

andes 决策规划个人笔记

目录

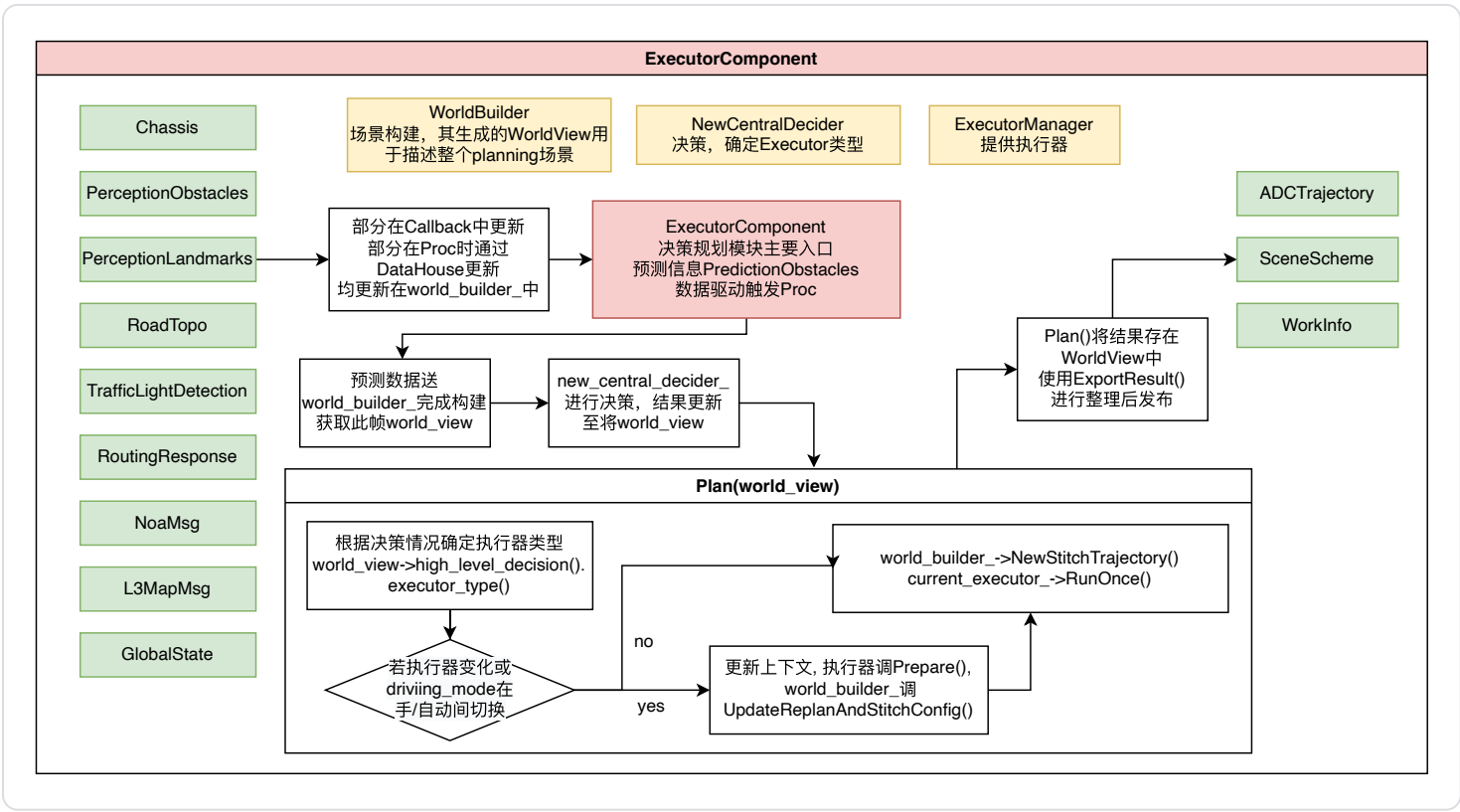
- Component
 - ExecutorComponent
 - StateMachineComponent
- 数据结构
 - WorldView
 - ReferenceLineInfo
 - BaseContext
- 工具类
 - WorldBuilder
 - NewCentralDecider
 - ExecutorManager
 - Executor & Stage & Task
- 具体场景下的Executor & Stage & Task
 - CityCruiseExecutor
 - LaneFollowCityStage
 - LKPathBoundaryDecider
 - LKFixedPiecewiseJerkPathOptimizer
- 算法原理
 - Nudge决策整体流程
 - 基于模型的Nudge决策
 - LK Path 边界生成
 - LK Path QP建模
 - 参考线相关
 - TrajectoryStitcher



