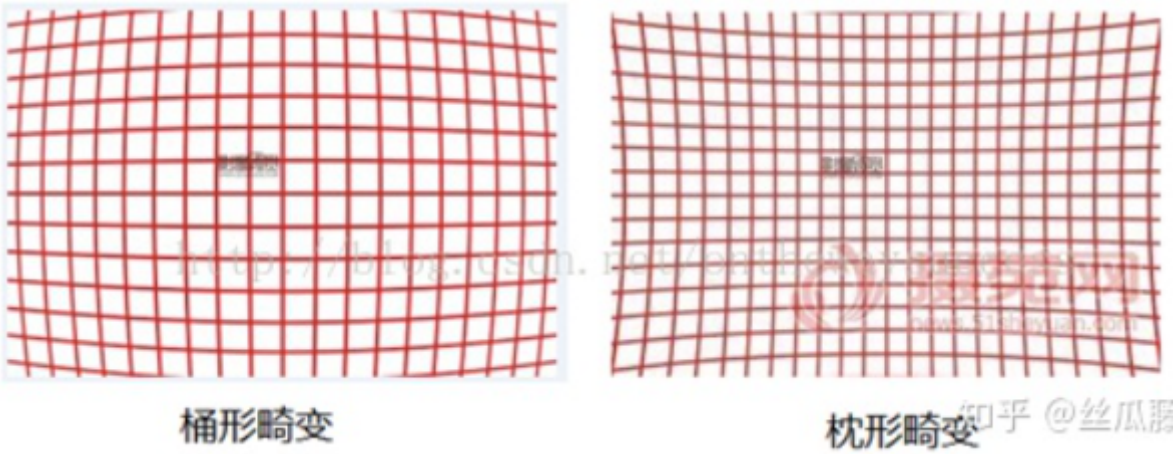
车道线识别

一、畸变校正

1.1 畸变简介

图像会出现“拉伸”或者“扭曲”的直观感受，分为径向畸变和切向畸变。

径向畸变：光学镜头在生产制造的过程中，很难保证厚度的均匀，离透镜中心越远的地方光线弯曲越大，从而产生径向畸变



切向畸变：由于镜头与图像传感器不完全平行造成的

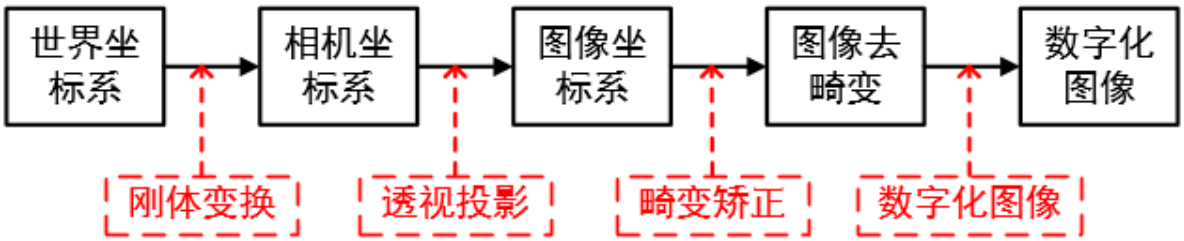
1.2 摄像头参数

- 相机矩阵：焦距(Fx,Fy)、光学中心(Cx,Cy)
- 畸变系数：D=（K1,K2,P1,P2,K3）
- 相机内参：相机矩阵和畸变系数
- 相机外参：通过旋转和平移变换将3D的坐标转换为相机2维的坐标，旋转和平移为外参

1.3 摄像头标定

相机的标定过程实际上就是在4个坐标系转化的过程中求出相机的内参和外参的过程。

像素坐标系(p)、图像坐标系(i)、相机坐标系(c)、世界坐标系(w)，成像过程基本分为：物理坐标变换、投影变换、畸变矫正、像素变换



（1）世界坐标系 ⇒ 相机坐标系：**求解摄像头外参（旋转和平移矩阵）**

（2）相机坐标系 ⇒ 图像坐标系：**求解相机内参（摄像头矩阵和畸变系数）**

（3） 图像坐标系 ⇒ 像素坐标系：求解像素转化矩阵（可简单理解为原点从图片中心到左上角，单位厘米变行列）

二、透视变换

将图像投影到一个新的视平面。距离摄像头越近的点，看起来越大，越远的点看起来越小。对这幅图像透视校正的目的就是要纠正这种形变.透视变换能保持“直线性”，即原图像里面的直线，经透视变换后仍为直线

$$\begin{bmatrix} x' & y' & w' \end{bmatrix} = \begin{bmatrix} u & v & w \end{bmatrix} * \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$Transform = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} T_1 & T_2 \\ T_3 & a_{33} \end{bmatrix}$$

