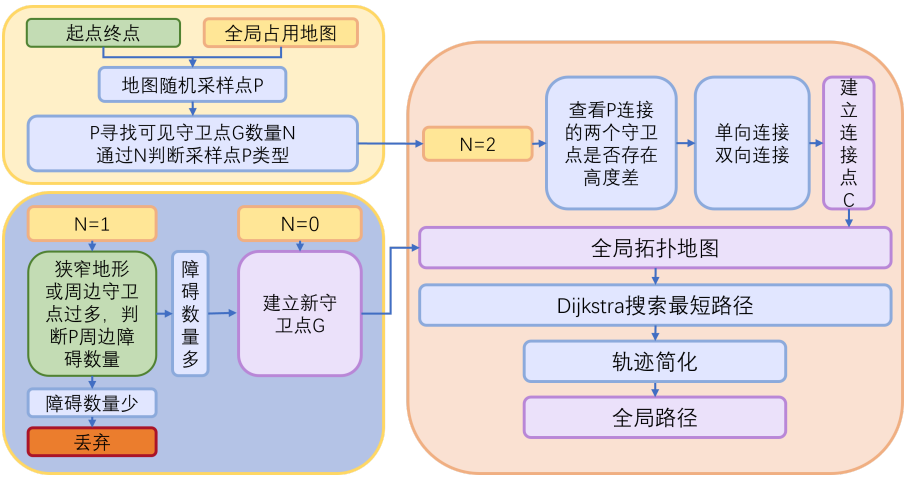
哨兵开源技术文档

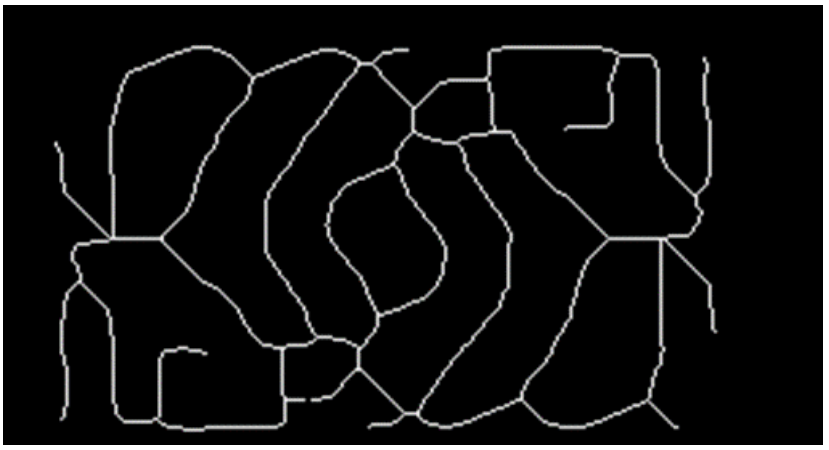
1. 规划轨迹生成部分—拓扑路径搜索：

拓扑路径搜索重要的是找到从起点到终点的**最短路径**,以及从起点到终点的**所有可行路径**，场地可通行路径太多，从A到B点的路径很多，23赛季只通过A*的搜索可以做到最优，但是只是在A*规定的**启发式函数下的最优**，而不是我们理解的路径最短，且在不同任务要求下可能存在寻找多条可达路径的任务，因此我们用topo路径搜索作为全局路径搜索的算法(参考fast-planner)：



全局路径搜索流程

拓扑路径搜索通过构建全局拓扑路径连接图来找到近似最短路径。我们定义两种节点：**守卫点**(Guard)和**连接点**(Connection)，守卫点负责探索，而连接点用于连接两个守卫点。两个不同的守卫点彼此不可见或距离较远。连接点只能连接两个守卫点，无其他相邻连接点。守卫点可同时连接多个连接点但不可与其他守卫点相连。同时每个点还具有一个**相邻但无法连接**的**数据点集用于**记录如台阶等单向连接边。为了提高整体的地图的采样率，我们对栅格地图进行了**骨架提取**，得到了整体的车道中心线图如下图。意味着我们在采样点时只针对车道中心线进行采样，这样可以提高整体地图拓扑结构的采样效率(实际调试发现车道中心线非常有必要，全图采样效率还是挺低的)

