# FatigueDrivingReco的程序文档

## 文档说明

文档主要对项目的程序进行说明和描述程序的思想。

* 程序的功能
* 程序的思想
* 程序的源码
* 注意之处（程序中比较难理解，比较特殊的地方）
* 待改进之处（能使得效果更好的地方）

## 程序内容

### main（）函数

* 1. 程序的功能

首先，利用Adaboost算法检测人脸，紧接着根据人脸的先验知识分割出大致的人眼区域。然后，对人眼大致区域的图像进行图像增强处理（中值滤波、非线性点运算），接着利用Ostu算法计算最佳分割阈值，对图像进行二值化处理。

然后定位人眼的具体位置，具体有以下几个步骤。首先利用直方图积分投影，根据设定的阈值判断并消除眉毛区域。然后分割出左眼和右眼的图像，分别对左右眼的图像计算直方图和直方图积分投影，从而分别确定左右眼的中心位置。

最后，根据定位出的左右眼的中心位置，人为设定人眼矩形框的大小，根据矩形框内的像素特征判断眼睛的睁开闭合状态。有三个特征，眼睛长宽比R，黑色像素占总像素的比例α，以虹膜中心点为中心的1/2中间区域的黑色像素比例β。根据模糊综合评价的思想，将这三个指标划分为相同的4个级别（见下表），然后根据百分比组合成一个函数。最终根据函数值与阈值比较，确定眼睛的睁开、闭合状态。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 闭合 | 可能闭合 | 可能睁开 | 睁开 | 标准 | 权重 |
| Value | 0 | 2 | 6 | 8 | -- | -- |
| R | (0, 0.8]  (3, 无穷] | (0.8, 1.2] | (1.2, 1.5]  (2.5, 3] | (1.5, 2.5] | 2.0 | 0.2 |
| α | (0, 0.4] | (0.4, 0.5] | (0.5, 0.6] | (0.6, 1] | 0.65 | 0.4 |
| β | (0, 0.3] | (0.3, 0.45] | (0.45, 0.6] | (0.6, 1] | 0.55 | 0.4 |

为了判定驾驶员是否处于疲劳驾驶状态，需要对很多帧视频进行上述处理，根据PERCLOS原理和制定的判断规则，判断最终状态。

* 1. 程序的思想
  2. 程序的源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：检测人脸，检测人眼，识别人眼闭合状态，判断是否处于疲劳驾驶状态

改进：

1. detectFace（）中用了直方图均衡化，到时看有没有必要

2. 二值化的效果不太理想，到时用实际的驾驶图片测试再看看怎么改进。

二值化之前一定要做图像增强：非线性点运算或直方图均衡化。

在OSTU找到的最优阈值基础上减了一个常数，但减太多了，导致整张图片很灰暗的情况下二值化效果很差。

3. detectFace子函数中有一个budge：返回的objects在子函数外被释放了！

说明：

Date：2014.08.14

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <highgui.h>

#include <cv.h>

#include <cxcore.h>

#include "histogram.h"

#include "memory.h"

#include "time.h"

#include "ostuThreshold.h"

#include "detectFace.h"

#include "histProject.h"

#include "linetrans.h"

#include "nonlinetrans.h"

#include "getEyePos.h"

#include "recoEyeState.h"

#include "recoFatigueState.h"

#define DETECTTIME 30 // 一次检测过程的时间长度，用检测次数衡量

#define FATIGUETHRESHOLD 180 // 判断是否疲劳驾驶的阈值

extern CvSeq**\*** objectsTemp **=** **NULL;** // 传递objects的值回来main（）

int main**()**

**{**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 主程序用到的参数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IplImage **\*** srcImg **=** **NULL;** // 存放从摄像头读取的每一帧彩色源图像

IplImage **\*** img **=** **NULL;** // 存放从摄像头读取的每一帧灰度源图像

CvCapture **\*** capture**;** // 指向CvCapture结构的指针

CvMemStorage**\*** storage **=** cvCreateMemStorage**(**0**);** // 存放矩形框序列的内存空间

CvSeq**\*** objects **=** **NULL;** // 存放检测到人脸的平均矩形框

double scale\_factor **=** 1.2**;** // 搜索窗口的比例系数

int min\_neighbors **=** 3**;** // 构成检测目标的相邻矩形的最小个数

int flags **=** 0**;** // 操作方式

CvSize min\_size **=** cvSize**(**40**,** 40**);** // 检测窗口的最小尺寸

int i**,** globalK**;**

// 绘制人脸框选用的颜色

int hist**[**256**];** // 存放直方图的数组

int pixelSum**;**

int threshold**;** // 存储二值化最优阈值

clock\_t start**,** stop**;** // 计时参数

IplImage**\*** faceImg **=** **NULL;** // 存储检测出的人脸图像

int temp **=** 0**;** // 临时用到的变量

int temp1 **=** 0**;** // 临时用到的变量

int count **=** 0**;** // 计数用的变量

int flag **=** 0**;** // 标记变量

int **\*** tempPtr **=** **NULL;** // 临时指针

CvRect**\*** largestFaceRect**;** // 存储检测到的最大的人脸矩形框

int **\*** horiProject **=** **NULL;** // 水平方向的投影结果(数组指针)

int **\*** vertProject **=** **NULL;** // 垂直方向的投影结果(数组指针)

int **\*** subhoriProject **=** **NULL;** // 水平方向的投影结果(数组指针)

int **\*** subvertProject **=** **NULL;** // 垂直方向的投影结果(数组指针)

int WIDTH**;** // 图像的宽度

int HEIGHT**;** // 图像的高度

int rEyeCol **=** 0**;** // 右眼所在的列数

int lEyeCol **=** 0**;** // 左眼所在的列数

int lEyeRow **=** 0**;** // 左眼所在的行数

int rEyeRow **=** 0**;** // 右眼所在的行数

int eyeBrowThreshold**;** // 区分眉毛与眼睛之间的阈值

uchar**\*** rowPtr **=** **NULL;** // 指向图片每行的指针

uchar**\*** rowPtrTemp **=** **NULL;** // 指向图片每行的指针, 中间变量

IplImage**\*** eyeImg **=** **NULL;** // 存储眼睛的图像

CvRect eyeRect**;** // 存储裁剪后的人眼的矩形区域

CvRect eyeRectTemp**;** // 临时矩形区域

IplImage**\*** lEyeImg **=** **NULL;** // 存储左眼的图像

IplImage**\*** rEyeImg **=** **NULL;** // 存储右眼的图像

IplImage**\*** lEyeImgNoEyebrow **=** **NULL;** // 存储去除眉毛之后的左眼图像

IplImage**\*** rEyeImgNoEyebrow **=** **NULL;** // 存储去除眉毛之后的右眼图像

IplImage**\*** lEyeballImg **=** **NULL;** // 存储最终分割的左眼框的图像

IplImage**\*** rEyeballImg **=** **NULL;** // 存储最终分割的右眼框的图像

IplImage**\*** lMinEyeballImg **=** **NULL;** // 存储最终分割的最小的左眼框的图像

IplImage**\*** rMinEyeballImg **=** **NULL;** // 存储最终分割的最小的右眼框的图像

int lMinEyeballBlackPixel**;** // 存储最终分割的最小的左眼框的白色像素个数

int rMinEyeballBlackPixel**;** // 存储最终分割的最小的右眼框的白色像素个数

double lMinEyeballBlackPixelRate**;** // 存储最终分割的最小的左眼框的黑色像素占的比例

double rMinEyeballBlackPixelRate**;** // 存储最终分割的最小的右眼框的黑色像素占的比例

double lMinEyeballRectShape**;** // 存储最小左眼眶的矩形长宽比值

double rMinEyeballRectShape**;** // 存储最小右眼眶的矩形长宽比值

double lMinEyeballBeta**;** // 存储最小左眼眶的中间1/2区域的黑像素比值

double rMinEyeballBeta**;** // 存储最小右边眼眶的中间1/2区域的黑像素比值

int lEyeState**;** // 左眼睁（0）、闭（1）状态

int rEyeState**;** // 右眼睁（0）、闭（1）状态

int eyeState**;** // 眼睛综合睁（0）、闭（1）状态

int eyeCloseNum **=** 0**;** // 统计一次检测过程中闭眼的总数

int eyeCloseDuration **=** 0**;** // 统计一次检测过程中连续检测到闭眼状态的次数

int maxEyeCloseDuration **=** 0**;** // 一次检测过程中连续检测到闭眼状态的次数的最大值

int failFaceNum **=** 0**;** // 统计一次检测过程中未检测到人脸的总数

int failFaceDuration **=** 0**;** // 统计一次检测过程中连续未检测到人脸的次数

int maxFailFaceDuration **=** 0**;** // 一次检测过程中连续未检测到人脸的次数的最大值

int fatigueState **=** 1**;** // 驾驶员的驾驶状态：疲劳驾驶（1），正常驾驶（0）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 创建显示窗口 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cvNamedWindow**(**"img"**,** CV\_WINDOW\_AUTOSIZE**);** // 显示灰度源图像

cvNamedWindow**(**"分割后的人脸"**,** 1**);** // 显示分割出大致眼眶区域的人脸

cvNamedWindow**(**"大致的左眼区域"**,** 1**);** // 显示大致的左眼区域

cvNamedWindow**(**"大致的右眼区域"**,** 1**);** // 显示大致的右眼区域

cvNamedWindow**(**"l\_binary"**);** // 显示大致右眼区域的二值化图像

cvNamedWindow**(**"r\_binary"**);** // 显示大致左眼区域的二值化图像

cvNamedWindow**(**"lEyeImgNoEyebrow"**,** 1**);** // 显示去除眉毛区域的左眼图像

cvNamedWindow**(**"rEyeImgNoEyebrow"**,** 1**);** // 显示去除眉毛区域的右眼图像

cvNamedWindow**(**"lEyeCenter"**,** 1**);** // 显示标出虹膜中心的左眼图像

cvNamedWindow**(**"rEyeCenter"**,** 1**);** // 显示标出虹膜中心的右眼图像

cvNamedWindow**(**"lEyeballImg"**,** 1**);** // 根据lEyeImgNoEyebrow大小的1/2区域重新划分的左眼图像

cvNamedWindow**(**"rEyeballImg"**,** 1**);** // 根据rEyeImgNoEyebrow大小的1/2区域重新划分的右眼图像

