项目介绍

1. **项目背景**

描述

ROBOWATCH 是全球最早将移动机器人和安全市场成功衔接起来的技术公司，开创了无人平台技术和遥控车辆技术在安全领域的应用先河，开创了德国服务机器人的技术路线和应用体系。

我们欢迎您加入 ROBOWATCH 上海大家庭，作为一名软件（实习）工程师。您将通过 2-3 个月时间对于无人驾驶有详细地了解，并负责选择一个开源的识别车道线的算法并实现功能。

2. **目的**

运用 C++，使用开源算法来实现简易车道线识别，锻炼无人驾驶图像处理能力。

3. **前期准备相关平台，软件，图片，视频**

平台：我本人再次选用 Ubuntu Linux14.04，由于之前做的项目是基于 ubuntu 平台，加以由于 OpenCV 安装已经完成故而作此选择。

软件：前文已经提起过，为实现图像处理功能我们需用到一个已经成型的图像处理库 OpenCV 3.4.6。

图片：在网络上搜寻并选择以下几张图片进行测试。在后续测试中发现图片中较多汽车会对测试结果造成一定干扰。

项目进程

1. 实现步骤一 ：使用 OpenCV 读取并显示图像/视频

图像读取显示:

```
Mat image = imread("line.jpg");
imshow("Test", image);
waitKey(0);
```

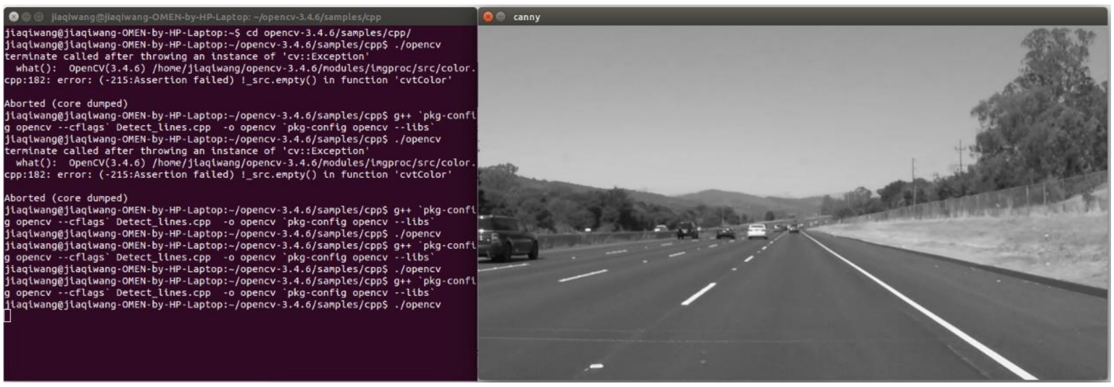
视频读取显示:

```
VideoCapture capture("solidWhiteRight.mp4");
while(1){
    Mat image;
    capture >> image;
    imshow("Test", image);
    waitKey(10);
}
```

2. 实现步骤二 ：图像灰度化 GRAY 处理降低色彩多样性以便更加容易识别白色

在此处动用颜色转化函数 cvtColor 以及 OpenCV 自带的 COLOR_BGR2GRAY 来实现图像灰度化的目的，具体代码如下:

