寻路系统

AlrayQiu 邱耀鑫

Alliance 2024 算法组

版本：0.10

日期：2024 年 9 月 15 日

摘　要

哨兵机器人作为全自动机器人，需要具备自主决策并导航到目的地的能力，其中导航到目的地的功能是高级战术的基石，我接下来会介绍哨兵系统各个部分以及导航系统在哨兵系统中的位置，导航系统的整体架构，我们对导航的整体目标以及为了这个目标我们会做什么样的工作，本文采用自顶向下的方式进行讲解，先讲解最切合题目的内容，然后根据需要进行其他部分讲解，最后讲解完整体的导航架构

请注意，2024 赛季导航组只有两个人，本文内容均为一人工作，且大多需求是后面突然想做的，以及由于人手不够又有很多时间在处理其余关键的功能，本身有一些算法的实现是为了快速实现，而非最优效能，许多原理也并没有深究，部分算法带有着按寻思之力，但是总体上是可以不错的工作的（在场下）

目录

1 系统架构 6

1.1 哨兵系统 . . . 6
1.2 导航系统 . . . 7
1.3 导航设计目标 . . . 8
1.4 导航工作管线 . . . 8
1.5 导航系统开发的一些约定 . . . 9

2 一切的开始：路径规划 10

2.1 传统 A* 算法 . . . 11
2.1.1 采样方法 . . . 11
2.1.2 启发函数 h(n) . . . 12
2.1.3 抵达代价 g(n) . . . 13
2.1.4 h(n) 与 g(n) 大小与寻路效果的影响与原因 . . . 14
2.1.5 算法流程 . . . 15
2.1.6 CloseList 的优化 . . . 15
2.1.7 算法实现 . . . 15
2.2 kinodynamic A* . . . 16
2.2.1 A* 算法的不足 . . . 16
2.2.2 状态空间方程 . . . 17
2.2.3 采样方式 . . . 17
2.2.4 启发函数 h(n) 与代价函数 g(n) . . . 18
2.3 Hybrid A* . . . 18
2.3.1 Kinodynamic A* 的不足 . . . 18
2.3.2 剪枝策略 . . . 18
2.3.3 不使用考虑障碍物的启发函数的原因 . . . 19
2.3.4 算法实现 . . . 19
2.4 情况简化与算法实现 . . . 19
2.4.1 可简化的条件 . . . 19
2.4.2 简化可行的原因 . . . 20
2.4.3 算法细节设计 . . . 20

3 导航系统基石：地图生成 21

3.1 地图内容与变换 . . . 22
3.1.1 地图内容约定 . . . 22
3.1.2 局部地图索引变换 . . . 23
3.2 ESDF 生成算法 . . . 23
3.2.1 ESDF 介绍 . . . 23

3.2.2 基于搜索的生成算法 . . . 24
3.3 语义地图 . . . 25
3.4 地图膨胀的意义和距离场的意义 . . . 25
3.5 地图文件格式 . . . 25

4 降采样算法 27
4.1 算法原理 . . . 28
4.1.1 数学表达式 . . . 28
4.2 算法解释 . . . 28

5 求取最优路径：优化器 30
5.1 BSpline 基础 . . . 31
5.1.1 de-Boor 表达式 . . . 31
5.1.2 BSpline 的性质 . . . 32
5.2 Toeplitz 矩阵计算 B 样条 . . . 32
5.2.1 Toeplitz 矩阵 . . . 33
5.2.2 Toeplitz 矩阵表示 Bspline . . . 34
5.3 求值与求导数 . . . 36
5.3.1 均匀 B 样条 . . . 36
5.3.2 非均匀 B 样条 . . . 36
5.3.3 算法实现（类 C） . . . 38
5.4 通过 MinimumAcc 获取控制点 . . . 39
5.4.1 快速获取节点处的物理量 . . . 39
5.4.2 构造二次规划目标函数 . . . 39
5.4.3 构造路径硬约束 . . . 40
5.5 时间重分配 . . . 41
5.6 控制点优化 . . . 44
5.6.1 障碍物优化 . . . 44
5.6.2 平滑优化 . . . 44
5.6.3 非线性优化库优化 . . . 44
5.7 非法路径修正 . . . 45

6 MPC 轨迹跟踪 46

7 附录 47
7.1 表 1：M 矩阵 . . . 47

List of Figures

1 哨兵架构 . . . 6

2 导航系统架构 7
3 导航工作管线 8
4 地图 11
5 A* 网格拓展示例 11
6 启发函数 12
7 两个节点拓展到同一个节点 13
8 抵达函数 14
9 A* 缺陷 16
10 Hybrid A* 采样示意图 18
11 原论文对启发函数的讨论 19
12 地图坐标与索引 22
13 局部地图示意 22
14 ESDF 效果 24
15 ESDF 生成 25
16 膨胀地图 26
17 采样方法 28
18 稀疏效果 29
19 3 阶 DeBoor 图示 31
20 Bspline 的闭包特性 33
21 非均匀 B 样条速度采样对比 37
22 4 阶 B 样条一阶导数 38
23 理论与实际储存位置 38
24 硬约束 40
25 Matlab 二次规划函数 41

1 系统架构

1.1 哨兵系统

哨兵系统是指我们在场上各部分的关联，也指明了各部分负责人应该与什么组进行对接，从而进一步讨论实现什么样的接口提供给其他部分使用，图1 为哨兵架构图，与一般对导航的定义不同的是，我们将定位单独拆分为一个系统，这是根据负责人不同做出的划分

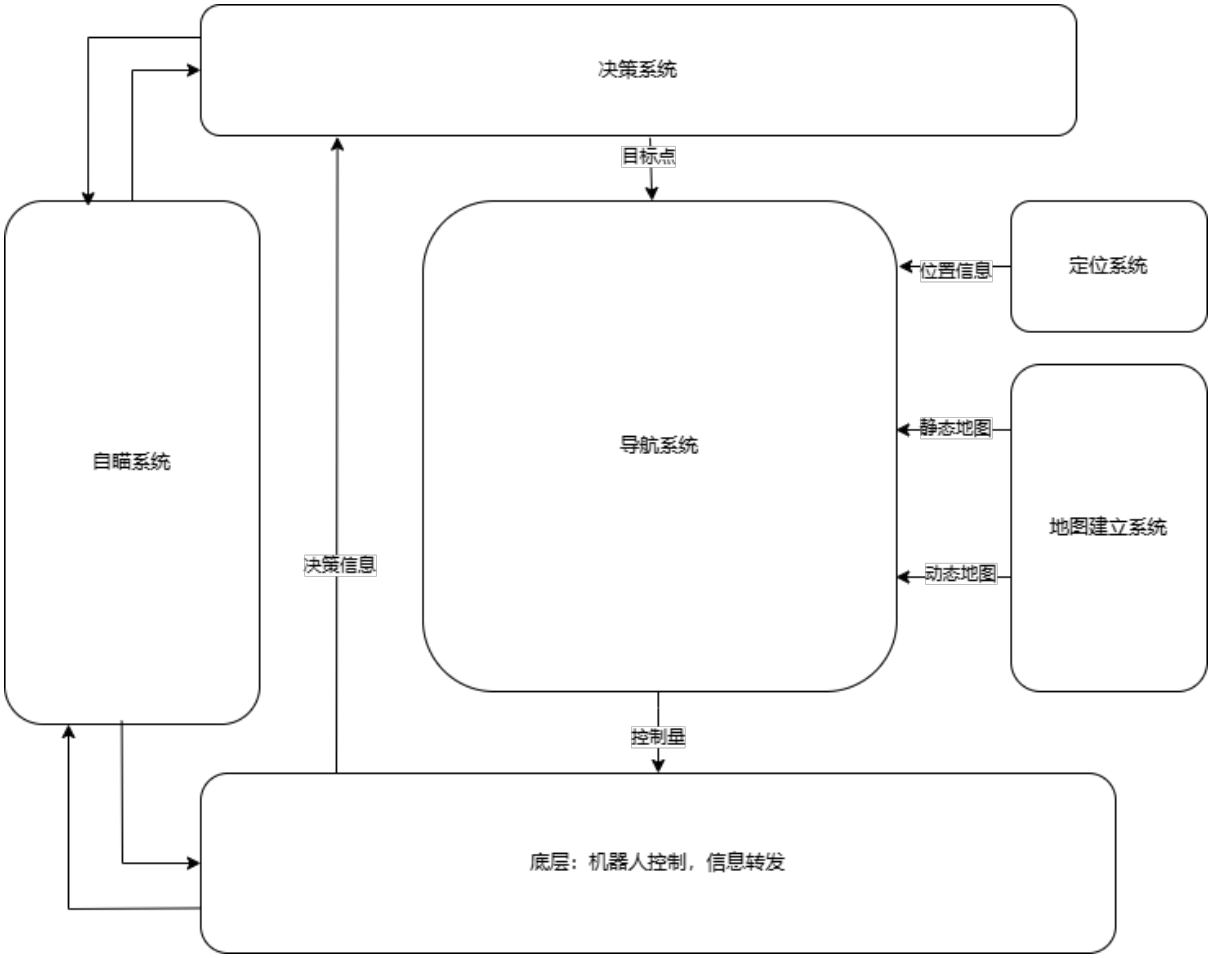


图 1: 哨兵架构

注意

不论是哨兵还是其他的半自动兵种，导航系统的接口都不应该改变

1.2 导航系统

在上一小节中我们学习了导航系统在整体架构中的位置，以及相对应的接口，接下来我会给出导航系统细节的执行部分，如图2所示，对于本图，有几个小点需要注意，事实上导航系统是一个很抽象的概念，这几个小框中，左边地图相关理应放到地图生成的部分，但是我们会发现地图建立系统在运行时仅提供动态地图，而我们又自己处理了一份 $ESDF$(欧式距离场)，这是因为不同的算法使用的地图不同，所需要的处理不同，但是除了部分基于矢量地图的算法，如 $(NavMesh)$ 之外，其地图均可由普通栅格地图计算得到，注意语义地图是静态地图，且我们在运行时不关心语义，因为 $RoboMaster$ 不会突然出现一个楼梯，在比赛中不会有新的单向通过道路出现，将地图的一部分放在导航系统中可以减少替换算法时的行为，但这也是人手不够的举措，如果想分离出去也是可以，同理路径追踪其实也是一个大课题，这边放在导航系统中也是因为人手不够，临时放入，但事实上除了做导航的人会很累以外，没有什么坏处

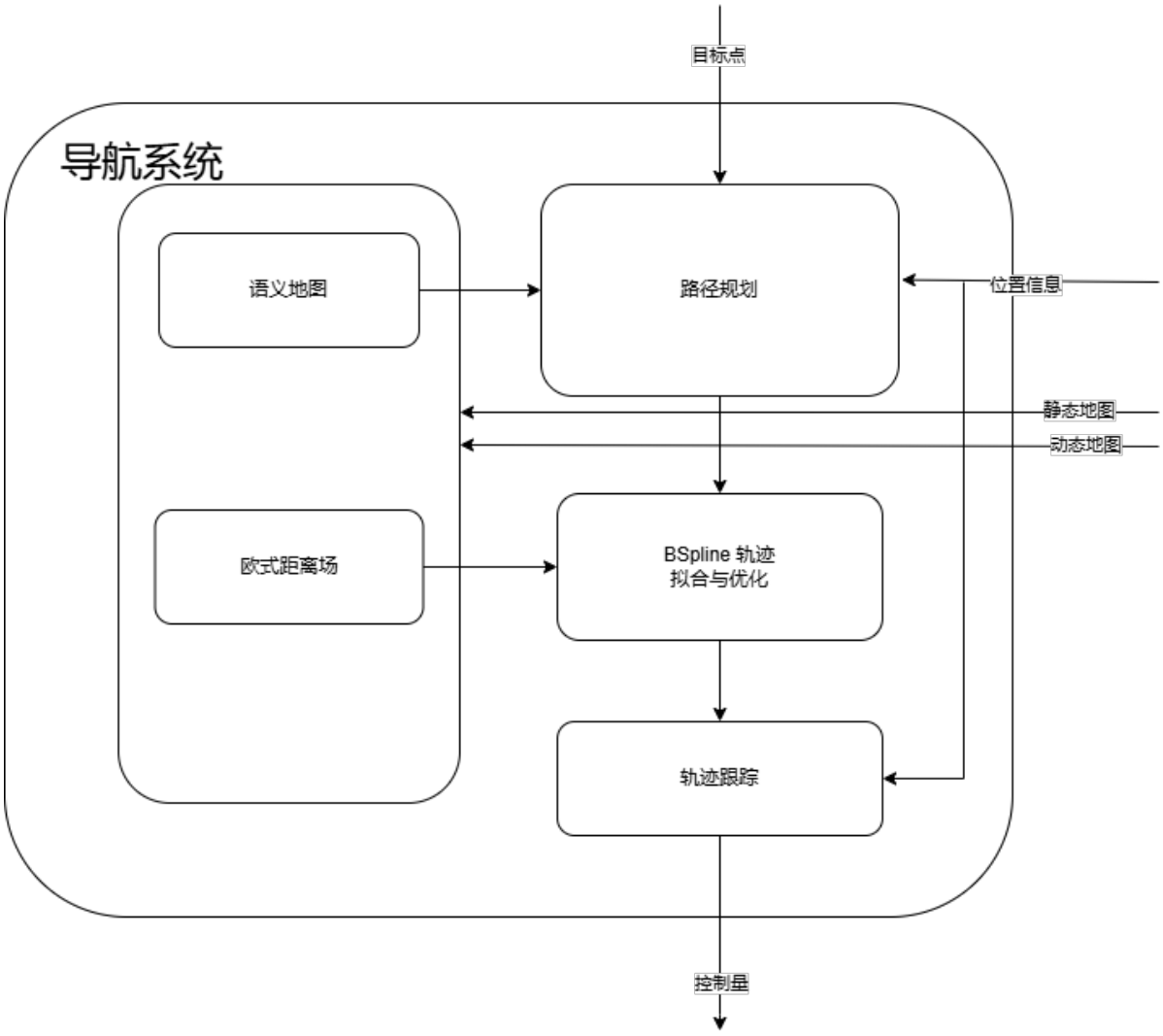


图 2: 导航系统架构

1.3 导航设计目标

导航的设计目标是提供一个高速，反应迅速，易于实现，高度模块化，且各部分彻底解耦合的系统，请大家记住这个目标，因为后面我们对算法做的一切改动或者实现的新算法都是为此目标服务的，同时我们需要一个多线程架构进行不同速度的操作，比如轨迹的得出可能因为很多原因是无法达到很高的速度的，但是我们的轨迹跟踪应该要有一个比较高的频率，来达到一个比较好的效果

1.4 导航工作管线

上一节提供了导航的相关部分，那是一个抽象的，并不关乎实际实现的架构，不论实际实现的算法是什么，都应该遵循这样的一个组织结构，本小节是 2024 赛季哨兵实际使用的工作管线，部分与实际代码不一样的是全国赛后仔细思考后的改进，基本上可以达到设计目标

