基于 CNN 的人脸识别与虹膜实现疲劳驾驶检测系统

前言：基于目前的完成情况和检索的有关文献资料与数据集情况，综合考虑项目难易程度后决定将项目更改为人脸识别+疲劳检测；目前已完成疲劳检测和 python 实现人脸识别。

该方案设计思想：采用 PS+PL 架构实现方案设计，该项目可分为三个方面进行，采用 AN5642 双目摄像头+HDMI 显示架构实现；主要分为基于 CNN 卷积神经网络实现人脸识别+形态学定位人眼区域实现疲劳检测构成。

目录

一、人脸检测

该部分设计的创新点：

（1） 采用 YES 颜色空间进行肤色检测初步定位人脸位置，相比于采用 YCBCR 颜色空间而言能更好的排除复杂背景干扰。

（2） 采用三步人脸区域定位法使得检测效果更佳；第一步的 YES 初定位，第二步的区域内 R 分量优化人脸特征，第三步的唇色检测提升人脸检测准确率。

对比图

方法一：基于改进的 YCBCR 阈值空间

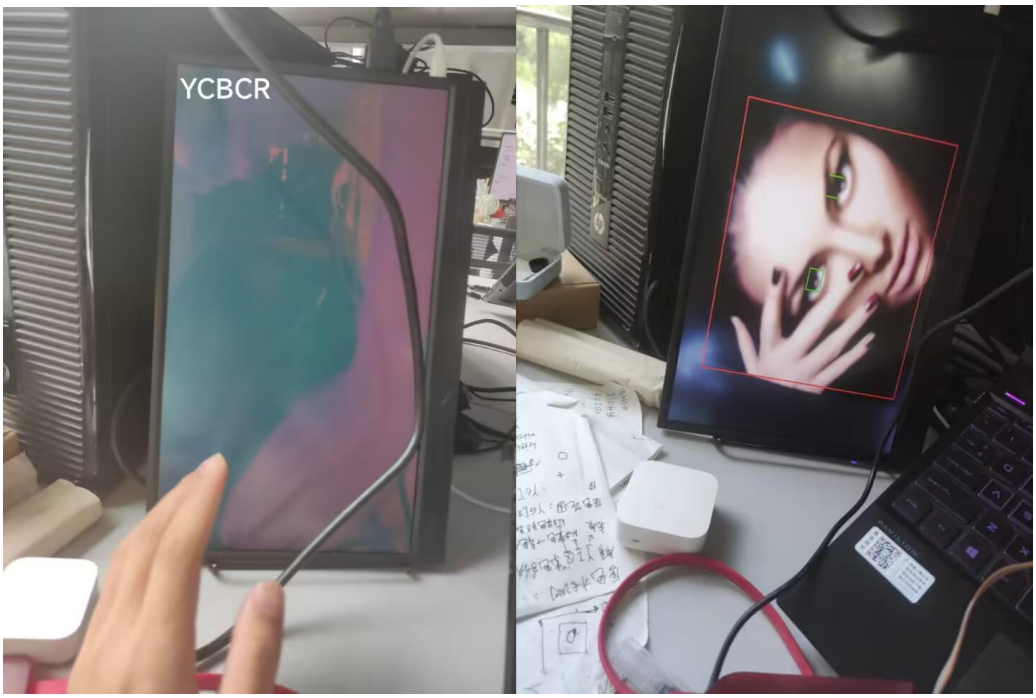


图 1 基于改进的 YCBCR 算法

方法二：基于改进的 YES 颜色空间算法

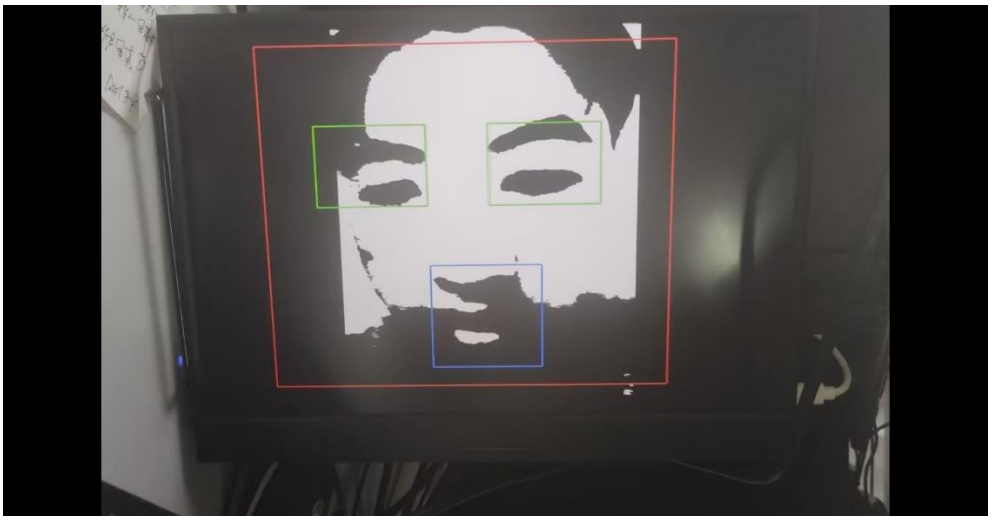


图 2 基于改进的 YES 颜色空间算法

背景：

依赖于复杂模型和大量计算资源，在应用到现 实场景的人脸检测任务时受到限制，因此，实 时轻量级复杂人脸检测方法受到广泛关注。人脸检测是智能视频监控的关键技术之一。 如今人脸检测技术广泛应用于人机交互领域， 在面部情绪自动化分析、视频实时人像跟踪 等视觉任务中发挥重要作用。传统的人脸检 测技术使用依赖于先验知识设计的特征描述算子，应用广泛的有哈尔特征算子（haar-like fea- ture）、梯度方向直方图统计（histogram of ori-ented gradient，HOG）等；而后将所提取的特 征使用分类算法归类，诸如支持向量机（support vector machine，SVM）、迭代提升方法（boost- ing）等。传统的机器学习方式不能自动设计适用于目标数据的特征提取算子，导致其泛化能 力弱，实际使用效果较差；随着人工智能的快 速发展，深度学习技术作为成果之一，在视觉 目标检测方面同样成绩斐然。以人脸检测子领 域为例，采用深度学习技术的算法准确率和速 度不断提高，达到甚至超越人类水平。目前主流的检测方法，主要分为 two-stage（两阶段检 测）和 one-stage（一阶段检测）两类。以 Faster R CNN 为代表的两阶段检测算法由于其先产生候选检测框，再对候选框分类的运行逻辑导致计 算开销大、推理速度慢，难以部署于具有实时 性要求的低算力设备上。一阶段检测算法则不 需要产生候选框，直接将目标在输入图像位置 上的求解过程转化为预设检测框中心位置的回 归问题从而求解，因此检测速度一般比两阶段 算法更快，代价是精度有所降低。这类算法常见的有 SSD 和 YOLO 系列。然而，面对监控视频中存在的小分辨率、遮挡、背景干扰以及 姿态变化等

复杂人脸，现有算法的高准确率仍依赖于复杂模型和大量计算资源，在应用到现实场景的人脸检测任务时受到限制，因此，实时轻量级复杂人脸检测方法受到广泛关注。而本项目仅是在采用特征检测的基础上做出优化，以达到检测效果和速度都得到提升的程度。