Component

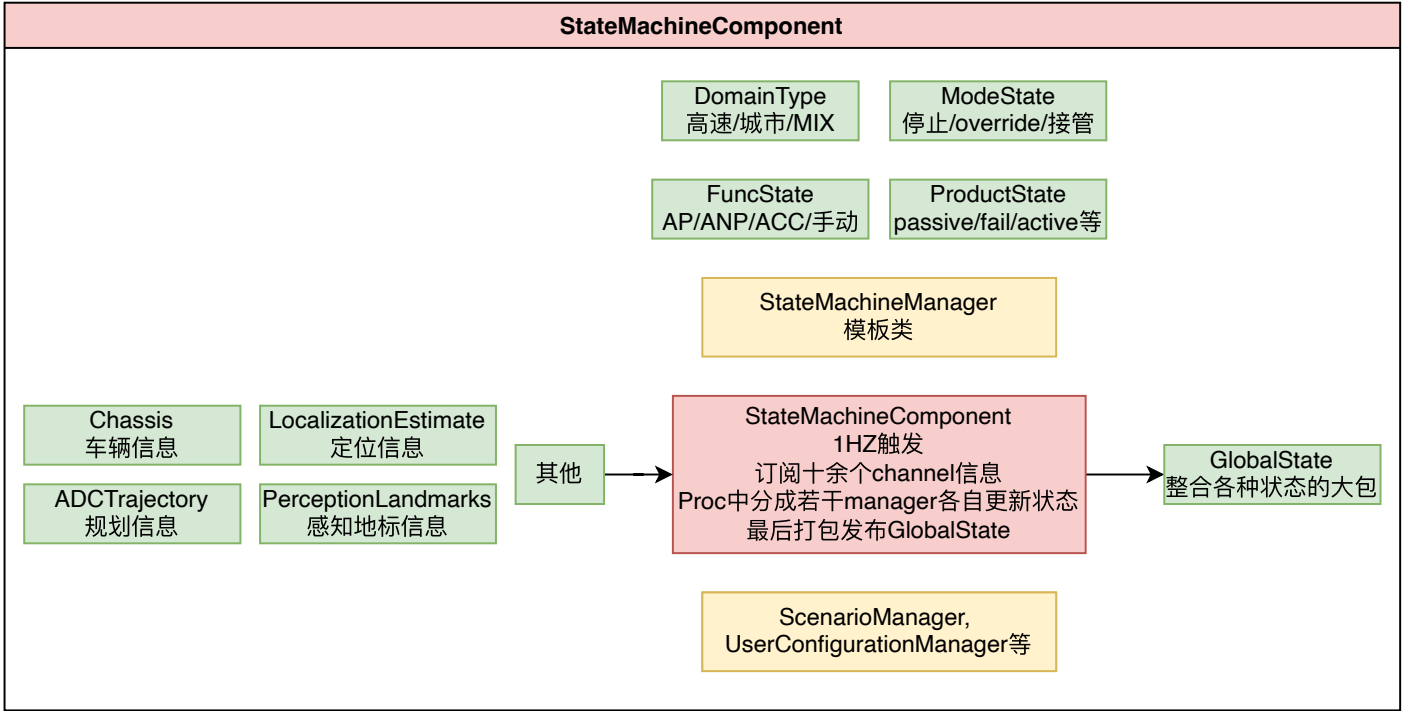
ExecutorComponent

PNC模块最外层流程



StateMachineComponent

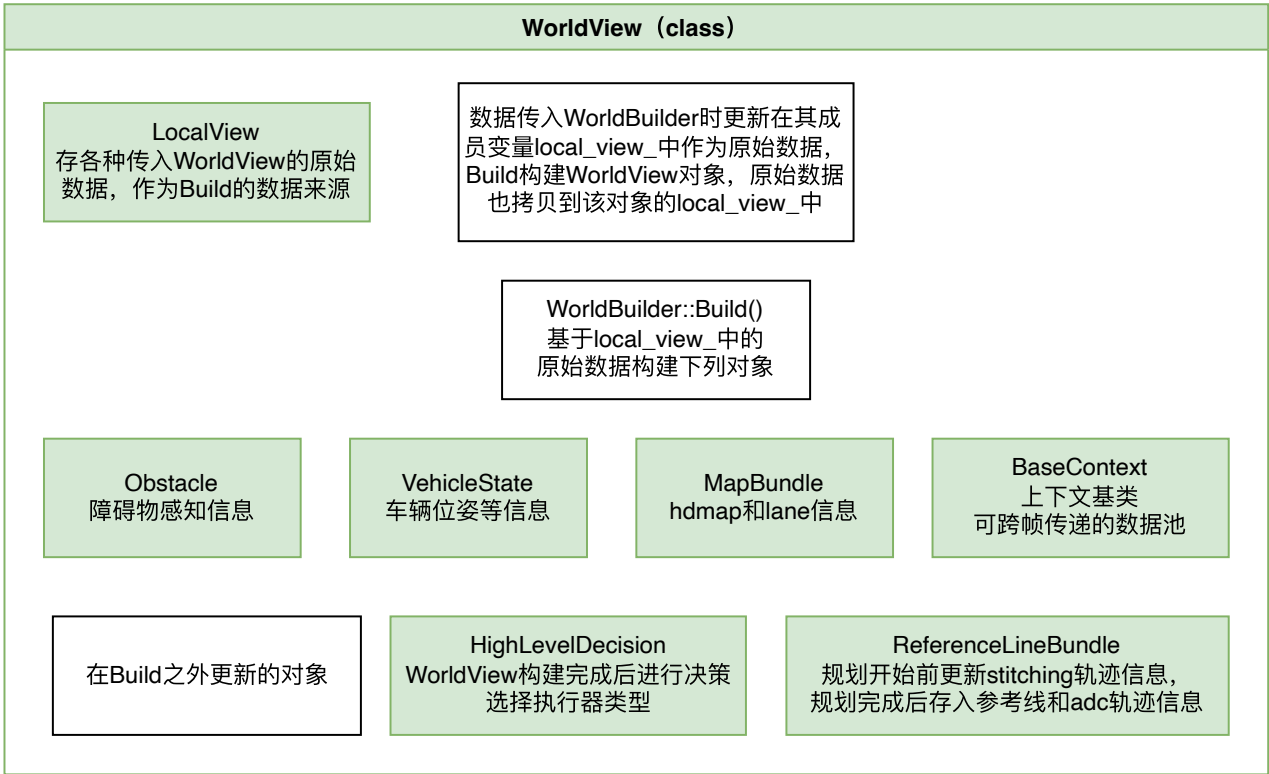
最外层的自动驾驶模式切换相关的状态机, 应该不属于pnc主体模块



数据结构

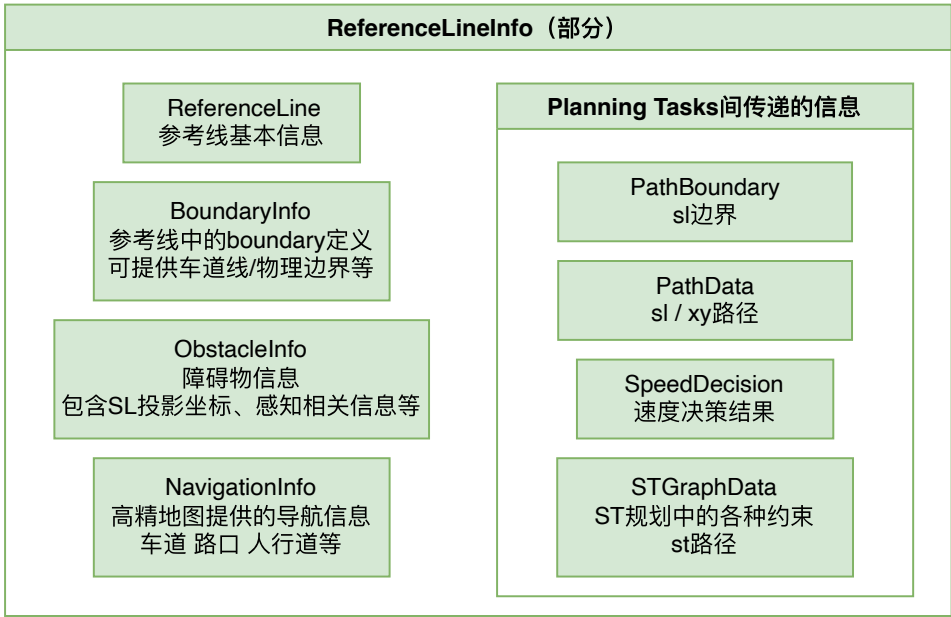
WorldView

存储决策规划相关的几乎所有其他模块传入信息，本体实例在WorldBuilder中，每帧重新构建



ReferenceLineInfo

参考线数据结构，决策规划Tasks的主要操作对象，每帧重新构建，在Tasks间传递中间数据除了参考线本身外，还整合了很多WorldView中的其他信息，并投影到参考线对应的SL坐标系下



BaseContext

整个决策规划流程主要依赖WorldView和其中的ReferenceLineInfo进行各类数据的存取。但除了障碍物，自车，地图，车道/参考线等基础数据外，还有许多与算法或业务逻辑相关的其他数据，这些信息统一在WorldView的Context中存取。

可以在帧间传递

每个Executor对应的具体场景，都有一个继承自BaseContext的派生类，用来更具体地定义在该场景中用到的上下文信息。

工具类

WorldBuilder

ExecutorComponent成员，用于WorldView的构建

根据更新好的一系列数据(包括且不限于ExecutorComponent的输入)，通过Build() { CreateObstacles(), BuildVehicleState(), BuildMapBundle(), BuildContext() } 构建WorldView实例。

其为WorldView的友元类，通过直接更新WorldView对象的私有成员进行构建

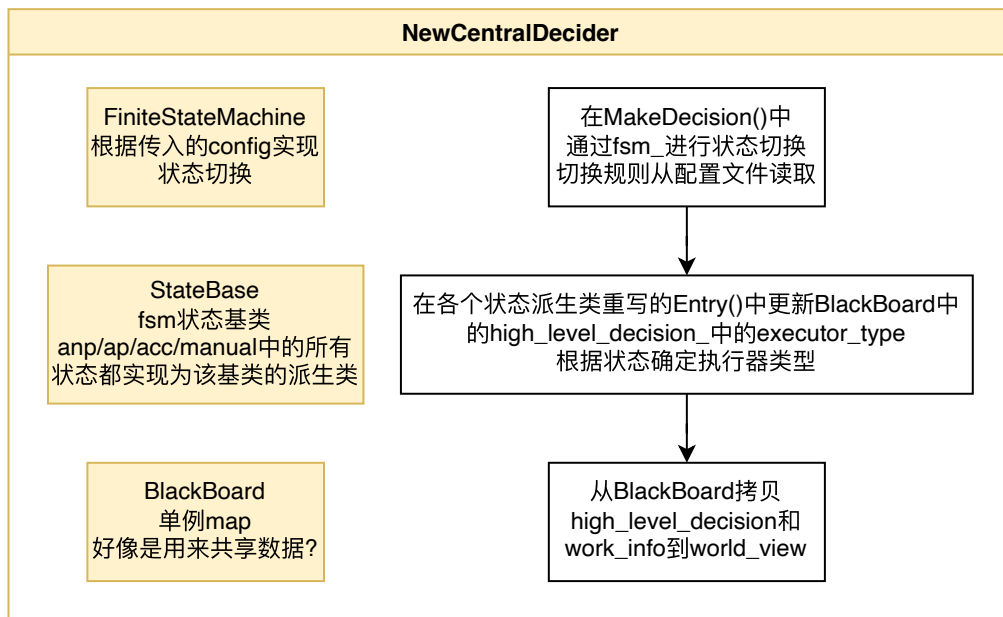
NewCentralDecider

在ExecutorComponent Proc流程中进行的决策，用于场景决策，确定后续使用的executor。用fsm明确定义了各状态间的转换关系。

个人理解和StateMachineComponent功能有一定重合，也负责Domain（H/U）和Func（ANP/AP/ACC/MAN）间的切换。

但其更接近执行端，还可进行具体细分场景的状态切换（LineKeeping/左右转/泊车等），并为其选择不同的执行器。

- 状态派生类定义：modules/decider/state/
- 状态机切换规则的配置：onboard/conf/new_decider/



ExecutorManager

提供根据执行器类型获取对应executor的接口

Init时根据配置文件在executor_table_中注册并按配置参数初始化了所有Executor

配置文件：onboard/conf/executor/executor_manager.config

另外该类在Init时还在TaskFactory中用默认参数初始化了所有Task

Executor & Stage & Task

每个场景对应一个执行器，每个执行器内包含若干stage，每个stage由一系列task组成，tasks共同操作RefLineInfo完成PNC算法

各executor及其包含stage和task的配置文件：onboard/conf/executor/

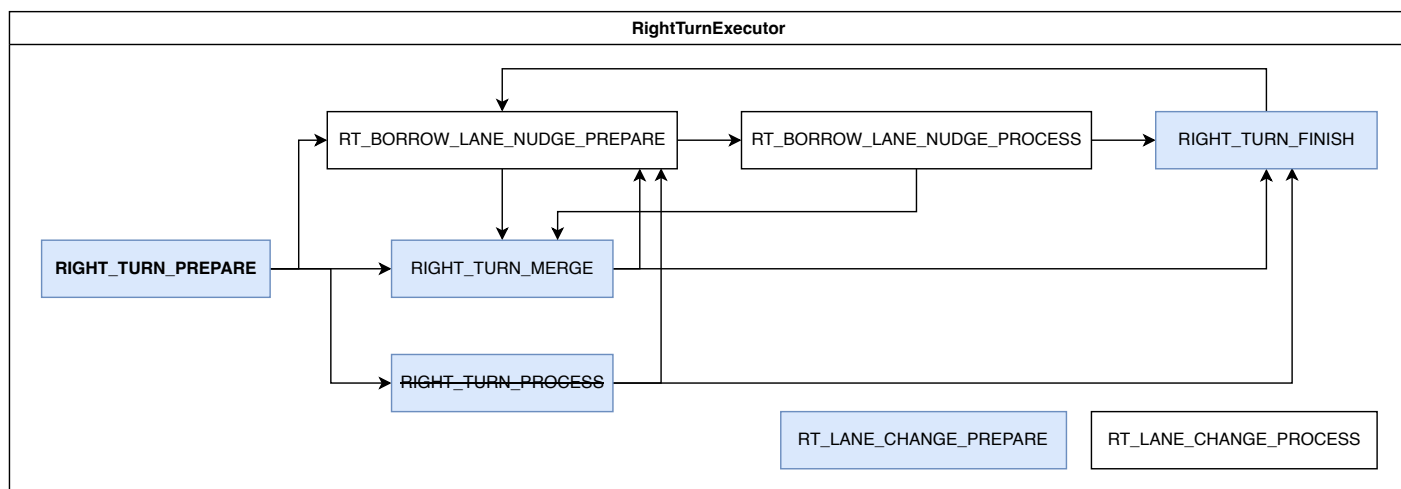
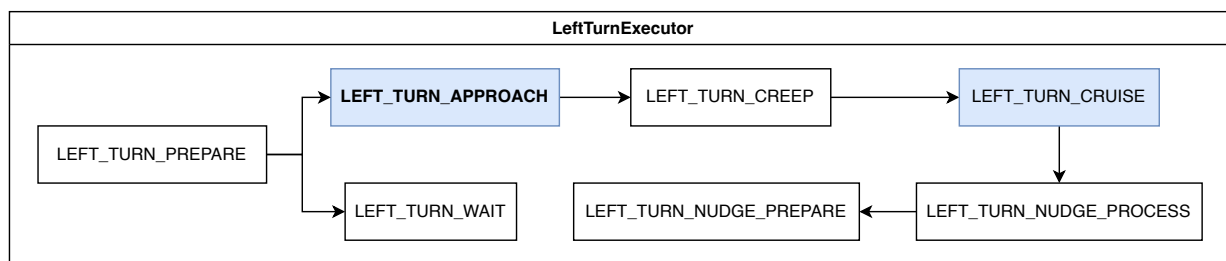
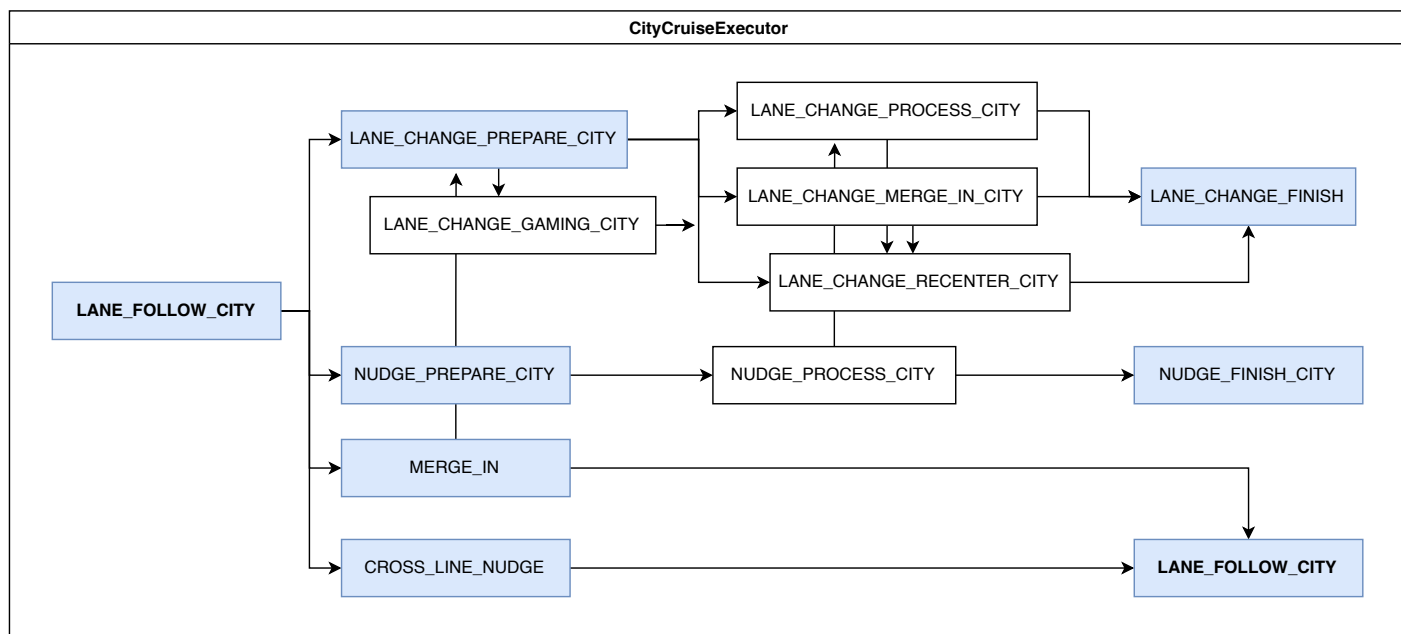
- Executor的切换由场景决策确定，Stage的切换由其内部决定，Task则按Stage中的定义顺序执行
- Stage主要用于定义各场景下不同状态需要调用的Task序列，以及状态切换规则和各种业务逻辑，故每个Stage独一无二，不被复用；