$T_1 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ 表示图像线性变换。

$T_2 = \begin{bmatrix} a_{13} & a_{23} \end{bmatrix}^T$ 用于产生图像透视变换。

$T_3 = \begin{bmatrix} a_{31} & a_{32} \end{bmatrix}$ 表示图像平移。

给定透视变换对应的四对像素点坐标，即可求得透视变换矩阵；反之，给定透视变换矩阵，即可对图像或像素点坐标完成透视变换

三、平均基尔霍夫变换

众所周知, 一条直线在图像二维空间可由两个变量表示. 如:

<1>在笛卡尔坐标系: 可由参数: 斜率和截距(m,b) 表示。

<2>在极坐标系: 可由参数: 极径和极角(r, θ)表示。、

霍夫变换就是通过极坐标系下的点上直线通过的概率。

一般来说我们可以通过设置直线上点的阈值来定义多少条曲线交于一点我们才认为检测到了一条直线。

霍夫线变换是一种用来寻找直线的方法，在使用霍夫线变换之前，首先要对图像进行边缘检测的处理，也即霍夫线变换的直接输入只能是边缘二值图像。

OpenCV支持三种不同的霍夫线变换，它们分别是：

- 标准霍夫变换(Standard Hough Transform，SHT)
- 多尺度霍夫变换（Multi-Scale Hough Transform，MSHT）
- 累计概率霍夫变换(Progressive Probabilistic Hough Transform ，PPHT)。**它在一定的范围内进行霍夫变换，计算单独线段的方向以及范围，从而减少计算量，缩短计算时间。**

```
def averageSlopeIntercept(lines):
    """
    求平均的斜率和截距
    :param lines: 霍夫变换后得到的所有的直线
    :return: 最终的左边和右边车道线的两条直线(直线返回的是斜率和截距)
    """
    left_lines = []  # (slope, intercept)
    left_weights = []  # 权重为线段的长度(length,)
    right_lines = []  # (slope, intercept)
    right_weights = []  # (length,)
    for line in lines:
        for x1, y1, x2, y2 in line:
            # 若出现垂直或者水平的线则忽略
            if x2 == x1 or y1 == y2:
                continue
            slope = (y2 - y1) / (x2 - x1)
            intercept = y1 - slope * x1
            # 图像中y是从图像最底部为0开始的,所以右边的直线斜率大于0；左边的直线的斜率小于0
            # 得到每个直线的长度
            length = sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)
            # 防止左线出现过于平缓的线，即可能识别到蓝线
            if slope < -1 and x1 < 320 and x2 < 320:  # y is reversed in image
                left_lines.append((slope, intercept))
                left_weights.append(length)
            #  去除过于平缓的直线
            elif slope > 0.1 and x1 >= 320 and x2 >= 320:
                right_lines.append((slope, intercept))
                right_weights.append(length)
    # np.dot便是的是矩阵点乘.平均直线为最终的直线结果
    left_lane = np.dot(left_weights, left_lines) / np.sum(left_weights) if
len(left_weights) > 0 else None
    right_lane = np.dot(right_weights, right_lines) / np.sum(right_weights) if
len(right_weights) > 0 else None
    return left_lane, right_lane  # (slope, intercept), (slope, intercept)
```

红绿灯识别

一、OpenCV识别

1.1 提取感兴趣区域

在图像处理过程中，我们可能只对某一个特定的区域感兴趣，所以我们需要进行感兴趣区域的提取。最常用的操作就是：图像的裁剪

```
//左上角的x,y,图像的width,heigth
image_ROI = cv_image(cv::Rect(520,100,120,150));
```

1.2 HSV颜色提取

HSV更加符合人眼的色彩知觉。

H--Hue即色相，用度数来描述，红色对应0度，绿色对应120度，蓝色对应240度

S--Saturation饱和度，色彩的深浅度(0~100%)

