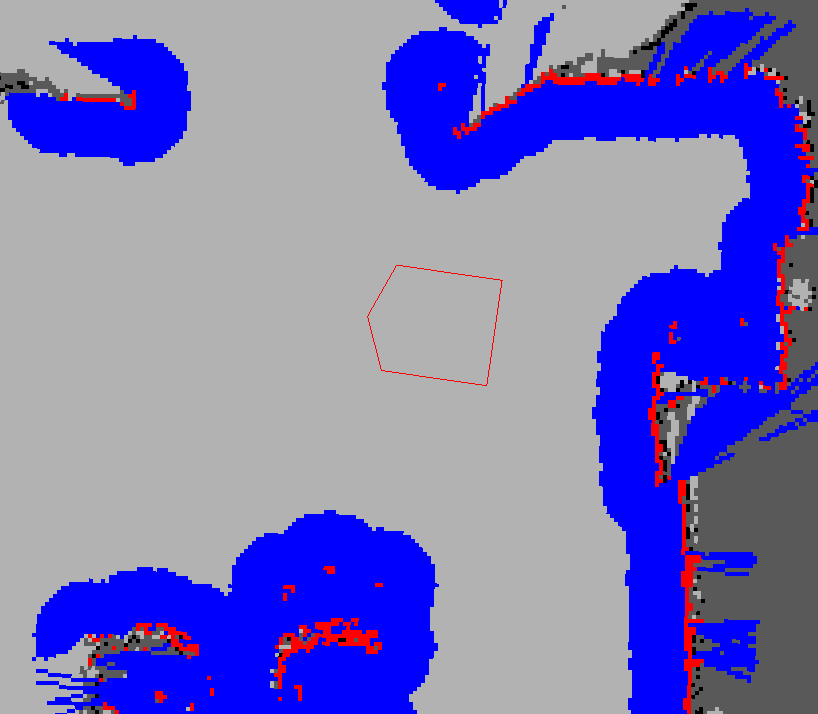


图 3-2 代价地图

在典型的代价地图如图3-2所示，各种颜色的栅格表示内容如下：

（1）红色——被占用的范围（障碍物）。绝对不允许进入。

（2）浅灰色——经过探测发现的可行的区域。

（3）深灰色——没有获取信息的未知区域。

（4）蓝色——将红色区域按照指定的比率放大之后的区域。

（5）五边形——表示移动机器人在地面上的投影轮廓。

使用这样约定的代价地图，投影轮廓不能与障碍物区域发生碰撞，五边形中心也不能和膨胀区域发生碰撞。

### 3.2.2恢复动作（ recovery behaviors）

如果机器人一旦陷入到按照当前规划路径无法脱身的区域，将会执行一系列的恢复动作尝试脱离此区域。图3-3展示了恢复动作的过程：

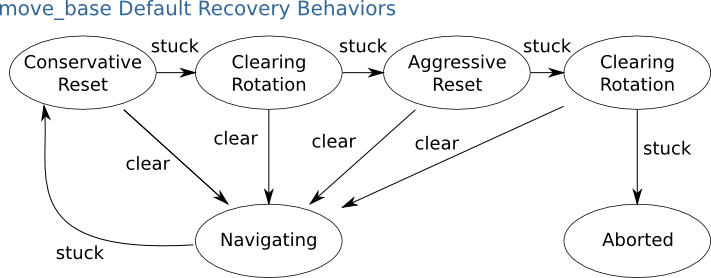


图 3-3 恢复动作流程

当机器在导航过程中感知到自己被卡住时，将会执行一系列操作以尝试清理周围空间：首先，恢复行为将从周围清理用户指定范围以外的障碍。如果仍未找到路径，将会在当前位置进行各个方向的搜索。这个过程中，算法试图清理周围的空间信息。如果这也失败了，恢复动作将进行更加激进的措施。这一环节中，按照上一步继续进行清理，但是机器人动作更加激进。在用户指定的长方形形空间内，机器人将再次朝各个方向转动以清理地图。如果上述的所有动作执行结束之后，算法仍然无法找到符合要求的道路，导航算法将认为目标不可达，退出导航并告知用户导航失败。如果上述的所有动作执行结束之后，机器人仍然无法找到可行的道路，导航算法将抛出异常。此时，算法认为不能到达用户指定点，退出导航并告知用户导航失败。

### 3.2.3 路径规划

机器人导航的路径规划问题主要分为全局路径规划和局部路径规划，这两者是根据对环境信息获取程度划分的。

全局规划通常需要在已知环境中进行，属于一种事前规划，可以找到最优解，一旦环境发生变化，未及时更新地图时，该方法就不能达到预期效果。

局部路径规划通常用在未知或部分已知的环境中，系统根据传感器实时获取到环境障碍物的信息，并做出相应规划，这对系统的实时计算处理能力有着较高的要求。由于缺乏全局环境信息，结果很可能不是最优的。

​ 全局路径规划和局部路径规划并没有本质的区别，很多适用于全局路径规划的方法经过改进也可以用于局部路径规划，而适用于局部路径规划的方法同样经过改进后也可适用于全局路径规划。两者协同工作，机器人可更好的规划从起始点到终点的行走路径。