1、功能与应用场景

1.1、功能：

能够快速且高效的检测出人脸区域

1.2、应用场景：

（1）、无人零售店、便利店、超市。可以用作客流统计应用。

（2）、公交车、网吧、火车站等人流密集的公共场所。

2、模块内容及算法介绍

2.1、人脸检测系统架构：

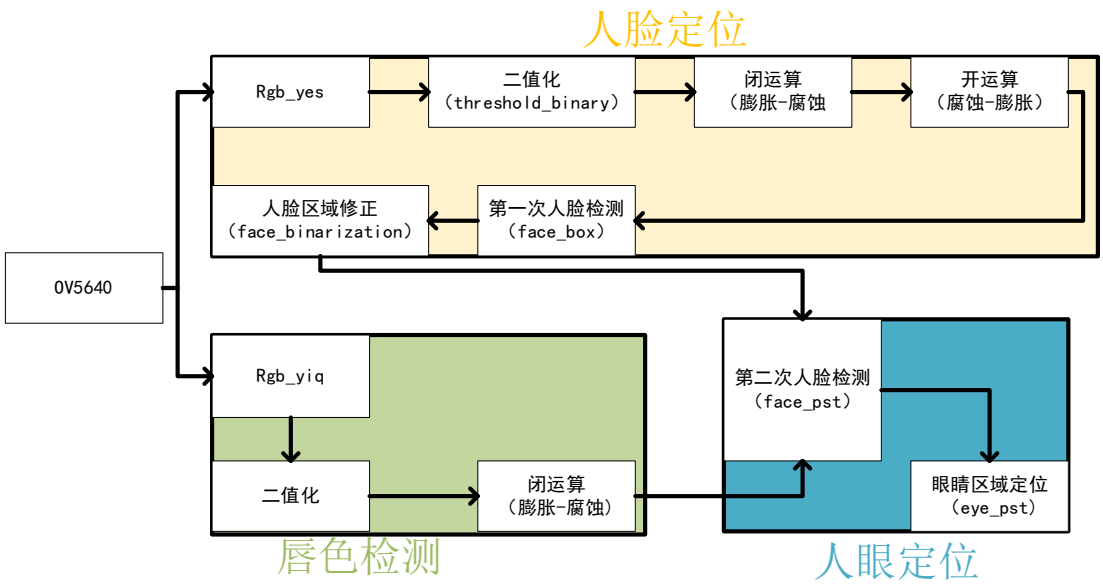


图 2.1 人脸检测架构

2.2、人脸检测各个模块设计：

2.2.1、第一次人脸区域检测算法（粗调）

目的：排除复杂背景下对目标检测的困难性

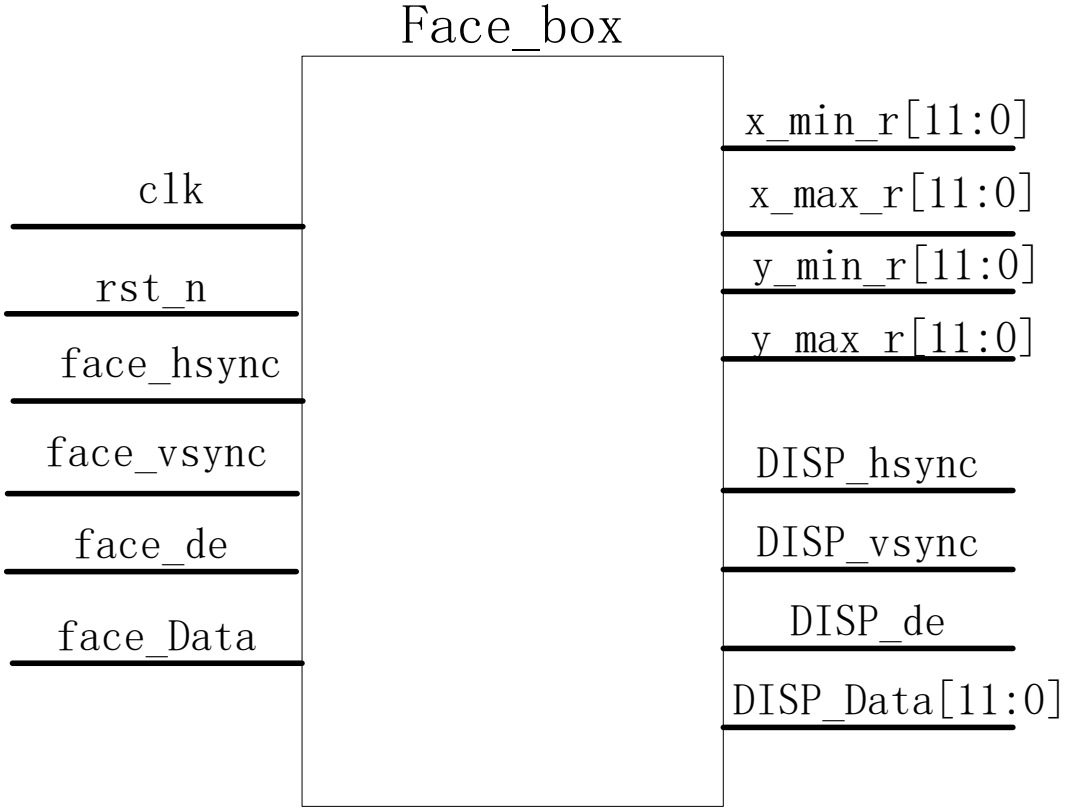


图 2.2.1　第一次人脸检测

经过图像预处理之后通过初步定位人脸区域，得到第一次检测到的坐标位置参数，本次检测采用的算法为基于二值图像的初步坐标定位；及经过人脸肤色的 RGB 到 YES 颜色空间转换得到 YES 的值并利用一系列的图像预处理，得到初步的背景和人脸区分，并将初步人脸区域赋值白色，背景为黑色，并通过水平垂直投影算法得到人脸区域初步坐标值。

部分模块代码如下：

```
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        x_min <= H_DISP;
    end
    else if(pos_vsync) begin
        x_min <= H_DISP;
    end
    else if(face_data==8'hff && x_min > face_x && face_de) begin
        x_min <= face_x;
```

```
        end
    end

    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            x_max <= 'd0;
        end
        else if(pos_vsync) begin
            x_max <= 'd0;
        end
        else if(face_data==8'hff && x_max < face_x && face_de) begin
            x_max <= face_x;
        end
    end

    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            y_min <= V_DISP;
        end
        else if(pos_vsync) begin
            y_min <= V_DISP;
        end
        else if(face_data==8'hff && y_min > face_y && face_de) begin
            y_min <= face_y;
        end
    end

    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            y_max <= 'd0;
        end
        else if(pos_vsync) begin
            y_max <= 'd0;
        end
        else if(face_data==8'hff && y_max < face_y && face_de) begin
            y_max <= face_y;
        end
    end
```

2.2.2、人脸区域修正

目的：优化检测区域内特征不明显性

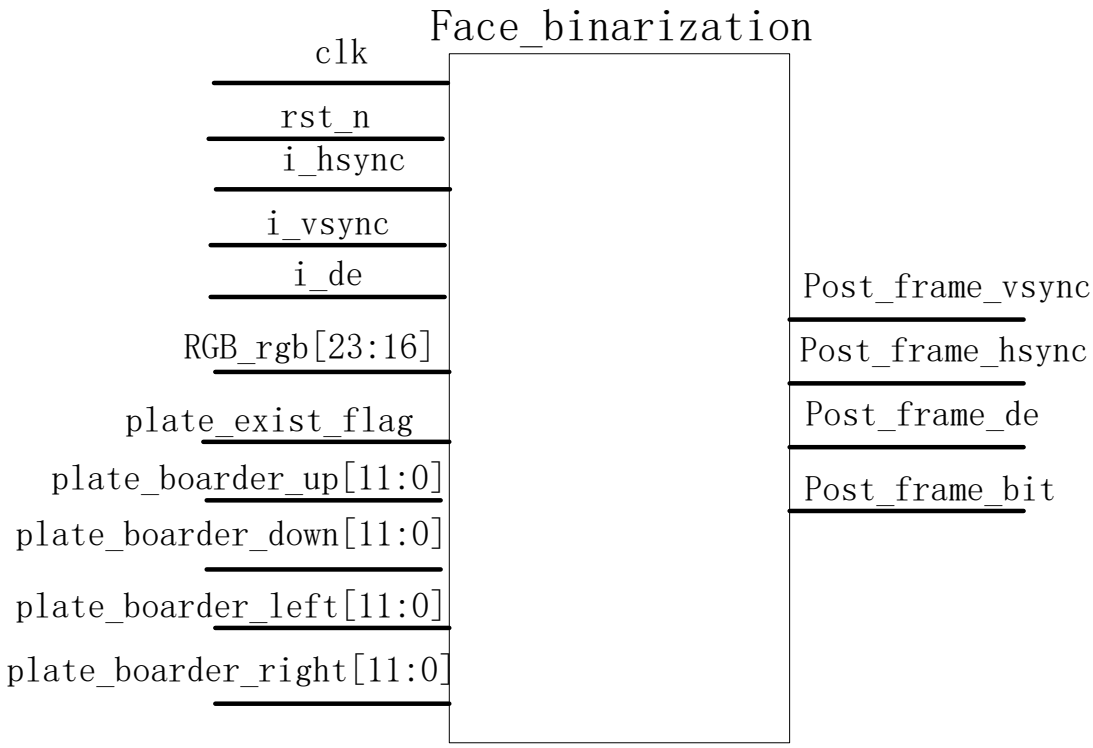


图 2.2.2 人脸区域修正模块

经过人脸区域初步定位，已经粗略的得到了所需要检测的区域，现通过对该区域内部干扰因素进行优化，从第一次检测的效果图可以看出，虽然可以很好的排除复杂背景的影响，但是这也导致了区域内特征不明显，通过对区域内的 RGB 的 R 分量进行预处理，调节到特定的阈值范围，区域外不变，便对该区域内人脸特征进行了优化，使特征更明显，更易于检测。

部分代码如下：

```
always @(posedge clk or negedge rst_n) begin
if(!rst_n)
    post_frame_Bit <= 1'b0;
else if(plate_exist_flag)
    if((y_cnt > plate_boarder_up    ) && (y_cnt < plate_boarder_down ) &&
       (x_cnt > plate_boarder_left) && (x_cnt < plate_boarder_right)    ) begin
        if(per_frame_Red_r > BIN_THRESHOLD)   //阈值
            post_frame_Bit <= 1'b1;
        else
            post_frame_Bit <= 1'b0;
    end
    else
        post_frame_Bit <= 1'b0;
end
```

