



CILQR 规划算法 & 轨迹树规划

1. CiLQR 规划算法

相比上述传统的来自无人机的规划算法，CILQR(Constrained Iterative LQR)则是较新的基于车辆运动学或动力学的优化方法，相比于以上的基函数规划方法其有几个明显的优点：

- 把轨迹规划问题处理成一个轨迹最优控制问题，直接考虑的运动学约束和控制量约束，理论上可以直接兼容覆盖泊车场景
- CILQR可以直接在自车坐标系下进行规划，因而不需要去对感知的提供的车道线去做平滑了，而是可以直接使用视觉提供的车道线的 $C0, C1, C2, C3$ 参数来作为控制参考输入，和边界条件
- 整个求解算法，完全使用基础的矩阵运算，不需要使用任何求解器依赖库（只需要EIGEN 库），算法对于测试和开发过程的debug 完全是白盒，利于分析和工程化的优化改进

同时，相比于传统的IPM等通用非线性优化方法，CILQR在车辆轨迹规划的特殊应用场景下的求解效率也更为高效,因为：

- 利用了动态轨迹的方法将全局优化问题转换为一个个独立的局部优化问题，简化了问题
- 相比于DDP，CILQR 对车辆运动学约束省去了对hessian 矩阵的计算，而只保留了一阶展开（只计算雅可比矩阵），所以计算速度更快。（对应车辆运动约束这个特殊的形式，计算hessian 矩阵消耗的时间要远高于从二阶收敛降为一阶收敛多做的迭代次数）
-

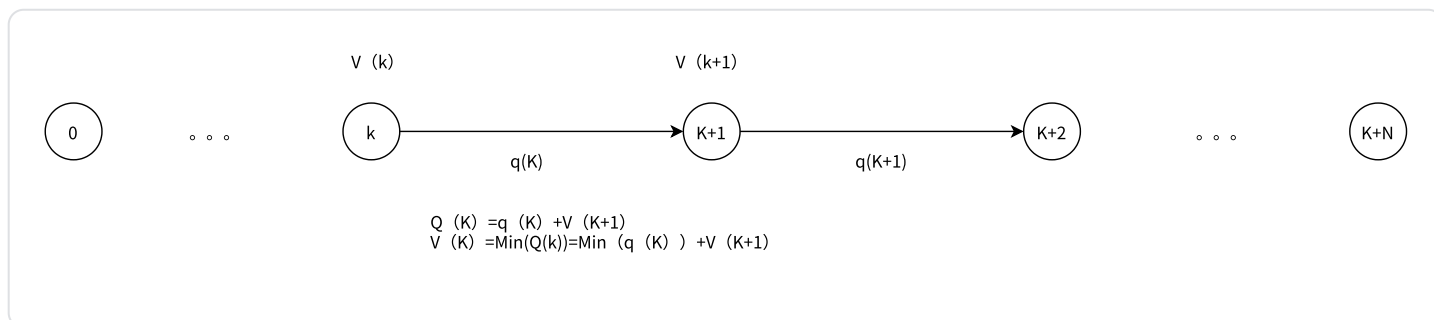
CILQR 方法在传统线性LQR算法的基础上：

- 通过引入线性化迭代求解，来解决目标函数和系统状态方程的非线性问题
- 通过引入barrier function，来解决不等式约束的问题

下面我们就从基础LQR开始详细介绍推导CILQR算法。

1.1 LQR 的动态规划求解思路

LQR的求解基本思路是将整个轨迹规划问题，理解为一个离散的马尔科夫链的序列优化求解问题。



车辆的轨迹可以理解为在离散的时间序列上，给定一个步长的控制量，车辆状态从当前状态转移到下一个状态的过程。而整个轨迹规划过程，即从当前状态，寻找一个最优控制序列，将车辆从当前状态，经过指定的时间步数，转移到目标状态的问题。

使用动态规划思想，我们可以将认为：

- 某个节点到终端节点的路径的最小累计cost 即为该节点的状态价值 $V(k)$
- 从 k 节点给定一个控制量，其从当前 k 状态根据系统运动方程转移到 $k+1$ 状态产生的cost 为 $q(k)$
- 假如从 k 步给定一个任意的控制量，车辆转移到 $k+1$ 状态，而从 $k+1$ 状态到终端状态，我们确保能够一直选择最优控制量，那么我们就可以认为当前 k 步在给定控制量下的动作价值 $Q(k) = q(k) + V(k+1)$
- 这样寻找一个最优的控制序列 u_1, u_2, \dots, u_N 的问题就可以结构成一个嵌套的从终端状态寻找最优控制量的子问题。 V, q, Q 均为标量代表cost
- 如果优化求解问题的目标函数可以被写成 X, U 的二次型函数，且运动学等式约束可以写成 X, U 的线性组合，那这就一个标准的LQR 优化问题。

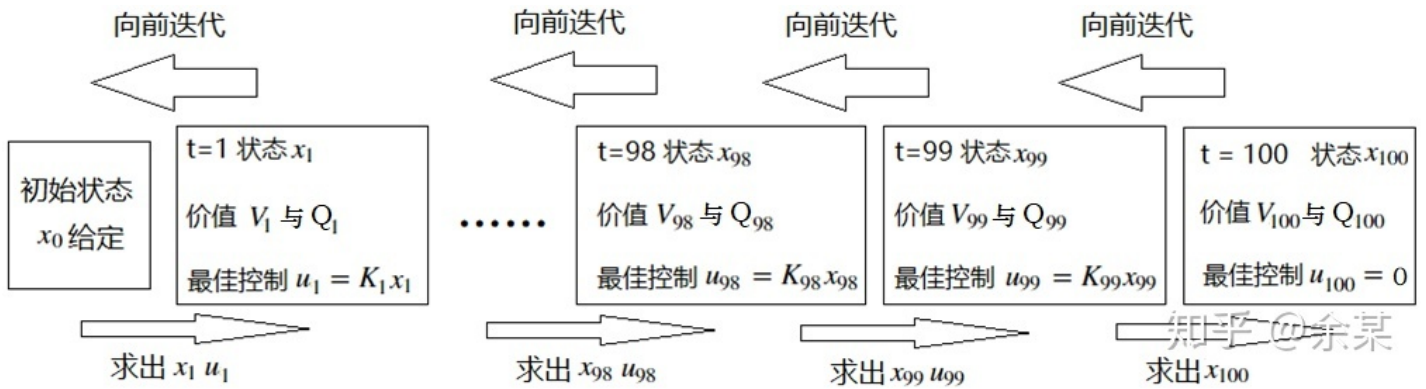
$$\min_{u_1, u_2, \dots, u_n} = \sum_{i=1}^n C(k) \quad (1)$$

$$X_{k+1} = F \begin{bmatrix} X_k \\ U_k \end{bmatrix} + f \quad (2)$$

$$C(k) = \frac{1}{2} \begin{bmatrix} X_k \\ U_k \end{bmatrix}^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} \begin{bmatrix} X_k \\ U_k \end{bmatrix} + \begin{bmatrix} X_k \\ U_k \end{bmatrix}^T \begin{bmatrix} D_x \\ D_u \end{bmatrix} \quad (3)$$

$$\text{其中 } C_{ux} = C_{xu}^T$$

这样从终端状态X (N) 我们可以求出上一步的最优控制反馈增益K (k-1) , 以及对应的状态价值矩阵C (k-1) , 这样一步步往前推, 直到初始状态X (0) . 然后根据K (0) 正向求出每一步的最优控制量和最优状态。



- 按照上述的backward ,forward 优化求解方式, 我们可以以如下 例子, 进行如下推导归纳:
 - 在终端状态 (举例为第100步) 下, 因为是我们的目标状态, 因此没有cost, 因此, 可以认为 $V(100) = 0$

则在第99步时, 其action value, 是:

$$Q(99) = \frac{1}{2} \begin{bmatrix} X_{99} \\ U_{99} \end{bmatrix}^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} \begin{bmatrix} X_{99} \\ U_{99} \end{bmatrix} + \begin{bmatrix} X_{99} \\ U_{99} \end{bmatrix}^T \begin{bmatrix} D_x \\ D_u \end{bmatrix} + V(100) \quad (4)$$

要得到第99步的最优控制, 是的Q(99)最小, 则可以对 U_{99} 求导:

$$\frac{\partial Q(99)}{\partial U_{99}} = 0 \text{ 可以得到最优控制}$$

$$U_{99}^* = K_{99} X_{99} + k_{99} \quad (5)$$

$$K_{99} = -C_{uu}^{-1} C_{xu} \quad (6)$$

$$k_{99} = -C_{uu}^{-1} D_u \quad (7)$$

将最优控制量5式 代入 4式, 即可得到99步的state value.

$$V(99) = \frac{1}{2} X_{99}^T V_{99} X_{99} + X_{99}^T v_{99} + const \quad (8)$$

其中

$$V_{99} = C_{xx} + K_{99} C_{ux} + C_{xu} K_{99} + K_{99}^T C_{uu} K_{99} \quad (11)$$

$$v_{99} = C_{xu} k_{99} + K_{99}^T C_{uu} K_{99} + D_x + K_{99}^T D_u \quad (12)$$

再往前推, 计算第98步, 其action value:

$$Q(98) = \frac{1}{2} \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix}^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix}^T \begin{bmatrix} D_x \\ D_u \end{bmatrix} + V(99)$$