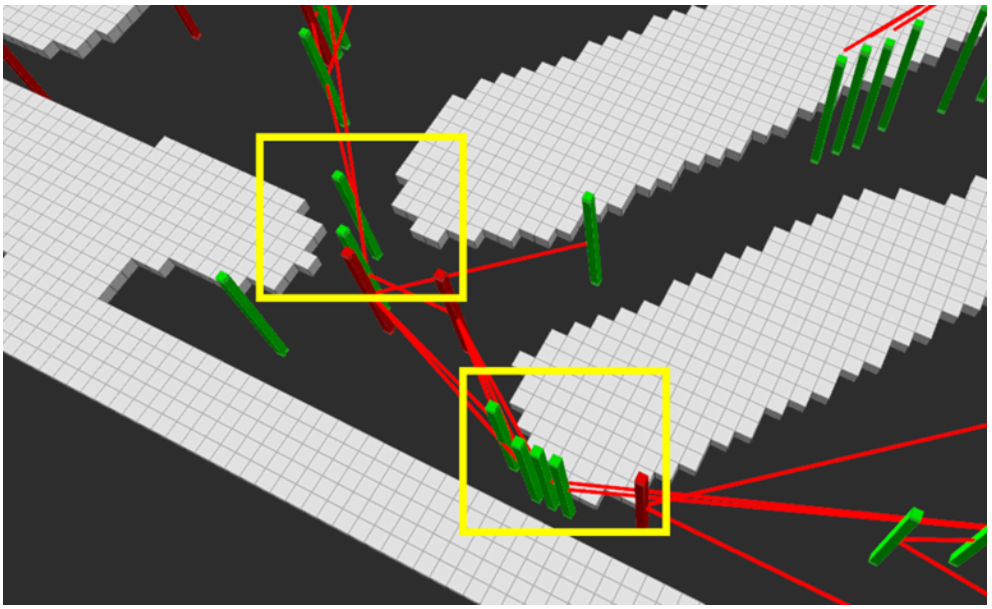


车道中心线地图

采样得到新点p后对拓扑图中的所有守卫点进行是否可见判断：连接新采样点p与拓扑图中的某守卫点 p2，判断两点之间是否有**障碍物**，如果没有障碍则进行**初步高度判断，**连接两点并进行步进采样，若采样的前后两点未出现**过大的高度差**，则认为两点直接可见，不满足上述条件的任一条件就认为两点不可见。

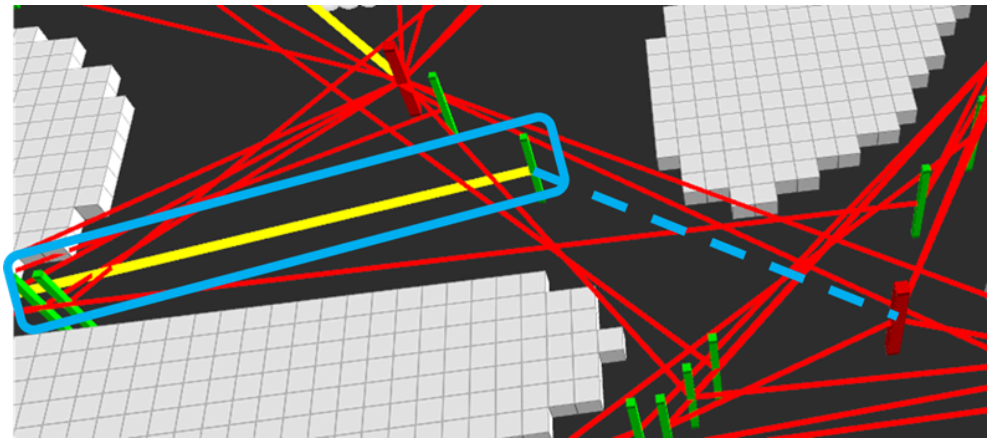
对于采样得到的新点p，我们将拓扑图中所有**守卫点**都与其进行判断，如果存在两个可见点Q1Q2则进行一次**连接性判断**，在连接性判断终如果Q1Q2已经由一个连接点连接起来了，则认为新采样点p产生的该次连接**冗余**，则扔掉其中一个点，继续进行拓扑可见性判断，且可见点对变量**exist_visual加一**；如果两个守卫点从未被连接过，则终止判断并返回，将该采样点p作为**连接点连接两个守卫点**，并将采样点p加入拓扑图。若对拓扑图的所有守卫点进行是否可见判断数量始终为0，则认为该点不与任何其他点连接, 新采样点p能代表该片区域内的拓扑结构，因此其可以作为新的守卫点加入拓扑图。