2.2.3、第二次人脸检测（细调）

目的：精确定位人脸区域

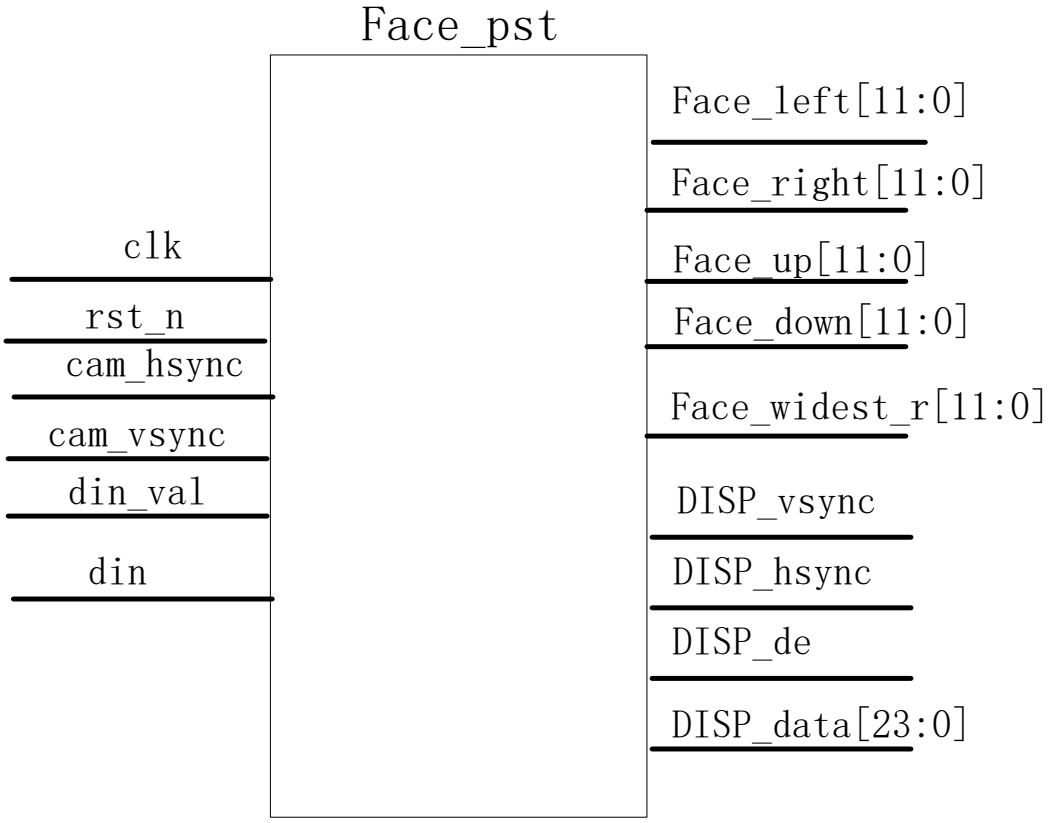


图 2.2.3 第二次人脸检测模块

经过区域内的人脸检测优化，再采用图像预处理的闭运算消除部分噪点，采用第二次检测算法实现最后的定位；首先通过行列统计，统计水平和垂直方向上的白色像素点和黑色像素点，利用工具计算出一幅人脸水平方向上的总像素值，通过判断当前统计的一行像素值大小与特定值比较，若像素值接近该特定值则该行像素满足人脸区域，通过计算该区域内每一行的比较值，排除非人脸区域，并利用水平垂直投影法找出最后的人脸区域范围。

部分代码如下：

```
always @(posedge module_clk or negedge module_rst_n) begin
    if( !module_rst_n ) begin
        r_sta_1cnt          <= 12'd0    ;
        face_up             <= 12'd0    ;
        face_up_tmp         <= 12'd0    ;
        face_up_lock        <= 1'b0     ;
        face_up_lock_tmp <= 1'b0        ;
    end
    else if(cnt_1 == 1024) begin
        if(r_sta == 1 ) begin
            r_sta_1cnt <= r_sta_1cnt + 1    ;
            if(face_up_lock_tmp == 0 && face_up_lock == 0) begin
```

```
                face_up_tmp <= cnt_r          ;
                face_up_lock_tmp <= 1         ;
            end
            if(r_sta_1cnt >= 50) begin
                face_up_lock <= 1                    ;
                face_up <= face_up_tmp - 10          ;
                face_down <= face_up_tmp + 520       ;
            end
        end
        else if(r_sta == 0 ) begin
            r_sta_1cnt <= 0       ;
            if( r_sta_1cnt < 200) begin
                face_up_lock_tmp <= 0       ;
            end
        end
    end
    else if(cnt_r == 768 && cam_href == 0) begin
        face_up_lock <= 0             ;
        face_up_lock_tmp <= 0       ;
        r_sta_1cnt <= 12'd0           ;
    end
end
//寻找脸左右边界
always @(posedge module_clk or negedge module_rst_n) begin
    if( !module_rst_n ) begin
        face_left_tmp           <= 12'd0          ;
        face_right_tmp          <= 12'd0          ;
        face_left_tmp_min    <= 12'd0       ;
        face_right_tmp_max <= 12'd0          ;
        face_left_tmp_lock <= 0                ;
    end
    else if(cnt_1 != 1024 && cnt_r != 768 && cam_href != 0) begin
        if(din == 1 ) begin
            if(face_left_tmp_lock == 0) begin
                face_left_tmp <= cnt_1          ;
                face_left_tmp_lock <= 1         ;
            end
            if(face_left_tmp_lock == 1 ) begin
                face_right_tmp <= cnt_1         ;
            end
        end
        else if(din == 0) begin
            if( rsta_tmp_r < 200 ) begin
                face_left_tmp_lock <= 0         ;
```

```
                end
                else if(rsta_tmp_r >= 200 && face_left_tmp_lock == 1) begin
                    face_left_tmp_min   <= (face_left_tmp   < face_left_tmp_min)   ? face_left_tmp     :
face_left_tmp_min       ;
                    face_right_tmp_max <= (face_right_tmp > face_right_tmp_max) ? face_right_tmp   :
face_right_tmp_max      ;
                end
            end
        end
        else if( cnt_l == 1024 && cnt_r != 768) begin
            face_left_tmp_lock <= 0         ;
        end
        else if(cnt_r == 768 && cam_href_neg == 1) begin
            face_left   <= face_left_tmp_min            ;
            face_right <= face_right_tmp_max        ;
            face_left_tmp_min   <= 12'd0            ;
            face_right_tmp_max <= 12'd0             ;
        end
    end
```

2.2.4、唇色检测

目的：通过唇色定位嘴唇位置

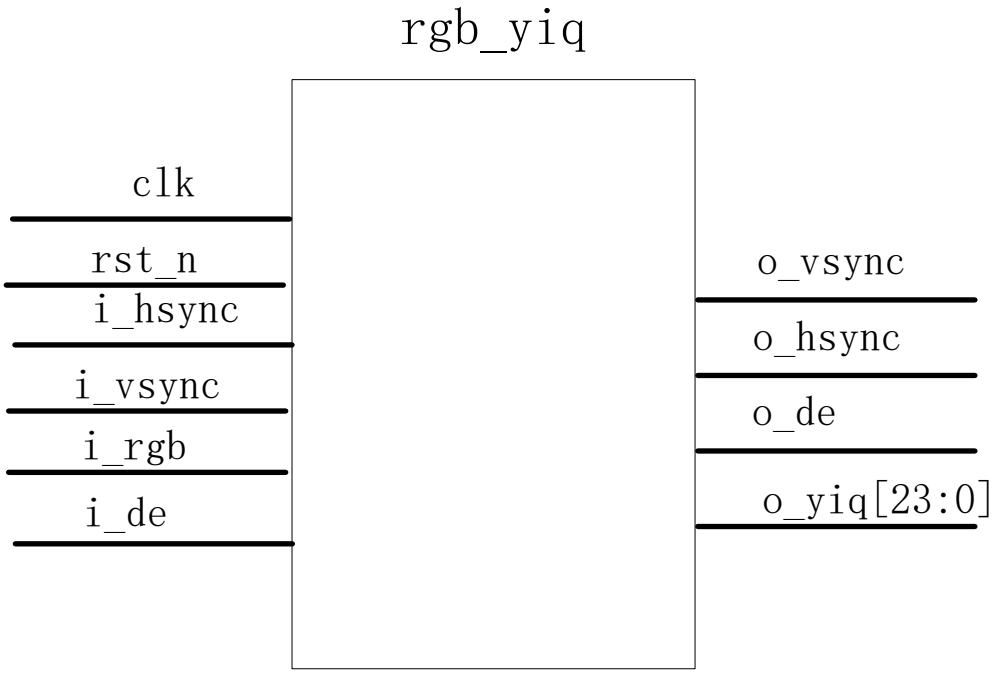


图 2.2.4 第二次人脸检测模块

通过之前的算法处理，已经得到了不错的检测效果，最后通过融合唇色特征，得到更精确的人脸区域；唇色检测通过 RGB 转 YIQ 的颜色空间特征，对 YIQ 唇色范围进行计算得出嘴唇的区域坐标，最后融合之前的人脸区域坐标，得出最终的人脸检测。

部分代码如下：

```
parameter Y_TH          = 220,
parameter Y_TL          = 80 ,
parameter i_TH          = 78,
parameter i_TL          = 12,
parameter q_TH          = 25,
parameter q_TL          = 7

assign en0          =i_yiq[23:16] >=Y_TL    &&    i_yiq[23:16] <= Y_TH;
assign en1          =i_yiq[15: 8] >=i_TL    &&    i_yiq[15: 8] <= i_TH;
assign en2          =i_yiq[ 7: 0] >=q_TL    &&    i_yiq[ 7: 0] <= q_TH;
```

