计算机与网络学院

《机器视觉》课程设计报告

基于 OpenCV 的车道线与道路标示牌的实时检测

姓名 1：张朝中　　姓名 2：

学号 1：201641404106　　学号 2：

指导老师评语： 评定成绩：　　签名：　　日期：

目录

一、概述

1.1 课程设计的目的

据调查，道路交通事故中有 1/3 是由车辆变道或车辆偏离其正常行驶车道区域所导致的。美国联邦公路局的研究表明：如果可以获得车辆与车道之间的相对位置信息，则可以防止 53%左右的车道偏离事故，因此针对路面标线检测的研究是实现车道偏离警告系统的关键技术，它也是人工智能应用领域中一个很重要的组成模块，其对于实现车辆的完全自主驾驶具有深远的意义。

1.2 课程设计任务及要求

1) 利用 C/C++代码实现，用 git 进行版本管理，代码提交应该包含有开发日志（git 提交日志）。

2) 测试和验证视频样本由老师提供，也可自行采集，自行采集的应该包括至少两类目标：明显的道路线和一定数量的道路标识牌。

3) 提供比较友好的用户接口，可以由用户自行加载不同的视频。应该包含合适的输出界面，将结果呈现给用户。

4) 检测流程应该包括“道路预处理→车道线特征提取→车道线检测”

