“软件质量管理”技术报告



项目：**基于opencv的疲劳驾驶检测系统**

**作者姓名 姚翔，段瑞凯，张望望**

**指导教师 刘晋**

**学科(专业) 电子信息（软件工程）**

**所在学院 信息工程学院**

**提交日期 二零二一年六月**

# 引言

## 1.1 动机

随着社会经济的不断发展，人们的生活水平不断提升，机动车持有量不断增加，但与此同时这也给我们的生活带来了潜在的危害，交通事故的发生频率越来越大，给人们带来了巨大的人身安全隐患和经济损失。

调查研究显示，疲劳是影响驾驶员安全驾驶的一个重要因素。驾驶员疲劳时，会出现视线模糊、腰酸背疼、动作呆板、手脚发胀或有精力不集中、反应迟钝、思考不周全、精神涣散、焦虑、急躁等现象。如果仍勉强驾驶车辆，则很可能导致交通事故的发生。据不完全统计全世界每年发生的交通事故高达10 多亿次，而由于驾驶员的疲劳驾驶引起的事故约占总事故件数的 20% ～30%，且疲劳驾驶造成的事故死亡率占所有交通事故死亡率的 70% 左右。故近年来，驾驶员疲劳驾驶问题已受到世界各国越来越多研究人员的关注，其中针对疲劳驾驶检测方法而进行的研究更具重要的现实意义。

由此可见，如果疲劳驾驶的问题不得到很好的解决，机动车驾驶事故依旧没有办法得到控制，会给社会带来很大的危害。随着我国经济的快速发展,汽车已成为主要的交通工具,疲劳驾驶已成为引发交通事故的主要原因之一.因此,研究疲劳驾驶检测系统,实现对驾驶员疲劳状态的监控具有重要意义.在各种疲劳检测方法中,主要有基于传感设备的接触式检测,如对驾驶员的脑电图、心电图等生理信号进行检测和基于模式识别、图像处理的非接触式检测.目前,国外在这方面的研究主要偏向第 1 种方法,利用各种传感器采集驾驶员的生理数据进行疲劳检测,例如美国研制的方向盘监视装置 S.A.M,通过安装的传感器检测方向盘的异常来判断驾驶员疲劳情况,欧洲研发的 AWAKE 驾驶诊断系统,利用 CCD 传感器和力学传感器获取的信号以判断驾驶员是否疲劳. 由于第 2 种方法具有非侵入性、比较自然、实时的特点,已成为驾驶员疲劳检测领域的研究热点.

现有的基于计算机视觉的疲劳检测机制中通常都是在对人眼精确定位后利用眼睛单一特征进行分析,判定出人眼状态.该类算法易受外界环境干扰,如光线、眼镜等因素,误判率高,具有一定的局限性.经查阅相关文献，疲劳在人体面部表情中表现出大致三个类型：打哈欠（嘴巴张大且相对较长时间保持这一状态）、眨眼（或眼睛微闭，此时眨眼次数增多，且眨眼速度变慢）、点头（瞌睡点头）。

## 1.2 要解决的问题

为实现疲劳驾驶检测工程，本系统要提供以下几个功能：

（1）通过 CAM9100-U 摄像头模块采集视频

（2）直方图均衡，光线补偿，中值滤波，阈值分割

（3）使用基于 Haar 特征的 AdaBoost 算法对待检图像进行人脸定位

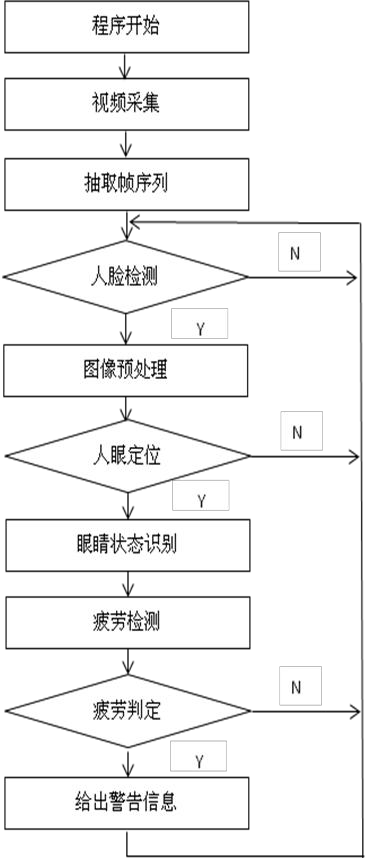
（4）对二值化后的人脸图进行水平灰度积分投影得到上下眼睑坐标

（5）计算眼睑距离，根据 P80 来判断驾驶员是否处于疲劳

（6）使用波形图进行报警，预留功能是声光报警

# 系统框架/流程

2.1工作流程图 本系统的工作流程如图 3.2 所示。程序开始时，摄像头开始工作，并开始实时抓拍，驾驶员图像帧序列，处理一帧时，首先，进行人脸检测以缩小眼睛定位的区域、然后，对抓拍的图像进行灰度化、直方图均衡化、光照补偿、中值滤波、阈值分割等预处理；接着，使用水平投影进行人眼定位；最后，根据 P80 进行疲劳判别。