2.2.5、眼睛区域检测

目的：定位人眼区域用于疲劳检测

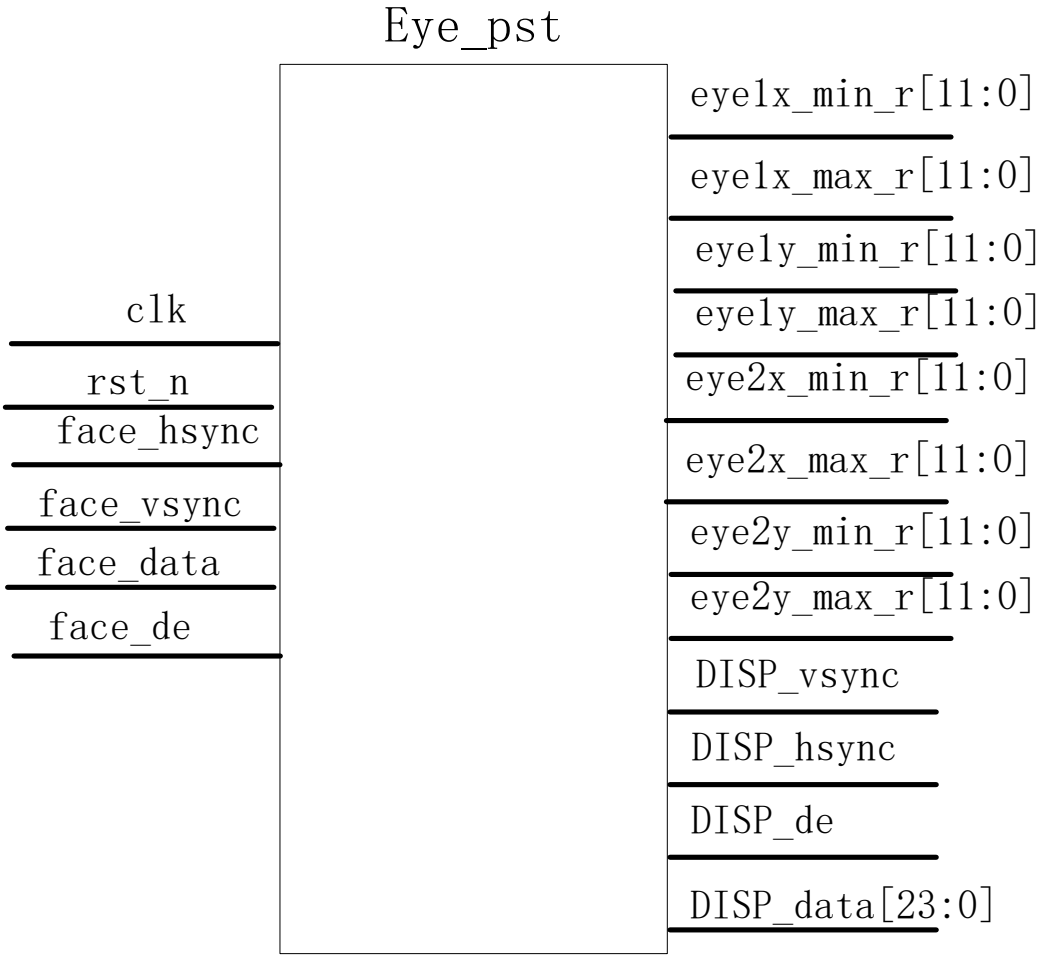


图 2.2.5 第二次人脸检测模块

人脸检测已经确定了人脸位置，根据人脸的对称性，可以初步得到眼睛区域，之前采用的人脸区域优化模块，便可以区分开眼睛和人脸其他地方的区别，眼睛区域为黑色像素点，其他区域为白色像素点，再通过水平垂直投影对眼睛区域进行定位，便能得到最后的眼睛区域位置。

部分代码如下：

```
//帧运行：左眼框选
        always @(posedge clk or negedge rst_n) begin
            if(!rst_n) begin
                eye1x_min <= H_DISP;
            end
            else if(pos_vsync) begin
                eye1x_min <= H_DISP;
            end
        else if( face_x >= face_left && face_x <= center &&    face_y >= face_up && face_y <= face_widest_r)
begin
             if(face_data==8'h00 && eye1x_min > face_x && face_de) begin
                eye1x_min <= face_x;
```

```
            end
            end
        end
        always @(posedge clk or negedge rst_n) begin
            if(!rst_n) begin
                eye1x_max <= 'd0;
            end
            else if(pos_vsync) begin
                eye1x_max <= 'd0;
            end
        else if( face_x >= face_left && face_x <= center &&    face_y >= face_up && face_y <= face_widest_r)
begin
             if(face_data==8'h00 && eye1x_max < face_x && face_de) begin
                eye1x_max <= face_x;
            end
            end
        end
        always @(posedge clk or negedge rst_n) begin
            if(!rst_n) begin
                eye1y_min <= V_DISP;
            end
            else if(pos_vsync) begin
                eye1y_min <= V_DISP;
            end
        else if( face_x >= face_left && face_x <= center &&    face_y >= face_up && face_y <= face_widest_r)
begin
             if(face_data==8'h00 && eye1y_min > face_y && face_de) begin
                eye1y_min <= face_y;
            end
            end
        end
        always @(posedge clk or negedge rst_n) begin
            if(!rst_n) begin
                eye1y_max <= 'd0;
            end
            else if(pos_vsync) begin
                eye1y_max <= 'd0;
            end
        else if( face_x >= face_left && face_x <= center &&    face_y >= face_up && face_y <= face_widest_r)
begin
             if(face_data==8'h00 && eye1y_max < face_y && face_de) begin
                eye1y_max <= face_y;
            end
        end
```

```
    end
    //帧运行：右眼框选
    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            eye2x_min <= H_DISP;
        end
        else if(pos_vsync) begin
            eye2x_min <= H_DISP;
        end
    else if( face_x > center && face_x <= face_right &&   face_y >= face_up && face_y <= face_widest_r)
begin
         if(face_data==8'h00 && eye2x_min > face_x && face_de) begin
            eye2x_min <= face_x;
        end
        end
    end
    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            eye2x_max <= 'd0;
        end
        else if(pos_vsync) begin
            eye2x_max <= 'd0;
        end
    else if( face_x > center && face_x <= face_right &&   face_y >= face_up && face_y <= face_widest_r)
begin
         if(face_data==8'h00 && eye2x_max < face_x && face_de) begin
            eye2x_max <= face_x;
        end
    end
    end
    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            eye2y_min <= V_DISP;
        end
        else if(pos_vsync) begin
            eye2y_min <= V_DISP;
        end
    else if( face_x > center && face_x <= face_right &&   face_y >= face_up && face_y <= face_widest_r)
begin
         if(face_data==8'h00 && eye2y_min > face_y && face_de) begin
            eye2y_min <= face_y;
        end
    end
    end
```

```
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        eye2y_max <= 'd0;
    end
    else if(pos_vsync) begin
        eye2y_max <= 'd0;
    end
else if( face_x > center && face_x <= face_right &&   face_y >= face_up && face_y <= face_widest_r) begin
     if(face_data==8'h00 && eye2y_max < face_y && face_de) begin
        eye2y_max <= face_y;
    end
end
end
```

二、疲劳检测

该部分设计的创新点：

相比于常见的 EAR(Eye Aspect Ratio)眼睛纵横比判断眨眼频率，本次设计采用帧差法运动目标检测，来统计眨眼频率，优点在于其检测速度更快，更准确。

下图 1 为原始图片；图 2 为检测到闭眼状态；图 3 为判定为疲劳状态。

图 1 原始图片

图 2 检测到闭眼状态

图 3 判定为疲劳状态

背景：

眼睛是也灵之窗，驾驶员的也理、生理状态能够从眼部状态得到不同程度的反映。过去几十年里，基于眼部特征的疲劳驾驶检测算法层出不穷，各种可监测特征如眨眼频率、PERCLOS、眼睛闭合比例）、注视方向、眼球运动、瞳孔反应等都能用于评价疲劳状态。

同时，基于眼部特征的疲劳驾驶检测算法是非接触式的，有学者认为这些算法是最适合在实际应用场景下使用的。基于眼部特征的疲劳检测算法主要有两种——基于表象的被动算法和基于红外光源的主动算法。前者是在人脸检测的基础上根据眼部区域的特征定位人眼，通常包含两个步骤；首先，使用人脸检测算法（神经网络、肤色模型、ＳＶＭ、支持向量机、主元分析等）定位人脸；随后使用比例模型或者分类器等算法结合一些相应的图像处理操作定位人眼。本项目在特征检测眨眼次数判断疲劳检测的基础上采用帧差法实现。该实现效果速度较快，准确率较高。