若迭代到最后只有一个守卫点能与新采样点p相连，则需要进行是否建立守卫点的判断处理, 若可见性判断结束后**exist_visual**>2,相当于在可见性判断中该新采样点**重复多次**与已经连接过的守卫点相连，则认为该区域内存在多对守卫点可以代表该片区域的地图拓扑结构，因此新采样点属于**冗余点，需舍弃；**反之则搜索该采样点p邻域内占用栅格的数量，若占用栅格数量较多，则认为该点周围障碍物过多，**属于狭窄路口或不易扩展到的桥洞**等地形, 如下图。因此在该采样点直接建立新的守卫点并加入拓扑图中 （通过这些方法拓扑地图结构的同时又不忽略掉部分特殊地形，可以做到高效全面的获取全地图拓扑结构，下图狭窄地形拓扑搜索，红色点代表守卫点，绿色点为连接点，黄框表示狭窄路段）



我们进行可见性检查判断后，进行下一步：**高度检查。**在建立连接点时，新连接点p与两个守卫点p1p2需要进行高度连接性检查，具体为连接p与p1 ,分别从p步进采样到p1以及从p1步进采样到p，步进时检查当前点的高度与上一采样点的高度之差，　如果从p到p1的搜索过程中出现高度差大于一定值且当前点高度小于上一采样点，则认为p到p1的高度检查过程中遇到了不可通行的地形突变点(如**上行台阶**)，则不将p1放入p的邻接点集中，反之若高度差处于一定范围内但当前点高度大于上一采样点, 则认为此处为可通行的地形突变点(如台阶，飞坡点)，通过高度检查，我们可以在搜索的过程中就可以避免上台阶和反向飞坡等类似的不可进行的动作，反之可以轻易做出下台阶和飞坡等动作，具有更好的**地形适应能力（如图2.2.4所示，蓝色虚线表示该连接点未与守卫点相连。即从连接点到连接终端守卫点步进遍历，在台阶处出现了高度差大于阈值，被判定为不可通行连接，因此该连接点并**

没有与守卫点相连，所以该连接点可视化时并没有两个邻接守卫点。因此在搜索时不会出现上台阶的路径）



在得到了地图拓扑图后我们直接对起点和终点在拓扑图上进行Dijkstra搜索，就得到全局近似最短路径，如果想获得多个可行路径则可以进行深度优先搜索与冗余路径合并。到全局近似最短路径后我们还会进行路径剪枝，尽量将弯弯绕的路径剪成直的，路径剪枝相关代码可以参考fast-planner等论文。

后续可能的优化：这里能看出来，车道中心线是针对的静态地图，当周围有障碍物时可能就会失效，因此实际采样在采样车道中心线的同时，也会采样车中心周围6m半径的点(雷达置信可探测范围)进行采样，其实可以直接试一下动态车道中心线提取，实时得到动态的车道中心线地图。（但是不是很重要）

优化topo地图的采样效率并降低图的复杂度，如果要进行深度优先搜索拿到从A到B多条可行路径，复杂度高的拓扑地图会对搜索算法的性能产生很大的影响，目前的拓扑方式还有很多可以优化的地方

规划轨迹生成部分—轨迹优化(参考minco的轨迹建模方式)：

轨迹优化整体**输入**为上一步拓扑路径搜索得到的全局路径，首先将路径点按照离散步长步进采样得到路径点点集point_set，point_set即为轨迹优化的初值。整体轨迹优化的过程为：根据上一步得到的优化点集作为三次样条插值的控制点进行插值。对于三次样条插值的轨迹，其第i段是一个3次多项式，具体优化方式如下：

