项目总结

基于 ROS 系统的无人驾驶智能小车

- OpenCV 完成车道线识别：
  - 缩放图像：卷积神经网络中，对图像的Resize是必不可少的步骤，一般网络对图像的大小有特定的要求，比如YOLO系列的图像尺寸大小要为`608*608`或者`512*512`,所以我们要将大尺寸图像进行Resize后输入到网络模型中。

    问题1：图像Resize前后，是否应该保持宽高比例一致?

    答：两种均可，但是要保持模型的训练和推理时操作方式一致。保持宽高比例的做法一般是用增加padding的方式，然后用固定颜色填充，保证图像画面中部的内容不变形。通常一般推荐使用直接拉伸的方式去做图像Resize，原因是增加padding填充后会对网络带来一定噪音，影响模型准确性。

    问题2：图像Resize过程，应该选择什么样的插值方式?

    答：插值：对图像进行放大时要增加像素，缩小时要删除像素值。比如OpenCV中的INTER_LINER线性插值，我们只需要在推理和训练的时候保持一致即可。同时，Resize的次数要保持一致。

    ```
    cv2.resize(front_frame,(320,240),interpolation = cv2.INTER_AREA) # 将我们要检测图像进行一个缩放，减小处理数据量，加快处理
    ```

  - 畸变校正

    ```
    front_frame = cv2.undistort(front_frame, camera_params["intrinsicMat"], camera_params["distortionCoe"], None, camera_params["intrinsicMat"]) # 对相机进行畸变校正，还原图像的真实性。分为径向畸变校正和横向畸变校正
    ```

  - 透视变换

    ```
    perspective_img = PerspectiveTransformation(image) # 将图像投影到正视图，即将三维视图转换为二维视图
    ```

  - 平均基尔霍夫变换

    通过基尔霍夫变换找到图像中的线段，根据线段的长度进行一个权重加成，得到一条平均的基尔霍夫直线。

    ```
    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)
```

- OpenCV红绿灯识别
  - 获取感兴趣区域ROI

```
// 参数：左上角x坐标、y坐标、宽、高
image_ROI = cv_image(cv::Rect(520,100,120,150)); //采用基于区域的图像的分割
技术分割出指定的区域，简化了图像的处理
```

  - 轮廓预处理

```
//转化成hsv
cvtColor(image_ROI, hsv_image, CV_BGR2HSV);
//hsv阈值调整:调整绿灯的hsv阈值
inRange(hsv_image, cv::Scalar(69, 51, 147), cv::Scalar(157, 240, 255),
hsv_range_image);
threshold(hsv_range_image, binary_image, 127, 255, cv::THRESH_BINARY);
 //自适应二值化
// 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);
```

  - 轮廓检测

    根据轮廓面积的大小是否达到阈值来判断是否识别到红绿灯

```
// 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());
sort(contours.begin(),contours.end(),ContourArea);
if(0 == contours.size())
    return false;
//判断面积的大小
if(cv::contourArea(contours[0]) >= 2500)
    return true;
else
    return false;
```

- PID控制
  - 基于增量式的PID控制

    通过取20次PID的平均值作为要调节的值，来达到缓和的情况；同时防止PID的超调，将过于错误的PID的值进行去除或者减缓其值大小。

    增量式PID中不需要累加。控制增量Δu(k)的确定仅与**最近3次的采样值**有关，容易通过加权处理获得比较好的控制效果，并且在系统发生问题时，增量式不会严重影响系统的工作,适合于带积分的控制对象，有稳态误差。

    位置式PID是一种非递推式算法，可直接控制执行机构（如平衡小车），u(k)的值和执行机构的实际位置（如小车当前角度）是一一对应的，因此在执行机构不带积分部件的对象中可以很好应用

```
struct PID
{
    float target_value;
    float actual_value;
    float current_error;
    float last_error;
    float last_last_error;
    float kp;
    float ki;
    float kd;
}front_pid_control,rear_pid_control;