图2.1 疲劳驾驶检测流程图

2.2 灰度化 在色彩空间 RGB 模型中，若红、绿、蓝三个分量相等时，则彩色表示一种灰度

颜色，那么这个相同的值就是灰度值，所以，在计算机中，灰度图像的每一个像素

用一个字节就可以存储，其范围为 0-255。

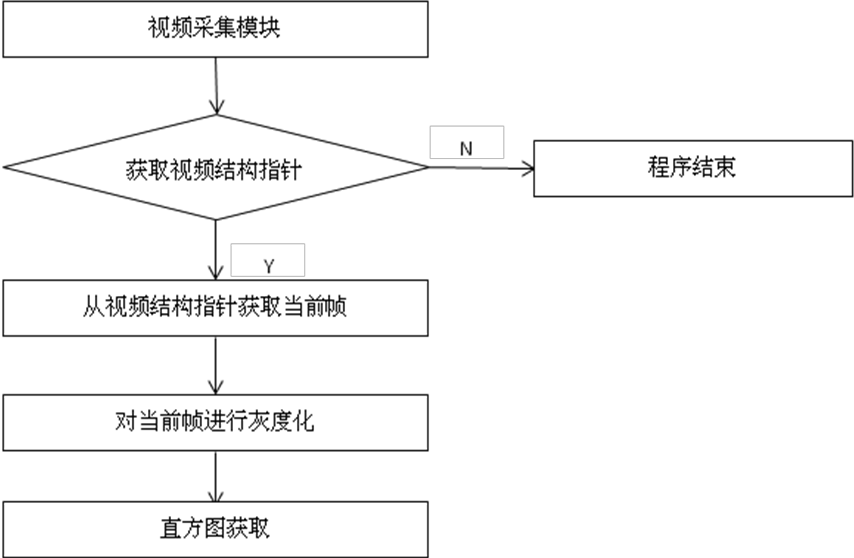


图2.2 灰度化流程图

2.3求直方图 在对彩色图片进行灰度化之后，就要进行直方图提取。直方图为灰度级的频率函数，表示的是灰度图中从低到高每一灰度级像素的频率或每一灰度级的像素个数。横坐标为灰度级，纵坐标是该像素出现的频率或者该灰度级出现的像素个数，这样的分布就是直方图，描述的是图像灰度分布的情况。在 OpenCV 中，我们使用 cvCreateHist 创建直方图。

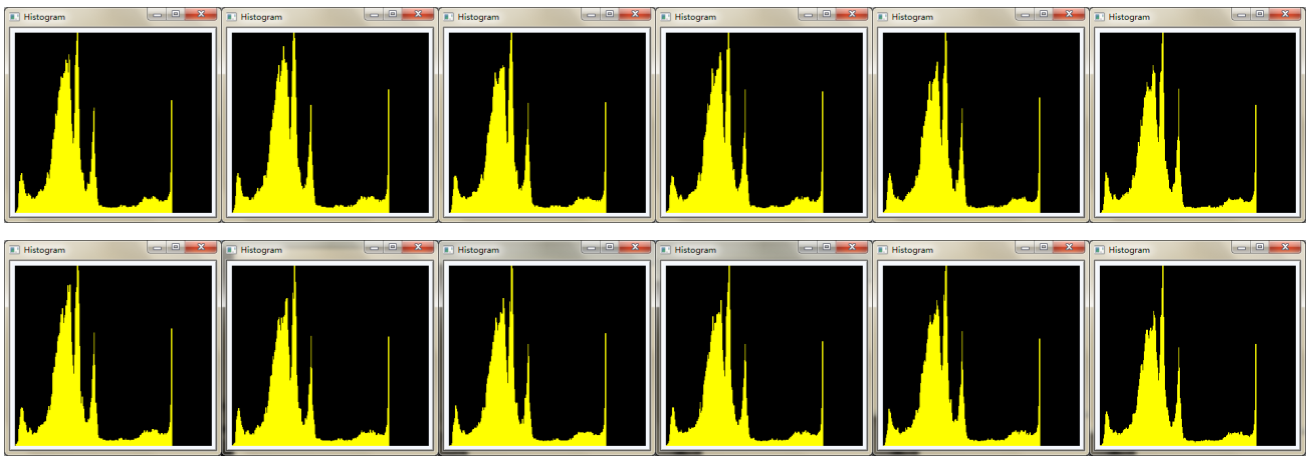


图2.3 直方图

# 特征提取

## 3.1 数据获取与特征提取

下载开源数据集

shape\_predictor\_68\_face\_landmarks.dat

官方下载地址：<https://pypi.org/simple/dlib/>

基于dlib库对人脸特征进行提取，在视频流中抓取人脸特征、并保存为64x64大小的图片文件。

注意的是：因为我们后面会对人脸数据集进行训练识别，因此，这一步非常重要。

光线——曝光和黑暗图片因手动剔除

摄像头的清晰度也比较重要——在哪台笔记本识别，就要在那台笔记本做数据集采集，我用了同学在其他笔记本采取的数据，因为电脑配置，在后面的训练中出现不能识别或错误识别的情况，因此，尽量同一设备——采取数据集和做人脸识别。