全局轨迹优化：

1. 规划插值：

$$
\begin{aligned}
S_i(x) &= a_i + b_i\left(x - x_i\right) + c_i\left(x - x_i\right)^2 + d_i\left(x - x_i\right)^3 \\
S_i'(x) &= b_i + 2c_i\left(x - x_i\right) + 3d_i\left(x - x_i\right)^2 \\
S_i''(x) &= 2c_i + 6d_i\left(x - x_i\right)
\end{aligned}
$$

根据$S_i\left(x_i\right) = a_i + b_i\left(x_i - x_i\right) + c_i\left(x_i - x_i\right)^2 + d_i\left(x_i - x_i\right)^3 = y_i$ 我们得到$a_i = y_i$,因此我们用$h_i = x_{i+1} - x_i$来表示步长，因此我们得到：$b_i + 2h_ic_i + 3h_i^2d_i = b_{i+1}$

根据导数相等，我们得到：$b_i + 2h_ic_i + 3h_i^2d_i = b_{i+1}$

根据二阶导数相等，我们得到$2c_i + 6h_id_i = 2c_{i+1}$

因此我们设置$m_i = S_i^{''}(x_i) = 2c_i$, 则我们得到$m_i + 6h_id_i = m_{i+1}$, 因此得到$d_i = \frac{m_{i+1} - m_i}{6h_i}$

带入，我们得到：$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{2}m_i - \frac{h_i}{6}\left(m_{i+1} - m_i\right)$

$$
\begin{aligned}
a_i &= y_i \\
b_i &= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{2}m_i - \frac{h_i}{6}\left(m_{i+1} - m_i\right) \\
c_i &= m_i/2 \\
d_i &= \frac{m_{i+1} - m_i}{6h_i}
\end{aligned}
$$

我们把上式带入导数等式，得到：$h_im_i + 2\left(h_i + h_{i+1}\right)m_{i+1} + h_{i+1}m_{i+2} = 6\left[\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i}\right]$

因此我们得到了可以直接求解mi的线性方程组，未知数是m

在自然边界条件下：$m_0 = 0, m_n = 0$

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & & 0 \\
h_0 & 2\left(h_0 + h_1\right) & h_1 & 0 & & 0 \\
0 & h_1 & 2\left(h_1 + h_2\right) & h_2 & & 0 \\
0 & 0 & h_2 & 2\left(h_2 + h_3\right) & & h_3 \\
\vdots & & & \ddots & \ddots & \\
 & & 0 & h_{n-2} & 2\left(h_{n-2} + h_{n-1}\right) & h_{n-1}
\end{bmatrix}
\begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ \vdots \\ \cdots \end{bmatrix}
= 6\begin{bmatrix}
0 \\
\frac{y_2 - y_1}{h_1} - \frac{y_1 - y_0}{h_0} \\
\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\
\frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\
\vdots \\
\frac{y_n - y_{n-1}}{h_{n-1}} - \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \\
0
\end{bmatrix}
$$

矩阵本身维度n+1,且系数矩阵为对角占优，且为带状矩阵

夹持边界条件(注意夹持边界条件的下标，根据导数相等推得)：

$$
\begin{aligned}
&S_0'(x_0) = A \Longrightarrow b_0 = A \\
&\Longrightarrow A = \frac{y_1 - y_0}{h_0} - \frac{h_0}{2} m_0 - \frac{h_0}{6}(m_1 - m_0) \\
&\Longrightarrow 2h_0 m_0 + h_0 m_1 = 6\left[\frac{y_1 - y_0}{h_0} - A\right] \\
&S_{n-1}'(x_n) = B \Rightarrow h_{n-1} m_{n-1} + 2(h_{n-1}) m_n = 6\left[B - \frac{y_n - y_{n-1}}{h_{n-1}}\right] \\
&\Rightarrow \quad h_{n-1} m_{n-1} + 2h_{n-1} m_n = 6\left[B - \frac{y_n - y_{n-1}}{h_{n-1}}\right]
\end{aligned}
$$