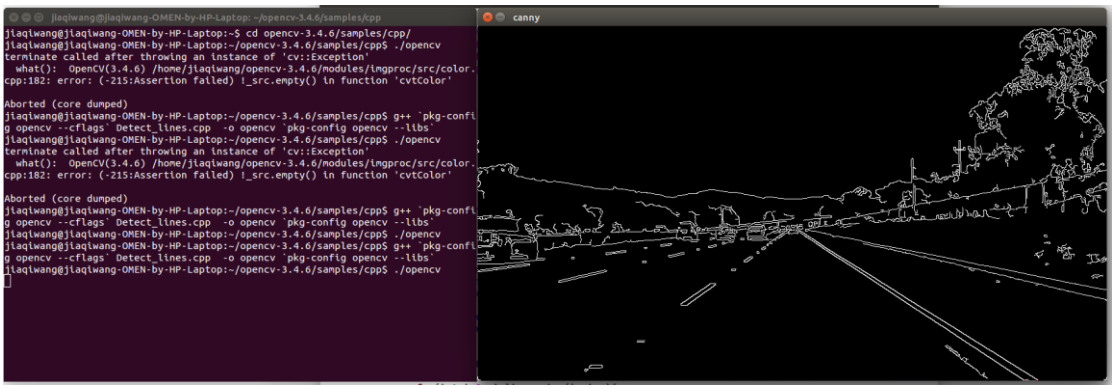
```
cvtColor(image, frame, COLOR_BGR2GRAY);
```



3. 实现步骤三：图像边缘化提取

通过 CANNY 函数以及设置好的高低阈值(min = 50, max = 140)来实现图片边缘的提取方便之后的处理，具体代码如下：

```
Mat frame,image;
int low_threshold = 40;
int high_threshold = 150;
Canny(frame,frame,low_threshold, high_threshold);
```



4. 实现步骤四 ： 图像感兴趣区域截取(中心下三角区域)

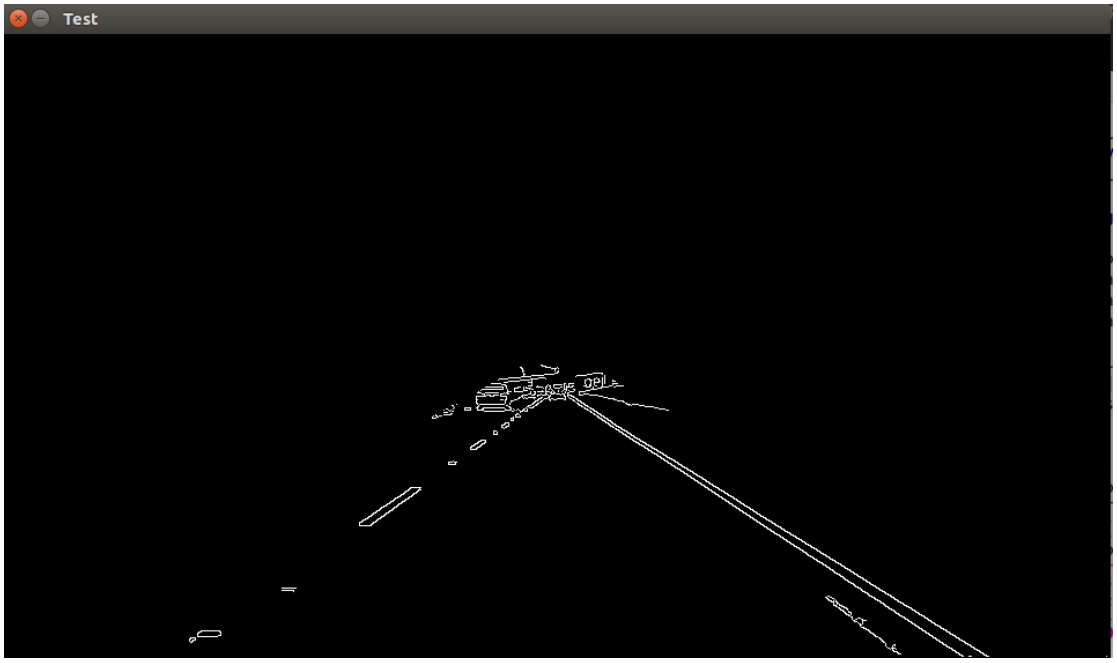
通过选取感兴趣区域来缩小处理的范围并出去当前车道以外的车道对处理做出的干扰，在此处我选用中心下三角区域因为其涵盖当前车道的完整图像。首先我们需要定义感兴趣区域的三个顶点坐标，此处为 (0, 540), (480, 270), (960, 540)，之后使用 polyLine 函数进行曲线拟合此处为直线，最后动用 polyfill 函数来黑化感兴趣区域外的图像，具体代码如下：

```
//确定定点
Point root_points[1][3];
root_points[0][0] = Point(0,539);
root_points[0][1] = Point(480,270);
root_points[0][2] = Point(959,539);

//拟合线段
const Point* ppt[1] = {root_points[0]};
int npt[] = {3};
polylines(frame, ppt, npt, 1, 1, Scalar(0,0,0),1,8,0);

//将感兴趣区域外全部黑化
Mat mask,dst,line_image;
frame.copyTo(mask);
```