将8式代入

$$Q(98) = \frac{1}{2} \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix}^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix}^T \begin{bmatrix} D_x \\ D_u \end{bmatrix} + \frac{1}{2} X_{99}^T V_{99} X_{99} + X_{99}^T v_{99} + \text{const} \quad (13)$$

由于 $X_{99} = F * \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + f$ 代入，

$$Q(98) = \frac{1}{2} \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix}^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix}^T \begin{bmatrix} D_x \\ D_u \end{bmatrix} + \frac{1}{2} (F * \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + f)^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} (F * \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + f) + (F * \begin{bmatrix} X_{98} \\ U_{98} \end{bmatrix} + f)^T \begin{bmatrix} D_x \\ D_u \end{bmatrix} \quad (14)$$

可以写成：

$$Q(98) = \frac{1}{2} X_{98}^T \hat{Q} X_{98} + X_{98}^T \hat{q} + \text{const} \quad (15)$$

其中：

$$\hat{Q} = \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} + F^T V_{99} F = \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \quad (16)$$

$$\hat{q} = \begin{bmatrix} D_x \\ D_u \end{bmatrix} + F^T V_{99} F + F^T v_{99} = \begin{bmatrix} q_x \\ q_u \end{bmatrix} \quad (17)$$

再次对Q(98) 对 U_{98} 求偏导，获得该步的最佳控制率：

$$U_{98}^* = K_{98} X_{98} + k_{98} \quad (18)$$

$$K_{98} = -Q_{uu}^{-1} Q_{ux} \quad (19)$$

$$k_{99} = -Q_{uu}^{-1} q_u$$

将18式代回 14式 即可以得到第98步的价值：

$$V(98) = \frac{1}{2} X_{98}^T V_{98} X_{98} + X_{98}^T v_{98} + \text{const}$$


$$V_{98} = Q_{xx} + K_{98}^T Q_{ux} + Q_{xu} K_{98} + K_{98}^T Q_{uu} K_{98} \quad (11)$$

$$v_{98} = Q_{xu} k_{98} + K_{98}^T Q_{uu} K_{98} + q_x + K_{98}^T q_u \quad (12)$$

○ 以此方式，不断往前类推，直至达到初始状态X(0).

从以上手动的推导过程可以看到第99步的最优控制率K99，k99 实际上是一个特例，因为V（100）=0， 所以相应项被省略了，如果我们按照完整的形式来写，应该为：

Backward recursion



for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$


$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$

知乎 @王沃河

◦ Forward 求解状态:

到这里，我们求出了每一步的最优控制律，从初始状态开始就可以逐步推导出每一步的最优状态。

Forward recursion



for $t = 1$ to T :

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

1.2 ILQR的迭代求解思路

回顾LQR的求解推导过程，我们之所以可以这么推导，是因为：

- Cost 函数 $C(k) = \frac{1}{2} \begin{bmatrix} X_k \\ U_k \end{bmatrix}^T \begin{bmatrix} C_{xx} & C_{xu} \\ C_{ux} & C_{uu} \end{bmatrix} \begin{bmatrix} X_k \\ U_k \end{bmatrix} + \begin{bmatrix} X_k \\ U_k \end{bmatrix}^T \begin{bmatrix} D_x \\ D_u \end{bmatrix}$ 是标准线性二次型
- 系统状态转移方程是X, U的线性组合 $X_{k+1} = F \begin{bmatrix} X_k \\ U_k \end{bmatrix} + f$

但是在轨迹规划问题中，cost 函数有可能包括非线性项，同时系统状态方程通常是非线性的。

$$X_{k+1} = f(X_k, U_k)$$

$$C_{k+1} = l(X_k, U_k)$$

f, l均为非线性函数。

此时我们就不能直接使用上面的方式进行直接推导了，而需要先把两个非线性函数线性化。

对于cost函数L我们进行二阶泰勒展开。

- 如果我们把系统方程f 只做一阶泰勒展开，那这种方式就是ILQR，如果把f 函数也做二阶展开，那这种方式就是DDP。所以ILQR是DDP的特例。
- 使用DDP的方式，优化迭代是按二次速度收敛的，所以其需要的迭代次数更少，但是单次迭代的计算量更大。ILQR则反之。对于轨迹规划和MPC控制这种问题而言，ILQR效果相对而言更高。

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$
$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

移项，

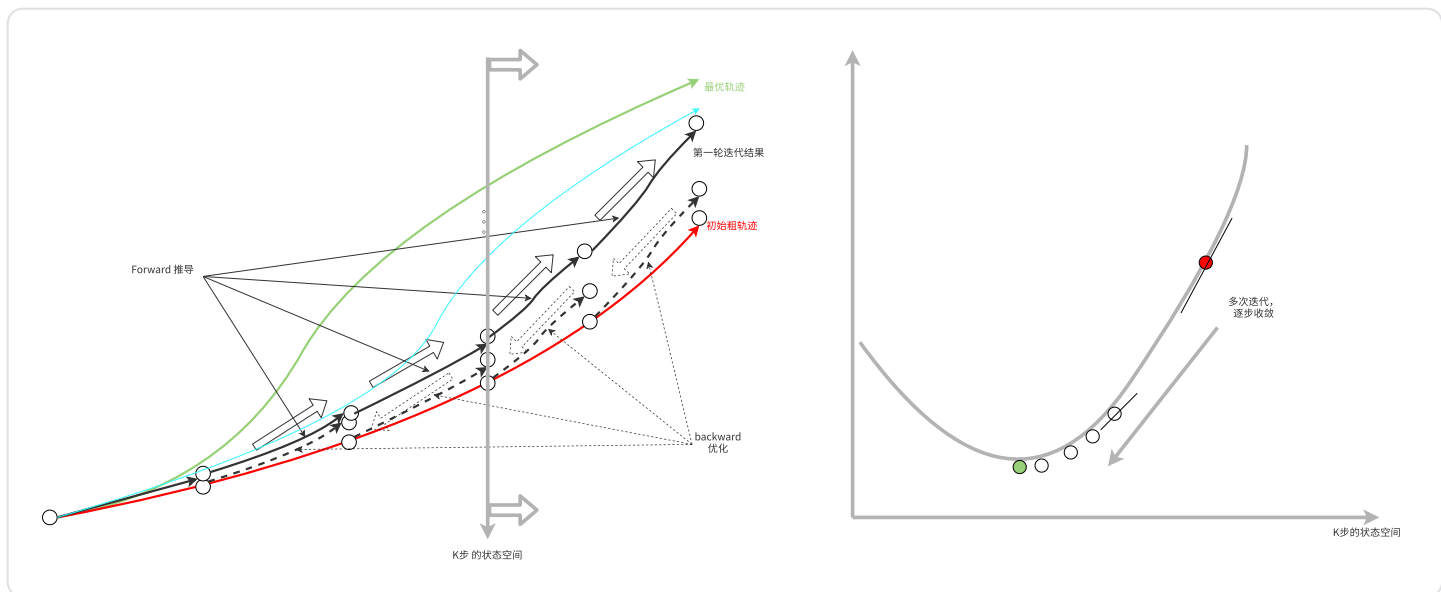
$$f(\mathbf{x}_t, \mathbf{u}_t) - f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \approx \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$
$$c(\mathbf{x}_t, \mathbf{u}_t) - c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \approx \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

重新定义一下符号，我们发现，又神奇的变回了LQR中几乎一模一样的形式(5)(6)，意味着我们可以使用LQR了。

$$\bar{f}(\delta \mathbf{x}_t, \delta \mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}$$

单步的目标函数上图写为c (x,u) , 在下面的推导中，我们写成 $l(x, u)$

通过对两个函数的线性化处理，我们发现在两个节点之间的cost 的**增量**又变成标准二次型，而系统状态的**变化量**又变成了输入增量的线性组合。所以我们有可以使用LQR的求解方式了。这样我们就可以把原问题转化为，针对原有轨迹上每一步的控制计算 最优修正量 δU ,使得每一步的对应cost下降最快。（换个角度考虑，如果我们能保证，每一步的控制量经过修正，每一步cost 都能以最快的速度降低，经过多步的反复计算，我们一定能够收敛到一个最优的轨迹上）。同时以为 δU 足够小，我们就可以在该状态上进行线性化。因此，同样使用action value 和state value 的概念，我们需要优化的是，每一步上的action value 的增量。



ILQR的求解过程由三步组成，在每次优化中，这三步将被反复执行，直至总的cost 收敛，这也是ILQR中iterative 的来源。

初始轨迹和梯度向量/海塞矩阵计算

根据初始轨迹提供的初始状态量，求解对应点上的目标函数的一阶梯度向量，和海塞矩阵 和系统状态方程的雅可比矩阵。

由初始轨迹Traj_init $(x_0, x_1, x_2, \dots, x_N)$

我们可以计算得到每个节点上的的雅可比矩阵 和海塞矩阵。

$$l_k = \begin{bmatrix} l_x \\ l_u \end{bmatrix}_{x_k u_k} = \begin{bmatrix} \frac{\partial l}{\partial x} \\ \frac{\partial l}{\partial u} \end{bmatrix}_{x_k u_k}$$

$$L_k = \begin{bmatrix} l_{xx} & l_{xu} \\ l_{ux} & l_{uu} \end{bmatrix}_{x_k u_k} = \begin{bmatrix} \frac{\partial^2 l}{\partial x \partial x} & \frac{\partial^2 l}{\partial x \partial u} \\ \frac{\partial^2 l}{\partial u \partial x} & \frac{\partial^2 l}{\partial u \partial u} \end{bmatrix}_{x_k u_k}$$

$$F_k = \begin{bmatrix} f_x \\ f_u \end{bmatrix}_{x_k u_k} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial u} \end{bmatrix}_{x_k u_k}$$

