智能小车自动驾驶与路标识别说明文档

小组成员：范传进、许家路、赵依洋、赵昊泽

参考资料：

第一部分：**全流程，从零搞懂基于PaddlePaddle的图像分割**

图像分割必备知识点 | Unet详解 理论+ 代码

图像实例分割评价指标

UAS数据集

第二部分：**【校园AI Day-AI workshop】交通信号标志图像分类**

PaddleDetection

PadlleX

中国交通标志数据集

三四部分：**人工智能引航计划**

最详细、最完整的相机标定讲解

透视变换（perspective transformation）

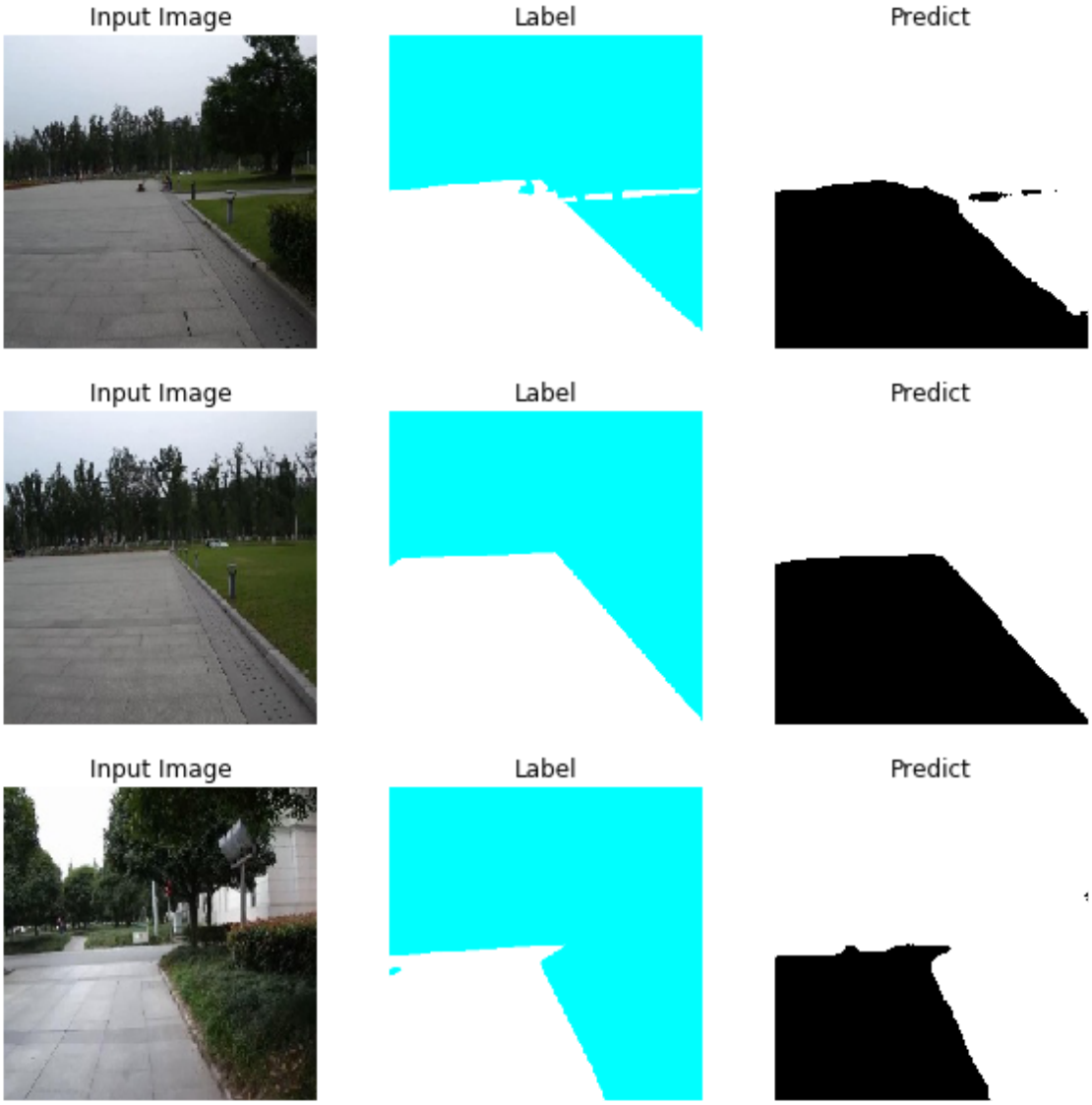
Introduction to the A* Algorithm

Algorithms for Automated Driving

一、基于Paddle和U-Net的自动驾驶道路分割

使用U-Net深度学习网络，在Paddle开源深度学习平台，实现**对图像道路、非道路部分的分割。**

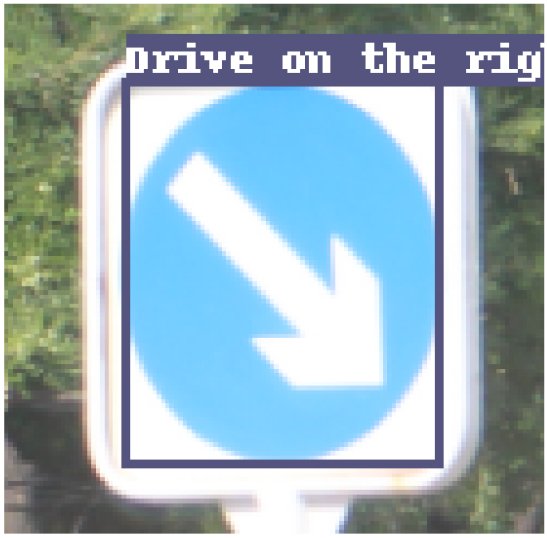
模型**PA=0.979, IoU=0.952**，分割效果较好，如下图所示，详情见**基于Paddle和U-Net的自动驾驶道路分割 - 飞桨AI Studio**



