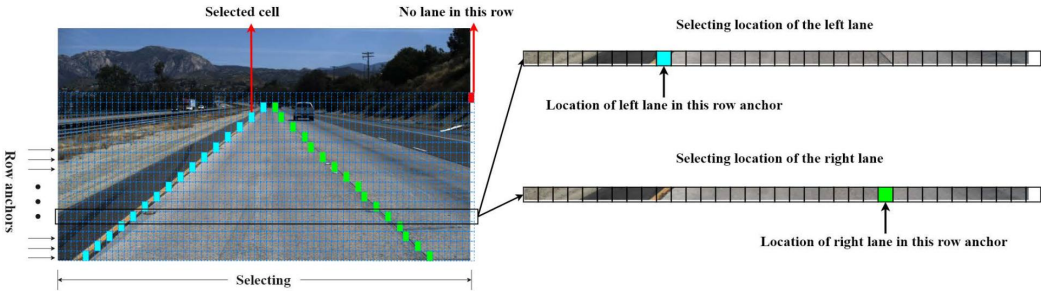
车道线检测交接文档

1.车道线检测的方案分类：基于anchor，基于关键点，基于像素级分类

2.基于anchor的方案：Ultra-Fast-Lane-Detection
github主页：https://github.com/cfzd/Ultra-Fast-Lane-Detection



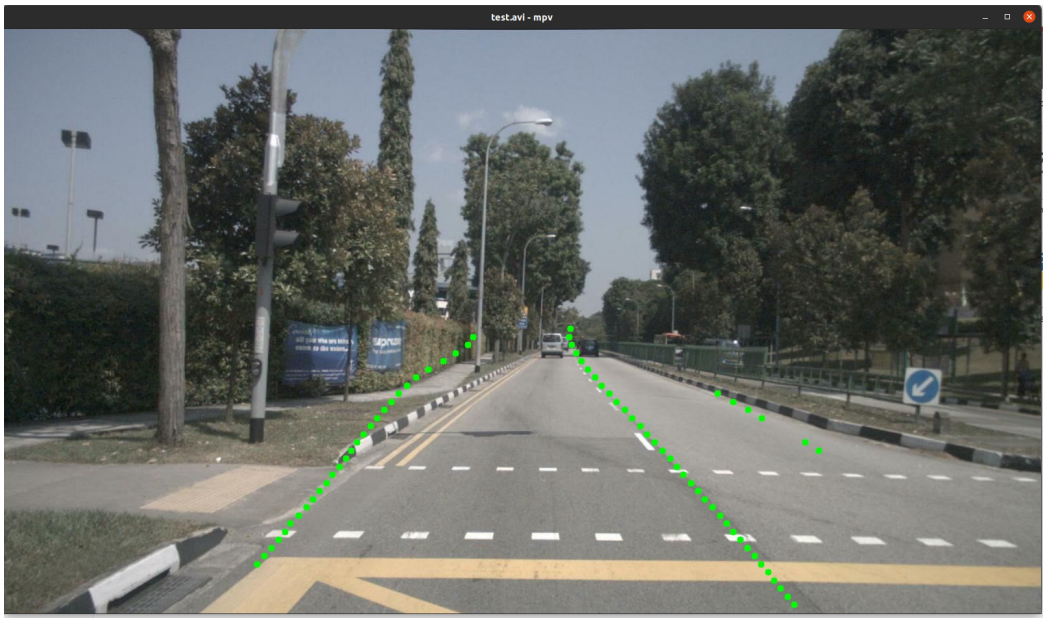
主要思路：将画面分割为较大的像素块，再用固定形状和高度的anchor去匹配，Loss与位置和形状相关。

优点：避免逐个像素的分类，运行速度快

缺点：

1）端到端的网络学习了相机的内外参，原论文的预训练模型是基于Tusimple和CULane数据集，直接应用于Nuscenes时由于内外参和预训练模型学习到的不一致会出现较大的误差，最典型的表现为路经末端上飘。

e.g.