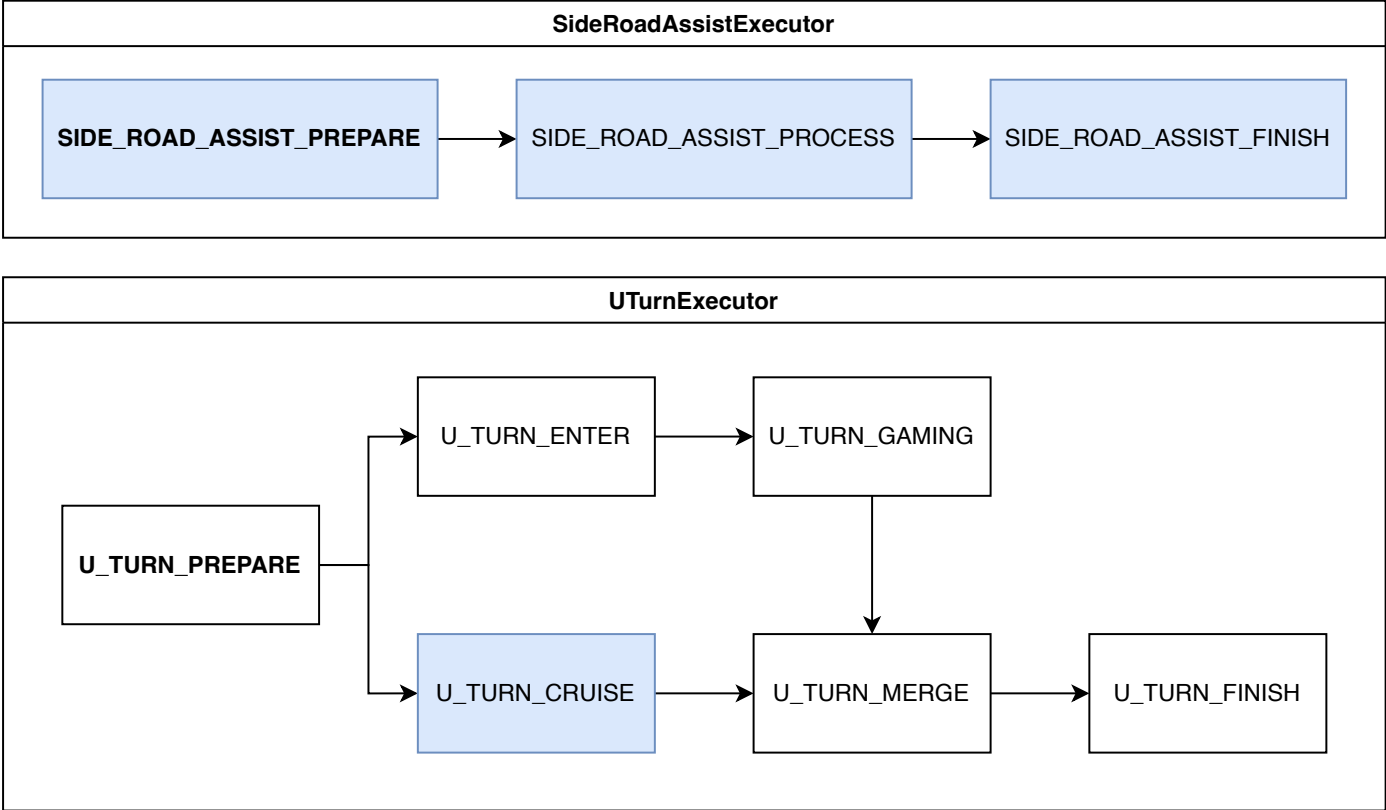
而Task作为处理较为独立问题的基础算法模块，有些Task会被多个Stage调用

- Executor在初始化时就实例化了其包含的所有Stage，而Stage通过TaskFactory获取初始化好的Task指针。所以其实在ExecutorComponent

完成Init时，所有写入配置文件的Executor/Stage/Task派生类就都已经实例化完成了，切换和调用的时候只是在各个层级的table中取对应指针

城市场景下的主要Executor和各自的Stage跳转（仅整理用到LK Path的）：





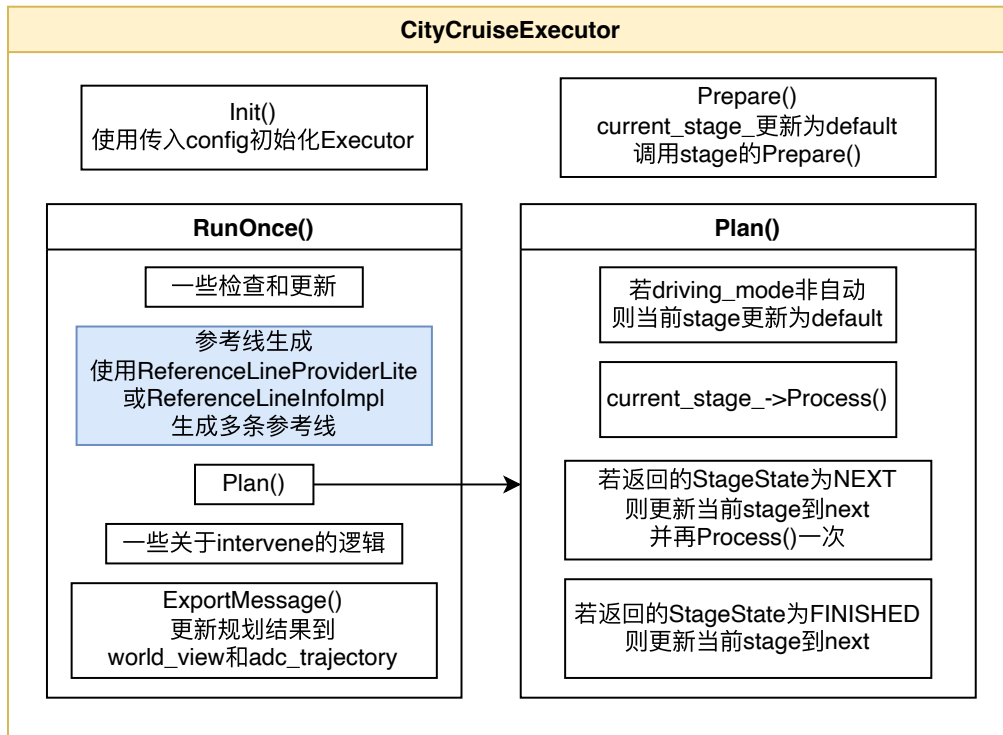
具体场景下的Executor & Stage & Task

CityCruiseExecutor

城市巡航场景执行器

实现基本架构，参考线生成，Stage切换(由其内部触发)，override相关等

通过ExportMessage()对外更新Stage执行后的一些情况变化



</> CityCruiseExecutor 下的 Stages :

C++

```

1 LANE_FOLLOW_CITY /
2 NUDGE_PREPARE_CITY / NUDGE_PROCESS_CITY / NUDGE_AWAY_CITY / NUDGE_FINISH_CITY /
3 LANE_CHANGE_PREPARE_CITY / LANE_CHANGE_GAMING_CITY / LANE_CHANGE_CHASING_CITY /
4 LANE_CHANGE_PROCESS_CITY / LANE_CHANGE_ONLANE_CITY / LANE_CHANGE_MERGE_IN_CITY /
5 LANE_CHANGE_RECENTER_CITY / LANE_CHANGE_FINISH_CITY /
6 MERGE_IN /
7 CROSS_LINE_NUDGE
8 // 这些Stages基本只在CityCruiseExecutor下被使用

```

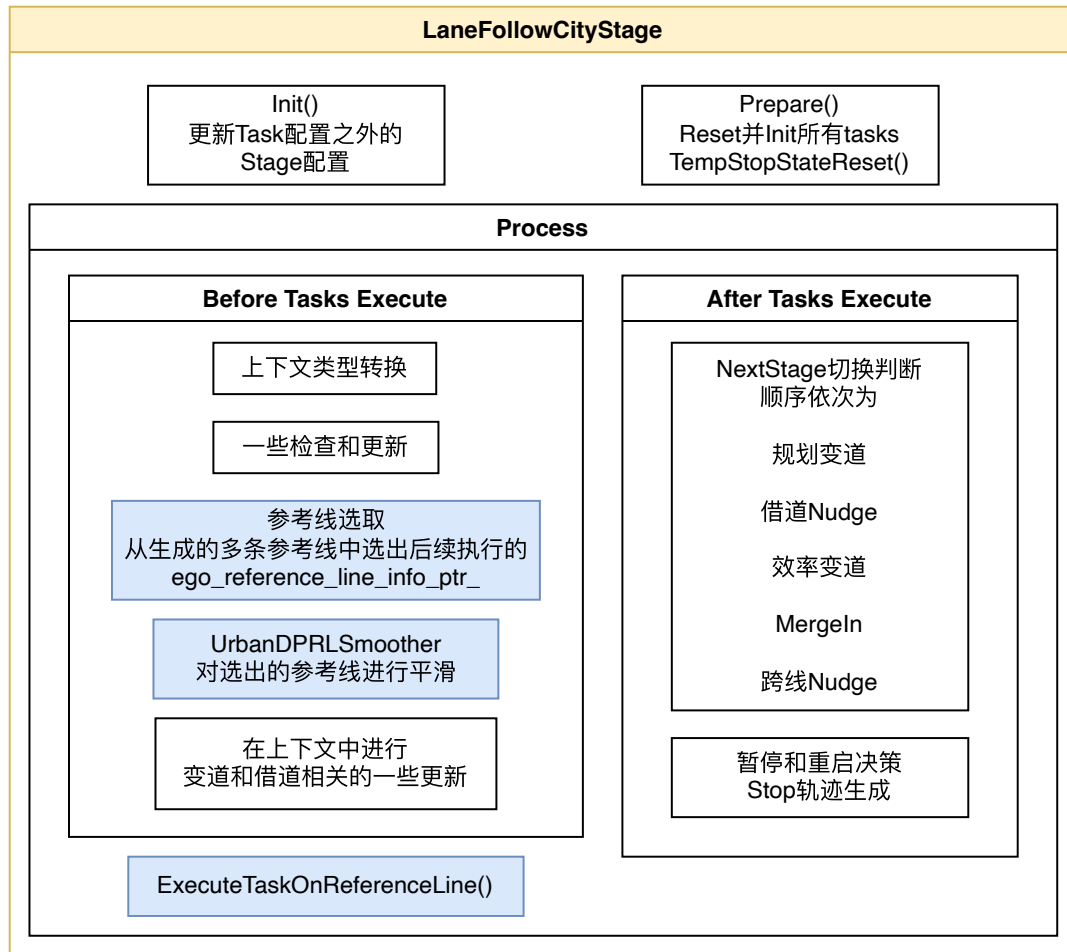
Executor的源文件在modules目录下，而Context，Stage和Task则在lib下，各种业务逻辑细节也主要实现在后面三个层级内