我们采用的是视频流截图，也可以用爬虫在百度图片上进行爬取

（1）提取特征点

图3-1-1为68个特征数据点显示效果

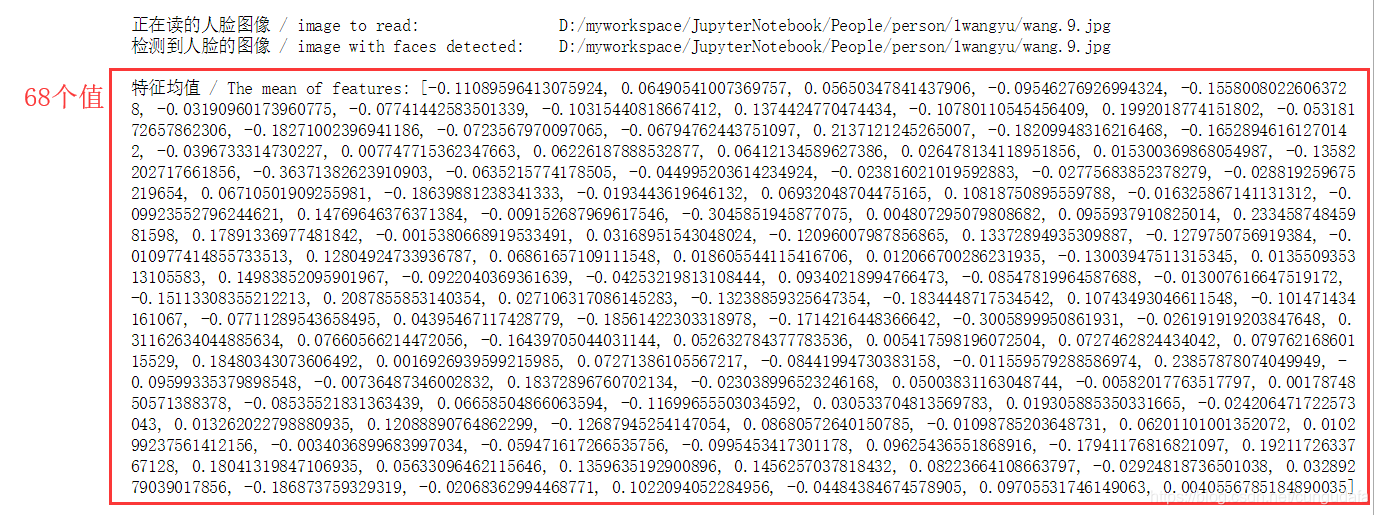


图3-2-1 直方图

（2）获取特征数据集写入csv

根据数据集，模型----训练----》68个特征数据

写入csv，如图3-1-2

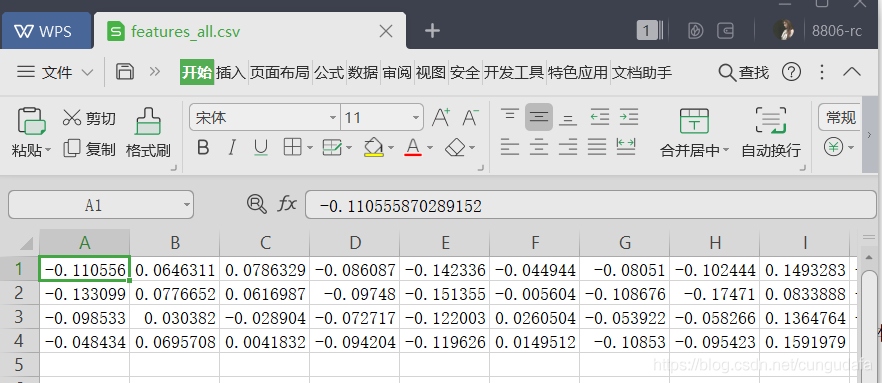


图3-2-2 直方图

（3）计算特征数据集的欧氏距离作对比

图中为我的运行效果：选取最接近（欧氏距离最小）的值，标注为person 1，如图3-1-3

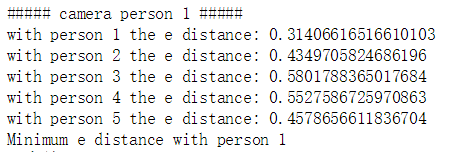


图3-2-3 直方图

获取特征点并保存数据源码：get\_features.py

# 从人脸图像文件中提取人脸特征存入 CSV

# Features extraction from images and save into features\_all.csv

# return\_128d\_features() 获取某张图像的128D特征

# compute\_the\_mean() 计算128D特征均值

from cv2 import cv2 as cv2

import os

import dlib

from skimage import io

import csv

import numpy as np

# 要读取人脸图像文件的路径

path\_images\_from\_camera = "D:/myworkspace/JupyterNotebook/People/person/"

# Dlib 正向人脸检测器

detector = dlib.get\_frontal\_face\_detector()

# Dlib 人脸预测器

predictor = dlib.shape\_predictor("D:/myworkspace/JupyterNotebook/People/model/shape\_predictor\_68\_face\_landmarks.dat")

# Dlib 人脸识别模型

# Face recognition model, the object maps human faces into 128D vectors

face\_rec = dlib.face\_recognition\_model\_v1("D:/myworkspace/JupyterNotebook/People/model/dlib\_face\_recognition\_resnet\_model\_v1.dat")

# 返回单张图像的 128D 特征

def return\_128d\_features(path\_img):

img\_rd = io.imread(path\_img)

img\_gray = cv2.cvtColor(img\_rd, cv2.COLOR\_BGR2RGB)

faces = detector(img\_gray, 1)

print("%-40s %-20s" % ("检测到人脸的图像 / image with faces detected:", path\_img), '\n')

# 因为有可能截下来的人脸再去检测，检测不出来人脸了

# 所以要确保是 检测到人脸的人脸图像 拿去算特征

if len(faces) != 0:

shape = predictor(img\_gray, faces[0])

face\_descriptor = face\_rec.compute\_face\_descriptor(img\_gray, shape)

else:

face\_descriptor = 0

print("no face")

return face\_descriptor

# 将文件夹中照片特征提取出来, 写入 CSV

def return\_features\_mean\_personX(path\_faces\_personX):

features\_list\_personX = []

photos\_list = os.listdir(path\_faces\_personX)

if photos\_list:

for i in range(len(photos\_list)):

# 调用return\_128d\_features()得到128d特征

print("%-40s %-20s" % ("正在读的人脸图像 / image to read:", path\_faces\_personX + "/" + photos\_list[i]))

features\_128d = return\_128d\_features(path\_faces\_personX + "/" + photos\_list[i])

# print(features\_128d)

# 遇到没有检测出人脸的图片跳过

if features\_128d == 0:

i += 1

else:

features\_list\_personX.append(features\_128d)

else:

print("文件夹内图像文件为空 / Warning: No images in " + path\_faces\_personX + '/', '\n')

# 计算 128D 特征的均值

# N x 128D -> 1 x 128D

if features\_list\_personX:

features\_mean\_personX = np.array(features\_list\_personX).mean(axis=0)

else:

features\_mean\_personX = '0'

return features\_mean\_personX

# 读取某人所有的人脸图像的数据

people = os.listdir(path\_images\_from\_camera)

people.sort()

with open("D:/myworkspace/JupyterNotebook/People/feature/features2\_all.csv", "w", newline="") as csvfile:

writer = csv.writer(csvfile)

for person in people:

print("##### " + person + " #####")