1、功能与应用场景

1.1、功能：

实现疲劳驾驶检测并通过 HDMI 显示检测结果

1.2、应用场景：

对出租车、客车、公交车、货车等各类营运车辆，实时监控车内情况。

2、模块内容及算法介绍

2.1、疲劳检测架构

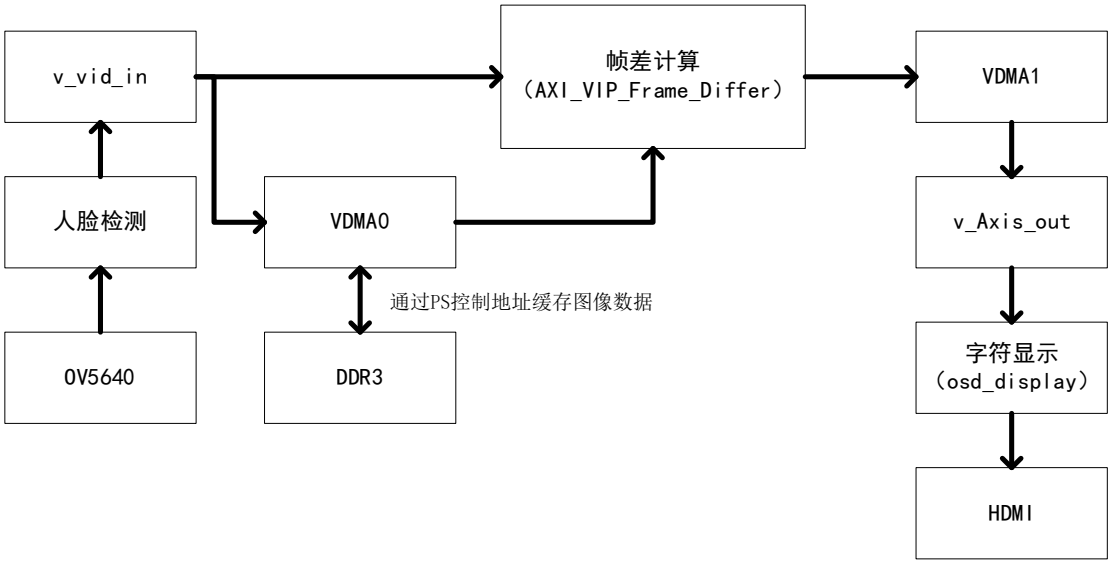


图 2.1 疲劳检测架构

2.2、疲劳检测模块设计

目的：通过帧差法在之前的人眼区域进行疲劳检测

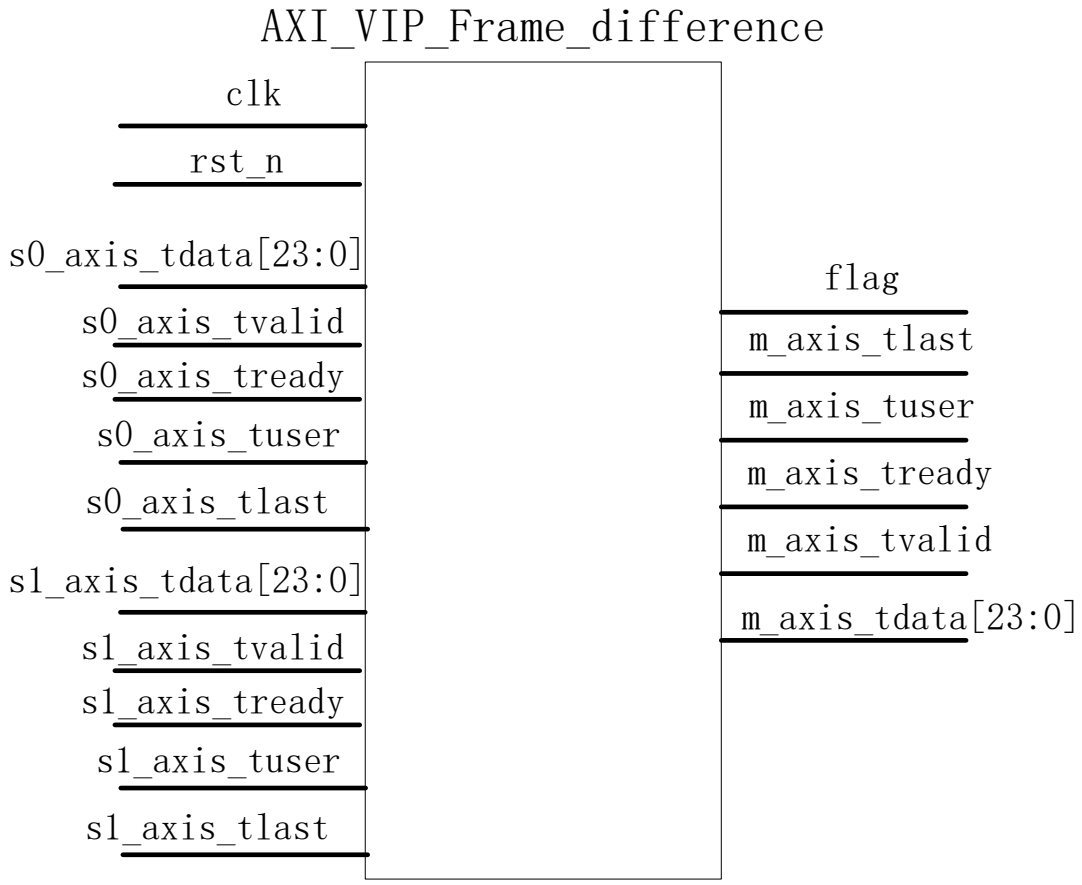


图 2.2 疲劳检测模块

原理：首先通过 OV5640 采集图像数据，将采集到的图像分两路输出，一路图像数据送到 VDMA 进行缓存等待另一路图像数据到来，同时读出两路图像数据，经过简单的预处理，将两路图像数据送到帧差计算模块计算帧差结果，利用之前的人脸检测模块得到的眼睛区域位置，结合帧差结果后进行水平垂直投影画框输出。

帧差法模块部分代码如下：

```
//帧差标志位，为 1 表示两帧数据差别超过阈值

always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        frame_difference_flag      <= 1'b0;
    end
    else begin
        if(s0_img_y > s1_img_y) begin
            if(s0_img_y - s1_img_y > Diff_Threshold) begin
                frame_difference_flag <= 1'b1;
                end
            else begin
```

```
                frame_difference_flag <= 1'b0;
                end
        end
        else begin
            if(s1_img_y - s0_img_y > Diff_Threshold) begin
                frame_difference_flag <= 1'b1;
                end
            else begin
                frame_difference_flag <= 1'b0;
                end
        end
    end
end
```

2.2.1、图像缓存模块设计

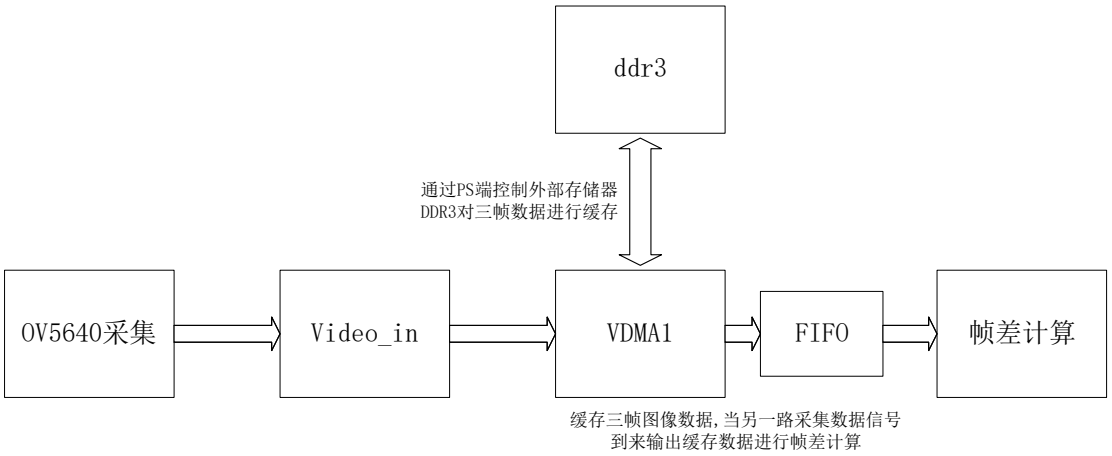


图 2.2.1 图像缓存流程

由摄像头采集的图像数据通过 video_in 将摄像头时序转换为 AXI 协议输入，将其中一路图像数据传入 VDMA1 通过与 PS 端交互进行缓存三帧图像数据，等另一路图像数据到来同时读出到帧差计算模块，计算帧差结果。

部分代码如下：

（1）PS 端通过地址控制 VDMA 缓存三帧数据

```
//frame buffer的起始地址，用于缓存一张图片作帧差
unsigned int const frame_buffer_addr = (XPAR_PS7_DDR_0_S_AXI_BASEADDR
                                        + 0x1000000);
//用作VDMA显存
unsigned int const frame_buffer_addr1 = (XPAR_PS7_DDR_0_S_AXI_BASEADDR
                                        + 0x3000000);
```

```
//初始化 VDMA
    run_vdma_frame_buffer(&vdma, VDMA_ID, vd_mode.width, vd_mode.height,
                                        frame_buffer_addr,0,0,BOTH);
//初始化 VDMA1
    run_vdma_frame_buffer(&vdma1, VDMA_ID1, vd_mode.width, vd_mode.height,
                                        frame_buffer_addr1,0,0,BOTH);
```

（2）采用 FIFO 来对齐缓存的输入时序

//将 VDMA 的图像缓存到 FIFO 中

```
assign s1_axis_tready = ~fifo_full;
always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        fifo_wr_en <= 1'b0;
       fifo_wr     <= 1'b0;
        fifo_data  <= 24'd0;
    end
    else begin
        if(s1_axis_tvalid & s1_axis_tready & s1_axis_tuser)
            fifo_wr_en <= 1'b1;
        if(s1_axis_tvalid & s1_axis_tready) begin
            fifo_wr    <= 1'b1;
            fifo_data  <= s1_axis_tdata;
        end
        else  begin
            fifo_wr    <= 1'b0;
            fifo_data  <= fifo_data;
        end
    end
end
```

//缓存来自 VDMA 的图像数据

```
video_fifo u_video_fifo (
  .clk                      (clk),
  .srst                     (~rst_n),

  .din                      (fifo_data),
  .wr_en                    (fifo_wr & fifo_wr_en),
  .full                     (),

  .rd_en                    (fifo_rd & fifo_rd_en),
  .dout                     (fifo_q),
```

```
    .empty                          (fifo_empty),
    .almost_full                    (fifo_full),
    .almost_empty                   ()
);
```

//摄像头输入像素的同时，从 FIFO 中读出 VDMA 像素，以进行帧差运算

```
always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        fifo_rd_en <= 1'b0;
        fifo_rd    <= 1'b0;
    end
    else begin
        if(s0_axis_tvalid & s0_axis_tready & s0_axis_tuser & fifo_full)
            fifo_rd_en <= 1'b1;
        if(s0_axis_tvalid & s0_axis_tready)
            fifo_rd    <= 1'b1;
        else
            fifo_rd    <= 1'b0;
    end
end
```