cvNamedWindow**(**"lkai"**,** 1**);** // 左眼进行开运算之后的图像

cvNamedWindow**(**"rkai"**,** 1**);** // 右眼进行开运算之后的图像

cvNamedWindow**(**"lMinEyeballImg"**,** 1**);** // 缩小至边界区域的左眼虹膜图像

cvNamedWindow**(**"rMinEyeballImg"**,** 1**);** // 缩小至边界区域的右眼眼虹膜图像

capture **=** cvCreateCameraCapture**(**0**);**

**if(** capture **==** **NULL** **)**

**return** **-**1**;**

**for(** globalK **=** 1**;** globalK **<=** DETECTTIME**;** globalK **++** **){**

start **=** clock**();**

srcImg **=** cvQueryFrame**(**capture**);**

img **=** cvCreateImage**(**cvGetSize**(**srcImg**),** IPL\_DEPTH\_8U**,** 1**);**

cvCvtColor**(**srcImg**,** img**,** CV\_BGR2GRAY**);**

**if(** **!**img **)**

**continue;**

cvShowImage**(**"img"**,** img**);**

cvWaitKey**(**20**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 检测人脸 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cvClearMemStorage**(**storage**);** // 将存储块的 top 置到存储块的头部，既清空存储块中的存储内容

detectFace**(**

img**,** // 灰度图像

objects**,** // 输出参数：检测到人脸的矩形框

storage**,** // 存储矩形框的内存区域

scale\_factor**,** // 搜索窗口的比例系数

min\_neighbors**,** // 构成检测目标的相邻矩形的最小个数

flags**,** // 操作方式

cvSize**(**20**,** 20**)** // 检测窗口的最小尺寸

**);**

// 提取人脸区域

**if** **(** **!**objectsTemp**->**total **){**

printf**(**"Failed to detect face!\n"**);** // 调试代码

failFaceNum **++;** // 统计未检测到人脸的次数

failFaceDuration **++;** // 统计连续未检测到人脸的次数

// 检测过程中判断全是闭眼和检测不到人脸的情况，没有睁开眼的情况，导致maxEyeCloseDuration = 0;

**(**eyeCloseDuration **>** maxEyeCloseDuration**)** **?** maxEyeCloseDuration **=** eyeCloseDuration **:** maxEyeCloseDuration**;**

eyeCloseDuration **=** 0**;**

**if(** globalK **==** DETECTTIME **){**

// 当一次检测过程中，所有的过程都检测不到人脸，则要在此更新 maxFailFaceDuration

**(**failFaceDuration **>** maxFailFaceDuration**)** **?** maxFailFaceDuration **=** failFaceDuration **:** maxFailFaceDuration**;**

printf**(**"\nFATIGUETHRESHOLD: %d\n"**,** FATIGUETHRESHOLD**);**

printf**(**"eyeCloseNum: %d\tmaxEyeCloseDuration: %d\n"**,** eyeCloseNum**,** maxEyeCloseDuration**);**

printf**(**"failFaceNum: %d\tmaxFailFaceDuration: %d\n"**,** failFaceNum**,** maxFailFaceDuration**);**

// 进行疲劳状态的判别

fatigueState **=** recoFatigueState**(**FATIGUETHRESHOLD**,** eyeCloseNum**,** maxEyeCloseDuration**,** failFaceNum**,** maxFailFaceDuration**);**

**if(** fatigueState **==** 1 **)**

printf**(**"驾驶员处于疲劳驾驶状态\n\n"**);**

**else** **if(** fatigueState **==** 0 **)**

printf**(**"驾驶员处于正常驾驶状态\n\n"**);**

// 进入下一次检测过程前，将变量清零

globalK **=** 0**;**

lEyeState **=** 1**;**

rEyeState **=** 1**;**

eyeState **=** 1**;**

eyeCloseNum **=** 0**;**

eyeCloseDuration **=** 0**;**

maxEyeCloseDuration **=** 0**;**

failFaceNum **=** 0**;**

failFaceDuration **=** 0**;**

maxFailFaceDuration **=** 0**;**

fatigueState **=** 1**;**

cvWaitKey**(**0**);**

**}**

**continue;**

**}**

**else{**

// 统计连续未检测到人脸的次数中的最大数值

**(**failFaceDuration **>** maxFailFaceDuration**)** **?** maxFailFaceDuration **=** failFaceDuration **:** maxFailFaceDuration**;**

failFaceDuration **=** 0**;**

// 找到检测到的最大的人脸矩形区域

temp **=** 0**;**

**for(**i **=** 0**;** i **<** **(**objectsTemp **?** objectsTemp**->**total **:** 0**);** i **++)** **{**

CvRect**\*** rect **=** **(**CvRect**\*)** cvGetSeqElem**(**objectsTemp**,** i**);**

**if** **(** **(**rect**->**height **\*** rect**->**width**)** **>** temp **){**

largestFaceRect **=** rect**;**

temp **=** rect**->**height **\*** rect**->**width**;**

**}**

**}**

// 根据人脸的先验知识分割出大致的人眼区域

temp **=** largestFaceRect**->**width **/** 8**;**

largestFaceRect**->**x **=** largestFaceRect**->**x **+** temp**;**

largestFaceRect**->**width **=** largestFaceRect**->**width **-** 3**\***temp**/**2**;**

largestFaceRect**->**height **=** largestFaceRect**->**height **/** 2**;**

largestFaceRect**->**y **=** largestFaceRect**->**y **+** largestFaceRect**->**height **/** 2**;**

largestFaceRect**->**height **=** largestFaceRect**->**height **/** 2**;**

cvSetImageROI**(**img**,** **\***largestFaceRect**);** // 设置ROI为检测到的最大的人脸区域

faceImg **=** cvCreateImage**(**cvSize**(**largestFaceRect**->**width**,** largestFaceRect**->**height**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**img**,** faceImg**,** **NULL);**

cvResetImageROI**(**img**);** // 释放ROI

cvShowImage**(**"分割后的人脸"**,** faceImg**);**

eyeRectTemp **=** **\***largestFaceRect**;**

// 根据人脸的先验知识分割出大致的左眼区域

largestFaceRect**->**width **/=** 2**;**

cvSetImageROI**(**img**,** **\***largestFaceRect**);** // 设置ROI为检测到的最大的人脸区域

lEyeImg **=** cvCreateImage**(**cvSize**(**largestFaceRect**->**width**,** largestFaceRect**->**height**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**img**,** lEyeImg**,** **NULL);**

cvResetImageROI**(**img**);** // 释放ROI

cvShowImage**(**"大致的左眼区域"**,** lEyeImg**);**

// 根据人脸的先验知识分割出大致的右眼区域

eyeRectTemp**.**x **+=** eyeRectTemp**.**width **/** 2**;**

eyeRectTemp**.**width **/=** 2**;**

cvSetImageROI**(**img**,** eyeRectTemp**);** // 设置ROI为检测到的最大的人脸区域

rEyeImg **=** cvCreateImage**(**cvSize**(**eyeRectTemp**.**width**,** eyeRectTemp**.**height**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**img**,** rEyeImg**,** **NULL);**

cvResetImageROI**(**img**);** // 释放ROI

cvShowImage**(**"大致的右眼区域"**,** rEyeImg**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 二值化处理 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 图像增强：直方图均衡化在detectFace中实现了一次；可尝试非线性点运算

/\*\*\* 二值化左眼大致区域的图像 \*\*\*/

//lineTrans(lEyeImg, lEyeImg, 1.5, 0); // 线性点运算

cvSmooth**(**lEyeImg**,** lEyeImg**,** CV\_MEDIAN**);** // 中值滤波 默认窗口大小为3\*3

nonlineTrans**(**lEyeImg**,** lEyeImg**,** 0.8**);** // 非线性点运算

memset**(**hist**,** 0**,** **sizeof(**hist**));** // 初始化直方图的数组为0

histogram**(**lEyeImg**,** hist**);** // 计算图片直方图

// 计算最佳阈值

pixelSum **=** lEyeImg**->**width **\*** lEyeImg**->**height**;**

threshold **=** ostuThreshold**(**hist**,** pixelSum**,** 45**);**

cvThreshold**(**lEyeImg**,** lEyeImg**,** threshold**,** 255**,** CV\_THRESH\_BINARY**);**// 对图像二值化

// 显示二值化后的图像

cvShowImage**(**"l\_binary"**,**lEyeImg**);**

/\*\*\* 二值化右眼大致区域的图像 \*\*\*/

//lineTrans(rEyeImg, rEyeImg, 1.5, 0); // 线性点运算

cvSmooth**(**rEyeImg**,** rEyeImg**,** CV\_MEDIAN**);** // 中值滤波 默认窗口大小为3\*3

nonlineTrans**(**rEyeImg**,** rEyeImg**,** 0.8**);** // 非线性点运算

memset**(**hist**,** 0**,** **sizeof(**hist**));** // 初始化直方图的数组为0

histogram**(**rEyeImg**,** hist**);** // 计算图片直方图

// 计算最佳阈值

pixelSum **=** rEyeImg**->**width **\*** rEyeImg**->**height**;**

threshold **=** ostuThreshold**(**hist**,** pixelSum**,** 45**);**

cvThreshold**(**rEyeImg**,** rEyeImg**,** threshold**,** 255**,** CV\_THRESH\_BINARY**);**// 对图像二值化

// 显示二值化后的图像

cvShowImage**(**"r\_binary"**,**rEyeImg**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 检测人眼 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* 如果有明显的眉毛区域，则分割去除眉毛 \*\*/

// 分割左眼眉毛

HEIGHT **=** lEyeImg**->**height**;**

WIDTH **=** lEyeImg**->**width**;**

// 分配内存

horiProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

vertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** horiProject **==** **NULL** **||** vertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**horiProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**vertProject **+** i**)** **=** 0**;**

histProject**(**lEyeImg**,** horiProject**,** vertProject**);** // 计算直方图投影

lEyeRow **=** removeEyebrow**(**horiProject**,** WIDTH**,** HEIGHT**,** 10**);** // 计算分割眉毛与眼框的位置