#### 全局路径规划

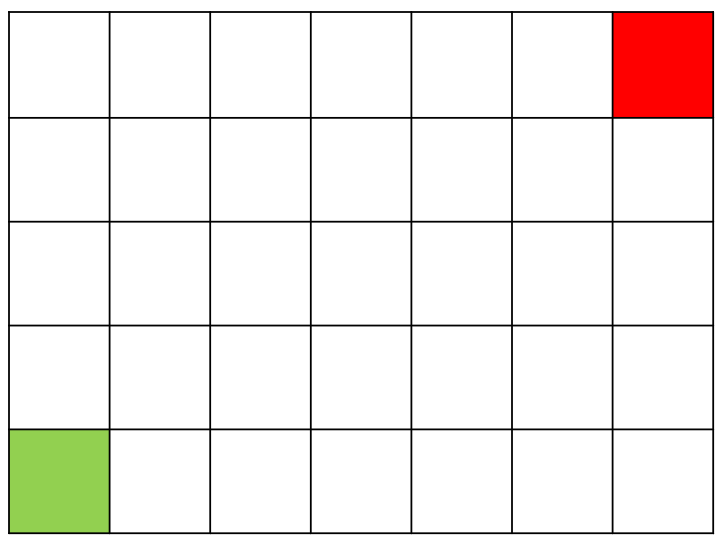
基本的全局路径规划算法分为基于图搜索的算法和基于采样的算法。

基于搜索的算法有广度优先算法，Dijkstra算法和A\*算法等

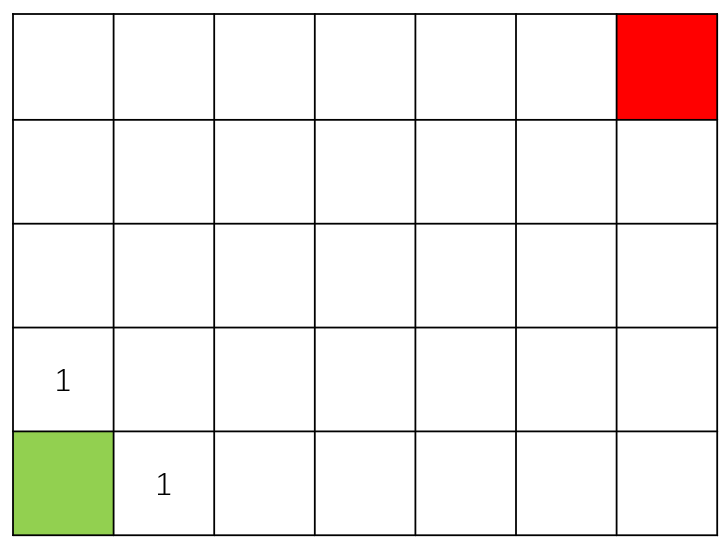
广度优先搜索算法(Breadth First Search,BFS)是最基本的图搜索算法之一，是Dijkstra、A\*等很多重要路径规划算法的原型。广度优先搜索在路径长度上具有最优性和完备性，即如果在起始点和目标点间有路径解存在，那么一定可以得到解，如果得不到解那么一定说明没有解存在，并且如果neng得到解，则该解在所有的可行解中一定是最短路径。

下面介绍广度优先算法的原理：

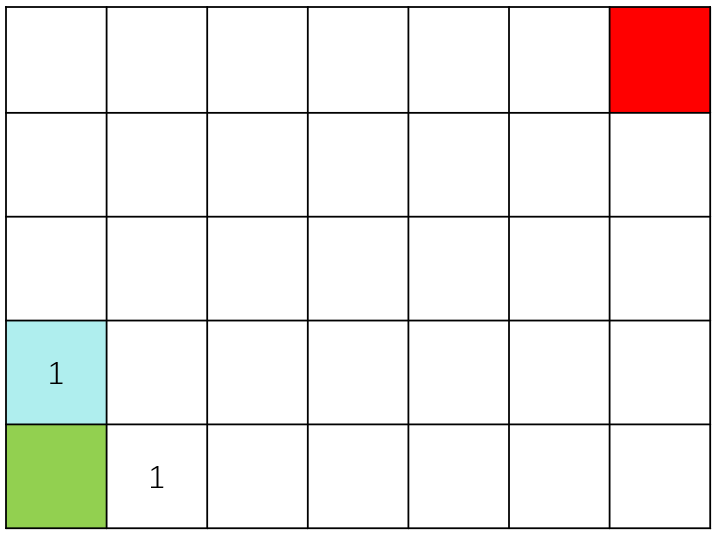
以如图所示的环境为例，绿色栅格为路径规划的起点，红色栅格为终点，白色栅格为未探索过的区域，蓝色表示已经探索过的节点。这里规定机器人只能上下左右移动，不能直接到达斜对角的节点，所以对于每一次扩展，只考察节点周围上下左右的四个邻居节点。



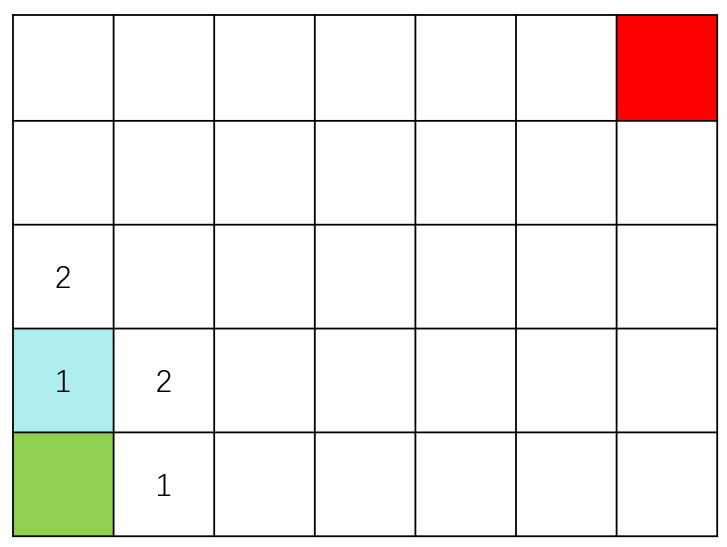
首先考虑起点周围的节点，扩展其上下左右四个邻居节点，并记录父子关系。标记该节点和起点之间的曼哈顿距离作为代价。如下图所示：



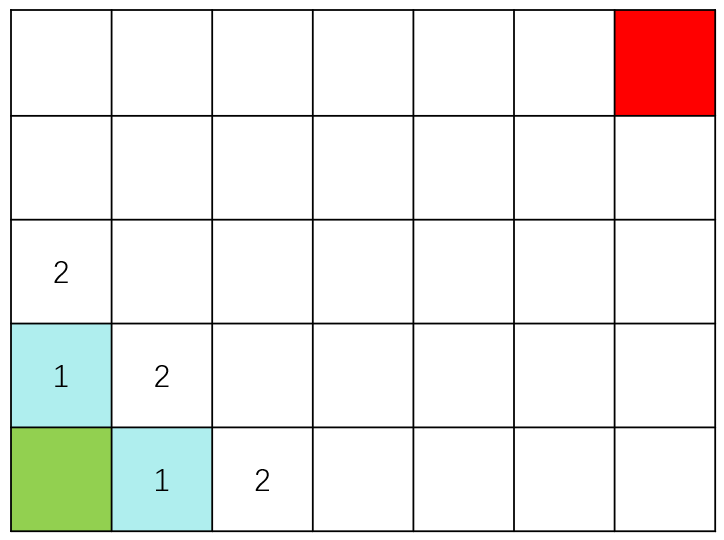
选择代价最小的节点作为下一步要扩展的节点。此时位于起点上方和右侧的节点代价相同，对于这种多个邻居节点具有相同的最小代价情况下，按照人为指定的规则选取其中之一，在这里规定优先从上方开始按照顺时针方向选取：