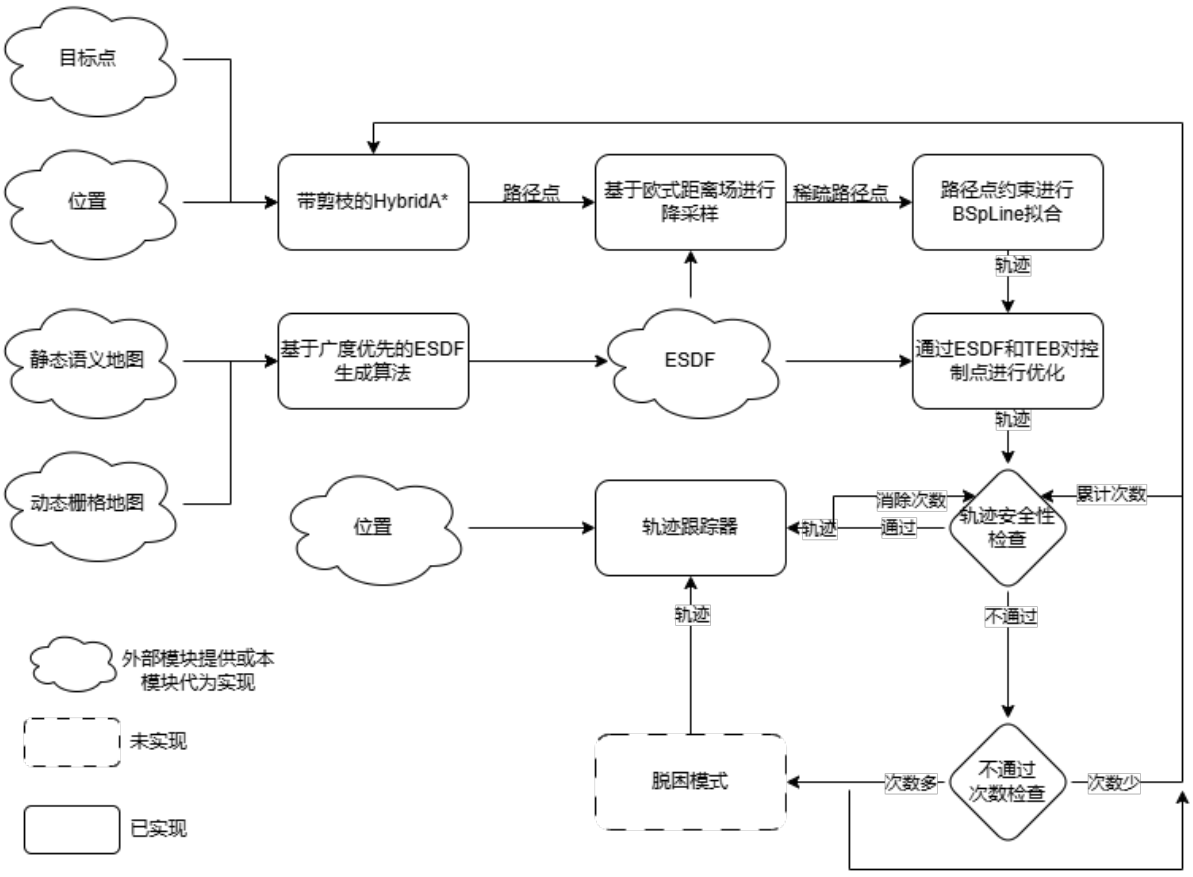


图 3: 导航工作管线

1.5 导航系统开发的一些约定

- 我们的车辆为全向底盘，即使是舵轮（详情见舵轮注意事项与解算文章）
- 每一章讲解的只是单独模块的实现算法，我们假设前置条件已经获取到
- 文章中可能会出现部分多线程相关的数据同步问题，请不要钻牛角尖，我已经全部实现完毕（详情见 $TLARC$ 详解）
- 对于每个小模块之间，由我进行接口类的编写，你可以假设这个接口类提供了你所需要的一切
- 本文算法都有对应的论文作为理论基础，如果想深入学习某一个算法，可以自己查找，这里不列出参考文献
- 所有有关坐标的代码，都使用 Ros 系

> 注意
>
> 我们需要用到大量现代相关知识，没有把握的请先移步线性代数的艺术

2 一切的开始：路径规划

在有障碍的地图上, 获得一系列点，使得机器人依次经过这些点不会与障碍物发生碰撞且能够到达目标点

常见的路径规划方法有基于采样的算法：如 $RRT*$（快速随机生成树），也有基于搜索的算法，如：$A*, Dijkstra$ 等, 我们选择基于搜索的 A* 算法进行寻路，请注意，本文的 HybridA* 本身为 A* 的变种算法，A* 变种算法只是在剪枝方式，采样点生成方式上有区别，所以我们会先进行 A* 的讲解，之后慢慢拓展到我们实际采用的算法

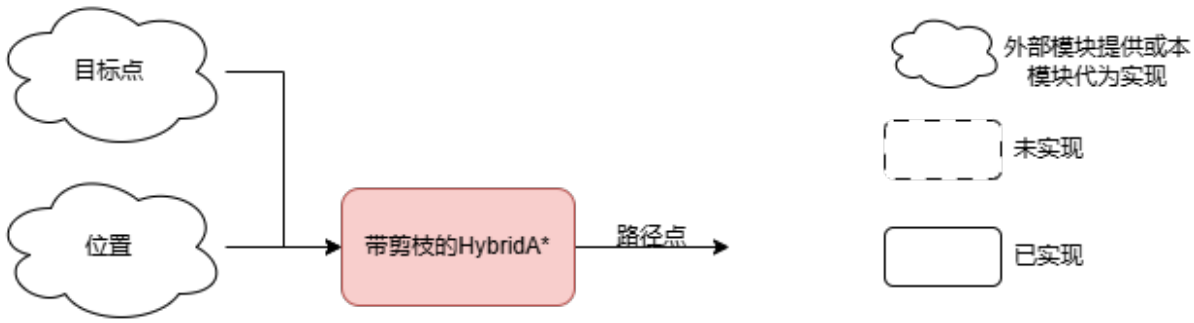
相信通过接下来的一个小节，大家会理解 A* 各个部分的作用，进而快速理解 HybridA*，同时，我会给出 HybridA* 的工程代码和仿真平台

注意

哪怕觉得自己会 A* 算法也要过一下第一小节，里面可能会有不一样的角度，而那个理解方式可能在后面会有用到

警告

一知半解是十分危险的



2.1 传统 A* 算法

$A*$ 算法以及 $A*$ 算法的变种有一些共同的特点，变种 $A*$ 只是在这些部分上进行修改，原理上极度相似，本小节先详解 $A*$ 算法，作为上世纪老算法，大概也能算的上人尽皆知，但是一知半解是很危险的，这可能会导致在后面 $HybridA*$ 的理解上出现问题，在之后的小节中，我不会对优化剪枝的 $HybridA*$ 算法中每个点进行讲解，只会挑出不同点进行讲解 如图，我们要从

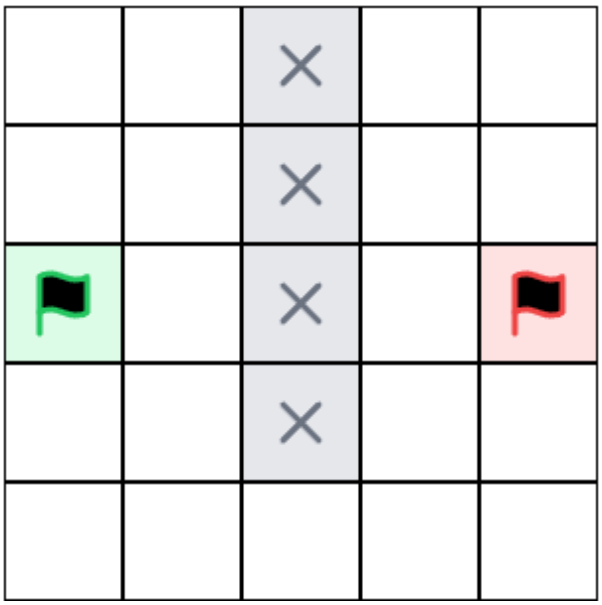


图 4: 地图

绿色旗帜搜索一条到达红色旗帜的路径，机器人只可以上下左右进行移动，为了防止钻牛角尖，我们来简单解释下，我们通过某种方式把实际上的地图变化成这样一张带有障碍物和起点终点位置的栅格地图

2.1.1 采样方法

对于一个格子，他可以达到的是周围的 4 个（因为只可以上下左右移动）可到达格子，这很好理解，每个格子在被拓展的时候，记录下它的父母格子，也就死他是从什么地方到达这个各自的，举个栗子，如图5所示，黄色的格子是绿色格子拓展而来，他们自动记录下父物体，而我

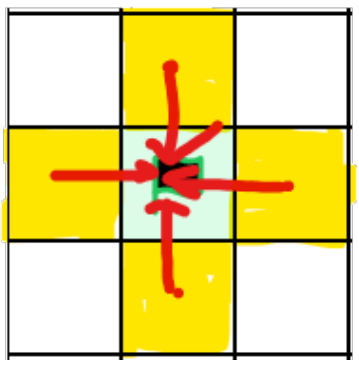


图 5: A* 网格拓展示例

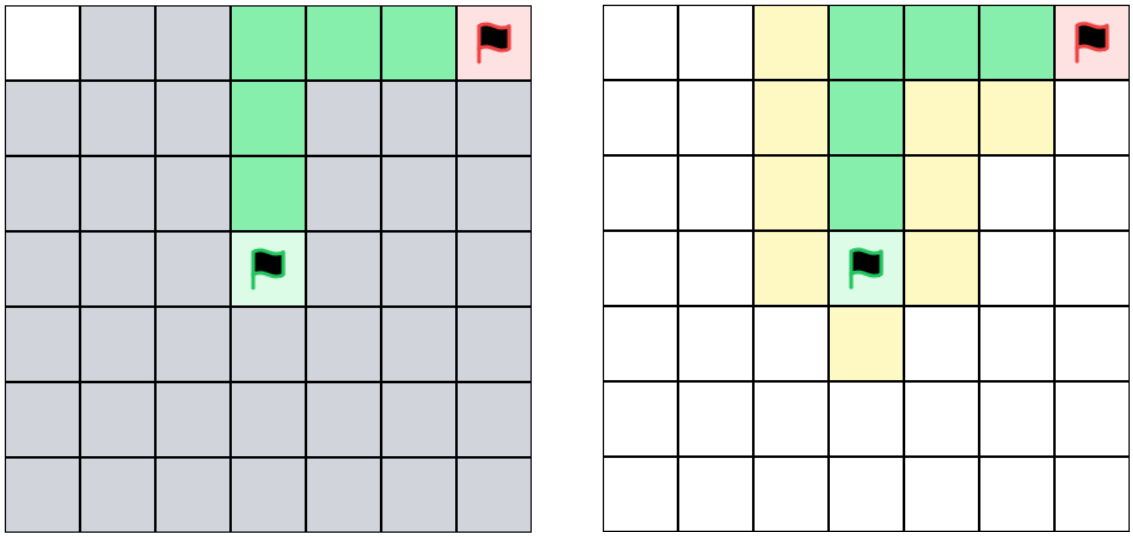
们选取这四个点，是因为我们规定了机器人只能上下左右移动，我们把这四个点记录下来，构成的集合我们称作 $OpenList$，已经走过的点，我们也记录下来，称为 $CloseList$, 那么每次拓展格子，就是从 $OpenList$ 中拿出一个格子，进行拓展，拿到 $CloseList$ 之外的点，加到 $OpenList$ 中

> 注意
>
> 在实际实现中使用染色的方法实现 $OpenList$ 和 $CloseList$，对于不再次搜索 $CloseList$ 有疑问的，或者认为可能出现非最短路径的，可以先认为这是一种剪枝策略，就是让我们搜索次数变少的，对于普通 A* 算法，我们有数学证明这么做一定能得到最短路径，可以自己证明下然后来找我验证，对于之后介绍的，则是一种单纯的剪枝算法，为了更快的运行速度

现在我们用一个新的角度来理解上面的行为，注意，接下来的内容很绕，但是十分重要，如果说每个格子代表一个状态，包括：x,y,parent 三个信息，那么，每次更新，就是在可以到达的格子中选取一个格子，然后，通过我们规定的控制量：上，下，左，右，采样出四个点，当然，因为有些点达不到或者已经到达了不会每次拓展 4 个点，将采样到的点添加进入 $OpenList$ 中，这就是全过程，重点来了，既然我们可以通过控制量采样，是否也能进行状态量的采样，然后计算出到达这个状态的路径，这是可行的，但这并非我们的重点

2.1.2 启发函数 h(n)

现在我们得到了四个候选的格子，我们现在的目标是选出最有可能到达目的地的格子，显然，单纯的四个格子没有办法获取到，那么，我们只要给每个格子赋个值，然后选择最有可能到达的点优先进行拓展不就好了



(a) 不带启发函数的效果　　(b) 带启发函数的效果

图 6: 启发函数

为了达到这个目的，我们引入了启发函数 h(n)，普通 A* 中最常见的 h(n) 为曼哈顿距离，公式为

$$h(n) = |target.x - n.x| + |target.y - n.y|$$

,target 为目标点，n 为当前点

上图为带启发函数与不带启发函数的对比，可以发现合适的启发函数可以大大降低我们搜索的格子数量，图中染色格子为搜索到的格子，鹅黄色为在 $OpenList$ 中的格子

2.1.3 抵达代价 g(n)

现在我们思考一个情况，当你越来越靠近目标，h(n) 会越来越大，这个时候，如果出现了障碍物，那么它必然是从靠近障碍物的地方开始搜索的，但是很明显，这样找出来的路径，必然不会是最短的，那么我们需要一个新的量，来获得每个格子比较正确的代价值，另一个情况如图，当两个节点拓展到同一个节点时，新节点的父节点应该是谁

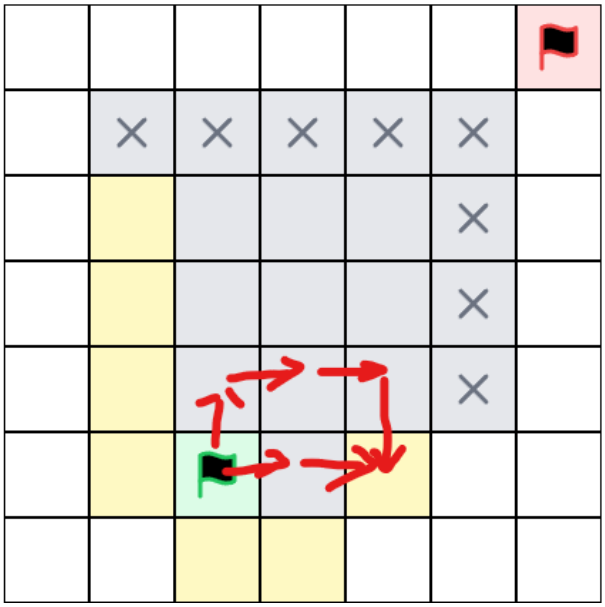


图 7: 两个节点拓展到同一个节点

这时需要的函数为抵达代价函数，这个函数的实际意义是，目前到达这个格子实际花费的代价，g(n) 的计算方法是

$$g(n) = calc\_g(n, parent) + g(parent) \tag{1}$$

其中 $calc\_g$ 函数是计算点 n 和点 parent 之间增加了多少的代价，也可以表示为 $dg(n)$，因为 A* 是按照网格搜索的，因此只要把一路上的 $dg(n)$ 加起来便是 g(n)