2）anchor的设计限制了路口，人行横道等复杂路况无法识别

*2022.8作者团队更新了V2版本，github主页：https://github.com/cfzd/Ultra-Fast-Lane-Detection-v2

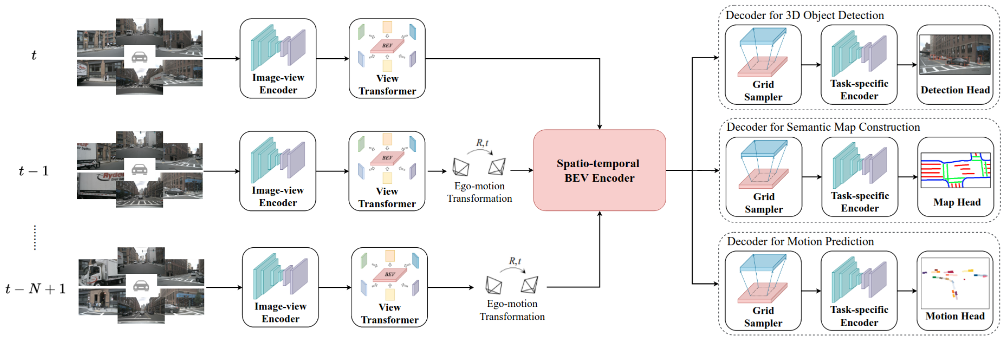
主要更新的方面：增加了不同形状的anchor使得适应性提升，但是具体效果没有试

3.基于分割的方案：BEVerse
github主页：https://github.com/zhangyp15/BEVerse

1）基于mmdet3D的框架，数据集由pipline读入，在每层读入时增加标注的字典键值，模型由config文件中按照backbone-neck-head的方式配置。

*mmdet3D主页：https://mmdetection3d.readthedocs.io/zh_CN/latest/getting_started.html

2）主要思路



BEVerse有small和tiny两个版本，small版本在单gpu上消耗显存约为32G，主页上提供两个版本在Nuscenes上的预训练模型。

*尝试训练了tiny版本，但是指标不如主页上所示，占用显存约12G。

模型利用前两帧和当前帧共三帧的环绕图像完成三个任务：

①3D物体检测

②BEV地图的绘制

③后四帧的运动状态预测

对于任务②，在分割预测时模型主要分为三类：道路线（红色），人行横道线（绿色），路边（蓝色），其他（无）。

3）运行模型

①依赖和数据准备见项目主页

②在BEVerse目录下运行

```
python tools/test.py projects/configs/beverse_small.py
 ckpts/beverse_small.pth --eval=segm --mtl --show
```