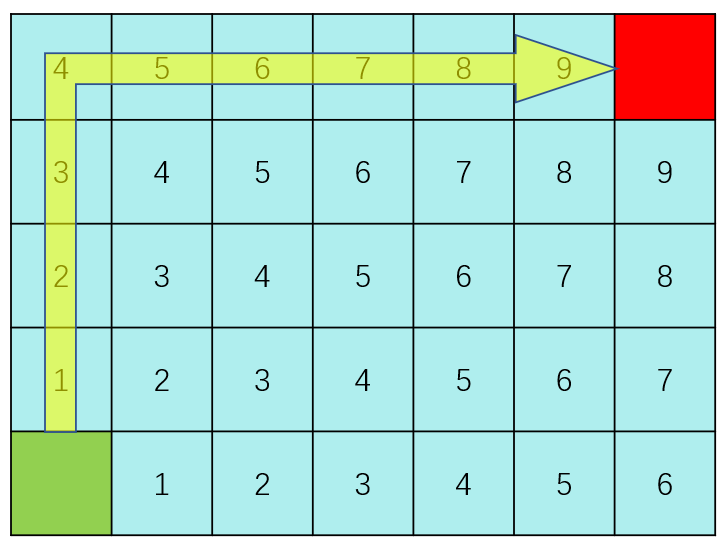
扩展蓝色节点：



此时，扩展的节点中选取代价最小的节点并按照前面的方法继续扩展：



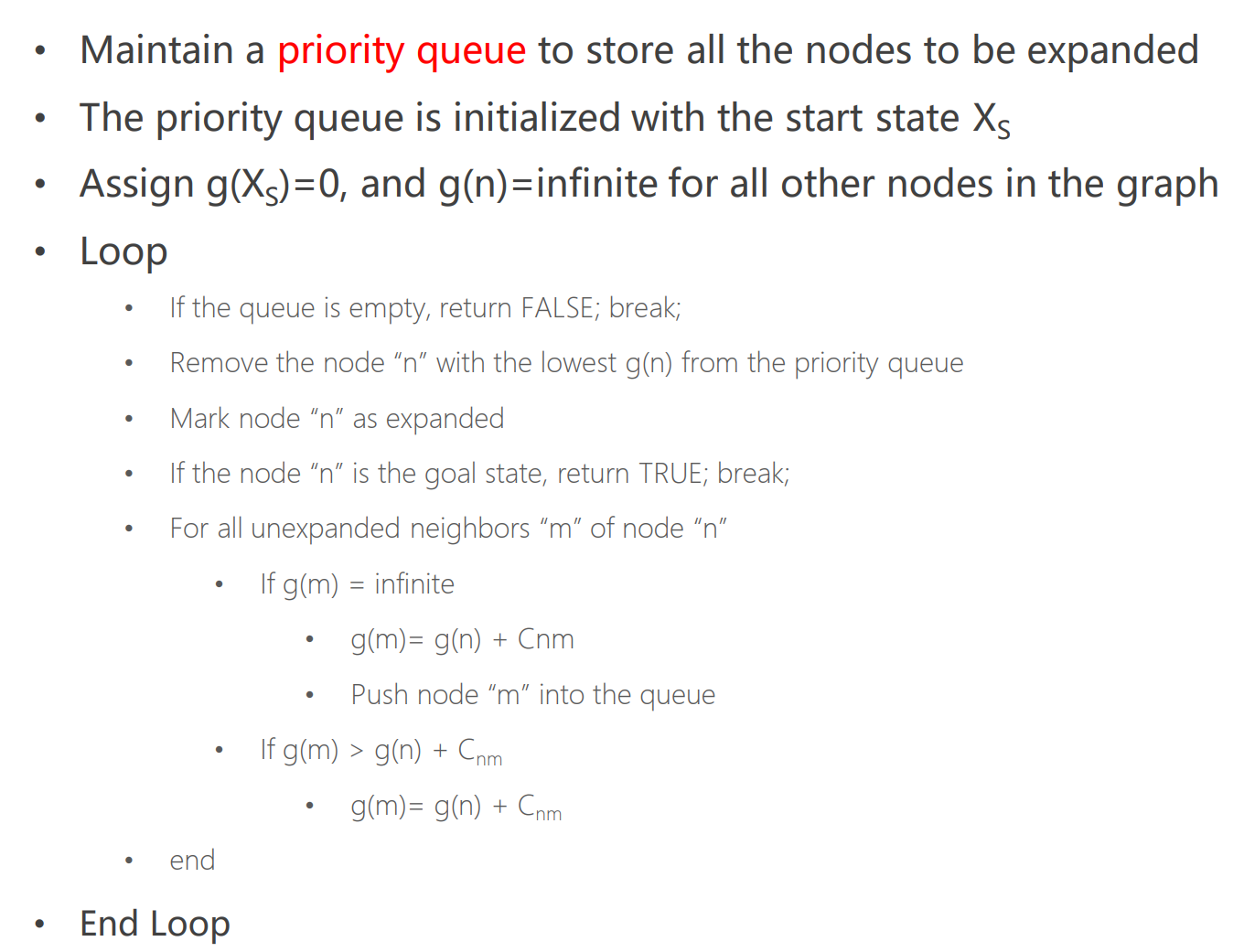
按照这种方法，直到搜索到目标点或者再也没有节点可以扩展，搜索结束。若在最后一步中搜索到了目标点，通过节点之间的父子关系，由最后一步发现目标点的节点向前回溯，生成路径。如图：



Djistra

广度优先算法只适用于无权图，即机器人到达每个邻居节点的代价相同。广度优先算法的改进算法Djistra算法适用于有权图的地图结构。与DFS算法类似，Dijkstra算法每轮都选择路径代价之和最小的节点进行扩展，但是在这里的环境中，节点之间的代价是不同的。对于节点之间的代价均相同的情况，Dijkstra算法退化为BFS算法。

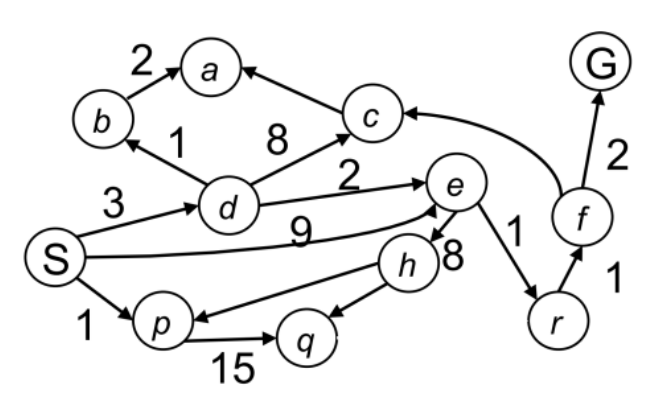
算法的伪代码：



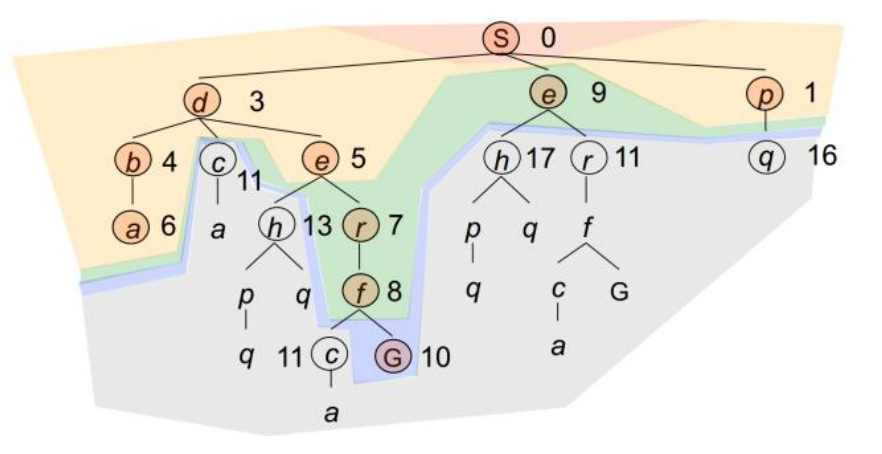
在开始搜索之前，构造优先级队列储存所有将要被扩展的节点。优先级队列具有自动排序的性质，对其存储的所有节点，根据自动进行排序，并在下一次弹出节点时优先弹出最小的节点。优先级队列中的节点被初始化为状态，即将起点送入优先级队列中。对于起点有且对于其他所有的节点有 . 开始遍历所有节点：