Backward 推导：

$$\delta Q(k) = \frac{1}{2} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} l_{xx} & l_{xu} \\ l_{ux} & l_{uu} \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} + \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} l_x \\ l_u \end{bmatrix} + \delta V(k+1) \quad (13)$$

$$\text{而 } \delta V(k+1) = \frac{1}{2} \delta X_{k+1}^T V_{k+1} \delta X_{k+1} + \delta X_{k+1}^T v_{k+1} \quad (14)$$

$$\text{由于 } \delta X(k+1) = \begin{bmatrix} f_x & f_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} \quad (15)$$

将15式代入 14，就可以将13式写为

$$\delta Q(k) = \frac{1}{2} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} l_{xx} & l_{xu} \\ l_{ux} & l_{uu} \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} + \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} l_x \\ l_u \end{bmatrix} + \frac{1}{2} \left(\begin{bmatrix} f_x & f_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} \right)^T V_{k+1} \left(\begin{bmatrix} f_x & f_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} \right) + \left(\begin{bmatrix} f_x & f_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} \right)^T v_{k+1} \quad (16)$$

对其展开后重新整理，可以得到：

$$\delta Q(k) = \frac{1}{2} \begin{bmatrix} 1 \\ \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta X_k \\ \delta U_k \end{bmatrix} \quad (17)$$

其中：

$$\begin{aligned} Q_x &= l_x + f_x^T v_{k+1} \\ Q_u &= l_u + f_u^T v_{k+1} \\ Q_{xx} &= l_{xx} + f_x^T V_{k+1} f_x \\ Q_{xu} &= l_{xu} + f_x^T V_{k+1} f_u \\ Q_{ux} &= l_{ux} + f_u^T V_{k+1} f_x \\ Q_{uu} &= l_{uu} + f_u^T V_{k+1} f_u \end{aligned} \quad (18)$$

对 δU_k 求导，得到最优控制率

$$\begin{aligned} \frac{\delta Q(k)}{\delta U_k} &= 0 \\ \text{可以得到：} \\ \delta U_k^* &= K_k \delta X_k + k_k \\ K_k &= -Q_{uu}^{-1} Q_{ux} \\ k_k &= -Q_{uu}^{-1} Q_u \end{aligned} \quad (19)$$

将其代入16式，可得

$$\delta V(k) = -0.5 Q_u Q_{uu}^{-1} Q_u \quad (20)$$

$$V_k = Q_x - Q_u Q_{uu}^{-1} Q_{ux} \quad (21)$$

$$v_k = Q_{xx} - Q_{xu} Q_{uu}^{-1} Q_{ux} \quad (22)$$

海塞矩阵Regulation

由于计算中，我们需要对 Q_{uu} 求逆矩阵，但是我们又无法保证 Q_{uu} 是正定的，因此我们需要在发现 Q_{uu} 非正定时，使用LM系数法对 K_k ， k_k 进行修正：

$$\tilde{Q}_{uu} = l_{uu} + f_u^T (V_{k+1} + \mu I) f_u \quad (23)$$

$$\tilde{Q}_{ux} = l_{ux} + f_u^T (V_{k+1} + \mu I) f_x \quad (24)$$

$$\begin{aligned} K_k &= -\tilde{Q}_{uu}^{-1} \tilde{Q}_{ux} \\ k_k &= -\tilde{Q}_{uu}^{-1} Q_u \end{aligned} \quad (25)$$

将修正后的 K_k ， k_k 代入 16式，可以计算得到

$$\delta V(k) = 0.5 k_k^T Q_{uu}^{-1} k_k + k^T Q_u \quad (26)$$

$$V_k = Q_x + K_k^T Q_{uu} k_k + K_k^T Q_u + Q_{ux} k_k \quad (27)$$

$$v_k = Q_{xx} + K_k^T Q_{uu} K_k + K_k^T Q_{ux} + Q_{ux}^T K_k \quad (28)$$

而 μ 是 >0 的系数，当 μ 接近于0时，迭代优化收敛的更快，但是系统容易不稳定；当 μ 值很大时，则系统可以变得更稳定，不会发散，但是系统收敛的更慢，需要更多步的迭代，计算时间增长。

因此，我们需要在每一步的backward pass 时动态的去调整 μ 参数：

- 在接近最优点附近时，逐渐调小 μ
- 而当原有 Q_{uu} 非正定时，则需要快速增加 μ ，并重新去尝试backward pass

我们按照如下方法，动态的去调整系数 μ ：



- 上一个时间步的 μ 被保存下来，作为这一时间步的初始值
- 上一个迭代步的 μ 被保存下来，作为这一轮迭代步的初始值
- 在backward 反向推导过程中修正，我们对每一个节点上的 Q_{uu} 进行Cholesky 分解来检查其是否是正定的，如果不是，则通过如下方式增加 μ 后，重新从终端状态尝试 backward 推导：
 - $d\mu = \max(d\mu * \text{MuFactor}, \text{MuFactorr});$
 - $\mu = \max(\mu * d\mu, \text{MuFactor});$
- 在本轮forward pass 之后，检查linear search条件是否满足，如果满足，则通过如下方式减小 μ
 - $d\mu = \min(d\mu / \text{MuFactor}, 1/ \text{MuFactor});$
 - $\mu = \mu * d\mu * (\mu > \text{MuMin});$
- 在本时间步完成迭代优化后，本时间点的最终dmu, mu被保存下来作为下一时间步的初始值

Forward 推导：

通过backward 步得到的最优控制率K, k，我们就可以从初始状态，再一步步反推，得到新的修正过的轨迹。然后以新的轨迹，回到第一步，以新的轨迹点上的状态，重新计算对应点上的雅可比矩阵和海塞矩阵。

$$x_0 = x_{start} \quad (22)$$

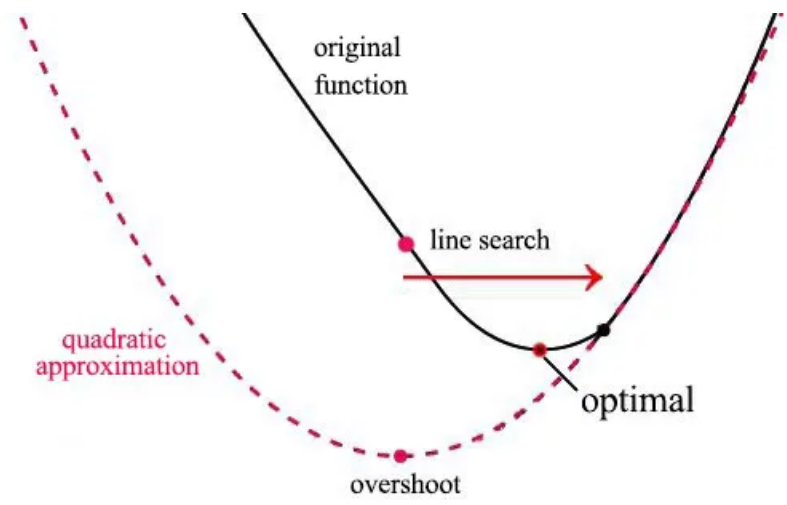
每一个节点上上新的控制量，可以在旧轨迹控制量的基础上，做相应修正，然后算出新的状态量

$$U_k = \hat{U}_k + K_k \delta X_k + \alpha k_k \quad (23)$$

$$X_{k+1} = f(X_k, U_k)$$

其中 α （线性搜索因子）是一个修正系数，用于解决在接近最优点附近时，可能出现的overshot 现象。 α 默认为1，在每次迭代后，我们需要对总的cost进行计算，并对cost 的下降梯度进行检查，如

果梯度比预期小，则需要通过降低 α 后，重新进行Forward 推导，直到下降梯度条件得到满足。



$$\Delta J(\alpha) = \alpha \sum_{i=1}^{N-1} \mathbf{k}(i)^T Q_{\mathbf{u}}(i) + \frac{\alpha^2}{2} \sum_{i=1}^{N-1} \mathbf{k}(i)^T Q_{\mathbf{uu}}(i) \mathbf{k}(i).$$

When comparing the actual and expected reductions

$$z = [J(\mathbf{u}_{1..N-1}) - J(\hat{\mathbf{u}}_{1..N-1})] / \Delta J(\alpha),$$

we accept the iteration only if

$$0 < c_1 < z.$$

1.3 CILQR的求解思路和barrier function 方式

解决非线性函数的优化问题。还有最后一个问题，ILQR无法直接考虑不等式约束，而这在轨迹规划中是必须考虑的。

在CILQR中，我们将不等式约束，通过一个barrier function 转换成等效cost 来处理。

$$\begin{aligned} d(x_k) &\leq 0 \\ d(u_k) &\leq 0 \end{aligned}$$

以上为在每个时间步上对状态和控制的约束条件，而且约束函数d也可以是非线性函数。

首先我们构造一个barrier function, 将约束函数处理成一个可导方程形式。Barrier function 有不同的形式可用，比如（指数函数，log 函数，平方函数），以下以指数函数为例：

$$b(x_k) = q_1 e^{q_2 d(x_k)} \tag{24}$$

当d(xk)>=0 时， barrier function就会趋于无穷大，从而实现等效约束。

但是由于 $d(x_k)$ 和 $b(x_k)$ 都是非线性函数，我们还不能直接使用，需要对其在 x_k 状态下进行线性化：

将式24 在 x_k 处进行泰勒二阶展开

$$\delta b(x_k) = b(x_k + \delta x) - b(x_k) = \delta x^T \frac{\partial b}{\partial x} + \delta x^T \frac{\partial^2 b}{\partial x^2} \delta x = \delta x^T g + \delta x^T G \delta x \quad (25)$$