LaneFollowCityStage

城市巡航场景下的车道线保持状态

执行tasks前：转换上下文，选取并平滑当前参考线，进行一些变道相关信息的更新

执行tasks后：各个Stage的切换判断，若均不切换则保持当前状态，进行暂停和启动判断



</> LaneFollowCityStage下的Tasks:

C++

- 1 LK_PATH_BOUNDARY_DECIDER / LK_FIXED_PIECEWISE_JERK_PATH_OPTIMIZER /
- 2 NAVI_CIPV_DECIDER / OBSTACLE_INTERACTION_DECIDER / TRAFFIC_LIGHT_DECIDER / CROSSWALK_DECIDER /
- 3 LANE_CHANGE_TRAFFIC_FLOW_CALCULATOR / LANE_CHANGE_CITY_TRIGGER_DECIDER /
- 4 BORROW_LANE_NUDGE_TRIGGER_DECIDER / LANE_CHANGE_SPEED_DECIDER / ROAD_SPEED_LIMIT_DECIDER /
- 5 MERGE_IN_TRIGGER_DECIDER / NAVI_PF_SPEED_OPTIMIZER / NAVI_SPEED_OPTIMIZER
- 6 // 这些Tasks并不只在该Stage下被使用, 某些Task会被许多Stages复用

Decider和Optimizer都是Task的派生类, 各种Tasks又是他们的派生类

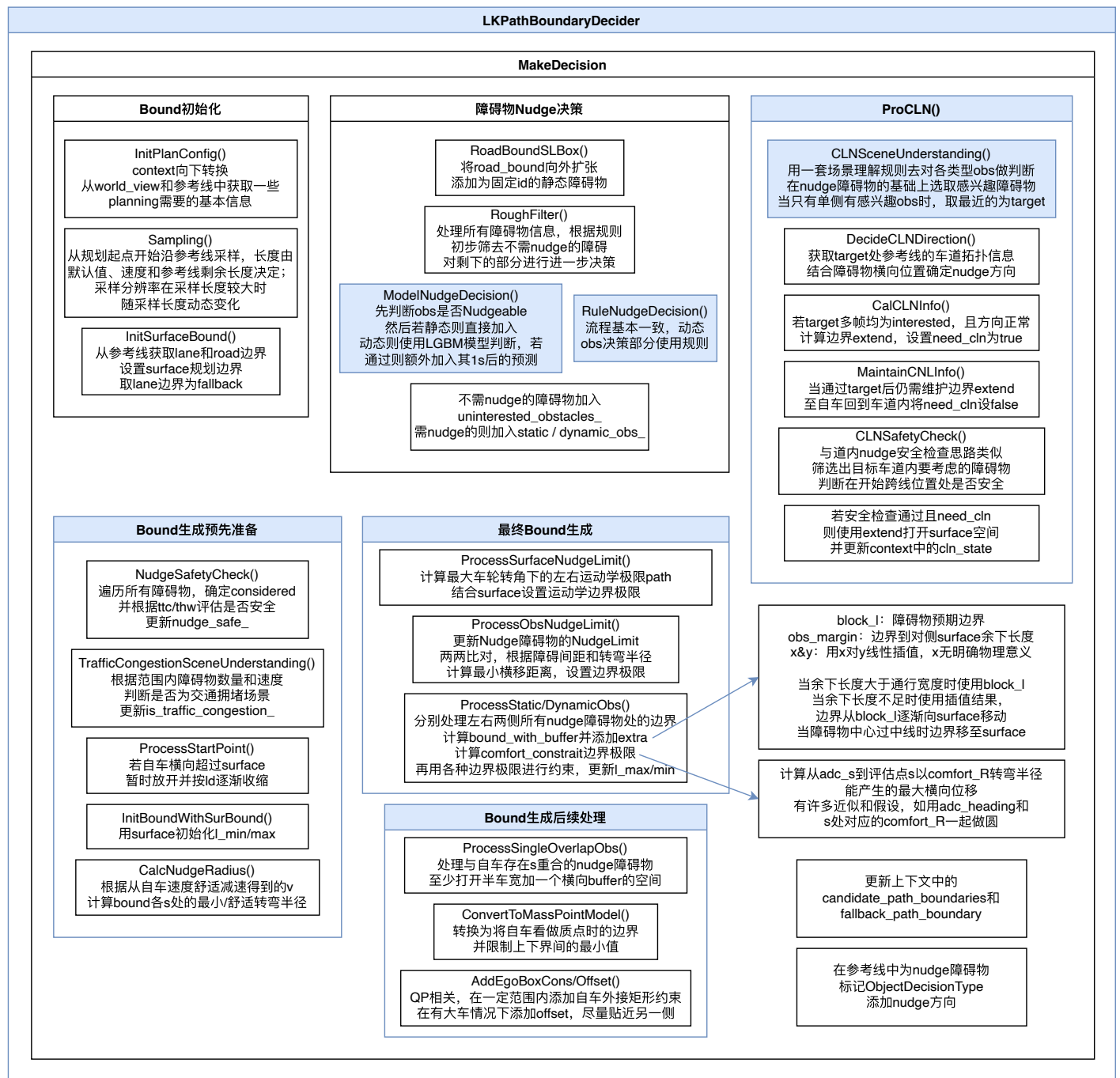
这层包装比较简单, 就是在Execute()里对输入做完整性检查, 然后分别调用MakeDecision()和Optimize(), 派生类分别实现这两个函数即可

LKPathBoundaryDecider

计算沿参考线的横向边界, 为后续优化求解提供横向上下界约束

相关功能为车道保持、道内nudge、跨线nudge

输入为参考线、障碍物、自车信息，输出为沿参考线的boundary；实现方式主要为在不同场景下设计不同规则对边界进行调整



备忘录

- 画图画出的是lane、buffer、质点模型边界；未画出surface，surface为min(lane, lane - 0.2(有近硬边界时), 硬边界 - 0.4)
- RoadBox是一定范围内硬边界的外接矩形，真正的硬边界可通过0.4的buffer找到
- 吃掉等效车宽的一定是ConvertToMassPoint上下同减
- 吃掉buffer的一定是某种NudgeLimit (or Overlap)

被调用情况

CITY_CRUISE_EXECUTOR:

LANE_FOLLOW_CITY、NUDGE_PREPARE_CITY、NUDGE_FINISH_CITY、
LANE_CHANGE_PREPARE_CITY、

LANE_CHANGE_FINISH_CITY、MERGE_IN、CROSS_LINE_NUDGE

LEFT_TURN_EXECUTOR:

LEFT_TURN_APPROACH、LEFT_TURN_CRUISE

RIGHT_TURN_EXECUTOR:

RIGHT_TURN_PREPARE、RIGHT_TURN_PROCESS、RIGHT_TURN_FINISH、
RT_BORROW_LANE_NUDGE_PREPARE、

RT_LANE_CHANGE_PREPARE、RIGHT_TURN_MERGE

SIDE_ROAD_ASSIST_EXECUTOR:

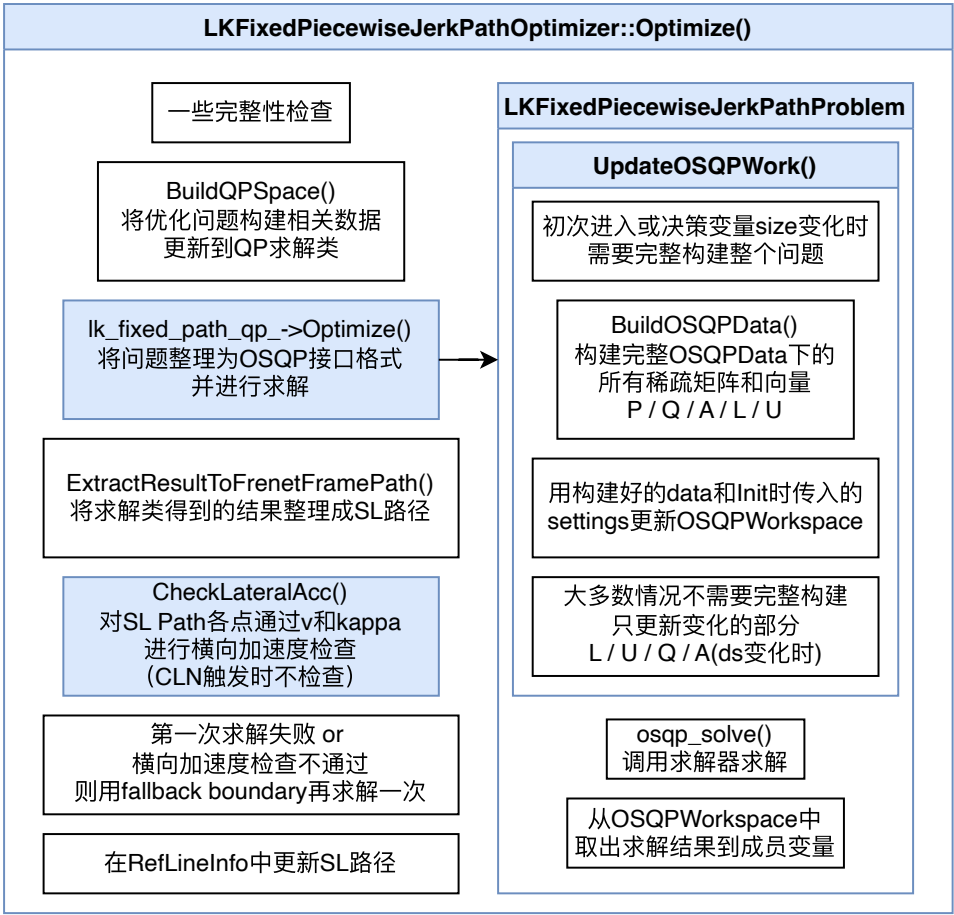
SIDE_ROAD_ASSIST_PREPARE、SIDE_ROAD_ASSIST_PROCESS、SIDE_ROAD_ASSIST_FINISH

U_TURN_EXECUTOR:

U_TURN_CRUISE

LKFixedPiecewiseJerkPathOptimizer

根据QP问题模型和LKPathBoundaryDecider给出的boundary等约束，整理并求解QP问题



加速度检查CheckLateralAcc: 根据 ref_kappa & dkappa 、 $l / dl / ddl$ 计算path在xy下的 kappa , 若 $\max(5, \text{adc_v})^2 * \text{kappa} < \text{max_lateral_acc}(0.9)$ 则fallback

// lateral acc = $v^2 * \text{kappa}$

// lateral jerk = $2 * v * dv/dt * \text{kappa} + v^2 * ds/st * \text{delta_kappa}/ds$

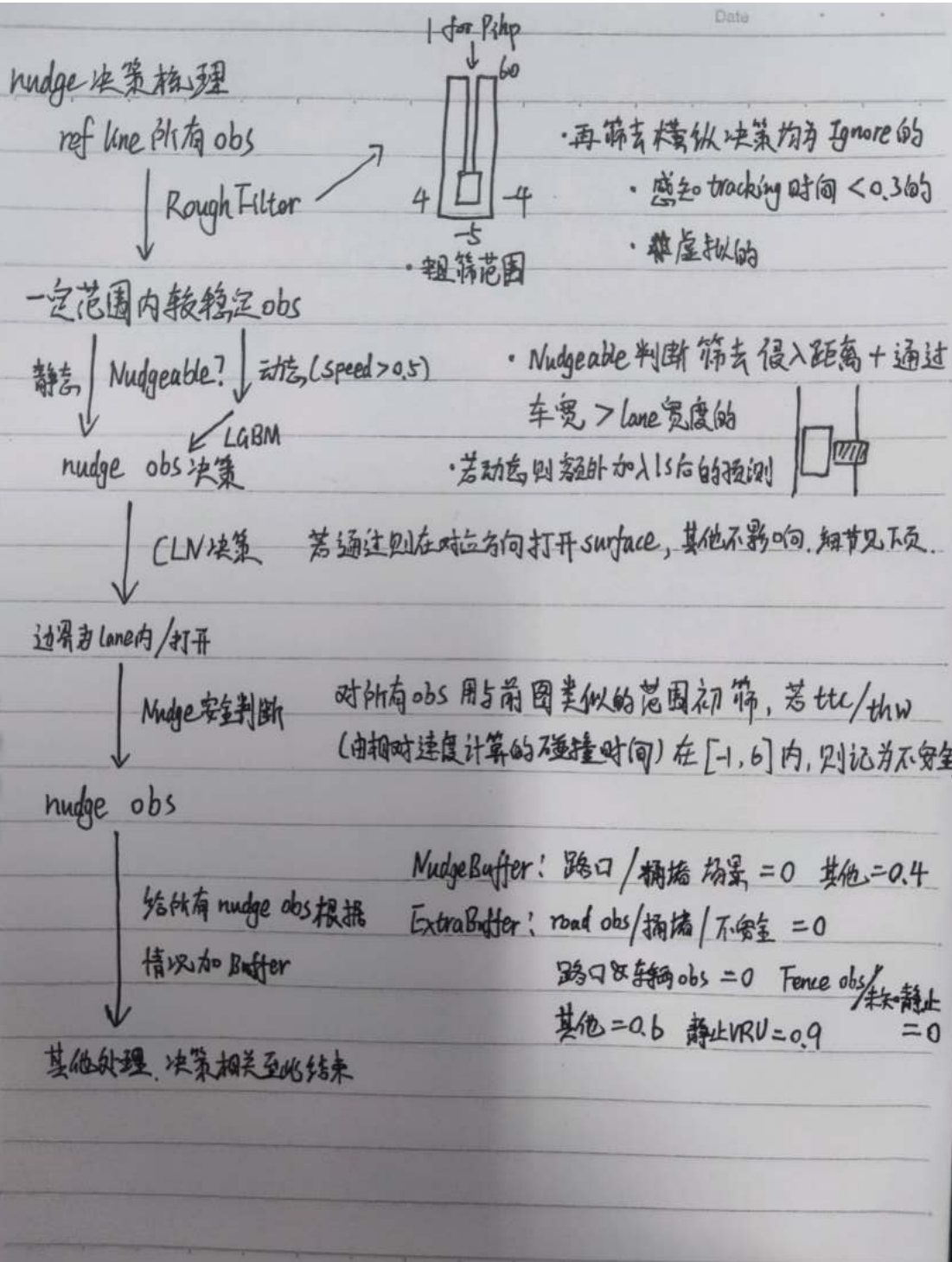
// lateral jerk = $2 * v * a * \text{kappa} + v^3 * \text{dkappa}$

计算 dkappa 和 lateral_jerk , 超出阈值则fallback

被调用情况和LKPathBoundaryDecider完全相同

算法原理

Nudge决策整体流程



CLN决策

前面得到的 nudge obs

判断是否感兴趣:

若为 VRU, 筛去以下情况:

• $|dl| \geq 0.2$

• 离 lane 较远

• 余下空间 > 1.2 上帧非 CLN, 且 ΔS 较远 $tcl > 10$ • $adc_sd > 5$ 且 $\Delta S < 15$ • $\Delta S < 10$

• obs 在 road 边缘外

若为 UNKNOWN / UNKNOWN_UNMOVABLE, 筛去:

• $obs.speed \geq 0.5$

• lane 边缘外

• 余下空间 > 1.0

若为 VEHICLE, 筛去:

• $obs.speed > 4$ • 入侵车道 > 1.75 • 余下空间 > 1.0 总结: ΔS 中距离, $obs.speed$ 较慢, ~~靠近~~ 入侵车道较多, 则标记

感兴趣 obs

↓ target 选择

若仅在左右中的一侧存在 interested obs, 则取其中 ΔS 最近的为 target

target obs

↓ 方向判断

结合车道拓扑信息, 判断 nudge 方向车道类型以及空间是否足够 (类型为 ROAD)

target obs & dir

↓ tracking

若 target obs 的 interested 计数 > 3 , 且 dir 不为 NONE, 则 need_cln (一个跨帧 tracking)

计算得到 extend

↓ 安全检查

对所有 obs,

consider 范围:

lane

←→

动态 $S: \infty$ 静态 $S: \text{extend 范围内}$

根据车道类型,

对后/前方 obs 的安全性

进行判断, 主要考虑

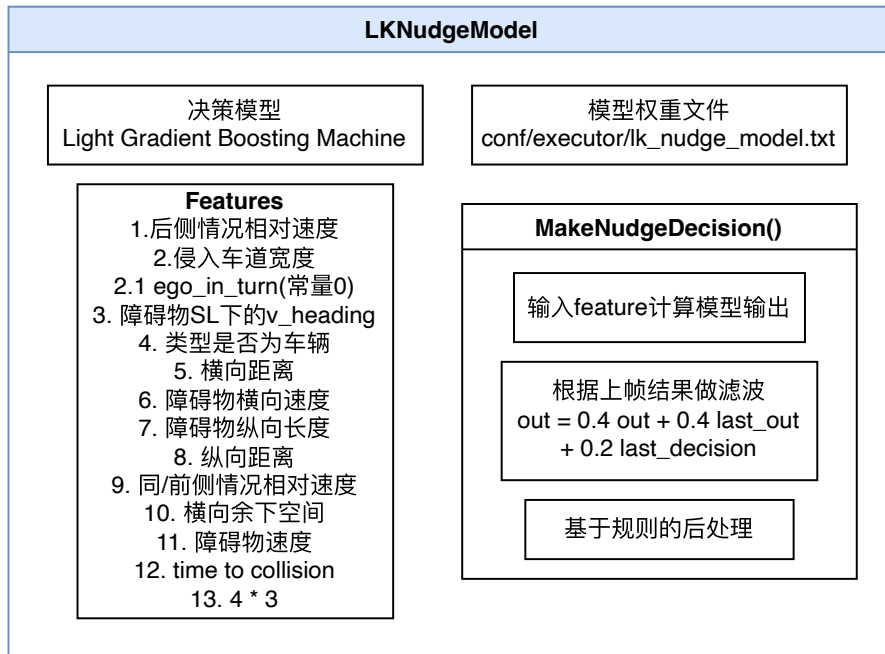
extend_start - S 附近

CLN 决策通过, 打开 surface 空间

Q:

MaintainCNInfo() 中的 move_dist 是指整个 SL 坐标系移动的 dist 么?

基于模型的 Nudge 决策



基于模型的Mudge决策.

模型使用的基本原理是决策树 (CART, 分类 & 回归树)

核心模型是 GBDT (Gradient Boosting Decision Tree)

LGBM 是 GBDT 一个较好的工程实现, 是 xgboost 的升级版.

GBDT 中的 Boosting 是一种集成学习方法, 将单个弱分类器串联以形成强分类器.

故先介绍单个 CART 的训练流程.

• CART 构建

决策树的训练与 nn 完全不同, 不需多轮迭代更新权重, 而是用带 label 数据逐层生成一棵二叉树 (CART)。核心是最优 feature 的选择和最优分割值的确定。

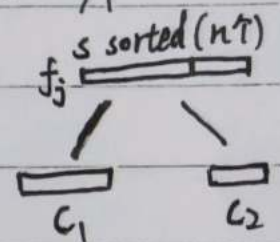
记样本 $S = [s_0, \dots, s_n]$, 特征 $s_i.f = [f_0, \dots, f_m]$, 标签 $s_i.l = 0/1$

从根结点开始生成, 遍历所有特征, 按 f_j 对 S 排序, 寻找最优点.

最优点应为均方误差 MSE 最小值点, 对应 f 为最优特征

$$MSE = \sum_{i=0}^n \frac{1}{2} (s_i.l - \bar{l}_1)^2 + \sum_{i=0}^n \frac{1}{2} (s_i.l - \bar{l}_2)^2$$

其描述了各分割点对样本的分类能力.



逐层重复即可

重复执行至生成结束, 剪枝方法和退出条件不在此展开.

最终得到一棵记录了每个节点的特征和分割值的二叉树. 叶子结点的值为其剩下所有 $s_i.l$ 的均值. 送入新 S 即可根据此树产生决策结果.

• Gradient Boosting DT

GBDT 中 B 指多个 CART 串联, 而 G 则表示新 CART 的生成基于梯度下降.

若以 MSE 为代价函数, 将串联起的 DT 序列作为模型, 则可推导出结论如下:

以 1 为学习率执行一步 GD, 等价于在背面串一个以前面模型计算的残差 (residual) 为拟合目标的新 CART. 即 $new_l = l - model(s.f)$ 具体推导不展开于此

可见每迭代一次 (GD 一步), 就生成一棵 CART, 但此单棵树只能输出前面的 res.