# Get the mean/average features of face/personX, it will be a list with a length of 128D

features\_mean\_personX = return\_features\_mean\_personX(path\_images\_from\_camera + person)

writer.writerow(features\_mean\_personX)

print("特征均值 / The mean of features:", list(features\_mean\_personX))

print('\n')

print("所有录入人脸数据存入 / Save all the features of faces registered into: D:/myworkspace/JupyterNotebook/People/feature/features\_all2.csv")

# 分类算法/匹配算法

## 4.1 视频采集模块

视频采集主要是通过本系统自己的 CAM8100-U 摄像头对受摄者进行实时数据采集，本系统中，摄像头距离受摄者大约 0.5 米左右，这与实际环境驾车环境中，驾驶员座位和理论摄像头安装位置是吻合的。

视频采集模块的工程流程是，程序开始后，检测有无摄像头，若有，则进入图像预处理模块，若无则给出错误信息。

模块流程图如图 4-1 所示：

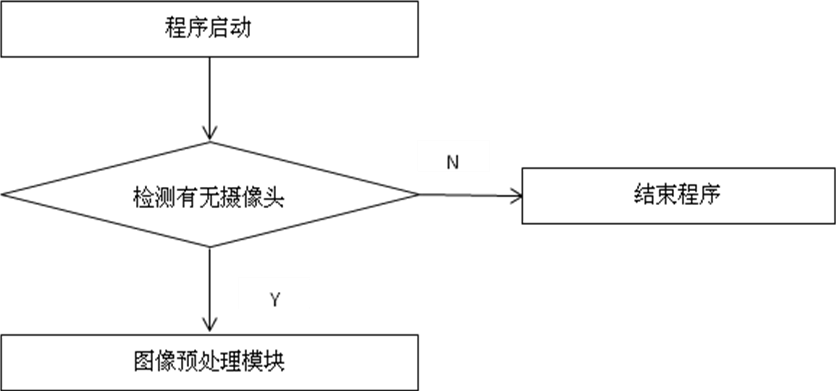


图 4-1 视频采集流程

## 4.2 人脸检测

本系统采用人脸检测算法是基于 Haar 特征的 AdaBoost 算法，这个算法的第一步是对人脸样本的选取，人脸样本的选取我们是基于如下理由考虑的，在实际的驾驶环境中，首先往往车主是同一人，那么训练的是同一个人在不同情况下不同表情下的人脸，并且背景很有可能会相同，比方说，靠背坐骑，以及后排座椅和装饰等， 所以我们在本系统中选取的是 YALE 人脸数据库和在实验室环境下模拟的驾驶员图片作为正样本，另外由于受硬件条件的影响，比如说 CPU 的处理能力，AdaBoost 算法的训练时间会有所不同，而负样本是拍摄的任意一组非人脸图片。其大小为24 26 。正负样本示例如图 4-2-1、4-2-2 所示：



图 4-2-1 训练用人脸图像



图 4-2-2 训练用非人脸图像

最后得到的层叠分类器由 17 个强分类器组成，如表 4-2-1 所示：

表 4-2-1 训练的层叠分类器特征数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 层数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 特征数 | 5 | 7 | 8 | 10 | 13 | 13 | 17 | 21 | 31 | 16 | 17 | 17 | 19 | 14 | 20 | 15 | 19 | 13 |

在第二章中，我们已经讲过层叠分类器的组成及迭代的过程，通过实验的结果，更加证明了之前所述，即，在最开始的阶段，大部分的负样本，即非人脸都会因为不满足层叠分类器的特征函数，输出的阈值过高，所以，在初始的几层迭代中，约 80%以上的负样本即非人脸都会被排除掉，同时，由于随着迭代的深入，强分类器的阈值也会越来越高，这样，提取的人脸特征也会越来越接近真实值，最后输出的结果就是定位出的真实人脸。

YALE 人脸数据库，来自于耶鲁大学的人脸采集项目组，通过征集志愿者的人脸图像，来实现多人脸多表情的目的，丰富了图形图像处理的数据。来自此数据库的 15 个人的 10 种不同表情的人脸共 150 张组成，图片大小为11292（单位：像素）。其中一些图片：

图 4-2-3 YALE 数据库人脸图像测试集

实验室环境下拍摄的人脸图片，一共 125 张，大小为640 480 （单位：像素）。部分图片：

图 4-2-4 拍摄人脸图像测试集

对二个测试集，利用训练好的分类器进行人脸检测。测试结果如下：

表 4-2-2 人脸检测实验结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试集 | 图像数 | 人脸数 | 正检 | 误检 | 正检率 | 误检率 | 耗时(毫秒) |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 人脸数 | 人脸数 |  |  |  |
| YALE | 150 | 150 | 138 | 2 | 0.92 | 0.01 | 3.1 |
| 拍摄人脸 | 125 | 125 | 103 | 7 | 0.82 | 0.05 | 9.2 |

在表 4-2-2 的结果中，我们可以看出，AdaBoost 的正检率是比较高的，这跟选取的人脸样本以及迭代的强分类器是相关的，当人脸样本包含的信息较丰富、人脸的光照信息条件比较优越以及倾斜度较小的情况下，检测出的人脸是很准确的，同时耗时也非常少，反之，当选取的样本集比较单调时，并且光照不均和有遮挡物等情况下，搜索的条件就会越来越复杂，是的不仅在最初阶段的人脸检测出现偏差和漏检，并且致使末尾阶段的错检率也会显著增加。

从表 4-2-2 的耗时来看，对于11292 背景不复杂、光照条件较优越的目标图像， 平均检测耗时为 3.1ms，亦在相同的摄像头的硬件条件下，设置为 24 帧/s 的速度是没有问题的，基本上能够满足实际驾驶情况的需要。并且，对于640 480 尺寸的相同背景的图片进行的实验结果表明，平均检测时间为 9.2ms。