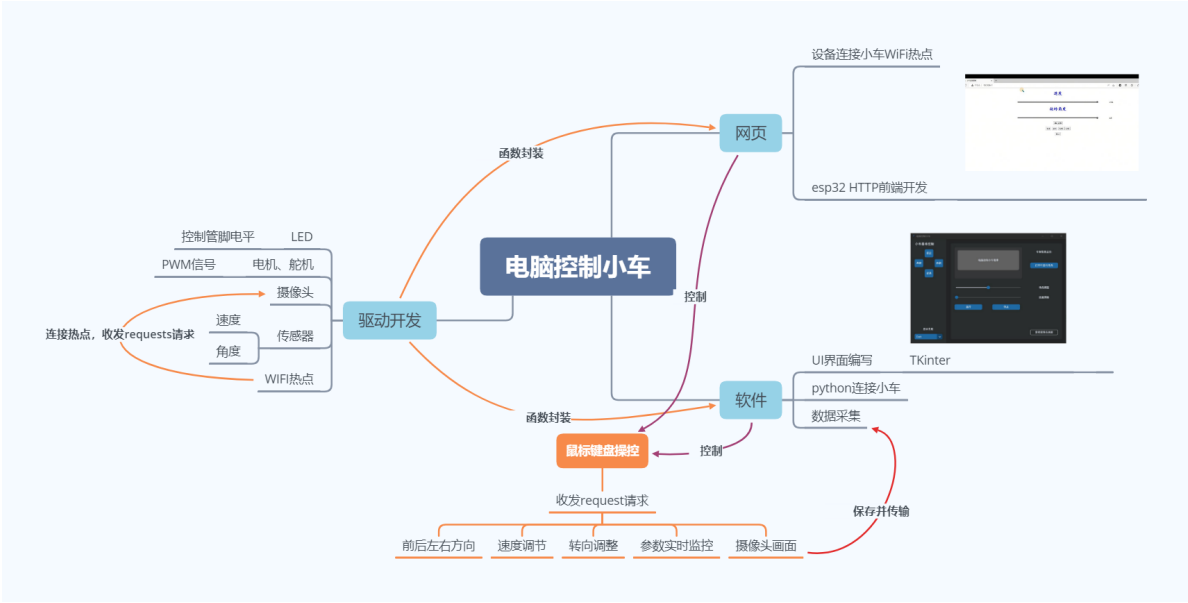
二、基于PP-YOLOv2的交通标志检测与识别

采用骨干网络为ResNet50_vd_ssld的PPYOLOv2算法，同样在Paddle开源深度学习平台训练，实现**对64类交通标志的识别。**

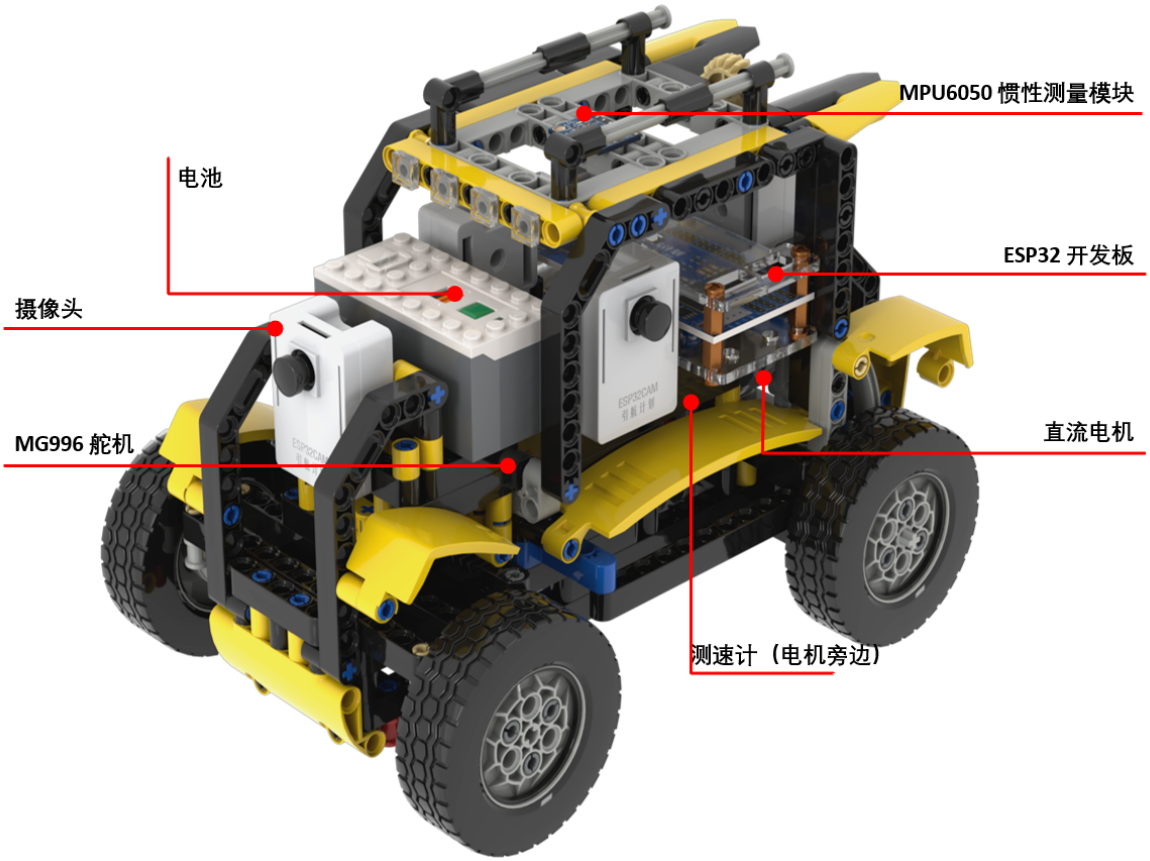
模型**mAP=96.27%**，识别精度较高，如下图所示，详情见**基于PP-YOLOv2的交通标志检测与识别 - 飞桨AI Studio)**