$OpenList$ 中的格子代价变为

$$f(n) = h(n) + g(n)$$

, 这样一来，每个点的代价得到了一个比较正确的预估，因为相同格子的 h(n) 必然相同，而 g(n) 与到达该格子的实际花费有关，也就是

那这样会保证不能绕弯吗，并不能，或者说，是否得到一个最短路径，取决于 h(n) 和 g(n) 的关系，以及 $OpenList$ 的实现方式

2.1.4 h(n) 与 g(n) 大小与寻路效果的影响与原因

如果

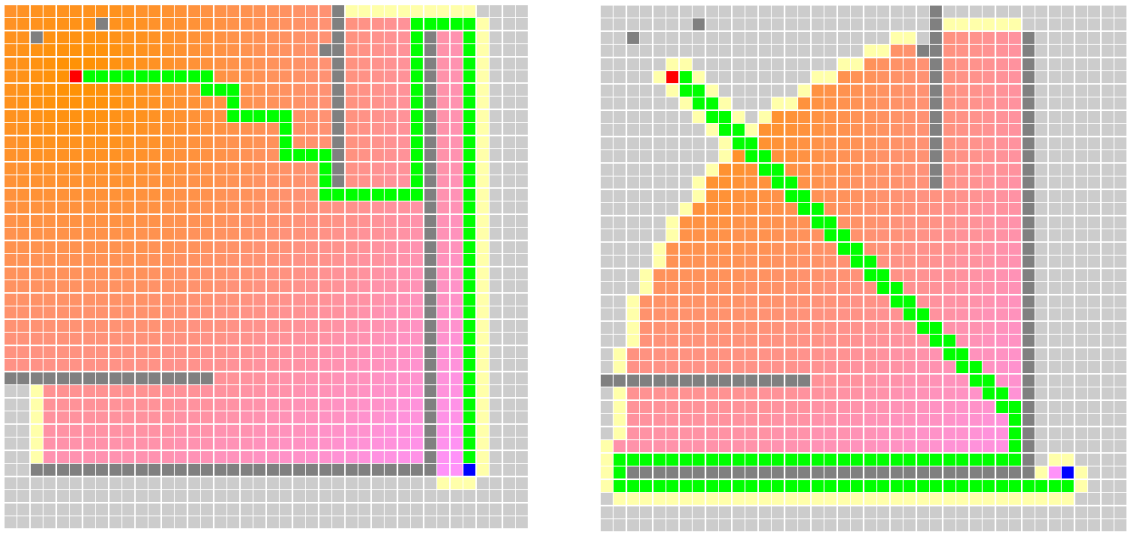
$$\frac{dh(n)}{dn} < \frac{dg(n)}{dn}$$

此时走的远的点的代价一定大于走的近的点，此时会倾向于在近处多搜索，这样得到的路径必然最短，但是会搜索更多的格子

如果

$$\frac{dh(n)}{dn} > \frac{dg(n)}{dn}$$

此时走的远的点的代价一定小于走的近的点，此时会倾向于在末端处多搜索，这样得到的路径可能不是最短, 如下图所示



(a) 曼哈顿距离　　(b) 平方后欧式距离

图 8: 抵达函数

那么当

$$\frac{dh(n)}{dn} = \frac{dg(n)}{dn}$$

此时算法的行为取决于你的 $OpenList$ 如何编写，一般我们在工程上会把 $OpenList$ 实现为一个插入到相同项后的优先队列，也就是优先队列本身存在时间上的关系，这样可以保证在值相同时，优先搜索靠近的点, 虽然前面提到最好是新元素在后，但是其实影响不大，当你无法断定使用的类是否符合的时候可以不管

由上面的介绍，可以发现，A* 算法是一个结合了广度优先搜索和贪心策略的搜索方式，h(n) 的值代表了该点到达目的地的期望价值，但是，期望上离得最近的点，在直线上存在障碍物的时候就不再是全局最优的点，因此我们引入了 g(n)，将 h(n) 与 g(n) 结合，就有了期望上全局起点到终点的最小代价，这样我们每次选择的点就更加接近全局最优

同时也可以发现，当 $h(n) -> 0; g(n)! = 0$ A* 退化为广度优先搜索，当 $g(n) -> 0; h(n)! = 0$ A* 退化为贪婪选择局部最优的搜索方法

2.1.5 算法流程

输入为一个起点，一个终点，从起点开始，把起点加入 OpenList, 开始主循环

主循环：

- 从 OpenList 中取出最优点放入 CloseList
- 调用顶部点的采样方法，获得 4 个新点，每个点在被采样时计算自己的 G 和 H 的信息
- 获得的点检查满足在地图中，不在不可到达区域，不在 CloseList 中
- 满足条件的点加入 OpenList
- 是否找到终点或者 OpenList 中不存在点跳出主循环

跳出后：如果找到目标点，根据 Parent 的内容递归地获取一条路径，如果没有找到目标点，返回无

2.1.6 CloseList 的优化

对于已知大小的地图，若地图太大，在后期检查元素是否存在于 CloseList 中是一个很大的开销，因此我们采用涂色的方式，新建一个数组，大小与地图相同，如果在 CloseList 中，则该位置为 1，否则为 0，需要判断该元素是否存在知道取出该值即可

比如有个 Node(x = 3,y = 4), 此时判断是否在 CloseList 中的操作为 Colored[3,4] == 1, 添加进 CloseList 的方法为 Colored[3,4] = 1, 此方法的思想为 hash, 但是要求 hash 函数必须每种可能都有一个对应，这里的 Hash 函数其实就是 $x * Map.Size.Y + y$

2.1.7 算法实现

本来 A* 我是不留算法实现相关内容的，但是考虑到大多数人可能没有很多代码经验，无法选择写出消耗较小的代码，遂提供实现思路与细节

一般 A* 算法以及相关变种算法需要实现两个类搜索本身的 AStar 类和采样相关的 Node 类，Node 类为栅格点的状态，在普通 A* 算法中，数据有 x，y，Parent，g,

Node 类需要实现的接口有：

- $floatCalcDeltaGValue(Nodethis, NodeParent)$; 计算 G 值变化量, 加上 parent 的 G 值变为自身 G 值
- $floatCalcDeltaHValue(Nodethis)$; 计算当前点到达目标点的 H 值
- $Collection < Node > GenerateChildren(Node3this, GridMapmap)$; 生成自己周围的可达点，过滤部分不在这边实现，此函数永远返回全部的点，不论是有无搜索过的，除了无法抵达的点

$AStar$ 类需要实现的接口有：

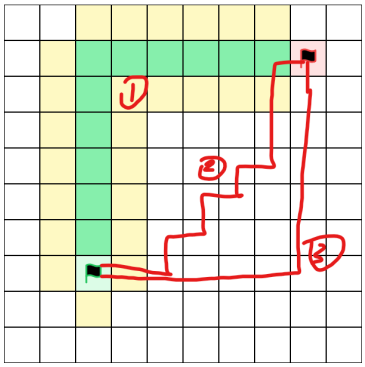
- $Collection < Node > Search(Nodestart, Nodeend)$; 计算路径

$AStar$ 类中有一个栅格地图，$OpenList$ 但是没有 $CloseListCloseList$ 使用 $Colored$ 数组来进行代替，$OpenList$ 使用优先队列进行实现,$Search$ 函数主要的点在于，$OpenList$ 的维护，$Colored$ 数组的维护，以及对于选定点的采样点的判断，注意对于每次从 $OpenList$ 中拿出点的时候，需要判断一次这个点所在格子是否为 $CloseList$

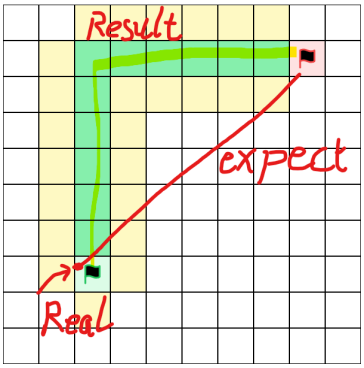
2.2 kinodynamic A*

2.2.1 A* 算法的不足

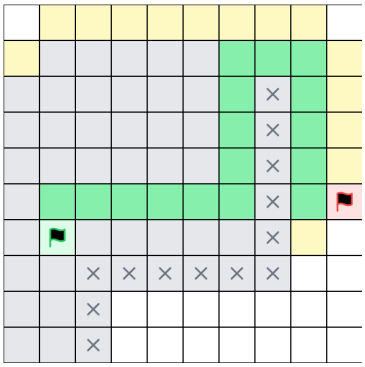
查看前面 A* 算法的采样方式，我们会发现，不论如何表示这个网格，它取到的点总是同一个位置，这显然不符合实际的运动，以及这样的采样方式，不论是 4 网格采样，8 网格采样，或者是 28 网格采样, 他的方向总是固定的，他的代价也是固定的，也就是他有多个相同抵达代价的路径，对于网格而言，他是最短路径，但是在现实世界的连续空间下，显然不是最短的，以及 A* 算法的采样方式，明显不符合动力学约束，也就是对路径点的跟踪需要不断的加减速，或者以不可能的姿态运动，因此，我们需要有一种可以在世界真实坐标下寻路，并且拥有多向寻路的算法，由于我们的轨迹交由后续步骤处理，在这里只要得到大致世界最短的路径即可，由于我们是全向底盘，所以可以允许略微的不满足动力学，这些优化交由后续 BSpline 优化器进行



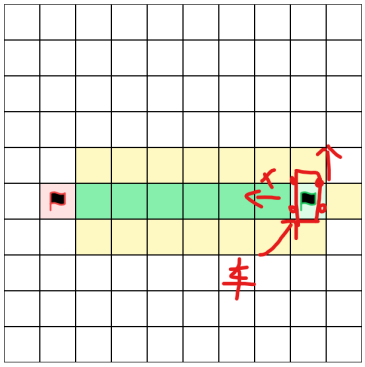
(a) 多条相等代价路径



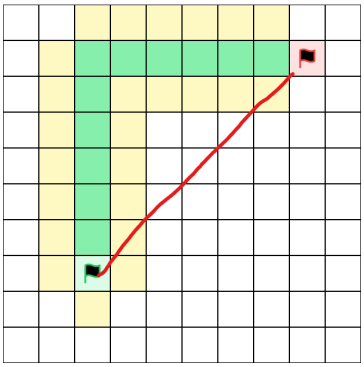
(b) 无法正确表示在空间中位置



(c) 喜欢贴近障碍物



(d) 无法满足动力学约束



(e) 无法走斜线

图 9: A* 缺陷

由于上面的问题，我们放弃传统 A* 算法，转而选择 A* 算法的变种 $KinodynamicA*$，这是一种考虑了动力学约束的寻路方法，可以满足：

- 连续空间的寻路
- 路径点不收到格子限制
- 满足车辆的动力学约束

2.2.2 状态空间方程

由于是动力学约束的 A*, 因此我们需要先解出其控制量与状态量之间的关系，对于全向底盘，我们仅考虑在全局坐标下，状态量 X 与输入量 U 为

$$X = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix}, U = \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}$$

得到的状态空间方程为

$$\dot{X} = AX + BX \tag{2}$$

,

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

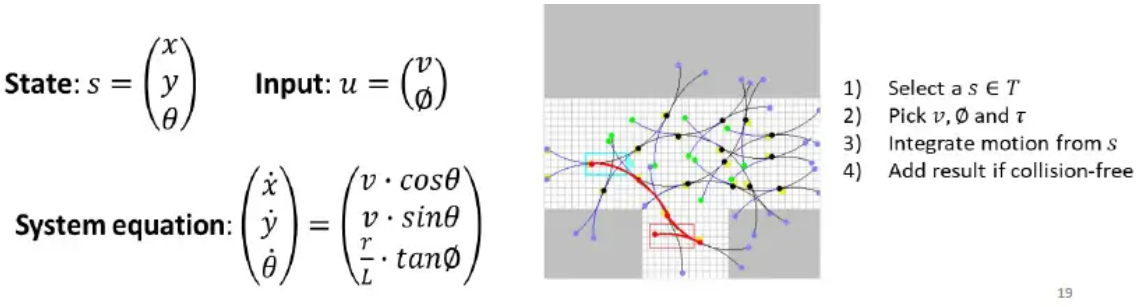
对于某些论文或者博客，喜欢把状态空间方程求解为状态转移方程，请注意，状态转移方程是为求得中间的轨迹，防止无法通过，对于全向底盘，无需这一步，下面是状态转移方程的一般形式，在看到其他文章中的状态转移方程请不要惊讶或疑惑

$$s(t) = e^{At}s_0 + \int_0^t e^{A(t-\theta)}Bd\theta \times u_m$$

2.2.3 采样方式

与 A* 算法不同的地方是，在采样方式上，我们抛弃了网格，每个 Node 储存的内容为 X 向量和父节点，通过公式2获得当前位置不同采样下的 $\dot{X}$，然后固定一个迭代时间 T，算出生成的状态

$$X_n = X_{parent} + \dot{X}_u \times T \tag{3}$$



KinoDynamic A* 效果图

其中 $\dot{X}_u$ 为在特定 u 下，X 取当前点 (上述公式中的 X_{parent}) 通过公式2得出，由于笔者是个懒狗，我们来看别人的图片, 注意图片左边的状态量与输入量，这是基于自行车模型的状态控件方程，其实不用太管是什么，只要按照前面说的方法进行 Node 采样就好了，这里的空间状态方程也比较简单，知道大概是这么个效果就好了

2.2.4 启发函数 h(n) 与代价函数 g(n)

注意，我们又回到全向底盘的讨论中了，对于全向底盘，我们不关心目标的角度，因此，启发函数使用欧氏距离即可，复习下 g(n) 求取公式1，在 $KinoDynamicA*$ 中，g(n) 方程需要一部分的改变，因为我们已经是在空间中的直接两点间路径，所以 $calc_g(n, parent)$ 应该为两点间直线距离

2.3 Hybrid A*

2.3.1 Kinodynamic A* 的不足

$KinodynamicA*$ 已经可以做到获得一系列很好的坐标点，但是其采样到的点十分的多，同一时间会有很多的点在 $OpenList$ 中，这会导致搜索的量变得十分巨大，也就是无法做到快速求解出结果，常用的在中间计算路径曲线的做法也无法保证达到最好的路径

2.3.2 剪枝策略

在 $KinodynamicA*$ 的基础上，我们重新使用加入网格的限制，具体表现为同一个网格只选择一次，这个网格与传统 A* 的网格相同，但是数据记录的 Node 保存的是真实坐标的位置，在加入了剪枝策略之后，$HybridA*$ 可以以一个较快的速度进行搜索，虽然最后的效果会不如 $KinodynamicA*$，但由于后续步骤中的优化器的存在，使得我们在路径规划这一步得出的路径点可以不用那么的完美