1. 如果优先级队列为空，则表示未发现可行路径，返回，跳出循环。
2. 从优先级队列中弹出代价最小的节点
3. 标记节点为已经被扩展过了
4. 判断，如果节点是目标点，则路径规划完成，结束循环
5. 对于节点的所有未被扩展过的邻居节点：若，说明节点还未被扩展过，则更新，其中为从节点到节点的代价，并将节点压入优先级队列。若节点有，则更新

以下图所示的地图结构为例说明路径规划过程，机器人需要从S节点出发到达G节点。

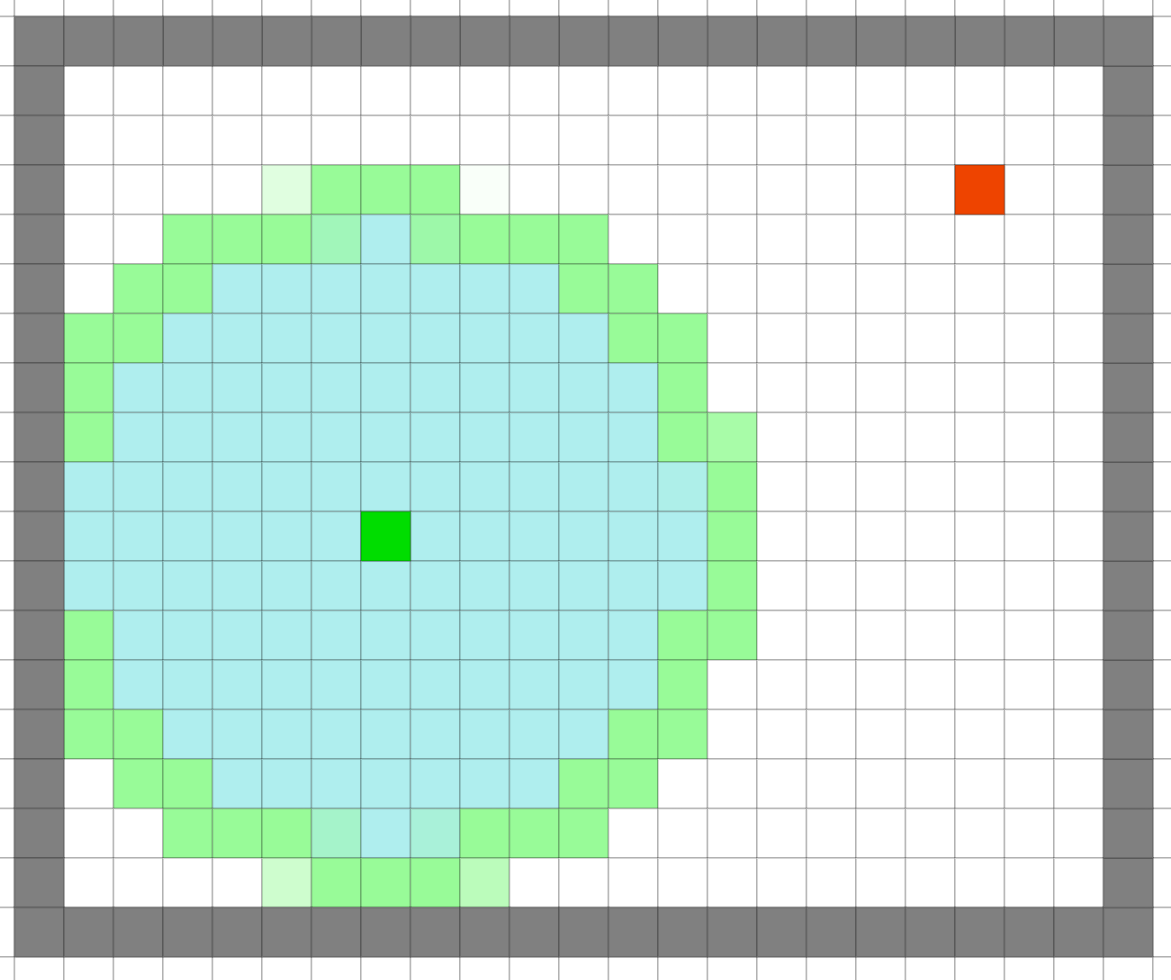
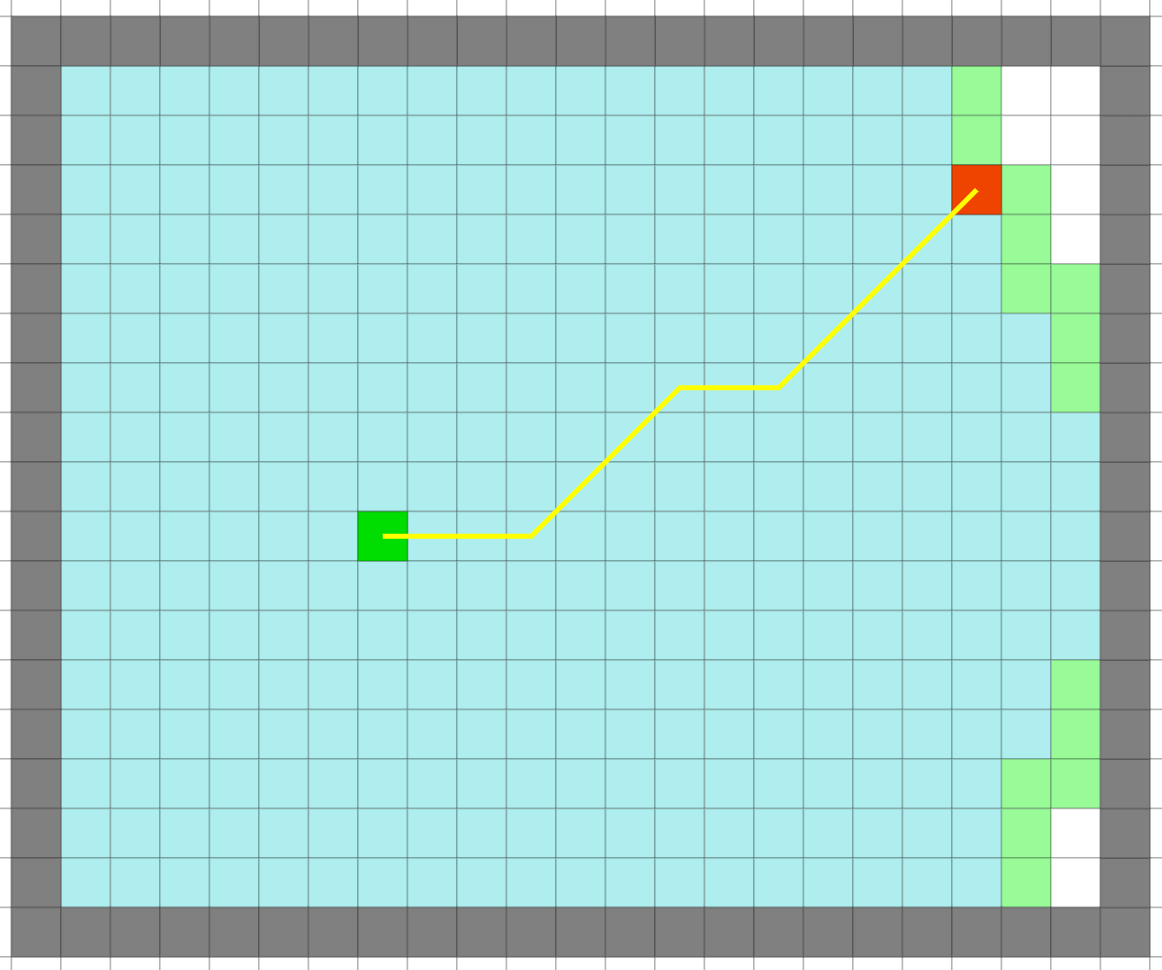


