上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

无人驾驶算法开发

第2章 基于OpenCV车道线识别

黄宏成

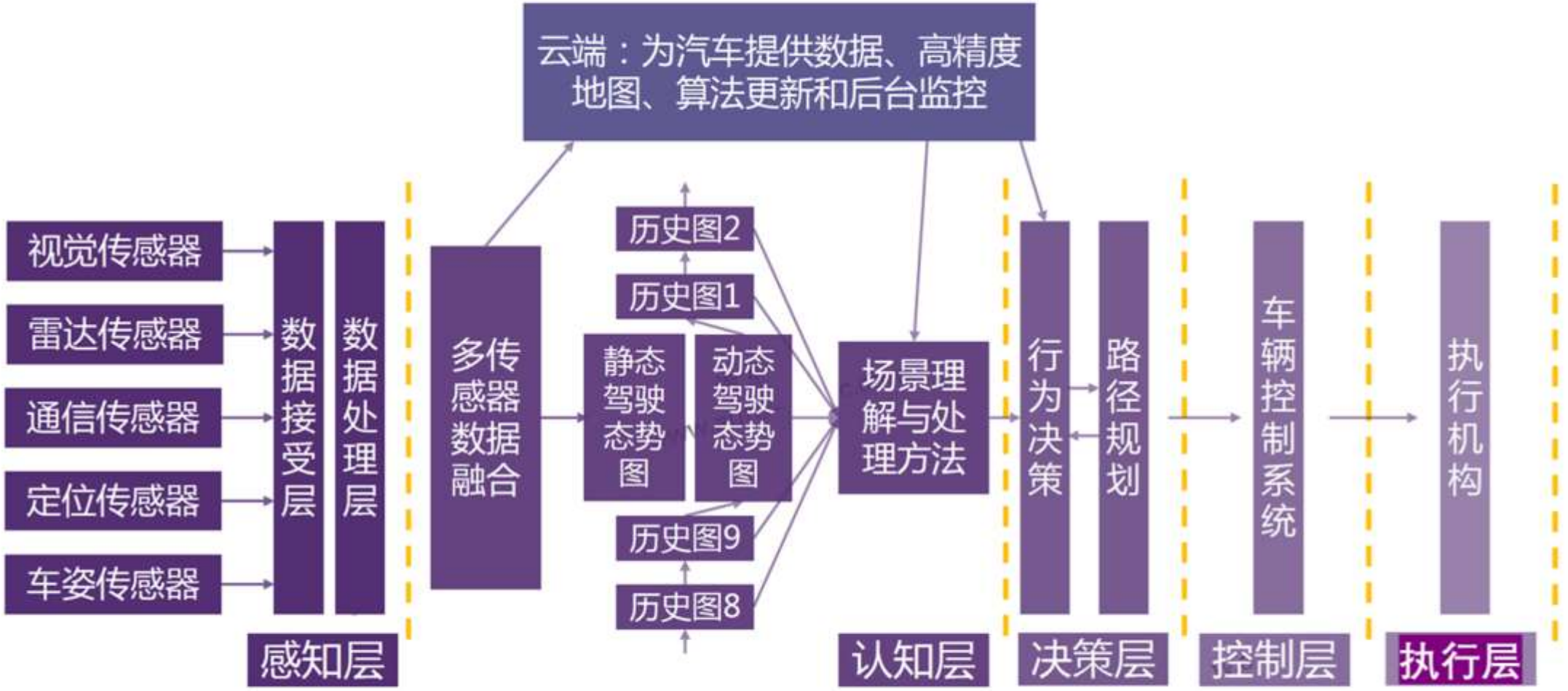
饮水思源 · 爱国荣校

目 录

01

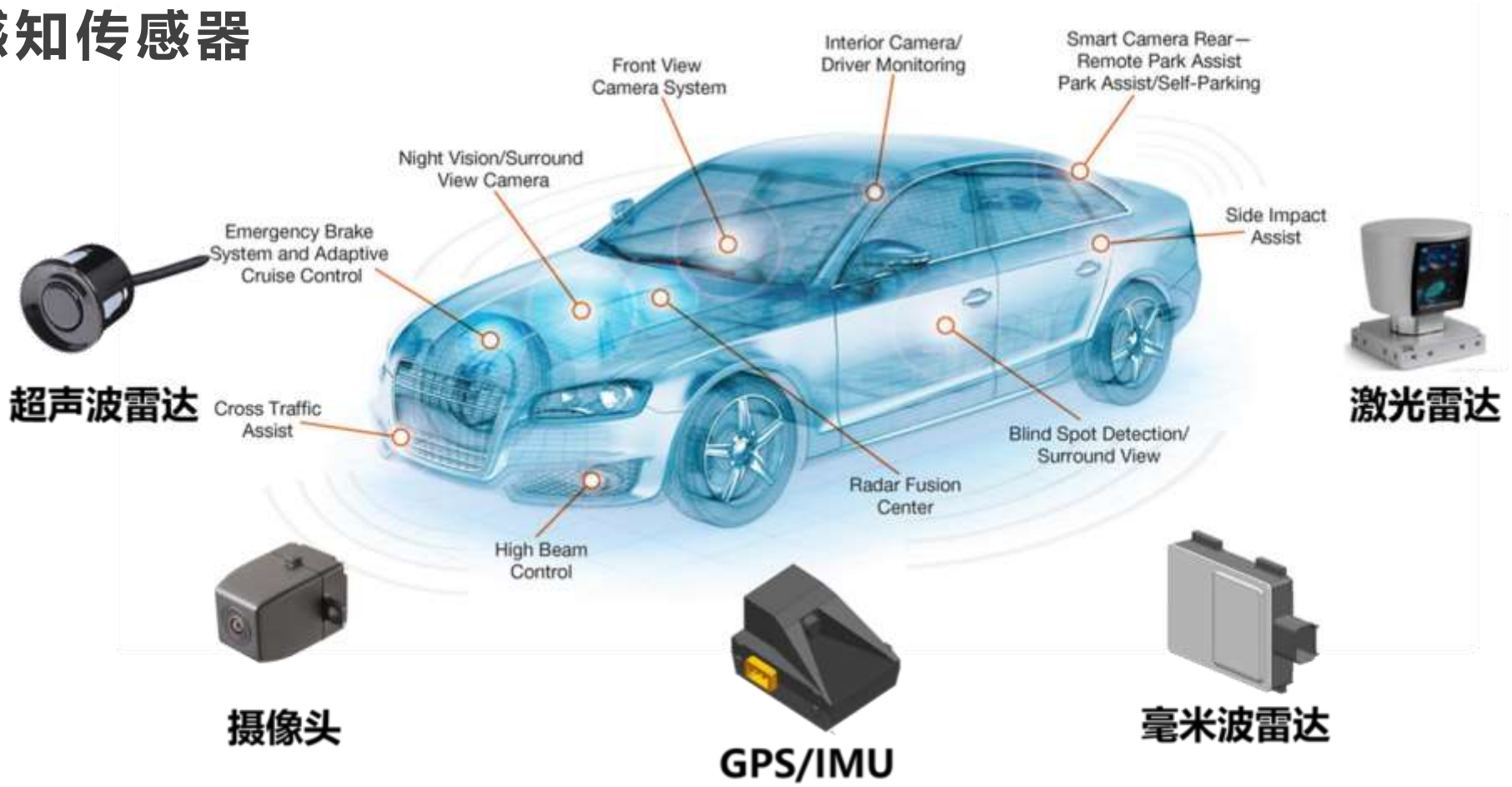
机器视觉在自动驾驶中的应用

智能网联系统架构


云端：为汽车提供数据、高精度地图、算法更新和后台监控
视觉传感器
雷达传感器
通信传感器
定位传感器
车姿传感器
数据接受层
数据处理层
感知层
多传感器数据融合
历史图2
历史图1
静态驾驶态势图
动态驾驶态势图
历史图9
历史图8
场景理解与处理方法
认知层
行为决策
路径规划
决策层
车辆控制系统
控制层
执行机构
执行层

感知系统

感知传感器


Front View Camera System
Interior Camera/ Driver Monitoring
Smart Camera Rear – Remote Park Assist Park Assist/Self-Parking
Night Vision/Surround View Camera
Emergency Brake System and Adaptive Cruise Control
Side Impact Assist
超声波雷达
激光雷达
Cross Traffic Assist
Blind Spot Detection/ Surround View
Radar Fusion Center
High Beam Control
摄像头
GPS/IMU
毫米波雷达