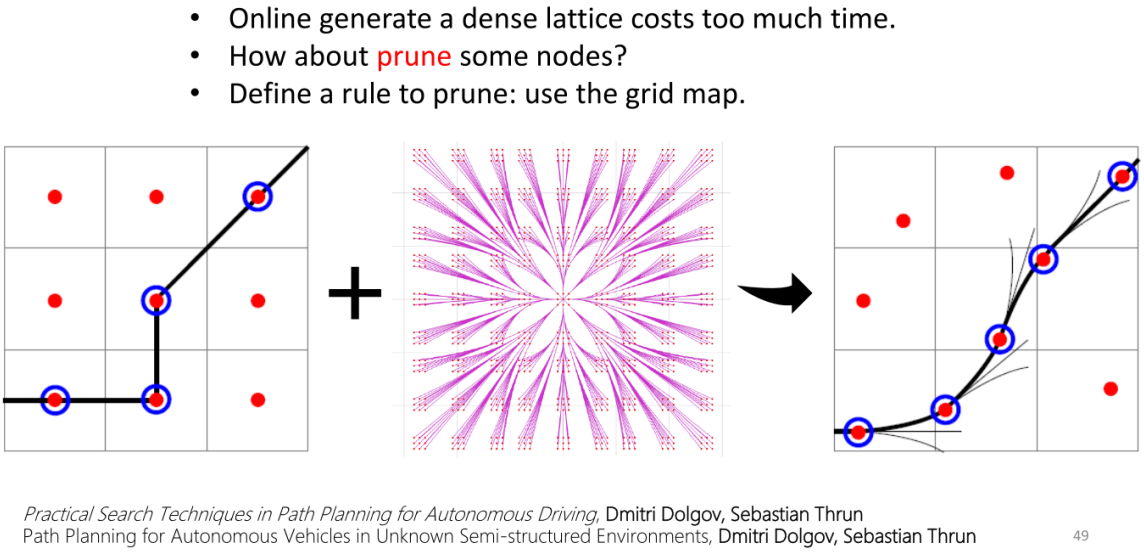


图 10: Hybrid A* 采样示意图

这边还是偷点别人的图，可以看到，$HybridA*$ 在 $KinodynamicA*$ 的基础上，对每个网格只保留了一个点，这样可以有效的减少需要搜索的节点数量

2.3.3 不使用考虑障碍物的启发函数的原因

启发函数虽然叫函数，但是并不是只能计算得出，在原论文中提出的考虑障碍物的启发函数做法是拓展节点时使用普通 A* 进行一次近距离的搜索，如果无法得到达到目标点的路径，则看作有障碍，那么启发函数的值就会小的多，这么做可以防止 $HybridA*$ 在死胡同搜索，但是每次采样都多进行了一次搜索，我们考虑实现难度，代码效率和地图类型，使用的是不带障碍的启发函数，也就是简单的考虑距离，这边进行简单的讲解时为了防止你们自己看原论文或者其他博客的时候，看到相关内容无法理解，因为所有的博客都是偷原论文的图，但是没有讲解如何实现，导致容易误认为带障碍物的会比不带障碍物的启发函数效率高很多，事实上只有在地图存在死胡同的时候效率会高，其余时候会更低，下面附上原论文相关图片

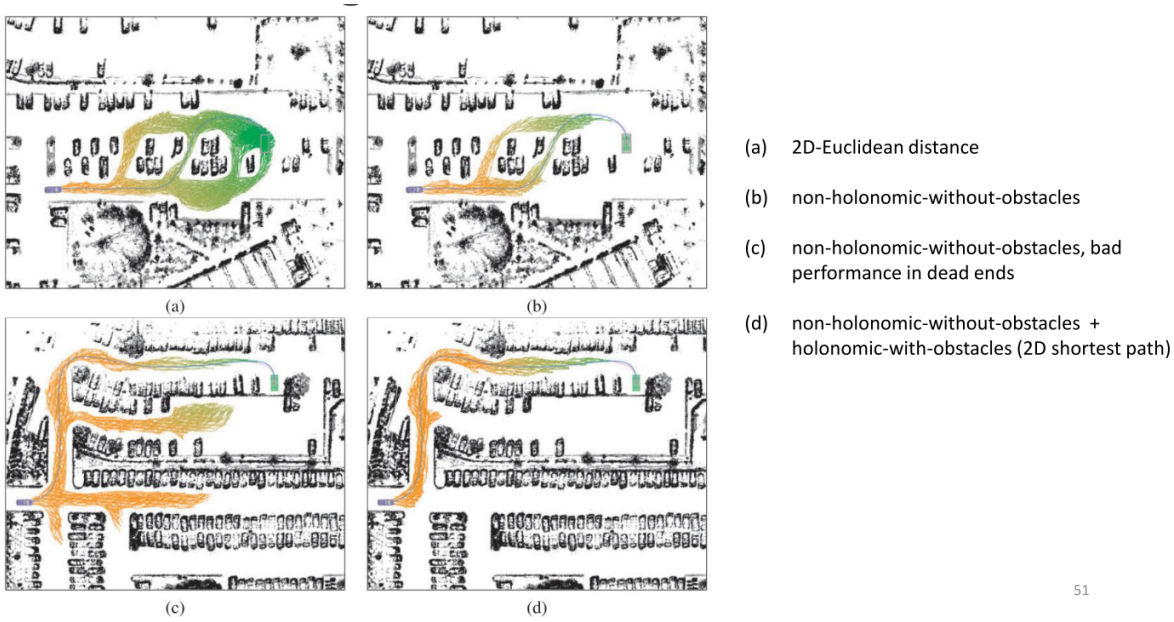


图 11: 原论文对启发函数的讨论

2.3.4 算法实现

前面说过 A* 和 A* 变种算法的实现基本一样，这里的 $HybridA*$ 实现差不多一致，Node 类需要注意的点在 $KinodynamicA*$ 的介绍中讲解过了，AStar 类中，可以使用涂色的方式做一次优化，其余的细节基本与传统 A* 实现一致，接口还是那些接口，实现还是十分简单的

2.4 情况简化与算法实现

2.4.1 可简化的条件

前面举得例子都不是真正我们在代码上使用的，只是作为让各位最低限度理解这个算法，到足够在简化后的情况下使用，维护，甚至改进的程度，当你自己看论文的时候，其实很多部分是为了轨迹才进行的计算，亦或是车辆使用自行车模型之类的才需要进行的操作，在我们的管线和机器人的机械结构上，我们有以下优点:

- 全向底盘，无需考虑方向

- 无需计算出轨迹
- 不需要十分平滑
- 无需考虑不同速度

2.4.2 简化可行的原因

全向底盘无需考虑方向这点，虽然在开的太快还是会翻车或者漂移，但是这一部分会在之后的优化通过放缩时间进行修正，无需考虑速度也是因为这部分工作交由轨迹优化进行

通过之前对寻路工作管线的了解，我们会发现在之后会通过 BSpline 拟合得到轨迹，我们只要保证得到的路径点是安全不通过障碍物的即可

无需十分平滑是通过之后的降采样，和对轨迹控制点进行优化，最后会得到一条平滑，耗能最低的轨迹，路径本身并不需要十分的平滑，但是也不可以太过于扭曲，比如传统 A* 得到的明显偏离最优路径太多的路径点

2.4.3 算法细节设计

首先通过前面的简化，我们得到状态量 X，输入量 U 和状态空间方程，其中 U 的模长 v 应该为膨胀距离的 1/3 到 1/10，这是为了以后降采样和优化考虑，详情：4.2

$$X = \begin{bmatrix} x \\ y \end{bmatrix}, U = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

$$\dot{X} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} X + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} U = U$$

可以发现这个方程十分的简单，这里我们可以顺便把离散的状态转移方程给出，依据状态转移方程，我们可以实现 Node 类的 GenerateChildren 函数

$$X = X + U \times T \tag{4}$$

由于不考虑具体速度只取速度方向和一个定长速度，因此，在原坐标系中进行控制量 U 的表示

$$U = \begin{bmatrix} v \\ \theta \end{bmatrix}$$

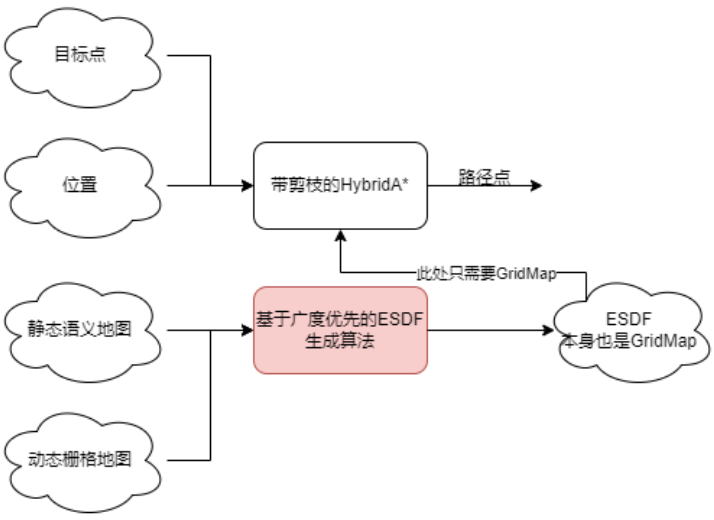
其中的 v 为定长，我们将 θ 赋值为 360° 内的 k 个值，这样我们完成了状态空间的采样

Node 类：Node 类记录下我们所需要的数据，包括每个 Node 的位置 x，y，状态更新的时间 T，和每个节点的父节点，通过公式4，进行新点的采样，需要注意的是，在赛场上会有台阶，这是单向通过道路，因此需要多一步判断，地图的网格有一部分具有方向标记，需要拓展格子的方向在方向标记的 +-90° 内才能通过，可以通过一次点乘实现

AStar 类：算法本身搜索的方式并无太大改变，不再赘述

3 导航系统基石：地图生成

地图是导航系统最基本的数据，常见的地图有栅格地图和矢量地图，不同的算法对于地图有不同的要求，比如传统 A* 以及变种算法需要正常的网格地图，NavMesh 需要使用矢量地图进行搜索，我们的目标，一方面制作出语义地图，用来保证单向通过路径的可行性，另一方面，我们需生成 EDSF（欧氏距离场），这张地图用于后续的优化器进行轨迹优化



3.1 地图内容与变换

3.1.1 地图内容约定

地图数据应该满足一定的排列方式，我们的多数内容为基于 ROS 系，因此我们地图的储存方式也按照 ROS 系的方式，以右下角为索引（0，0），前向为第一维度正方向，左向为第二维度正方向，基于数学上的习惯，将第一维度称为 X，第二维度称为 Y，如图12所示。

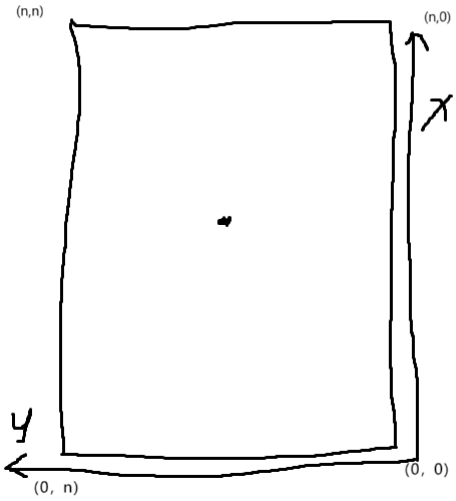


图 12: 地图坐标与索引

全局地图数据需要保存地图索引原点在世界坐标中的真实值，全局地图必然是与世界坐标轴平行，局部地图需要保证中心点和车辆重合，理应提供一张和全局坐标正方向平行的全局地图，但去年不知为何没有提供，转而提供了和机器人坐标平行的地图，因此等会会介绍地图转换方法，以备不时之需，如下图所示：

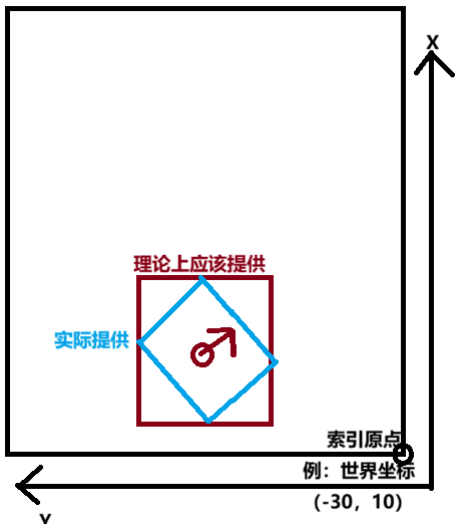


图 13: 局部地图示意

语义地图部分，由人工在静态地图构建的时候进行绘制，动态地图叠加进全局地图的时候继承，基本上可以分为以下的方式

- 绘制多份地图
- 动态生成
- 方向向量

其中，绘制多份地图然后通过位置切换使用的方式不够帅，被我抛弃，动态生成太消耗算力，且不是导航系统拿着小小栅格地图可以做的，需要在点云数据上进行处理，在 RoboMaster 并不需要，等哪天会动态出现斜坡了再说，因此我们选择使用方向向量方法，因此全局地图需要额外记录方向向量

3.1.2 局部地图索引变换

对于理想中的局部地图，即根据世界坐标排布的地图，索引变换异常简单:

$$Index_{global} = Index_{local} - \frac{Size_{local}}{2} + \frac{Position - Origin}{Resolution}$$

对于车辆坐标系的局部地图，则需要先进行旋转，而车辆具有三维坐标，因此需要先进行变换，获得车辆在 xoy 面的角度:

> 注意
>
> 这里的公式不是真正四元数旋转向量的公式，而是代码中的表现形式

原始传入的 3 维信息有 3 维坐标和一个四元数，我们需要使用这个四元数对（1，0，0）方向向量做变换，得到目前机器人在三维空间中的正向坐标

$$Forward = [1, 0, 0] * Q$$

然后通过 Forward，投影到 xoy 平面获得二维方向向量

$$Set : Forward.z = 0 \; then \; Forwrd.Normalize$$

然后通过求夹角的方程获得角度值，一般几何库会有封装这样的函数，直接调用即可，如果没有, 按照下面的公式进行求取

$$\theta = \arccos\left(\frac{v1 * v2}{len(v1)len(v2)}\right) \tag{5}$$

以上步骤在架构中的 Transform 类中实现，即天然提供平面上的角度，这是我在架构中会实现的，无需每次求取，之后需要做的事情是，直接通过这个角度，计算索引值的偏移量，算法如下

$$Index_{global} = Index_{local} - \frac{Size_{local}}{2} + \frac{Position - Origin}{Resolution} \tag{6}$$

最后如果要进行地图融合，需要先创建附件进行融合

3.2 ESDF 生成算法

3.2.1 ESDF 介绍

ESDF，中文欧式距离场，在可行区域的值代表离最近的障碍物的距离，在障碍物内部的值是最近出障碍物的距离，注意，这张地图并非用在路径规划，也就是第二章的内容，这张地图是使用在后续的优化器中，用来使路线远离障碍物所使用，ESDF 的值为-1 到 1 的小数，绝对值越接近 1，那么这个网格越远离障碍物边缘