因此我们得到了两个新的方程，左侧右侧都要进行更新：

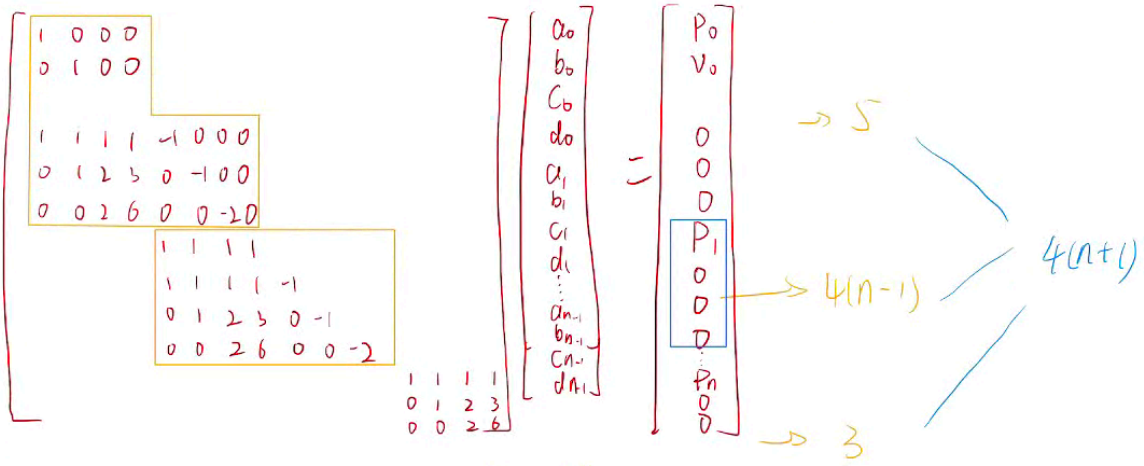
$$
\begin{bmatrix}
2h_0 & h_0 & 0 & \cdots & \cdots & 0 \\
h_0 & 2(h_0 + h_1) & h_1 & 0 & & \vdots \\
0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \vdots \\
\vdots & 0 & \ddots & \ddots & \ddots & 0 \\
0 & \cdots & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\
0 & \cdots & \cdots & 0 & h_{n-1} & 2h_{n-1}
\end{bmatrix}
$$

2. 规划插值（另一种控制方式，无需中间变量直接解算）

$$
\begin{aligned}
x &= a + bs + cs^2 + ds^3 \\
x' &= b + 2cs + 3ds^2 \\
x'' &= 2c + 6ds \\
x''' &= 6d
\end{aligned}
$$

我们展开得到线性方程式组：

$$
\begin{cases}
P_0 = a_0 \\
v_0 = b_0 \\
a_0 + b_0 + c_0 + d_0 - a_1 = 0 \\
b_0 + 2c_0 + 3d_0 - b_1 = 0 \\
2c_0 + 6d_0 - 2c_0 = 0 \\
\vdots \\
a_{n-1} + b_{n-1} + c_{n-1} + d_{n-1} = P_n \\
b_{n-1} + 2c_{n-1} + 3d_{n-1} = 0 \\
2c_{n-1} + 2d_{n-1} = 0
\end{cases}
$$



中间n-2组条件，而且是带状矩阵，可直接求解。

MINCO的建模方式实际表明了多段多项式轨迹规划的实质是**轨迹的所有相关物理量**都可以根据轨迹的**控制点集P**与**每段的时间T**控制，因此在优化的过程中只需要优化控制点P的位置与每段的时间T即可调整轨迹，达到包括但不限于平滑，避障，特定位置的加减速处理等多任务优化。

(**注意**：代码中并没有使用原生的MINCO，而是只优化了控制点位置，考虑了避障和平滑，将每段的时间T作为常量，因此对初始轨迹的最优性有一定的要求，如果想要改原生MINCO的同学可以基于现在的代码框架也可以参考MINCO的开源源码进行适配)