5) 检测后的结果应该能实时输出，例如：检测到的车道线实时与视频显示在同一窗口。

6) 实时检测出视频中的道路标识，从视频中把道路标识分割出来，并显示在窗口中。

1.3 开发该系统软件环境及使用的技术说明

运行环境：Win7 系统 64 位

运行平台：Qt Creator 4.5.1

Opencv 库：OpenCV_VERSION 3.2.0

相关技术：Qt 界面编程，Opencv 库函数的使用

二、系统设计的基本概念与原理

2.1 工作流程

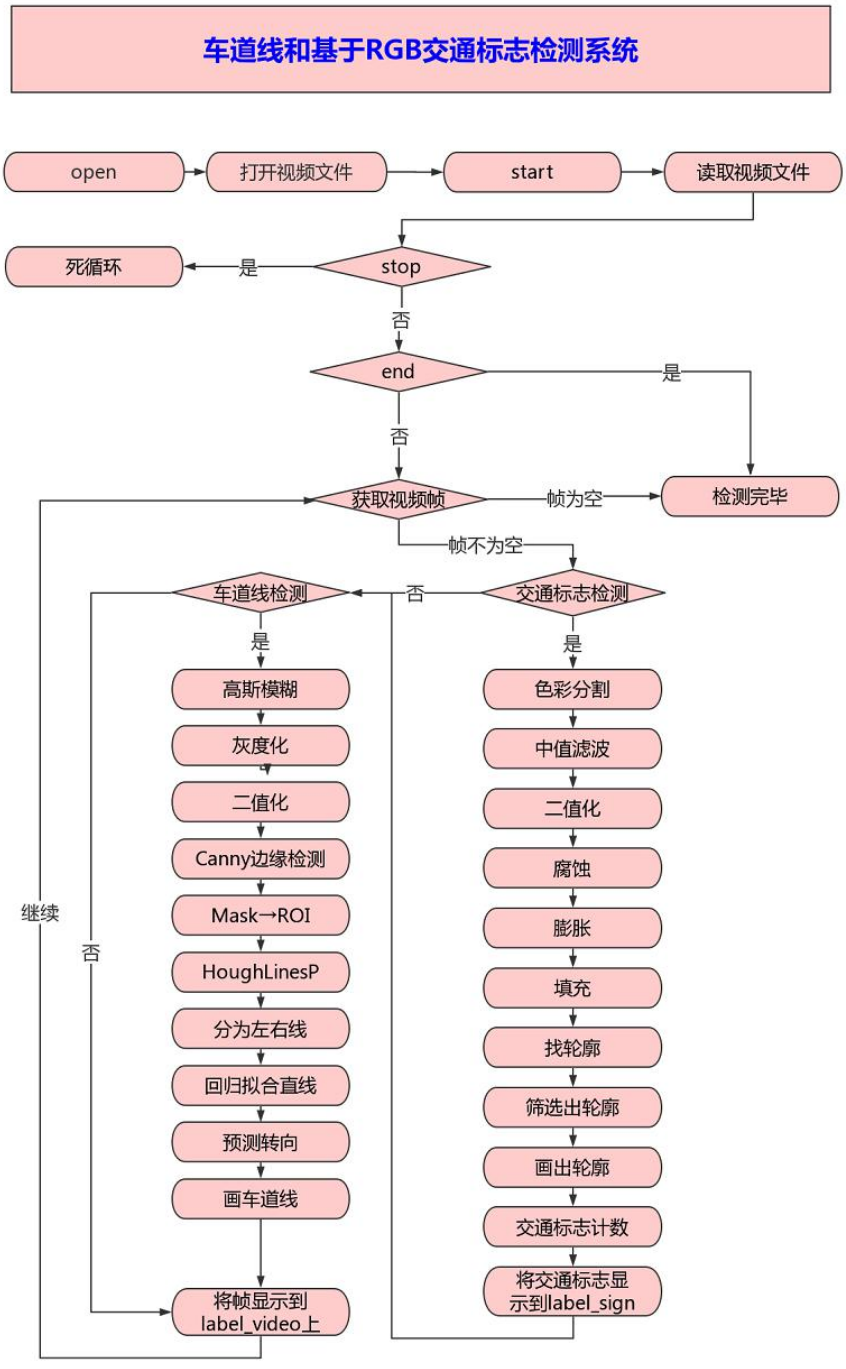

图 1 车道线和基于 RGB 的交通标志检测流程

在图 1 中，用户点击 open 按钮，打开想要检测的车道视频，再点击 start 按钮，开始进行检测，可以通过 lane，sign 复选框，来确定是否要检测车道线和交通标志，可以通过 stop 按钮来暂停当前检测(按 start 按钮可以恢复检测)，也可以按 end 结束当前检测。

2.2 车道线检测算法设计

图 2 车道线检测流程

在成功打开视频文件的条件下，开始进行车道线的检测。

在图 2 中，首先，读取视频的第一帧，判断是否为空，若没空则进行高斯模糊来降噪，再二值化，灰度化来进行边缘检测，通过 Mask 来确定感兴趣的检测区域，通过 Hough 变换查找直线，再将所有直线通过斜率，图像中心来分为左右线，紧接着，通过 fitLine 函数来拟合左右线得到左右线的初始点和终点，通过消失点(即左右线的交点)来预测转向，在帧上画上直线，最后将帧显示到 label_video 上。返回继续获取下一帧，循环以上的操作，直至帧为空，即为检测完毕。

2.3 道路标示牌检测算法设计

图 3 基于 RGB 的交通标志检测流程

在成功打开视频文件的条件下，开始进行车道线的检测。

在图 3 中，首先，读取视频的第一帧，判断是否为空，若不为空则进行色彩分割，将色彩分割后的图像进行中值滤波来降噪，二值化图像后，再进行腐蚀，膨胀，填充的形态学的处理，再进行轮廓的查找，进而筛选轮廓，在帧上画出轮廓，并记录交通标志的个数，最后将帧显示到 label_sign 上，并把截取的交通标志显示到对应的 label_sign 上。返回继续获取下一帧，循环以上的操作，直至帧为空，即为检测完毕。

2.4 用户接口设计

首先我想一下用户怎样操作，才会感觉简单，而且操作性强。我就先画了一下界面的草稿(如下图 4)。提供了打开视频文件（可以让用户自己选择视频），开始检测，暂停检测（在检测中，可以暂停，点击开始检测后，可以恢复检测），结束检测（用户可以在检测中，结束检测）等功能。