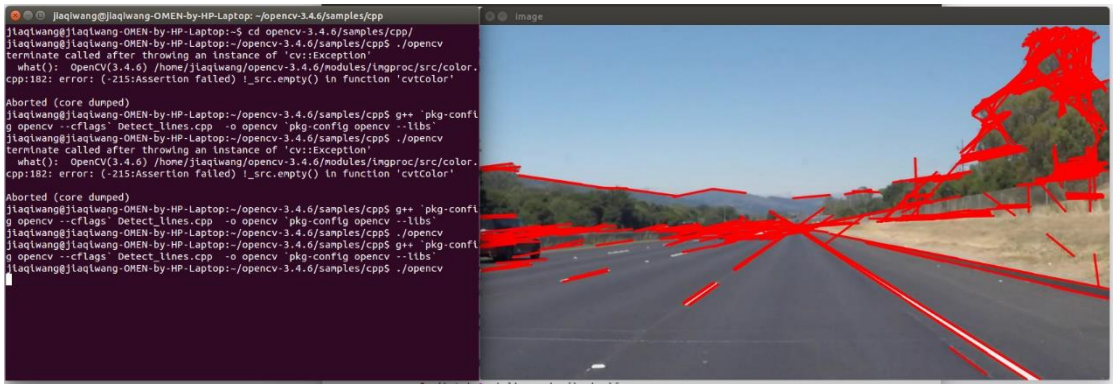
```
mask.setTo(cv::Scalar::all(0));
fillPoly(mask, ppt, npt, 1, Scalar(255,255,255));
frame.copyTo(dst,mask);
```



5. 实现步骤五 ：霍夫效应识别图像中直线

此处我选择动用霍夫效应来识别我们需要提取的特征：直线，具体代码如下：

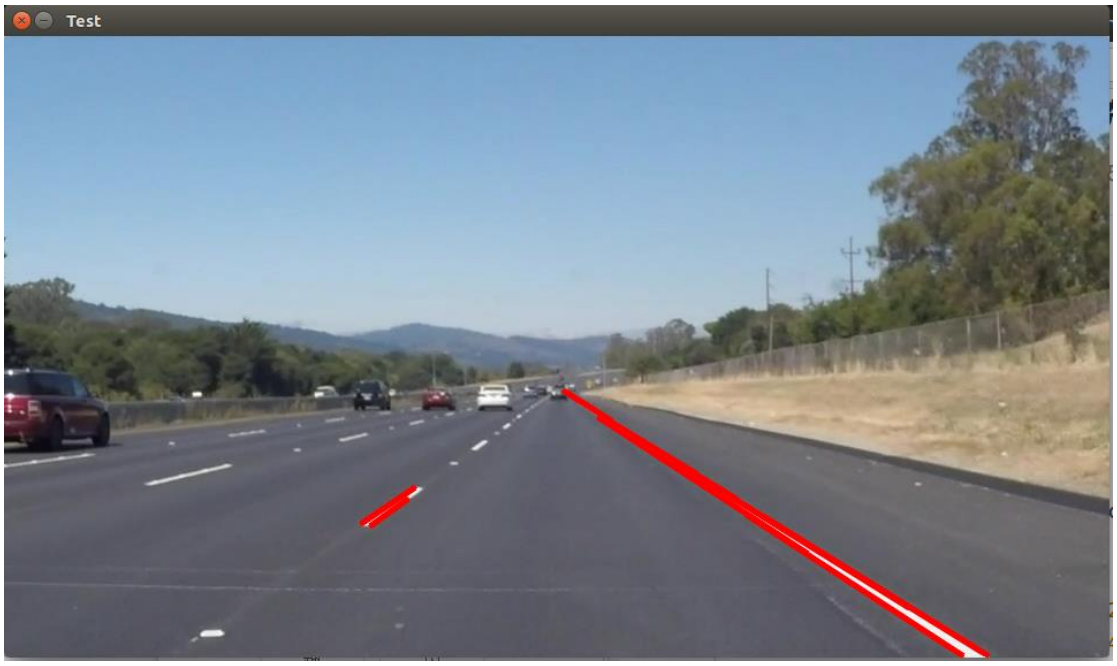
```
HoughLinesP(dst, lines, 1, CV_PI/180, 80, 20, 1);
```