最终模型的输出, 是其中所有 CART 输出的累加之和.

- [🔗 决策树系列（三）： CART\(分类回归树\)-详细原理解析](#)
- [🔗 GBDT\(梯度提升决策树\)——来由、原理和python实现](#)
- [🔗 机器学习算法中 GBDT 和 XGBOOST 的区别有哪些？ - 知乎](#)
- [🔗 【白话机器学习】 算法理论+实战之LightGBM算法](#)

Nudge Model 特征设计

Date

用于判断自车一定范围内的动态障碍物是否认为其需要被 nudge.

目前有13种 feature, 顺序无关. [,] 内为训练时输入的数据范围

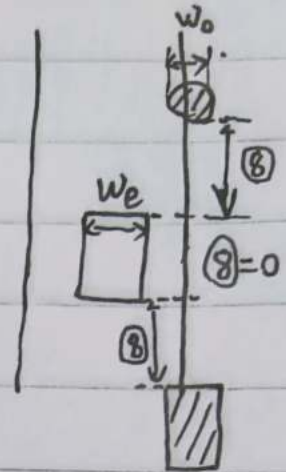
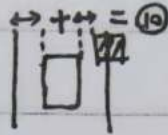
• 距离相关特征:

5. 横向距离 $[-2.3, 7.3]$ $|\Delta center_l| - \frac{1}{2}w_e - \frac{1}{2}w_o$

8. 纵向距离 $[-17.6, 77.2]$

10. 横向空间余量 $[-4.8, 5.5]$

2. 侵入车道宽度 $[-2.7, 5.8]$



• 速度相关特征

1. 纵向相对速度 \times 是否为右侧障碍. $[-12, 33]$

9. 纵向相对速度 \times 是否为同/前侧障碍 $[-15.5, 33.3]$

6. 障碍物横向速度 $[0, 17.4]$ $|ld_obs|$

11. 障碍物速度 $[0, 13, 22]$

• 障碍物本身特征

3. obs在SL系下的速度 heading. $[0, \pi]$

4. obs类型是否为 VEHICLE. $[0, 1]$

13. $4. \times 3.$ $[0, \pi]$

7. obs 纵向长度. $[0, 11, 17.8]$

• 其他

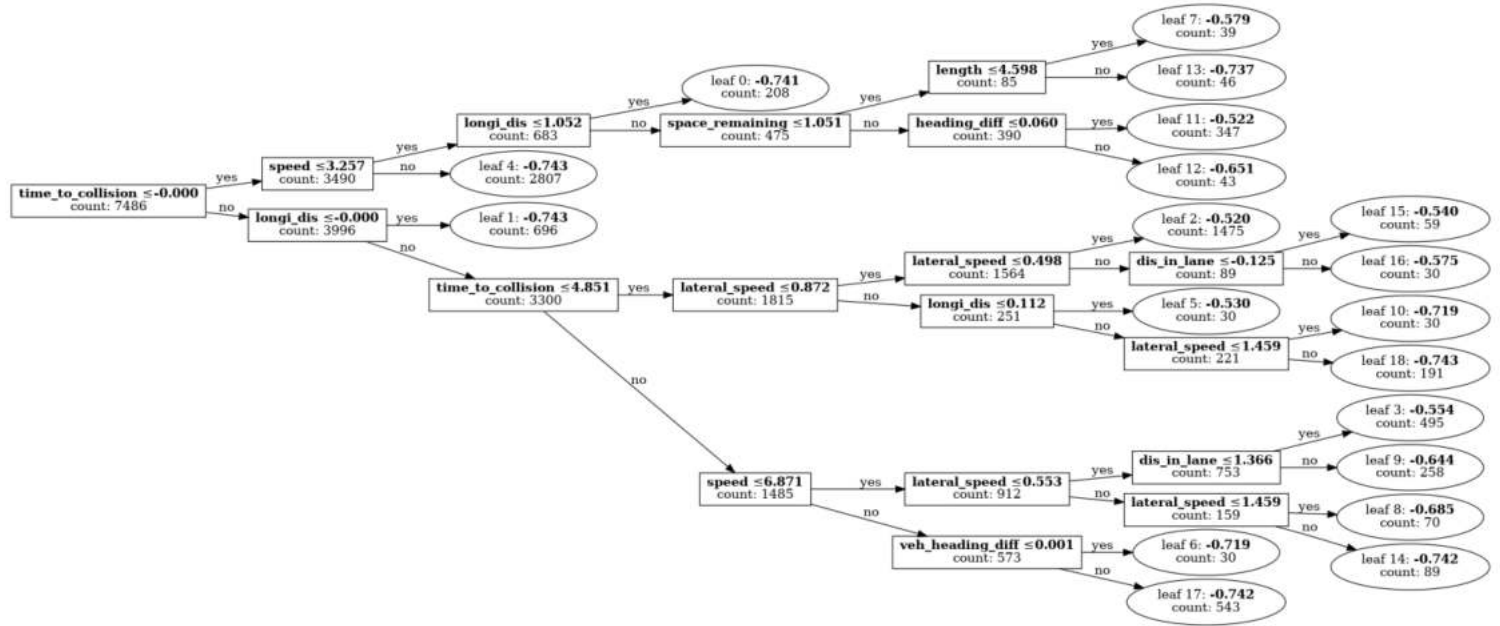
12. time to collision. $[-16000, 24000]$

根据GBDT的生成原理, 其前几棵树的特征和分割点选取应该具有可解释性的.

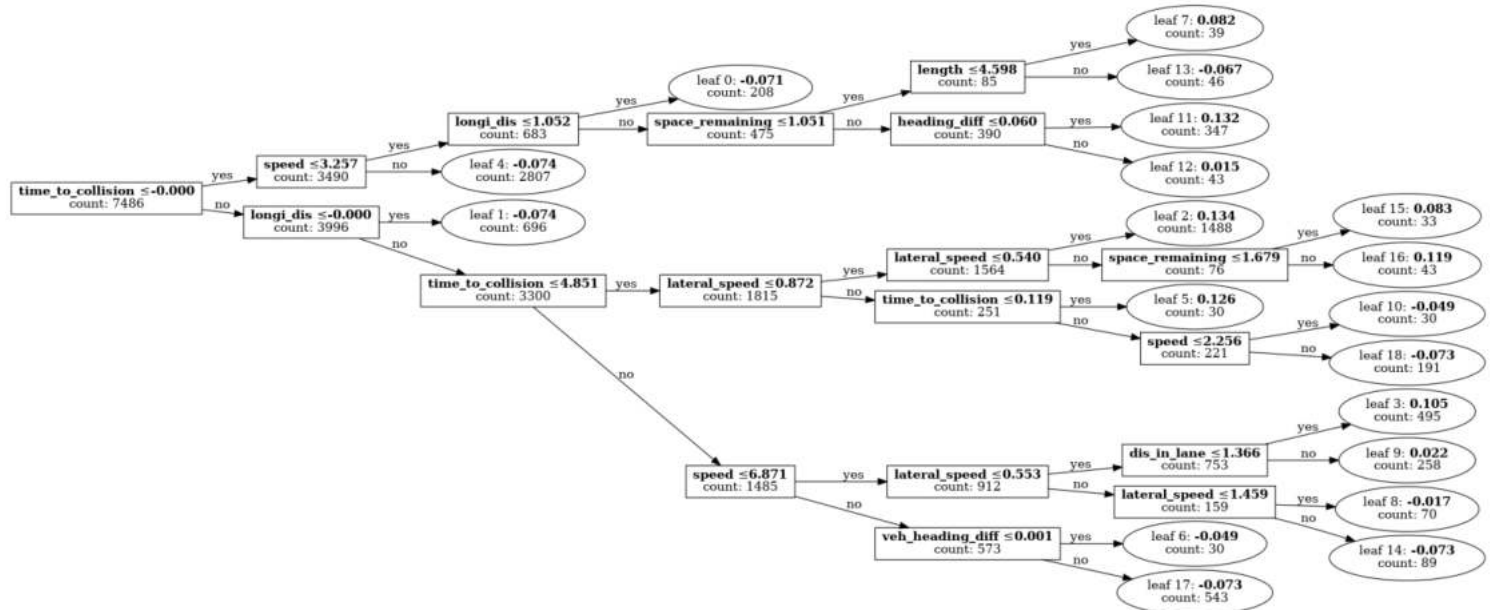
使用7000多条数据训练GBDT模型, 迭代100次生成100个子树串联相加, 取其生成的前两个树可视化如下:

LGBM中最优特征的选取不是按行进行 (Level-wise), 而是各节点独立进行 (Leaf-wise)

整个GBDT输出是所有子树输出的和，但从单个子树的叶节点值也可看出分类的倾向，可以一定程度上对模型决策的过程进行解释



Tree0



Tree1

附：基于规则的Nudge决策

</>

C++

```

1 LKPathBoundaryDecider::RuleNudgeDecision() {
2     // 静态障碍物(speed < 0.5), 则nudge
3     // 动态障碍物
4     // 若VRU 横向速度 < 0.35 && 纵向速度 < 2, 则nudge
5     // 若车辆 横向速度 < 0.2 && 纵向速度 < 2, 则nudge
6     // 左右转参数并不相同
7 }

```

LK Path 边界生成

进行Nudge决策，给需要nudge的障碍物加buffer，并对其过多（影响求解或舒适性）部分进行限制。

LK Path Boundary 生成

Date

1. 沿参考线采样, 作为边界基准.
2. 用 lane/road 宽度加 Buffer 初始化 surface
3. 选取需要 Nudge 的障碍物.

按左右 & 动/静 存入 4 个 vector.

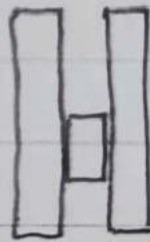
不需 nudge 的则加入 uninterested 集合.

3.1. 将 road 边界处理为静态障碍物.

3.2. 对 ref-line info 提供的障碍, 按一定规则筛去一部分.

3.3. 对剩下部分中障碍, 判断是否可 nudge (空间是否足够), 若可且为静态障碍物, 则 nudge 障碍物列表; 对动态障碍物则使用模型/规则判断是否加入. 若加入动态 obs, 则再加入其一段时间后的预测. (15)

4. CLN 处理, 具体细节此处省略, 最后使用得到的 extend 更新 surface.
(道内 nudge 时不开启)



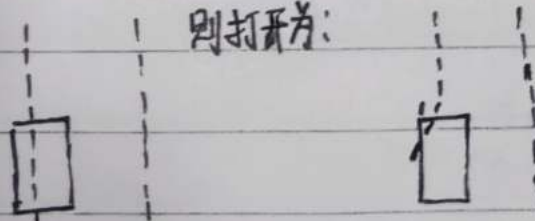
5.1 检查类似左图 (前后更长) 范围内的动态障碍物; 5.2 类似但包含静态

5. 边界生成前的一些处理.

5.1. Nudge 安全检查. 检查自车一定 SL 范围内所有感兴趣障碍物的 ttl/thw.

5.2. 根据一定范围内障碍物的数量和速度判断是否为交通堵塞场景.