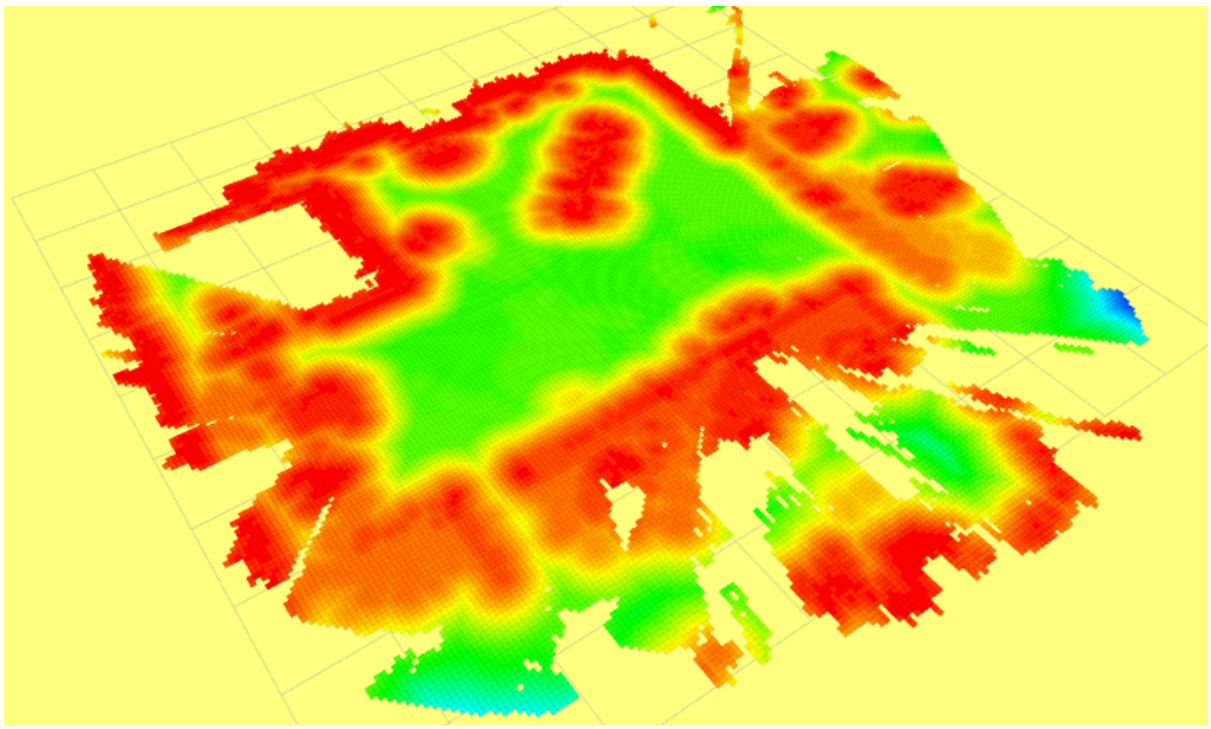


图 14: ESDF 效果

3.2.2 基于搜索的生成算法

注意

这个计算方法与传统的 ESDF 计算方法不同，为我自研算法，在网上是无法查找到的，而且也没名字，固定大小的地图中有较高的效率

算法原理为：当前点与距离最近障碍物之间不可能有其他的障碍物，且这个最近障碍物大概率与相邻格子相同，当我们迭代方向为逐步远离，那么每次获取到一个格子，这个格子记录的障碍物必然是最近的障碍物，而以记录的障碍物对外部格子进行拓展，就能获得周围格子可能最近的障碍物，此时记录到达这个格子的距离，当其他的格子也拓展到这个格子，就进行比较，这个算法同样具有 OpenList 和 CloseList 用来记录格子状态，从 OpenList 中取出来的点我们认为记录的最近障碍物就是事实上最近的障碍物，这个算法可能会存在一定的误差，但是实际使用效果极佳，误差可忽略

激光雷达获取到的局部障碍地图只有表面一层，与 ESDF 只记录表面的特性相符合，同时通过静态地图预先记录一段，这样每次更新只需要修改局部地图相关的内容即可

算法的一般步骤为：

- 获取障碍表面网格，具体为周围既有障碍物又有可行区域，加入 OpenList
- 生成障碍指向地图，每个格子默认指向自己，被搜索到的时候根据情况指向障碍物
- 从 OpenList 取出一个格子，搜索周围，搜索到时根据自身记录的最近障碍物（默认指向自己）距离和目前进行搜索的格子进行对比，如果更远就更新，如果记录的点不为障碍物也进行更新
- OpenList 使用普通队列即可，因为我们是广度优先的搜索方式

- 对于本身是障碍物的格子，格子携带的值需要加一个负数
- 设定一个最远距离 MaxEffectDistance，如果距离过远不加入 OpenList
- 重复上述步骤，直到 OpenList 为空

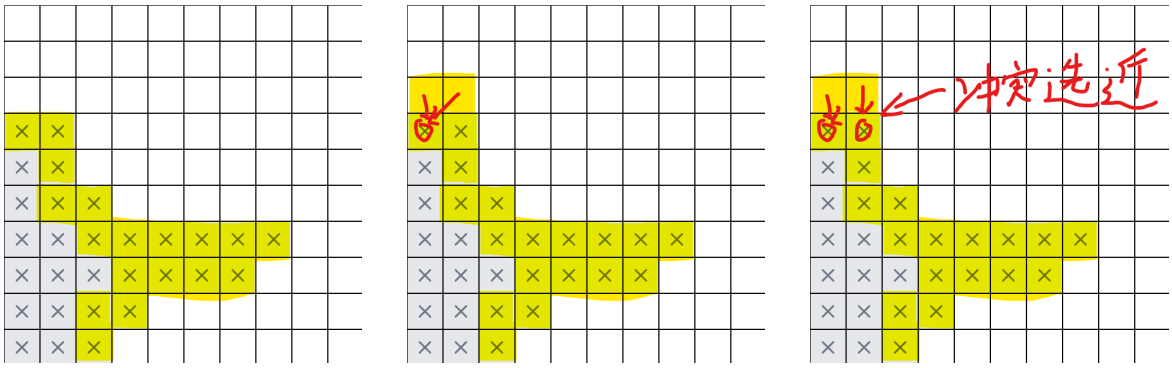


图 15: ESDF 生成

实现代码超链接，主要查看 Update 函数

3.3 语义地图

语义地图是通过标签来实现处理多种地形的算法，在 RoboMaster 中，只有台阶，因此可以简单的在某些网格添加一个方向向量来表示单向通过的路径以及方向，语义地图是新内容，可以简单的通过涂色或者任何方法生成，只要可以表示单向通过的路径，并且指明方向即可

3.4 地图膨胀的意义和距离场的意义

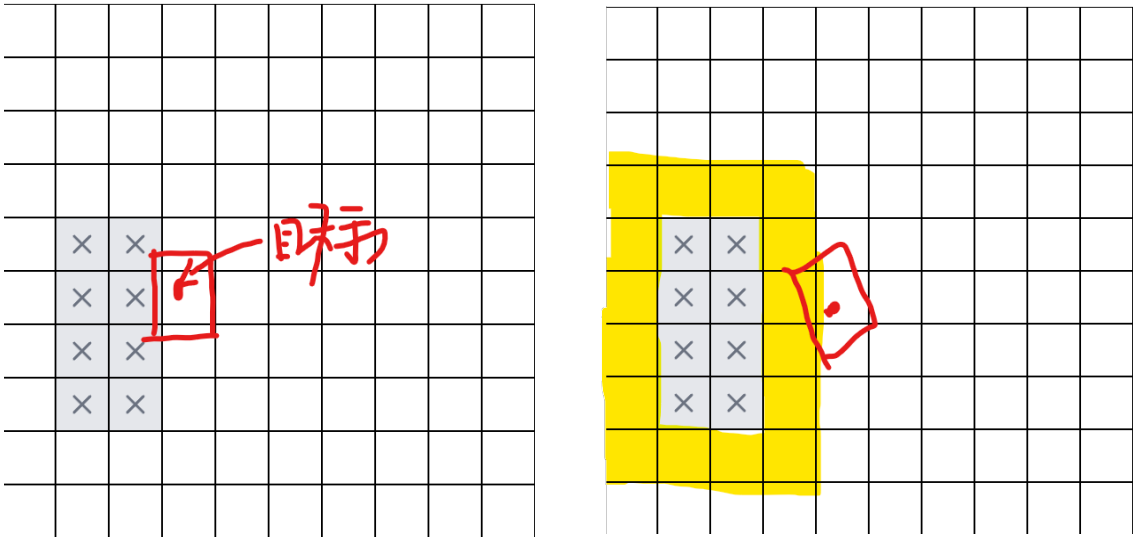
地图膨胀的意义在于防止路径规划到一个不可能的位置，也就是现实中机器人已经碰撞，但是中心位置并未碰撞，因为我们是使用中心点的位置进行路径规划的，下面两张图很好的说明了膨胀地图的作用，同理，语义地图也应该画的粗一点，因为我们只检测采样点的首尾。距离场则是为了另外的操作，他在此架构中与路径规划并无关系，虽然可以用于路径规划，膨胀作为一种硬约束，保证了路径规划得到的路径点不会经过障碍物，而之后在优化路径的时候，会根据 ESDF 来进行降采样和控制点优化，这是后面两章的内容，如果只是想要创建地图，并不需要相关知识，只要理解碰撞作为硬约束防止碰撞，ESDF 作为软约束进一步防止碰撞并且优化路径

3.5 地图文件格式

地图文件是用于保存静态地图的，关于内容的坐标系，储存方式见本章第一节，这里不在赘述，内容行与行用换行隔开，编码方式 ASCII, 内容之间空格隔开

首行：原点 X 原点 Y 网格边长 X 轴总格数 (A) Y 轴总格数 (B)

接下来 A 行 B 列数据为静态 ESDF 的值，第一个数据为索引（0，0）数据，向右第一维度增加，向下第二维度增加，也就是一行表示一个固定第二维度的所有数据，一列表示一个固定第一维度的所有数据



未膨胀　　膨胀后

图 16: 膨胀地图

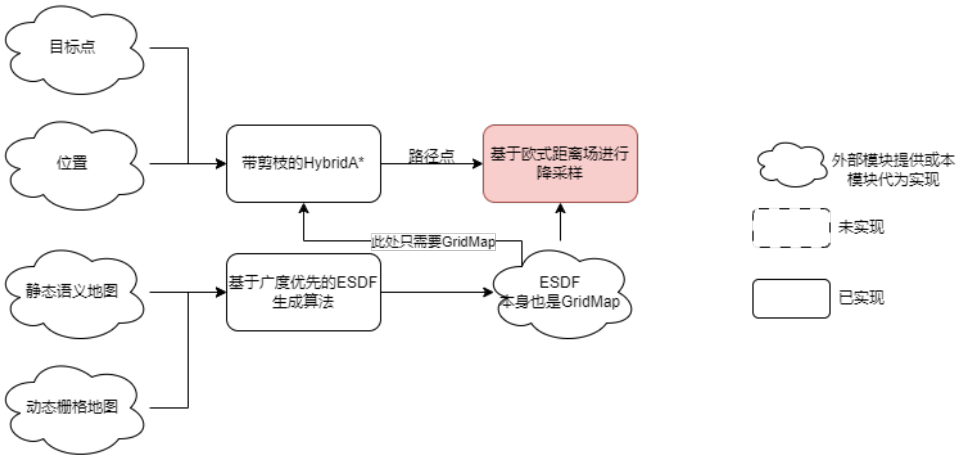
接下来 A 行 B 列数据为语义地图向量的 X 分量，第一个数据为索引（0，0）数据，向右第一维度增加，向下第二维度增加，也就是一行表示一个固定第二维度的所有数据，一列表示一个固定第一维度的所有数据，

接下来 A 行 B 列数据为语义地图向量的 Y 分量，第一个数据为索引（0，0）数据，向右第一维度增加，向下第二维度增加，也就是一行表示一个固定第二维度的所有数据，一列表示一个固定第一维度的所有数据

4 降采样算法

降采样是路径规划和轨迹优化之间的衔接步骤，理论上这一步并非必须的，但是在之后进行轨迹优化的过程中，可能会有路径点过多导致拟合优化的效率变低，因此，我们会通过一些策略进行降采样，以提升之后优化器的效果和性能

这一部分，归入优化可能会更好，但是优化器难度较大，如果路径规划的人可以理解并实现，就交给做路径规划的



4.1 算法原理

降采样的作用是在路径规划得到的路径点序列中选取合适的点，用于下一步优化，因此，我们的选取策略必然需要考虑下一步的需要，但是要明白原因，需要一些数学上的知识，这显然对大学生不太友好（虽然学下一章的时候就要全学了），好在，算法本身十分的简单，完全可以做到不需要理解就实现它

我们的想法是做一个与障碍物距离相关的采样频率，如果这一段离障碍物近，就多采样几个点，这样可以保证拟合得到的路径与障碍物无碰撞，对于远离障碍物的部分，也不可以太过稀疏，否则会有奇怪的路径出现

由于我们获取轨迹的频率要求不是很高，因此降采样的参数可以给的比较保守

4.1.1 数学表达式

约定 ESDF 地图格子 n 的值为 v_n，在值为 a 的时候采样频率为 ξ，在值为 b 的地方，其采样频率为 ζ，那么，我们可以获得在任何一个地方的采样频率（$1 >= a > b >= 0$）

$$\phi = \frac{v_n - b}{(a-b)} \times (\xi - \zeta) + \zeta \tag{7}$$

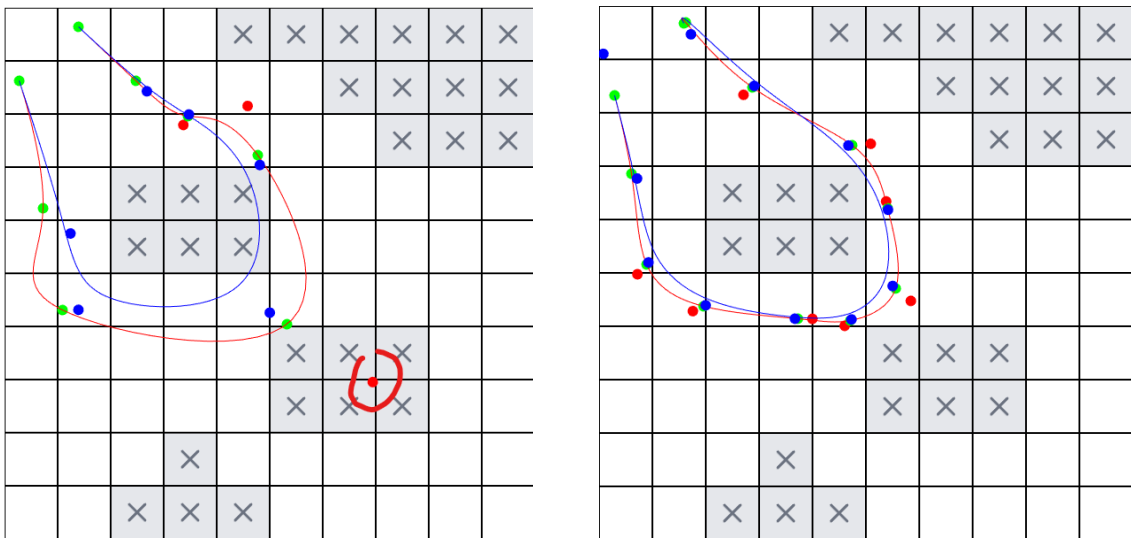
接下来实现通过采样频率进行，这边直接给出公式，如果难以理解可以往下看详解

$$\sum_{last\ sample}^{this\ sample} \frac{1}{\phi} = 1 \tag{8}$$

接下来给出 ξ 和 ζ 的值测试即可，下面有一个大致可用的经验公式，$ExpansionDistance$ 是膨胀距离，v 是 $HybridA*$ 中的状态空间，也就是速度向量长度

$$\begin{aligned} \xi &= \frac{a}{ExpansionDistance} \times v \\ \zeta &= \frac{b}{ExpansionDistance} \times v \end{aligned} \tag{9}$$