2.2.3、字符显示模块设计

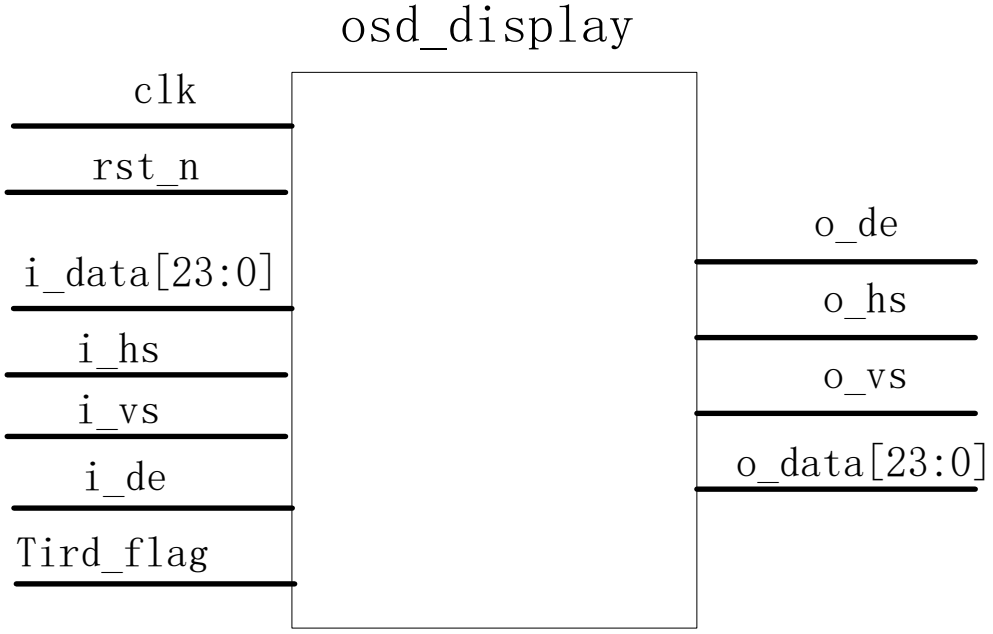


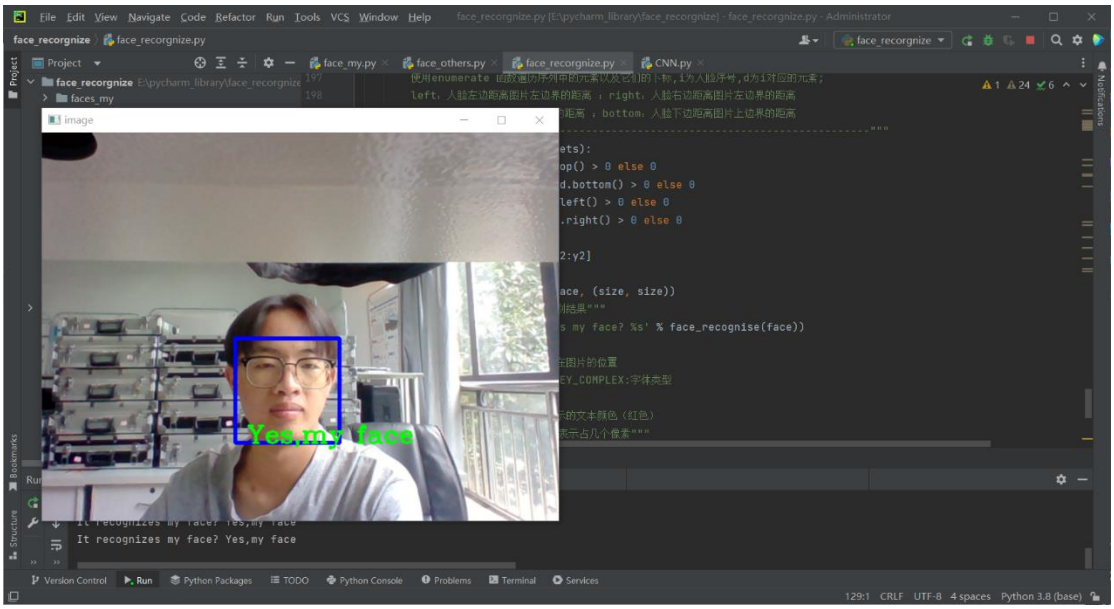
图 2.2.3 字符显示模块

该模块用于判断是否出现疲劳状态，根据疲劳检测的标志信号 Tird_flag 来判断状态，当 flag 信号拉高，进行计数加一，如果计数结果达到一定数值则判定该时刻处于疲劳状态，显示屏显示疲劳，持续一定时间后清零，等待下次疲劳状态出现继续显示。

三、基于 CNN 的人脸识别

待完善……

识别模块的 python 验证



参考文献：（1）一种基于改进 YUV 的人脸检测方法
（2）基于肤色和唇色信息的人脸检测方法的研究
（3）基于 FPGA 平台的实时人眼检测系统设计
（4）基于 Adaboost 人脸检测算法的研究
（5）基于 FPGA 的实时人脸检测算法研究
（6）基于 FPGA 卷积神经网络的人脸识别系统设计
（7）基于深度可分离卷积的人脸识别系统设计
另外一些参考的是之前我们做水果识别的一些图像预处理。

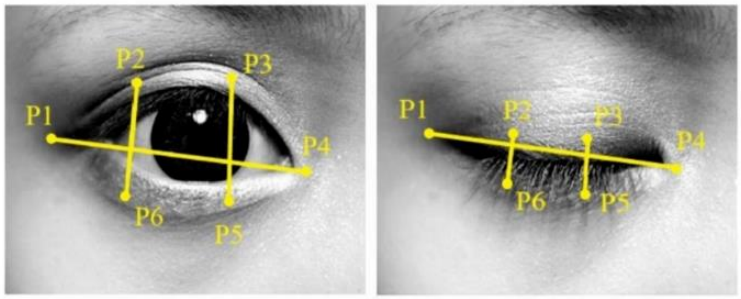
目前实现的效果以及比较

附录：

下面是一些算法的原理资料的整理

一、第一篇

当训练好人体面部关键点定位模型后，就是模型使用过程了，使用过程中再对视频中检测出现的关键点进行分析。按照上面的思路，既然是对眨眼次数的计算，那么这里需要做的就是**计算眼睛轮廓点中关键位置的定位点之间的距离比**，以下图为例，当计算出相关位置的点之间的距离比（轮廓点构成的高度与宽度比）小于一定范围内容的时候，我们认为属于眨眼，当这个距离保持在一定时间后（比如可以设置成3秒），我们认为此时已经不是眨眼了，而是闭眼。



$$EAR = \frac{\|P_2 - P_6\| + \|P_3 - P_5\|}{2\|P_1 - P_4\|}$$

上图是根据眼睛的纵横比表征眼睛的睁开程度，即眼睛宽度与眼睛长度的比例。其原理是，在睁眼状态下，人眼的宽度与长度的比值是一个相对固定的数值，当眼睛闭合时，人眼的宽度将极剧变小，此时眼睛的纵横比就会发生变化。此参数通常被用于眨眼频率计算。

对于是否处于疲劳状态，需要有一个时间段内闭眼帧数的测试，根据卡内基梅隆研究所提出的度量疲劳/瞌睡的物理量 PERCLOS （Percentage of Eyelid Closure over the Pupil， over Time， 简称PERCLOS） 其定义为单位时间内 （一般取1 分钟或者 30 秒） 眼睛闭合一定比例 （5%或15%） 所占的时间， 满足下式时就认为进入了疲劳状态：

PERCLOS 的计算公式如下：

$$PERCLOS = \frac{\text{眼睛闭合帧数}}{\text{检测时间段总帧数}} X\ 100\%$$

根据国内外学者的大量实验分析可得，当 f ≥ 0.12时，则可认为驾驶员处于疲劳状态；当 f < 0.12 时，则认为驾驶员处于清醒状态。

我在前面已经讲了眨眼计算的算法和计数思路，那么按照这个方法，打哈欠这个指标就很容易实现了，实现方法和眨眼计数一样，也需要对人的嘴角轮廓进行定位，然后根据嘴角轮廓的关键点构成的高和宽之间的距离比来判断是否属于打哈欠的行为。

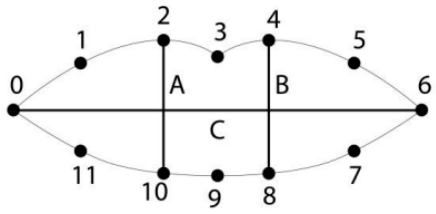


Fig 3: Mouth region with reference coordinate points

$$Am = d(mouth[2], mouth[10]) \quad (4.1)$$

$$Bm = d(mouth[4], mouth[8]) \quad (4.2)$$

$$Cm = d(mouth[0], mouth[6]) \quad (4.3)$$

Am and Bm measure the vertical distance of the eye and Cm calculates the horizontal dimensions of the eye.

$$MAR = (Am + Bm) / (2.0 * Cm) \quad (4.4)$$

嘴部纵横比表征嘴部的张开程度，即嘴部高度与嘴部宽度的比例。当打哈欠时，嘴部将极剧变大，此时嘴部的纵横比就会发生变化。

在研究疲劳的反应中发现眼睛的闭合程度与疲劳程度有很大的相关性，并且认为i它是最可靠，最准确的信息。PERCLOS是percentage of eyelid closure over the pupil over time的缩写，即是在单位时间例闭合时间所占的百比率。PERCLOSE方法被认为是机动车辆最有效的、车载的、实时的、非接触式的疲劳测评方法。通常PERCLOSE方法有3种标准

p70:表示当瞳孔面积有超过70%被眼睑遮住时，就认为眼睛处于闭合状态，统计在单位时间内眼睛闭合时间所占的时间比例

p80:表示当瞳孔面积有超过80%被眼睑遮住时，就认为眼睛处于闭合状态，统计在单位时间内眼睛闭合时间所占的时间比例

EM:表示当瞳孔面积有超过50%被眼睑遮住时，就认为眼睛处于闭合状态，统计在单位时间内眼睛闭合时间所占的时间比例

实验表明，p80比较准确

2.3 PERCLOS 检测算法

PERCLOS 值为人眼闭合时间占总时间的百分比，通过图 6 可说明其测量原理[8]。

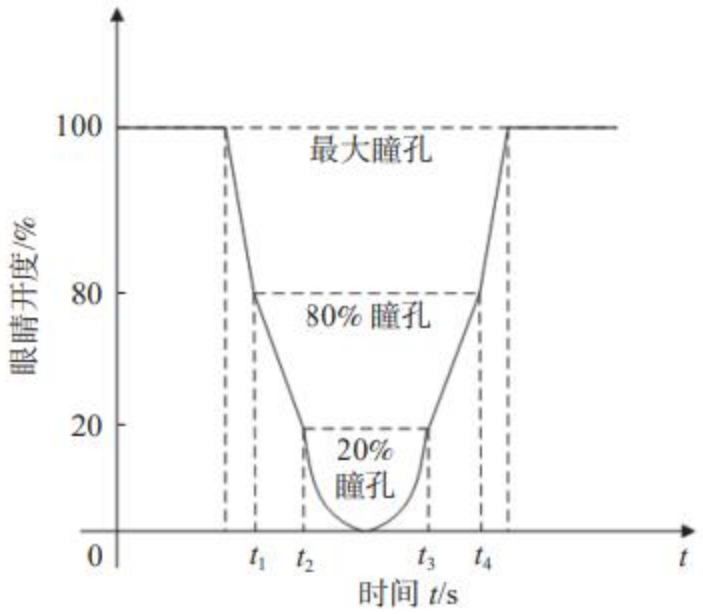


图 6 PERCLOS 值的测量原理

PERCLOS 值的计算方法为：

$$f = \frac{t_3 - t_2}{t_4 - t_1} \times 100\% \qquad (1)$$

将眼睛检测的 PERCLOS 值与预先设定的阈值进行比较以达到判断疲劳驾驶的目的，若驾驶员的眼睛闭合时间在 3 s 以上且 PERCLOS 值在 40% 以上，即被系统认定为疲劳驾驶状态，随即发出警报。