图 4-2-5 YALE 的一些实验结果：



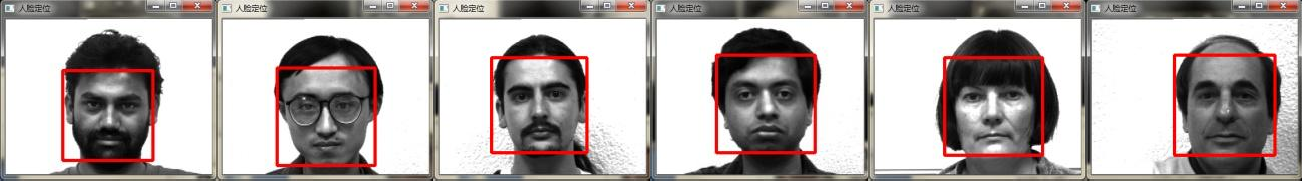


图 4-2-5 YALE 的AdaBoost 人脸检测效果图

## 4.3 图像处理

图像预处理的目的是为了获取RGB 图像的灰度图用来计算后面所有算法的初始输入，并且通过图像预处理可以使得被灰度化的图像能够比较真实地接近原图的水平，损失的图像信息能够比较好地被恢复，同时尽量减少了噪声点和光照的消极影响。接着是在灰度图的基础上进行二值化，使用的方法是 OTSU 算法。OTSU 算法选择类间方差最大的灰度值为阈值，具有很好的效果。计算流程如下：

1. 计算直方图并归一化直方图
2. 计算图像灰度平均值 avgValue
3. 计算直方图的零阶 W[i]和一级矩 U[i]
4. 计算并找到最大的类间方差(between-class，variance)

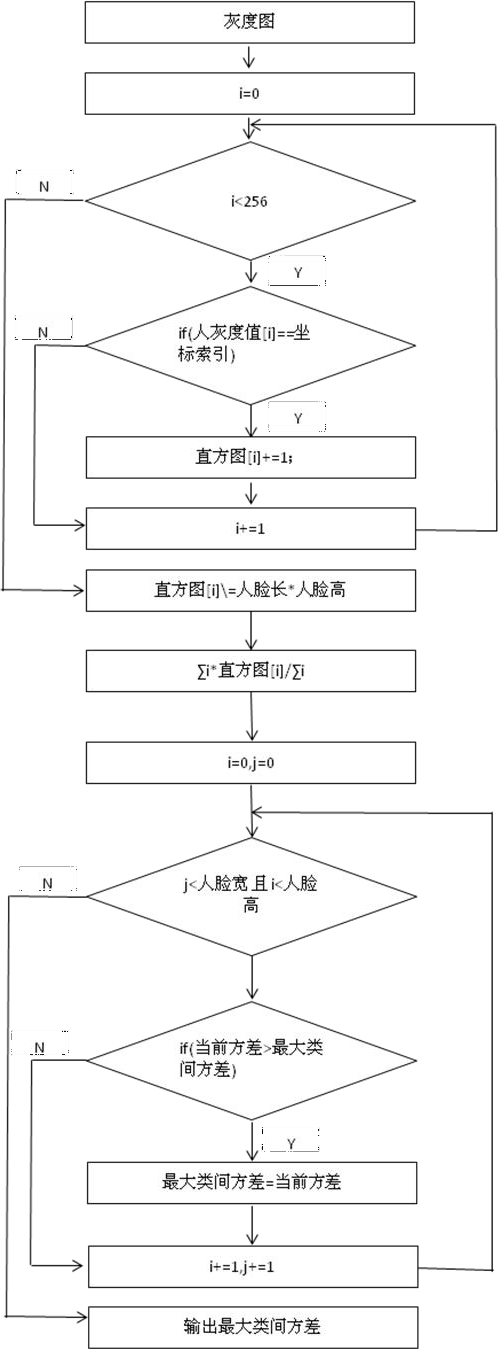
*variance*[*i*]  (*avgValue* *w*[*i*]  *u*[*i*]) (*avgValue* *w*[*i*]  *u*[*i*]) / (*w*[*i*](1 *w*[*i*])) 对次最大方差的灰度值即为目标阈值。程图见图 4-3-1：

图 4-3-1 OTSU 自适应阈值

*J*

（中值和方差i直方）2

最大类间方差（J）= i=0 ，J=0,1,2,...,255

*J J*

直方 (1 直方)

i=0

通过 OTSU 算法可以动态地获取每一帧每一时刻的最佳阈值分割，图 4-3-2 是对上面图进行自适应 OTSU 二值化的效果图：

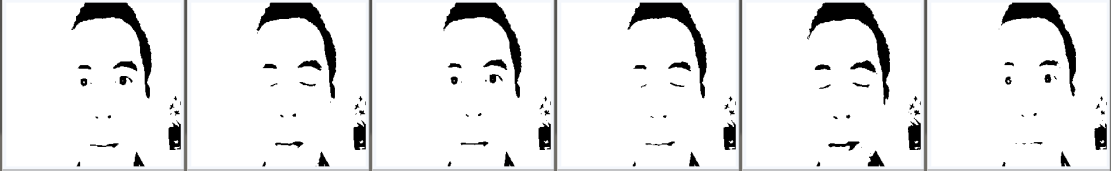




图 4-3-2 OTSU 算法二值图

## 4.4 人眼定位

本系统中人眼的定位依靠水平投影，根据眼睛的特点，在横坐标上，眉毛和眼睛的横坐标大致相同，这样的话，在水平投影上，波谷最多的区域基本上可以确定为眼部的横坐标区域；同样的道理，由于，在垂直投影上，波谷最多的那个区域可以认为是眼部的纵坐标。