感知系统

- 智能化要求车辆具有强大的环境感知能力,车载传感器是硬性需求
- 为适应复杂路况与天候，智能车辆需要多种类型的传感器

LIDAR

- 较高的测距建图精度与大视场，可用于大范围的环境感知
- 受天气影响较大，成本高昂

毫米波雷达

- 较远的感知范围，用于前后向车辆障碍检测，成本相对低廉
- 视场与精度一般不如LIDAR

摄像头

- 感知信息丰富，配合红外镜头可实现日夜工作，可用于行人识别，辅助定位。
- 数据处理工作量大，多视场信息融合难度大

超声波雷达

- 技术成熟，成本较低，可大量使用，应用于盲点监测和倒车辅助
- 感知范围近，适用于相对速度较低的环境

感知系统

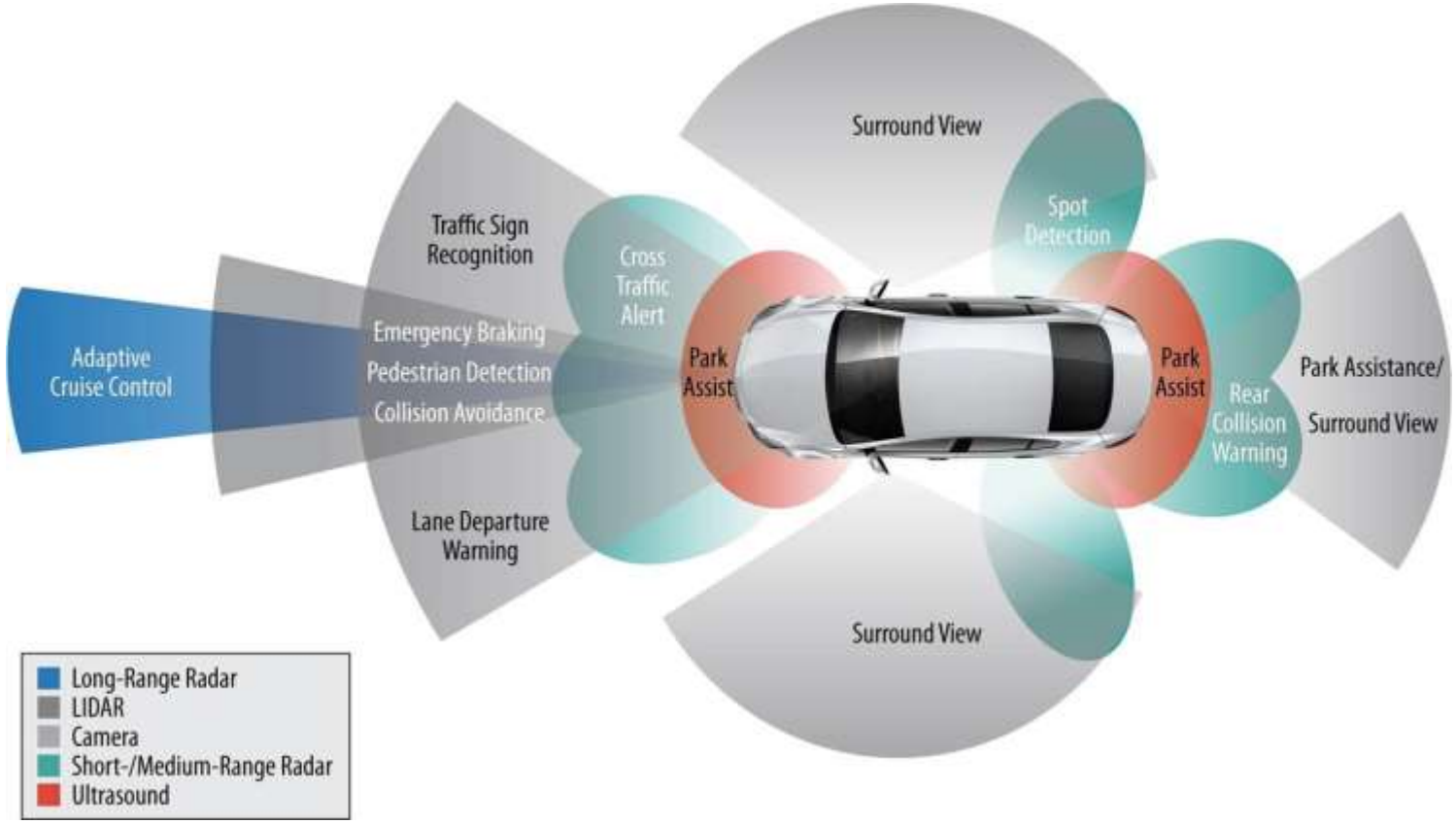
智能汽车常用传感器指标对比

传感器 指标项	激光雷达		毫米波雷达 24G/77G	视觉相机		红外	超声波雷达
	前向	全向		单目	双目		
成本（美元）	1000-8000		100-150	150-300		50-200	1-5
探测距离（m）	80-150		100-250	6-100		150-400	5
探测角度（°）	15-360		10-70	30		30	120
精度	优	优	良	一般	优	一般	一般
分辨率	优	优	良	一般	优	一般	一般
动态范围	优	良	优	一般	良	良	一般
主动与被动	主动	主动	主动	被动	被动	被动	主动
误报率	良	良	优	良	良	良	良
温度适应性	优	优	优	优	优	良	良
黑暗适应性	优	优	优	一般	一般	优	优
天气适应性	良	良	优	一般	一般	一般	良

多种传感器各有优势与劣势，传感器相互补充，形成对周围环境的精确感知。

感知系统

- 最远探测距离：摄像头 50–250米，超声波雷达 3–10米，毫米波雷达–10/50/100/200米，激光雷达–300米



感知系统

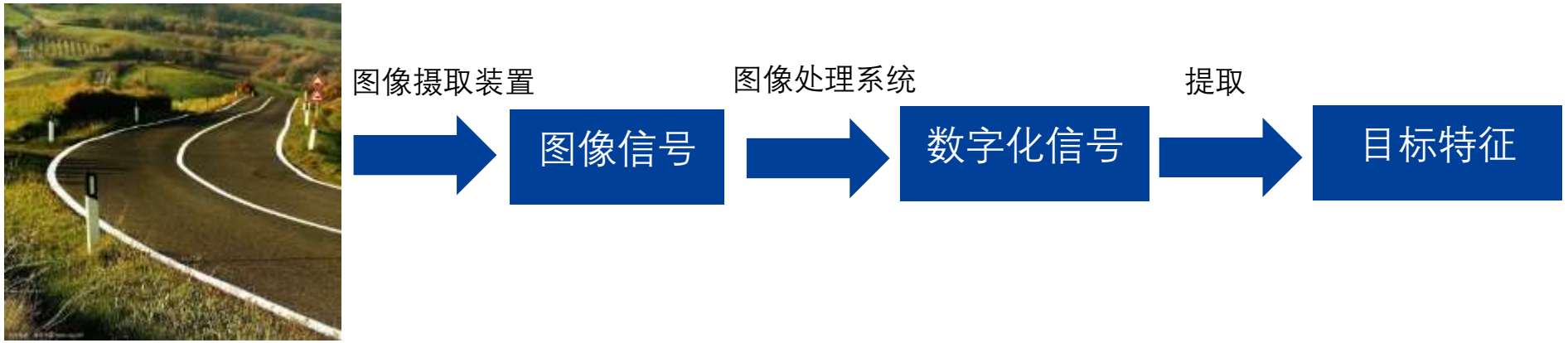
自动驾驶感知技术路线

视觉派
(Tesla)

激光雷达派
(Waymo)

机器视觉

- 机器视觉系统：通过图像摄取装置（分CMOS和CCD两种）将被摄取目标转换成图像信号，传送给专用的图像处理系统，得到被摄目标的形态信息，根据像素分布和亮度、颜色等信息，转变成数字化信号；图像系统对这些信号进行各种运算来抽取目标的特征，实现自动识别功能。



机器视觉

- 根据镜头的数目可以分为单目相机和双目相机

  单目体积小、对计算要求小、但测距不准确；双目体积大、对计算要求高，但测距精度高。

- 按照摄像头功能来看，除普通摄像头外，还有鱼眼摄像头和红外摄像头

  鱼眼其构造仿照鱼类眼睛成像，成像范围广；红外摄像头通过收集红外线，可在夜间成像。

单目摄像头

双目摄像头

鱼眼摄像头

红外摄像头

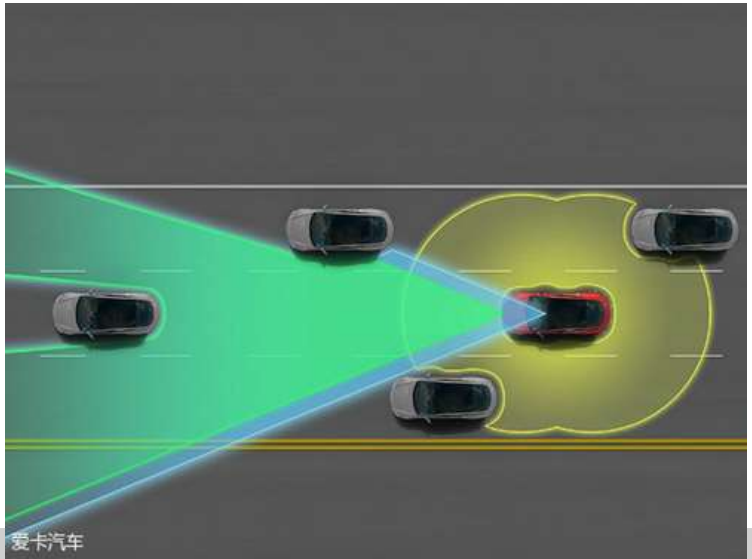
车载摄像头由于具有成本低、成像色彩丰富、体积小等优点已经被广泛应用到智能汽车，在车道线识别、交通信号灯识别等应用中具有不可替代的地位。

机器视觉应用

Tesla自动驾驶（Autopilot）感知系统

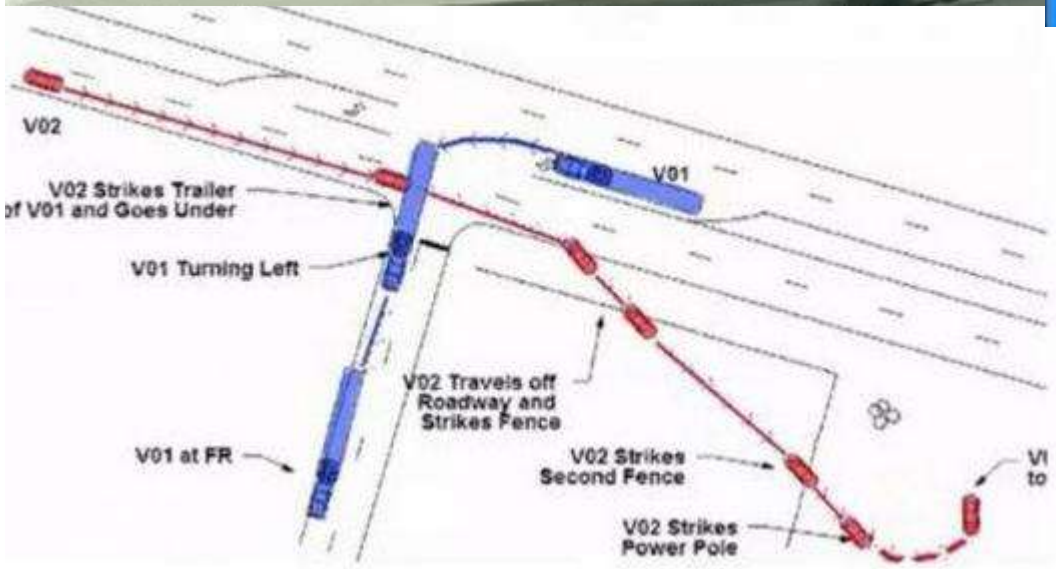
1、车头6个、车尾6个，共12个超声波传感器；
2、前挡风玻璃下内后视镜上端长焦、标准、短焦各一，共3个前视摄像头；
3、外后视镜前端翼子板后端两侧各一个，共2个后测视摄像头；
4、前后门中柱靠上端两侧各一个，共2个侧视摄像头；
5、后窗上部中间1个后视摄像头；
6、一个在后车牌位置上部专门 负责倒车的倒车摄像头1个；
7、前保险杠中部的毫米波雷达1个。