4.2 算法解释

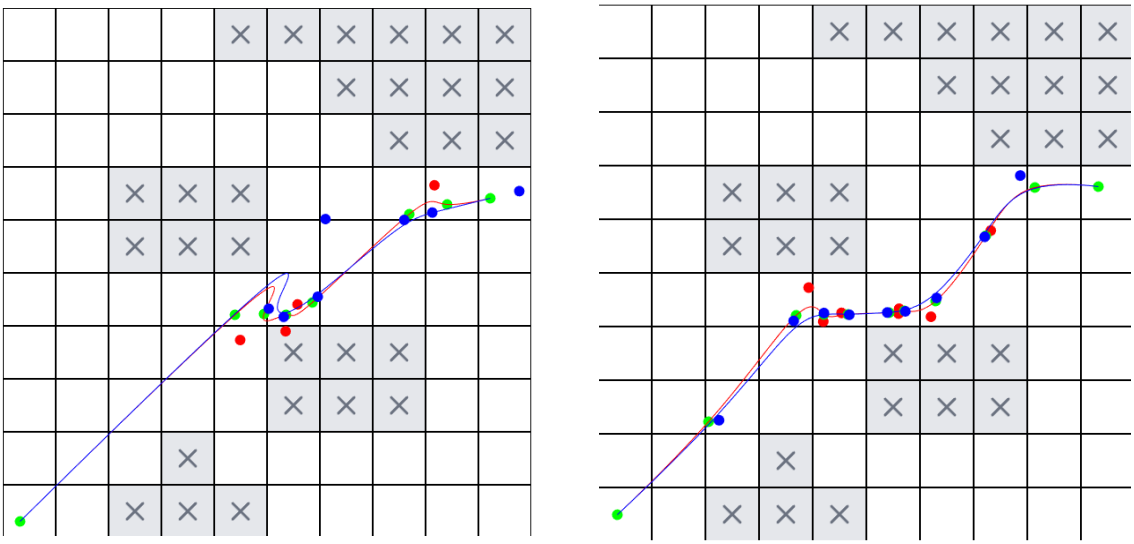


稀疏采样　　提高障碍物采样频率

图 17: 采样方法

这一小节用于缓解不理解原理写代码浑身难受的症状，由于涉及到比较复杂的数学知识，且本小节不会进行深入讲解，如果想要细想，往下阅读轨迹优化关于 BSpLine 的部分，如图，绿色为采样后的路径点，红色为原始 Bspline 控制点和轨迹，可以发现，采样频率越高的地方，控制点生成的越密集，也越靠近绿色的点（采样点）因此，在障碍密集的地方，适当提高采样频率，以此防止出现穿墙的采样点，导致下一步使用 ESDF 进行优化碰撞时，反向优化，第二张图适当提高了采样频率，不再有障碍物穿墙，蓝色点为进行弹性伸缩优化之后的点，请无视，别问，问就是懒得关绘制

第二，为何以 ESDF 最大距离作为标准进行，因为在地图中，膨胀距离约等于车体半径，是一个较为不错的长度，不会因为过于前面过于稀疏，后面过于平缓而生成扭曲的曲线 可以看到，



过于稀疏效果　　正常效果

图 18: 稀疏效果

稀疏的地方和密集的地方差距不该过大，过大的点间距会导致曲线变得十分的丑陋，这个值实际上应该自己调整，但是为了自适应，给出一份数学表达式，还记的 hybridA* 最后实现细节我们规定了 v 的大小，这便是理由，请注意，这个大小是相对量，与绝对坐标并无关系

5 求取最优路径：优化器

优化器，全文最难的部分，需要强大的现代知识作为保底，才能够比较轻松的理解 BSpline 的构造，同时带有大量抽象的优化，但是，他又是全文最重要的点之一，是我们这个规划器对于其他寻路不同的，具有优势的地方，讲真，我完全可以直接给各位一个 BSpline 求解器，事实上我已经这么做了，但是，我觉得起码要有一个人可以重复我的工作，进而达到更高，因此我将我的理论研究也写出来，都不会也没有关系，我已经做好了大部分，各位会调用即可，写到这的时候我已经神志不清胡言乱语了，无法用很好的语言讲解清除，我会尽量把知识点和为什么需要这个知识告诉各位的

警告

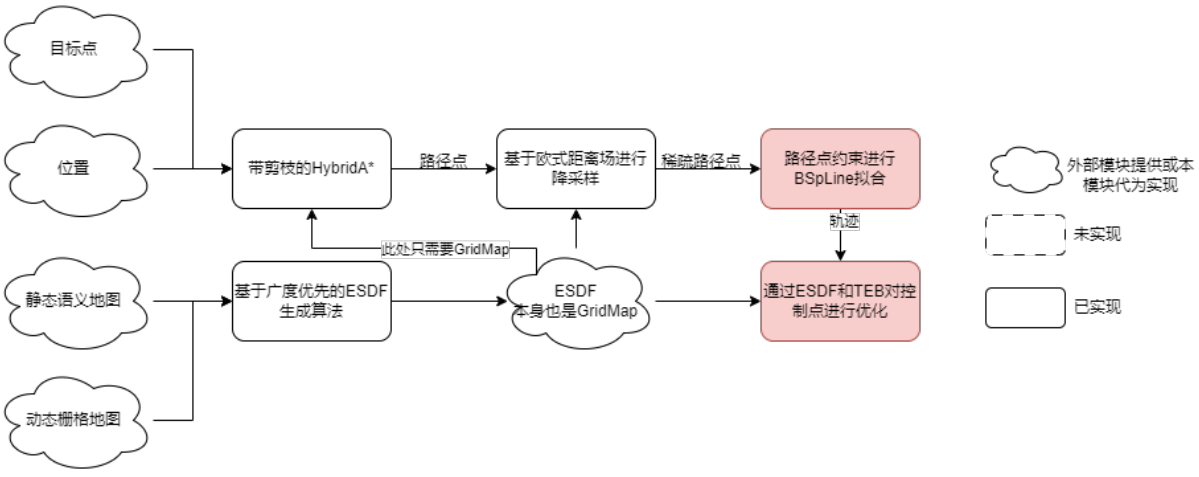
BSpline 相关的如果实在想不明白，不要死磕，先记住怎么通过约束求控制点，这是很容易的

警告

如果觉得优化器会导致怪的结果，打消你的顾虑，我们在前面降采样的一切就是为了防止优化出问题

警告

本章引用的论文原文有多处错误，以本文为主



5.1 BSpline 基础

5.1.1 de-Boor 表达式

样条曲线中，阶数 = 次数 +1

BSpline（中文 B 样条），是一种连续平滑的曲线，n 阶 B 样条具有 n 阶可导的特性，B 样条由控制点生成，基础生成的公式为

$$\begin{cases} B_{j,k}(t) & = & \frac{t-t_j}{t_{j+k-1}-t_j}B_{j,k-1} + \frac{t_{j+k}-t}{t_{j+k}-t_{j+1}}B_{j+1,k-1} \\ B_{j,1}(t) & = & \begin{cases} 1 & t \in [t_j, t_{j+1}) \\ 0 & t \notin [t_j, t_{j+1}) \end{cases} \end{cases} \tag{10}$$

公式 10 称为 De-Boor 表达式，是 Bspline 的最原始的表达形式，我知道你看的一脸懵逼，我能帮你的大概只有给你解释下符号含义，并且给你一张示意图，式子中的 $B_{j,k}$ 为基函数，第一个下标为第 j 个 t 代表的 B 样条基函数的意思，k 为 k 阶的意思，如何理解基函数? 举个例子：一组基函数

$$X = [1, x, x^2]$$

它可以表示所有的二次函数，假设有参数列表

$$C = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

那么有

$$f(x) = XC$$

而 $B_{j,k}$ 类似于 $1, x, x^2$ 根据定义，k 阶 B 样条有 k 个控制点，因此，在一组时间 T 下，确定的时间 t, 此时 t_j 为第一个小于 t 的 T 值，所有可能的控制点会取得的值的基函数为

$$B(t) = [B_{j,k}(t), B_{j+1,k}(t), \ldots, B_{j+k-1,k}(t)]$$

假设控制点序列为 $P = [p_1, p_2, \ldots, p_n]$, 此时获得的值为

$$BC \quad Or \quad \sum_{i=1}^{i=n} p_i B_{i,k}(t)$$

通过 de-Boor 表达式，我们可以得到 Bspline 的 n 阶导数递推式 (复合函数求导和数学归纳)，这

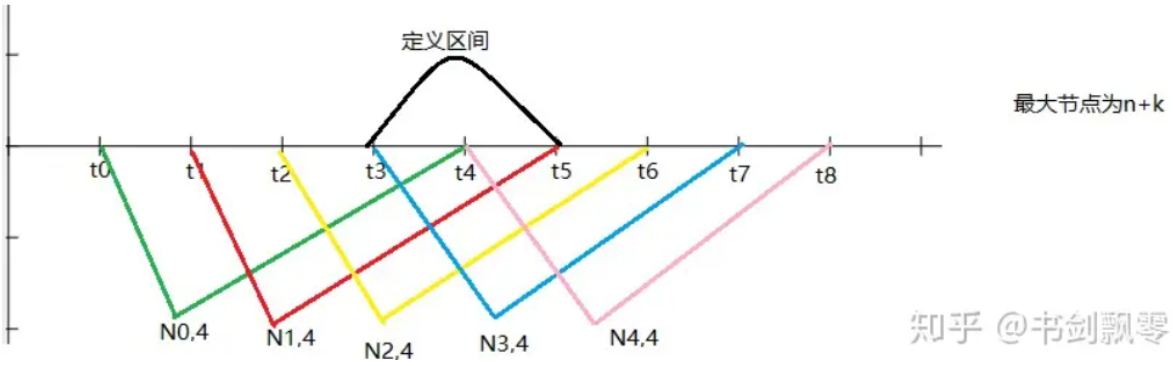


图 19: 3 阶 DeBoor 图示

里给出一阶导数

$$\begin{cases} \frac{dB_{j,k}(t)}{dt} &= \frac{k-1}{t_{j+k-1}-t_j}B_{j,k-1} - \frac{k-1}{t_{j+k}-t_{j+1}}B_{j+1,k-1} \\ B_{j,1}(t) &= \begin{cases} 1 & t \in [t_j, t_{j+1}) \\ 0 & t \notin [t_j, t_{j+1}) \end{cases} \end{cases}$$

设 B 样条关于时间 t 的曲线函数为

$$C(t) = \sum_{i=1}^{i=n} P_i B_{i,k}(t)$$

导函数为

$$\begin{aligned} \frac{dC(t)}{dt} &= \sum_{j=1}^{j=n} \frac{dB_{i,k}(t)}{dt} \\ &= \sum_{j=1}^{j=n-1} \frac{(P_{i+1}-P_i)(k-1)}{t_{j+k}-t_{j+1}}B_{j,k-1} + \frac{P_1(k-1)}{t_k-t_1}B_{1,k-1} - \frac{P_n(k-1)}{t_{n+k}-t_{n+1}}B_{n+1,k-1} \\ &= \sum_{j=1}^{j=n-1} \frac{(P_{i+1}-P_i)(k-1)}{t_{j+k}-t_{j+1}}B_{j,k-1} \end{aligned}$$

这里可以通过基函数的某些性质得到后两项为 0，但是我学识浅薄无法证明，只好拿来别人证明过的结论

因此，我们得出导数公式

$$\begin{aligned} C'(t) &= \sum_{j=1}^{j=n-1} QB_{j,k-1} \\ Q &= \sum_{j=1}^{j=n-1} \frac{(P_{i+1}-P_i)(k-1)}{t_{j+k}-t_{j+1}}B_{j,k-1} \end{aligned} \tag{11}$$

可以发现 Bspline 的导数也是 Bspline, 这一点十分重要，在之后会用到

5.1.2 BSpline 的性质

B 样条由许多性质，第一个：B 样条由多段连续函数组成，通俗理解就是在两段 B 样条的交界处，n 阶连续，即

$$\lim_{t \to 0} B_{j,k} = \lim_{t \to t_{j+k-1}} B_{j-1,k}$$

也就是说，在没有算错的情况下，B 样条天生平滑，且 n 阶可导，便于优化。第二个：B 样条曲线在其控制点闭包内, 如图 这是后期优化路径对障碍物距离的重要特性，也是我们优化器只对控制点优化的原理之一

5.2 Toeplitz 矩阵计算 B 样条

DeBoor 是一种效率极低的算法，且无法通过简单的方法由约束反向获取控制点，因此我们需要一种方法线性化 Bspline 的计算过程，将其变为控制点向量 C，时间 t 和参数矩阵 M 的乘法

本小节为承上启下的部分，之后通过路径点求解控制点也需要用到相关知识，请努力理解，论文依据：General Matrix Representations for B-Splines。

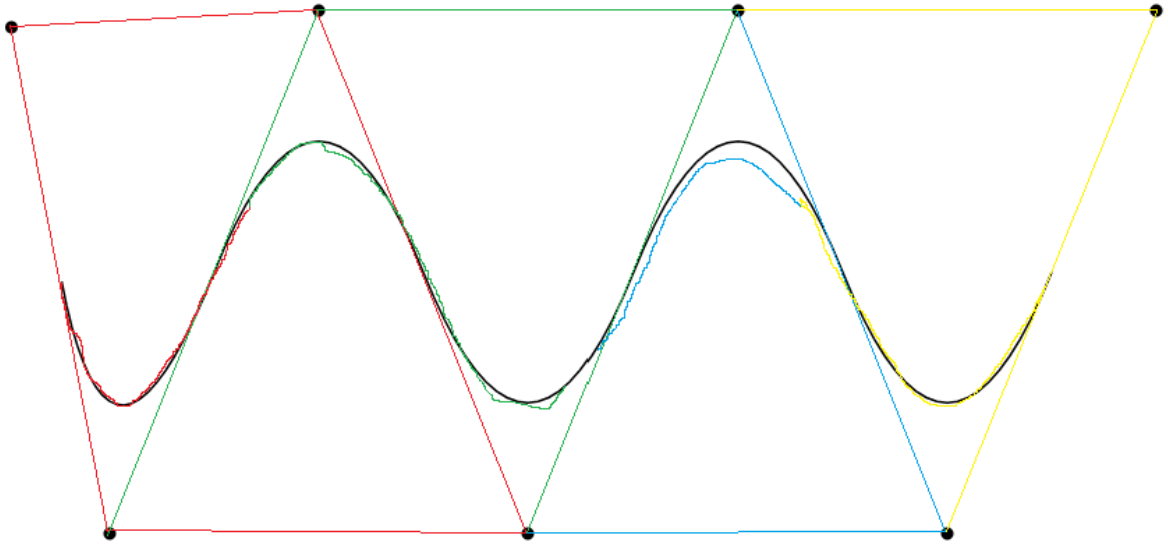


图 20: Bspline 的闭包特性

5.2.1 Toeplitz 矩阵

Toeplitz 矩阵具有与主对角线相同方向上的数字相同的特点, 有带状矩阵形式:

$$T = \begin{bmatrix} \alpha_0 & \alpha_1 & \dots & \alpha_n & 0 \\ \alpha_{-1} & \alpha_0 & \dots & ... & \\ \dots & \alpha_{-1} & \dots & \dots & \alpha n \\ \alpha_{-m} & \dots & & \dots & \dots \\ \dots & & \dots & \dots & \\ 0 & \alpha_{-m} & & \dots & \alpha_0 \end{bmatrix}$$

和下三角矩阵形式:

$$T = \begin{bmatrix} \alpha_0 & & & & 0 \\ \alpha_1 & \alpha_0 & & & \\ \dots & \alpha_1 & \dots & & \\ \alpha_m & \dots & & \dots & \\ \dots & & \dots & \dots & \\ 0 & \alpha_m & & \dots & \alpha_0 \end{bmatrix}$$

对于多项式

$$g(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$
$$h(x) = d_0 + d_1 x + d_2 x^2 + \cdots + d_m x^m$$
$$f(x) = g(x)h(x)$$