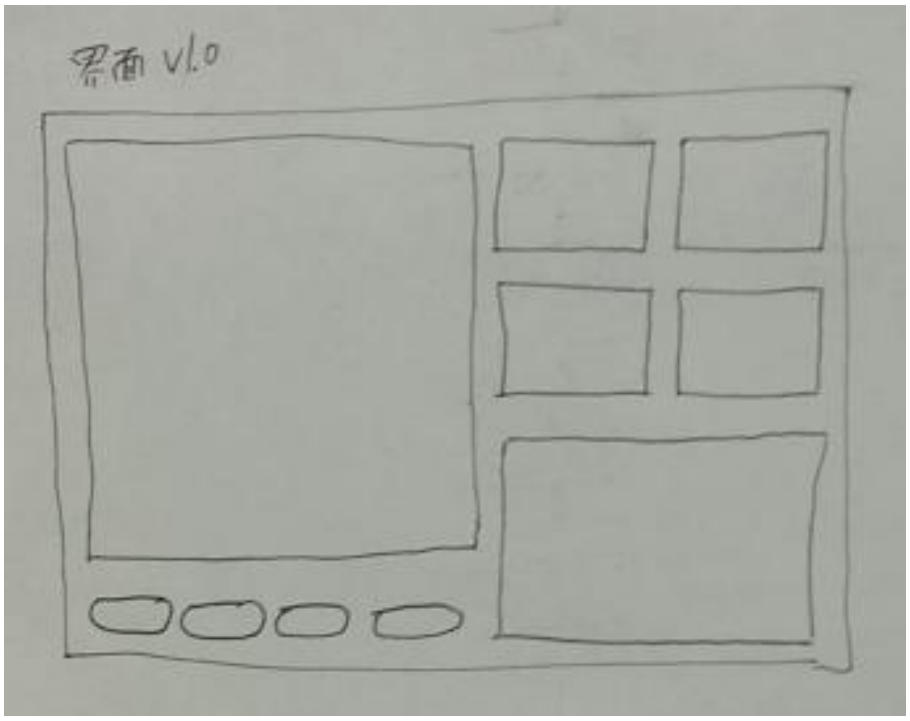

图 4 界面 v1.0 草图

然后再开始做界面，在实际做界面的过程中，发现不足，再进行完善。

界面 v1.0 如下图 5:

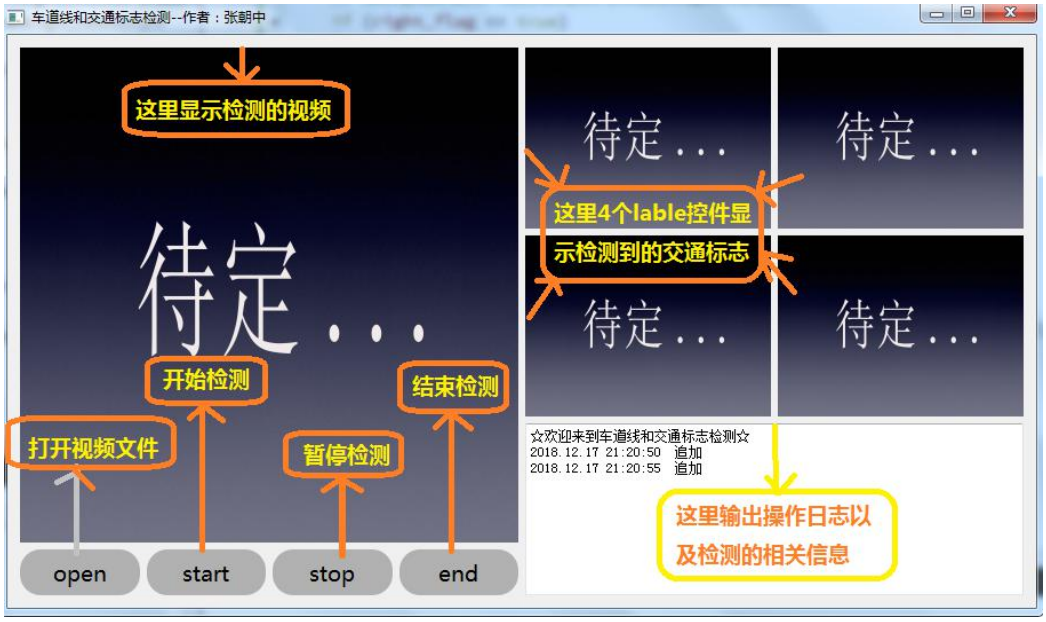

图 5 界面 v1.0

上面图 5 是进行车道线检测的界面，紧接着要增加交通标志的检测功能，于是，我添加了 2 个 Checkbox 来供用户是否要检测车道线或者交通标志，以及一个新按钮 showSign 来供用户查看系统检测到的所有交通标志。