V--value色调，纯度，色彩的亮度(0~100%)

```
//转化成hsv
cvtColor(image_ROI, hsv_image, CV_BGR2HSV);
//hsv阈值调整:调整绿灯的hsv阈值，输出是一个二值化图像
inRange(hsv_image, cv::Scalar(69, 51, 147), cv::Scalar(157, 240, 255),
hsv_range_image);
```

1.3 二值化

二值化算法是取一个阈值，将大于阈值的像素点取值1即显示白色，而小于阈值的像素点自然赋值为0即显示为黑色。 CV_ADAPTIVE_THRESH_MEAN_C （取整幅图像的均值像素点）和 CV_ADAPTIVE_THRESH_GAUSSIAN_C， 它是一个从均值或加权均值(区域中（x，y）周围的像素根据高斯函数按照他们离中心点的距离进行加权计算)提取的常数

```
//自适应二值化
threshold(hsv_range_image, binary_image, 127, 255, cv::THRESH_BINARY);
```

1.4 形态学腐蚀

形态学：根据图像形状进行简单操作，对二值化图像/灰度图像进行操作，只需要输入两个操作：一个是原始图像，另一个是称为结构化元素或者核(决定了操作的性质)。包括了开运算、闭运算、梯度等

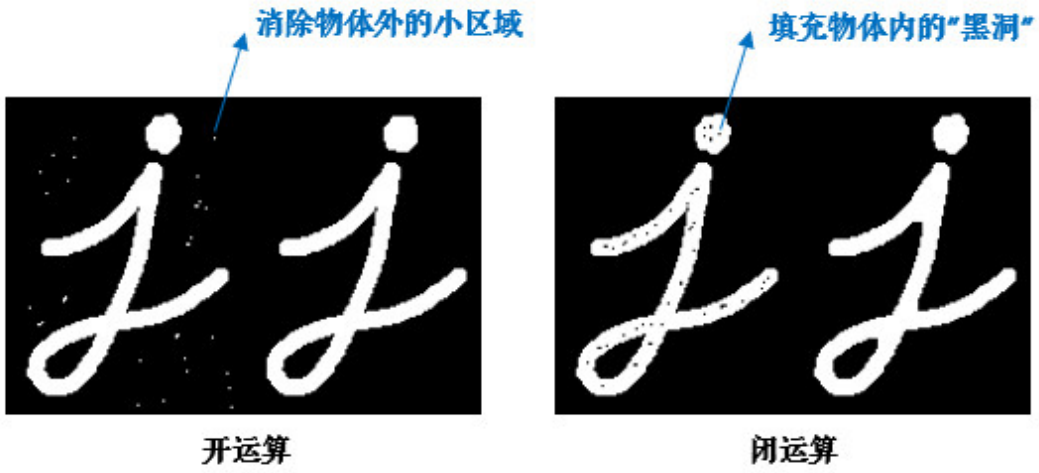
腐蚀：把图片变“瘦”，其原理就是在原图的小区域内去局部最小值

因为是二值化图，只有 0 和 255，所以小区域内有一个是 0 该像素点就为 0：

膨胀：取的是局部最大值，效果就是把图片变"胖"，只需要指定核大小即可。

开运算：先腐蚀后膨胀，可以消除黑色背景中白色杂点

闭运算：先膨胀后腐蚀，可以消除白色前景的黑色杂质



形态学梯度：膨胀图减去腐蚀图 (dilation - erosion)，得到**物体轮廓。**

```
// 形态学腐蚀
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3,3),
cv::Point(-1, -1));
morphologyEx(hsv_range_image, open_image, CV_MOP_OPEN, kernel);
```

1.4 轮廓检测

简介：边缘检测能够检测出边缘，但是不是连续的，而轮廓可以将边缘连接起来成为一个整体，便于后续计算。

原理：轮廓跟踪，通过顺序逐点跟踪目标边界，最终找到所有点构成的就是边界。先从上到下，从左到右在图片中找到第一个边界点E0，之后通过4连通域(上、下、左、右)或者8连通域（不仅包括上、下、左、右还包括的左上、右上、左下、右下）来找下一个边界点。