该图可以抽象为下图所示的形式：



首先从S节点出发，遍历S节点的邻居节点d，e，p，选择代价最小的邻居节点p。扩展p，p只有一个邻居节点q。遍历d为当前已经扩展的代价最小的节点，选择d进行下一步的扩展。按照上述方法，扩展的顺序为S-p-d-b-e-a-r-f-e-G，最终搜索到终点G，按照节点之间的父子关系，回溯得到路径S-d-e-r-f-G。

BFS算法和Dijkstra算法从起点开始由近到远地遍历地图中的所有节点，虽然可以得到完备且最优的解，但是探索了大量背离目标点方向的节点，造成时间和空间复杂度上的浪费。如下图所示的Dijkstra搜索过程，探索起点左侧和下方的节点是没有意义的。为了使算法更有目的性地优先向目标进行探索，提出了A\*算法。

A\*是一种基于经典的启发式搜索算法**错误!未找到引用源。**，是基于Dijkstra算法和广度优先算法的优化算法。A\*算法与Dijkstra算法的不同之处在于代价函数的选取，A\*算法的代价函数中增加了启发式函数，使算法具有启发性。其基本的搜索流程如下：

1、从起点开始，扩展当前节点，即遍历当前节点的邻居节点并记录节点之间的父子关系。

2、对每个邻居节点计算代价。

3、选取代价最小的邻居节点进行下一步的扩展，已经被访问过的节点不再进行扩展。

4、选择的代价最小的节点，重复上述步骤。

5、搜索发现目标点，根据节点之间的父子关系回溯生成路径。若在发现目标点之间已经没有节点可以扩展了，则算法结束，未发现可行路径。

算法的核心在于计算代价函数，采样点计算公式如下：

(3-1)

其中：

——表示节点到起点的距离代价，一般可以使用节点和起点之间的曼哈顿距离、对角距离或欧式距离等。在ROS navigation框架中，A\*全局规划默认每一次只能到达当前位置上下左右方向的栅格中，在每一次扩展时，只搜索当前节点的上下左右四个邻居节点，故在计算 时使用曼哈顿距离。

——启发式函数，用来衡量节点到目标点的代价，一般可以使用节点到目标点之间的曼哈顿距离、对角距离或欧式距离等，也可以是其他人为规定的启发函数，用来引导算法启发性地向目标点进行搜索。

常被称为启发函数，启发函数影响着A\*算法的行为：

当始终非常小趋向于0时，路径点的选取将完全取决于，即从起点开始，按照距离起点从近到远的顺序依次遍历所有节点直到发现目标点。算法退化为广度优先搜索或Dijkstra算法。

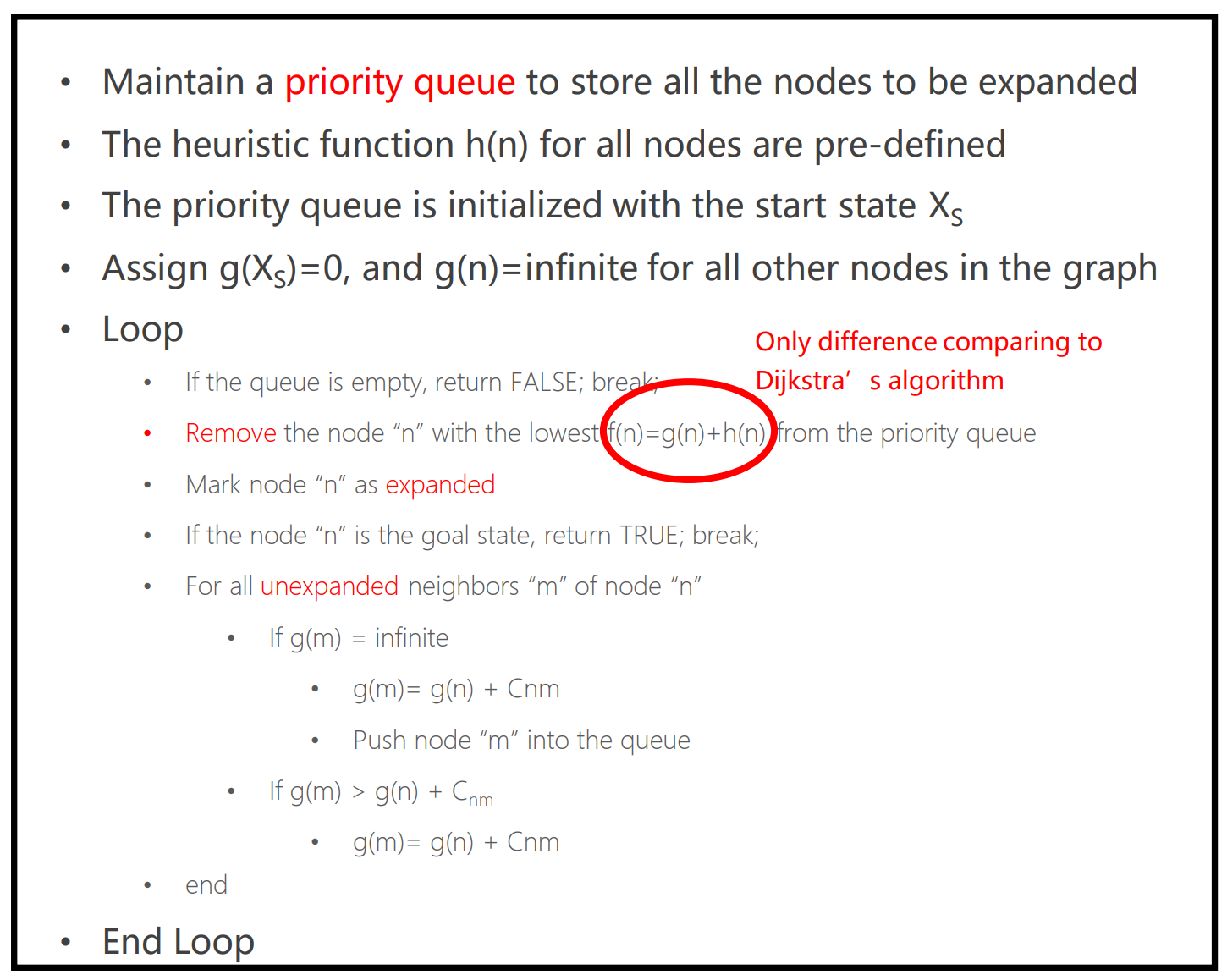
当远大于时，选取下一步扩展节点的顺序将取决于，即每次都选取最靠近目标点的节点，以最贪心的形式向目标点搜索。此时算法退化为贪心搜索。