6. 实现步骤六：霍夫效应识别感兴趣区域中车道线

基于进程 5 做出的图像中直线识别，在此进程我将直线提取只应用于感兴趣区域即下三角区域，具体代码如下：

```
//感兴趣区域内直线显示
        image.copyTo(line_image);
        for(int i=0; i<lines.size(); i++){
            Vec4i l = lines[i];
            line(line_image, Point(l[0],l[1]),
Point(l[2],l[3]), Scalar(0,0,255), 3 ,CV_AA);
        }
```



项目总结

1. 总代码

```
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;

//定义边缘化检测最大最小阈值
int low_threshold = 40;
int high_threshold = 150;

int main(int argc, const char **argv)
{

    //视频读取
    VideoCapture capture("solidWhiteRight.mp4");
    while(1){
        Mat frame,image;
        vector<Vec4i> lines;
        capture >> image;

        //读取图片
        //image = imread("line.jpg");

        //转化至灰度图
        cvtColor(image, frame, COLOR_BGR2GRAY);

        //边缘检测
        Canny(frame,frame,low_threshold, high_threshold);

        //定义感兴趣区域的三个定点坐标(图片中心下方三角形)，图片大小为
960*540，左上角坐标为(0,0)
        Point root_points[1][3];
        root_points[0][0] = Point(0,539);
        root_points[0][1] = Point(480,270);
        root_points[0][2] = Point(959,539);
```

```
        //拟合线段
        const Point* ppt[1] = {root_points[0]};
        int npt[] = {3};
        polylines(frame, ppt, npt, 1, 1, Scalar(0,0,0),1,8,0);

        //将感兴趣区域外全部黑化
        Mat mask,dst,line_image;
        frame.copyTo(mask);
        mask.setTo(cv::Scalar::all(0));
        fillPoly(mask, ppt, npt, 1, Scalar(255,255,255));
        frame.copyTo(dst,mask);

        //霍夫效应检测图片中直线
        HoughLinesP(dst, lines, 1, CV_PI/180, 80, 20, 1);

        //感兴趣区域内直线显示
        image.copyTo(line_image);
        for(int i=0; i<lines.size(); i++){
            Vec4i l = lines[i];
            line(line_image, Point(l[0],l[1]), Point(l[2],l[3]),
Scalar(0,0,255), 3 ,CV_AA);
        }
        imshow("Test", line_image);
        waitKey(10);
    }
    return 0;
}
```

2. 遗留问题

- 弯道检测
- 恶劣天气造成的检测影响
- 偏离当前车道中心点距离计算

以上问题皆可用高级车道线识别算法解决。

3. 项目应用

对于此项目的具体应用，以下仅凭个人猜想：

提取偏离中心点距离，结合 ROS (控制车辆轮胎) 以及 pid 控制器 (通过距离差值的大小来调节车辆左右调节速度，提高安全性) 来实现车辆行驶在车道线的正中心的目的。