各传感器（图中从右至左）	可测最大距离（m）
长焦摄像头	250
毫米波雷达	160
标准摄像头	150
侧视摄像头	80
短焦摄像头	60
超声波传感器	8
后视摄像头	50
后测视摄像头	100



机器视觉应用

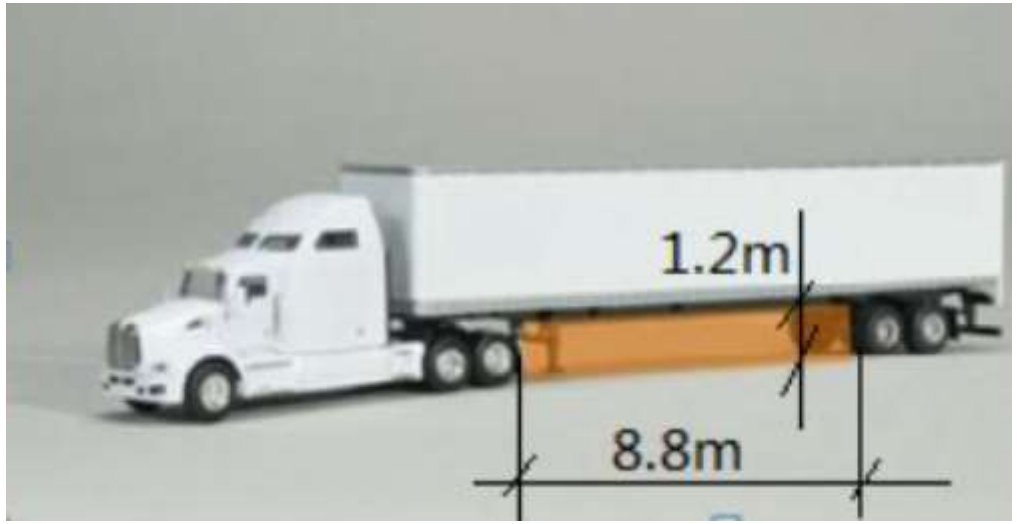
Tesla自动驾驶（Autopilot）感知系统



机器视觉应用

Tesla自动驾驶（Autopilot）感知系统

视觉系统在强光下把货车白色车身误认为是一朵白云
Tesla毫米波雷达安装位置离地约0.4m；货车底盘高约1.2m，橙色区域8.8m长，该区域是雷达的盲区



机器视觉应用

- Tesla自动驾驶（Autopilot）感知系统

机器视觉应用

- Tesla纯视觉驾驶系统（FSD10.0）

02

OpenCV基础

为什么选择OpenCV

- OpenCV于1999年由**Intel**建立，如今由Willow Garage提供支持。OpenCV是一个基于Apache2.0许可（开源）发行的跨平台计算机视觉库，可以运行在Linux、Windows和Mac OS操作系统上。用C++编写，它轻量级而且高效——由一系列 C 函数和少量 C++ 类 构成，同时提供了Python、Ruby、MATLAB等语言的API接口，实现了图像处理和计 算机视觉方面的很多通用算法。最新版本是4.3 ， 2020年4月6日发布 。
- 优势：跨平台（win Linux，Macos，Android）、开源免费、高效(下载人数1400万，4.7万用户社区人数)，快速部署
- 应用领域： 包含300多种图像处理和计算机视觉方面的函数与库，广泛应用与以下领域,1、人机互动2、物体识别3、图像分割4、人脸识别5、动作识别6、运动跟踪7、机器人8、运动分析9、机器视觉10、结构分析11、汽车安全驾驶

安装Anaconda和OpenCV模块

- Anaconda指的是一个开源的Python发行版本，其包含conda、Python等180多个科学包及其依赖项。 因为包含了大量的科学包。支持 Linux, Mac, Windows系统，提供了包管理与环境管理的功能，可以很方便地解决多版本python并存、切换以及各种第三方包安装问题
- Anaconda 下安装OpenCV模块
- pip install opencv-python

Jutyper Notebook使用

标题

1个#是一级标题，2个#是二级标题，以此类推。支持六级标题。

字体

- 加粗，要加粗的文字左右分别用两个*号包起来
- 斜体，要倾斜的文字左右分别用一个*号包起来
- 斜体加粗，要倾斜和加粗的文字左右分别用三个*号包起来
- 删除线，要加删除线的文字左右分别用两个~~号包起来

Jutyper Notebook使用

列表

- 无序列表用 - + * 任何一种都可以，- + * 跟内容之间要有一个空格
- 有序列表，数字加点，序号跟内容之间要有空格
- 列表嵌套，上一级和下一级之间敲三个空格即可

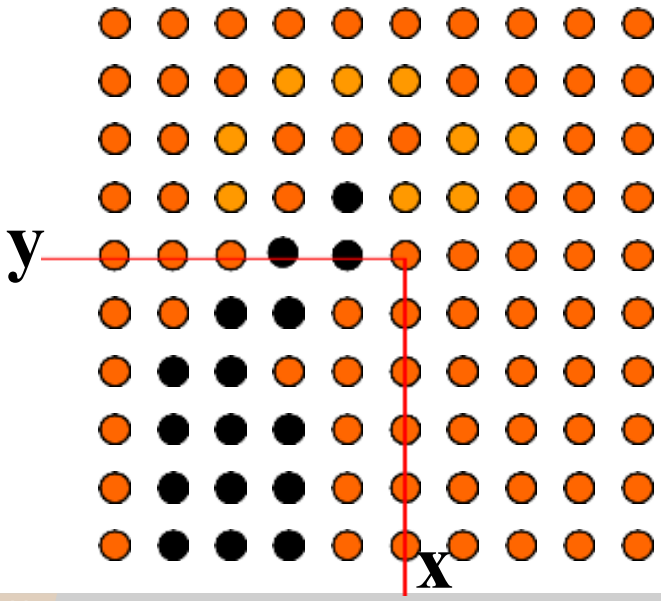
表格

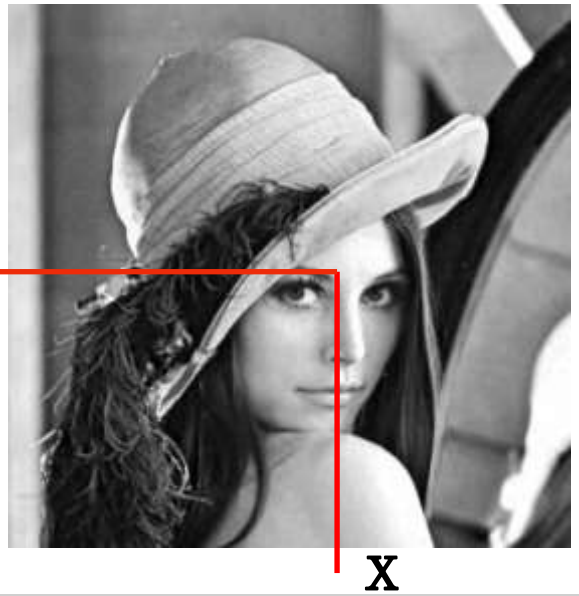
- 表头|表头|表头---|:--:|---:内容|内容|内容内容|内容|内容第二行分割表头和内容。- 有一个就行，为了对齐，多加了几个文字默认居左-两边加：表示文字居中-右边加：表示文字居右注：

图像处理与视觉基础

像素与数字图像的表征

- 数字图像由二维的元素组成，每一个元素具有一个特定的位置（x, y）和幅值f(x, y)，这些元素就称为像素。像素组成的二维排列，可以用矩阵表示，用于图像的表征。
- 灰度图：对于单色（灰度）图像而言，每个像素的亮度用一个数值来表示，通常数值范围在0到255之间，0表示黑、255表示白，其它值表示处于黑白之间的灰度。
- 彩色图：彩色图像可由三个（如BGR,HSV）二维灰度（或亮度）函数f(x,y)组成，也可以用红、绿、蓝三元组的二维矩阵来表示。 通常，三元组的每个数值也是在0到255之间，0表示相应 的基色在该像素中没有，而255则代表相应的基色在该像 素中取得最大值。