可以转化为 Toeplitz 矩阵的表示形式

$$X = [1, x, x^2, \dots, x^{m+n}]$$

$$f(x) = X \begin{bmatrix} c_0 & & & & 0 \\ c_1 & c_0 & & & \\ \dots & \alpha_1 & \dots & & \\ c_n & \dots & & \dots & \\ & \dots & \dots & \dots & \\ 0 & c_n & & \dots & c_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \dots \\ d_m \\ 0 \dots 0 \end{bmatrix} = X \begin{bmatrix} c0 & & & & 0 \\ c1 & & & c0 & \\ \dots & & & \dots & \\ \dots & & & & c_0 \\ c_n & \dots & & \dots & c_{n-n} \\ & & \dots & & \dots \\ & & & c_n & c_{n-1} \\ 0 & & & & c_n \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \dots \\ d_m \end{bmatrix}$$

5.2.2 Toeplitz 矩阵表示 Bspline

回顾公式10，对 Bspline 进行变基，由 t 空间变为幂基 u 空间 (最优化理论，老学长还在学，先记着吧，就是个归一化将基函数变为凸集的边界) 满足

$$B_{j,k-1}(u) = [1, u, u^2, ..., u^{k-2}] \begin{bmatrix} N_{0,j}^{k-1} \\ N_{1,j}^{k-1} \\ N_{2,j}^{k-1} \\ \dots \\ N_{k-2,j}^{k-1} \end{bmatrix}$$

根据公式10，每段 $B_{j,k-1}$ 由 k-1 段时间段组成，且只有在时间段内，N 不会变（把低阶基函数展开）可证，因此，我们的 u 应该在一段时间 t 内，为了计算方便，再进行放缩，以此获得一般情况下的 N，由此

$$u = \frac{t - t_i}{t_{i+1} - t_i}$$

其中,t_i 为第一个低于 t 的时间节点

这里 N 为常数，在特别的 j，k 以外，对每一段 t 也不同，查看公式10，图为 k=3 的图像, 可以发现对于第 k 阶 B 样条，需要 t 最高次为 k-1，因此在这里做出这样的假设再次我再次列出公式10，注意到

$$B_{j,k}(t) = \frac{t - t_j}{t_{j+k-1} - t_j} B_{j,k-1} + \frac{t_{j+k} - t}{t_{j+k-t_{j-1}}} B_{j+1,k-1}$$

我们用托普利兹矩阵重写 $B_{j,k}(t)$, 注意，这一步非常自然，请好好阅读托普利兹矩阵的介绍，并且对比与一般使用矩阵表示多项式的不同

对第一项做分解，可得系数 N 的递推公式：

$$
\begin{aligned}
\frac{t-t_j}{t_{j+k-1}-t_j}B_{j,k-1} = & \frac{(t_{i+1}-t_i)*u+t_i-t_j}{t_{j+k-1}-t_j} \\
= & [1,u,^2,\ldots,u^{k-2},u^{K-1}]\begin{bmatrix} N_{0,j}^{k-1} \\ N_{1,j}^{k-1} \\ \cdots \\ N_{k-2,j}^{k-1} \\ 0 \end{bmatrix}\frac{t_i-t_j}{t_{j+k-1}-t_j} \\
& +[1,u,u^2,u\ldots,u^{k-2}]\begin{bmatrix} N_{0,j}^{k-1} \\ N_{1,j}^{k-1} \\ \cdots \\ N_{k-2,j}^{k-1} \end{bmatrix}\frac{(t_{i+1}-t_i)*u}{t_{j+k-1}-t_j} \\
= & [1,u,u^2,\ldots,u^{k-2},u^{K-1}]\begin{bmatrix} N_{0,j}^{k-1} & 0 \\ N_{1,j}^{k-1} & N_{0,j}^{k-1} \\ \cdots & N_{1,j}^{k-1} \\ N_{k-2,j}^{k-1} & \cdots \\ 0 & N_{k-2,j}^{k-1} \end{bmatrix}\begin{bmatrix} \frac{t_i-t_j}{t_{j+k-1}-t_j} \\ \frac{t_{i+1}-t_i}{t_{j+k-1}-t_j} \end{bmatrix}
\end{aligned}
\tag{12}
$$

因此有

$$
\begin{aligned}
B_{j,k}(u) = & [1,u,u^2,u\ldots,u^{k-2}] \\
& \left(\begin{bmatrix} N_{0,j}^{k-1} & 0 \\ N_{1,j}^{k-1} & N_{0,j}^{k-1} \\ \cdots & N_{1,j}^{k-1} \\ N_{k-2,j}^{k-1} & \cdots \\ 0 & N_{k-2,j}^{k-1} \end{bmatrix}\begin{bmatrix} \frac{t_j-t_j}{t_{j+k-1}-t_j} \\ \frac{t_{i+1}-t_i}{t_{j+k-1}-t_j} \end{bmatrix}+\begin{bmatrix} N_{0,j}^{k} & 0 \\ N_{1,j}^{k} & N_{0,j}^{k} \\ \cdots & N_{1,j}^{k} \\ N_{k-2,j}^{k} & \cdots \\ 0 & N_{k-2,j}^{k} \end{bmatrix}\begin{bmatrix} \frac{t_i-t_j}{t_{j+k-1}-t_j} \\ \frac{t_{i+1}-t_i}{t_{j+k-1}-t_j} \end{bmatrix}\right)
\end{aligned}
\tag{13}
$$

引用第一小节的内容，对于某个时间点 t, 有 $t \in [t_i, t_{i+1})$, 控制点集合 $V = [v_1, v_2, \ldots, v_{k-1}]$, 其 k 阶 Bspline 的值为

$$
\begin{aligned}
C_k(t) = & [B_ik(u), B_{i+1}k(u)], \ldots, B_{i+k-1}k(u)]V \\
= & [1,u,u^2,\ldots,u^{k-1}]M^k(i)V \\
\\
u = & \frac{t-t_i}{t_{i+1}-t_i} \\
\\
where & \quad t(i+1) > u \geq t(i)
\end{aligned}
\tag{14}
$$

$$
M^k(i) = \begin{bmatrix} N_{0,i}^K & N_{0,i+1}^K & \cdots & N_{0,i+k-1}^K \\ N_{1,i}^K & N_{1,i+1}^K & \cdots & N_{0,i+k-1}^K \\ \cdots & \cdots & \cdots & \cdots \\ N_{k-1,i}^K & N_{k-1,i+1}^K & \ldots^K & N_{k-1,i+k-1}^K \end{bmatrix}
$$

求此矩阵的方法为利用公式12不断展开 N, 即可得到 $M^k(i)$ 与 $M^{k-1}(i)$ 的递推公式，这一步计算量大且思维量小，本文不再赘述，这边给出 $M^k(i)$ 的结论：在均匀 B 样条的情况下，时间节

点间距相等，那么

$$\begin{bmatrix} \frac{t_i - t_j}{t_{j+k-1} - t_j} \\ \frac{t_{i+1} - t_i}{t_{j+k-1} - t_j} \end{bmatrix} = \begin{bmatrix} \frac{i-j}{k-1} \\ \frac{1}{k-1} \end{bmatrix}$$

有 M_k 与 i 无关，因为求出的 M_k 递推式与 i 有关的部分全可以消除，如 $t_{i+1} - t_i = 1$，所以，我们可以直接通过已知的 M_k 求出均匀 B 样条在某个时间 t 的值

5.3 求值与求导数

5.3.1 均匀 B 样条

在解出控制点之后，我们需要获取连续空间上的值和导数，这一节注意下标，我们一直以来是当前点往后延申的算法，我们在最开始只有 t，和时间序列 T，我们需要计算 t 的位置和 u 的值，这一步用数学表达式意义不大，故使用类 C（C，C++，C#）代码表述

```
1    t = Math.Min(Math.Max(t, _t[0]), _t[n_ - 1]);
2    int k = 0;
3    while (_t[k + 1] < t) k++;
```

上述代码中，k 代表的是 i，_t 是时间序列，t 是实际的时间, 以此求得 i，而 u 只要根据 u 的定义求解即可

$$u = \frac{t - t_i}{t_{i+1} - t_i}$$

求解值

$$y = [1, u, u^2, u^3] M^4 X.Row(i : i + 3)$$

求解导数

$$\dot{y} = [0, 1, 2u, 3u^2] M^4 X.Row(i : i + 3)$$
$$\ddot{y} = [0, 0, 2, 6u] M^4 X.Row(i : i + 3)$$

5.3.2 非均匀 B 样条

非均匀 B 样条求解难度较大，且有首尾特殊情况需要考虑，首先考虑下面的系数矩阵

$$M^4(i) = \begin{bmatrix} \frac{(t_{i+1}-t_i)^2}{(t_{i+1}-t_{i-1})(t_{i+1}-t_{i-2}} & 1 - m_{0,0} - m_{0.2} & \frac{(t_{i+1}-t_i)^2}{(t_{i+2}-t_{i-1})(t_{i+1}-t_{i-1})} & 0 \\ -3m_{0,0} & 3m_{0,0} - m_{1,2} & \frac{3(t_{i+1}-t_i)(t_i-t_{i-1})}{(t_{i+2}-t_{i-1})(t_{i+1}-t_{i-1})} & 0 \\ 3m_{0,0} & -3m_{0,0} - m_{2,2} & \frac{3(t_{i+1}-t_i)^2}{(t_{i+2}-t_{i-1})(t_{i+1}-t_{i-1})} & 0 \\ -m_{0,0} & m_{0,0} - m_{3,2} - m_{3,3} & m_{3,2} & \frac{(t_{i+1}-t_i)^2}{(t_{i+3}-t_i)(t_{i+2}-t_i)} \end{bmatrix}$$

$m_{3,2} = \frac{-m_{2,2}}{3} - m_{3,3} - \frac{(t_{i+1}-t_i)^2}{(t_{i+2}-t_i)(t_{i+2}-t_{i-1})}$

$m_{i,j}$为 i 行 j 列

(15)

可以发现时间 t 与前 2，后 2 有关，也就是有 5 个时间点与非均匀 B 样条的值有关，而在首尾部分，因为我们进行控制点求取的时候使用了均匀 B 样条的系数矩阵，如果想要起始点和终点没有偏差，必须要使用均匀 B 样条对于非均匀 B 样条，时间节点的改变并不会改变曲线形状，只会改变每个时刻抵达的位置，速度，或者加速度 上图是非均匀 B 样条速度采样对比，绿色是固

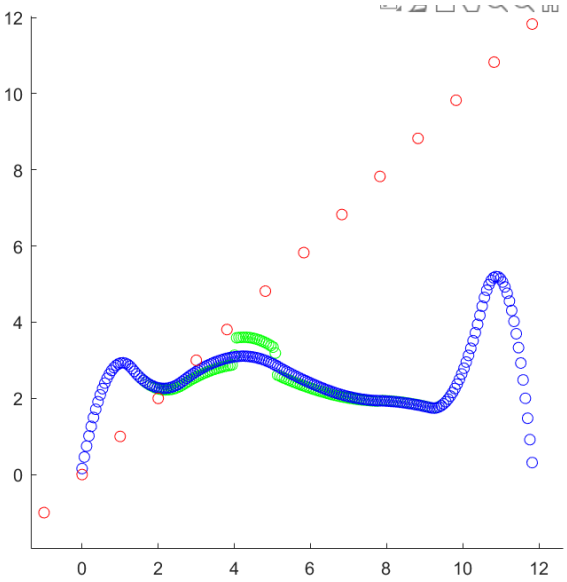


图 21: 非均匀 B 样条速度采样对比

定使用均匀 B 样条求非均匀 B 样条的速度，可以发现速度在时间节点处突变，而使用 $M^4(i)$ 作为系数矩阵的时候速度连续分布，在之后我们会对每段 B 样条重新分配时间，这是为了最小化运行时间, 因此，求取非均匀 B 样条是必要的

现在我们来求取非均匀 B 样条，非均匀 B 样条无法想均匀 B 样条一样对 u 向量简单求导，因为 u 和 $M^4(i)$ 都是与 t 相关的函数，我们实际上在求导 u 的时候，做的是 $\frac{df(u)}{du}$，但是对非均匀 B 样条而言，求导得到的并非简单的 $[\ 0 \quad 1 \quad 2u \quad 3u^2\ ]M^4(i)$（为了之后表示方便，我们把 $[\ 0 \quad 1 \quad 2u \quad 3u^2\ ]$ 称为 $\vec{u}^{-1}$）而是 $\vec{u}^{-1}M^4(i) + \vec{u}\frac{dM^4(i)}{dt}\frac{dt(u)}{du}$，因此我们需要回归 $deBoor$ 表达式，回顾公式11，k 阶 B 样条的一阶导数为 k-1 阶 B 样条，且控制点只需要简单进行计算即可得到，我们以一阶导数为例

首先查看系数矩阵

$$M^3(i) = \begin{bmatrix} \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}} & \frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}} & 0 \\ -2m_{0,0} & 2m_{0,0} & 0 \\ m_{0,0} & -m_{0,0} - m22 & \frac{t_{i+1}-t_i}{t_{i+2}-t_i} \end{bmatrix} \tag{16}$$

对控制点进行重新计算，

$$Q = \sum_{j=1}^{j=n-1} \frac{(P_{i+1} - P_i)(k-1)}{t_{j+k} - t_{j+1}} B_{j,k-1}$$

注意时间序列，一个方便的方法就是，0 索引从时间点为 0 的地方开始 上图为一阶导数结果，蓝色为差分位置得到的速度，绿色为通过上述算法得到的速度，可以发现，绿色与蓝色几乎重合，

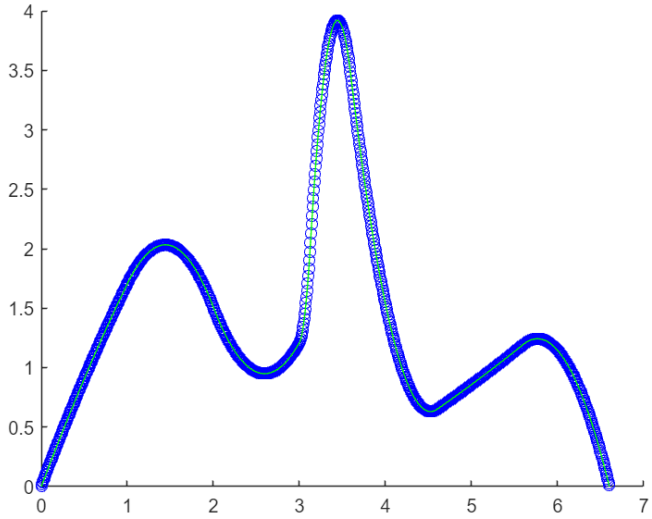


图 22: 4 阶 B 样条一阶导数

且之后积分一阶导数得到的数值结果为轨迹末端与前端之差

5.3.3 算法实现（类 C）

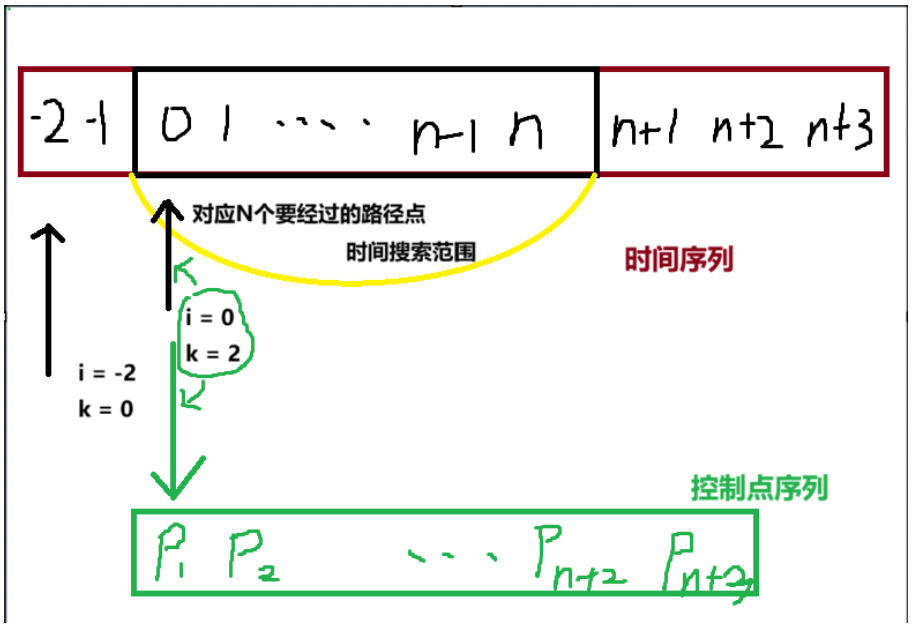


图 23: 理论与实际储存位置

观察前面的系数矩阵 $M^4(i)$ 你会发现有一项 t_{i-2}，我们都知道，t 为 0 的时候为 i=0, 这样才能够求出正确的解，但是现在出现了 i - 2 项，因此，我们需要变动我们的 0 点，因此，时间序列真正的初始值其实在下标为 2 的时候