于是有了如下界面 v2.0，如下图 6：

图 6 界面 v2.0

三、系统实施详细说明及结果

3.1 道路检测算法的实现

```
/**
*@功能 MASK 图像，以便仅检测构成通道一部分的边缘
*@参数 img_edges 是前一个函数的边缘图像
*@return output 仅表示所需边缘的二进制图像
*/
Mat LaneDetector::mask(Mat img_edges)
{
    Mat output;
    Mat mask = Mat::zeros(img_edges.size(), img_edges.type());

    // 自己调出感兴趣的位置
    Point pts[4] = {
        Point(80, 368),
        Point(280, 210),
        Point(320, 210),
        Point(640, 368)
    };

    // 创建二进制多边形 mask
    fillConvexPoly(mask, pts, 4, Scalar(255, 0, 0));

    // 将 img_edges 和 mask 相乘得到 output
    bitwise_and(img_edges, mask, output);

    return output;
}


/**
*@功能 按斜率对所有检测到的 Hough 线进行排序( 分左右线)
*@参数 lines 是包含所有检测到的 Hough 线的向量
*@参数 img_edges 用于确定图像中心
*@return output 包含所有左右线的向量
*/
std::vector<std::vector<Vec4i> >
LaneDetector::lineSeparation(std::vector<Vec4i> lines, Mat img_edges)
```

```
{
    std::vector<std::vector<Vec4i> > output(2);
    size_t j = 0;
    Point ini;
    Point fini;
    double slope_thresh = 0.3;
    std::vector<double> slopes;
    std::vector<Vec4i> selected_lines;
    std::vector<Vec4i> right_lines, left_lines;

    // 计算所有检测到的线的斜率
    for (auto i : lines)
    {
        ini = Point(i[0], i[1]);
        fini = Point(i[2], i[3]);

        // slope = (y1 - y0)/(x1 - x0)
        double slope = (static_cast<double>(fini.y) -
static_cast<double>(ini.y)) / (static_cast<double>(fini.x) -
static_cast<double>(ini.x) + 0.00001);

        // 如果斜率太水平，则丢弃该线
        // 否则，保存它们的斜率
        if (std::abs(slope) > slope_thresh)
        {
            slopes.push_back(slope);
            selected_lines.push_back(i);
        }
    }

    // 将线条分成右线和左线
    m_image_center = static_cast<double>((img_edges.cols / 2));
    while (j < selected_lines.size()) {
        ini = Point(selected_lines[j][0], selected_lines[j][1]);
        fini = Point(selected_lines[j][2], selected_lines[j][3]);

        // 将线分类为左侧或右侧的条件
        if (slopes[j] > 0 && fini.x > m_image_center && ini.x > m_image_center)
        {
            right_lines.push_back(selected_lines[j]);
            m_right_flag = true;
```

```
        }
        else if (slopes[j] < 0 && fini.x < m_image_center && ini.x <
m_image_center)
        {
            left_lines.push_back(selected_lines[j]);
            m_left_flag = true;
        }
        j++;
    }
    //右边线
    output[0] = right_lines;
    //左边线
    output[1] = left_lines;
    return output;
}


/**
*@功能 回归采用所有分类线段的初始点和最终点，并使用最小二乘法从它们中拟合新
线。
*@参数 left_right_lines 是 lineSeparation 函数的输出
*@参数 inputImage 用于获取行数
*@return output 包含两个车道边界线的初始点和最终点
*/
std::vector<Point> LaneDetector::regression(std::vector<std::vector<Vec4i> >