③测试Nuscenes的valid数据集结果：

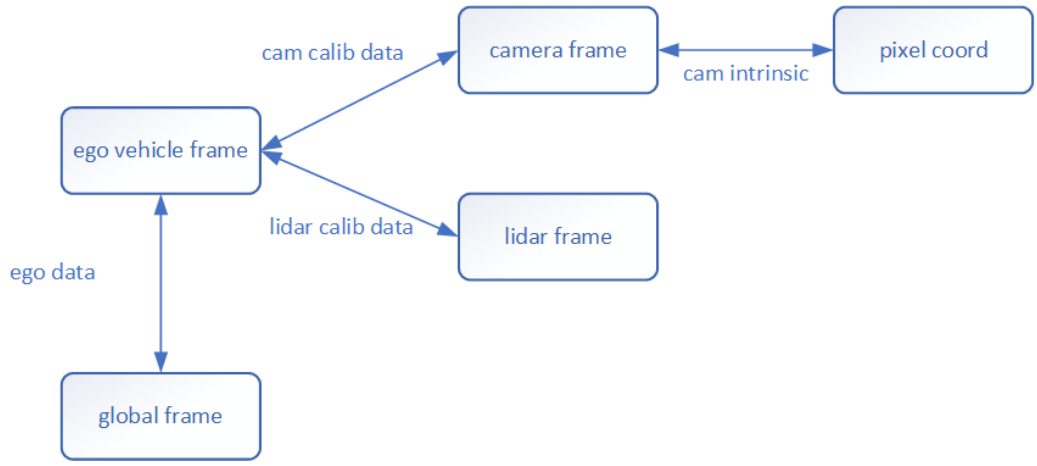
```
[Validation 5119 / 6019]: motion metrics:
grid = 30x30: iou = 61.384, pq = 54.335, sq = 74.227, rq = 73.201, denominator = 8205950.000,
grid = 100x100: iou = 40.847, pq = 36.135, sq = 70.765, rq = 51.063, denominator = 24284100.000,

Formating bboxes of pts_bbox
Start to convert detection format...
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] 6019/6019, 97.6 task/s, elapsed: 62s, ETA:     0s
Results writes to /tmp/tmp19e_0rkt/results/pts_bbox/results_nusc.json
Evaluating bboxes of pts_bbox
mAP: 0.3512
mATE: 0.6244
mASE: 0.2694
mAOE: 0.3999
mAVE: 0.3292
mAAE: 0.1827
NDS: 0.4950
Eval time: 116.8s

Per-class results:
Object Class    AP      ATE     ASE     AOE     AVE     AAE
car     0.562   0.456   0.154   0.073   0.311   0.188
truck   0.264   0.618   0.209   0.085   0.289   0.200
bus     0.365   0.673   0.193   0.068   0.780   0.337
trailer 0.136   0.994   0.240   0.487   0.204   0.038
construction_vehicle    0.072   0.897   0.485   1.196   0.106   0.327
pedestrian      0.418   0.629   0.296   0.637   0.409   0.172
motorcycle      0.344   0.605   0.259   0.375   0.414   0.197
bicycle 0.301   0.487   0.253   0.541   0.121   0.002
traffic_cone    0.533   0.417   0.330   nan     nan     nan
barrier 0.515   0.467   0.276   0.137   nan     nan
{'pts_bbox_NuScenes/car_AP_dist_0.5': 0.2315, 'pts_bbox_NuScenes/car_AP_dist_1.0': 0.5004, 'pts_bbo
```

3）后续开发

Nuscenes视图转换：



参考：https://blog.csdn.net/qq_16137569/article/details/121066977

①BEV转前视图

提取BEVerse生成的BEV图红色的车道线部分进行坐标转换，公式为：

front_points = ego2img @ [x y 0 1].T

其中需要考虑x、y做的归一化，即：

x = 0.15 * (–i + w/2)

y = 0.15 * (–j + h/2)