这里轨迹优化唯一的坑点就是**如何获取障碍物**，障碍物我是直接找的周围的所有栅格点做的L2损失计算，且只搜索路径周围1m(具体多少可调，会比较影响速度和避障性能)的占用栅格点，然后，我们设置R=0.3作为生效半径，即为这个障碍物距离我的路径小于R时才会去计算loss，这样可以有效防止狭窄路径端路径左右栅格点的数量不同导致路径被优化到栅格点少的障碍物里，同时我们也不需要对障碍物建模(因为实际上的建模只能建模为凸包，把一个不知凹凸的几何体分解为多个凸包本身就是NP hard问题，而且建模不准确直接用圆或者椭圆容易减少可通行空间，考虑到比赛以通过性为主我们放弃了这类凸包式的方案(比如飞行走廊))，同时我们也考虑了2.5m半径内的栅格点，是通过射线(raycast)得到的障碍物点，防止lbfgs优化过头了把点优化出了1m半径之后就撞到了障碍里了

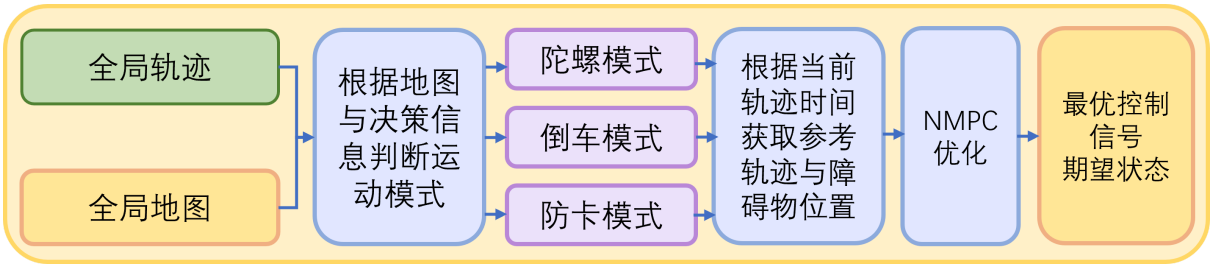
后续可能优化的点：

第一点就是minco的原始论文的方法，要是能联合时空优化速度可以更快，通过对不同段的轨迹施加不同的速度加速度约束达到控制目的(如过桥洞前减速，下坡减速等)

第二点就是其实可以考虑障碍物建模方式，飞行走廊其实是比这套代码的避障方式要好，现在的软约束计算形式主要是计算量大，导致很长的路径计算耗时大，飞行走廊实际的通过性和求解成功率并不会比现在这套方案低多少

此外优化算法可以别用lbfgs了，ceres里面的算法他不香么

局部轨迹跟踪部分—MPC：



上面得到了一条光滑无碰撞的轨迹后，这就是我们要跟踪的目标参考轨迹了，**要在保证安全平滑的同时跟踪目标参考轨迹， 我们考虑Tube-MPC的思想，将哨兵运动学模型模型整体简化为两轮阿克曼模型，**

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{v} = a \\ \dot{\theta} = w \end{cases}$$

因此我们的状态量为车身位置(x,y)，车身线速度v 以及车身朝向角θ，控制量为车身加速度a与车身朝向角速度w。对运动学模型离散化后得到模型$x(k+1) = f_d\big(x(k),u(k),F_c(k)\big)$，我们根据离散化模型以及当前的反馈状态预测时间步$k+j$的状态表示为$x(k+j|k)$,对应的控制输入表示为$u(k+j|k)$,得到一整个预测时域的控制与状态序列。因此我们可以把 MPC 损失函数表示为：↵

$$J = \sum_{j=0}^{N} \|x(k+j \mid k) - x_{ref}(k+j)\|_Q^2 + \|u(k+j \mid k)\|_{R^\epsilon}^2$$

其中为对应的参考轨迹，同时考虑运动安全性，我们设置安全约束：跟轨迹优化差不多，我们优化的参考轨迹附近1m内找障碍物，这些障碍物栅格作为MPC的硬约束进行处理，因此我们安全约束表示为：

$$\|p(x(k+j \mid k)) - o_i\|_P^2 > r^2$$

因此NMPC的总体框架为：

$$\begin{aligned} &\min J\left(x_k, u_k\right) \\ &\text{s.t.} \left|\left|x(k+j \mid k)\right|\right| \leq x_{\max} \\ &\quad \|p(x(k+j \mid k)) - o_i\|_P^2 > r^2 \end{aligned}$$