// 分割右眼眉毛

HEIGHT **=** rEyeImg**->**height**;**

WIDTH **=** rEyeImg**->**width**;**

// 分配内存

horiProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

vertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** horiProject **==** **NULL** **||** vertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**horiProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**vertProject **+** i**)** **=** 0**;**

histProject**(**rEyeImg**,** horiProject**,** vertProject**);** // 计算直方图投影

rEyeRow **=** removeEyebrow**(**horiProject**,** WIDTH**,** HEIGHT**,** 10**);** // 计算分割眉毛与眼框的位置

// 显示去除眉毛后的人眼大致区域

eyeRect **=** cvRect**(**0**,** lEyeRow**,** lEyeImg**->**width**,** **(**lEyeImg**->**height **-** lEyeRow**));** // 去眉毛的眼眶区域在lEyeImg中的矩形框区域

cvSetImageROI**(**lEyeImg**,** eyeRect**);** // 设置ROI为去除眉毛的眼眶，在下面释放ROI

lEyeImgNoEyebrow **=** cvCreateImage**(**cvSize**(**eyeRect**.**width**,** eyeRect**.**height**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**lEyeImg**,** lEyeImgNoEyebrow**,** **NULL);**

cvShowImage**(**"lEyeImgNoEyebrow"**,** lEyeImgNoEyebrow**);**

eyeRectTemp **=** cvRect**(**0**,** rEyeRow**,** rEyeImg**->**width**,** **(**rEyeImg**->**height **-** rEyeRow**));** // 去眉毛的眼眶区域在rEyeImg中的矩形框区域

cvSetImageROI**(**rEyeImg**,** eyeRectTemp**);** // 设置ROI为去除眉毛的眼眶，在下面释放ROI

rEyeImgNoEyebrow **=** cvCreateImage**(**cvSize**(**eyeRectTemp**.**width**,** eyeRectTemp**.**height**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**rEyeImg**,** rEyeImgNoEyebrow**,** **NULL);**

cvShowImage**(**"rEyeImgNoEyebrow"**,** rEyeImgNoEyebrow**);**

///////////////// 定位眼睛中心点在去除眉毛图像中的行列位置 ///////////////////

HEIGHT **=** lEyeImgNoEyebrow**->**height**;**

WIDTH **=** lEyeImgNoEyebrow**->**width**;**

// 分配内存

subhoriProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

subvertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** subhoriProject **==** **NULL** **||** subvertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**subhoriProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**subvertProject **+** i**)** **=** 0**;**

histProject**(**lEyeImgNoEyebrow**,** subhoriProject**,** subvertProject**);** // 重新对分割出的左眼图像进行积分投影

lEyeRow **=** getEyePos**(**subhoriProject**,** HEIGHT**,** HEIGHT**/**5**);** // 定位左眼所在的行

lEyeCol **=** getEyePos**(**subvertProject**,** WIDTH**,** WIDTH**/**5**);** // 定位左眼所在的列

HEIGHT **=** rEyeImgNoEyebrow**->**height**;**

WIDTH **=** rEyeImgNoEyebrow**->**width**;**

// 分配内存

subhoriProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

subvertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** subhoriProject **==** **NULL** **||** subvertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**subhoriProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**subvertProject **+** i**)** **=** 0**;**

histProject**(**rEyeImgNoEyebrow**,** subhoriProject**,** subvertProject**);** // 重新对分割出的右眼图像进行积分投影

rEyeRow **=** getEyePos**(**subhoriProject**,** HEIGHT**,** HEIGHT**/**5**);** // 定位右眼所在的行

rEyeCol **=** getEyePos**(**subvertProject**,** WIDTH**,** WIDTH**/**5**);** // 定位右眼所在的列

/\*

printf("\*\*\*\*\*\*\*\*\*\*\*\* image of eyes without eyebrow \*\*\*\*\*\*\*\*\*\*\*\n");

printf("Left eye: width: %d\theight: %d\n", lEyeImgNoEyebrow->width, lEyeImgNoEyebrow->height);

printf("Right eye: width: %d\theight: %d\n", rEyeImgNoEyebrow->width, rEyeImgNoEyebrow->height);

printf("Right eye: WIDTH: %d\tHEIGHT: %d\n", WIDTH, HEIGHT);

printf("Centers positon of Eyes. lEyeRow: %d lEyeCol: %d\trEyeRow: %d rEyeCol: %d\n\n", lEyeRow, lEyeCol, rEyeRow, rEyeCol);

\*/

// 标记眼睛的位置

cvCircle**(**lEyeImgNoEyebrow**,** cvPoint**(**lEyeCol**,** lEyeRow**),** 3**,** CV\_RGB**(**0**,**0**,**255**),** 1**,** 8**,** 0**);**

cvCircle**(**rEyeImgNoEyebrow**,** cvPoint**(**rEyeCol**,** rEyeRow**),** 3**,** CV\_RGB**(**0**,**0**,**255**),** 1**,** 8**,** 0**);**

cvShowImage**(**"lEyeCenter"**,** lEyeImgNoEyebrow**);**

cvShowImage**(**"rEyeCenter"**,** rEyeImgNoEyebrow**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 判断人眼睁闭状态 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

////////////////// 分割出以找到的中心为中心的大致眼眶 /////////////////

// 左眼眶

HEIGHT **=** lEyeImgNoEyebrow**->**height**;**

WIDTH **=** lEyeImgNoEyebrow**->**width**;**

// 计算大致眼眶的区域: eyeRect

eyeRect **=** cvRect**(**0**,** 0**,** WIDTH**,** HEIGHT**);**

calEyeSocketRegion**(&**eyeRect**,** WIDTH**,** HEIGHT**,** lEyeCol**,** lEyeRow**);**

/\*

printf("\*\*\*\*\*\*\*\*\*\*\*\*lEyeImgNoEyebrow\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("width: %d\theight: %d\n", WIDTH, HEIGHT);

printf("\*\*\*\*\*\*\*\*\*\*lEyeballRect\*\*\*\*\*\*\*\*\*\*\n");

printf("eyeRect.x = %d\teyeRect.width = %d\n", eyeRect.x, eyeRectTemp.width);

printf("eyeRect.y = %d\teyeRect.height = %d\n\n", eyeRectTemp.y, eyeRectTemp.height);

\*/

cvSetImageROI**(**lEyeImgNoEyebrow**,** eyeRect**);** // 设置ROI为检测到眼眶区域

lEyeballImg **=** cvCreateImage**(**cvGetSize**(**lEyeImgNoEyebrow**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**lEyeImgNoEyebrow**,** lEyeballImg**,** **NULL);**

cvResetImageROI**(**lEyeImgNoEyebrow**);**

cvShowImage**(**"lEyeballImg"**,** lEyeballImg**);**

// 右眼眶

HEIGHT **=** rEyeImgNoEyebrow**->**height**;**

WIDTH **=** rEyeImgNoEyebrow**->**width**;**

// 计算大致眼眶的区域: eyeRectTemp

eyeRect **=** cvRect**(**0**,** 0**,** WIDTH**,** HEIGHT**);**

calEyeSocketRegion**(&**eyeRect**,** WIDTH**,** HEIGHT**,** rEyeCol**,** rEyeRow**);**

/\*

printf("\*\*\*\*\*\*\*\*\*\*\*\*rEyeImgNoEyebrow\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("width: %d\theight: %d\n", WIDTH, HEIGHT);

printf("\*\*\*\*\*\*\*\*\*\*rEyeballRect\*\*\*\*\*\*\*\*\*\*\n");

printf("eyeRect.x = %d\teyeRect.width = %d\n", eyeRect.x, eyeRect.width);

printf("eyeRect.y = %d\teyeRect.height = %d\n\n", eyeRect.y, eyeRect.height);

\*/

cvSetImageROI**(**rEyeImgNoEyebrow**,** eyeRect**);** // 设置ROI为检测到眼眶区域

rEyeballImg **=** cvCreateImage**(**cvGetSize**(**rEyeImgNoEyebrow**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**rEyeImgNoEyebrow**,** rEyeballImg**,** **NULL);**

cvResetImageROI**(**rEyeImgNoEyebrow**);**

cvShowImage**(**"rEyeballImg"**,** rEyeballImg**);**

/////////////////////////// 闭运算 ///////////////////////////

cvErode**(**lEyeballImg**,** lEyeballImg**,** **NULL,** 2**);** //腐蚀图像

cvDilate**(**lEyeballImg**,** lEyeballImg**,** **NULL,** 2**);** //膨胀图像

cvShowImage**(**"lkai"**,** lEyeballImg**);**

cvErode**(**rEyeballImg**,** rEyeballImg**,** **NULL,** 1**);** //腐蚀图像

cvDilate**(**rEyeballImg**,** rEyeballImg**,** **NULL,** 1**);** //膨胀图像

cvShowImage**(**"rkai"**,** rEyeballImg**);**

/////////////////// 计算最小眼睛的矩形区域 ////////////////////

///////////////////////////左眼

HEIGHT **=** lEyeballImg**->**height**;**

WIDTH **=** lEyeballImg**->**width**;**

// 分配内存

subhoriProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

subvertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** subhoriProject **==** **NULL** **||** subvertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**subhoriProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**subvertProject **+** i**)** **=** 0**;**

histProject**(**lEyeballImg**,** subhoriProject**,** subvertProject**);**

// 计算左眼最小的矩形区域

eyeRectTemp **=** cvRect**(**0**,** 0 **,** 1**,** 1**);** // 初始化

getEyeMinRect**(&**eyeRectTemp**,** subhoriProject**,** subvertProject**,** WIDTH**,** HEIGHT**,** 5**,** 3**);**

/\*

printf("eyeRectTemp.y: %d\n", eyeRectTemp.y);

printf("eyeRectTemp.height: %d\n", eyeRectTemp.height);

printf("eyeRectTemp.x: %d\n", eyeRectTemp.x);

printf("eyeRectTemp.width: %d\n", eyeRectTemp.width);

\*/

// 计算最小左眼矩形的长宽比, 判断眼睛状态时用的到

lMinEyeballRectShape **=** **(**double**)**eyeRectTemp**.**width **/** **(**double**)**eyeRectTemp**.**height**;**