三、网页/软件控制小车运动



3.1驱动开发



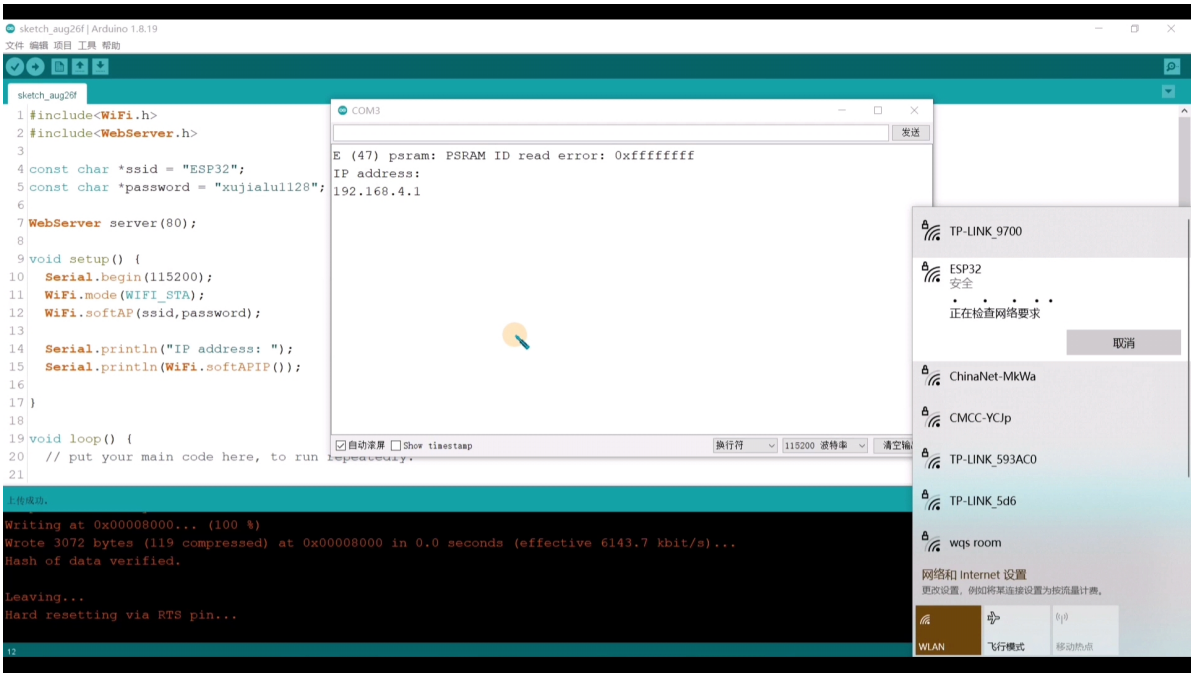
通过设置频率，信号通道，分辨率等数值，再将引脚和通道进行绑定，根据小车电机连接管脚，设置输出管脚与输入管脚，通过写入占空比的方式控制小车前进或后退，小车的左转右转需要在小车前进的过程中使用舵机调整角度，舵机同样通过写入pwm信号对角度进行控制，最终将小车前进、后退、左转、右转四个动作分别封装成一个函数以便后期调用。以下是控制小车舵机的例子：

```
const int d1 = 26;
const int d2 = 27;
const int d3 = 15;
const int freq = 3000;
const int freq_duo = 50;
const int resolution = 8;
const int channel1 = 0;
const int channel3 = 2;
void setup() {
  ledcSetup(channel1,freq,resolution);
  ledcAttachPin(d1,channel1);
  ledcSetup(channel3,freq_duo,resolution);
  ledcAttachPin(d3,channel3);


}

void loop() {
  pinMode(d2,OUTPUT);
  digitalWrite(d2,LOW);
  ledcWrite(channel1,255);
  ledcWrite(channel3,6.4);
}
```

通过导入WIFI和webServer两个头文件，调用相关函数将事先设置的WIFI名称和密码进行配置，通过串口输出IP地址以便下一步网站的实现。



3.2网页控制小车

通过编写前端代码实现前端网页以实现向小车端发送请求，利用server.send(),server.on(),server.begin()三个函数实现网站的部署以及小车端对于网页传来请求的接收，通过接收网站的请求，小车调用相关函数从而实现小车前进，后退，转弯等一系列操作以及对旋转角度运动速度的调整从而实现网页控制小车运动。以下是部分代码展示：

网页界面的展示：

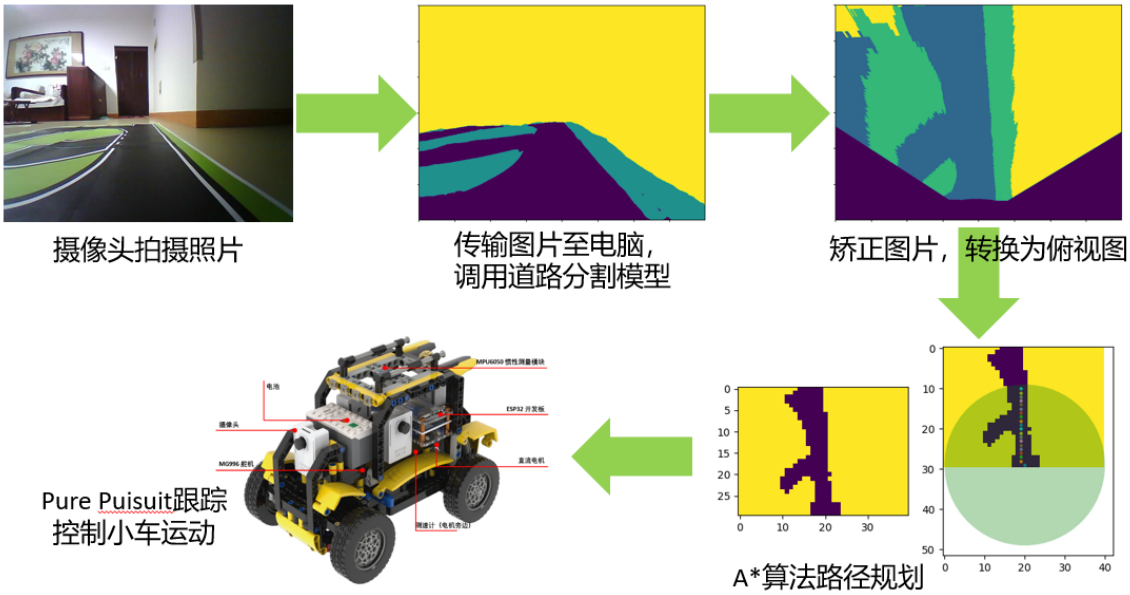
3.3程序控制小车

此外，我们使用Tkinter作为UI的开发工具，用CustomTKinter库作为设计辅助，开发了**基于Python用于小车控制的UI界面**，通过python的requests模块收发http请求，上述功能均可在其中实现，且可以进一步**封装为软件使用。**

```
import tkinter
import tkinter.messagebox
import customtkinter
import requests
```

