**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： | 肖肇斌 | 姓名2： | 陈佳婷 |
| 学号1： | 201641402213 | 学号2： |  |

|  |
| --- |
| 指导老师评语:  评定成绩: 签名: 日期: |

目录

**[一、概述](#_Toc532238799)** [1](#_Toc532238799)

[1.1 课程设计的目的 1](#_Toc532238800)

[1.2 课程设计任务及要求 1](#_Toc532238801)

[1.3 开发该系统软件环境及使用的技术说明 1](#_Toc532238802)

**[二、系统设计的基本概念与原理](#_Toc532238803)** [1](#_Toc532238803)

[2.1 系统基本工作流程 1](#_Toc532238804)

[2.2 车道线检测算法设计 1](#_Toc532238805)

[2.3 道路标示牌检测算法设计 1](#_Toc532238806)

[2.4 用户接口设计 1](#_Toc532238807)

**[三、系统实施详细说明及结果](#_Toc532238808)** [1](#_Toc532238808)

[3.1道路检测算法的实现 1](#_Toc532238809)

[3.2 道路标示牌检测算法实现 1](#_Toc532238810)

[3.3系统运行结果及算法性能 1](#_Toc532238811)

**[四、课程设计总结](#_Toc532238812)** [1](#_Toc532238812)

**[参考文献](#_Toc532238813)** [2](#_Toc532238813)

**一、概述**

1.1 课程设计的目的

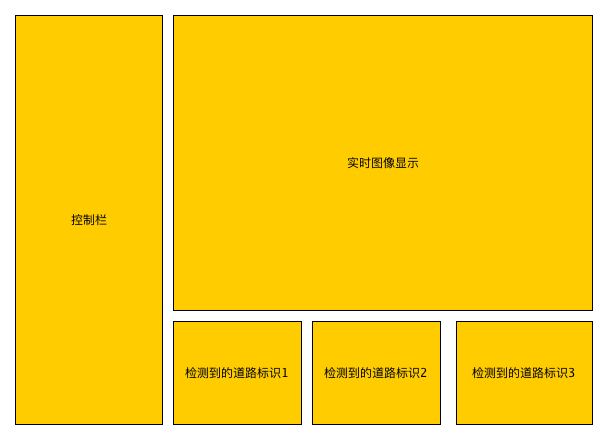
据调查，道路交通事故中有 1/3 是由车辆变道或车辆偏离其正常行驶车道区域所导致的。美国联邦公路局的研究表明：如果可以获得车辆与车道之间的相对位置信息，则可以防止 53%左右的车道偏离事故，因此针对路面标线检测的研究是实现车道偏离警告系统的关键技术，其对于实现车辆的完全自主驾驶具有深远的意义。

1.2 课程设计任务及要求

**利用OpenCV库和其他开源工具，实现车道线和道路标示牌的实时检测**

系统的基本功能要求如下：

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。
7. 界面实现可以参考下图，也可以按照自己的方式实现。



1.3 开发该系统软件环境及使用的技术说明

开发环境: ubuntu,Qt,Clion,github

所用技术: Qt,opencv,c++

**二、系统设计的基本概念与原理**

**应包含核心代码**

2.1 系统基本工作流程

系统整个工作流程图及文字描述

2.2 车道线检测算法设计

算法流程图，文字描述

2.3 道路标示牌检测算法设计

算法流程图，文字描述

2.4 用户接口设计

接口设计思路，文字描述

**三、系统实施详细说明及结果**

## 3.1 道路检测算法的实现

关键代码及文字说明

## 3.2 道路标示牌检测算法实现

关键代码及文字说明

//合并的路标检测函数  
**void** RoadSign::checkRoadSign(Mat inputImage,Mat outputImage,**double** colorLimit[],**int** binaryLimit[],**int** erodeNum[],**int** dilateNum[],**int** closeNum[],**double** angleLimit[],**double** wid\_hei[],**int** sizeOfArea,**int** someArea,**int** someOperate){  
 Mat hsv,operateMat;;  
 Mat operateImage = inputImage.clone();  
  