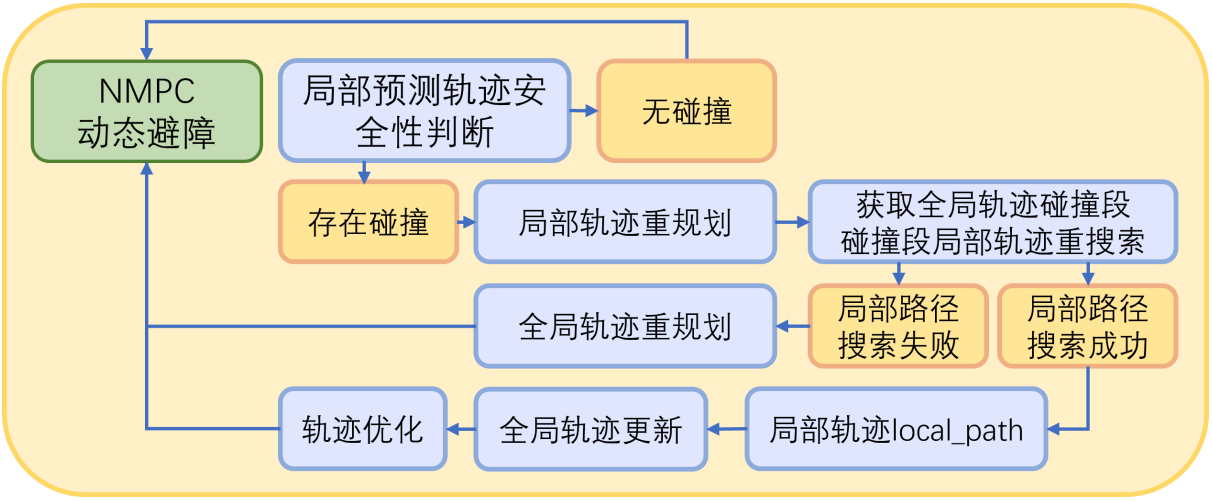
最后求解，NMPC这玩意自己学一下就行，跟无约束线性MPC一样，但是求解器使用的OCS2，这个是eth自己写的大型MPC求解器，本质就是各种非线性优化库，注意我们代码使用的是OCS2的SQP求解器，SQP本质是上一个非线性求解器，全名是序列二次优化，相当于迭代线搜索多次解QP问题，OCS2的很多算法为了加快求解速度都是不支持硬约束，只能解软约束问题，注意实际上很多非线性优化是可以直接解硬约束问题的，基本上拉格朗日乘子法都是ok的

此外代码中还有大量的模式，这些都是跟比赛进程或者底盘运动模式(如底盘跟随或者小陀螺行进有关的)，唯一需要特别说的就是**防卡(摆脱)模式**，这个模式是为了防止在行进过程中车辆因控制误差或感知误差或者被对面特意卡住而设计的，即为判断轨迹的期望速度，**如果底盘的速度在很长的一段时间内都很低**，则判断底盘目前处于被卡住的状态，则将MPC计算出底盘的**期望转速/转角**与**行进速度**直接取负，相当于模拟被卡住后倒车转弯的行为，并触发重规划。

后面可以优化的：动态障碍物躲避，今年没有做的根本原因是感知没办法做到很准，导致我没办法信任感知的具体数据，或者就要狠狠滤波才能进行速度估计，但是这样在障碍物一多的情况下优化出来的轨迹会很奇怪，且这种情况下MPC可能会直接撞静态的物体。

飞行走廊：这玩意挺重要的，去年没用的根本原因是想躲动态障碍物，我飞行走廊怎么去处理这个玩意啊，但是最后动态避障也没做出来，导致实际上的距离软约束效果虽然还行，但是我为了减少约束个数，当原始参考轨迹在障碍物里的时候才去考虑障碍物的软约束，这样处理减少了计算量，且在狭窄路段不会出现MPC的异常求解信号，但是坏处就是如果我实际车辆跟踪情况不是很好的情况下不保证无碰撞安全。如果动态避障还是做不好的话，可以直接用飞行走廊，这样可以用OSQP求解