left_right_lines, Mat inputImage)
{
    std::vector<Point> output(4);
    Point ini;
    Point fini;
    Point ini2;
    Point fini2;
    Vec4d right_line;
    Vec4d left_line;
    std::vector<Point> right_pts;
    std::vector<Point> left_pts;

    // 如果检测到右线，则使用线的所有初始点和最终点拟合线
    if (m_right_flag == true)
    {
        for (auto i : left_right_lines[0])
```

```
    {
        ini = Point(i[0], i[1]);
        fini = Point(i[2], i[3]);

        right_pts.push_back(ini);
        right_pts.push_back(fini);
    }

    if (right_pts.size() > 0)
    {
        //  直线拟合函数(组成右边线)
        fitLine(right_pts, right_line, CV_DIST_L2, 0, 0.01, 0.01);
        m_right_m = right_line[1] / right_line[0];
        m_right_b = Point(right_line[2], right_line[3]);
    }
}


// 如果检测到左线，则使用线的所有初始点和最终点拟合一条线
if (m_left_flag == true)
{
    for (auto j : left_right_lines[1])
    {
        ini2 = Point(j[0], j[1]);
        fini2 = Point(j[2], j[3]);

        left_pts.push_back(ini2);
        left_pts.push_back(fini2);
    }

    if (left_pts.size() > 0)
    {
        //  直线拟合函数(组成左边线)
        fitLine(left_pts, left_line, CV_DIST_L2, 0, 0.01, 0.01);
        m_left_m = left_line[1] / left_line[0];
        m_left_b = Point(left_line[2], left_line[3]);
    }
}

// 获得了一个斜率和偏移点，应用线方程来获得直线的初始和终点
int ini_y = inputImage.rows;
```

```
    //与刚兴趣的区域相同
    int fin_y = 210;

    double right_ini_x = ((ini_y - m_right_b.y) / m_right_m) + m_right_b.x;
    double right_fin_x = ((fin_y - m_right_b.y) / m_right_m) + m_right_b.x;

    double left_ini_x = ((ini_y - m_left_b.y) / m_left_m) + m_left_b.x;
    double left_fin_x = ((fin_y - m_left_b.y) / m_left_m) + m_left_b.x;

    output[0] = Point(right_ini_x, ini_y);
    output[1] = Point(right_fin_x, fin_y);
    output[2] = Point(left_ini_x, ini_y);
    output[3] = Point(left_fin_x, fin_y);

    return output;
}


/**
*@功能 预测车道是左转，右转还是直线
*@功能 通过消失点相对于图像中心的位置来完成
*@return output 字符串，表示左转或右转或直线
*/
std::string LaneDetector::predictTurn()
{
    std::string output;
    double vanish_x;
    double thr_vp = 10;

    // 消失点是两条车道边界线相交的点  m1x+b1-y1 = m2x+b2-y2
    vanish_x = static_cast<double>(((m_right_m*m_right_b.x) -
(m_left_m*m_left_b.x) - m_right_b.y + m_left_b.y) / (m_right_m - m_left_m));

    // 消失点位置决定了道路转弯的位置
    if (vanish_x < (m_image_center - thr_vp))
        output = "Left Turn";
    else if (vanish_x >(m_image_center + thr_vp))
        output = "Right Turn";
    else if (vanish_x >= (m_image_center - thr_vp) && vanish_x <=
(m_image_center + thr_vp))
        output = "Straight";
```

```
    return output;
}
```