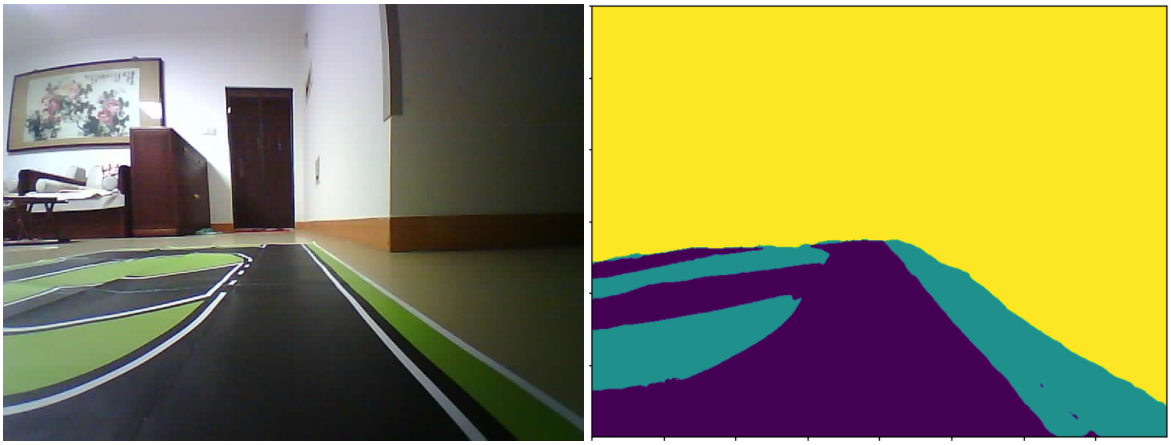
四、小车自动驾驶、识别交通标志

小车自动驾驶整体算法逻辑：



4.1.1 道路分割模型训练及调用

为了针对性地训练用于小车地图道路分割的模型，我们使用小车摄像头采集了271张小车地图的照片，并分为道路、草地、其他区域三个类别进行标注，仍使用第一部分的方法进行训练与调用。事实证明，使用这种数据集，模型精度可以显著提高。



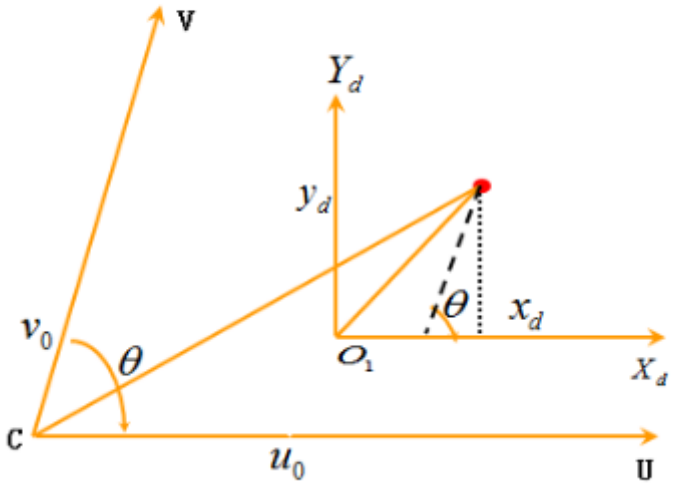
4.1.2 相机标定

4.1.2.1 相机内参的标定

在图像测量过程中，为确定空间物体表面某点的三维几何位置与其在图像中对应点之间的相互关系，必须建立摄像机成像的几何模型,这些几何模型参数就是摄像机参数。在大多数条件下这些参数必须通过实验与计算才能得到，这个求解参数的过程就称之为相机标定。简单来说是从世界坐标系换到图像坐标系的过程，也就是求最终的投影矩阵的过程。其标定的目的就是为了得到相机**内参**。

1）图像坐标系(x,y)至像素坐标系(u,v)

一般情况，两轴不互相垂直



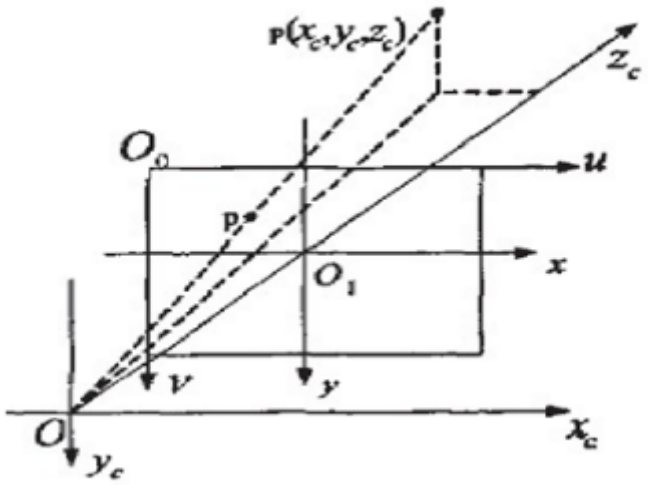
$$u = u_0 + \frac{x_d}{dx} - \frac{y_d \cot\theta}{dx}$$

$$v = v_0 + \frac{y_d}{dy \sin\theta}$$

写成矩阵形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & -f_u \cot\theta & u_0 \\ 0 & f_v/\sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}, \text{其中} f_u = \frac{1}{dx}, f_v = \frac{1}{dy}$$

2）相机坐标系(X_c, Y_c, Z_c)至图像坐标系(x, y)