其中：

$$g = \frac{\partial b}{\partial x} = q_1 q_2 e^{q_2 d(x_k)} \frac{\partial d}{\partial x} \Big|_{x_k} \quad (26)$$

$$G = \frac{\partial^2 b}{\partial x^2} = q_1 q_2^2 e^{q_2 d(x_k)} \left(\frac{\partial d}{\partial x} \Big|_{x_k} \right) \left(\frac{\partial d}{\partial x} \Big|_{x_k} \right)^T + q_1 q_2 e^{q_2 d(x_k)} \frac{\partial^2 d}{\partial x^2} \Big|_{x_k} \approx q_1 q_2^2 e^{q_2 d(x_k)} \left(\frac{\partial d}{\partial x} \Big|_{x_k} \right) \left(\frac{\partial d}{\partial x} \Big|_{x_k} \right)^T \quad (27)$$

这里我们忽略二次高阶项，只保留一次项，这样barrier function 的增量也可以表示成标准的二次型。这样我们就可以直接将约束条件的barrier function 加入到 目标函数中，当成目标函数的一部分，这样我们就可以把有约束的CILQR问题，处理成无约束的ILQR问题来求解。

这里我们重新写下CILQR下的优化求解流程：

Repeat 如下步骤，直至V收敛：

Step1:

- 根据初始的轨迹，计算每个节点上的雅可比矩阵，海塞矩阵 l_k, L_k, F_k
- 根据初始轨迹，计算在每个节点上的状态、控制量的barrier 一阶，二阶矩阵 g, G
- 将barrier cost 项加入到原始的cost 矩阵中, 对目标函数的梯度矩阵和海塞矩阵更新

$$l_k = l_k + g$$

$$L_k = L_k + G$$

Step2:

使用ILQR的backward 流程，按顺序计算对应时间步上的矩阵：

- Action value 的摄动矩阵 $Q_x, Q_u, Q_{xx}, Q_{xu}, Q_{ux}, Q_{uu}$
- 通过以上摄动矩阵计算，最优控制率矩阵 K_k, k_k
- 根据最优控制率，计算当前步的state value 矩阵 $\delta V(k), V_k, v_k$

从终端状态向前反推，直至x0

Step3: 根据step2 计算得到的最优控制率矩阵 K_k, k_k

- 初始化线性搜索因子 $\alpha = 1$
- Repeat 直到梯度下降比例条件满足
 - 从初始状态x0，从前往后推算更新后的轨迹状态 x_k 和控制量 u_k 直至终端步
 - $U_k = \hat{U}_k + K_k \delta X_k + \alpha k_k$
 - $X_{k+1} = f(X_k, U_k)$
 - 计算期望的cost 下降梯度与实际的下降梯度的比例z ,如果z 小于设定阈值，则减小 α 后重新进行forward 推导。

end

如果，迭代次数超过设定阈值，则认为规划失败，退出。

End steps.

1.4 CILQR 在轨迹规划上的问题描述

$$x^*, u^* = \arg \min_{x, u} \phi(x_N) + \sum_{k=0}^{N-1} L^k(x_k, u_k) \quad (30a)$$

$$s.t. \quad x_{k+1} = f(x_k, u_k), \quad k = 0, 1, \dots, N-1 \quad (30b)$$

$$x_0 = x_{start} \quad (30c)$$

$$d(x_k, O_j^k) > 0, \quad k = 1, 2, \dots, N. \quad j = 1, 2, \dots, m \quad (30d)$$

$$\underline{u} < u_k < \bar{u}, \quad k = 1, 2, \dots, N-1 \quad (30e)$$

其中：

- x_k, u_k 分别为第 k 步上的车辆状态和控制量
 - 其中车辆状态量 $x = (x, y, v, a, \theta)$ 分别为自车笛卡尔坐标下的 x, y 坐标, v, a 为车轴方向的行驶速度/加速度, θ 为航向角
 - 控制量为 $u = (J, yr)$ 分别为每一步上的车辆纵向jerk和yawrate
- x_0 为初始状态
- $x_{k+1} = f(x_k, u_k)$ 为车辆运动学方程为如下的非线性方程

$$v(k+1) = v(k) + a(k) * t_s$$

$$a(k+1) = a(k) + J(k) * t_s$$

$$\theta(k+1) = \theta(k) + yr * t_s$$

$$x(k+1) = x(k) + \cos(\theta(k)) * (v(k) * t_s + 1/2 * a(k) * t_s^2 + 1/6 * J(k) * t_s^3)$$

$$y(k+1) = y(k) + \sin(\theta(k)) * (v(k) * t_s + 1/2 * a(k) * t_s^2 + 1/6 * J(k) * t_s^3)$$

- $L^k(x_k, u_k)$ 为第 k 步上的 $cost$ ，由如下项加权组成：
 - 对控制的惩罚（纵向jerk 的惩罚 和 对yawrate的惩罚）代表了舒适性

$$c_{comfort} = U^T W_c U$$

W_c 为jerk 和yawrate 的权重

- 对横向加速度的惩罚

$$c_{centripetal} = W_l (v * yr)^2$$

- 对偏离粗轨迹的惩罚

参考线是来自于决策的粗轨迹点的横向信息，经过插值后得到。仅作为参考，确保规划出来的轨迹与决策确定的大致轨迹一致。

$$C_{ofs} = (y(k) - y_r(k))^T Q (y(k) - y_r(k))$$

Q 为横向偏离的权重。

- 对偏离参考速度的惩罚

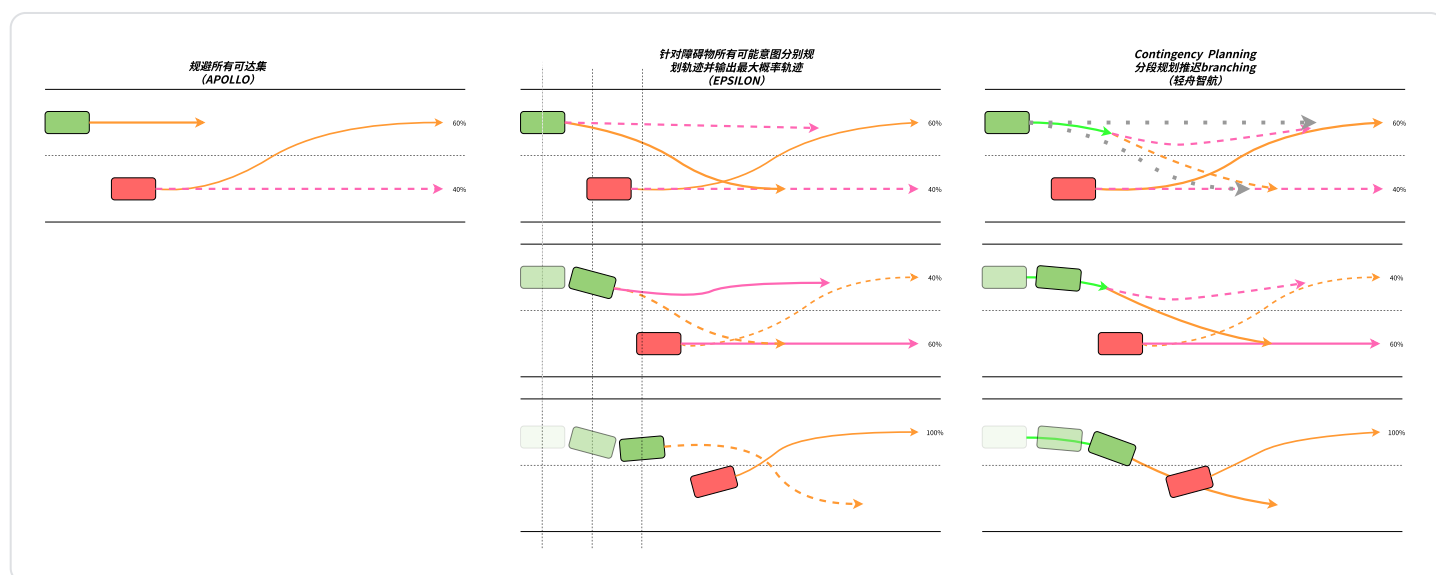
参考速度是来自于决策的粗轨迹点的纵向信息，经过插值后得到。仅作为参考，确保规划出来的轨迹与决策确定的大致轨迹一致。

- $d(x_k, O_j^k) > 0$ 为自车在任意步上与障碍物的相对距离 > 0 的约束
 - 如果障碍物使用椭圆表示，自车使用两个等效圆表示，可以通过计算两个圆心到障碍物椭圆中心的距离来表示碰撞约束。
 - 对于路沿等道路线性边界约束，则可以直接使用路沿的五次多项式来表述x的约束。
- $x_{l0} < x_k < x_{up}$ 为高阶状态量的box约束，包括
 - 车辆速度在给定范围内 (道路限速)
 - 车辆加速度在给定范围内
- $u_{l0} < u_k < u_{up}$ 为控制量的box 约束
- 由于规划是在自车坐标系下完成的，在每次规划完后，需要将轨迹重新转换到UTM下，并完成 stitch 并传给control 模块。

2. 应变式规划Contingency Planning & 轨迹树规划 (他山之石)