避障与重规划—三段式重规划逻辑



为了适配比赛场景的要求，我们要优先保证运动的连续性与可通行能力，这也是为啥我们没有选择更为保守的飞行走廊。因此我们将其表示为惩罚或损失的软约束的形式，更期望其能优先保证运动连续性。我们认为频繁的重规划尤其是对于轨迹规划此类多解系统会影响运动的连续性，尤其是起点终点有多种拓扑路段可达的情况下，因此频繁的重规划我们无法保证多次重规划路径**拓扑**的**一致性**，因此我们设计了三阶段的避障重规划系统来进行处理。

第一层局部动态避障，在遇到动态障碍物时，NMPC的障碍物约束具有一定的避障能力，可以解决很多临时的**局部动态避障**问题，因此我们将其作为重要的动态避障方式。若MPC优化得到的预测状态序列连续多次出现与障碍物碰撞的情况，则认为局部避障算法遇到了较为极端的情况，需要进行重规划，则发布重规划标志位，进入第二层对轨迹进行局部重规划。轨迹**局部重规划**算法将对原路径进行碰撞段判断，找到与障碍物碰撞的路径段后对该段进行局部拓扑路径重搜索，如果能重搜索成功则将局部路径融入全局路径，路径剪枝后重新进行轨迹优化。若无法搜索到局部路径，再进入第三层进行**全局重规划**。

感知与地图处理—如何处理局部障碍，桥洞和台阶飞坡

只使用普通的二维栅格地图方案局限性较大，要同时考虑多种地形必须考虑所有维度的空间信息。但哨兵机器人属于地面移动机器人，其可控移动基本上是沿着某一平面进行运动，直接使用三维地图方案信息冗余且计算量较大。因此我们的哨兵地图感知方案为**2.5维方案**，通过加入高度地图，作为地图每个点的先验高度信息，即可实时感知不同地形下不同动态障碍物的位置。实际的栅格地图是根据得到的**高度地图**进行**边缘检测**并膨胀后得到的，实际的处理为: 我们得到建图后的点云，在对其进行矫正处理后按照z轴高度映射得到了整体地图的高度图，我们使用激光雷达获得的实时点云数据及深度相机得到的置信度局部点云图，对于点云的每个点，其z轴位置与先验高度图对应位置的高度z1之差处于某一区间内则认为其为动态障碍物，即$\delta_{\max} > z_1 - z > \delta_{\min}$, 时认为在该位置处存在障碍物(可参考2023年哈工大在青工会讲的内容)。过高或过低的点云高度差对于哨兵本体来说都认为是可通行的区域，得到了高度地图也就有了占用地图，有了占用地图进行骨架提取后也就有了车道中心线图，再通过topo搜索我们就可以正确处理台阶，飞坡，坡道等地形。**桥洞处理**也是很工程化的处理：因为我们的占用图中桥洞那一片区域不是占用的，是可以通行的，但是这样就无法通过高度检查，即为虽然桥洞**入口出口的栅格是非占用的(下图占用图的黑色部分)，他们的邻域的点也都是非占用的，但是他们的邻域存在很大的高度差，因此我们仅仅通过占用与高度变换的不匹配就可以找到所有的桥洞出入口，这样我们就能知道桥洞的区域，所以我们就知道桥洞在哪了，我们给每个栅格都加两个属性：即为该栅格是否存在第二高度以及第二高度的高度是多少，这样我们就可以正确处理桥洞了，同理我们在采样的时候采样到了桥洞区域就同时把桥洞上和桥洞下的点都进行一次topo链接判断。**

后面可以优化的：三维感知处理！其实现在这套主要是处理桥洞比较麻烦，但是我倾向于如果官方不大改地图加很多桥洞的话就没必要改，而且三维感知处理最麻烦的就是地图，现在建图如果存在其他在移动的东西，比如人，机器人，一多那个地图直接就烂的不成样子，直接把这个点云用于三维感知的话会很寄，要修图，或者建图的时候把人和机器人想办法去掉，有点麻烦，或者就结合二维和三维，把处理完的二维作为三维的先验信息来处理杂点。

然后是桥洞的处理，期望更多的nice的算法可以自适应的识别桥洞现在的这个方法只能自适应识别桥洞的出入口，且对地图的精度有要求，至于怎么把出口的矩阵连起来，这个还没有什么好办法。