//printf("\nlMinEyeballRectShape: %f\n", lMinEyeballRectShape);

cvSetImageROI**(**lEyeballImg**,** eyeRectTemp**);** // 设置ROI为检测到最小面积的眼眶

lMinEyeballImg **=** cvCreateImage**(**cvGetSize**(**lEyeballImg**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**lEyeballImg**,** lMinEyeballImg**,** **NULL);**

cvResetImageROI**(**lEyeballImg**);**

cvShowImage**(**"lMinEyeballImg"**,** lMinEyeballImg**);**

//////////////////////// 统计左眼黑像素个数 /////////////////////

HEIGHT **=** lMinEyeballImg**->**height**;**

WIDTH **=** lMinEyeballImg**->**width**;**

// 分配内存

subhoriProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

subvertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** subhoriProject **==** **NULL** **||** subvertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**subhoriProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**subvertProject **+** i**)** **=** 0**;**

histProject**(**lMinEyeballImg**,** subhoriProject**,** subvertProject**);**

// 统计lEyeballImg中黑色像素的个数

temp **=** 0**;** // 白像素个数

**for(** i **=** 0**;** i **<** WIDTH**;** i **++** **)**

temp **+=** **\*(**subvertProject **+** i**);**

temp **/=** 255**;**

lMinEyeballBlackPixel **=** WIDTH **\*** HEIGHT **-** temp**;**

lMinEyeballBlackPixelRate **=** **(**double**)**lMinEyeballBlackPixel **/** **(**double**)(**WIDTH **\*** HEIGHT**);**

//printf("WIDTH \* HEIGHT: %d\tlMinEyeballBlackSum;%d\n\n", WIDTH \* HEIGHT, lMinEyeballBlackPixel);

//printf("lMinEyeballBlackPixelRate;%f\n\n", lMinEyeballBlackPixelRate);

// 统计lMinEyeballImg中的1/2区域内黑像素的比例

lMinEyeballBeta **=** 0**;**

lMinEyeballBeta **=** calMiddleAreaBlackPixRate**(**subvertProject**,** **&**eyeRectTemp**,** WIDTH**,** HEIGHT**,** lEyeCol**,** lMinEyeballBlackPixel**);**

//printf("lMinEyeballBeta; %f\n\n", lMinEyeballBeta);

////////////////////////////////////右眼

HEIGHT **=** rEyeballImg**->**height**;**

WIDTH **=** rEyeballImg**->**width**;**

// 分配内存

subhoriProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

subvertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** subhoriProject **==** **NULL** **||** subvertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**subhoriProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**subvertProject **+** i**)** **=** 0**;**

histProject**(**rEyeballImg**,** subhoriProject**,** subvertProject**);**

// 计算右眼最小的矩形区域

eyeRectTemp **=** cvRect**(**0**,** 0 **,** 1**,** 1**);**

getEyeMinRect**(&**eyeRectTemp**,** subhoriProject**,** subvertProject**,** WIDTH**,** HEIGHT**,** 5**,** 3**);**

// 计算最小右眼矩形的长宽比，判断眼睛状态时用的到

rMinEyeballRectShape **=** **(**double**)**eyeRectTemp**.**width **/** **(**double**)**eyeRectTemp**.**height**;**

//printf("\nrMinEyeballRectShape: %f\n", rMinEyeballRectShape);

cvSetImageROI**(**rEyeballImg**,** eyeRectTemp**);** // 设置ROI为检测到最小面积的眼眶

rMinEyeballImg **=** cvCreateImage**(**cvGetSize**(**rEyeballImg**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**rEyeballImg**,** rMinEyeballImg**,** **NULL);**

cvResetImageROI**(**rEyeballImg**);**

cvShowImage**(**"rMinEyeballImg"**,** rMinEyeballImg**);**

//////////////////////// 统计右眼黑像素个数 /////////////////////

HEIGHT **=** rMinEyeballImg**->**height**;**

WIDTH **=** rMinEyeballImg**->**width**;**

// 分配内存

subhoriProject **=** **(**int**\*)**malloc**(**HEIGHT **\*** **sizeof(**int**));**

subvertProject **=** **(**int**\*)**malloc**(**WIDTH **\*** **sizeof(**int**));**

**if(** subhoriProject **==** **NULL** **||** subvertProject **==** **NULL** **){**

printf**(**"Failed to allocate memory\n"**);**

cvWaitKey**(**0**);**

**return** **-**1**;**

**}**

// 内存置零

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++)**

**\*(**subhoriProject **+** i**)** **=** 0**;**

**for(**i **=** 0**;** i **<** WIDTH**;** i **++)**

**\*(**subvertProject **+** i**)** **=** 0**;**

histProject**(**rMinEyeballImg**,** subhoriProject**,** subvertProject**);**// 计算直方图积分投影

// 统计lEyeballImg中黑色像素的个数

temp **=** 0**;**

**for(** i **=** 0**;** i **<** WIDTH**;** i **++** **)**

temp **+=** **\*(**subvertProject **+** i**);**

temp **/=** 255**;**

rMinEyeballBlackPixel **=** WIDTH **\*** HEIGHT **-** temp**;**

rMinEyeballBlackPixelRate **=** **(**double**)**rMinEyeballBlackPixel **/** **(**double**)(**WIDTH **\*** HEIGHT**);**

//printf("WIDTH \* HEIGHT: %d\trMinEyeballBlackSum;%d\n\n", WIDTH \* HEIGHT, rMinEyeballBlackPixel);

//printf("rMinEyeballBlackPixelRate; %f\n\n", rMinEyeballBlackPixelRate);

// 统计lMinEyeballImg中的1/2区域内黑像素的比例

rMinEyeballBeta **=** 0**;**

rMinEyeballBeta **=** calMiddleAreaBlackPixRate**(**subvertProject**,** **&**eyeRectTemp**,** WIDTH**,** HEIGHT**,** rEyeCol**,** rMinEyeballBlackPixel**);**

//printf("temp:%d\trMinEyeballBeta; %f\n\n", temp, rMinEyeballBeta);

// 判断眼睛睁闭情况

lEyeState **=** 1**;** // 左眼状态，默认闭眼

rEyeState **=** 1**;** // 右眼状态，默认闭眼

eyeState **=** 1**;** // 眼睛综合状态，默认闭眼

**if(** lMinEyeballBlackPixel **>** 50**)**

lEyeState **=** getEyeState**(**lMinEyeballRectShape**,** lMinEyeballBlackPixelRate**,** lMinEyeballBeta**);**

**else**

lEyeState **=** 1**;**

**if(** rMinEyeballBlackPixel **>** 50**)**

rEyeState **=** getEyeState**(**rMinEyeballRectShape**,** rMinEyeballBlackPixelRate**,** rMinEyeballBeta**);**

**else**

rEyeState **=** 1**;**

**(**lEyeState **+** rEyeState**)** **==** 2 **?** eyeState **=** 1 **:** eyeState**=**0**;**

// 统计眼睛闭合的次数

**if(** eyeState **==** 1 **){**

eyeCloseNum **++;** // 统计 eyeCloseNum 眼睛闭合次数

eyeCloseDuration **++;**

**if(** globalK **==** DETECTTIME**){**

// 检测过程中判断全是闭眼情况，没有睁眼和检测不到人脸的情况

**(**eyeCloseDuration **>** maxEyeCloseDuration**)** **?** maxEyeCloseDuration **=** eyeCloseDuration **:** maxEyeCloseDuration**;**

eyeCloseDuration **=** 0**;**

**}**

**}**

**else{**

**(**eyeCloseDuration **>** maxEyeCloseDuration**)** **?** maxEyeCloseDuration **=** eyeCloseDuration **:** maxEyeCloseDuration**;**

eyeCloseDuration **=** 0**;**

**}**

**}** // 承接判断是否检测到人脸的if语句

/\*

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* 眼睛状态 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("lEyeState: %d\trEyeState: %d\n", lEyeState, rEyeState);

printf("eyeState: %d\n\n\n\n", eyeState);

\*/

// 计时：执行一次循环的时间

stop **=** clock**();**

//printf("run time: %f\n", (double)(stop - start) / CLOCKS\_PER\_SEC);

printf**(**"eyeState: %d\n"**,** eyeState**);**

// 调整循环变量，进入下一次检测过程

**if(** globalK **==** DETECTTIME **){**

printf**(**"\nFATIGUETHRESHOLD\*\*\*\*\*: %d\n"**,** FATIGUETHRESHOLD**);**

printf**(**"eyeCloseNum: %d\tmaxEyeCloseDuration: %d\n"**,** eyeCloseNum**,** maxEyeCloseDuration**);**

printf**(**"failFaceNum: %d\tmaxFailFaceDuration: %d\n"**,** failFaceNum**,** maxFailFaceDuration**);**

// 进行疲劳状态的判别

fatigueState **=** recoFatigueState**(**FATIGUETHRESHOLD**,** eyeCloseNum**,** maxEyeCloseDuration**,** failFaceNum**,** maxFailFaceDuration**);**

**if(** fatigueState **==** 1 **)**

printf**(**"驾驶员处于疲劳驾驶状态\n\n"**);**

**else** **if(** fatigueState **==** 0 **)**

printf**(**"驾驶员处于正常驾驶状态\n\n"**);**

// 进入下一次检测过程前，将变量清零

globalK **=** 0**;**

lEyeState **=** 1**;**

rEyeState **=** 1**;**

eyeState **=** 1**;**

eyeCloseNum **=** 0**;**

eyeCloseDuration **=** 0**;**

maxEyeCloseDuration **=** 0**;**

failFaceNum **=** 0**;**

failFaceDuration **=** 0**;**

maxFailFaceDuration **=** 0**;**

fatigueState **=** 1**;**

char c **=** cvWaitKey**(**0**);**

**if(** c **==** 27 **)**

**break;**

**else**

**continue;**

**}**

**}** // 承接检测过程的 for 循环

// 释放内存

cvDestroyWindow**(**"分割后的人脸"**);**

cvDestroyWindow**(**"大致的左眼区域"**);**

cvDestroyWindow**(**"大致的右眼区域"**);**

cvDestroyWindow**(**"l\_binary"**);**

cvDestroyWindow**(**"r\_binary"**);**

cvDestroyWindow**(**"lEyeImgNoEyebrow"**);**

cvDestroyWindow**(**"rEyeImgNoEyebrow"**);**

cvDestroyWindow**(**"lEyeCenter"**);**

cvDestroyWindow**(**"rEyeCenter"**);**

cvDestroyWindow**(**"lEyeballImg"**);**

cvDestroyWindow**(**"rEyeballImg"**);**

cvDestroyWindow**(**"lkai"**);**

cvDestroyWindow**(**"rkai"**);**

cvDestroyWindow**(**"lMinEyeballImg"**);**

cvDestroyWindow**(**"rMinEyeballImg"**);**

cvReleaseMemStorage**(&**storage**);**

cvReleaseImage**(&**eyeImg**);**

free**(**horiProject**);**

free**(**vertProject**);**

free**(**subhoriProject**);**

free**(**subvertProject**);**

**return** 0**;**

**}**

* 1. 注意之处
* 最佳识别效果的图像大小：500x550，太小了识别效果骤减
* 为了传递人脸检测的序列结果到主函数中，设定了一个外部变量CvSeq\* objectTemp
* 主函数涉及到多个自定义的阈值：根据先验知识分割人眼区域，Ostu阈值减去常数CONST，区分眉毛与眼睛的阈值eyeBrowThreshold，判断眼睛具体位置时用到的中间区域，判断眼睛状态的getEyeState（）中的阈值，
* 5
  1. 待改进之处
* 程序中多次用到了图像增强的算法，理清楚程序的结构，看能不能优化。
* detectFace中有直方图均衡化的代码，看是否需要进行均衡化处理？直方图均衡化对增强比较暗的图像效果很明显。
* 二值化效果有待改进，尤其是CONST的值的确定！直方图均衡化对增强比较暗的图像效果很明显。
* 理清楚主函数中内存的使用情况，尤其是指针变量！
* 自定义的阈值要根据汽车室内的监控图像质量的大小进行最后的调试！