3.2 道路标示牌检测算法实现

```
/**
*@功能 色彩分割
*@参数 src 源图像
*@return matRgb 色彩分割的图像
*/
Mat SignDetector::colorSegmentation(Mat src)
{
    int width = src.cols;//图像宽度
    int height = src.rows;//图像高度
    double B = 0.0, G = 0.0, R = 0.0;
    //获取 matRgb
    Mat matRgb = Mat::zeros(src.size(), CV_8UC1);
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            // 获取 BGR 值
            B = src.at<Vec3b>(y, x)[0];
            G = src.at<Vec3b>(y, x)[1];
            R = src.at<Vec3b>(y, x)[2];

            //红色
            if (R - G > 50 && R - B > 50 )   // && B<50
            {
                matRgb.at<uchar>(y, x) = 255;
                continue;
            }
            //绿色
            if (G - R > 50 && G -B > 30)    //&& R < 100
            {
                matRgb.at<uchar>(y, x) = 255;
                continue;
            }
            //蓝色
```

```
            if (B - G > 50 && B - R > 50)   //&& G < 100
            {
                matRgb.at<uchar>(y, x) = 255;
                continue;
            }
            //黄色
            if (G - B > 50 && R - B > 50)
            {
                matRgb.at<uchar>(y, x) = 255;
                continue;
            }
        }
    }
    return matRgb;
}


/**
*@功能 寻找轮廓
*@参数 src 源图像
*@参数 contours 轮廓点集的最小矩形
* @return boundRect 包围
*/
vector<Rect> SignDetector::myfindContours(Mat src, vector<vector<Point>>
&contours)
{
    vector<Vec4i> hierarchy;        //分层
    findContours(src, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));//寻找轮廓
    vector<vector<Point>> contours_poly(contours.size());  //近似后的轮廓点集
    vector<Rect> boundRect(contours.size());  //包围点集的最小矩形 vector

    //获取包围轮廓的最小矩形
    for (int i = 0; i < contours.size(); i++)
    {
        approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true); //对多边形
曲线做适当近似，contours_poly[i]是输出的近似点集
        boundRect[i] = boundingRect(Mat(contours_poly[i])); //计算并返回包围
轮廓点集的最小矩形
    }
    return boundRect;
```

```
}


/**
*@功能 筛选轮廓
*/
for (int i = 0; i < contours.size(); i++)
{
    Rect rect = boundRect[i];

    //若轮廓矩形内部还包含着矩形，则将被包含的小矩形取消
    bool inside = false;
    for (int j = 0; j < contours.size(); j++)
    {
        Rect t = boundRect[j];
        if (rect == t)
            continue;
        else if (signdetector.isInside(rect, t))      //是否有交集
        {
            inside = true;
            break;
        }
    }
    if (inside)         //矩形 t 在矩形 rect 中，忽略
        continue;

    //高宽比限制
    float ratio = (float)rect.width / (float)rect.height;
    //轮廓面积限制
    //float Area = (float)rect.width * (float)rect.height;
    float dConArea = (float)contourArea(contours[i]);
    //float dConLen = (float)arcLength(contours[i], 1);
    if (dConArea < 600)
        continue;
    if(ratio > 3)
        continue;
}


/**
*@功能 交通标志的计数
```

```
*
*/
//cur_count 不为 0，则有检测到
if(cur_count)
{
    //(1)第一次检测到，直接将 cur_count 赋值给 sign_count
    if(first)
    {
        m_sign_count = cur_count;
        //将第一次检测标志置为 false
        first =false;
        //将没有检测到标志置 false
        isZero = false;

        //将检测到图像显示到 label,并保存
        for(int i=0; i< cur_count;i++)
        {
            showSaveImage(roi);
        }
    }
    else{//非第一次检测

        if(isZero == true)//(2)如果没有检测到后，再一次检测到,则直接加上
cur_count
        {
            m_sign_count += cur_count;
            //将没有检测到标志置 false
            isZero = false;

            //将检测到图像显示到 label,并保存
            for(int i=0; i< cur_count;i++)
            {
                showSaveImage(roi);
            }
        }else if(cur_count > last_count)  //(3)由检测 1 个，持续当前检测变成 2
时，则增加了 1 个
        {
            m_sign_count += (cur_count-last_count);

            //将检测到图像显示到 label,并保存
            for(int i=0; i<(cur_count-last_count);i++)
```

```
                {
                    showSaveImage(roi);
                }
            }
        }
        //将当前的个数赋值给上一次
        last_count = cur_count;
    }
    else{
        //将没有检测到标志置 true
        isZero = true;
    }
```