其中i和j代表BEV中的车道线点，w、h为BEV图的尺寸，0.15为每个点的尺寸

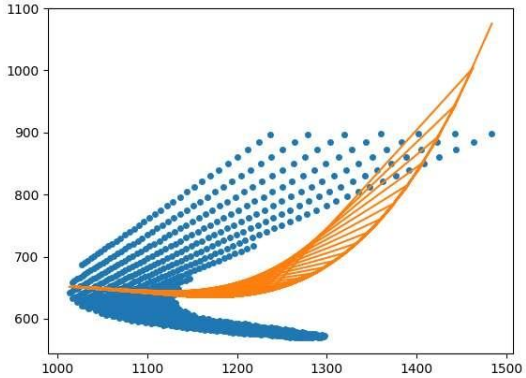
②前视图车道线聚类和拟合

聚类方法：DBSCAN

*需要排除标签为–1的杂点

拟合方法：polyfit拟合横轴x的三次曲线

*缺点：在某些情况下polyfit会拟合失败，例如车道线轨迹中包含纵轴的多次项



③卡尔曼滤波跟踪

目的：稳定前步骤的投影结果，匹配方法为最大权匹配，由于线与线之间无法计算IoU，所以匹配状态和观测时使用公式：

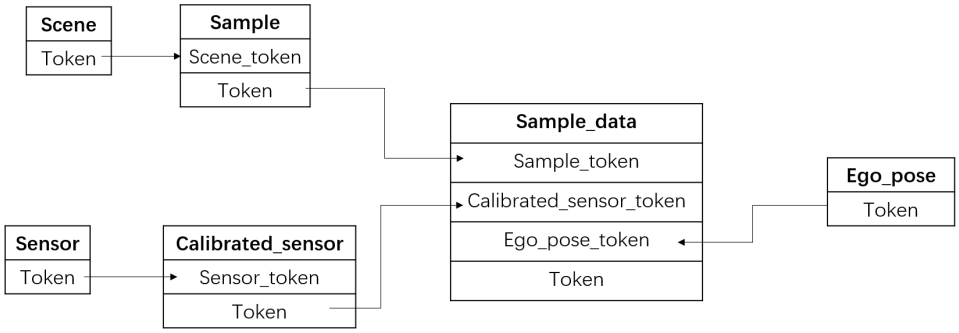
weight = 100 – 0.0001*|s_c3 – m_c3| – 0.1*|s_c2 – m_c2|

其中三个常值均为经验值，s表示state，m表示measure，c3表示车道线的截矩，c2表示车道线的斜率，具体可以参考：https://jishuin.proginn.com/p/763bfbd5b18b

*另，卡尔曼滤波中为平衡输入和状态还可以继续调参达到更好的效果

④前视图车道线转全局视角画一个scene的地图

首先需要获得在每一个keyframe（即test.py所用的部分数据）车辆的位置ego_pose，根据Nuscenes的官方工具教程Nuscenes-devkit(主页https://github.com/nutonomy/nuscenes-devkit)可知，每一个json文件中的标注数据都可以由唯一的标识token确定，而不同的json文件之间token也可以作为同一时刻不同属性的链接方式，具体如下图所示：



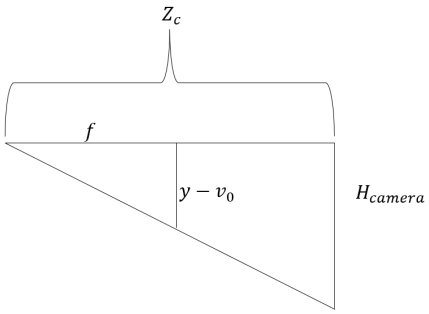
其中用箭头指向的部分代表同一个值在不同json文件里的名称。

在test.py的img_meta中可以获得sample data的token，根据上图重新解析Nuscenes数据集即可获得ego_pose，内外参等数据。

*参考：https://github.com/nutonomy/nuscenes-devkit/blob/master/python-sdk/tutorials/nuscenes_tutorial.ipynb

二次投影：前视图->BEV

根据相机坐标系和图像坐标系的转换关系建立相似三角形。



其中y为前视图上的某一车道线的点，v0、f为已知内参，Hcamera为已知外参，故可以求得该点的深度Zc。

*注意：由于采集数据时标定的误差，转换后的深度与原深度会存在较大的误差，在Beverse中所生成的BEV图在车辆行使方向最远探测可达30m，但二次投影后只能达到24m。

将车道线由图像坐标系转到ego坐标系：

ego_point = img2ego @ [x, y, 1].T * Zc

将车道线由ego坐标系转到全局坐标系：

global_point = ego_pose_roration @ ego_point + ego_pose_translation

4）代码说明

二次开发代码在/BEVerse/Kalman/z_touyint.py 和z_pages.py中。

①执行test.py获得train和val数据集中的车辆前视图和前视图的det_gt_CAM_FRONT.png，gt.png，装有sample data token的tokens.txt。

②在/BEVerse/Kalman/下执行get_tokens.py，获得每个keyframes对应的location，scene token，内外参，ego_pose等

③运行z_pages.py或z_touyint.py