5.3. Process Start Point. 若起始点处自车越过了 surface, 则额外打开空间, 并以起始时的 dl 回正. 若:




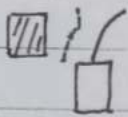
5.4 用 surface 初始化最终边界 L_{max}/L_{min} .

5.5 计算转弯半径. 计算各采样点处以 comfort-deacc 减速得到的速度, 分别使用不同的 kappa-coeff 计算各点处速度对应的最小/最舒适转弯半径.

6. 最终边界生成

主要分为两部分：①. 由 nudge 障碍物提供的较远边界。

②. 由各种因素计算的边界限制，对①中边界进行约束。

例如：①.  ②.  白车运动学校限造成的边界限制将变成了！

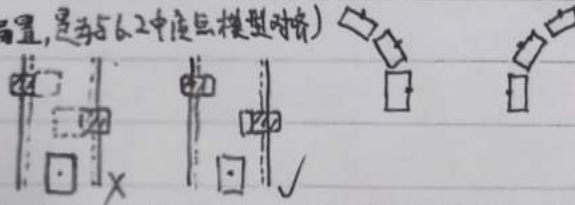
边界限制部分：

6.1 运动学限制

在 xy 坐标系下根据起始 heading 和最大前轮转角等计算车辆运动学校限

以后轮中心为离散递推基准，输出转向方向另一侧车辆中间形成的边界。

(最后输出时又加了半车宽偏置，是与 6.2 中点模型对齐)



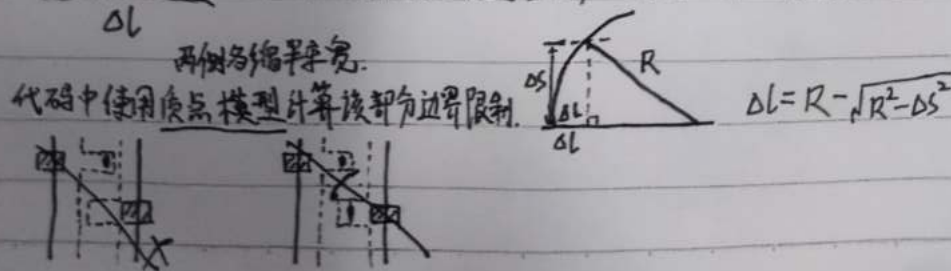
6.2 障碍物间距限制

当左右两侧障碍物间距较小时，无法使用 nudge 默认的较远边界通过；需计算在间距内能产生的最大横向位移，对较远边界进行收缩。

具体实现：将左侧每个 nudge 障碍物与右侧每个两两比较，对任意一组处理如下：

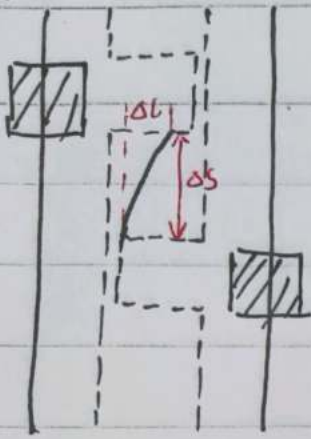
取较远 obs 纵向起点处的最小转弯半径 (5.5)，比较其与两 obs 的纵向间距。

若纵向间距更大则无事故发生 (程序里的表达比较不好理解)，若纵向间距较小则计算间距内最大横向位移： ΔL 为上文中的最小转弯半径， ΔS 为两 obs 纵向间距。



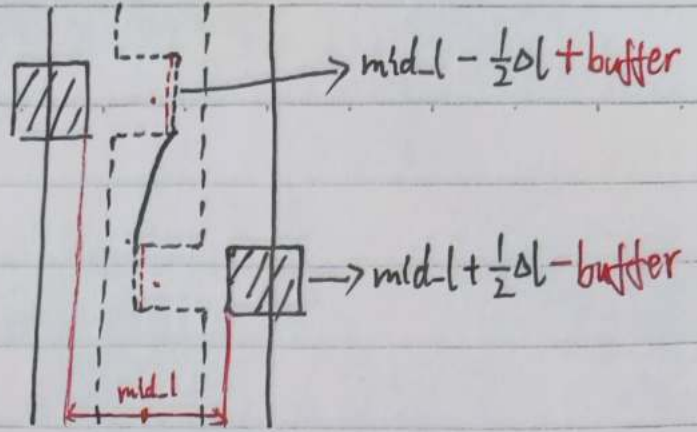
续6.2.

Date



由障碍物提供的原始边界.

若不加处理会导致求解失败.

由最大横向位移 ΔL 提供的边界限制两障碍中心向两侧扩张 $\frac{1}{2}\Delta L$, 再加通用横向buffer

6.3. 舒适性限制.

通过5.5中得到的舒适转弯半径, 计算从自车到各S以 gentle-radius 能产生的最大横向位移

此部分的几何模型存在一些假设和近似:

在S处以 adc-frenet-heading 和 S处对应的 gentle-radius 做圆, 来计算 ΔS 内的 ΔL 记 $R = S$ 处的 gentle-radius $\theta = \text{adc-frenet-heading}$, 以向右 nudge 情况为例:

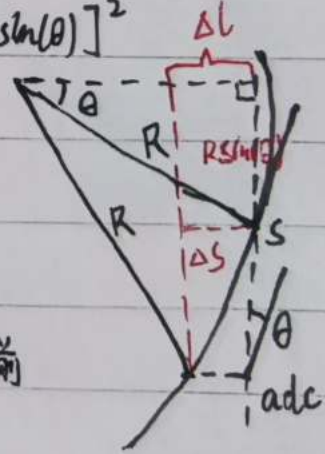
$$\text{comfort-constraint} = -R \cos(\theta) + \sqrt{R^2 - [\Delta S + R \sin(\theta)]^2}$$

此处得到的边界限制和6.2一样为质点模型

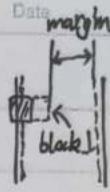
最后恢复为非质点模型的边界限制为:

$$\text{comfort-limit} = \text{adc-l} + \text{comfort-constraint} + \text{half-pass-width}.$$

(此部分逻辑实现在 CalcFinalLeft/Right Bound() 内, 但其与前两小节一样都是在算边界 limit, 故单独列出来)



边界生成部分.



6.4 障碍物边界生成.

处理所有 nudge 障碍物, 根据规则为其设置不同 buffer, 再根据加 buffer 后边界到对侧 surface 间余下距离调整边界: 若 margin 大于通行车宽则边界不动, 若小于则通过一种插值方式得到从 black_l 到同侧 surface 间的边界。最后再由规则添加 Extra Buffer, 大多数情况会得到一个推得较远的 nudge 边界。

然后再用前面得到的各种 limit 对该边界进行限制, 防止推得太远导致求解困难或不够舒适。(此处添加的半车宽偏置是为了将算 limit 时的质点模型恢复)

至此, 已得到由 obs 边界、surface、以及边界 limit 组成的非质点模型边界。

7. 后续处理.

7.1 对纵向与自车有重合的 nudge 障碍, 至少打开半车宽 + 横向标准 buffer 的空间。

7.2 对生成好的边界向内收缩左右各一个半通行车宽, 转化为质点模型以供优化算法使用。并为上下界收缩后距离过近或交叉等不合法情况留出一个最小空间。

7.3 添加 QP 中用到的相关信息。在一定 s 范围内添加自车外接矩形约束信息, 判断 nudge 障碍中是否存在大车, 若存在则添加 offset, 使 path 贴近另一侧。

7.4 将生成好的边界以及由 lane 边界构成的 fallback 更新到上下文。

Q:

6.2、6.3处均取较远R, 理论上会得到更大的 δI , 造成更宽松的限制, 是否应该取起点或中间点?

κ_{coeff} 是否具有物理含义? 类似最大向心加速度?

`EvalNudgeSafety()` { // same direction } 处为什么不考虑自车速度?

LK Path QP建模

LK Fixed Piecewise Jerk Path Problem QP

Date

决策变量: $[l_0 \dots l_{n-1} \quad \dot{l}_0 \dots \dot{l}_{n-1} \quad \ddot{l}_0 \dots \ddot{l}_{n-1} \quad \mu_0 \dots \mu_{m-1}]$ n 为沿参考线采样数目, m 为 ego box extra constraint 采样数目.总维度为 $3n+m$.

目标函数:

$$J = \sum_{i=0}^{n-1} \left[w_l l_i^2 + w_{dl} \dot{l}_i^2 + w_{ddl} \ddot{l}_i^2 + w_{dddl} \dddot{l}_i^2 + w_{ref} (l_i - l_{i,ref})^2 \right] + \sum_{j=0}^{m-1} w_{ego} \mu_j^2$$

\downarrow 接近参考线中心 \downarrow dl 尽量小 \downarrow ddl 小 \downarrow $dddl$ 小 \downarrow 接近 $l_{i,ref}$ 提供的偏置 offset 中心 \downarrow 松驰变量 μ 尽量小

约束:

- l_i 的上下界约束
- \dot{l}_0 和 \ddot{l}_0 的起始约束
- \dot{l}_{n-1} 终点约束
- 由 Piecewise Jerk 约定得到的物理性质约束.
- 自车外接矩形 ego box 额外约束.

最终形式:

$$\min: X^T P X + Q X$$

$$\text{s.t.}: L < A X < U$$

接下来对其中稍难理解部分以及矩阵构建细节进行整理

目标函数 $J = X^T P X + Q X$:

P 为二次项系数矩阵, Q 为一次项系数向量.

$$J = \sum w_l \dot{l}_i^2 + w_{dl} \dot{l}_i^2 + w_{ddl} \ddot{l}_i^2 + w_{dddl} \ddot{l}_i^2 + w_{ref} (l_i - l_{i-ref})^2 + \sum w_{ego} \mu_j^2$$

由于 $\dot{l}_i = \frac{l_{i+1} - l_i}{\Delta s} = \frac{1}{\Delta s^2} (l_{i+1}^2 - l_i^2 - 2l_i l_{i+1})$

$$(l_i - l_{i-ref})^2 = l_i^2 - 2l_{i-ref} l_i + l_{i-ref}^2$$

二次项 一次项 常数项

有 P, Q 如下:

$$P = \begin{bmatrix} \begin{bmatrix} w_l + w_{ref} \\ \vdots \\ w_l + w_{ref} \end{bmatrix}_{n \times n} & & & & \\ & \begin{bmatrix} w_{dl} \\ \vdots \\ w_{dl} \end{bmatrix}_{n \times n} & & & \\ & & \begin{bmatrix} w_{ddl}} & & \\ & 2w_{ddl} & & \\ & & \ddots & \\ & & & -2w_{ddl} \\ & & & & 2w_{ddl} - 2w_{ddl} \\ & & & & & w_{ddl} \end{bmatrix}_{n \times n} & & \\ & & & & \begin{bmatrix} w_{ego} \\ \vdots \\ w_{ego} \end{bmatrix}_{m \times m} \end{bmatrix}_{3n+m \times 3n+m}$$

$$Q = \begin{bmatrix} -2w_{ego} l_{i-ref} & \cdots & -2w_{ego} l_{i-ref} & 0 & \cdots & 0 \end{bmatrix}_{1 \times 3n+m}$$

l_{i-ref} 表示偏离参考线中心的偏置, 用于在遇大车等情况下偏移中心行驶.