具体方法：

- 设置一个dir数值，表示上一个像素点是如何找到本像素点的。
- 利用dir计算(dir + 3) mod 4（4连通域算式，若是8连通域则为(dir + 7) mod 8）表示本像素点开始搜索的方向。
- 这样顺时针搜索（根据0,1,2,3代表的指向的不同也可以设置成逆时针）到另一个值相同的像素点后，使该像素点为新像素点并且该像素点的dir为上一步除余的值。

```
// 轮廓检测
vector<vector<cv::Point>> contours;
vector<cv::Vec4i> hierarchy;
findContours(binary_image, contours,
hierarchy,cv::RETR_TREE,cv::CHAIN_APPROX_SIMPLE, cv::Point());
```

1.5 面积判断

确定轮廓的面积大小，将轮廓的大小进行一个排序，根据面积来判断其是否有效。

```
//比较轮廓面积(用来进行轮廓排序)
bool ContourArea(vector<cv::Point> contour1, vector<cv::Point> contour2)
{
  return cv::contourArea(contour1) > cv::contourArea(contour2);
}
sort(contours.begin(),contours.end(),ContourArea);
if(contours.size() == 0)
    return false;
//判断面积的大小
if(cv::contourArea(contours[0]) >= 2500)
    return true;
else
    return false;
```

蓝线检测

交通标志牌识别

一、YOLOV4-tiny检测

1.1 模型训练

- 编译

  使用GPU加速，CUDNN_HALF特定硬件加速、OPENCV：开启Opencv,AVX和OPENMP是CPU加速

- 数据准备

```
----VOCdevkit\
    |----VOC2020\   # 目录
    |    |----Annotations\
    |    |    |----00000001.xml # 图片标注信息
    |    |----ImageSets\
    |    |    |----Main\    # 训练：验证：测试=1:1:2
    |    |    |    |----test.txt    # 测试集
    |    |    |    |----train.txt   # 训练集
    |    |    |    |----val.txt # 验证集
    |    |----JPEGImages\
    |    |    |----00000001.jpg # 对应图片
```

- 运行脚本生成yolo训练数据
- 修改配置： `cfg/yolov4-tiny.cfg`

```
[net]
# Testing   #测试模式，测试时开启
#batch=1    #
#subdivisions=1 #
# Training  #训练模式，训练时开启，测试时注释
**batch=256**    # 每批数量，根据配置设置，如果内存小，改小batch和subdivisions，
batch和subdivisions越大，效果越好
**subdivisions=16** #
width=416
height=416
channels=3  # 输入图像width height channels 长宽设置为32的倍数，因为下采样参数是32，
最小320*320 最大608*608
momentum=0.9    # 动量参数，影响梯度下降速度
decay=0.0005    # 权重衰减正则项，防止过拟合
angle=0 # 旋转
saturation = 1.5    # 饱和度扩增
exposure = 1.5  # 曝光度
hue=.1  # 色调

learning_rate=0.00261   # 学习率，权值更新速度
burn_in=1000    # 迭代次数小于burn_in,学习率更新;大于burn_in,采用policy更新
max_batches = 500200    # 训练达到max_batches停止
policy=steps    # 学习率调整策略policy：constant，steps，exp，poly，step，sig，
RANDOM
steps=400000,450000 # 步长
scales=.1,.1    # 学习率变化比例
.....
```

- 修改`cfg/voc.names`和`cfg/voc.data`
- 开始训练

  可以使用多GPU进行训练、可以使用mAP可视化训练

Loss计算

在计算loss的时候，实际上是y_pre和y_true之间的对比：

- y_pre就是一幅**图像经过网络之后的输出，内部含有两个特征层的内容；其需要解码才能够在图上作画**
- y_true就是一个真实图像中，它的**每个真实框对应的(19,19)、(38,38)网格上的偏移位置、长宽与种类。其仍需要编码才能与y_pred的结构一致**

1.2 模型预测

- 视频预测
- 图像预测

```
conf_th=0.5
yolov4_initparams=
{"category_num":5,"model_path":"/home/nano/cv_bridge_ws/src/yolov4_tiny_trt_ros/
yolov4-tiny-608.trt"}
# 检查YOLOV4的参数
check(yolov4_initparams)
traffic_class={'obstacle':0,'stop':1,'straight':2,'left':3,'right':4}
# TRAFFIC_CLASSES_LIST =
('green_led','obstacle','red_led','stop','stright','around','left','right')
# 加载TensorRT模型
trt_yolo = TrtYOLO(yolov4_initparams["model_path"],
yolov4_initparams["category_num"], letter_box=False)
```