OpenCV基本操作

Python-openCV的图像处理操作

- 读取、显示指定位置的图片文件
- 读取视频操作
- 存储视频文件
- OpenCV中的绘画函数
- 对图片进行灰度化
- 对图片进行二值化

03

车道线识别

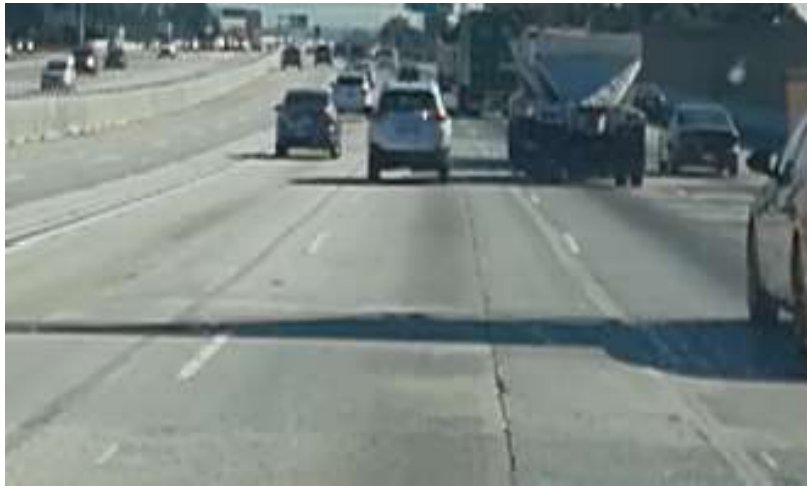
车道线识别

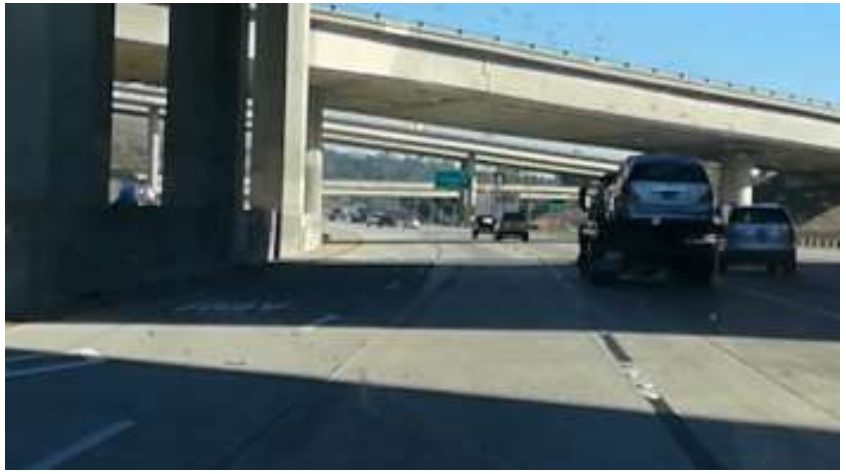
车道线检测的目标与意义

车道线检测系统通过运用数字图像处理及模式识别技术，在道路图像中有效提取车道线的信息并拟合车道线，帮助自动驾驶过程中的道路线识别、自动泊车及防碰撞预警等系统

车道线识别

- 几种车道线检测的恶劣情况





车道线识别

车道线检测框架

车道线识别

正常条件下车道线检测的主要步骤

- 视频读入——分帧——预处理
- 灰度变换（Gray Scale Transformation）
- 高斯平滑处理（Gaussian Smoothing）
- 基于Canny算子的边缘检测（Canny Edge Detection）
- 基于兴趣区域的边缘过滤（ROI (Region of Interest) Based Edge Filtering）
- 基于哈夫变换的车道线检测（Hough Transformation）
- 将检测出的车道线与原图合并（Lane Extrapolation）

04

简单车道线识别

简单车道线识别

程序框架

1、#读图，或者读视频，为了演示方便，我们这里处理单帧图像

- img = cv2.imread('E:\\linedetect\\video_2[00_00_02][20180226-100403-1].JPG')

2、#选取关键兴趣区域，减少计算量

- roi_vtx = np.array([[(0, img.shape[0]), (460, 325), (520, 325), (img.shape[1], img.shape[0])]])

3、#取图，灰度化

- gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

4、#高斯平滑，减少边缘干扰

- blur_gray = cv2.GaussianBlur(gray, (blur_ksize, blur_ksize), 0, 0)

5、#利用canny算子，进行边缘检测

- edges = cv2.Canny(blur_gray, canny_lthreshold, canny_hthreshold)

6、#集中到边缘检测的兴趣区域，进一步减少运算量

- roi_edges = roi_mask(edges, roi_vtx)

7、#利用哈夫变换变换，进行直线检测

- line_img = hough_lines(roi_edges, rho, theta, threshold, min_line_length, max_line_gap)

8、#将检测出来的直线与院图进行合成

- res_img = cv2.addWeighted(img, 0.8, line_img, 1, 0)

简单车道线识别

- **图像平滑**
- 图像平滑主要用来抑制噪声，等价于抑制频域中的高频成分
- 边缘也属于频域中的高频，平滑同时会模糊承载图像中重要信息的 边缘，需要集中考虑具有边缘保持功能的平滑方法
- 此类方法的基本思路：仅使用邻域中与被处理像素有类似性质的点 进行平均
- 图像平滑的常用方法
  - 平均(averaging)
  - 限制数据有效性下的平均(averaging with limited data validity)
  - 反梯度平均(averaging according to inverse gradient)
  - 旋转掩膜平均(averaging using a rotating mask)
  - 中值滤波(median filtering)
  - 高斯平滑（Gaussian Smoothing）

简单车道线识别

- 高斯平滑(Gaussian Smoothing)的原理与应用

- Gaussian Smoothing是对图片apply一个Gaussian Filter，可以起到模糊图片和消除噪声的效果。Gaussian Filter是一种低通过滤器，能够抑制图片中的高频部分，而让低频部分顺利通过。
- 什么是图片的高频部分？——下图给出了一个比较极端的例子。相机ISO适当时能够得到右侧图片，画质细腻；如果ISO过大，就会导致产生左侧图片，画质差，噪点多。这些噪点就是图片中的高频部分，表示了像素值剧烈升高或降低。

The impact of luminance and, especially chrominance noise, is visible in these two heavily cropped, detail shots -- the first at ISO 12800 and the second at ISO 800, where noise is well-controlled -- with its High ISO Noise Reduction turned off. Chrominance noise can be seen as the little pastel-colored specks in mid-tone and dark areas. This was taken several years ago with an older EOS Rebel T1i camera, but today's DSLRs definitely have better performance at high ISO settings.

简单车道线识别

- 在opencv中，实用gauss滤波函数做平滑

利用高斯平滑，去除高频噪点的同时，实现边缘保持，为下一步的边缘检测做好准备。

blur_ksize,表示高斯矩阵的长与宽,都是5，标准差取0时OpenCV会根据高斯矩阵的尺寸自己计算。概括地讲，高斯矩阵的尺寸越大，标准差越大，处理过的图像模糊程度越大。 cv2.GaussianBlur(src,ksize,sigmaX) kSize:核大小,sigmaX:高斯核在x轴的标准差,如果为0会根据核的宽和高重新计算

```
blur_ksize = 5 # Gaussian blur kernel size
blur_gray = cv2.GaussianBlur(gray, (blur_ksize, blur_ksize), 0, 0)
```

灰度图

高斯平滑图

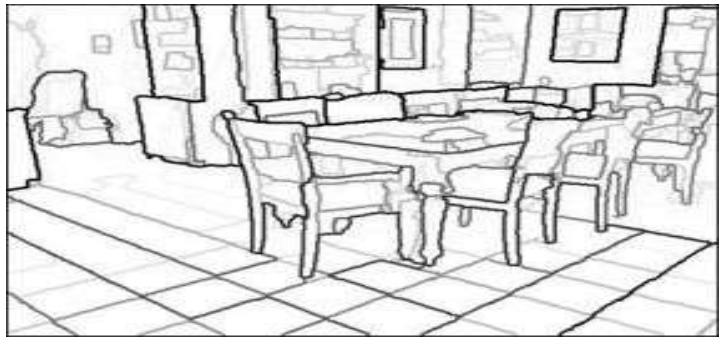
简单车道线识别

- 图像中的边界与边缘

- 边界(border)
  - 区域R的边界是其自身的**像素集合**，其中的每个点具有一个或多个R外的邻接点。指检测图像中的对象边界，更偏向于关注上层语义对象

- 边缘(edge)
  - 边缘是图像上灰度的不连续点，或者灰度变化剧烈的地方，它是一个**有大小和方向的矢量**

边界和边缘不同，**边界**是与区域有关的全局概念，而**边缘表示图像函数的局部性质**；边界与边缘也相互关联，一种寻找边界的方法是连接显著的边缘



边缘检测的Canny准则

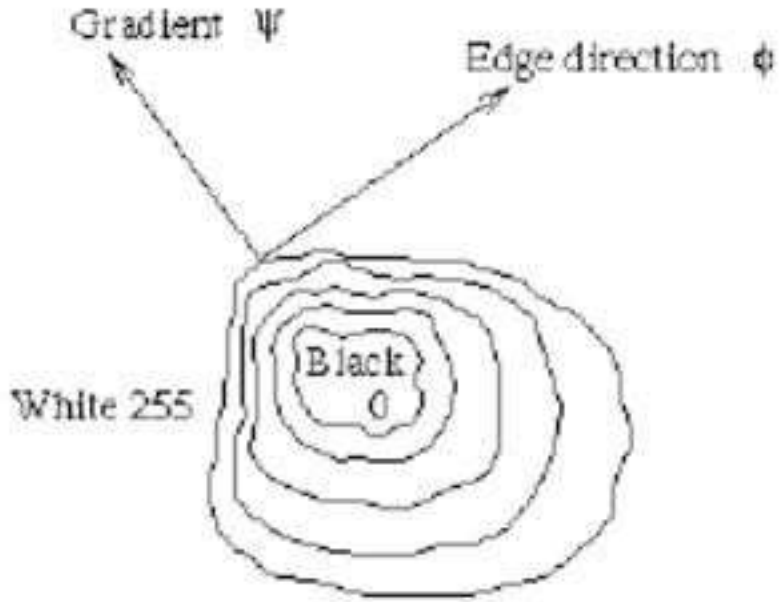
John F. Canny在1986年发明了Canny Edge Detection技术，其基本原理是对图片中各个点求gradient，gradient较大的地方往往是edge。

- 假设
  - 信号：受白噪声影响的阶跃边缘
  - 滤波器：有限脉冲响应滤波器
- Canny最优化准则
  - 检测标准（最大信噪比准则）：要有好的检测结果，不丢失重要的边缘，不应有虚假的边缘
  - 定位标准（最优过零点准则）：实际边缘位置与检测到的边缘位置间的偏差最小
  - 单响应标准（多峰值响应准则）：对实际上的同一边缘要有低的响应次数
- Canny以一维形式为例，给出了三条准则的数学表达式，将寻找最优滤波器的问题转换为泛函的约束优化问题

Canny准则下的最优边缘检测滤波器

- 针对一维信号和前两个最优化准则，可用变分法求得滤波器的完整解
- 加入第三个最优化准则，则需要通过数值优化方法得到最优解

- 对于二维情况，阶跃边缘由位置、方向、幅度确定，可以证明，将图像与一对称的二维高斯函数做卷积再沿梯度方向微分，就构成了一个简单而有效的滤波器
- 因此，我们之前选择高斯函数做卷积进行滤波。



Canny边缘检测（I）

- 双阈值技术

边缘检测通过阈值化确定突出的边缘，对噪声引起的单边缘虚假响应会造成边缘不连续，这是由于滤波结果超出或低于阈值所致，这种问题可通过滞后阈值化处理解决

- 图像信号的响应大于高阈值，它一定是边缘
- 图像信号的响应小于低阈值，它一定不是边缘
- 图像信号的响应在高低阈值之间，如果它与大于高阈值的像素相连，它也可能是边缘
- 高、低阈值可根据对信噪比的估计确定