如果完全等于节点到终点实际最短路径长度时，则A\*算法将找到最佳路径，并且速度很快。可惜的是，并非所有场景下都能做到这一点。因为在没有达到终点之前，我们很难确切算出距离终点还有多远。

虽然更大的可以使算法更有针对性地向目标点搜索，提高搜索的速度，但是破坏算法的最优性。为了保证A\*算法具有最优性，即生成的路径是所有可行路径中最短的，需要对进行限制。可以证明，当没一个节点的都满足以下条件时，A\*算法具有最优性：

其中代表机器人从当前节点到达目标点的真实的最短路径长度。对于平面栅格地图中，每次只能到达上下左右格点的机器人而言，选取曼哈顿距离可以使A\*算法总是具有最优性，如机器人允许朝对角方向移动，则可以使用对角距离。如果机器人允许向任何方向移动，则可以使用欧式距离。

算法的伪代码：

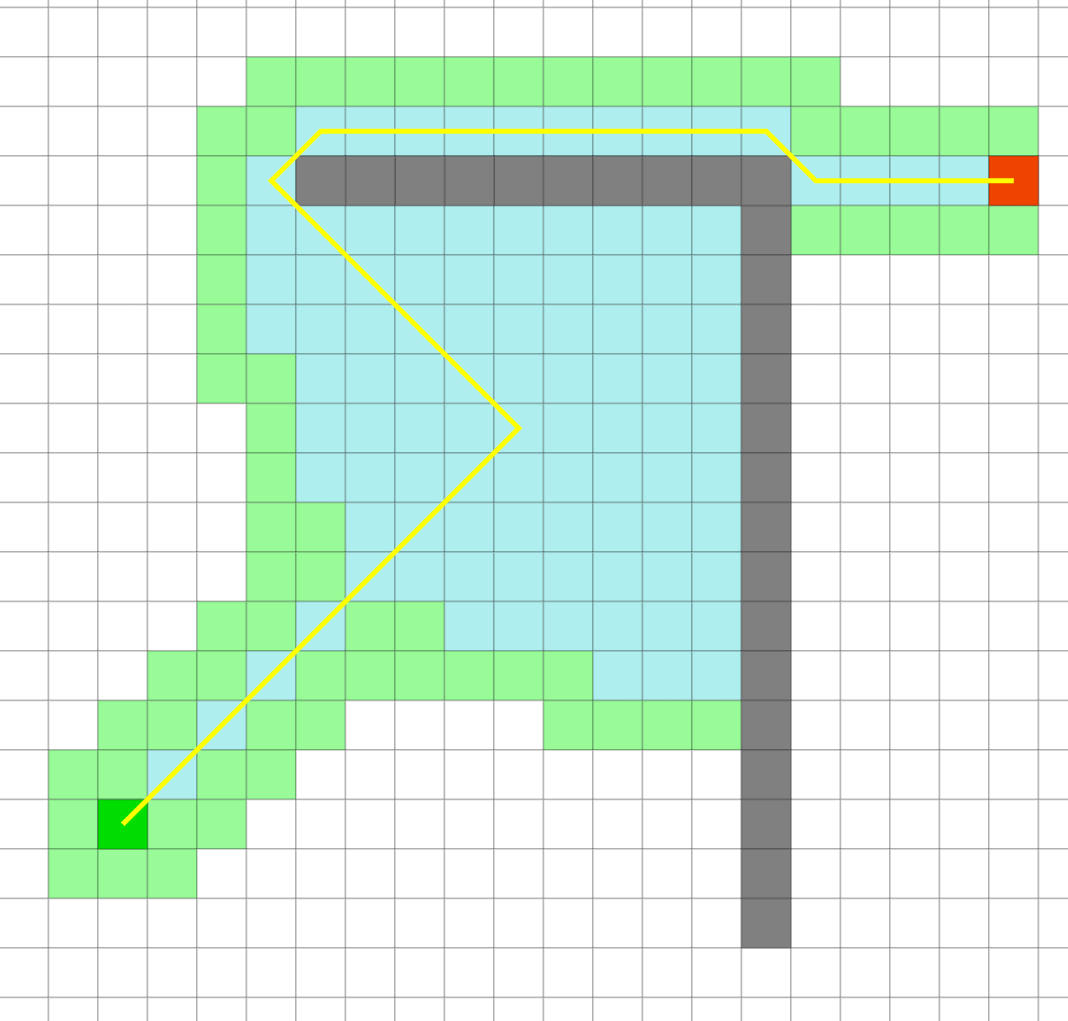
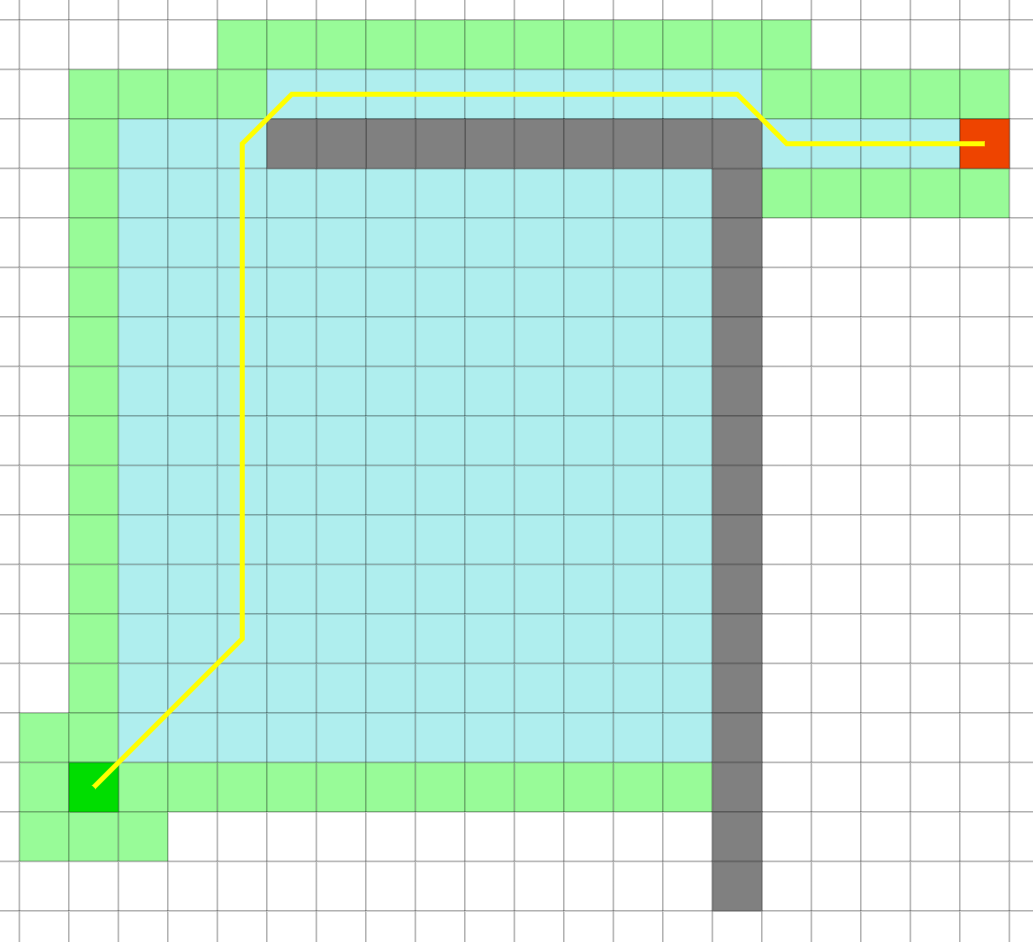
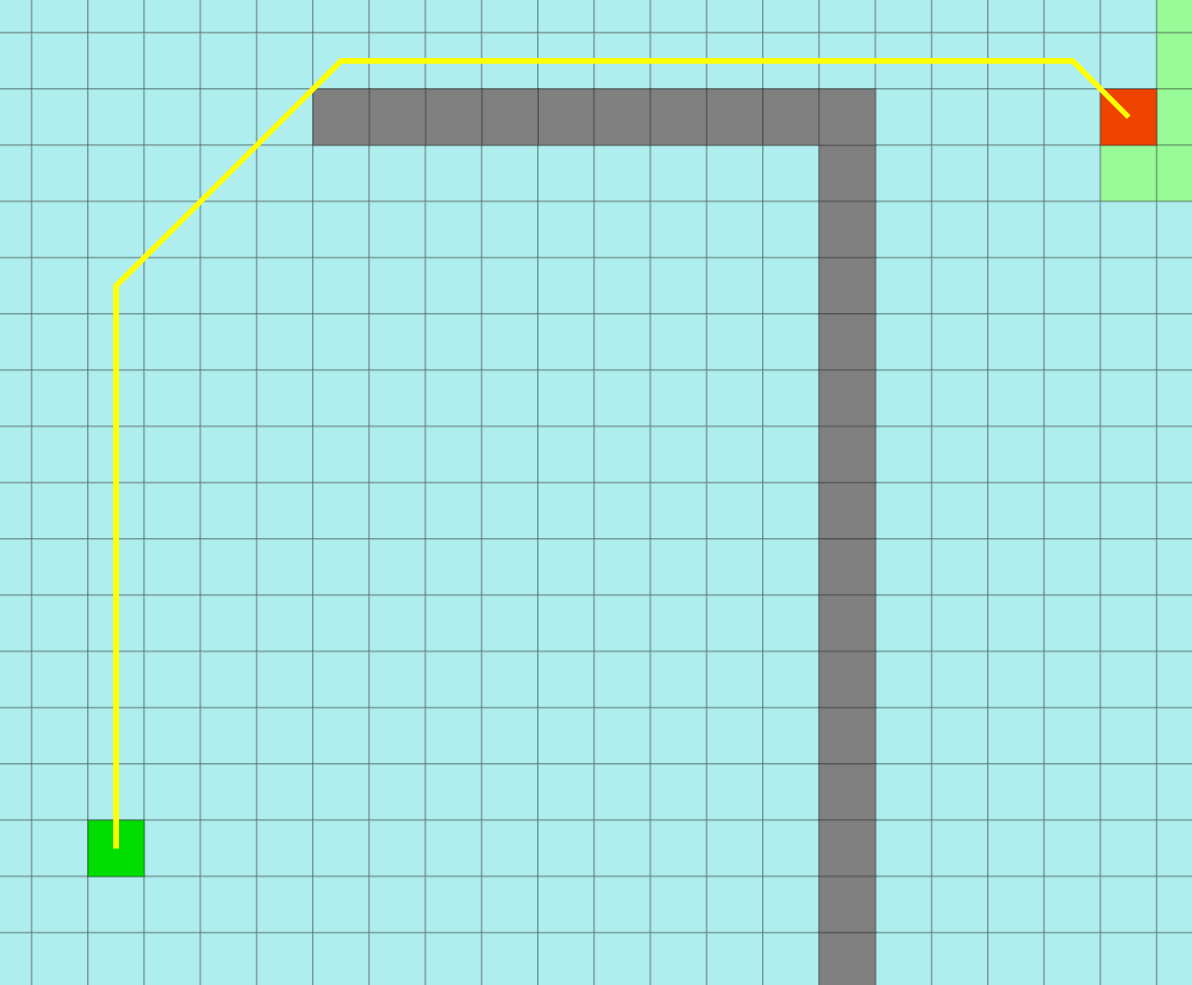