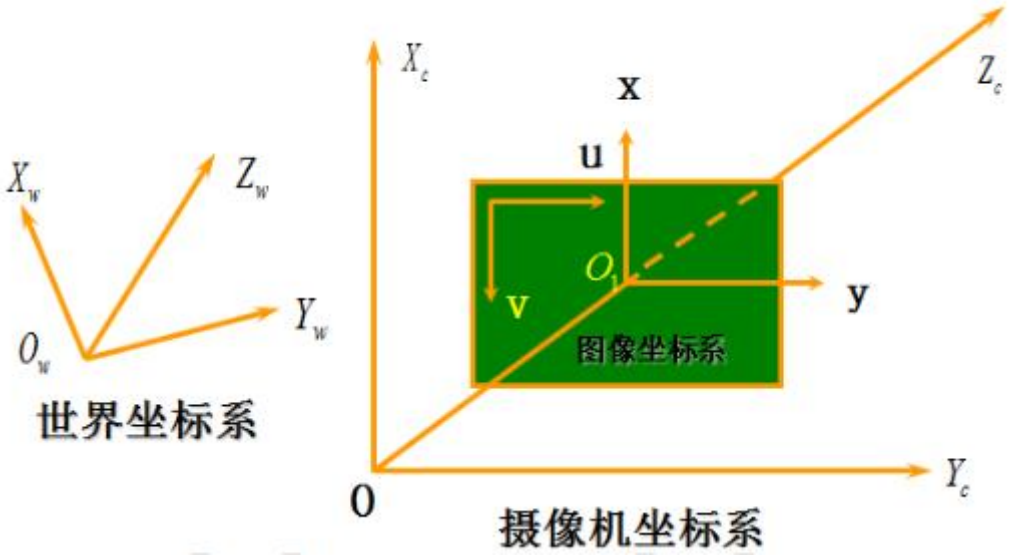


根据小孔成像原理，图像坐标系应在相机坐标系的另一边，为倒立反向成像，但为方便理解和计算，故投影至同侧。

根据三角形相似性原理得：

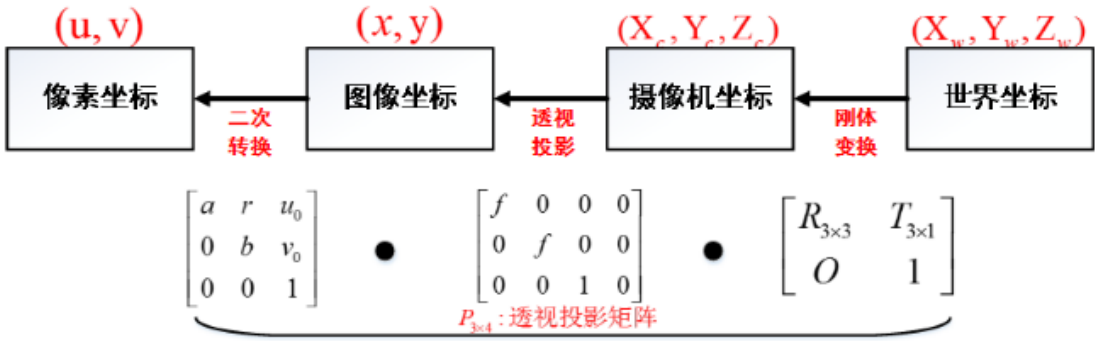
$$Z_c \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

3）世界坐标系(X_w, Y_w, Z_w)至相机坐标系(X_c, Y_c, Z_c)



$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3\times3} & T_{3\times1} \\ O & 1 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

4）合并公式



$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{f}{S_x} & r & u_0 \\ 0 & \frac{f}{S_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{3,3} & T_{3,1} \\ O & 1 \end{bmatrix} \begin{bmatrix} X_\pi \\ Y_\pi \\ Z_\pi \\ 1 \end{bmatrix} = K_{3,3} \begin{bmatrix} R_{3,3} & T_{3,1} \\ O & 1 \end{bmatrix} \begin{bmatrix} X_\pi \\ Y_\pi \\ Z_\pi \\ 1 \end{bmatrix}$$

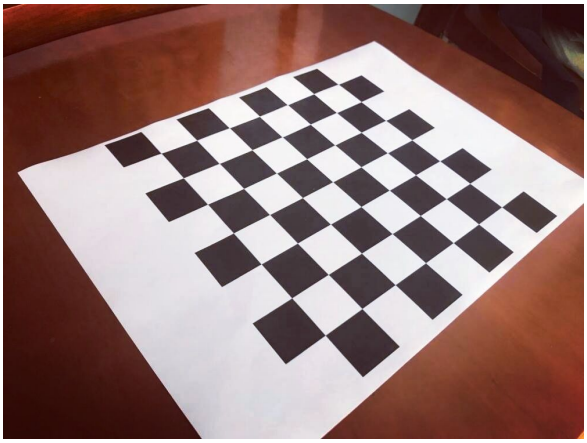
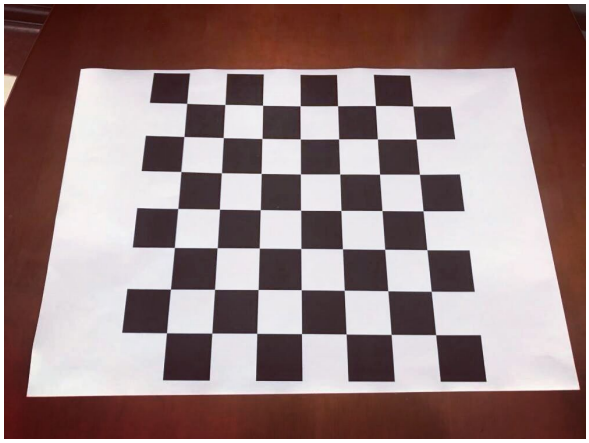
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P_{3,4} \begin{bmatrix} X_\pi \\ Y_\pi \\ Z_\pi \\ 1 \end{bmatrix}$$