针对未来的不确定性，其实有两种处理方式（其实也代表了目前业内两种规划路线）

- 强决策- 综合评估未来的不确定性，根据未来的发生的不同可能概率，综合确定一个最佳的确定性的相应微观策略（比如，从左边绕or从右边绕，加速or 减速），然后将这个确定性的策略传给规划模块，有规划模块为此规划出一条确定性的轨迹。（代表：**华为、蔚来、小鹏**）
- 强规划- 把针对不确定性处理的微观决策留到规划中直接考虑，针对未来可能发生的不同场景，各自进行规划，然后根据cost 和权重，逐渐收敛到眼前的规划上。整个规划结果以树的形式展现，眼前的规划为树根，针对未来可能的场景规划的轨迹为树枝分叉。（代表：**轻舟智航，大疆车载**）



2.1 Contingency Planning 应变式规划

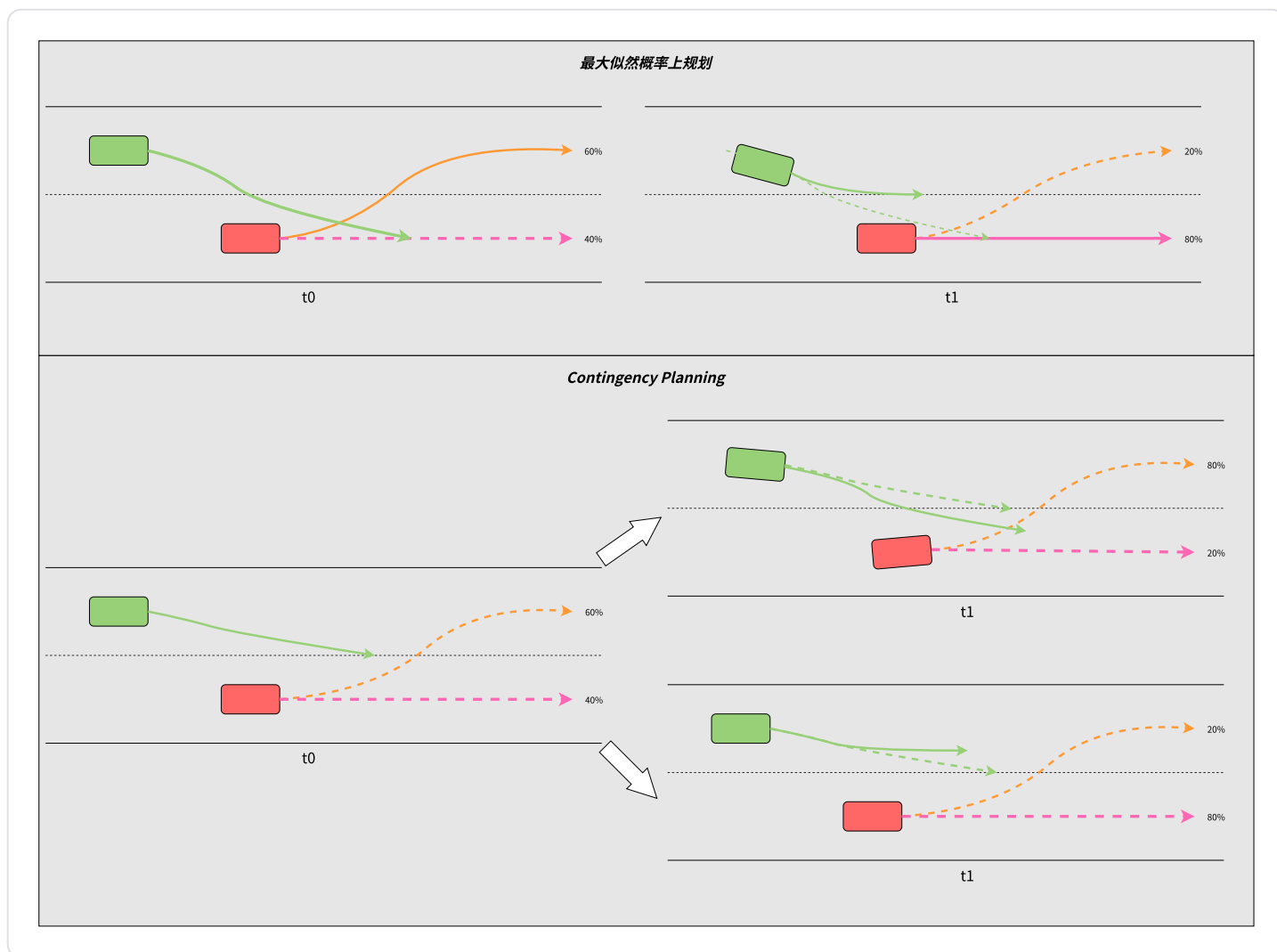
针对目标意图和轨迹的不确定性，不是简单的进行黑与白的选择，而是根据概率进行比例响应。

不同于常规的确定性规划，应变式的规划目标是规划结果能够最优的应对可能的不同未来场景。因此轨迹规划通过优化形式进行规划时，目标函数将按照不同场景分别计算然后根据场景概率加权起来得到总的cost.

典型例子1：当前侧前方车辆有两种可能意图40% 概率直行，60%概率变道

我们为自车规划一条最优轨迹，
$$\text{egotraj} = \text{Min}[\text{cost}(\text{pair}(\text{ego_traj}, \text{直行Traj})) * 60\% + \text{cost}(\text{pair}(\text{ego_traj}, \text{变道Traj})) * 40\%]$$

而随着下一帧对应概率的变化，自车的轨迹也会相应的根据概率的变化相应的比例变化。



典型例子2：纵向博弈下的Contingency Planning

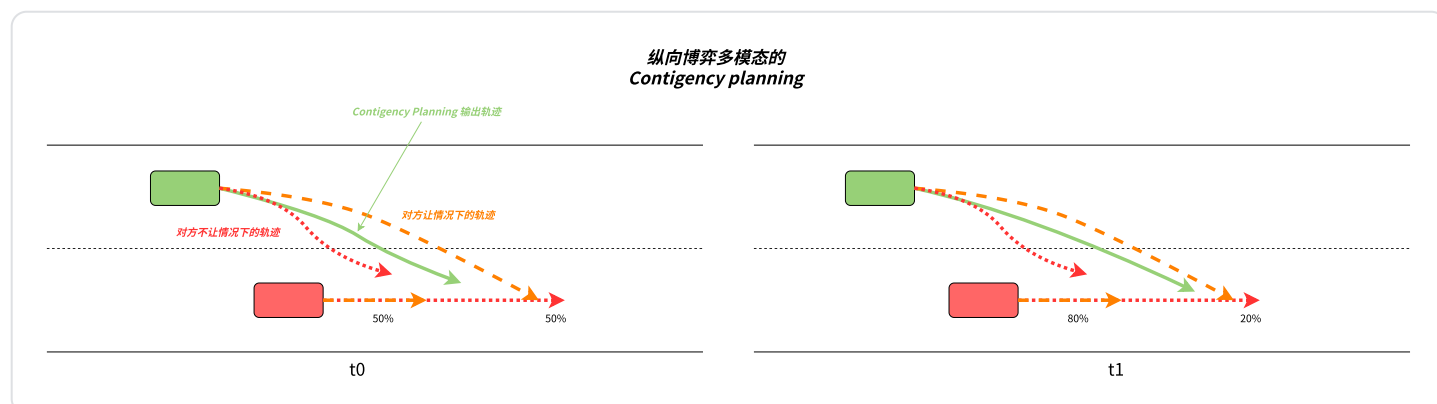
以之前介绍的斯塔伯格博弈基础上的意图估计算法为例：算法输出的是博弈目标在让与不让下的概率估计。那在这样的概率输出下，如何规划轨迹？