Canny Edge Detection精妙的地方在于它有两个参数：low_threshold和high_threshold。算法先比较gradient与这两个threshold的关系，如果gradient > high_threshold，就承认这是一个edge point；如果gradient < low_threshold，就断定这不是edge point；对于其他的点，如果与edge point相连接，那么这个点被认为也是edge point，否则不是。

Canny边缘检测（II）

算法：Canny边缘检测

1. 将图像f与尺度为σ的二维高斯函数h做卷积以消除噪声

$$g = f * h(\sigma)$$

2. 对g中的每个像素计算梯度的大小和方向

$$|\nabla g| = \sqrt{g_x^2 + g_y^2} = \sqrt{(\frac{\partial g}{\partial x})^2 + (\frac{\partial g}{\partial y})^2} \qquad \theta = \arctan(\frac{g_x}{g_y})$$

3. 根据像素梯度方向，获取该像素沿梯度的邻接像素
4. 非极大值抑制：遍历g，若某个像素的灰度值与其梯度方向上前后两个像素的灰度值相比并非最大，则该像素不是边缘

Canny边缘检测（III）

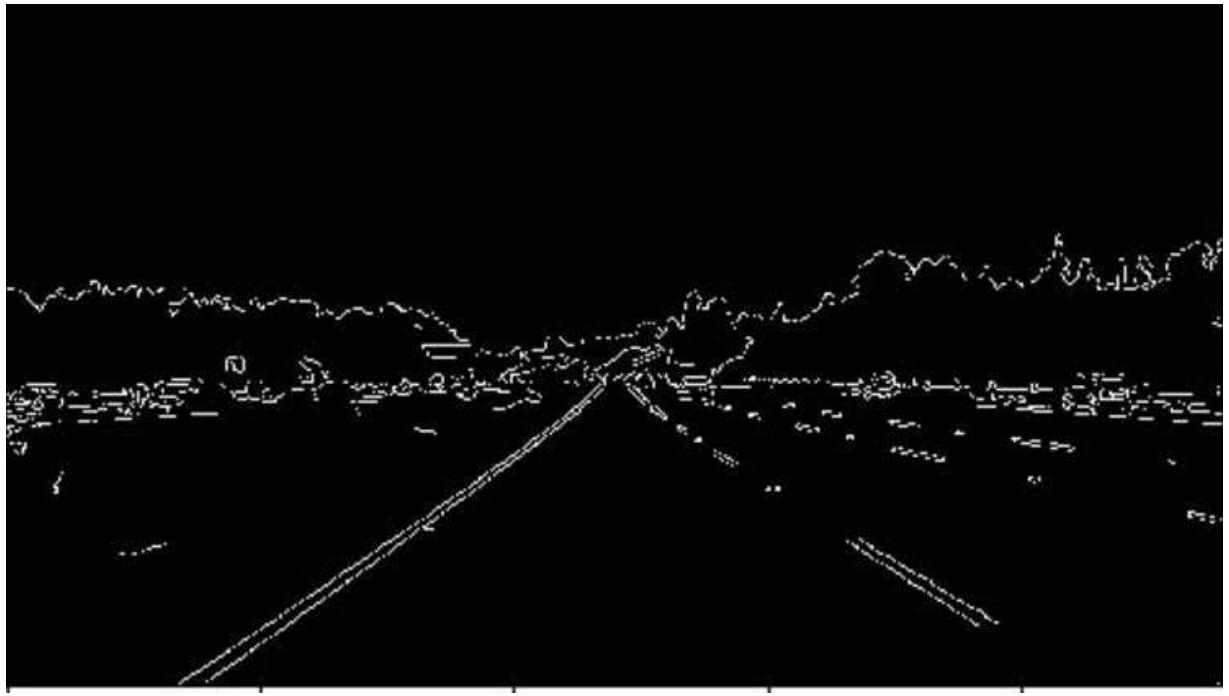
算法：Canny边缘检测（续）

5. 滞后阈值化处理：设定高低阈值，凡是大于高阈值的一定是边缘；凡是小于低阈值的一定不是边缘；检测结果在高低阈值之间的，看其周边8个像素中是否有超过高阈值的边缘像素，有则为边缘，否则不是边缘
6. 对于递增的标准差σ'，重复上述步骤1－5
7. 用特征综合方法，收集来自多尺度的最终边缘信息

- 参考资料
  - “A Computational Approach to Edge Detection”, IEEE Trans. PAMI, Vol. PAMI-8, No. 6, 1986, pp:679~698

基于Opencv的Canny边缘检测

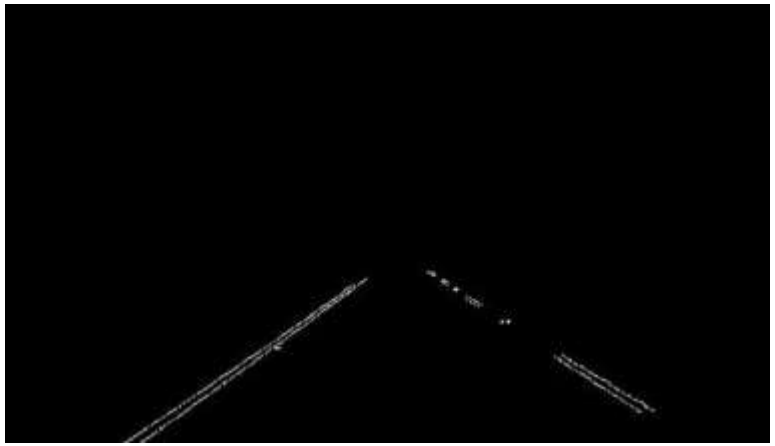
```
canny_lthreshold = 50
canny_hthreshold = 150 #
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
```



基于ROI的去除其他不感兴趣区域边缘线

- 边缘检测后东西有点儿太多了，我们只关心车道线
- 由于camera相对于车是固定的，而无人车相对于车道的左右位置也是基本固定的，所以车道在camera视频中基本保持在一个固定区域内！据此我们可以画出一个大概的Region of Interest (ROI)，过滤掉ROI之外的edges。

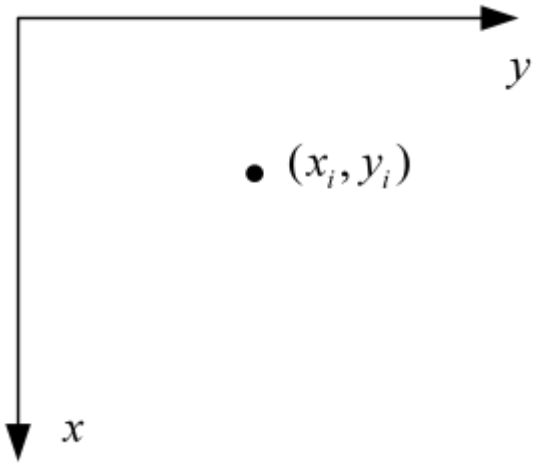
```
def roi_mask(img, vertices):
  mask = np.zeros_like(img)
  mask_color = 255
    cv2.fillPoly(mask, vertices, mask_color)
 masked_img = cv2.bitwise_and(img, mask)
 return masked_img
roi_vtx = np.array([[(0, img.shape[0]), (460, 325), (520, 325), (img.shape[1], img.shape[0])]])
roi_edges = roi_mask(edges, roi_vtx)
```



基于Hough变换的线检测

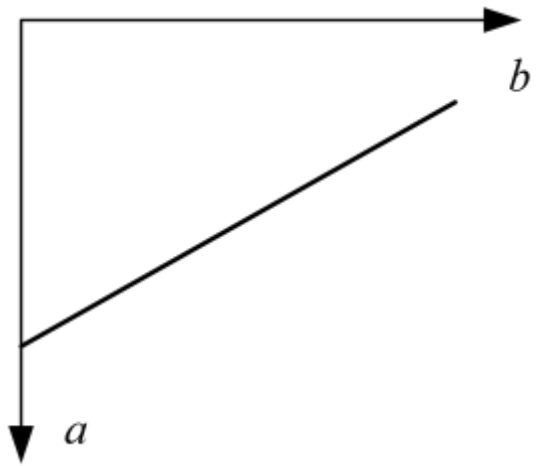
在数字图像中，往往存在着一些特殊形状的几何图形，像检测马路上的一条直线，检测人眼的圆形等等，有时我们需要把这些特定图形检测出来，hough变换就是这样一种检测的工具。