 //滤波  
 **if**(someOperate) {  
 blur(operateImage, operateImage, Size(3, 3));  
 GaussianBlur(operateImage, operateImage, Size(3, 3), 0, 0);  
 medianBlur(operateImage, operateImage, 5);  
 }  
  
 //RGB转为HSV图像显示  
 cv::cvtColor(operateImage, hsv, cv::*COLOR\_BGR2HSV*);  
  
 //阈值分割  
 //cv::inRange(hsv, cv::Scalar(214-BLUE,61-BLUE, 62-BLUE), cv::Scalar(214+BLUE, 61+BLUE,62+BLUE), blue\_mask);  
 cv::inRange(hsv, cv::Scalar(colorLimit[0], colorLimit[1],colorLimit[2]), cv::Scalar(colorLimit[3],colorLimit[4],colorLimit[5]),  
 operateMat);  
  
 //二值化  
 threshold(operateMat, operateMat, binaryLimit[0],binaryLimit[1], *CV\_THRESH\_BINARY*);  
  
 //腐蚀膨胀  
 erode(operateMat, operateMat, getStructuringElement(*MORPH\_RECT*, Size(erodeNum[0],erodeNum[1])));  
  
 dilate(operateMat, operateMat, getStructuringElement(*MORPH\_RECT*, Size(dilateNum[0],dilateNum[1])));  
  
 //闭运算  
 Mat kernel = getStructuringElement(*MORPH\_RECT*, Size(closeNum[0],closeNum[1]));  
 morphologyEx(operateMat, operateMat, *MORPH\_CLOSE*, kernel);  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*添加感兴趣区域\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 **if**(someArea) {  
 **int** width, height;  
 width = operateMat.cols;  
 height = operateMat.rows;  
  
 //original的参数  
 cv::Point pts[4] = {  
 Point(width / 5, height / 4),//左下角开始顺时针  
 Point(width / 5, 0),  
 Point(width \* 4 / 5, 0),  
 Point(width \* 4 / 5, height / 4)  
 };  
  
 // 创建二进制多边形，填充多边形获取感兴趣区域  
 cv::fillConvexPoly(operateMat, pts, 4, cv::Scalar(0, 0, 0));  
 //imshow("blue", operateMat);  
 }  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
 //查找轮廓并显示  
 vector<vector<Point>> contours;  
 vector<Vec4i> hierarchy;  
 findContours(operateMat, contours, hierarchy, CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE);  
 vector<vector<Point>> approxContours(contours.size());  
 **unsigned int** index = 0;  
 **while** (index < contours.size()){  
 approxPolyDP(contours[index],approxContours[index],20,**true**);  
 **if**(approxContours.at(index).size()>15){  
 //drawContours(outputImage,approxContours,index,Scalar(0,0,255),2);  
 index++;  
 **continue**;  
 }  
  
 //轮廓筛选  
 **if** (contours.at(index).size() >= 4 /\*轮廓角点初步筛选\*/  
 && hierarchy.at(index).val[2] >= 0 && hierarchy.at(index).val[3] < 0 /\*找具有内轮廓的最外层轮廓\*/) {  
 RotatedRect cnt = minAreaRect(contours.at(index));//找出外接矩阵  
 **if** (cnt.angle > angleLimit[0]&& cnt.angle <=angleLimit[1]) {//矩阵外界角度筛选  
 **int** middle = cnt.size.width / cnt.size.height;  
 **if**(middle >= wid\_hei[0]&&middle <= wid\_hei[1]&&cnt.size.area() > 100 && cnt.size.area() < sizeOfArea) {//矩阵长宽比和矩阵大小筛选  
 Point2f box[4];  
 cnt.points(box);  
 //画出外接矩阵  
 //rectangle(outputImage,box[0],box[2],Scalar(0,255,255),2);