2.5 眼部张合度检测 利用脸部参数测量眼睛的睁闭程度[11]，通过 计算眼睛的大小可以确定眼睛的睁闭状态。使用 Dlib 库可以提取眼睛的 6 个特征点。6 个特征点的 坐标分别是 p1、p2、p3、p4、p5、p6，

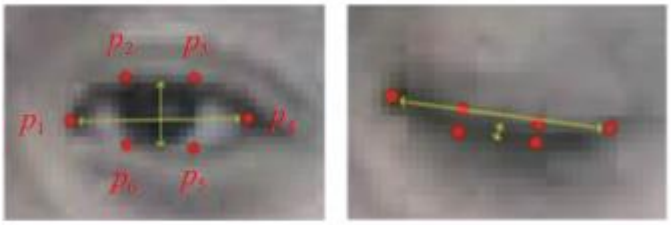


图 8　眼部特征点标记示意图

p1 p6 p5 p4 p3 p2 图 8　眼部特征点标记示意图 EAR 的计算公式为：

$$\mathrm{EAR} = \frac{\| p_2 - p_6 \| + \| p_3 - p_5 \|}{2 \| p_1 - p_4 \|}$$

分母和分子分别代表眼睛特征点的水平和垂 直距离。基于垂直点对的数量相比水平点对超出 了一倍，将分母值乘 2，以确保两组特征点的权重 相同。当人眼处于非闭合状态时，EAR 值只会在 一个小范围内上下浮动，当人眼处于闭合状态 时，EAR 值会迅速降低。

4　测试及结果

将实验设计的系统完全搭建好，模拟驾驶状态，通过真实的测试，采集人在正常和疲劳状态下的面部数据，对系统的实时性和准确性进行验证。疲劳检测效果如图 10 所示。

图 10　疲劳检测效果图

通过实验测试结果可知，在正常光环境下，当人员的驾驶状态保持向前平视时，系统检测到的 EAR 值将保持在 0.33~0.38 之间。疲劳驾驶条件下，EAR 参数均低于 0.15，而被测者眼睛完全闭上时，EAR 参数将保持在 0.08 左右。这说明系统将 EAR 的检测阈值设置为 0.25 是有效合理的。当驾驶员眼睛闭上超过 2 s 时，系统能够检测其为疲劳状态并发出警告，同时对于驾驶员正常眨眼的情况不会存在误判。此外，在光线增强的情况下，系统的鲁棒性仍然较高，可以准确区分驾驶员的正常状态和疲劳状态，但在光线微弱的条件下，比如夜晚，由于摄像头提取的眼部特征不清晰，*EAR* 值偶尔会较高。

二、第二篇

三、基于视觉特征的检测方法

基于视觉特征的疲劳检测方法通过图像传感器实时获取驾驶员图像，利用机器视觉和图像处理等技术分析驾驶员眼睛、嘴巴或者头部运动情况，判断驾驶员精神状态[13]。该方法因其非接触性，不会对驾驶员行车造成干扰，具有较好的实时性和鲁棒性，检测准确度高，伴随机器视觉、图像处理和模式识别技术的快速发展，使得该检测方法具有广阔的市场前景。

目前，国外疲劳检测系统使用较多，而国内大多数通过记录驾驶员行车时长进行疲劳判定，比如有高德地图、腾讯地图、百度地图等手机 APP 和车载电子狗等装置，以及一些基于路况试验进行粗略评测的疲劳检测装置，但这些装置的准确率都不能保障[14]。无论是国内还是国外，中低端车型上均没有配备疲劳检测装置，一是由于其造价成本高，二是因为检测准确性不能得到保证。

综上，本文以驾驶员面部为检测重点，选取眼睛和嘴巴作为检测区域，一方面计算眼睛纵横比（Eye Aspect Ratio, EAR）与眼睛二值图像黑色像素和，提取眼部疲劳参数；另一方面计算嘴巴纵横比（Mouth Aspect Ratio, MAR），提取嘴部疲劳参数。通过结合 PERCLOS（Percentage of Eyelid Closure over the Pupil over Time）算法 P80 标准、眨眼频率和打哈欠频率，经多特征融合建立疲劳检测模型，进行驾驶员精神状态判定。

3.采用基于 Haar-Like 特征的 AdaBoost 算法实现人脸检测，人脸检测完成后， 采用基于级联姿势回归算法实现人脸 68 个特征点定位。

4.根据人脸特征点位置精确定位眼睛和嘴巴，提出利用眼睛纵横比 EAR 值结 合相邻帧眼睛二值图像黑色像素和比例进行眼睛状态识别，嘴巴纵横比 MAR 值 进行嘴巴状态识别，提取疲劳特征信息。

5.提出利用多特征参数进行信息融合来建立疲劳检测模型，以 PERCLOS 算法 P80 标准结合眨眼频率和打哈欠频率作为评价标准，通过统计分析实验设定三个 评价指标。当驾驶员佩戴口罩只检测到眼睛时，利用眼睛状态进行疲劳判定；

6 驾驶员佩戴墨镜只检测到嘴巴时，利用嘴巴状态进行疲劳判定；当系统同时检测 到眼睛和嘴巴时，融合两个特征判断驾驶员精神状态。

2.1.2 眨眼频率

研究发现：当人处于疲劳状态时，眼睛部位的形状会发生明显变化。眨眼是指眼睛从睁开状态到闭合再到睁开状态的过程，正常人在正常情况下完成一次眨眼动作所需时间为 0.2s~0.3s[46]。正常人每分钟眨眼频率在 10~25 次之间，如果大于这个范围，驾驶员可能正处于疲劳状态，通过不停眨眼的方式克服疲劳；如果小于

这个范围，驾驶员可能正处于闭眼、发呆或者走神状态。因此，眨眼频率过低或过高表示驾驶员处于疲劳或者异常情况，通过单位时间内眨眼频率可判定驾驶员精神状态。通过实验统计设定阈值，当驾驶员一分钟内眨眼次数≤5 次或≥20 次时，判定驾驶员处于疲劳状态。

2.1.3 打哈欠频率

嘴部状态和眼部状态一样都能表征疲劳，当人处于疲劳状态时往往伴随着频繁的打哈欠行为。驾驶员正常行车途中，嘴巴状态基本处于闭合；当驾驶员与乘客沟通交流时，嘴巴状态为频繁正常张开，幅度不大；而当驾驶员出现打哈欠动作时，嘴巴处于张大状态，并且张开幅度较大，持续时间较长[13]。因此，当嘴巴出现张开幅度较大且持续时间较长时，可判定为打哈欠行为。类似于眨眼频率的判定，设定单位检测时间，若在该时间段内打哈欠的频率大于设定阈值，则判断驾驶员处于疲劳状态，通过实验统计设定阈值，当驾驶员一分钟内打哈欠次数≥2 次，判定驾驶员处于疲劳状态。

5.1 人眼粗定位

人眼定位根据其应用场景不同分为人眼区域粗定位及人眼精确定位两类。人眼区域粗定位用于检测、跟踪人眼，粗略定位眼部区域；而人眼精确定位则应用于提取眼部信息及判断眼睛状态。常用的人眼区域粗定位方法可分为利用人眼的几何特征、灰度特征等通过算法实现定位；以及通过机器学习，利用分类器实现人眼区域粗定位。前者对于光照强度较敏感，且需要根据输入图像调节参数，若受试者戴眼镜，其定位结果会受一定程度干扰。分类器定位人眼具有通用性，且其定位精度及速度得到保障，故本文采用分类器完成人眼粗定位。

将视频图像经基于 YCbCr 空间及 Haar-like 特征人脸检测定位之后，采用机器学习得到的基于 Haar-like 特征的人眼分类器进行人眼区域粗略定位，根据所训练的人眼样本数据感知并读取输入图像中的人眼特征，进而对图像进行眼部区域框选。

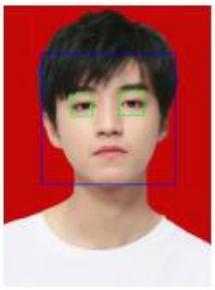


图 5-1 人眼粗定位实验结果图

Fig. 5-1 Coarse eye positioning results

不断调整 Haar 眼部检测函数中搜索窗口的比例系数，尽可能的减小将每次搜索窗口扩大的比例，提高检测精度；适当增加检测目标的相邻矩形的最小个数，增加重叠检测次数，以此获得精确地眼部区域定位。通过大量实验得到：将搜索窗口比例设置为 1.15~1.35 之间，检测目标相邻矩形最小数设置为 5~8，这样框选的眼部区域所保留的眼部信息是相对完整的，如图 5-1 所示。若这两个参数值的设定未分别处于以上两个区间以内，眼部区域将会出现多个矩形框框选，会对视