其中，$K_{3,4}$：摄像机内参矩阵，$P_{3,4}$：透视摄影矩阵，$s = Z_\epsilon$：尺度因子

综上，**相机标定流程**：

1. 打印一张棋盘格A4纸张（黑白间距已知），并贴在一个平板上

2. 针对棋盘格拍摄30张图片
3. 在图片中检测特征点（Harris特征）
4. 利用解析解估算方法计算出5个内部参数

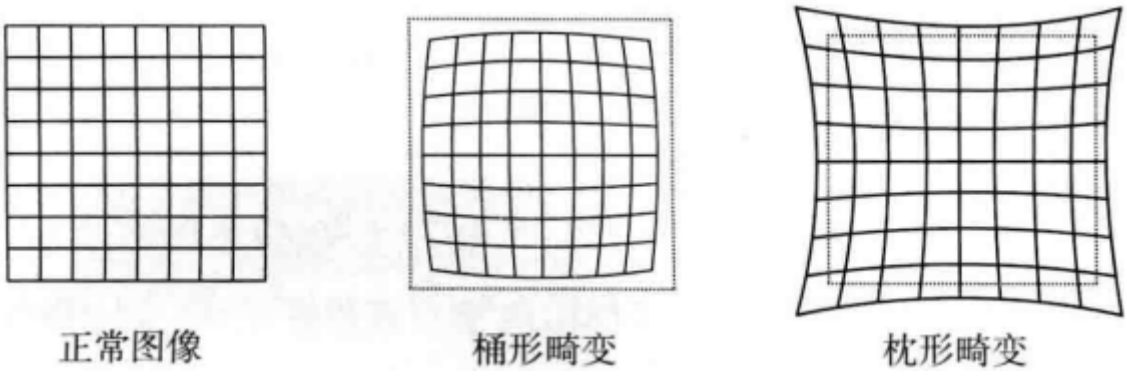



使用上述方法对小车相机进行标定后，得到相机内参为：

$$\begin{bmatrix} 252.8988 & 0 & 409.2385 & 0 \\ 0 & 252.9924 & 299.7036 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$，保存在internal.csv中

4.1.2.2 相机去畸变

实际的相机为了获得更好的成像效果，通常会在相机前方加入透镜，透镜的加入会使得光线传播受到影响，即真实世界的直线在图像中变成了曲线，这种叫径向畸变。径向畸变又分为桶形畸变和枕形畸变。



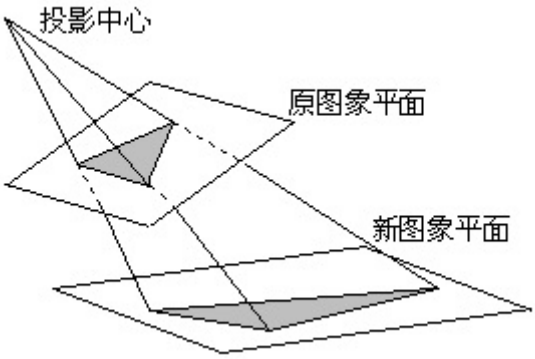
径向畸变可以看做坐标点沿长度方向发生了变化，即坐标点距离原点距离变了。通常使用的模型如下，假设畸变成多项式关系，使用4个参数k_1, k_2, k_3, k_4表达畸变：

$$\begin{aligned} x_{\text{distorted}} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + k_4 r^8\right) \\ y_{\text{distorted}} &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + k_4 r^8\right) \end{aligned}$$

据此得到小车摄像头的畸变参数：$k_1 : 0.6327, k_2 : 0.1601, k_1 : -0.4674, k_1 : 0.2126$，保存在distortion.csv中

4.1.3 透视变换

将相机拍摄图片的分割结果转换为俯视图，需要进行透视变换，透视变换通用的变换公式为：



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

u, v是原始图片左边，对应得到变换后的图片坐标x, y,其中$x = x^{'}/w^{'}, y = y^{'}/w^{'}$

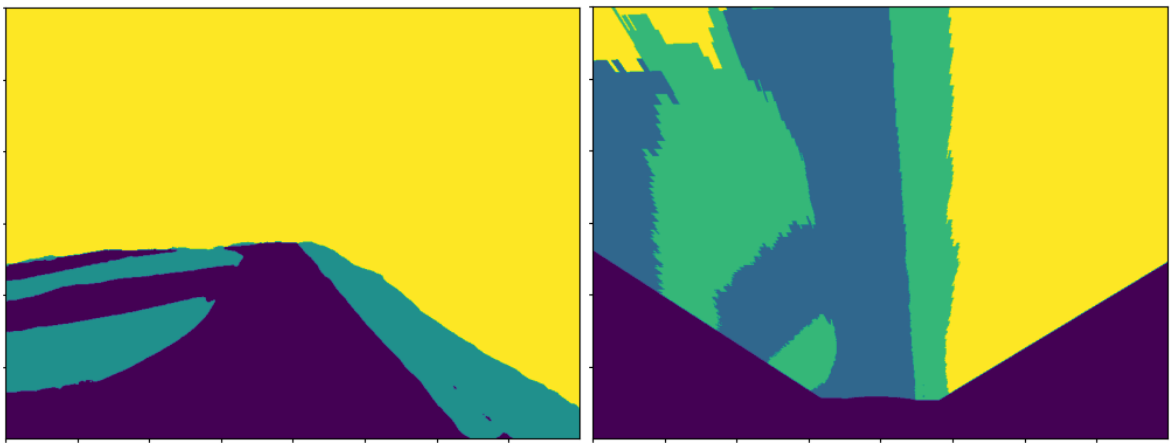
$$x = \frac{x'}{w'} = \frac{a_{11}u + a_{21}v + a_{31}}{a_{13}u + a_{23}v + a_{33}}$$
$$y = \frac{y'}{w'} = \frac{a_{12}u + a_{22}v + a_{32}}{a_{13}u + a_{23}v + a_{33}}$$

所以，已知变换对应的几个点就可以求取变换公式。反之，特定的变换公式也能新的变换后的图片。

由上述方法求出的透视变换参数为：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 0.0085319614 & -1.9624759357 & 647.3258107028 \\ -0.1877175581 & -1.3113553734 & 477.6317739588 \\ 0.0000231697 & -0.0032924414 & 1.0000000000 \end{bmatrix}$$

对4.1.1中图片进行透视变换，得到俯视图：



4.1.4 A*算法路径规划

使用A*算法根据俯视图进行路径规划

A*算法步骤：

1.从起点s开始，把s作为一个等待检查的方格，放入到"开启列表"中(“开启列表"就是一个存放等待检查方格的列表)

2.寻找起点s周围可以到达的方格(最多八个)，将它们放入到"开启列表”，并设置它们的父方格为s

3.从“开启列表”中删除起点s，并将s放入到“关闭列表”中(“关闭列表"存放的是不再需要检查的方格)