一样可以使用contingency planning. 分别针对估计的目标让与不让的轨迹，进行自车轨迹的优化计算：

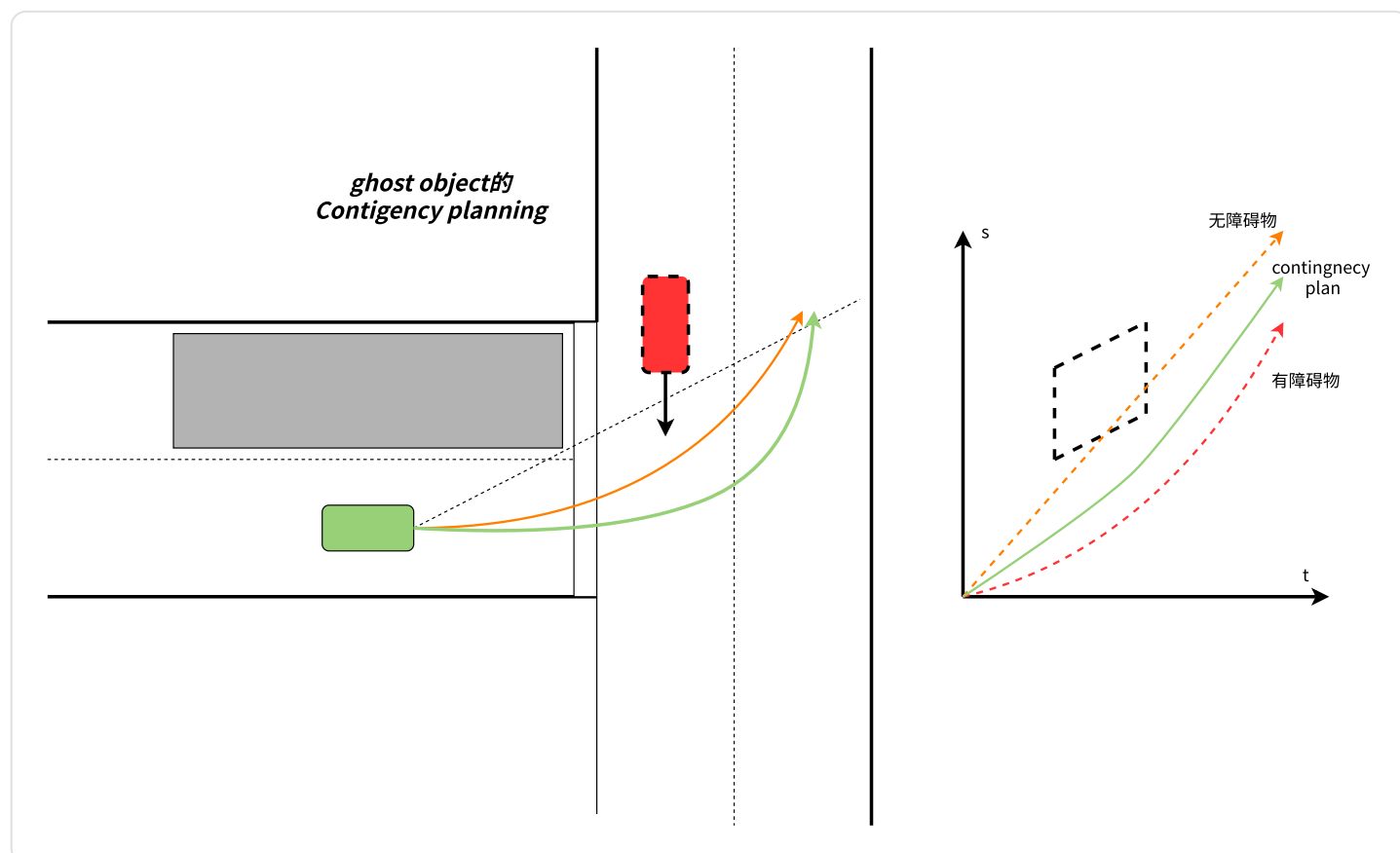
$$\text{egotraj} = \text{Min}[\text{cost}(\text{pair}(\text{ego_traj}, \text{让Traj})) * \text{让的概率} + \text{cost}(\text{pair}(\text{ego_traj}, \text{不让Traj})) * \text{不让的概率}]$$

这种规划方式规划出来的轨迹更为灵活：

- 针对不同的可能，轨迹不是黑与白的区别，针对不断变化的概率不确定性，自车的轨迹最大程度的保证的稳定性
- 在保证安全的前提下，逐渐逼迫博弈方，为自车赢得最大的博弈胜利空间



典型例子3：遮挡区域的潜在障碍物



2.2 轨迹树规划

contingency Planning 虽然在一定程度上解决了环境不确定性对自车轨迹的影响，但是还存在两个缺陷：

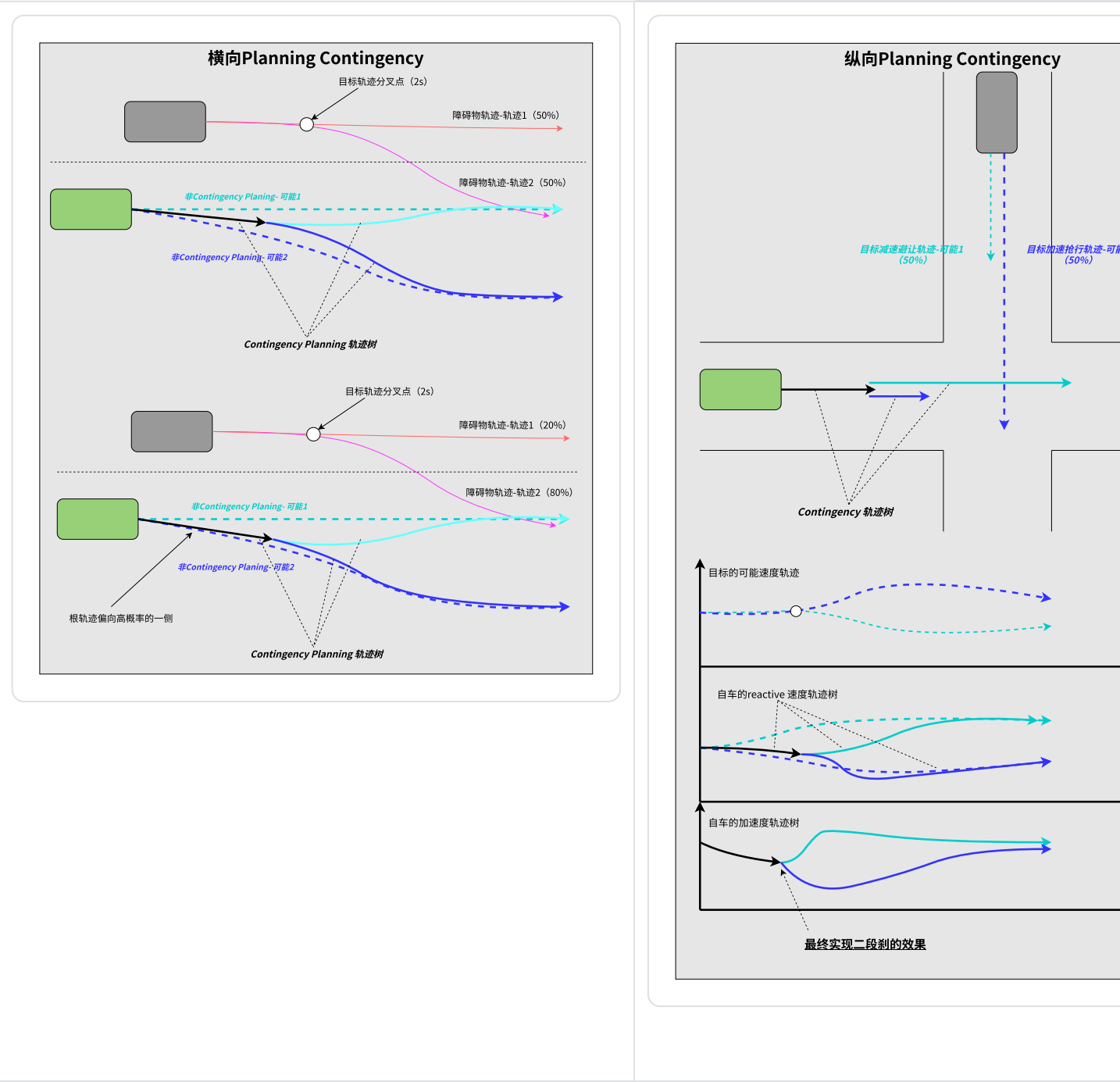
- 只能考虑1-2个不确定性，当不确定性较多时，这些不确定性又会相互影响，优化函数很快会复杂到无法处理。

- 当目标意图概率变化时，自车轨迹的近距离点依然可能有跳变，因为我们是直接将自车完整轨迹与不同可能模态下的轨迹进行加权组合规划的，所以不同的未来的可能轨迹将直接影响近距的轨迹点。

这就带来了轨迹树的概念：

人类开车时，不会立即对未来的可能事件，立即做出反应，而是预留出可能的反应空间即可。即眼前的规划，只要能够在可控的代价下应对潜在风险即可，不需要立即做出反应。

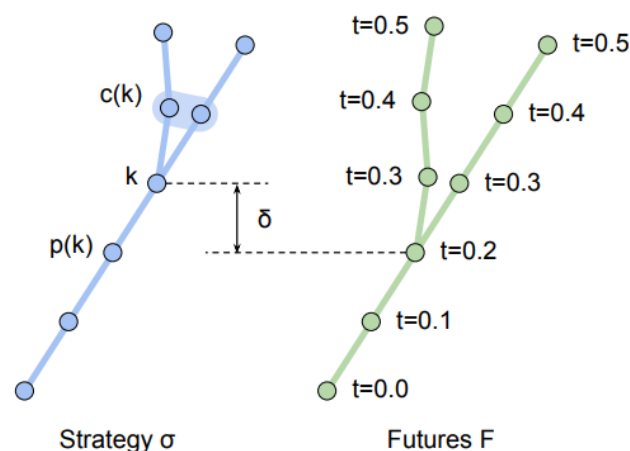
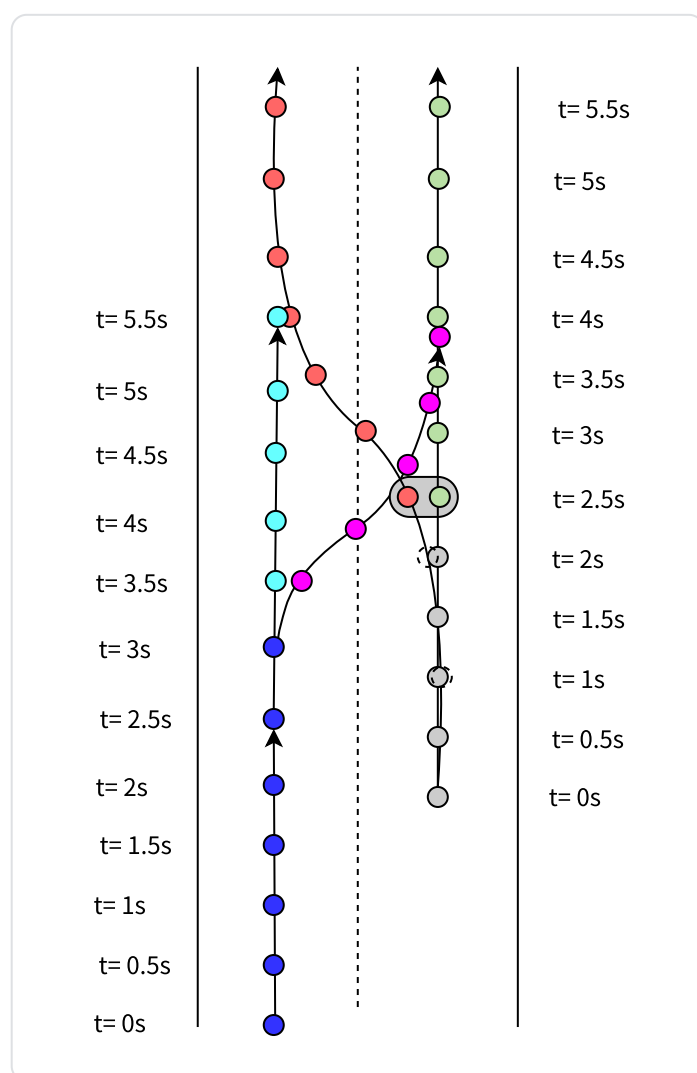
既然未来的可能模态存在不确定性，我们不需要立即显式的考虑，而是隐式的间接考虑就可以了。在保证安全的前提下，对未来可能出现的情况，进行一定的准备，但不是立即的响应，而是随机应变，即reactive planning.