基本原理：点 - 线的对偶性质



$$y_i = ax_i + b$$

XY平面

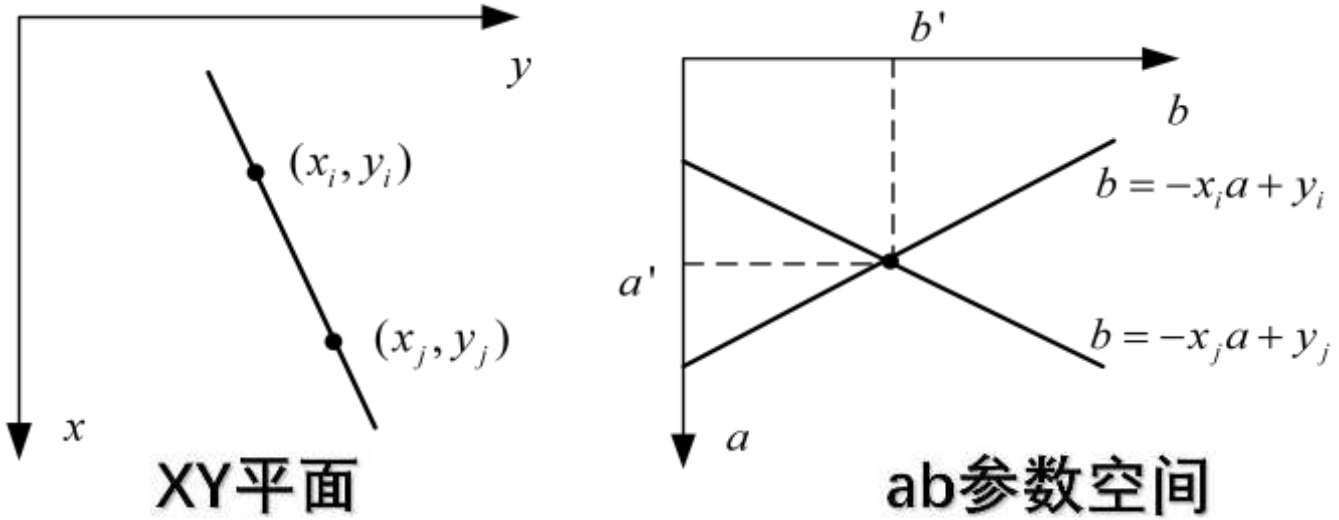


$$b = -x_i a + y_i$$

ab参数空间

直线检测

- 基于Hough变换的线检测
  - 基本原理：点 - 线的对偶性质



在XY平面检测直线，转化为在参数空间检测交点

Hough变换的性质

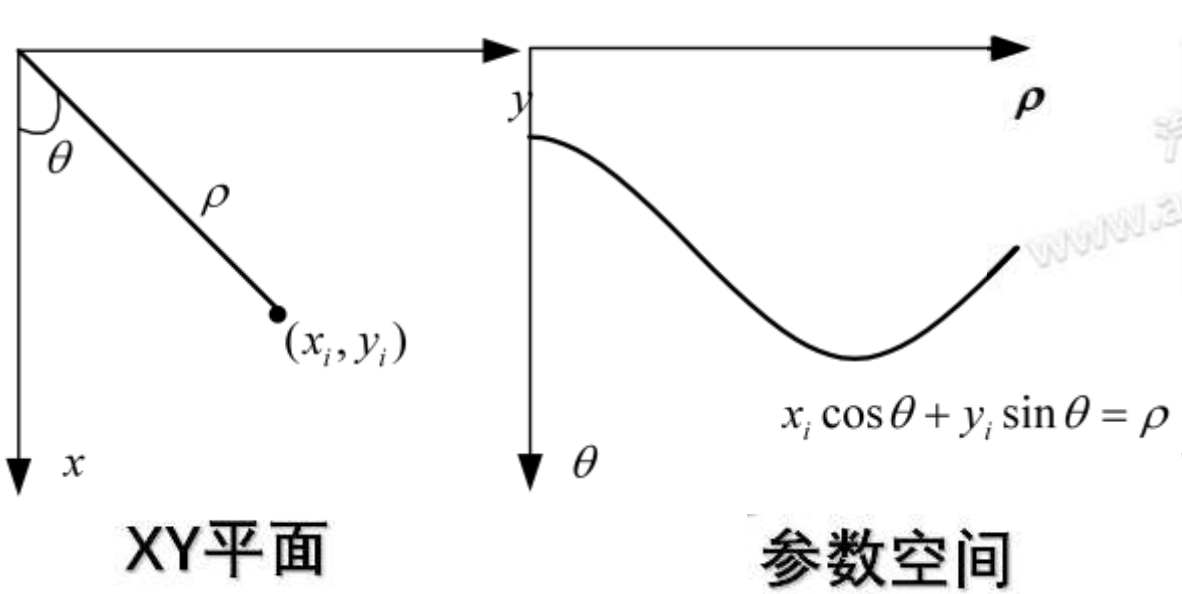
1、直角坐标系中的一个点映射到参数空间中为一条正弦曲线；
2、参数空间中的一个点对应直角坐标系中的一条直线；
3、直角坐标系中的共点线映射到参数空间中为一条曲线；
4、直角坐标系中的共线点映射到参数空间后为一个交于同一点的曲线簇

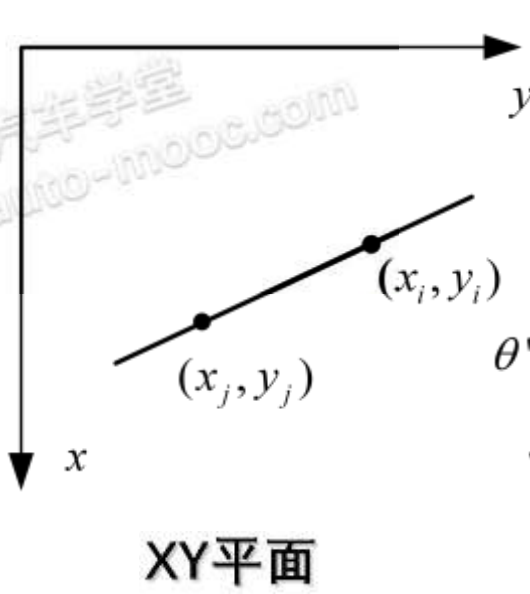
基于Hough变换的线检测

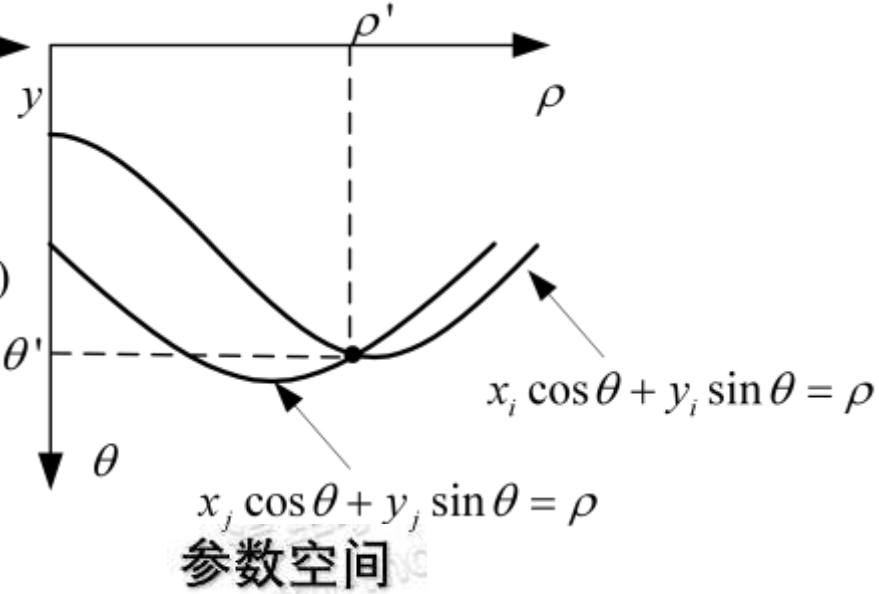
- 使用直线的标准表示法

$$x\cos\theta + y\sin\theta = \rho, \rho \geq 0, 0 \leq \theta \leq 2\pi$$

- 点－正弦曲线的对偶（若能在参数空间中确定交点，就实现了直线检测）







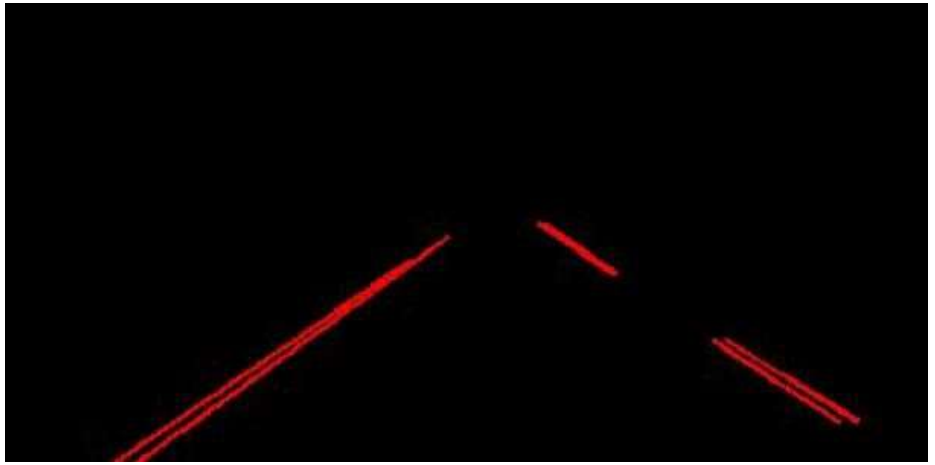
基于Hough变换的线检测算法

- 算法： 基于Hough变换的线检测

1. 将参数空间细分为m×n个累加单元，并设置累加器矩阵Q
2. 将累加器矩阵Q(m×n)的初始值置为零
3. 对于XY平面上的点(xi，yi)（i=1，2，3…s，s为平面上的已 知点数），令θ等于θ轴上允许的细分值，计算对应的ρ
4. 将得到的ρ值四舍五入为最接近ρ轴上的允许细分值
5. 在累加器矩阵Q中，找到所对应的单元，并将该单元的累 加值加1
6. 重复3-5步直到处理完XY平面上的所有点，在累加器矩阵Q 中，值最大单元对应的θ和ρ即为XY平面直线方程的参数

基于opencv的实现

在实际操作中，我们往往将Hough space划分为网格状，如果经过一个格子的线的数目大于某threshold，我们认为这个经过这个格子的线在原image space对应的点应在同一条线上。具备了这些知识，我们可以用Hough Transformation来找线啦！