//裁剪出筛选后  
 Rect rect=boundingRect(contours.at(index));  
 Mat ROI=outputImage(rect).clone();  
 cvtColor(ROI,ROI,CV\_BGR2RGB);  
 QImage image = QImage( (**const unsigned char**\*)(ROI.data), ROI.cols, ROI.rows, QImage::Format\_RGB888 );  
 MainWindow \*p=MainWindow::ptr;  
 **int** wid=ROI.cols>p->getUI()->roadSign0->width()? p->getUI()->roadSign0->width():ROI.cols;  
 **int** hei=ROI.rows>p->getUI()->roadSign0->height()?p->getUI()->roadSign0->height():ROI.rows;

//将裁剪的图片显示到窗体指定位置

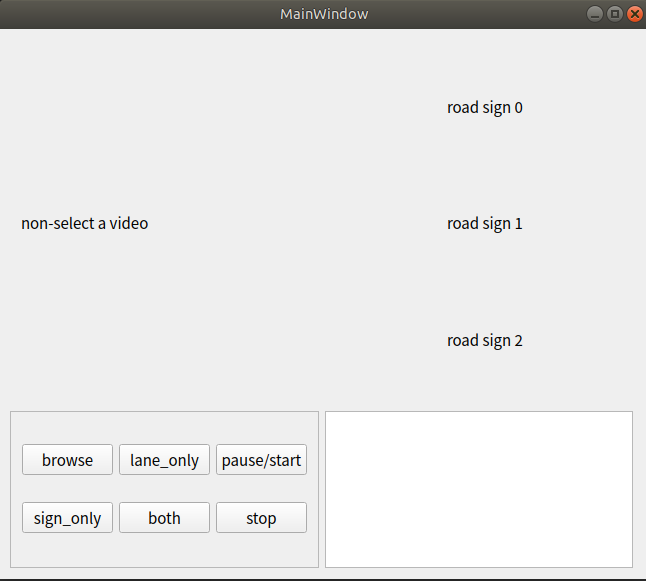
**if**(count==0){  
 p->getUI()->roadSign0->setPixmap( QPixmap::fromImage(image.scaled(wid,hei)));  
 //p->ui->outputImage->resize(100,100);  
 p->getUI()->roadSign0->show();  
 }**else if**(count==1){  
 p->getUI()->roadSign1->setPixmap( QPixmap::fromImage(image.scaled(wid,hei)));  
 //p->ui->outputImage->resize(100,100);  
 p->getUI()->roadSign1->show();  
 }**else**{  
 p->getUI()->roadSign2->setPixmap( QPixmap::fromImage(image.scaled(wid,hei)));  
 //p->ui->outputImage->resize(100,100);  
 p->getUI()->roadSign2->show();  
 }  
 count++;  
 count%=3;  
 }  
 }  
 }  
 index++;  
 }  
}  
  
//获取不感兴趣区域  
Mat RoadSign::mask(cv::Mat img) {  
 cv::Mat output;  
 **int** width, height;  
 width = img.cols;  
 height = img.rows;  
  
 cv::Mat mask = cv::Mat::zeros(img.size(), img.type());  
  
 cv::Point pts[4] = {  
 Point(width \* 2 / 5, height / 5),//左下角开始顺时针  
 Point(width \* 2 /5 , 0),  
 Point(width \* 4 / 5, 0),  
 Point(width \* 4 / 5, height / 5)  
 };  
  
 // 创建二进制多边形，填充多边形获取感兴趣区域  
 cv::fillConvexPoly(mask, pts, 4, cv::Scalar(255, 255, 255));  
  
 // 将边缘图像和掩膜进行与操作以获得输出  
 cv::bitwise\_and(img, mask, output);  
  
 cv::namedWindow("ROI", 0);  
 cv::resizeWindow("ROI", 800, 380);  
 cv::imshow("ROI", output);  
  
 **return** output;  
}

## 3.3系统运行结果及算法性能

给出算法处理的效率（每次处理所需要的时间）和算法的准确率的统计。

运行结果:

前端ui:



路标检测:



**四、课程设计总结**

简要描述设计过程中遇到的问题，解决的方式

# **参考文献**

**Qt官方文档**

**C++官方文档**

**opencv官方文档**