### detectFace()

* 1. 程序的功能

根据Adaboost算法检测出图片中的人脸。

* 1. 程序的思想
  2. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：检测图片中的人脸区域

输入：

IplImage\* srcImg, // 灰度图像

CvMemStorage\* storage, // 存储矩形框的内存区域

double scale\_factor = 1.1, // 搜索窗口的比例系数

int min\_neighbors = 3, // 构成检测目标的相邻矩形的最小个数

int flags = 0, // 操作方式

CvSize min\_size = cvSize(20, 20) // 检测窗口的最小尺寸

输出参数：

CvSeq\* objects // 检测到人脸的矩形框

说明：1. 识别的准确率和速度关键在于cvHaarDetectObject（）函数的参数的调整

2. 如果实际用于汽车内检测效果不佳时，可考虑自己搜集汽车室内图片然后训练分类器

3. 实际用于疲劳驾驶检测时，由于人脸位于图片的中央而且占的面积很大，可以将min\_size和scale\_factor调大一些，加快速度

4. 内含直方图均衡化

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "cv.h"

#include "stdlib.h"

#include "highgui.h"

extern CvSeq**\*** objectsTemp**;** // 传递objects的值会main（）

void detectFace**(**

IplImage**\*** srcImg**,** // 灰度图像

CvSeq**\*** objects**,** // 输出参数：检测到人脸的矩形框

CvMemStorage**\*** storage**,** // 存储矩形框的内存区域

double scale\_factor **=** 1.1**,** // 搜索窗口的比例系数

int min\_neighbors **=** 3**,** // 构成检测目标的相邻矩形的最小个数

int flags **=** 0**,** // 操作方式

CvSize min\_size **=** cvSize**(**20**,** 20**)** // 检测窗口的最小尺寸

**)**

**{**

// 程序用到的参数

const char**\*** cascadeName **=** "haarcascade\_frontalface\_alt2.xml"**;** // 级联分类器的xml文件名

// 读取级联分类器xml文件

CvHaarClassifierCascade**\*** cascade **=** **(**CvHaarClassifierCascade**\*)**cvLoad**(**cascadeName**,** 0**,** 0**,** 0**);**

**if(** **!**cascade **)** **{**

fprintf**(** stderr**,** "ERROR: Could not load classifier cascade\n" **);**

cvWaitKey**(**0**);**

exit**(-**1**);**

**}**

// 直方图均衡

//cvEqualizeHist(srcImg ,srcImg);

// 检测人脸

cvClearMemStorage**(**storage**);**

objects **=** cvHaarDetectObjects**(**

srcImg**,**

cascade**,**

storage**,**

scale\_factor**,**

min\_neighbors**,**

flags**,** /\*CV\_HAAR\_DO\_CANNY\_PRUNING\*/

min\_size

**);**

objectsTemp **=** objects**;** // 为了将objects的值传递回main函数

// 释放cascade的内存

cvReleaseHaarClassifierCascade**(&**cascade**);**

**}**

* 1. 注意之处
  2. 改进之处
* detectFace（）中有直方图均衡化的代码，看是否需要进行均衡化处理？
* 识别的准确率和速度关键在于cvHaarDetectObject（）函数的参数的调整
* 如果实际用于汽车内检测效果不佳时，可考虑自己搜集汽车室内图片然后训练分类器
* 实际用于疲劳驾驶检测时，由于人脸位于图片的中央而且占的面积很大，可以将min\_size和scale\_factor调大一些，加快速度，但要保证准确率！
* 可实现并行运算

### ostuThreshold（）

* 1. 程序功能：

用Ostu最大类间距方差法计算二值化阈值，然后减去自定义常数CONST

* 1. 程序思想：

由于用ostu计算得出的阈值进行二值化时效果不理想，因此考虑减去一个固定值来补偿。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：用Ostu最大类间方差法计算二值化阈值

输入：

hist：图像的直方图数组

pixelSum：图像的像素总和

CONST: 一个常数；为了适应各种特殊的要求，可实现在找到的最优分割阈值的基础上减去该常数

输出：

threshold：最优阈值

Date: 2014.08.14

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include <stdio.h>

int ostuThreshold**(**int **\*** hist**,** int pixelSum**,** const int CONST**)**

**{**

float pixelPro**[**256**];**

int i**,** j**,** threshold **=** 0**;**

//计算每个像素在整幅图像中的比例

**for(**i **=** 0**;** i **<** 256**;** i**++){**

**\*(**pixelPro**+**i**)** **=** **(**float**)(\*(**hist**+**i**))** **/** **(**float**)(**pixelSum**);**

**}**

//经典ostu算法,得到前景和背景的分割

//遍历灰度级[0,255],计算出方差最大的灰度值,为最佳阈值

float w0**,** w1**,** u0tmp**,** u1tmp**,** u0**,** u1**,** u**,**deltaTmp**,** deltaMax **=** 0**;**

**for(**i **=** 0**;** i **<** 256**;** i**++){**

w0 **=** w1 **=** u0tmp **=** u1tmp **=** u0 **=** u1 **=** u **=** deltaTmp **=** 0**;**

**for(**j **=** 0**;** j **<** 256**;** j**++){**

**if(**j **<=** i**){** //背景部分

//以i为阈值分类，第一类总的概率

w0 **+=** **\*(**pixelPro**+**j**);**

u0tmp **+=** j **\*** **(\*(**pixelPro**+**j**));**

**}**

**else** //前景部分

**{**

//以i为阈值分类，第二类总的概率

w1 **+=** **\*(**pixelPro**+**j**);**

u1tmp **+=** j **\*** **(\*(**pixelPro**+**j**));**

**}**

**}**

u0 **=** u0tmp **/** w0**;** //第一类的平均灰度

u1 **=** u1tmp **/** w1**;** //第二类的平均灰度

u **=** u0tmp **+** u1tmp**;** //整幅图像的平均灰度

//计算类间方差

deltaTmp **=** w0 **\*** **(**u0 **-** u**)\*(**u0 **-** u**)** **+** w1 **\*** **(**u1 **-** u**)\*(**u1 **-** u**);**

//找出最大类间方差以及对应的阈值

**if(**deltaTmp **>** deltaMax**){**

deltaMax **=** deltaTmp**;**

threshold **=** i**;**

**}**

**}**

printf**(**"Ostu Threshold: %d\n"**,** threshold**);**

printf**(**"real Threshold: %d\n"**,** threshold **-** CONST**);**

//返回最佳阈值;

**return** **(**threshold **-** CONST**);**

**}**

* 1. 注意之处
* 进行二值化处理之前，先进行了cvSmooth中值滤波处理、nonlineTrans非线性处理
  1. 改进之处
* 由于ostu计算得出的阈值不太符合要求，因此可以尝试其他的阈值选取方法！！
* 寻找动态确定CONST常数的方法，以适应更多不同情况。考虑原图很暗，ostu计算出来的阈值本来就很低，结果还被减去CONST导致阈值太低的情况！还有，由于图像太暗，导致二值化后黑色像素过多的情况。
* 可实现并行运算

### histProject（）

* 1. 程序功能

计算直方图在水平方向和垂直方向的积分投影

* 1. 程序思想

按行累加实现水平方向的积分投影；按列累加实现垂直方向的积分投影。在一次遍历像素点的过程中实现水平和垂直方向的积分投影。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：计算图像直方图在水平方向和垂直方向的投影

输入：

srcImg：源图像

输出：

horiProj: 水平方向的投影结果；1 \* height数组的指针，输入前记得初始化

vertProj：垂直方向的投影结果；1 \* width数组的指针，输入前记得初始化

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "cv.h"

void histProject**(**IplImage **\*** srcImg**,** int**\*** horiProj**,** int**\*** vertProj**)**

**{**

// 程序用到的参数

int i**,** j**;**

uchar**\*** ptr **=** **NULL;** // 指向图像当前行首地址的指针

uchar**\*** temp **=** **NULL;**

int HEIGHT **=** srcImg**->**height**;**

int WIDTH **=** srcImg**->**width**;**

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++){**

ptr **=** **(**uchar**\*)** **(**srcImg**->**imageData **+** i **\*** srcImg**->**widthStep**);**

**for(**j **=** 0**;** j **<** WIDTH**;** j **++){**

temp **=** ptr **+** j**;** // 减少计算量

**\*(**horiProj **+** i**)** **+=** **\***temp**;** // 计算水平方向的投影

**\*(**vertProj **+** j**)** **+=** **\***temp**;** // 计算垂直方向的投影

**}**

**}**

**}**

* 1. 注意之处
* 传递给histProject的图像必须是灰度图像
* 因为涉及到累加运算，所以horiProject和vertProject指针一定要初始化为0！！
  1. 改进之处
* 传递给histProject的图像必须是灰度图像
* 可实现并行运算