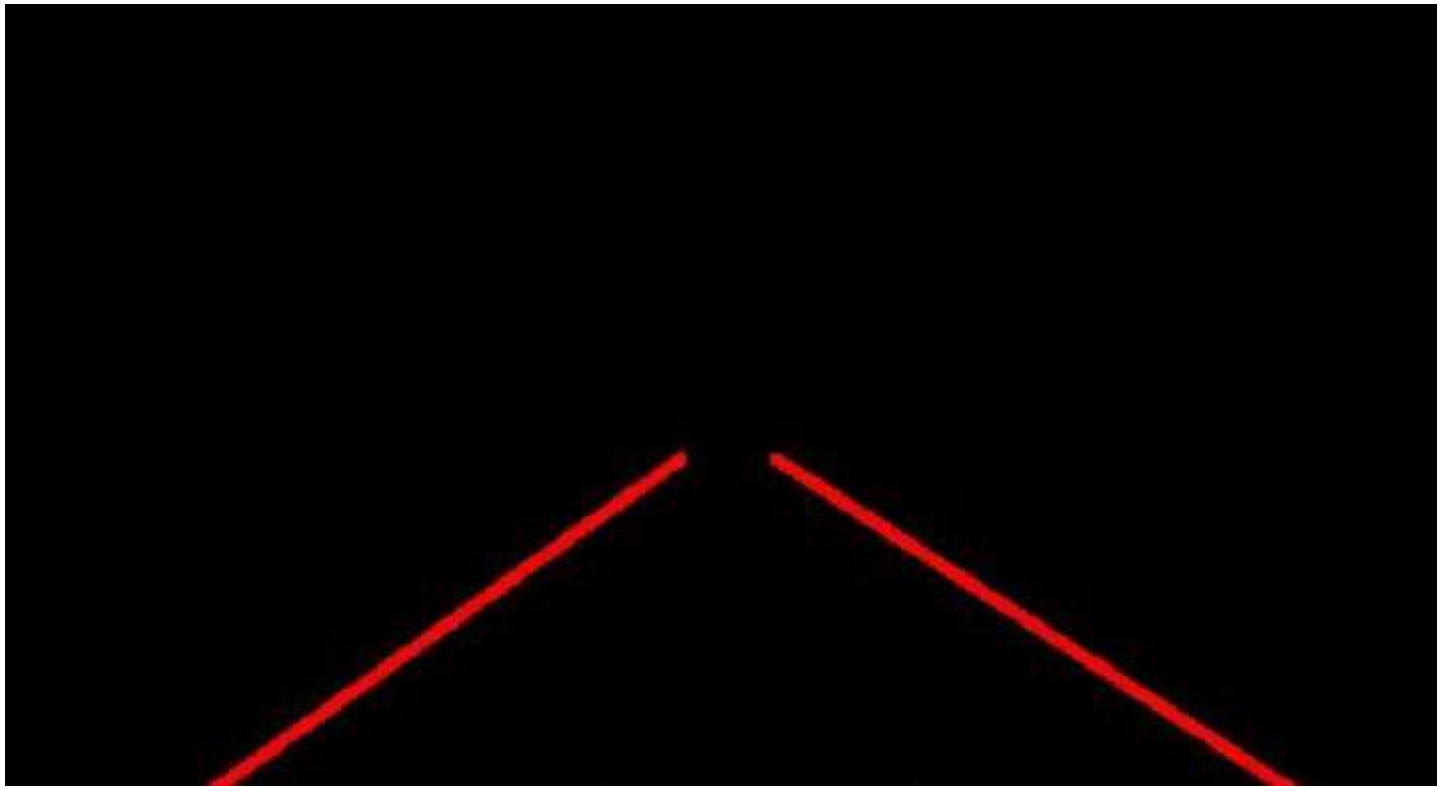


```
# Hough transform parameters
rho = 1
theta = np.pi / 180
threshold = 15
min_line_length = 40
max_line_gap = 20

def draw_lines(img, lines, color=[255, 0, 0], thickness=2):
  for line in lines:
    for x1, y1, x2, y2 in line:
      cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold,
                min_line_len, max_line_gap):
  lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
                          minLineLength=min_line_len,
                          maxLineGap=max_line_gap)
  line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
  draw_lines(line_img, lines)
  return line_img


line_img = hough_lines(roi_edges, rho, theta, threshold,
                       min_line_length, max_line_gap)
```

Lane Extrapolation

现在我们要根据得到的线计算出左车道和右车道，一种可以采用的步骤是：

1. 根据斜率正负划分某条线属于左车道或右车道
2. 分别对左右车道线移除outlier：迭代计算各条线的斜率与斜率均值的差，逐一移除差值过大的线
3. 分别对左右车道线的顶点集合做linear regression，得到最终车道。



定义draw_lanes 函数

```
def draw_lanes(img, lines, color=[255, 0, 0], thickness=8):
    left_lines, right_lines = [], []
    for line in lines:
        for x1, y1, x2, y2 in line:
            k = (y2 - y1) / (x2 - x1)
            if k < 0:
                left_lines.append(line)
            else:
                right_lines.append(line)

    if (len(left_lines) <= 0 or len(right_lines) <= 0):
        return img

    clean_lines(left_lines, 0.1)
    clean_lines(right_lines, 0.1)
    left_points = [(x1, y1) for line in left_lines for x1,y1,x2,y2 in line]
    left_points = left_points + [(x2, y2) for line in left_lines for x1,y1,x2,y2 in line]
    right_points = [(x1, y1) for line in right_lines for x1,y1,x2,y2 in line]
    right_points = right_points + [(x2, y2) for line in right_lines for x1,y1,x2,y2 in line]

    left_vtx = calc_lane_vertices(left_points, 325, img.shape[0])
    right_vtx = calc_lane_vertices(right_points, 325, img.shape[0])

    cv2.line(img, left_vtx[0], left_vtx[1], color, thickness)
    cv2.line(img, right_vtx[0], right_vtx[1], color, thickness)
```

迭代计算各条线的斜率，根据斜率分别对左右车道线移除outlier

```
def clean_lines(lines, threshold):
  slope = [(y2 - y1) / (x2 - x1) for line in lines for x1, y1, x2, y2 in line]
  while len(lines) > 0:
    mean = np.mean(slope)
    diff = [abs(s - mean) for s in slope]
    idx = np.argmax(diff)
    if diff[idx] > threshold:
      slope.pop(idx)
      lines.pop(idx)
    else:
      break


def calc_lane_vertices(point_list, ymin, ymax):
  x = [p[0] for p in point_list]
  y = [p[1] for p in point_list]
  fit = np.polyfit(y, x, 1)
  fit_fn = np.poly1d(fit)

  xmin = int(fit_fn(ymin))
  xmax = int(fit_fn(ymax))

  return [(xmin, ymin), (xmax, ymax)]
```

在原始图上划线叠加

```
cv2.addWeighted(img, 0.8, line_img, 1, 0)
```

构建process_an_image函数

将前面的代码打个包放到叫process_an_image的函数中

```
def process_an_image(img):
    roi_vtx = np.array([[(0, img.shape[0]), (460, 325), (520, 325), (img.shape[1], img.shape[0])]])
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    blur_gray = cv2.GaussianBlur(gray, (blur_ksize, blur_ksize), 0, 0)
    edges = cv2.Canny(blur_gray, canny_lthreshold, canny_hthreshold)
    roi_edges = roi_mask(edges, roi_vtx)
    line_img = hough_lines(roi_edges, rho, theta, threshold, min_line_length, max_line_gap)
    res_img = cv2.addWeighted(img, 0.8, line_img, 1, 0)
 return res_img
```

将划线合成到视频中

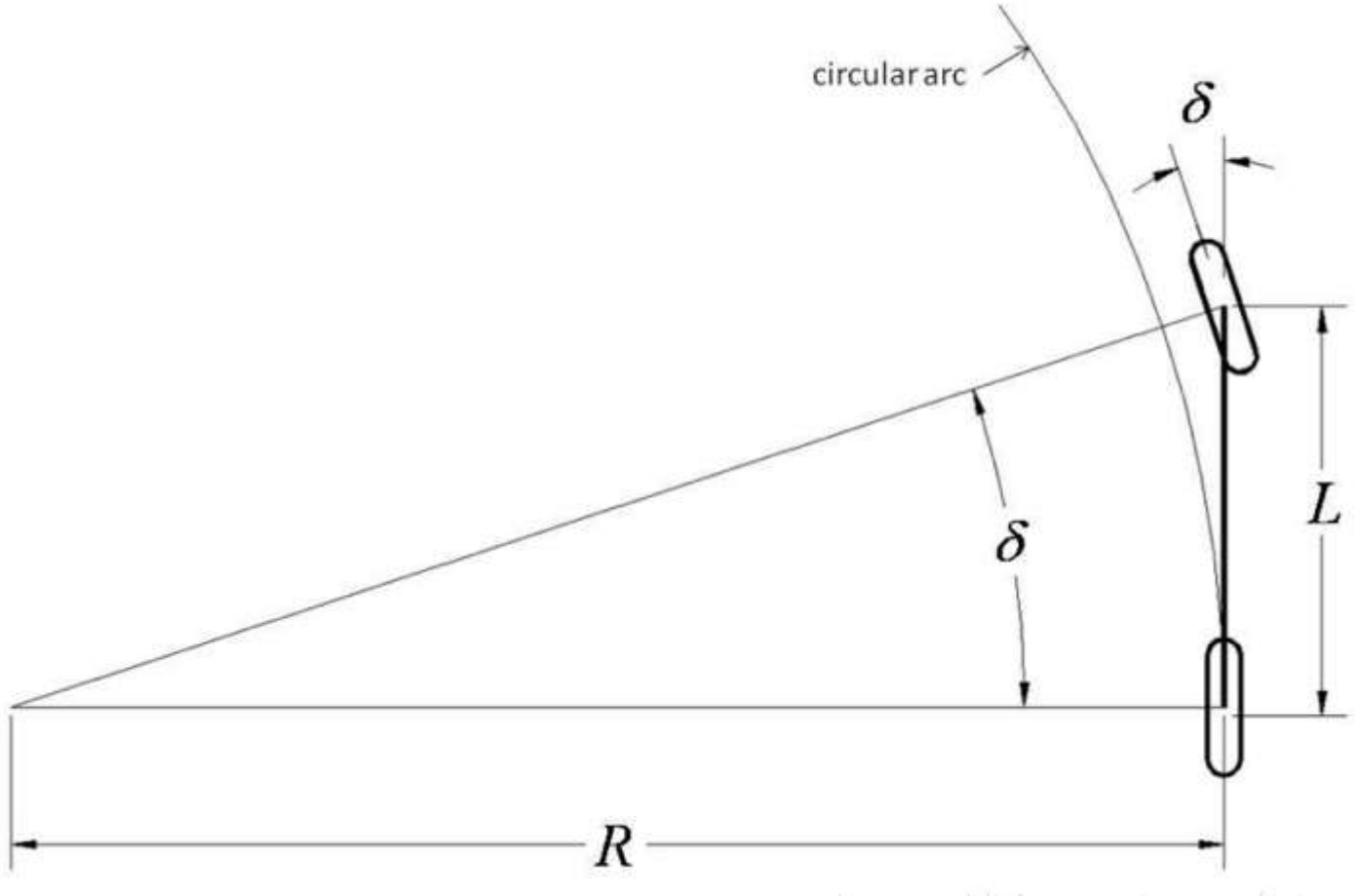
- **from** moviepy.editor **import** VideoFileClip
- output = 'video_1_sol.mp4'
- clip = VideoFileClip("video_1.mp4")
- out_clip = clip.fl_image(process_an_image)
- out_clip.write_videofile(output, audio=False)