Piecewise Jerk 连续性约束:

优化问题的决策变量为 $[l_0 \dots l_{n-1} \dot{l}_0 \dots \dot{l}_{n-1} \ddot{l}_0 \dots \ddot{l}_{n-1}]$ 即每个采样点的 P, v, a .
但还需对其添加某种约束来确保它们之间具有正确的 $P/v/a$ 间应有的物理关系.

Piecewise Jerk 即为一种方法, 其规定两离散点间认为是加速度一致, 即:

$$\ddot{l}_{i \rightarrow i+1} = \frac{\dot{l}_{i+1} - \dot{l}_i}{\Delta s} \quad (1).$$

由此式结合 $P/v/a$ 间的递推关系便可对决策变量间的物理连续性进行约束, 来保证它们确实表示了各离散点的 $P/v/a$.

$$\left. \begin{array}{l} (1) \\ v, a \text{ 约束} \end{array} \right\} \Rightarrow \dot{l}_{i+1} - \dot{l}_i - \frac{\Delta s}{2} \ddot{l}_i - \frac{\Delta s}{2} \ddot{l}_{i+1} = 0$$

$$\dot{l}_{i+1} = \dot{l}_i + \ddot{l}_i \Delta s + \frac{1}{2} \ddot{l}_{i \rightarrow i+1} \Delta s^2$$

$$\left. \begin{array}{l} (1) \\ P, v, a \text{ 约束} \end{array} \right\} \Rightarrow l_{i+1} - l_i - \Delta s \dot{l}_i - \frac{\Delta s^2}{3} \ddot{l}_i - \frac{\Delta s^2}{6} \ddot{l}_{i+1} = 0$$

$$l_{i+1} = l_i + \dot{l}_i \Delta s + \frac{1}{2} \ddot{l}_i \Delta s^2 + \frac{1}{6} \ddot{l}_{i \rightarrow i+1} \Delta s^3$$

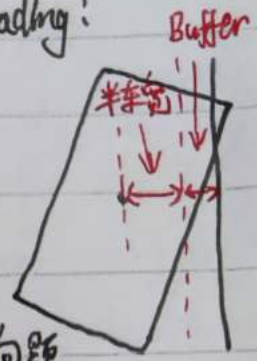
ego box 约束:

Piecewise Jerk SL 路径生成算法中, 将自车处理为质点, 将安全边界膨胀半个车宽 + 一定长度的 Buffer 作为约束。但这种方式没有考虑自车的 heading:

若 Buffer 较小则 heading 大时 可能越过边界

若 Buffer 较大则过严的约束会影响向正常能通过时的求解。

小 heading



故额外加入描述自车外接矩形的 ego-box 约束, 处理 heading 问题。

因 SL 路径生成中不直接描述 heading, 而是用 $\dot{l} = \frac{dl}{ds}$ 间接表示 (纵向运动带来的横向运动是由 heading 引起的)

故 ego box 约束表示为:

$$lb < l_i + \dot{l}_i \times half_veh_len + H_i < ub$$

即以 \dot{l} 向前半车长后的 l 进行约束, 但这种约束比实际更严格, 故加入松弛变量 H , 转化为软约束, 在目标函数中最小化 $\sum H^2$ 。

约束 $L < Ax < U$:

A 为独立约束数目行 \times $3n+m$ 列的约束矩阵, 每行表示一个独立约束.

L	0	n	A	$2n$	$3n$	$3n+m-1$	U
L_{min_0}	1	(L 上下界约束)					L_{max_0}
\vdots							\vdots
$L_{min_{n-1}}$		1					$L_{max_{n-1}}$
$init_l$			1 (起始 \dot{l})				$init_l$
$init_dl$				1 (起始 \ddot{l})			$init_dl$
0				1 (终止 \dot{l})			0
0		-1 1		$-\frac{\Delta s}{2} - \frac{\Delta s}{2}$		(Piecewise Jerk 物理约束1)	0
\vdots							\vdots
0			-1 1		$-\frac{\Delta s}{2} - \frac{\Delta s}{2}$		0
0	-1 1	$-\Delta s$		$-\frac{\Delta s^2}{3} - \frac{\Delta s^2}{6}$		(Piecewise Jerk 物理约束2)	0
\vdots							\vdots
0		-1 1	$-\Delta s$		$-\frac{\Delta s^2}{3} - \frac{\Delta s^2}{6}$		0
L_{lb_0}	lr rr	$left/right$ ratio	$lr \times hl$ $rr \times hl$				L_{ub_0}
\vdots			$half_veh_len$				\vdots
$L_{lb_{n-1}}$	lr rr		$lr \times hl$ $rr \times hl$				$L_{ub_{n-1}}$

最后部分为 ego-box 约束, 共 m 行, $0-n$ 中对应坐标由 s 对应关系得到,

左右两处表示线性插值各自对应权重, $left\ ratio + right\ ratio = 1$

L_{lb} 和 L_{ub} 与 L_{min}/L_{max} 基有一致, 但保留了一个 Buffer 的最窄空间.

Q:

add_ego_width_constraint, 为什么用 center_s, l' 用 rear_s?

// 3 rows in first n cols? 源自基类没改, 此处其实只需一行

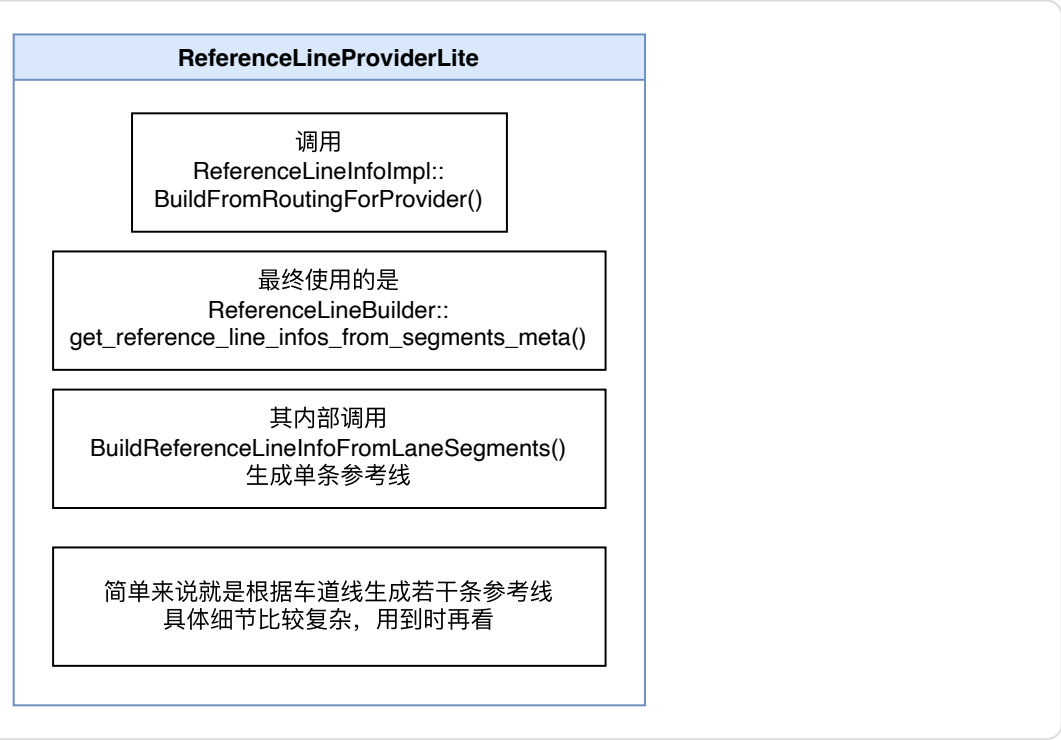
// 1 Compute kernel matrix, use identity matrix? 确实少减了个1, 影响不大?

参考线相关

时间有限，这部分只整理了一下流程，原理细节后续用到时再补充

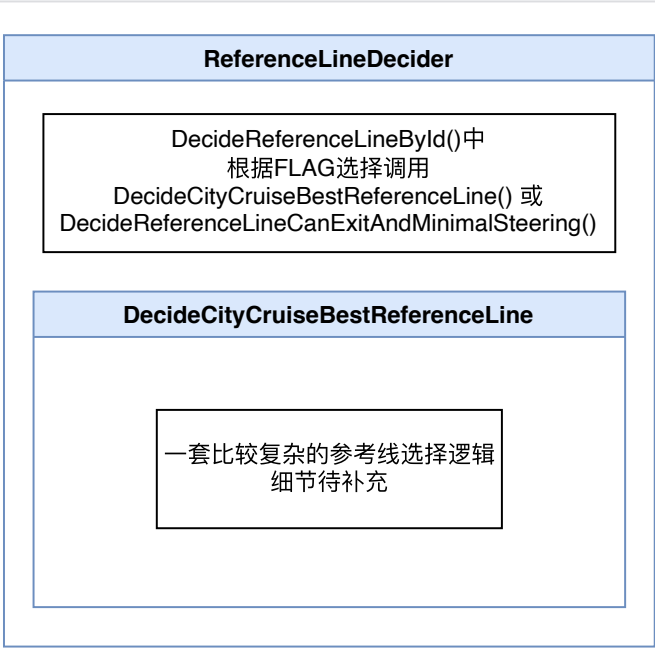
生成

根据WorldView生成多条参考线，更新到ReferenceLineBundle中



选取

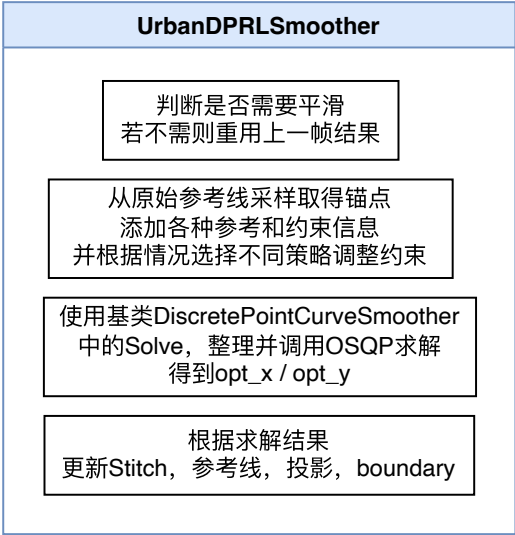
从WorldView的ReferenceLineBundle中的多条参考线中，选出要执行的目标参考线
ego_reference_line_info_ptr_



平滑

对选出的自行车参考线进行平滑

根据FLAG选择UrbanDPRLSmoother或ReferenceLineProcessor



优化问题具体原理和QP构建细节待补充，应该和apollo类似。决策变量为离散点xy坐标

TrajectoryStitcher

为减少轨迹规划帧间跳变的情况，当上帧轨迹存在且自车偏离轨迹不算太远时，将上帧轨迹前推一段的点作为下次规划的起点

前推的这段上帧轨迹即stitching_trajectory

细节待补充