### getEyePos（）

* 1. 程序功能：

找出数列中限定区域内的最低点的位置，即找到人眼的位置

* 1. 程序思想

先对直方图积分投影结果进行升序排序，然后找出最小值并且判断是否在设定的中间区域内，如果在则输出index索引值，否则对下一个最小值进行相同判断，直到找到第一个符合条件的最小值，然后返回该最小值的索引index。

* 1. 源码

#include <cv.h>

#include <stdlib.h>

**typedef** struct

**{**

int data**;**

int index**;**

**}**projectArr**;**

// qsort的函数参数

int cmpInc**(** const void **\***a **,**const void **\***b**)**

**{**

**return** **(\*(**projectArr **\*)**a**).**data **-** **(\*(**projectArr **\*)**b**).**data**;**

**}**

int getEyePos**(**int**\*** project**,** int size**,** int region**)**

**{**

// 参数

projectArr**\*** projectStruct **=** **NULL;**

projectArr**\*** projectTemp **=** **NULL;**

int i**,** j**,** pos**,** sizeTemp**,** temp**;**

// 分配projectStruct内存空间

projectStruct **=** **(**projectArr**\*)**malloc**(**size **\*** **sizeof(**projectArr**));**

projectTemp **=** **(**projectArr**\*)**malloc**(sizeof(**projectArr**));**

// 初始化内存空间

**for(**i **=** 0**;** i **<** size**;** i **++){**

**(**projectStruct **+** i**)->**data **=** **\*(**project **+** i**);**

**(**projectStruct **+** i**)->**index **=** i**;**

**}**

// 对project从小到大快速排序

//qsort(projectStruct, size, sizeof(\*project), cmpInc);

**for(**i **=** 0**;** i **<=** size **-** 2**;** i **++){**

**for(** j **=** 0**;** j **<** size **-** i **-** 1**;** j **++** **){**

**if(** **(**projectStruct **+** j**)->**data **>** **(**projectStruct **+** j **+** 1**)->**data **){**

**\***projectTemp **=** **\*(**projectStruct **+** j**);**

**\*(**projectStruct **+** j**)** **=** **\*(**projectStruct **+** j **+** 1**);**

**\*(**projectStruct **+** j **+** 1**)** **=** **\***projectTemp**;**

**}**

**}**

**}**

// 寻找中间区域的最小值及其位置

sizeTemp **=** size **/** 2**;**

temp **=** 0**;**

**for(** i **=** 0**;** i **<** size**;** i **++** **){**

temp **=** **(**projectStruct**+**i**)->**index**;**

**if(** **(**temp **>** sizeTemp **-** region**)** **&&** **(**temp **<** sizeTemp **+** region**)** **){**

pos **=** **(**projectStruct **+** i**)->**index**;**

// 防止指针越界访问位置元素出现负数

**if(** pos **<** 0**)**

**return** **-**1**;**

**break;**

**}**

**else{**

// 防止整个数列不存在符合条件的元素

**if(** i **==** size **-** 1 **)**

**return** **-**1**;**

**}**

**}**

//free(projectStruct);

free**(**projectTemp**);**

**return** pos**;**

**}**

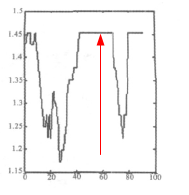
* 1. 注意之处
* projectStruct指针的内存释放有问题
* 升序排序的方法用的是冒泡排序
* 定义了外部变量结构体projectArr
  1. 改进之处
* 用快速排序对数列进行排序，可加快速度！
* 考虑投影值相同但是index不同的情况的处理办法，因为很多时候不能很准确找到中心点就是这个原因。
* 考虑加入左右眼二值化图像的参数，消除头发或者背景等大片黑块对中心点确定的影响！！

### removeEyebrow（）

* 1. 程序功能

搜索积分投影图的最低点，从而消除眉毛

* 1. 程序思想



找到眉毛与眼睛分割的点，然后去除分割点上方的部分，从而消除眉毛。在找分割点时，以3行像素的和为单位进行逐个逐个比较，找到最小的单位。然后以该单位为搜索起点，搜索第一个最高点，然后以该最高点为分割点，即图中箭头位置，去除分割点上方的部分。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：搜索积分投影图中的最低点，从而消除眉毛的函数

输入：

int\* horiProject: 数列的指针

int width: 数列的宽度

int height: 数列的高度

int threshold：分割眉毛的阈值，最多

输出：

返回找到的最低点行位置，结果为int类型，即眉毛与眼睛的分割线

说明：

1. 消除眉毛时可以调整eyeBrowThreshold来调整去除的效果

2. 同时可以调整连续大于阈值的次数count来调整效果。

Date: 2014.08.23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int removeEyebrow**(**int**\*** horiProject**,** int width**,** int height**,** int threshold**)**

**{**

// 参数

int temp**,** temp1**,** count**,** flag**,** i**;**

int eyeRow**;**

int eyeBrowThreshold**;**

// 定位人眼位置

eyeBrowThreshold **=** **(**width **-** threshold**)** **\*** 255**;** // 为了防止无法区分眼睛和眉毛的情况，可适当降低阈值

// 消除眉毛区域

temp **=** 100000000**;**

temp1 **=** 0**;**

count **=** 0**;**

flag **=** 0**;** // 表示当前搜索的位置在第一个最低谷之前

eyeRow **=** 0**;**

**for(**i **=** 0**;** i **<** height**;** i **=** i **+** 3**){**

count **++;**

temp1 **=** **\*(**horiProject **+** i**)** **+** **\*(**horiProject **+** i **+** 1**)** **+** **\*(**horiProject **+** i **+** 2**);**

**if(** **(**temp1 **<** temp**)** **&** **(**flag **==** 0**)** **){**

temp **=** temp1**;**

eyeRow **=** i**;**

count **=** 0**;**

**}**

**if** **(**count **>=** 3 **||** i **>=** height **-** 2**){**

flag **=** 1**;**

**break;**

**}**

**}**

// 搜索第一个大于眼睛与眉毛分割阈值的点

count **=** 0**;**

**for(** i **=** eyeRow**;** i **<** height**;** i **++** **){**

**if(** **\*(**horiProject **+** i**)** **>** eyeBrowThreshold**){**

eyeRow **=** i**;**

count **++;**

**if(** count **>=** 3 **){** // count: 统计共有多少连续的行的投影值大于阈值；

eyeRow **=** i**;**

**break;**

**}**

**}**

**else**

count **=** 0**;**

**}**

// 防止没有眉毛错删眼睛的情况，可根据实验结果调整参数！

**if(** eyeRow **>=** height **/** 2 **)**

eyeRow **=** 0**;**

**return** eyeRow**;**

**}**

* 1. 注意之处
* 消除眉毛时可以调整eyeBrowThreshold来调整去除的效果
* 同时可以调整连续大于阈值的次数count来调整效果。
* 调整单位的像素行数，可以一定程度提高判断的准确率，但是单位太大的话不利于处理比较小的图像。
  1. 改进之处
* 有时间的话可以考虑重新设置函数的变量，使函数更易于阅读
* 根据实际的图像调整参数，使得结果更准确！

### calEyeSocketRegion

* 1. 程序功能

特定功能函数：根据人眼的中心大致计算眼眶的区域，

* 1. 程序思想

以人眼中心为中心，向外扩展直到扩展后的区域为原图区域的1/2大小。超出边界的情况要特殊处理。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：特定功能函数：根据人眼的中心大致计算眼眶的区域

输入：

CvRect\* eyeRect: 眼眶矩形区域的指针

int width: 数列的宽度

int height: 数列的高度

int EyeCol：虹膜中心所在的列位置

int EyeRow：虹膜中心所在的行位置

输出：

以指针的方式返回眼眶的大致区域，eyeRect

说明：

Date: 2014.08.23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void calEyeSocketRegion**(**CvRect**\*** eyeRect**,** int width**,** int height**,** int EyeCol**,** int EyeRow**)**

**{**

// 参数

int temp**,** temp1**;**

temp **=** EyeCol **-** width **/** 4**;**

temp1 **=** EyeRow **-** height **/** 4**;**

**if(** **(**temp **<** 0**)** **&&** **(**temp1 **<** 0**)** **){**

eyeRect**->**x **=** 0**;**

eyeRect**->**width **=** width **/** 2 **+** temp**;**

eyeRect**->**y **=** 0**;**

eyeRect**->**height **=** height **/** 2 **+** temp1**;**

**}**

**else** **if(** **(**temp **<** 0**)** **&&** **(**temp1 **>** 0**)** **){**

eyeRect**->**x **=** 0**;**

eyeRect**->**width **=** width **/** 2 **+** temp**;**

eyeRect**->**y **=** temp1**;**

eyeRect**->**height **=** height **/** 2**;**

**}**

**else** **if(** **(**temp **>** 0**)** **&&** **(**temp1 **<** 0**)** **){**

eyeRect**->**x **=** temp**;**

eyeRect**->**width **=** width **/** 2**;**

eyeRect**->**y **=** 0**;**

eyeRect**->**height **=** height **/** 2 **+** temp1**;**

**}**

**else** **if(** **(**temp **>** 0**)** **&&** **(**temp1 **>** 0**)** **){**

eyeRect**->**x **=** temp**;**

eyeRect**->**width **=** width **/** 2**;**

eyeRect**->**y **=** temp1**;**

eyeRect**->**height **=** height **/** 2**;**

**}**

**}**

* 1. 注意之处
  2. 改进之处
* 有时间的话可以考虑重新设置函数的变量，使函数更易于阅读
* 根据实际的图像看是否需要调整当前比例！

### gerEyeMinRect（）

* 1. 程序功能

消除眼睛区域周边的白色区域，计算人眼最小的矩形区域。

* 1. 程序思想

从上下左右想中心搜索，如果搜索到有黑色像素的行或者列则停止搜索，并记录该处位置，从而得到最小的人眼区域。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：特定功能函数：计算人眼最小的矩形区域

输入：

CvRect\* eyeRect: 人眼最小的矩形区域的指针

int\* horiProject

int\* vertProject

int width: 数列的宽度

int height: 数列的高度

int horiThreshold：水平方向的阈值

int vertThreshold：垂直方向的阈值

输出：

通过指针返回CvRect\* eyeRect: 人眼最小的矩形区域的指针

说明：

1.

2.