5.2.1 特征点检测定位人眼

由基于回归树的人脸对齐算法知道：人脸形状是由 68 个特征点组成[65,66]，其中单只眼睛是由 6 个特征点组成。如图 5-2 所示为人眼特征点分布，P1~P6 是人脸特征点中对应眼睛的 6 个特征点，从眼角开始沿着眼睛顺时针分布，每只眼睛由 6 个(x, y)坐标表示。

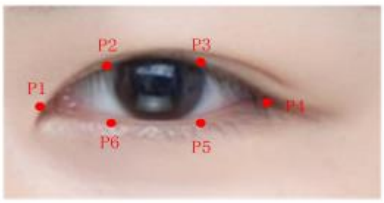


图 5-2　人眼特征点分布

Fig. 5-2　Distribution of human eye feature points

睁眼时

闭眼时

图 5-3　人眼二次定位

Fig. 5-3　Eye secondary localization

5.3.1 眨眼检测

眨眼是两只眼睛同时进行的，所以两只眼睛的 EAR 值是平均的，故取两只眼睛纵横比的平均值作为每一帧计算的结果。如果眼睛纵横比低于某值，然后再超过该值，那么将其记录为“眨眼”。通过上一节记录的实验数据，将 $EAR = 0.2$ 作为闭眼阈值，只要 $EAR < 0.2$ 时就默认为眼睛处于闭合状态。

图 5-7 所示为视频序列上人眼 EAR 信号示例图，图中眼睛纵横比是恒定的，然后 EAR 值迅速下降到接近 0，随后 EAR 值上升，这就表明一个单一的眨眼动作完成。因此，可以根据眼睛纵横比 EAR 值的大小判断眼睛的睁闭状态。

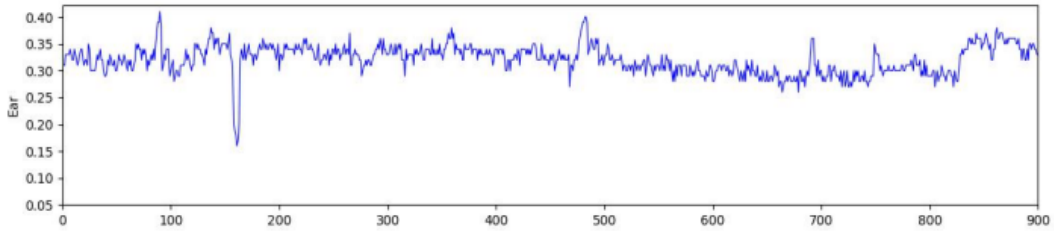


图 5-7　视频序列上的 EAR 值示例图

Fig. 5-7　Sample diagram of an EAR value on a video sequence

EYE_AR_THRESH 为眼睛纵横比的阈值，将其设置为 0.2，也可以根据自身的应用程序做相应调整；blink_counter 为脚本运行时发生的眨眼总次数，将其初始化为 0。当眼睛纵横比值小于所设阈值，就记录为眨眼，增加眨眼总次数。

根据单帧图像的眼睛纵横比值进行实时眨眼检测，结果如图 5-8 所示，只要

三、第三篇

本文选用 ZedBoard 开发板，其核心芯片为 Zynq 系列的 xc7z020-clg484-1，Zynq 芯片的结构是 ARM+FPGA。ARM 精于控制和易于搭建操作系统，FPGA 具有强大的并行计算能力。疲劳驾驶检测系统采用软硬协同的方法实现，可以充分发挥 ARM 和 FPGA 的各自优点。

FPGA 部分主要有人脸检测 IP、VDMA（Video Direct Memory Access）和 HDMI 接口等模块。VDMA 是 FPGA 和 DDR 之间的图像数据传输通道，HDMI 接口用来完成图像的高清显示。人脸检测 IP 使用 Vivado HLS（High-Level Synthesis）高层次综合工具设计，采用基于图片缩放的 AdaBoost 算法。其主要包含图像缩放、积分图计算、图像遍历、分类器检测、窗口合并等模块。图像缩放采用双线性插值法，利用积分图增量算法获得积分图，图像遍历通过设置行、列步长加速窗口遍历，窗口合并采用均值法。针对硬件设计，分类器检测模块采用串并行结合的结构加速待检窗口检测。

ARM 部分主要有 Linux 操作系统、疲劳特征判断、硬件 IP 驱动和操作界面等模块。硬件 IP 驱动控制 VDMA 和人脸检测 IP 运行，操作界面是用户和整个系统的交互界面。疲劳特征判断主要包括人脸分割、光照校正、边缘检测、特征拟合及判断、疲劳判断等。其中人脸分割通过“三庭五眼”的规律分割出人眼和嘴巴区域，光照校正采用的是自适应 Gamma 校正算法，边缘检测采用 Canny 算法，眼睛和嘴巴特征提取分别采用椭圆拟合和外接矩形的方法，通过长短轴之比和矩形的高来判断眼睛和嘴巴的状态，特征拟合判断的结果结合 PERCLOS 准则判断驾驶员疲劳状态。

本文在 ZedBoard 开发板上采用软硬协同设计的方法实现了疲劳驾驶检测系统，系统具有便携性、功耗低、实时性好等特点。

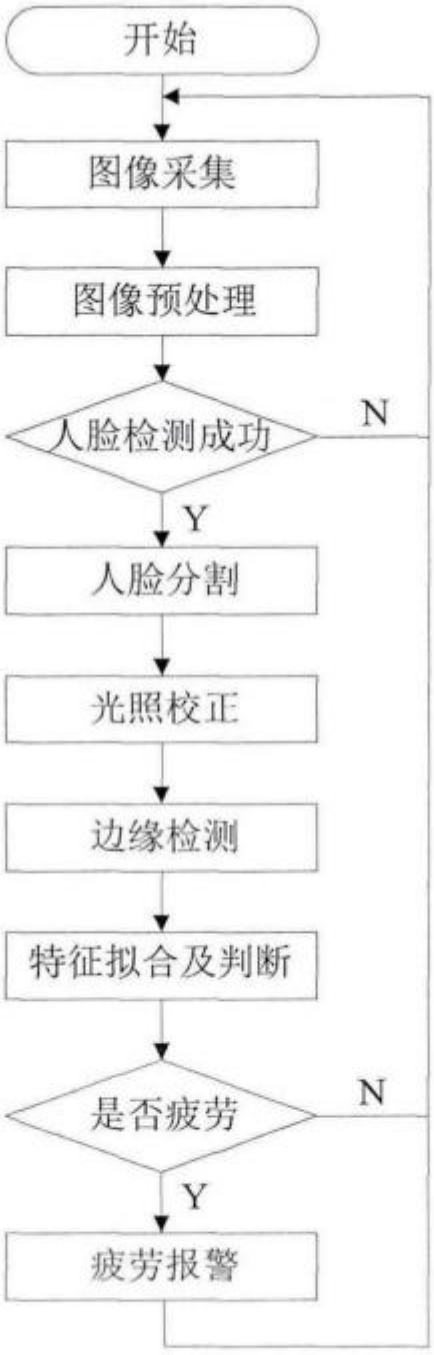


图 2.1　疲劳驾驶检测流程

Fig.2.1　The flow chart of driver fatigue detection

人眼定位，不仅定位速度慢，而且定位的准确度不高[21]。

2）检测虹膜法

人眼区域是一个近似椭圆型区域，并且形状易变化，没有一个固定的数学表达式可以表示。虹膜在眼睛里面，它的形状规则，不管任何情况下都是圆形，所以人眼定位中选取虹膜这个处理目标，通过虹膜的定位实现人眼定位。具体方法有两个，一个是利用虹膜圆形的特征采用 Hough 变化法进行虹膜的定位[22]。第二是利用虹膜区域和眼白区域的灰度差，使用灰度积分的方法定位虹膜 [23]。

用人眼的状态检测驾驶员的疲劳程度，先要对驾驶员的脸部进行检测，得到人脸及面部轮廓信息。虽然检测环境较为简单，但考虑到实时性的要求，采用快速的Sobel边缘检测算法。FPGA具有并行处理结构经过配置后可以满足对图像边缘检测的实时性要求，所以在FPGA中实现Sobel边缘检测算法。

人眼定位中先通过灰度积分投影的方法估计出人眼区域，再使用Hough变换检测虹膜，得到虹膜中心坐标，定位人眼。

本章基本处理的流程如图3.1所示：

图3.1 人眼检测和人眼定位的处理流程

建好，这将影响跟踪的效果[36]。

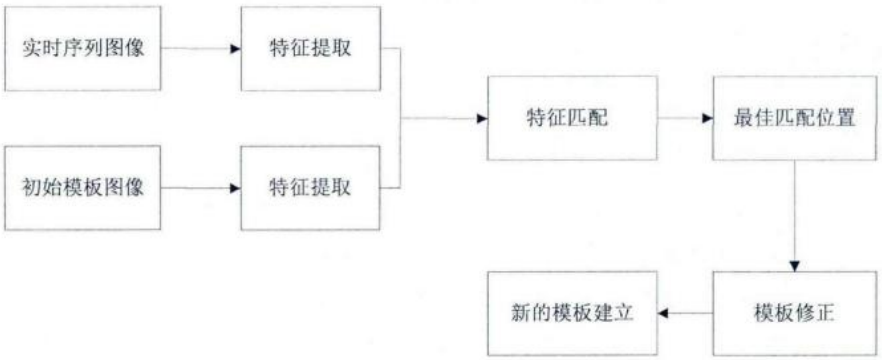


图4.1 特征点跟踪算法处理流程