与Dijkstra算法唯一的不同在于， A\*算法中使用替换了Dijkstra算法中的代价函数.

在Movebase框架中，使用了A\*算法的改进形式weighted A\*算法，与传统的A\*算法区别在于代价函数的计算。weighted A\*算法的代价函数为：

其中是人为设定的经验值，用来调整算法的贪心程度。越大的使得发挥的作用越大，这将指导搜索过程优先向着目标点的方向进行。虽然weighted A\*可能破坏最终规划到的路径的最优性，使得寻找到的路径不一定是可行路径中最短的，但是适当增大可以减少扩展的节点数量，提高搜索的速度。本质上，weighted A\*是一种牺牲最优性换取时间效率的改进方法。

图1为即经典A\*算法的规划结果，图2为的规划结果，可以看出，适当提高大大减少了扩展的节点数量，提高了规划的速度，但是规划得到的路径并不是最优的。图3为时的规划结果，算法即退化为Dijkstra算法或广度优先搜索算法，规划的结果如下图所示。虽然得到了最优的路径，但是搜索过程中扩展了大量的无用节点，拖慢了规划的速度。

**

#### 局部路径规划

 机器人在获得目的地信息后，首先经过全局路径规划规划出一条大致可行的路线，然后调用局部路径规划器根据这条路线及costmap的信息规划出机器人在局部时做出具体行动策略。常用的局部路径规划算法有动态窗口法(Dynamic Window Approach, DWA)、时间弹性带(TEB)和模型预测控制(MPC)。

# 动态窗口法

动态窗口法((Dynamic Window Approach, DWA)在速度空间中采样多组速度，并模拟出这些速度在一定时间内的运动轨迹，并通过评价函数对这些轨迹进行评价，选取最优轨迹对应的速度驱动机器人运动。 其规划过程基本流程如下：

1.在机器人的控制空间中对线速度和角速度进行离散采样

2.对于每个采样速度，从机器人的当前状态执行前向模拟，预测在之后的一段（短）时间内如果机器人执行这个采样速度之后产生的轨迹。

3.使用包含以下评价项的度量来对前向模拟产生的每个轨迹进行评分：与障碍物的距离、与目标的距离、与全局路径的距离和速度。剔除与障碍物相撞的非法轨迹。

4.选择得分最高的轨迹并将其对应的速度发送到移动机器人底盘。

5.重复执行上述步骤。

DWA算法计算量较小，适用于差速转向和全向移动机器人，但是不适用于阿克曼转向底盘。同时，DWA算法前瞻性不足，在执行当前规划时，不会考虑这一次规划的动作完成后的状态对后续规划的影响。在动态环境中，DWA算法规划效果也有所欠缺。