Date: 2014.08.23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void getEyeMinRect**(**CvRect**\*** eyeRect**,** int**\*** horiProject**,** int**\*** vertProject**,** int width**,** int height**,** int horiThreshold**=**5**,** int vertThreshold**=**3**)**

**{**

// 参数

int temp**,** temp1**,** i**;**

temp1 **=** **(**width **-** horiThreshold**)** **\*** 255**;**

**for(**i **=** 0**;** i **<** height**;** i **++){**

**if(** **\*(**horiProject **+** i**)** **<** temp1 **){**

eyeRect**->**y **=** i**;**

**break;**

**}**

**}**

temp **=** i**;** // 记录eyeRectTemp.y的位置

printf**(**"eyeRectTemp->y: %d\n"**,** eyeRect**->**y**);**

**if(** temp **!=** height **){**

// temp != HEIGHT: 防止没有符合\*(subhoriProject + i) < temp1条件的位置；如果temp != HEIGHT则一定有满足条件的位置存在

**for(**i **=** height**-**1**;** i **>=** 0**;** i **--){**

**if(** **\*(**horiProject **+** i**)** **<** temp1 **){**

temp **=** i**;**

**break;**

**}**

**}**

**if(** temp **==** eyeRect**->**y **)**

eyeRect**->**height **=** 1**;**

**else**

eyeRect**->**height **=** temp **-** eyeRect**->**y**;**

**}**

**else{**

eyeRect**->**height **=** 1**;**

**}**

printf**(**"eyeRectTemp.height: %d\n"**,** eyeRect**->**height**);**

temp1 **=** **(**height **-** vertThreshold**)** **\*** 255**;**

**for(** i **=** 0**;** i **<** width**;** i **++** **){**

**if(** **\*(**vertProject **+** i**)** **<** temp1 **){**

eyeRect**->**x **=** i**;**

**break;**

**}**

**}**

temp **=** i**;** // 记录eyeRectTemp.x的位置

printf**(**"eyeRectTemp.x: %d\n"**,** eyeRect**->**x**);**

**if(** temp **!=** width **){**

**for(**i **=** width**-**1**;** i **>=** 0**;** i **--){**

**if(** **\*(**vertProject **+** i**)** **<** temp1 **){**

temp **=** i**;**

**break;**

**}**

**}**

// 防止宽度为0，显示图像时出错！

**if(** temp **==** eyeRect**->**x **)**

eyeRect**->**width **=** 1**;**

**else**

eyeRect**->**width **=** temp **-** eyeRect**->**x**;**

**}**

**else{**

eyeRect**->**width **=** 1**;**

**}**

printf**(**"eyeRectTemp.width: %d\n"**,** eyeRect**->**width**);**

**}**

* 1. 注意之处
* 内涵调试用的输出语句，转化为硬件代码时记得删除调试语句。
  1. 改进之处
* 有时间的话可以考虑重新设置函数的变量，使函数更易于阅读

### lineTrans（）

* 1. 程序功能

对图像进行线性点运算，实现图像增强效果

* 1. 程序思想

遍历像素点，对每个像素点根据线性方程重新计算像素值。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：对图像进行线性点运算，实现图像增强

输入：

IplImage\* srcImg: 源灰度图像

float a：乘系数a

float b：常系数b

输出：

IplImage\* dstImg：输出经过线性变换后的图像

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "cv.h"

#include "highgui.h"

void lineTrans**(**IplImage**\*** srcImg**,** IplImage**\*** dstImg**,** float a**,** float b**)**

**{**

int i**,** j**;**

uchar**\*** ptr **=** **NULL;** // 指向图像当前行首地址的指针

uchar**\*** pixel **=** **NULL;** // 指向像素点的指针

float temp**;**

dstImg **=** cvCreateImage**(**cvGetSize**(**srcImg**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**srcImg**,** dstImg**,** **NULL);**

int HEIGHT **=** dstImg**->**height**;**

int WIDTH **=** dstImg**->**width**;**

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++){**

ptr **=** **(**uchar**\*)** **(**srcImg**->**imageData **+** i **\*** srcImg**->**widthStep**);**

**for(**j **=** 0**;** j **<** WIDTH**;** j **++){**

pixel **=** ptr **+** j**;**

// 线性变换

temp **=** a **\*** **(\***pixel**)** **+** b**;**

// 判断范围

**if** **(** temp **>** 255 **)**

**\***pixel **=** 255**;**

**else** **if** **(**temp **<** 0**)**

**\***pixel **=** 0**;**

**else**

**\***pixel **=** **(**uchar**)(**temp **+** 0.5**);**// 四舍五入

**}**

**}**

**}**

* 1. 注意之处
  2. 改进之处
* 转到硬件时可以用查表的方式实现相同的效果！
* 可实现并行运算

### nonlineTrans（）

* 1. 程序功能

对图像进行非线性点运算，实现图像增强效果

* 1. 程序思想

遍历像素点，对每个像素点根据非线性方程重新计算像素值。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：对图像进行线性点运算，实现图像增强

输入：

IplImage\* srcImg: 源灰度图像

float a：乘系数a

输出：

IplImage\* dstImg：输出经过线性变换后的图像

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "cv.h"

#include "highgui.h"

#include "cv.h"

void nonlineTrans**(**IplImage**\*** srcImg**,** IplImage**\*** dstImg**,** float a**)**

**{**

int i**,** j**;**

uchar**\*** ptr **=** **NULL;** // 指向图像当前行首地址的指针

uchar**\*** pixel **=** **NULL;** // 指向像素点的指针

float temp**;**

dstImg **=** cvCreateImage**(**cvGetSize**(**srcImg**),** IPL\_DEPTH\_8U**,** 1**);**

cvCopy**(**srcImg**,** dstImg**,** **NULL);**

int HEIGHT **=** dstImg**->**height**;**

int WIDTH **=** dstImg**->**width**;**

**for(**i **=** 0**;** i **<** HEIGHT**;** i **++){**

ptr **=** **(**uchar**\*)** **(**srcImg**->**imageData **+** i **\*** srcImg**->**widthStep**);**

**for(**j **=** 0**;** j **<** WIDTH**;** j **++){**

pixel **=** ptr **+** j**;**

// 非线性变换

temp **=** **\***pixel **+** **(**a **\*** **(\***pixel**)** **\*** **(**255 **-** **\***pixel**))** **/** 255**;**

// 判断范围

**if** **(** temp **>** 255 **)**

**\***pixel **=** 255**;**

**else** **if** **(**temp **<** 0**)**

**\***pixel **=** 0**;**

**else**

**\***pixel **=** **(**uchar**)(**temp **+** 0.5**);**// 四舍五入

**}**

**}**

**}**

* 1. 注意之处
  2. 改进之处
* 转到硬件时可以用查表的方式实现相同的效果！
* 可实现并行运算

### recoEyeState（）

* 1. 程序功能

通过模糊综合评价的思想对指标进行分级，然后组合成一个函数，通过计算当前眼睛的函数值与阈值比较，从而判断眼睛的状态。

* 1. 程序思想

根据最终提取出的人眼图像判断眼睛的睁开、闭合情况，可转化为判断评价问题，即根据现有的人眼数据，判断眼睛的状态。由于3个评价的指标评判眼睛状态的界限不太清晰，因此可通过模糊评价的方法对不同范围的指标划分等级，然后再将三个指标加权组合在一起。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 判断眼睛状态 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：通过模糊综合评价的思想判断眼睛的状态

输入：

double MinEyeballRectShape：眼睛矩形区域的长宽比

double MinEyeballBlackPixelRate：眼睛矩形区域黑像素点所占的比例

double MinEyeballBeta：眼睛中心1/2区域黑色像素点占总黑像素点的比例

输出：

返回人眼睁开闭合的状态0：睁开，1：闭合

说明：

1. 三个输入参数的阈值是自己设定的

2. 输出的结果参数的阈值需要调整

3. 为了转硬件方便，加快运算速度，将浮点运算转为了整数运算。

Date: 2014.08.22

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

int getEyeState**(**double MinEyeballRectShape**,** double MinEyeballBlackPixelRate**,** double MinEyeballBeta**)**

**{**

int eyeState**;**

int funcResult**;**

int shapeFuzzyLv**,** pixelFuzzyLv**,** betaFuzzyLv**;** // 三个参数对应的模糊级别的值

// 判定眼睛矩形区域的长宽比的模糊级别

shapeFuzzyLv **=** 0**;**

**if(** **(**MinEyeballRectShape **>=** 0**)** **&&** **(**MinEyeballRectShape **<=** 0.8**)** **)**

shapeFuzzyLv **=** 0**;**

**else** **if(** MinEyeballRectShape **<=** 1.2 **)**

shapeFuzzyLv **=** 2**;**

**else** **if(** MinEyeballRectShape **<=** 1.5 **)**

shapeFuzzyLv **=** 6**;**

**else** **if(** MinEyeballRectShape **<=** 2.5 **)**

shapeFuzzyLv **=** 8**;**

**else** **if(** MinEyeballRectShape **<=** 3 **)**

shapeFuzzyLv **=** 6**;**

**else**

shapeFuzzyLv **=** 0**;**

// 判定眼睛矩形区域黑像素点所占比例的模糊级别

pixelFuzzyLv **=** 0**;**

**if(** **(**MinEyeballBlackPixelRate **>=** 0**)** **&&** **(**MinEyeballBlackPixelRate **<=** 0.4**)** **)**

pixelFuzzyLv **=** 0**;**

**else** **if(** MinEyeballBlackPixelRate **<=** 0.50 **)**

pixelFuzzyLv **=** 2**;**

**else** **if(** MinEyeballBlackPixelRate **<=** 0.60 **)**

pixelFuzzyLv **=** 6**;**

**else** **if(** MinEyeballBlackPixelRate **<=** 1 **)**

pixelFuzzyLv **=** 8**;**

// 判定眼睛中心1/2区域黑色像素点占总黑像素点的比例的模糊级别

betaFuzzyLv **=** 0**;**

**if(** **(**MinEyeballBeta **>=** 0**)** **&&** **(**MinEyeballBeta **<=** 0.3**)** **)**

betaFuzzyLv **=** 0**;**

**else** **if(** MinEyeballBeta **<=** 0.45 **)**

betaFuzzyLv **=** 2**;**

**else** **if(** MinEyeballBeta **<=** 0.6 **)**

betaFuzzyLv **=** 6**;**

**else** **if(** MinEyeballBeta **<=** 1 **)**

betaFuzzyLv **=** 8**;**

// 模糊评价函数

eyeState **=** 1**;** // 默认是闭眼的

funcResult **=** 2 **\*** shapeFuzzyLv **+** 4 **\*** pixelFuzzyLv **+** 4 **\*** betaFuzzyLv**;**

