FatigueDrivingReco 的程序文档

一、 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）

二、 程序内容

1. main（）函数

a) 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。

b) 程序的思想

c) 程序的源码

d) 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

e) 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！

2. detectFace()

a) 程序的功能

根据 Adaboost 算法检测出图片中的人脸。

b) 程序的思想

c) 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);


}
```

d) 注意之处

e) 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

3. ostuThreshold（）

a) 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

b) 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。

c) 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);

}
```

d) 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

e) 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

4. histProject（）

a) 程序功能

计算直方图在水平方向和垂直方向的积分投影

b) 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。

c) 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
        }
    }


}
```

d) 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!!

e) 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

5. getEyePos（）

a) 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

b) 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。

c) 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;

}
```

d) 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

e) 改进之处

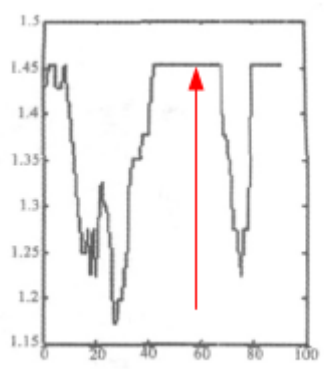
- 用快速排序对数列进行排序，可加快速度！
- 考虑投影值相同但是 index 不同的情况的处理办法，因为很多时候不能很准确找到中心点就是这个原因。
- 考虑加入左右眼二值化图像的参数，消除头发或者背景等大片黑块对中心点确定的影响!!

6. removeEyebrow（）

a) 程序功能

搜索积分投影图的最低点，从而消除眉毛

b) 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。

c) 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;
}
```

d) 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。

e) 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!

7. calEyeSocketRegion

a) 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，

b) 程序思想

以人眼中心为中心，向外扩展知道区域为原图的 1/2 区域大小。超出边界的情况要特殊处理。

c) 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;
    }

}
```

d) 注意之处

e) 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！

8. gerEyeMinRect（）

a) 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。

b) 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。

c) 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);

}
```

d) 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。

e) 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

9. lineTrans（）

a) 程序功能

对图像进行线性点运算，实现图像增强效果

b) 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。

c) 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
        }
    }

}
```

d) 注意之处

e) 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

10. nonlineTrans（）

a) 程序功能

对图像进行非线性点运算，实现图像增强效果

b) 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。

c) 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
        }
    }

}
```

d) 注意之处

e) 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

11. recoEyeState（）

a) 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。

b) 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。

c) 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;

}
```

d) 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。

e) 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。

三、 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

四、 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. 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！
3. 低头时人脸检测的准确率很低
4. 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。
5. 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！
6. 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
7. 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。
8. 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