3.3 系统运行结果及算法性能

(1) 车道线检测

如下图 7，在正常的道路面上，车道线可以被检测出来。

图 7 车道线检测结果 a

如下图 8，如果路面被其他东西模糊，车道线检测出错。

图 8 车道线检测结果 b

如下图 9，变道过程中时，车道线检测出错。

图 9 车道线检测结果 c

如下图 10，变道先内有指示标线时，车道线检测出错。

图 10 车道线检测结果 d

如下图 11，右车道线的最右点的坐标小于窗口的高度时(左车道线同理)，车道线检测出错。

图 11 车道线检测结果 e

(2) 交通标志检测

如下图 12，图 13,在正常的照明条件下，交通标志可以被检测出来。

图 12 交通标志检测结果 a

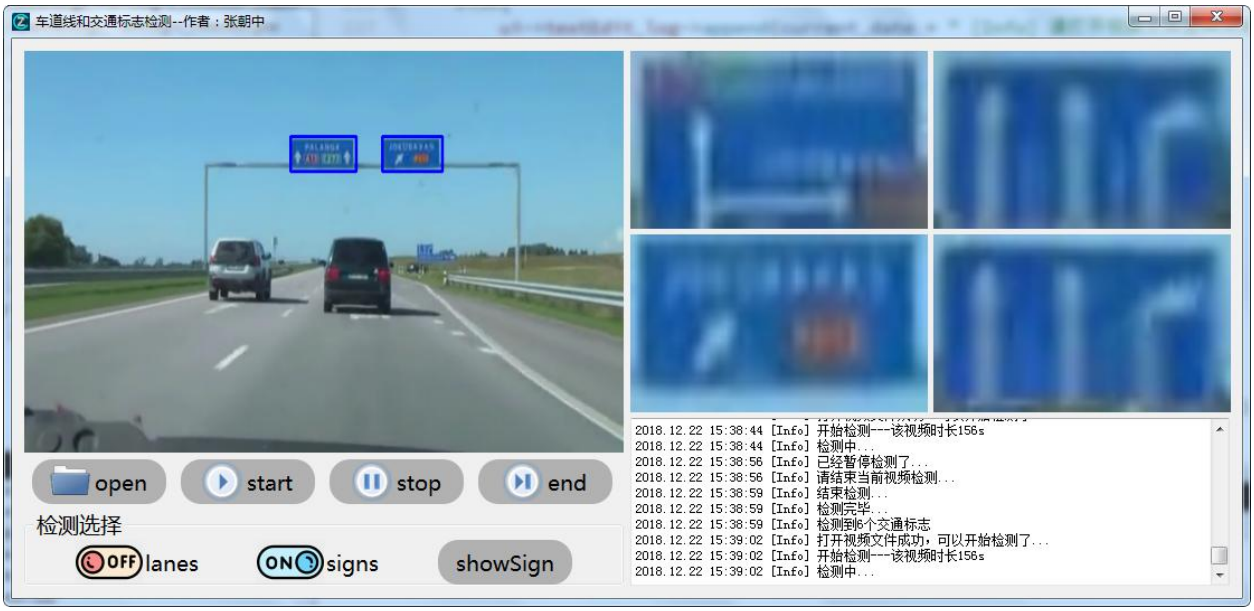

图 13 交通标志检测结果 b

如下图 14，没有检测的颜色，交通标志不可以被检测出来。

图 14 交通标志检测结果 c

如下图 15，交通标志太小时，不检测。

图 15 交通标志检测结果 d

如下图 16，显示检测到的交通标志。

图 16 显示交通标志检测结果

（3）车道线和交通标志同时检测

如下图 17，在正常道路面和正常光照条件下，车道线和交通标志可以被检测出来。

图 17 车道线和交通标志检测结果

四、课程设计总结

4.1 遇到的 bug

● 问题 1：不显示 opencv 窗口，卡死(目前不知道为什么不显示就卡死)。

解决方案：显示 opencv 窗口，然后通过获取窗口句柄将其隐藏。

● 问题 2：将帧显示到 label 上，颜色变了。

解决方案：由于 opencv 是 BGR,故需将帧转为 RGB,再显示。

● 问题 3：copyTo()出错。

解决方案：由于 copyTo()是浅拷贝，则要改为深拷贝 clone()。

● 问题 4：Ctrl+V 粗心地将=写成了==，赋值出错。

解决方案：将==改为=。

● 问题 5：opencv 窗口输出卡死

解决方案：加上 cvWaitkey(25)，等待一段时间。

● 问题 6：程序退出后，opencv 窗口程序还没有退出。

解决方案：重载 MainWindow 的 closeEvent 函数，将结束标志设置为 true。

● 问题 7：String 和 QString 类型转换问题(还有其他的类型转换问题，此处忽略)