我们将未来不同时间点可能出现的各种不确定性以树分叉的形式来表达，这个是和传统Contingency Planning的主要区别。这样整个规划空间不再是一个个独立的组合平行空间，而是随着不确定产生的

时间/位置点，产生一个规划空间树。

- 而针对每一个可能的模式分叉，自车的轨迹也相应的进行一次分叉。
- 随着模式分叉的时间、位置点不同，自车的轨迹也可以在不同的点进行相应进一步分叉，这样自车轨迹就成形成了一颗树状的结构
- 而树的根部的轨迹才会用来传给控制，自车轨迹的树枝，树杈不会直接影响根轨迹，而是通过车辆运动学约束和边界约束间接的向前传导到根轨迹。
- 环境的模式可以有多种形式：
 - 动态目标不同意图下的path（LK，LC的概率）
 - 博弈目标的不同意图下的speed profile（让和不让的概率）
 - 遮挡空间内，存在的潜在障碍物（障碍物存在概率）
 - 道路元素的不确定性（视觉建图中，产生的远处道路信息的不确定性）



2.2.1 环境分叉点的设计

不同于传统的轨迹规划，我们把不同模式下的轨迹总是从当前点位置开始分叉的，在轨迹树规划中，我们尽量将环境的分叉推迟，同样形成树状的表示。比如上图中，目标的两个模式的轨迹，我们根据

一定的规则，仅将横向位置达到车道线 / 纵向位置达到足够大的差异时，才进行分叉，之前的轨迹点处理为共用的轨迹根。

而遮挡区域的潜在目标则被处理为在潜在出现目标的位置点进行分叉：目标存在的轨迹，目标不存在则没有轨迹。

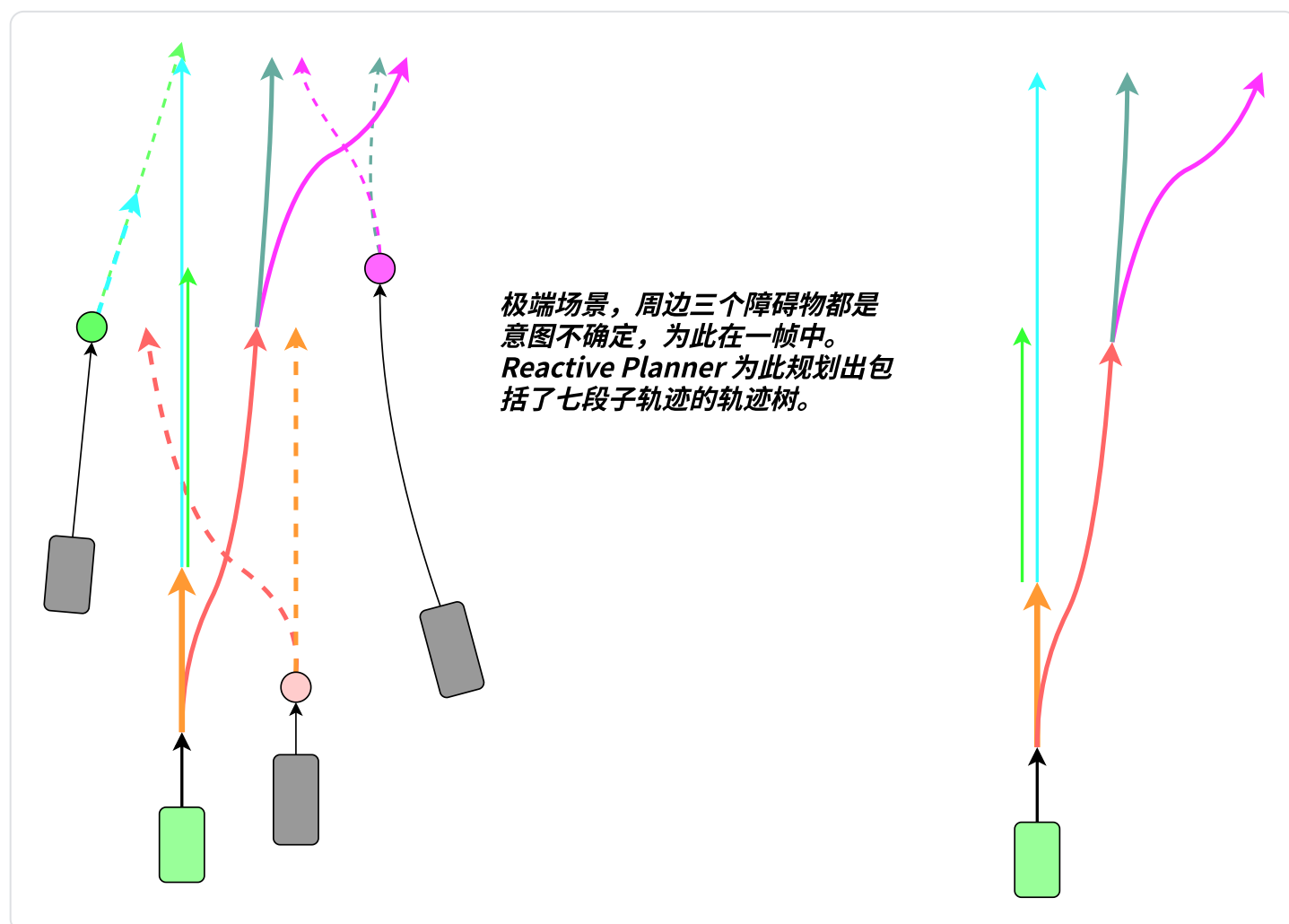
2.2.2 自车轨迹树分叉点的设计

在分叉点上的自车分叉决策：

在自车轨迹树分叉点的设计上，则可以根据决策规划架构的设计，有不同方式的处理：

- 根据前一步事先确定的决策结果，进行相应的规划空间分叉（比如上图中，已经决策了车道保持，则在相应的分叉点进行速度分叉）
- Or 直接融合多步决策，把多步多模态决策做到分叉点上（比如上图中，分别进行车道保持速度分叉，或者进行变道不变道的分叉），这样的话这分叉点后的子轨迹就有了两个模态了。这样相当于把决策规划完全融合起来了，决策用来决定在轨迹上进行什么样的分叉。而最后到底选择什么样的决策，可以根据最后不同模态下，整颗树的cost 大小来选择最优决策序列。

与EPSILON这样的采样决策来对比，由于我们是精确知道在什么时间/位置点需要进行决策变化（不像EPSILON中，只能固定的每2s 采样一个策略变化），因此在同样决策空间下，这种方式需要更少的算力，同时规划考虑的颗粒度更细。