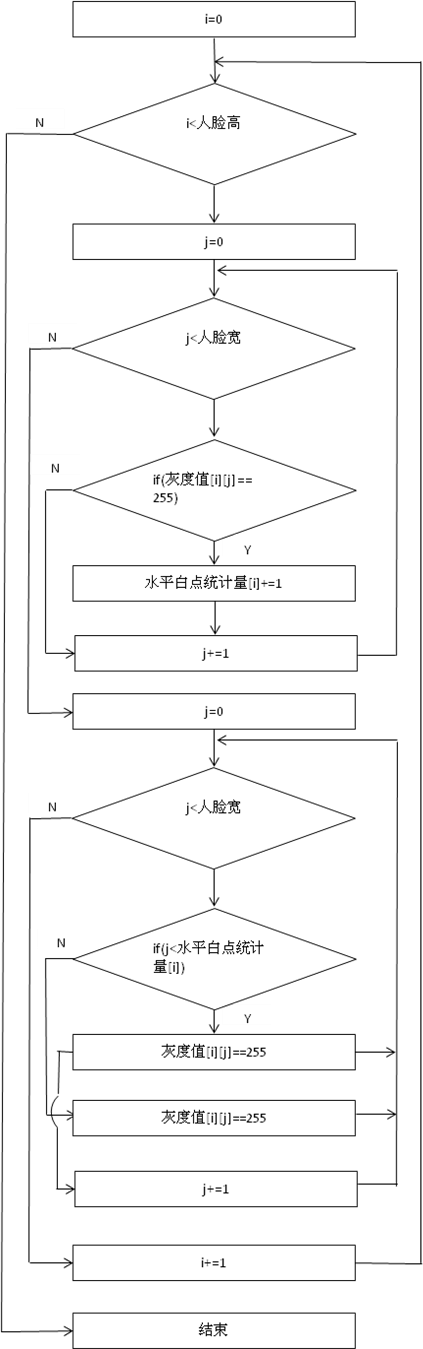
程序流程见图 4-4-1：

图 4-4-1 水平投影

表 4-4-1 为人眼定位检测结果：

表 4-4-1 人眼定结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试集 | 人眼数 | 正检数 | 误检数 | 正检率 | 误检率 | 每双眼平均  定位时间（ms） |
| YALE | 150\*2=300 | 286 | 14 | 0.95 | 0.05 | 189 |
| 拍摄人脸 | 125\*2=250 | 226 | 26 | 0.904 | 0.096 | 146 |

从表 4-4-1 可知，对于背景相对比较简单的 YALE 人脸数据库来讲，人眼的定位正确率很高，这是因为如前所述，在光照均匀的条件下，由人眼的几何特征而进行的水平投影具有比较良好的效果，而且噪声点也相对较少，所以计算的时间也相应地能够满足实时性要求。对于拍摄人脸则有些不同，虽然正确率也很高，可是相对来讲，背景复杂了，而且有偏移等干扰因素的存在也降低了人眼的正确定位率。

## 4.5 人眼状态检测

人眼定位是在检测到的人脸的基础上进行的，而人眼的定位，也是在前面已经检测到人脸上之后完成的。而状态的检测是根据定位出来的人眼的眼睑距离来确定的，在水平投影中定位出来的人眼的坐标中包含上下眼睑的坐标，闭合度大小=上下眼睑的距离差可以得到。

主要源码如下：# -\*- coding: utf-8 -\*-

# import the necessary packages

from scipy.spatial import distance as dist

from imutils.video import FileVideoStream

from imutils.video import VideoStream

from imutils import face\_utils

import numpy as np # 数据处理的库 numpy

import argparse

import imutils

import time

import dlib

import cv2

def eye\_aspect\_ratio(eye):

# 垂直眼标志（X，Y）坐标

A = dist.euclidean(eye[1], eye[5])# 计算两个集合之间的欧式距离

B = dist.euclidean(eye[2], eye[4])

# 计算水平之间的欧几里得距离

# 水平眼标志（X，Y）坐标

C = dist.euclidean(eye[0], eye[3])

# 眼睛长宽比的计算

ear = (A + B) / (2.0 \* C)

# 返回眼睛的长宽比

return ear

# 定义两个常数

# 眼睛长宽比

# 闪烁阈值

EYE\_AR\_THRESH = 0.2

EYE\_AR\_CONSEC\_FRAMES = 3

# 初始化帧计数器和眨眼总数

COUNTER = 0

TOTAL = 0

# 初始化DLIB的人脸检测器（HOG），然后创建面部标志物预测

print("[INFO] loading facial landmark predictor...")

# 第一步：使用dlib.get\_frontal\_face\_detector() 获得脸部位置检测器

detector = dlib.get\_frontal\_face\_detector()

# 第二步：使用dlib.shape\_predictor获得脸部特征位置检测器

predictor = dlib.shape\_predictor('D:/myworkspace/JupyterNotebook/fatigue\_detecting/model/shape\_predictor\_68\_face\_landmarks.dat')

# 第三步：分别获取左右眼面部标志的索引

(lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"]

(rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"]

# 第四步：打开cv2 本地摄像头

cap = cv2.VideoCapture(0)

# 从视频流循环帧

while True:

# 第五步：进行循环，读取图片，并对图片做维度扩大，并进灰度化

ret, frame = cap.read()

frame = imutils.resize(frame, width=720)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# 第六步：使用detector(gray, 0) 进行脸部位置检测

rects = detector(gray, 0)

# 第七步：循环脸部位置信息，使用predictor(gray, rect)获得脸部特征位置的信息

for rect in rects:

shape = predictor(gray, rect)

# 第八步：将脸部特征信息转换为数组array的格式

shape = face\_utils.shape\_to\_np(shape)