上图中，黑色部分为路径点所对应的时间节点，红色为我们真正需要的时间节点，在内存中起始点为 0，因此用 k 作为时间节点数组的索引，k 初始值为 2，并搜索到 2+n 处，从时间 t 如何确认对应 k 已经在上面列过，不再赘述，需要注意的是，代码中 k 初值应该为 2，且在下面进

行 k<=2+n 的判断，从图中可以看到 i 和 k 之间的关系，采用 i 作为控制点序列的索引值，一次从序列中取出阶数个点，如 4 阶 B 样条取 4 个点，关于代码实现的细节请看项目中代码

从上面，我们定义了 i 和 k，作为控制点和时间序列的索引，现在我们来看怎么求解和使用 $M^k(i)$ 注意这里 k 指阶数，其中 i 代表的意思是给从索引值为 i 开始的控制点序列使用的系数矩阵，而文中指的是时间序列的索引，因此，这里的 i 应该使用代码中的 k 代替

5.4 通过 MinimumAcc 获取控制点

> 注意
>
> M 矩阵已在附录列出

5.4.1 快速获取节点处的物理量

观察均匀 B 样条的 M^k 矩阵，会发现这是个常数矩阵，在节点处，套用公式

$$C(t) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \cdot M^k \cdot \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

当在节点处，u = 0 ，因此 $C(t) = M^k.Row(0) \cdot X$，由此我们构建一个线性方程组，

$$AX = b$$

其中 b 的每一行代表一个物理量，X 为控制点

有

$$A_{i,j:j+4} = M^k.Row(0)$$

代表 b 第 i 行为 j 到 j+3 控制点序列所控制的位置类似的，可以推出 b 第 i 行为 j 到 j+3 控制点序列所控制的速度和加速度的方程

$$\begin{cases} A_{i,j:j+4} = M^k.Row(0) \quad as \quad Position \\ A_{i,j:j+4} = M^k.Row(1) \quad as \quad velocity \\ A_{i,j:j+4} = M^k.Row(2) \quad as \quad ac1c \end{cases} \tag{17}$$

由此，我们有任意位置任意物理量的等式

5.4.2 构造二次规划目标函数

我们现在为了得到在速度上变化小的曲线控制点，以及路径跟踪过程中最小的能量消耗（与加速度成正比），使用优化的方法获取控制点计算第一段曲线的加速度绝对值之和

$$\begin{aligned} J(t) = & \int_0^1 ([0,0,2,6u] \cdot M_K \cdot X)^2 du \\ = & \int_0^1 ([0,0,2,6u] \cdot M_K \cdot X) \cdot ([0,0,2,6u] \cdot M_K \cdot X) \end{aligned}$$

不难看出

$$Rank([0,0,2,6u] \cdot M_K \cdot X) == 1$$

于是有

$$[0,0,2,6u]\cdot M_K\cdot X=([0,0,2,6u]\cdot M_K\cdot X)^T$$

带回原始公式

$$\begin{aligned} J(t)= & \int_0^1(X^TM^{k^T}[0,0,2,6u]^T)\cdot([0,0,2,6u]\cdot M_K\cdot X)^2 \\ = & X^THX \end{aligned} \tag{18}$$

$J(t)$ 就是我们的目标二次型函数，

5.4.3 构造路径硬约束

我懒得画图了，想象你的路径点周围有一堆的方框，这些方框是无碰撞的，那么只要我们的控制点所对应节点的位置在方框内，那么整条曲线在大多数位置都是无碰撞的那么方框是怎么确定的，我们可以反向思考，在地图膨胀的时候多膨胀一点，那么 HybridA* 找到的路径点必然是有一段无碰撞区间的只要求最大碰撞距离为半径的圆的最大内包矩形边长就可以了

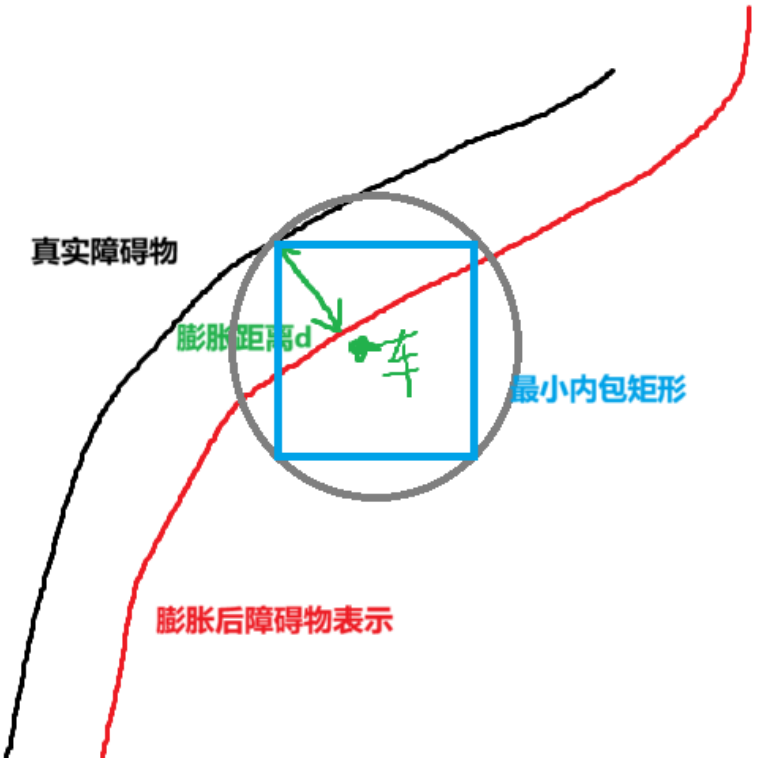


图 24: 硬约束

如图所示，因为场地障碍物并非横平竖直，因此我们只能以膨胀距离 d 为半径的圆的最小内包矩形作为我们的可行域区间，但是在优化时，我们只限制中心点位置，因此，我们需要进一步处理 dx,xy

$$\begin{cases} dx= & \sqrt{2}d-r \\ dy= & \sqrt{2}d-r \end{cases} \tag{19}$$

图 25: Matlab 二次规划函数

其中 r 为车辆半径，这里按照 Matlab 的二次规划符号进行编写

$$A = \begin{bmatrix} M_k Row(1) \\ \dots \\ \dots \\ M_k Row(1) \\ -M_k Row(1) \\ \dots \\ \dots \\ -M_k Row(1) \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \} n \\ \\ \\ \\ \} n \\ \\ \\ \end{matrix} \quad b = \begin{bmatrix} x_1 + dx \\ x_2 + dx \\ \dots \\ x_n + dx \\ -x_1 + dx \\ -x_2 + dx \\ \dots \\ -x_n + dx \end{bmatrix}$$

由于问题规模较小，大约有 100—200 个变量和双倍的约束，因此我们选择 Goldfarb-Idnani 方法

5.5 时间重分配

根据前面的分析，4 非均匀 B 样条每一段轨迹由上下 5 个时间约束，对于一个超时的时间，我们对这段时间相对应的变化率，得到新的时间，但是，因为每一段时间都是会影响多段速度，因此如果直接以一个大的数进行缩放，那么最后的结果必然是不好的

这里老学长已经不知道怎么跟各位解释这件事情了，我们直接看代码

```
function TimeLine = ReallocTimeLine(BsplineControlPoint,TimeLine)
v_limit = 6;
a_limit = 12;
ratio_limit = 1.01;
ControlPoint = zeros(size(BsplineControlPoint,1) - 1,size(
   BsplineControlPoint,2));
velfea = false;
while 1==1
    if(~velfea)
        for i = 1:size(BsplineControlPoint,1)-1
```

```
10            ControlPoint(i,:) = 3*(BsplineControlPoint(i+1,:) -
                  BsplineControlPoint(i,:))./(TimeLine( i + 3) -
                  TimeLine(i));
11        end
12        v_max = 0;
13        index = -1;
14        for k = 3:size(TimeLine,2)-3
15            m_00 = (TimeLine(k+1) - TimeLine(k))/(TimeLine(k+1) -
                  TimeLine(k-1));
16            m_01 = (TimeLine(k) - TimeLine(k-1))/(TimeLine(k+1) -
                  TimeLine(k-1));
17            v = [m_00,m_01]*ControlPoint(k-2:k-1,:);
18            v = sqrt(v*v');
19            if v > v_max
20                index = k;
21                v_max = v;
22            end
23        end
24        if v_max < v_limit
25            velfea = true;
26        end
27
28        i = index;
29
30        ratio = min(ratio_limit,v_max/v_limit)+1e-4;
31        t_old = TimeLine(i+2) - TimeLine(i-2);
32        t_new = ratio * t_old;
33        t_int = t_new / 4;
34
35        head = max(1,i-1);
36        tail = min(i+2,size(TimeLine ,2));
37        for j = head:tail
38            TimeLine(j) = t_int + TimeLine(j-1);
39        end
40        for j = tail+1:size(TimeLine ,2)
41            TimeLine(j) = (t_new - t_old) + TimeLine(j);
42        end
43    end
```

```
44
45      a_max = 0;
46      for i = 1:size(BsplineControlPoint,1)-1
47          ControlPoint(i,:) = 3*(BsplineControlPoint(i+1,:) -
               BsplineControlPoint(i,:))./(TimeLine( i + 3) - TimeLine(i
               ));
48      end
49      for i = 1:size(BsplineControlPoint,1)-2
50          ControlPoint(i,:) = 2*(ControlPoint(i+1,:) - ControlPoint(i
               ,:))./(TimeLine( i + 2) - TimeLine(i));
51      end
52
53      for k = 3:size(TimeLine,2)-4
54          a = ControlPoint(k-2,:);
55          a = sqrt(a*a');
56          if a > a_max
57                  index = k;
58                  a_max = a;
59          end
60      end
61      if a_max < a_limit
62          break;
63      end
64
65      ratio = min(ratio_limit,a_max/a_limit)+1e-4;
66      i = index;
67
68      t_old = TimeLine(i+2) - TimeLine(i-2);
69      t_new = (ratio)* t_old;
70      t_int = t_new / 4;
71
72      head = max(1,i-1);
73      tail = min(i+2,size(TimeLine ,2));
74      for j = head:tail
75          TimeLine(j) = t_int + TimeLine(j-1);
76      end
77      for j = tail+1:size(TimeLine ,2)
78          TimeLine(j) = (t_new - t_old) + TimeLine(j);
```

```
    end
end
if TimeLine(3) ~= 0
    diff = TimeLine(3);
    for j = 1:size(TimeLine ,2)
        TimeLine(j) = TimeLine(j) - diff;
    end

end
end
```

你看，很简单对吧，只要把之前的知识结合起来，就可以得到这个重分配算法啦, 不懂的线下问我吧，看不懂不怪你们，是我真的不知道咋讲，不会的多来问我
qq:1780284652 ID：Alray

5.6 控制点优化

5.6.1 障碍物优化

第一步，我们先对控制点进行推离障碍物的操作，对于每个控制点，有当前梯度向量 $\vec{v}$ 和当前 ESDF 取值 s, 根据公式

$$\begin{cases} f = \vec{v}h(s) \\ h(s) = (1-s)^2 \end{cases} \tag{20}$$

将 f 加上控制点即可以获取优化后的控制点

5.6.2 平滑优化

仿照 TEB 算法，每个控制点会对相邻的控制点施加一个弹簧力，也就是满足

$$f = kx$$

的力，其中 x 为控制点之间的距离，一般平滑优化会在障碍物优化之后，这是经验得到的结论，并且我们会反复优化几遍，以逼近最优

5.6.3 非线性优化库优化

> 注意
>
> 这一部分只是补充，我们实际上不用

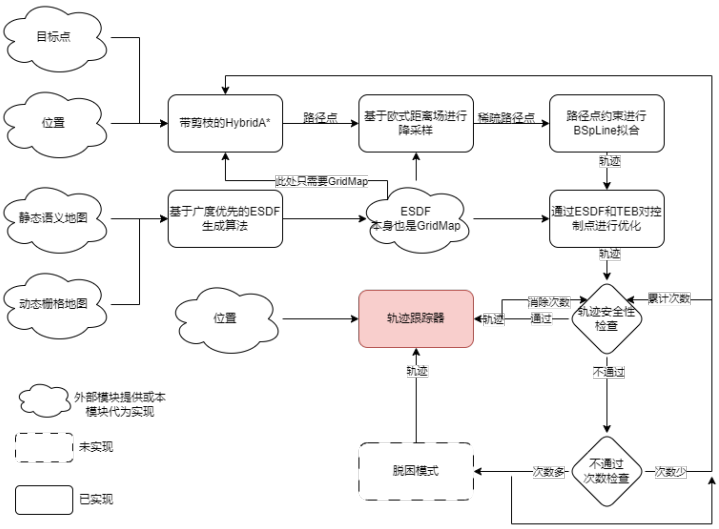
对 Bspline 进行采样，得到的每个点在 ESDF 上都有一个权值，这是障碍物乘法 f_1，对 Bspline 求 jerk，有 $f_2 = \sum Jerk$，这是平滑约束，将以上函数相加得到总的惩罚函数，放入非线性优化库中对控制点进行优化，求得最好的控制点，这个方法运行效率较低，且提升不大，因此不用

5.7 非法路径修正

前面有降采样对吧，这条路径虽然在节点处一定安全，但是可能会拓展出去，导致路径存在碰撞，这时候只要把所有有碰撞的轨迹节点之间加上之前搜索过的一个路径点就可以了,时间节点直接中间插入节点就可以了

6 MPC 轨迹跟踪

老学长要应付你们了,看他,不懂的来问我 $https://blog.csdn.net/qq_37705385/article/details/139030062$
仿真环境也给你们搭建好了https://github.com/AlrayQiu/PlannerLearning.git
我宣布，这个文档结束了，老学长写不下去了



7 附录

7.1 表 1：M 矩阵

$$M^1 = \begin{bmatrix} 1 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

$$M^3 = \frac{1}{2!} \begin{bmatrix} 1 & 1 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix}$$

$$M^4 = \frac{1}{3!} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$