- 注意：对于不同的情况，有些参数可能需要调节，主要是不同场景下参数和弯道线的参数调整

06

车辆跟踪模型

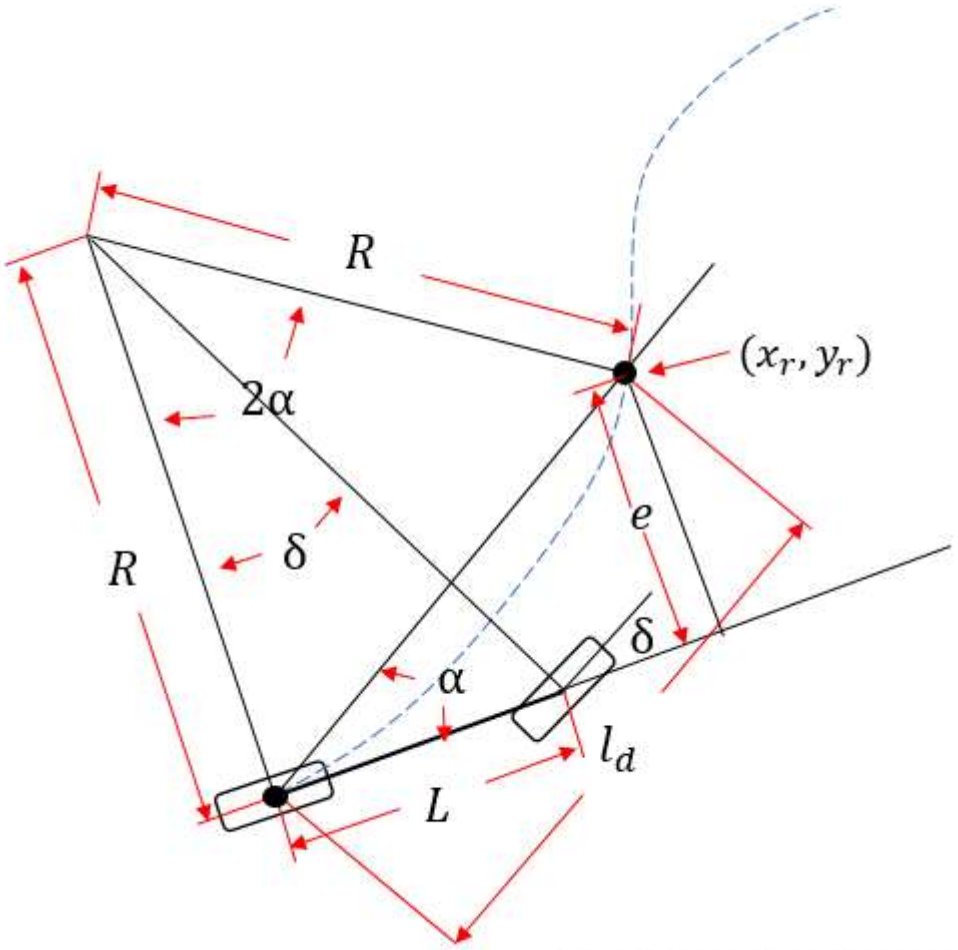
线性二自由度汽车模型



- L-轴距
- R-转弯半径
- δ-前轮转角

$$\tan(\delta) = \frac{L}{R}$$

纯几何跟踪法（pure pursuit）

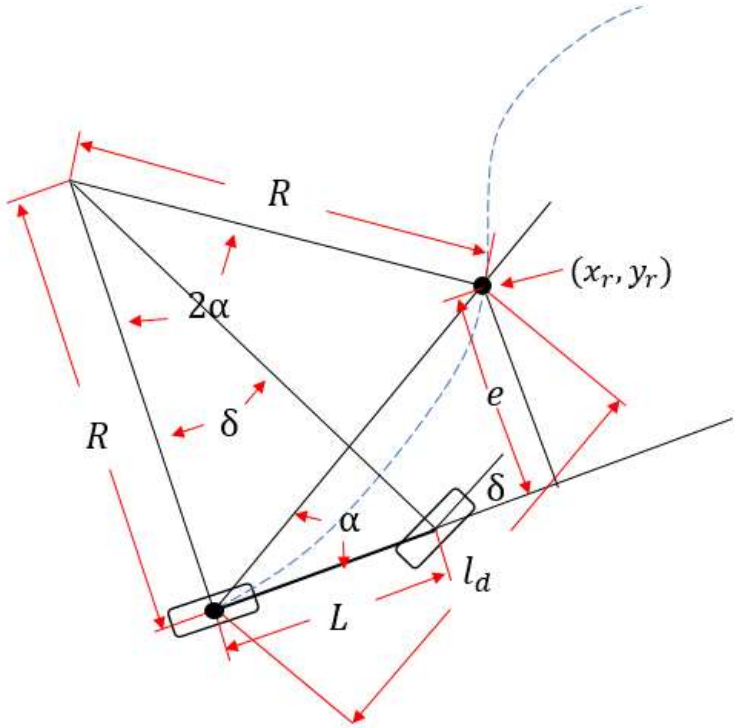


符号	物理量
R(m)	转弯半径
L(m)	轴距
δ(rad)	前轮转角
α(rad)	车身与预瞄点夹角
ld(m)	预瞄距离
e(m)	与预瞄点的横向偏差
xr(m)	预瞄点横坐标
yr(m)	预瞄点纵坐标

$$\delta = tan^{-1}(2Le/l_d^2)$$

纯几何跟踪法（pure pursuit)

推导过程



通过正弦定理可以推出：

$$l_d/(sin(2\alpha)) = R/(sin(\pi/2-\alpha))$$

即：

$$k = 1/R = 2sin\alpha/l_d$$

其中k为转弯圆弧的曲率。由上图还可推出：

$$tan(\delta) = L/R$$

$$sin\alpha = e/l_d$$

那么：

$$\delta = tan^{-1}(L/R) = tan^{-1}(kL) = tan^{-1}(2Lsin\alpha/l_d) = tan^{-1}(2Le/l_d^2)$$

纯几何跟踪法（pure pursuit)

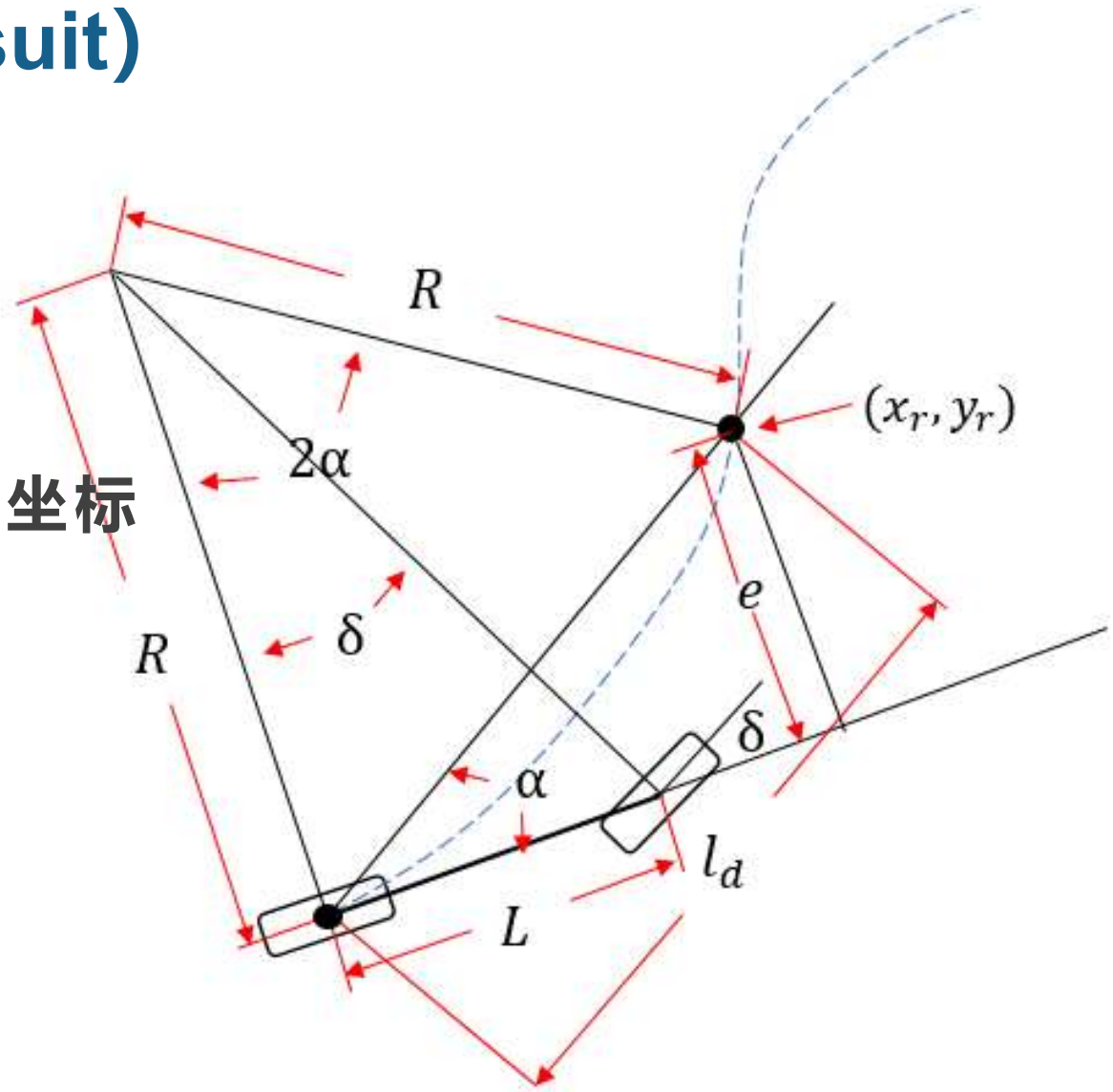
- 步骤

1.确定坐标系

2.确定目标点（goal point）坐标

3.计算e、ld

4. 利用公式计算δ



感谢您的关注

THANK YOU