打分对象一共6个，分别为：

base\_local\_planner::OscillationCostFunction **oscillation\_costs\_**（摆动打分）

base\_local\_planner::ObstacleCostFunction **obstacle\_costs\_**（避障打分）

base\_local\_planner::MapGridCostFunction **path\_costs\_**（路径跟随打分）

base\_local\_planner::MapGridCostFunction **goal\_costs\_**（指向目标打分）

base\_local\_planner::MapGridCostFunction **goal\_front\_costs\_**（前向点指向目标打分）

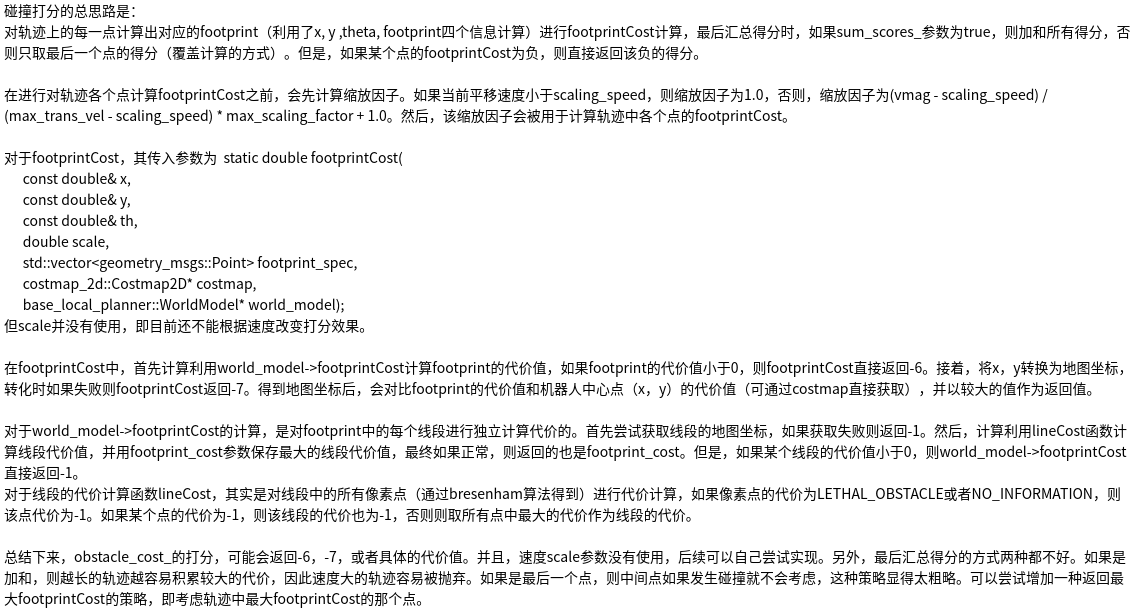
base\_local\_planner::MapGridCostFunction **alignment\_costs\_**（对齐打分）

1. **oscillation\_costs\_（摆动打分）**

该打分项主要利用了轨迹中的xv\_, yv\_, thetav\_三个参数进行摆动判断。摆动打分较为简单，如果xxx\_xxx\_only设置为了true，但是对应的规划路径速度方向相反，则返回代价值-5，否则返回0。

1. **obstacle\_costs\_（避障打分）**

总的来说就是计算轨迹点占用网格的代价，代价值越低路线越合理，但是负值表示路线不可用。需要注意的是代价值计算策略的选择，到底是考虑所有点，还是最后一个点，或是代价最大的点，需要根据实际情况权衡。



1. **path\_costs\_（路径跟随打分）**

利用函数更新局部地图上各点到路径的最近距离，然后判断局部轨迹点对路径的跟随性，代价值越低路线越合理，但是负值表示路线不可用。需要注意的是代价值计算策略的选择，到底是考虑所有点，还是最后一个点，或是代价最大的点，需要根据实际情况权衡。

goal\_costs\_打分项的思路类似。

goal\_front\_costs\_打分项和goal\_costs\_基本一样，只是参考的是机器人前向点和局部目标点的距离，而不是机器人原点和局部目标点的距离。

alignment\_costs\_打分项和path\_costs\_基本一样啊，只是参考的是机器人前向点和局部地图路径的距离，而不是机器人原点和局部地图路径的距离。