// 获取车道线的偏差
float change_value = 0.0f;
float car_speed = -25.0f,steering_angle = 0.0f;
//PID增量
front_pid_control.actual_value = msg->data.at(0);
for(int i =0; i< 20;i++)
{
    change_value += frontIncrementPID(320);
}
steering_angle = change_value / 20;

//visual_detect_flag收到1
if(visual_detect_flag == 1 && start_back == 0)
{
    // 正数往左，负数完右
    if(steering_angle < -5)
        steering_angle += -8;
```

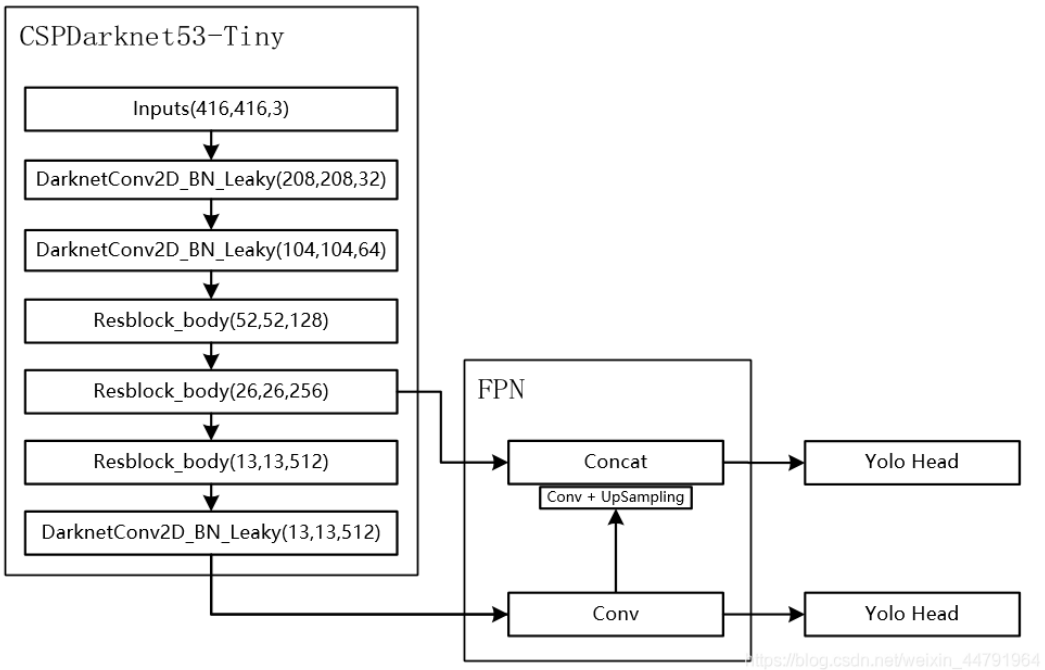
```
    if(steering_angle > 0)
        steering_angle += 8;
    ROS_INFO("car_speed:%f,steering_angle:%f",car_speed,steering_angle);
    // 识别到直行的话需要获取控制权，并且要调整偏转的角度
    if(straight_flag == 1)
    {
        // 防止过右
        if(steering_angle < -5)
            steering_angle = -5;
    }
    runCar(car_speed,steering_angle);
}
```

- 交通标志牌识别

  TensorRT 版本YOLOV4-tiny深度学习推理模型搭建,由于Jetson nano在普通YOLOv4-tiny上推理速度的局限性，检测的帧率为5帧左右，而使用tensorRT可以达到25FPS左右，达到实时检测。

  - YOLOV4-tiny

    架构：



    训练技巧：

    1. mosaic数据增强

       利用四张图片同时进行训练。对四张图片进行翻转、缩放、色域变换等

    2. Label Smoothing平滑

       原始的标签是0、1，在平滑后变成0.005(如果是二分类)、0.995。**对分类准确做了一点惩罚，让模型不可以分类的太准确，太准确容易过拟合。**

    3. CIOU

       将目标与anchor之间的距离，重叠率、尺度以及惩罚项都考虑进去，**使得目标框回归变得更加稳定，不会像IoU和GIoU一样出现训练过程中发散等问题。而惩罚因子把预测框长宽比拟合目标框的长宽比考虑进去。**

    4. 学习率余弦退火衰减

       学习率会先上升再下降，这是退火优化法的思想.上升的时候使用线性上升，下降的时候模拟cos函数下降。执行多次。

5. batch加速训练以及division分子模块进行训练

- 与激光雷达通信

  根据厂家编写的驱动程序，使用厂家提供的驱动接口，将激光雷达加入到ROS系统中。避障原理：分为停车避障、左转避障、右转避障。

  停车避障：当障碍物在小车正中央时停车