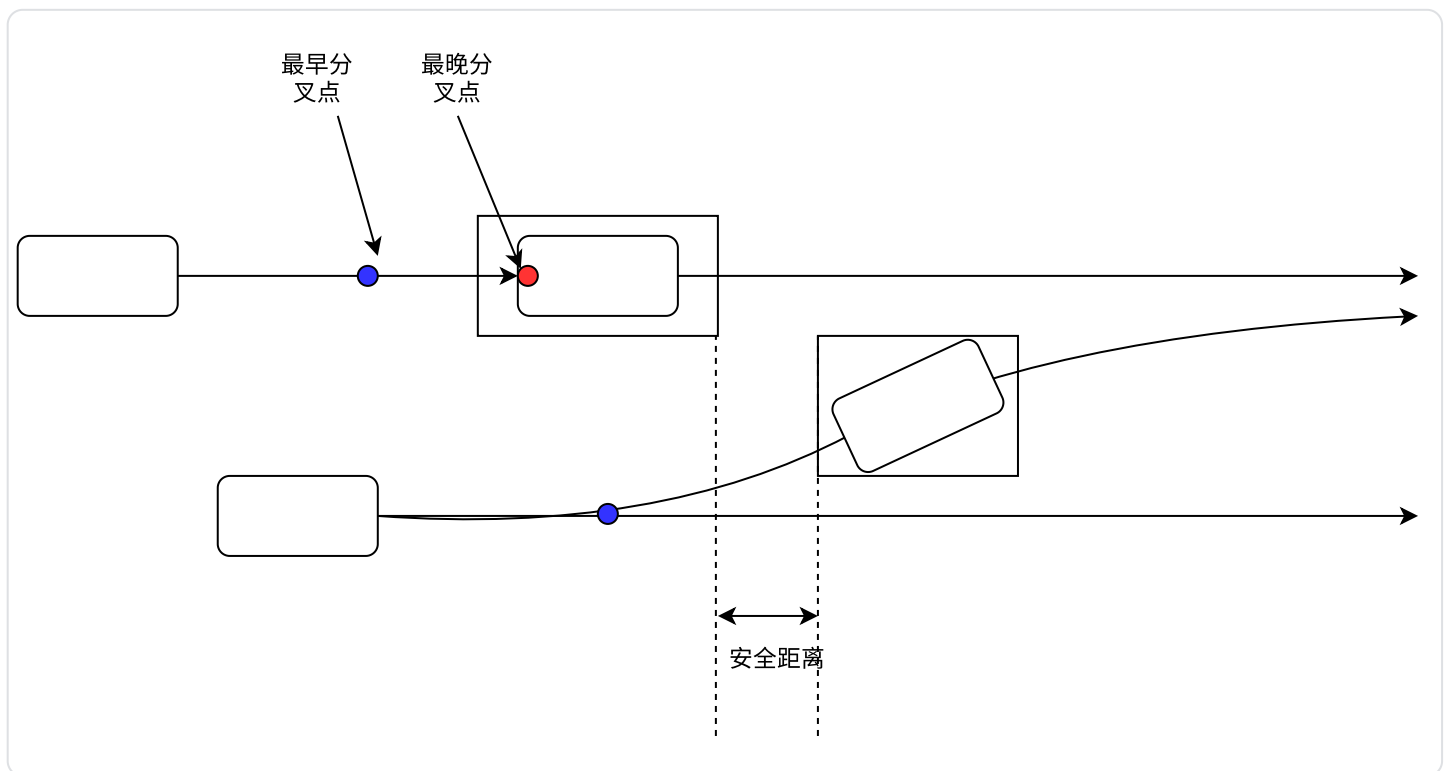
分叉点的选择：

理论上，环境在什么时间点，或位置点产生分叉，自车轨迹就应该在同样的时间点进行分叉。但为了提高根轨迹的稳定性，需要尽量把分叉点往后推（如果不确定未来会发生什么，只要还有响应的空间，那就先不管，只是先做些应对准备）。举个例子：

如果目标的横向Path存在不确定性，则：

- 先将目标预测的轨迹点，前部接近的一段合并后，形成轨迹root（记录对应的分叉时间点 t_{Branch} ）；以 $t_{Branch} + \sigma$ 时间为分叉点的最早点； σ 为自车感知系统的处理时延（实际目标意图确定后，到自车感知系统能够准确识别到确定意图的时间延迟，e.g. 0.1-0.2s）
- 而分叉的最晚点则可以根据安全范围设计，也就是说这个分叉点可以被尽量推迟到非进行应对处理不可的时间。对应最晚分叉点，可以按照类似RSS的安全距离定义（当然参数需要放宽），当横向距离产生overlap时，自车如果按照原始速度到达纵向安全距离位置的时间点。

这两个时间点的较大值作为最终的分叉点。



2.2.3 轨迹树的迭代收敛

回到最关键的点，自车的各个轨迹段并不能独立的进行规划，因为其受到前后段的子轨迹上的运动约束和空间约束的影响。

传统的Piecewise jerk 优化方法，只能给定确定cost function. 当我们存在分叉点时，直接使用QP求解的方式就无法实现了。需要使用ILQR求解，由于我们是在每一步上独立的对cost function进行寻优，且计算Action value时，可以根据概率作为权重来计算后续步的value。

- 当在子轨迹优化时，我们可以继续按照CILQR的方法来进行轨迹优化，且dynamics 和cost的梯度和hessian矩阵按照子轨迹上各自点状态计算
- 针对不同的障碍物意图轨迹，我们会处理成不同的约束条件施加到子轨迹上，对子轨迹进行优化

- 但是当backward pass 到达分叉点的时候，Q函数计算就需要按照概率同时考虑两个不同分支V function 来优化根轨迹。
- 由于两个子轨迹的state value function 同时会影响分叉点的优化状态，因此父轨迹的优化结果会根据概率同时考虑不同的未来可能，从而实现应变式规划。

我们回顾下，上一次介绍的ILQR的backward推导：

对于没有分叉的轨迹点，常规的Backward 推导：

$$\delta Q(k) = \frac{1}{2} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} l_{xx} & l_{xu} \\ l_{ux} & l_{uu} \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} + \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix}^T \begin{bmatrix} l_x \\ l_u \end{bmatrix} + \delta V(k+1) \quad (13)$$

$$\text{而 } \delta V(k+1) = \frac{1}{2} \delta X_{k+1}^T V_{k+1} \delta X_{k+1} + \delta X_{k+1}^T v_{k+1} \quad (14)$$

$$\text{由于 } \delta X(k+1) = \begin{bmatrix} f_x & f_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U_k \end{bmatrix} \quad (15)$$

而在轨迹树迭代优化时由于分叉节点有两个子节点，所以其推导时, 分叉节点的action value需要按概率加权计算(P1,P2 为和为100%的 模态轨迹概率)

$$\delta Q(k) = P1 \left(\frac{1}{2} \begin{bmatrix} \delta X_k \\ \delta U1_k \end{bmatrix}^T \begin{bmatrix} l_{xx} & l1_{xu} \\ l1_{ux} & l1_{uu} \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U1_k \end{bmatrix} + \begin{bmatrix} \delta X_k \\ \delta U1_k \end{bmatrix}^T \begin{bmatrix} l_x \\ l_u \end{bmatrix} + \delta V1(k+1) \right) + \\ P2 \left(\frac{1}{2} \begin{bmatrix} \delta X_k \\ \delta U2_k \end{bmatrix}^T \begin{bmatrix} l_{xx} & l2_{xu} \\ l2_{ux} & l2_{uu} \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U2_k \end{bmatrix} + \begin{bmatrix} \delta X_k \\ \delta U2_k \end{bmatrix}^T \begin{bmatrix} l_x \\ l_u \end{bmatrix} + \delta V2(k+1) \right)$$

$$\text{而 } \delta V1(k+1) = \frac{1}{2} \delta X1_{k+1}^T V1_{k+1} \delta X1_{k+1} + \delta X1_{k+1}^T v1_{k+1}$$

$$\text{而 } \delta V2(k+1) = \frac{1}{2} \delta X2_{k+1}^T V2_{k+1} \delta X2_{k+1} + \delta X2_{k+1}^T v2_{k+1}$$

$$\delta X1(k+1) = \begin{bmatrix} f_x & f1_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U1_k \end{bmatrix}$$

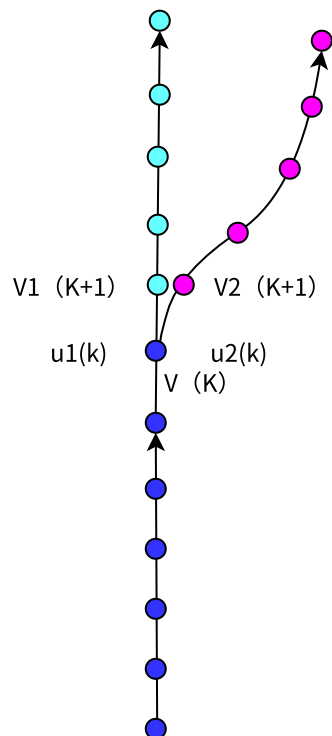
$$\delta X2(k+1) = \begin{bmatrix} f_x & f2_u \end{bmatrix} \begin{bmatrix} \delta X_k \\ \delta U2_k \end{bmatrix}$$

进行整理后，分叉点上

$$\delta Q(k) = \frac{1}{2} \begin{bmatrix} 1 \\ \delta X_k \\ \delta U1_k \end{bmatrix}^T * P1 * \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q1_{xu} \\ Q_u & Q1_{ux} & Q1_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta X_k \\ \delta U1_k \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 \\ \delta X_k \\ \delta U2_k \end{bmatrix}^T * P2 * \\ \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q2_{xu} \\ Q_u & Q2_{ux} & Q2_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta X_k \\ \delta U2_k \end{bmatrix}$$

要使得 $\delta Q(k)$ 取极小值：

$$\frac{\delta Q(k)}{[\delta U1_k \ \delta U1_k]} = 0$$



分叉点设计不同的控制量，来应
对不同的可能轨迹

$$Q1(K) = q1(K) + V1(K+1)$$

$$V1(K) = \text{Min}(Q1(K))$$

$$Q2(K) = q2(K) + V2(K+1)$$

$$V2(K) = \text{Min}(Q2(K))$$

$$Q(K-1) = q(k) + V1(K) * 30\% + V2(K) * 70\%$$

$$V(K) = \text{Min}(Q(K-1))$$

2.2.4 对于轨迹树规划的理解

- 以牺牲小概率模态下的体验（舒适性）来提高大概率模态下的体验的方式，以牺牲未来不确定模态下的舒适性来提高眼前规划的稳定性
- 非必要，不决策；如果未来有多种可能性，但当前不能确定时，尽量推迟做决策，并对眼前的规划进行预防性的处理，以保证能够以较小成本应对可能的未来（Reaction Causality）