# 第九步：提取左眼和右眼坐标

leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

# 第十步：构造函数计算左右眼的EAR值，使用平均值作为最终的EAR

leftEAR = eye\_aspect\_ratio(leftEye)

rightEAR = eye\_aspect\_ratio(rightEye)

ear = (leftEAR + rightEAR) / 2.0

# 第十一步：使用cv2.convexHull获得凸包位置，使用drawContours画出轮廓位置进行画图操作

leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)

cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

# 第十二步：进行画图操作，用矩形框标注人脸

left = rect.left()

top = rect.top()

right = rect.right()

bottom = rect.bottom()

cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 3)

'''

分别计算左眼和右眼的评分求平均作为最终的评分，如果小于阈值，则加1，如果连续3次都小于阈值，则表示进行了一次眨眼活动

'''

# 第十三步：循环，满足条件的，眨眼次数+1

if ear < EYE\_AR\_THRESH:# 眼睛长宽比：0.2

COUNTER += 1

else:

# 如果连续3次都小于阈值，则表示进行了一次眨眼活动

if COUNTER >= EYE\_AR\_CONSEC\_FRAMES:# 阈值：3

TOTAL += 1

# 重置眼帧计数器

COUNTER = 0

# 第十四步：进行画图操作，68个特征点标识

for (x, y) in shape:

cv2.circle(frame, (x, y), 1, (0, 0, 255), -1)

# 第十五步：进行画图操作，同时使用cv2.putText将眨眼次数进行显示

cv2.putText(frame, "Faces: {}".format(len(rects)), (10, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.putText(frame, "Blinks: {}".format(TOTAL), (150, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.putText(frame, "COUNTER: {}".format(COUNTER), (300, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.putText(frame, "EAR: {:.2f}".format(ear), (450, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

print('眼睛实时长宽比:{:.2f} '.format(ear))

if TOTAL >= 50:

cv2.putText(frame, "SLEEP!!!", (200, 200),cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 0, 255), 2)

cv2.putText(frame, "Press 'q': Quit", (20, 500),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (84, 255, 159), 2)

# 窗口显示 show with opencv

cv2.imshow("Frame", frame)

# if the `q` key was pressed, break from the loop

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# 释放摄像头 release camera

cap.release()

# do a bit of cleanup

cv2.destroyAllWindows()

## 4.6 报警模块

报警模块主要是根据疲劳状态来鉴定。在此，我们做出了一个心电图，横坐标是时间轴，纵坐标是人眼的闭合度，绿线是横坐标，红线部分是警戒线，也就是 P80 的值，这个值随不同的数据计算而来，比方说上下眼睑的距离为 1cm，那么这里就是 1\*0.8=0.8。当上下眼睑距离连续 3 秒之内位于 P80 下方时，给出报警信息。

# 运行结果

## 5.1 系统运行环境

硬件：cpu主存为8GB，amd4000处理器，影驰1080显卡

环境：Win10、Python3.7、anaconda3、JupyterNotebook

技术：Opencv：图像处理

Dlib：一个很经典的用于图像处理的开源库，shape\_predictor\_68\_face\_landmarks.dat是一个用于人脸68个关键点检测的dat模型库，使用这个模型库可以很方便地进行人脸检测，并进行简单的应用。

Numpy：基于Python的n维数值计算扩展。

Imutils ：一系列使得opencv 便利的功能，包括图像旋转、缩放、平移，骨架化、边缘检测、显示matplotlib 图像（imutils.opencv2matplotlib(image）。

wxFromBuilder： 可视化UI界面

## 5.2 运行结果

本实验的测试环境是模拟真实驾驶环境的实验室中进行的，摄像头放在受摄者约 0.5 米处，同时，对受摄者的一些活动加以限制，比方说在真实的驾驶

环境中，一般头部的活动范围会处于上下 30 度，左右在 30 度以内。摄像头的参数是采集的图片格式为 JPG，每秒采集 24 帧，大小设为 640 480 ，单次测试的时间间隔大约是 1 分钟，采集到的 24 601共 1440 帧图像。在报警信息中， 检测的人脸和定位的人眼均使用红框标注，波形图记录的是随时间推移，人眼上下眼睑的距离值，对于每一时刻的人眼眼睑距离计算其 PERCLOS 值，若小于 80%，则报警。实验结果如表 5-2-1 所示。

表 5-2-1 疲劳检测实验结果

|  |  |  |  |
| --- | --- | --- | --- |
| 眼睛闭合总帧数 | 眼睛开总帧数 | PERCLOS | 是否疲劳 |
| 999 | 441 | 69 | 否 |
| 1200 | 240 | 83 | 是 |
| 600 | 840 | 41 | 否 |
| 1140 | 300 | 79 | 否 |
| 264 | 876 | 18 | 否 |
| 699 | 741 | 49 | 否 |
| 1301 | 139 | 90 | 是 |
| 1082 | 358 | 75 | 否 |
| 457 | 983 | 31 | 否 |
| 777 | 663 | 54 | 否 |

## 5.3 结果分析

本文采用基于 Haar 特征的 AdaBoost 级联分类器算法实现人脸检测，对人脸的灰度图进行水平投影来确定人眼眼睑的距离，并由此来确定驾驶员是否处于疲劳状态。这个方法有以下一些特点：

* + 1. 比较准确地定位人眼

由表 4-3 可以看出，水平投影定位人眼比较准确。由于之前所述的人眼的分布符合正常人的五官分布，并且是在原始图像的基础上先检测出人脸然后再定位人眼， 相当于缩小了搜索范围，所以相对来讲精度比较高。

* + 1. 比较快速的定位人眼

由表 5-1 可以看出，定位人眼所付出的计算开销和时间开销是很少的，这是因为首先 AdaBoost 算法能够很精确地定位出人脸，缩小了人眼定位的范围，大大节省了人眼水平投影的计算量，另外就是光照条件相对较好，为图像的二值化创造了一个比较好的前提条件，从而在投影阶段，有干扰的部分和噪声点都比较少。