4.计算每个周围方格的F值

F=G+H:

G表示从起点A移动到指定方格的移动消耗，我们假设横向移动一个格子消耗10，斜向移动一个格子消耗14(具体值可以根据情况修改)

H表示从指定的方格移动到目标E点的预计消耗，我们假设H的计算方法，忽略障碍物，只可以纵横向计算

5.从"开启列表"中选择F值最低的方格a，将其从"开启列表"中删除，放入到“关闭列表"中

6.检查a所有临近并且可达的方格

a)障碍物和“关闭列表”中的方格不考虑

b)如果这些方格还不在“开启列表"中的话，将它们加入到“开启列表”，并且计算这些方格的F值，并设置父方格为a

c)如果某相邻的方格c已经在“开启列表”，计算新的路径从s到达方格c(即经过a的路径)

判断是否需要更新:G值是否更低一点

如果新的G值更低，则修改父方格为方格a，重新计算F值，H值不需要改变，因为方格到达目标点的预计消耗是固定的

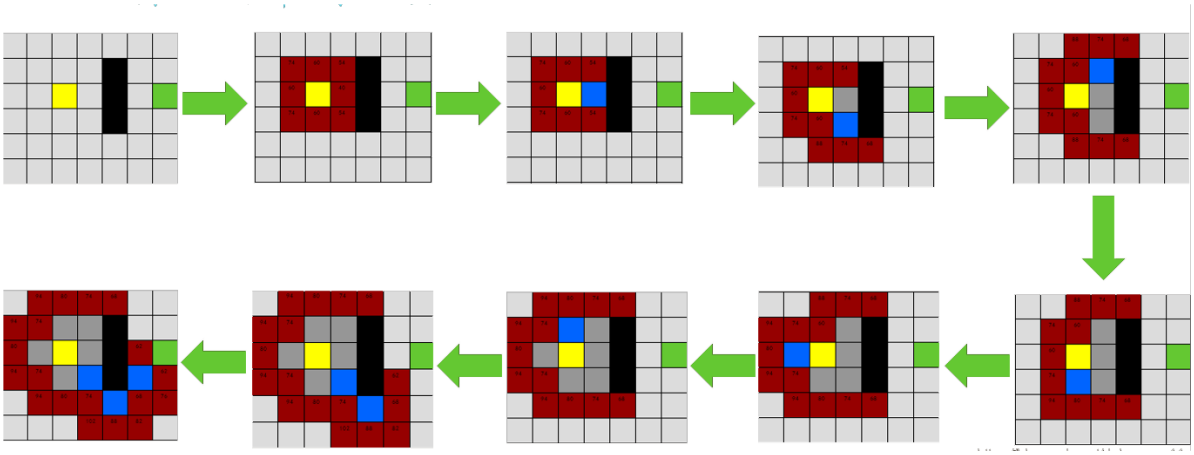
如果新的G值比较高，则说明新的路径消耗更高，则值不做改变（G值不变也不更新

7.继续从“开启列表"中找出F值最小的，从“开启列表"中删除，添加到“关闭列表”，再继续找出周围可以到达的方块，如此循环

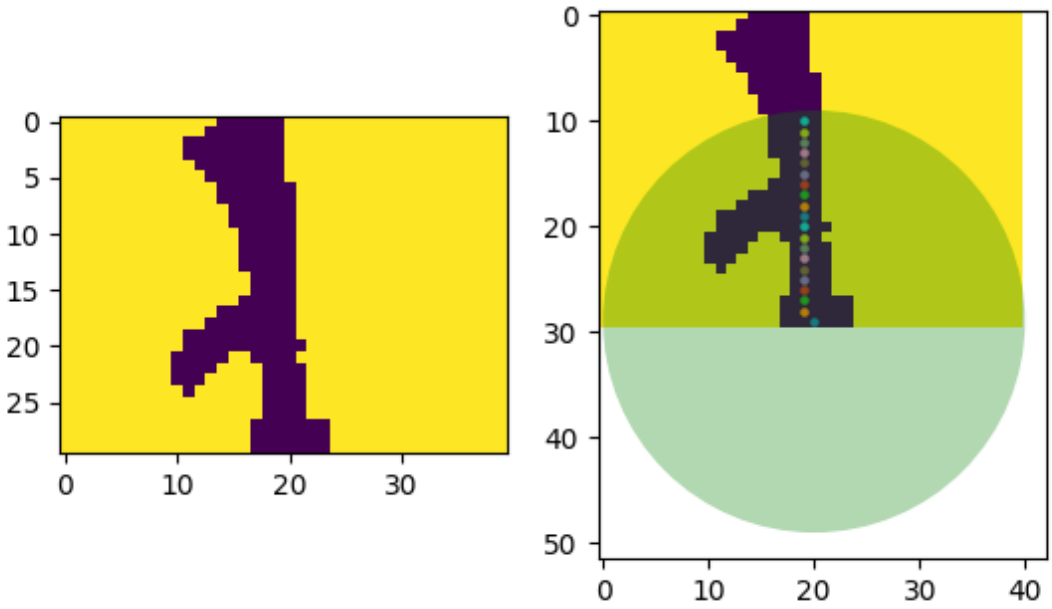
8.结束判断:

当“开启列表”中出现目标方块E时，说明路径已经找到

当“开启列表”中没有了数据，则说明没有合适路径



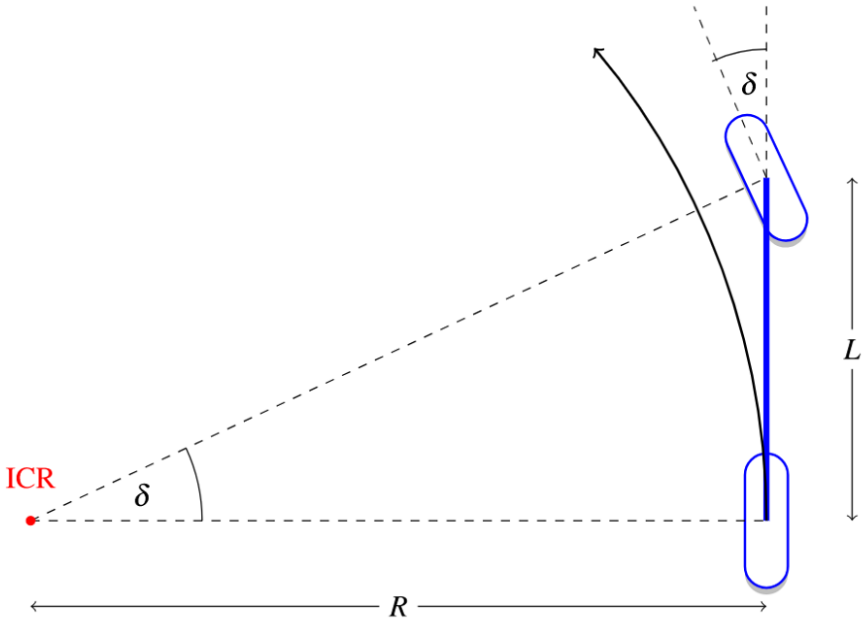
根据4.1.3中俯视图进行A*算法路径规划，得到行驶路径：



4.1.5 阿克曼转向几何与Pure Pursuit路径跟踪算法

4.1.5.1 阿克曼转向几何

阿克曼转向几何模型可简化为如下图所示的单车模型：



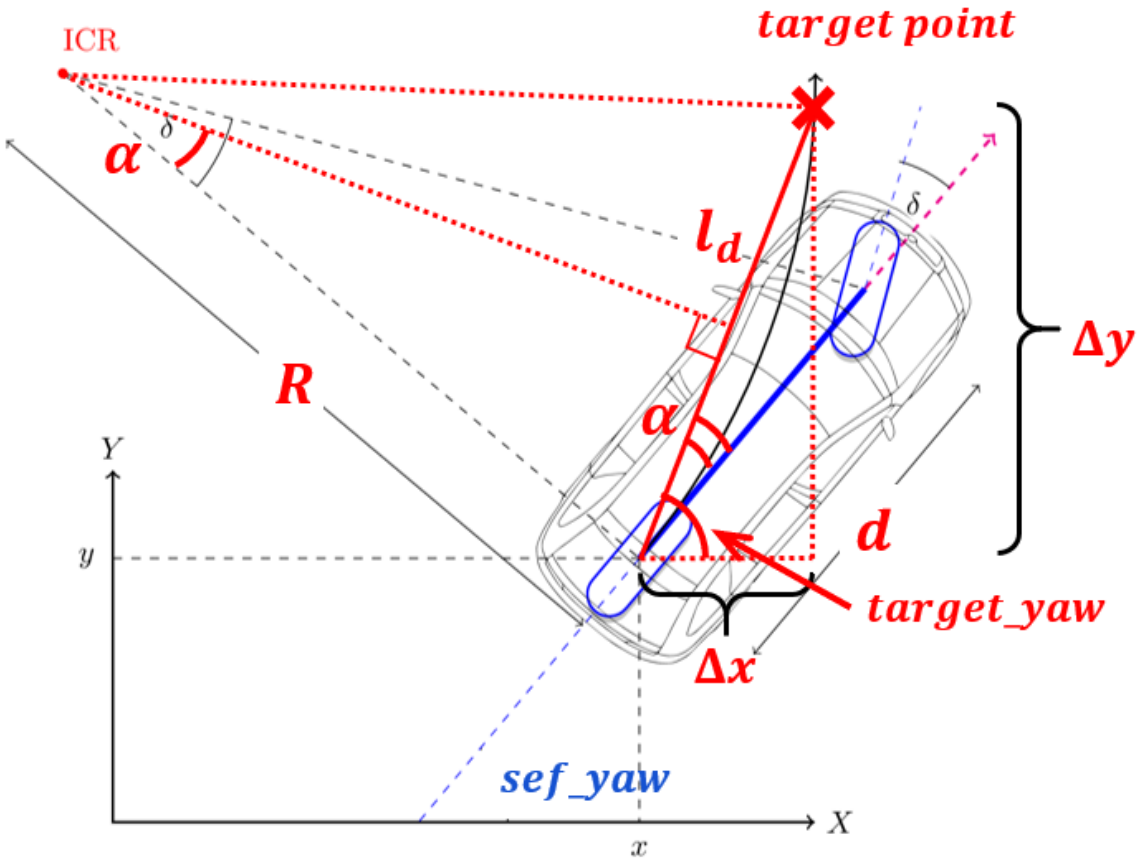
精确描述单车模型中几何关系的核心公式如下：

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

其中，δ为前轮转角，L为轴距，R为给定转向角运动时，无人车形成的轨迹圆的半径。

4.1.5.2 Pure Pursuit路径跟踪算法

Pure pursuit方法的依据是使如上图所示的**单车模型以合适的前轮转向δ运动，并恰好使无人车后轴中心经过当前的路点。** 这样一来，我们就可以**根据当前的路点以及单车几何模型计算当前的期望前轮转向角δ。**



根据上图，有以下几何关系：

$$\sin\alpha = \frac{l_d}{2R} \Rightarrow R = \frac{l_d}{2\sin\alpha}$$

进而有

$$\delta = arctan(\frac{2Lsin\alpha}{l_d})$$

其中，α为路点与车后轴连成的向量的角度与车航向角的差值，当路点在车的左边时，α > 0，反之则α < 0；l_d 为车后轴离路点的距离，又被称为**前视距离**。

而航偏角为

$$target_yaw = atan2(\Delta y, \Delta x)$$

故α可由下式求出：

$$\alpha = target_yaw - car_yaw$$

可由α推出R，进而由R推出δ，可据此控制小车运动。

4.1.6 MPU6050与计数器

MPU6050内部整合了三轴MEMS陀螺仪、三轴MEMS加速度计以及一个可扩展的数字运动处理器DMP。陀螺仪就是测角速度的，加速度传感器就是测角加速度的，二者数据通过算法就可以得到**PITCH、YAW、ROLL角**。

计数器可以计算一定时间内光被遮挡的次数，将计数器放在电机连接的齿轮处，齿轮中的间隙等间距分布，通过光被遮挡的次数即可测出小车速度。