**if(** funcResult **>=** 58 **)**

eyeState **=** 0**;**

**return** eyeState**;**

**}**

* 1. 注意之处
* 三个输入参数的阈值和模糊评价函数阈值都是自己设定的
* 为了转硬件方便，加快运算速度，将浮点运算转为了整数运算，即将百分数扩大了十倍。
  1. 改进之处
* 使用更客观的方法确定加权系数和等级分数！
* 可根据实际的图像，调整相应的参数与阈值。

### recoFatigueState（）

* 1. 程序功能

在一次检测过程完成后，根据闭眼总次数、连续闭眼最大值、未检测到人脸的总次数、连续未检测到人脸的最大值这四个因素，判断是否处于疲劳驾驶状态！

* 1. 程序思想

利用logistic方程分别构造四个因素对疲劳程度判断的函数方程，然后利用查表的方式计算出每个因素的贡献值，最后根据贡献值总和与阈值的比较得出结论。

* 1. 源码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

功能：特定功能函数——根据眼睛闭合状态和是否检测到人脸

判断驾驶状态：正常？疲劳？

输入：

int eyeCloseNum：检测过程中眼睛闭状态的总次数

int maxEyeCloseDuration：检测过程中眼睛连续闭合的最大次数

int failFaceNum：检测过程中未检测到人脸的总次数

int maxFailFaceDuration：检测过程中连续未检测到人脸的最大次数

说明：

Data：2014.09.22

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int eyeCloseNumTab**[]** **=** **{**2**,**2**,**4**,**6**,**9**,**14**,**20**,**29**,**39**,**50**,**61**,**72**,**80**,**86**,**91**,**94**,**96**,**98**,**98**,**99**,**99**,**100**,**100**,**100**,**100**,**100**,**100**,**100**,**100**,**100**,** 100**};**

int eyeCloseDurationTab**[]** **=** **{**2**,** 4**,** 9**,** 18**,** 32**,** 50**,** 68**,** 82**,** 91**,** 95**,** 98**,** 99**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**};**

int failFaceDurationTab**[]** **=** **{**2**,** 6**,** 14**,** 29**,** 50**,** 71**,** 86**,** 94**,** 98**,** 99**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**,** 100**};**

int recoFatigueState**(**int thresh**,** int eyeCloseNum**,** int maxEyeCloseDuration**,** int failFaceNum**,** int maxFailFaceDuration**)**

**{**

int eyeCloseValue**;** // 眼睛闭合次数的贡献值

int eyeCloseDurationValue**;** // 眼睛连续闭合次数的贡献值

int failFaceValue**;** // 未检测到人脸的总次数的贡献值

int failFaceDurationValue**;** // 连续未检测到人脸的贡献值

int compreValue**;** // 综合贡献值

// 查表计算四个指标的贡献值

eyeCloseValue **=** eyeCloseNumTab**[**eyeCloseNum**];**

eyeCloseDurationValue **=** eyeCloseDurationTab**[**maxEyeCloseDuration**];**

failFaceValue **=** eyeCloseNumTab**[**failFaceNum**];**

failFaceDurationValue **=** failFaceDurationTab**[**maxFailFaceDuration**];**

// 综合贡献值

compreValue **=** eyeCloseValue **+** eyeCloseDurationValue **+** failFaceValue **+** failFaceDurationValue**;**

printf**(**"\neyeCloseValue: %d\n"**,** eyeCloseValue**);**

printf**(**"eyeCloseDurationValue: %d\n"**,** eyeCloseDurationValue**);**

printf**(**"failFaceValue: %d\n"**,** failFaceValue**);**

printf**(**"failFaceDurationValue: %d\n"**,** failFaceDurationValue**);**

printf**(**"compreValue: %d\n\n"**,** compreValue**);**

**return** **(**compreValue **>=** thresh**)** **?** 1 **:** 0**;**

**}**

* 1. 注意之处
* 判断按是否处于疲劳驾驶状态的阈值 FATIGUETHRESHOLD 是自己设定的
  1. 改进之处
* 让每个因素的贡献值函数更加适合、精确。
* 根据实验确定更精确的阈值

## 项目的限制

1. 基本只能使用于白天光线较好的时候，夜晚无法使用
2. 戴眼镜的情况无法使用
3. 低头情况下，人脸检测的效果很差

## 项目改进方向

1. 调试参数：使用类似级联滤波器的调试方法，即逐级调试，使得每一级的输出效果都是最佳的！
2. 将所有阈值定义为常量
3. 变量太多，有些变量可重复使用的，但是为了方便阅读，定了更多变量，所以转硬件的时候可以最大程度的利用变量，较少变量数量。另外，功能类似的变量可以考虑用结构体整合到一起！
4. 低头时人脸检测的准确率很低
5. 人眼状态识别时，闭眼的情况识别不准确，很多时候将闭眼识别为睁开状态，可以考虑自己一个睁眼和闭眼的模板数列，然后比较人眼积分投影数列与模板数列的相似度。
6. 从二值化时候就分开左右眼进行处理能适应更多特殊情况，比如左右脸亮度相差太大的情况！
7. 可转化为函数的部分

* 消除眉毛的部分，放到getEyePos模块中
* 判断人眼睁闭状态中计算以人眼中心为中心的大致眼眶的模块，放到getEyePos模块中
* 计算最小眼睛的矩形区域中的确定最小眼睛区域eyeRectTemp的模块，放到getEyePos模块中
* 统计lMinEyeballImg中的1/2区域内黑像素的比例的模块，放到recoEyeState模块中

1. 模糊综合评价的模型可已选择突出主要因素的模型，指标的分数和权重可考虑用更客观的方式确定。
2. \* 对投影曲线进行递推滤波(消除毛刺影响)

\* 对于很暗的情况先灰度均衡化，然后非线性运算，用查表方式

\* 在缩小至最小眼球之前用中值滤波或者形态学处理，消除独立小

黑块的影响

1. 对疲劳状态的判断：用数据分析的方法对采集的多组数据不断的进行分析，看数据是否有明显的上升趋势，从而判断驾驶员是否处于疲劳驾驶状态。另外，还可以考虑才采用概率论的假设检验的方法判断是否处于疲劳驾驶状态。
2. 特殊情况

* 人眼区域的边界有大片黑块，造成人眼中心定位不准确，如何去除边界大块区域？



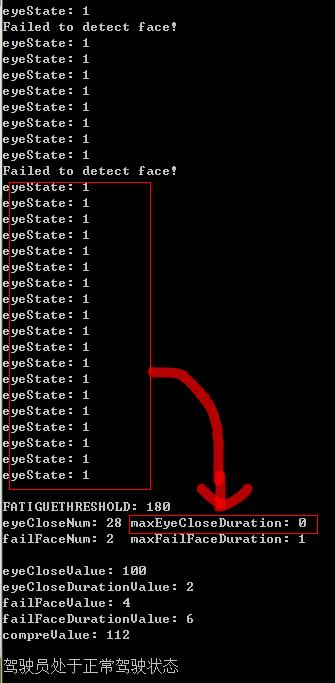
* 左右脸光照不均匀的情况二值化效果严重不准确！



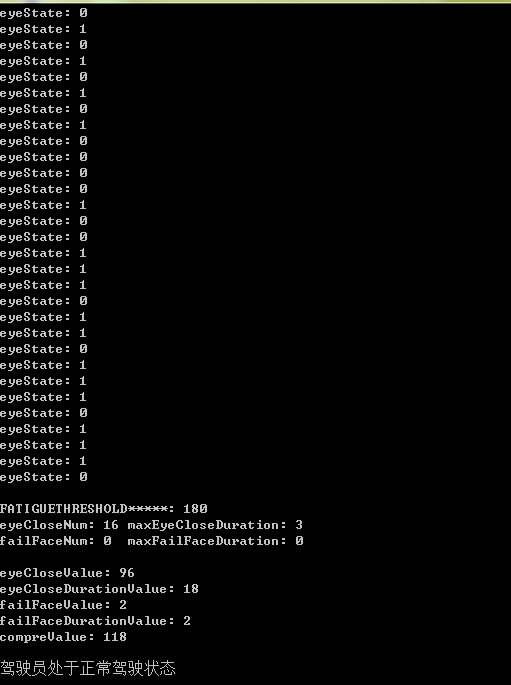
* 眼睛的
* 撒旦

疲劳状态检测的特殊情况

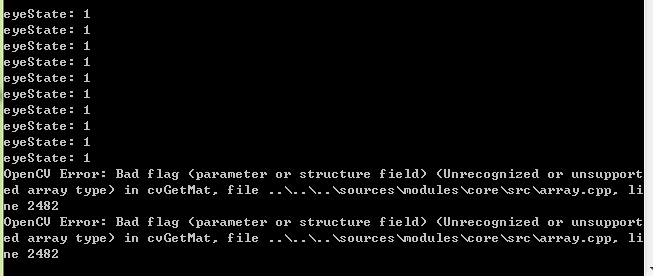
1. 检测过程中判断全是闭眼和检测不到人脸的情况，没有睁开眼的情况，导致maxEyeCloseDuration = 0;

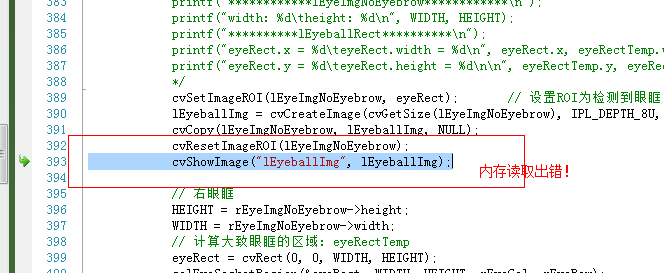


1. 眨眼与闭眼的频率很相近，即一次眨眼一次闭眼的情况，使得疲劳判断结果为正常！



1. 当判断为全1的时候，程序运行出现内存读取错误！





分析：原因不明，但是肯定和lEyeballImg 和 EyeCloseDuration有关。重点查看EyeCloseDuration一直增加不跳出的时候，lEyeballImg处的程序如何运行。