  左转避障：当障碍物在小车右侧，且距离大于左转安全距离时，左转

  右转避障：当障碍物在小车左侧，且距离大于右转安全距离时，右转

基于STM32+FPGA DAC 模块的高精度任意波形实现

一、STM32控制方波输出

1.1 使用立创 EDA 独立画出 STM32F103C8T6 最小系统的原理图、PCB 板、焊接、测试最小系统板

- 原理图绘制：
  - 电源部分：采用的是AMS1117稳压芯片
  - 晶振电路:STM32内部晶振精确度不高，我们可以使用外部晶振来提供准确的时钟，采用8MHz的晶体振荡器为高速时钟，为CPU提供准确的时钟频率；采用32.768kHz，**经过2的15次方分频后产生1秒的方波脉冲，用于计时。**
  - 复位和启动方式：将BOOT0通过串联一个电阻接地，默认从Flash启动。
  - 下载电路：建议使用JTAG的SWD模式下载
  - 外围IO口：合理地引出到开发板的两侧排针
- 绘制PCB图
  - 设计板子尺寸
  - 摆放元器件
  - 走线
    1. 电源主干线宽
    2. 晶振周围少走信号线或者用地线包围，以免造成干扰
    3. 走线不要靠近板子边缘
  - 铺铜：降低地线的阻抗，应该大面积铺地。
  - 丝印层

1.2 STM32控制DMA输出DAC

- 简介

  DMA为直接存储器访问，无需CPU直接控制传输，通过硬件为RAM何IO设备开辟一条直接传输的通路，使得CPU效率大大提高。

- DMA配置流程
  1. 使能DMA时钟
  2. 设置外设地址
  3. 设置存储器地址
  4. 设置传输数据量
  5. 配置数据流信息，启动传输
- 定时器
- 生成波形数据表
- DAC通过GPIO口来模拟高低电平的数据位，从而来控制输出

二. FPGA控制波形输出

- 时钟IP核确定FPGA时钟频率
- ROM IP核存储波形的数据
- UART串口通信控制波形使能禁止
- UART串口实现波形的动态变化
- ROM内数据通过数据总线（DAC模块的IO口）传输到DAC模块中。将输入的 ROM 数据发送至DA转换芯片的数据端口。
- DAC模块：3PD5651E芯片，125MHZ转换速率，数模转换位数为10位，最终输出的模拟电压范围是-5V~+5V。
- 波形动态可调：采用两点式生成波形对应的数据，对coe的数据进行读取，使用Qt做一个上位机软件，把数据点数均匀分布到coe文件的数据内。

基于FreeRTOS+Linux驱动的智能预警系统

- 创建室内烟雾、温湿度、障碍物测距、ESP8266蓝牙通信任务函数
- 移植uboot、linux内核,构建根文件系统，添加linux RTL8189FS wifi驱动到linux内核之中，移植wireless tools工具扫描wifi,移植并测试wpa_supplicant工具连接wifi
- 移植并测试QT库，编写TCP通信应用接收STM32发送的数据
    - socket IPC 可以使得在不同主机上的应用程序之间进行通信（网络通信），socket 是应用层与 TCP/IP 协议通信的中间软件抽象层，它是一组接口
    - bind->listen->accept->connect->read/recv 、write/send通信
    - bind->connect->send 、recv