* + 1. 人眼整体状态的识别率较高

这是因为如前所述，人眼眼睑在能够很好地得到定位的情况下，那么根据 P80 就能很好地算出人眼眼睑距离是否处于疲劳状态，可是由前面的实验还是可以说明， 若光照环境等条件没有降到无法接受的地步的话，对人眼闭合的正检率依然可以保持一个比较高的水平。

可是，值得注意是单眼的疲劳判别可能还有比较大的误差，最后对双眼状态的判别精准度在很大程度还是要取决于对单眼状态的检测率。在测试集中，使单眼检测状态较低的主要原因是：

1. 人眼先天性单眼比较小

对于某些测试的人来讲，他们本身眼睛就不大，所以在拍摄的过程采集的图片也会表现得比较小，这种情况当然属于个例，可能在实际环境中，我们也无法很准确地定位他的眼睛，所以在这种情况下，是很难通过计算机来判断是否属于疲劳状态。

1. 相似区域产生干扰

在二值化后进行水平投影的时候，由于有些人的某些特殊情况，比如说，女性的刘海遮住了眼部，或者某些男性的眼镜颜色偏浓，都会导致投影时产生多个大波峰，这样，就会错把刘海、眼睛定位为人眼。

1. 算法存在的缺陷

由于水平投影要求二值化后的图具有良好的边缘特性，因此，如果定位的人脸不准确，或者范围偏大的话，都会扩大投影的范围，加大计算量不说，更会产生额外的噪声点，所以人眼精确定位的前提是精确地检测人脸，因此，实际驾驶环境的的疲劳状态等于人眼疲劳状态正检率和人脸正检率的乘积。人脸检测对人脸定位若过窄或过宽，都会对人眼的正检率和状态分析产生重要影响。

# 结论和展望

本文通过 CCD 摄像头抓拍机动车驾驶员的人脸图像，通过基于 Haar 特征的AdaBoost 的级联分类器算法来对人脸进行检测，随后根据检测出的人脸，缩小人眼定位的范围，根据对人脸二值图的水平投影来判断人的上下眼睑的距离，并根据 P80 来判断驾驶员是否处于疲劳程度。说明在模拟的真实的驾驶环境的实验室中，该算法可以比较好地的将正检率与检测速率结合在一块，基本在实际驾驶环境下对疲劳检测的实时性要求。可是由于前述的本系统的一些缺点导致人脸检测的误检、漏检以及人眼定位时的众多干扰因素的存在也会导致人眼的定位不准确，就实际投入运行来讲，还有以下一些改进应该是可以完成的，具体讲，包含下述几点：

1. 提高检测人脸的正确率

由于基于 Haar 特征的 AdaBoost 级联分类器算法对于前期的人脸样本的训练非常重要，这直接关系到人脸检测的准确与否，所以对于基于 Haar 特征的 AdaBoost 人脸检测的方法，可以从下述几点入手：

一是迭代时分类器的权重可以适当增加，二是搜索的步长也可以适当减少。

1. 配合使用别的人脸检测算法

由于皮肤对光照比较敏感，通过配合使用肤色检测的方法，能够增加迭代分类器的阈值，使得提取的人脸特征更加准确。不过若联合使用肤色检测法有诸如可能会增加计算量等缺点，因此要权衡使用。

1. 改善所抓拍图像的质量

在实际驾驶环境中，摄像头抓拍的人脸图像可以说是第一手详细资料，因此如何改善和提高图片质量始终能够对后续工作产生积极影响。还有一种情况需要考虑的就是如果使用红外摄像头的话，对红外摄像头的去噪将会是一个极大的困难，虽然理论上，红外线能够照亮夜间散发红外线的物体，通过此方法，可以在晚间比较准确地找到瞳孔，但是引入的噪声点确是异常多。

1. 嵌入式平台的移植

实验的目的最终是将实验的结果最终移植到实际环境下，这就要求我们下一步在改善系统功能和性能的同时，要构筑在嵌入式平台下的疲劳驾驶检测系统。

# 参考文献

1. 毛品, 初秀民, 严新平等. 汽车驾驶员驾驶疲劳监测技术研究进展. 中国安全科学报, 2005, 15(3): 108-112
2. NTHSA. Droswy Driver and Automobile Report.
3. Kauichi S. Fatigue and Drowsiness Predictor for Pilots and Air Traffic controllers: 45[th] Annual ATCA Conference. 2000: 22-26
4. Mimuro T, Miiehi Y, Maemura T, Hayafune k. Mitsubishi Advanced Vehicle (ASV). Proeeedings of 4[th] world congress on Intelligent Transport System, Berlin, Germany1997.
5. EllenM. A, Graee. R, Steinfeld. A. A User-Centered Drowsy-Driver Detection and Warning System, Proceedings of ACM Designing User Experience 2003: 109-113.
6. Bill Council. Third-Party Testing and Stirrings of the New Software Engineering.IEEE Software, 1999, 16(6): 76-79
7. Yang J and Waibel A. A real-time facetacker. Proc. 3rd IEEE Workshop on Application of computer vision, 1996: 142-147
8. Shi J, Wu YP. Identification and Analysis of Driver’s Active Safety Factors Journal of Shanghai Jiaotong University. 2000, 34(4): 27-29
9. 杜志刚, 蒋宏, 潘晓东. 眼动仪在道路交通安全与环境评价中的应用. 第三届全国公路科技创新高层论坛论文集(下册). 2006
10. Yang M. H, Ahuja N, Kriegman D, A survey on face detection methods. http://vision. ai. uiuc. edu/yang/papers/survey. ps. gz
11. 吴春莹. 浅谈汽车驾驶员疲劳监测技术的研究进展. 机电一体化, 2007, 13(4): 43-47
12. FHWA. Eye-activity measures of fatigue and napping as a fatigue counter measure.Technical Report No. MC-99-028, 1999
13. 朱夏君. 人脸识别的研究及眼睛定位算法: [硕士学位论文]. 杭州: 浙江工业大学图书馆, 2003