解决方案：

①QString 转换 String　　string s = qstr.toStdString();

②String 转换 QString　　QString qstr2 = QString::fromStdString(s);

● 问题 8：Label 显示大小错乱。

解决方案：初始化时设定 label 的最小大小。

4.2 遇到的难点

▲ 难点 1：感兴趣区域(ROI)选定

解决方案：先设定帧的大小，再手动确定 Mask 的位置，大小

▲ 难点 2：如何预测转向

解决方案：通过消失点，用 m1x+b1-y1 = m2x+b2-y2 得出消失点 x,通过消失点的位置来判断转向

▲ 难点 3：获取 Hough 线，无法将多段线段连成一条

解决方案：回归拟合曲线:先计算所有直线斜率，根据中心点位置将线条分成右线和左线，fitLine（）拟合直线

▲ 难点 4：颜色分割

解决方案：粗略解决，设定 RGB 的之间关系

▲ 难点 5：计算检测到的交通标志的个数

解决方案：设计一个当前个数标志和一个上一次个数标志，将情况分为 3 种，记得标志的重新赋值。

参考文献

[1] xiao__run. C++ opencv 车道线识别. CSDN, 2018

https://blog.csdn.net/xiao__run/article/details/82746319

[2] hero_myself. 【OpenCV】利用霍夫变换进行直线检测. CSDN, 2015

https://blog.csdn.net/hero_myself/article/details/50250745

[3] 浅墨_毛星云. OpenCV 霍夫变换：霍夫线变换，霍夫圆变换合辑. CSDN, 2014

https://blog.csdn.net/poem_qianmo/article/details/26977557/

[4] 清风似水流. 绘图函数-fillConvexPoly. 博客园

https://www.cnblogs.com/hwm520hlf1314/p/3477945.html

[5] 徐大大平凡之路. opencv 二值图像的孔洞填充. CSDN, 2016

https://blog.csdn.net/hust_bochu_xuchao/article/details/51967846

[6] udacity. CarND-LaneLines-P1. github, 2016

https://github.com/udacity/CarND-LaneLines-P1

[7] mjlsuccess. OpenCV 在 Qt 中显示视频的两种方法. CSDN, 2014

https://blog.csdn.net/mjlsuccess/article/details/21696391

[8] 小马哥 MAX. Opencv3+VS2017 实现交通标志检测. CSDN, 2018

https://blog.csdn.net/majichen95/article/details/80380668

[9] kobesdu. 基于颜色的交通标志检测方法分析. CSDN, 2012

https://blog.csdn.net/kobesdu/article/details/8018757

[10] -牧野-. OpenCV 图像增强算法实现（直方图均衡化、拉普拉斯、Log、Gamma）. CSDM, 2016

https://blog.csdn.net/dcrmg/article/details/53677739

［11］-牧野-. 形态学边界提取. CSDM, 2016

https://blog.csdn.net/dcrmg/article/details/52089538

［12］不用先生. 【图像处理】彩色图像自适应对比度增强（OpenCV 实现）. CSDN, 2018

https://blog.csdn.net/u013921430/article/details/83865427

［13］紫荆飘香 V. opencv 如何隐藏窗口. CSDN, 2013

https://blog.csdn.net/ahuang1900/article/details/17386611?utm_source=blogxgwz1

［14］ 一去、二三里. Qt 之 QCheckBox. CSDN, 2016

https://blog.csdn.net/liang19890820/article/details/50976944

［15］v_xchen_v. Qt QListWidget 实现图片缩略图列表. CSDN, 2017

https://blog.csdn.net/v_xchen_v/article/details/71550498

［16］kelin6. opencv 连续保存多张图片到指定文件夹. CSDN, 2017

https://blog.csdn.net/kelin6/article/details/78912402/

［17］毛星云,冷飞雪等. OpenCV3 编程入门[M]. 北京：电子工业出版社, 2015.2

代码说明

本项目的 github 地址：

https://github.com/ZhangChaoZhong/LaneMarkings_TrafficSigns_Detection.git

具体的可以看我里面的代码和注释，谢谢。