## 一、机器人运动学方程

### 1、一般方程

以同步转向机器人（synchro-drive robot）为例，设t时刻在全局坐标系中的机器人坐标为，转角为。同步驱动机器人收到运动学约束，使得机器人的平移速度始终在转角的方向上。设和为机器人在和时刻的坐标，设为机器人在时刻的平移速度，表示其旋转速度。下文中，速度代指包含平移速度和旋转速度的广义速度。那么和可以表示为关于，和的函数：



同理，可以表示为：



而机器人在时刻的平移速度是初始平移速度和加速度的函数，转角是关于初始时刻角度和角加速度的函数，其中。式可以展开为：



在机器人实际的运动控制过程中，加速度只能离散地进行控制，即和在控制周期内保持不变。定义，式可以离散化的表示为。



### 2、近似方程

由生成的轨迹很复杂且计算复杂度高，并且执行交叉点检查等几何操作的成本也很高。为了推导出更实用的模型，我们可以将机器人在内的速度近似为一个定值，即使用近似代替式中的，得到近似形式的运动学表达式：



类似地，对于转角可以使用类似的方法进行近似。

将式写为简化形式：



其中：



对于y坐标有类似的表达形式：



其中：



观察运动方程不难发现，简化后的机器人运动在每个时间间隔内符合式所描述的圆弧



其中：



## 二、搜索空间

DWA算法中，机器人的速度空间分为三层：

1、受制于动力和机械结构的限制，机器人可以达到的最大速度和角速度存在上限，定义机器人可能达到的速度空间为。一般由机器人的参数给出。

2、为了保证机器人的安全，需要保证机器人以某一速度从当前位置开始，以允许的最大加速度减速，可以在障碍物前停下。定义此速度空间为。由式给出：

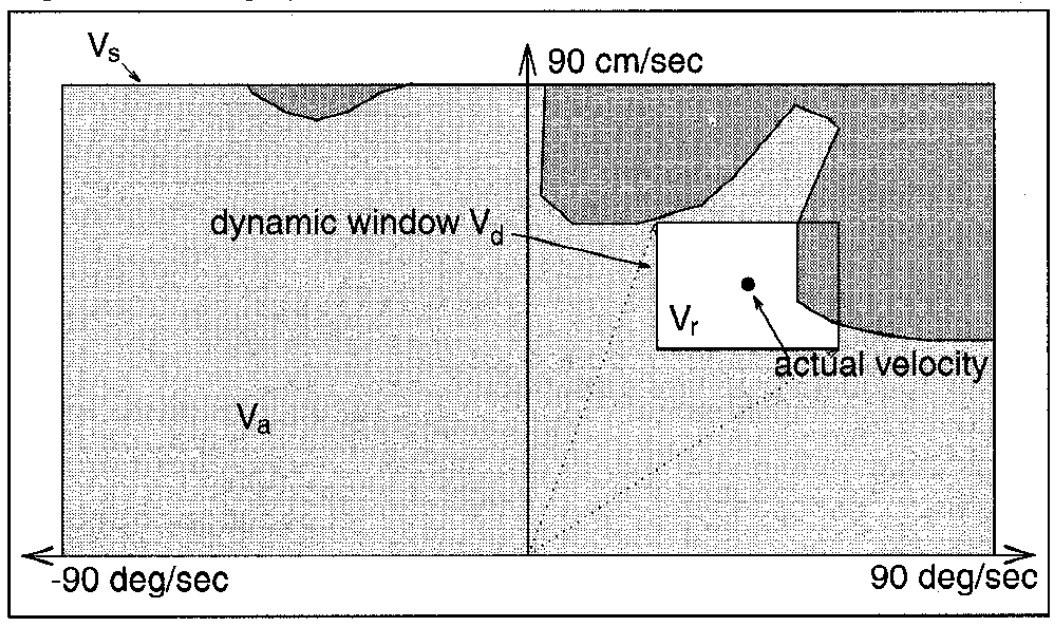


其中表示机器人在的轨迹下，到达障碍物的路径长度。

3、机器人在当前速度下，在规定的时间间隔内可以达到的速度范围，定义为Vd，即动态窗口。通过式给出：



图（1-1）表示了各个速度空间之间的关系：



图（1-1）浅色阴影区域表示安全的速度空间，深色阴影区域表示选择区域内速度将会发生碰撞。方框表示动态窗口的大小。

最终的搜索空间为，和的交集，即：



在搜索空间的进行离散采样得到的速度，首先满足了机器人运动学的要求并且可以避免碰撞。

## 三、前向模拟

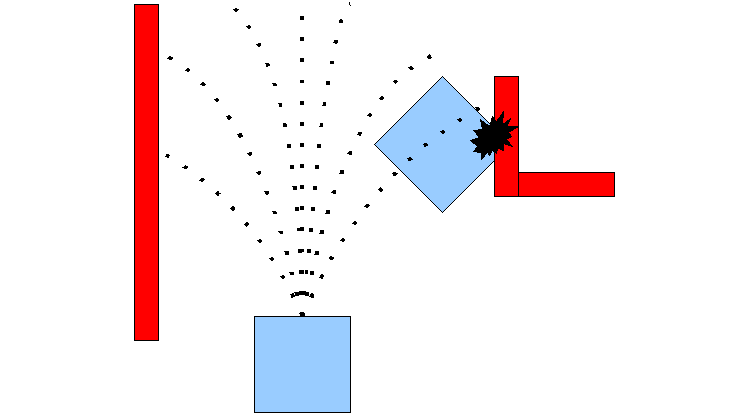
通过在搜索空间V进行离散采样，得到了一系列速度。应用前文中建立的简化运动学方程，对每一个采样的速度进行前向模拟，生成了一系列可行轨迹。可行轨迹的集合表示为：



轨迹生成的可以表示为：



其中机器人的起始位姿为，采样得到的速度为，模拟的时间间隔为，最终得到函数。给出了在时间段内任意时刻的机器人坐标和转角，即机器人的运动轨迹。



## 四、轨迹评估

对于每一条生成的轨迹，使用式的评价函数进行评估。



其中，为规划结束时机器人朝向与初始时目标航向的夹角，如图1-2所示。用来衡量机器人在轨迹结束时偏离目标航向的程度，保持机器人朝向目标位置行进，同时在避开障碍物之后帮助回到正确的方向。 表示机器人在的轨迹下，到达障碍物的路径长度，如果该路径上没有障碍物，则设置为较大的常数。通过该项使得机器人可以尽量避免与障碍物发生碰撞。用于评估机器人完成这条路径任务的进度快慢，只是速度的简单投影，使整体评价倾向于选择更快的行进速度。，，为调整三个分量权重的参数，使用将加权和归一化到之间。

最终得到的路径为候选路径中，使得评价函数最大化的路径，即：



规划基于A\*算法，该算法已经开源整合进入ROS的导航功能包。下面将对这一被动式的导航算法进行介绍，对它与视觉SLAM算法结合的自主探索过程不考虑感知质量而带来的