2019 年 SEU-Xilinx 国际暑期学校团队项目设计文档

(Project Paper Submission Template)

作品名称	基于 SEA-S7 平台的小球追踪
组员姓名	王路 陈光磊 姜亚超 田蕾
房间号及桌号	房间号：717 桌号：15

第一部分

小组成员分工

（各成员任务分配）

	姓名	任务分配
组长	王路	机械臂调试，SDK 部分代码编写并控制机械臂。
组员 1	陈光磊	sobel 算法实现及 IP 核生成，机械臂调试。
组员 2	姜亚超	目标位置计算及 IP 核生成，PPT 制作。
组员 3	田蕾	霍夫变换算法编写，技术文档撰写。

第二部分

设计概述 /Design Introduction

（请简要描述一下你的设计：1. 功能说明；2. 所用到的设备清单）

1.功能说明：本项目基于 SEA-S7 上已有的颜色识别与追踪案例，增加舵机控制功能来追踪小球，同时将颜色识别与追踪改为形状识别与追踪。

2.所用到的设备清单：

1>.Camera(OV5640 MIPI)

2>.SEA-S7

3>.HDMI Out

4>.机械臂

第三部分

详细设计 /Detailed Design

（请详细说明你作品要实现的所有功能以及如何组建系统以实现该功能，还包括为实现该功能需要用到的所有参数和所有操作的详细说明，必要的地方多用图表形式表述）

如因文档篇幅限制无法详述，可提供附件。

1. 将颜色识别与追踪改为形状识别与追踪

在将颜色识别改为形状识别之前，我们先学习清楚所给示例中是如何进行颜色识别的。该系统分为如下模块：

摄像头模块：小球追踪平台的摄像头采用 OV5640 摄像头，pclk 是像素时钟，作为输入，来自外部摄像头，依据此时钟来采样摄像头数据；vsync 是帧同步信号，作为输入，来自外部摄像头，采集完一帧会有一个 vsync 脉冲产生。

I^2C 模块和 uart 模块用于实现串口通信。

RGB to DVI video encoder 模块中，hsync 是水平同步信号，psync 是垂直同步信号，pixelclk 是像素时钟信号，serialclk 是连续时钟信号，输出视频。

bayer2rgb_v1_0 模块是将相机原始信号图片转换为 rgb 格式。

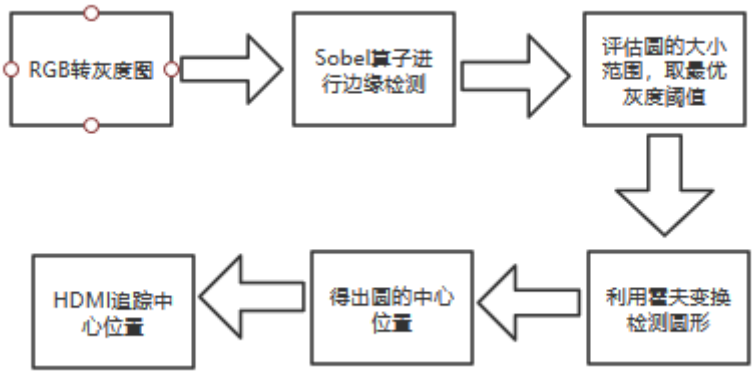
色彩空间转换模块 RGB2HSV 将输入的 RGB 像素转换成 HSV 像素，此 ip 内部调用了低延迟的除法器实现 Hue 分量与 Saturation 分量的高速计算。

色彩提取与坐标计算模块 ColorDetect 借助提取的 HSV 分量，之后每一帧都进行如下操作，遍历图像灭个像素，对 HSV 分量和小球相近的像素进行标注，然后对标注后的区域进行去噪，去噪后，对此区域进行中心点的计算，每一帧结束后都会将计算所得的中心坐标发送给舵机控制模块。该模块主要功能为检测颜色、标注并上色，同时计算标注区域的中心坐标。降噪是直接去除面积较小的噪点，在 ColorDetectip 中，有一个权值计算模块，中心点的计算会和权值相关，对于每个标注像素，都计算此像素和上一帧得到的中心点的距离，根据距离远近设定不同大小的权值，越接近中心的标注像素，权值

越大，否则越小。这样就会降低和乒乓球实际球体距离较远，但面积较大的噪点的影响。将待处理信号引出到其他 ip，erode ip 和 dilate ip，先通过 erode ip 对图像进行腐蚀，去掉小的噪点，但同时对非噪声区域也削去了一部分边缘；然后再经过 dilate ip 对图像进行膨胀，恢复被削去的非噪声区域的边缘。

因此，在所给模块中已经实现了视频输出、削除噪声、标注并上色以及计算标注区域的中心坐标这几大功能，所以我们需要解决的问题是处理帧缓存中的信号，识别不同形状的小球。识别形状，与轮廓检测有相似之处，轮廓检测有多种方案：1.Hu 不变矩原理 2.霍夫检测 3.sobel 检测

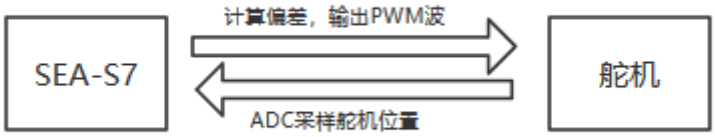
最终选定 sobel 边缘检测和 hough 圆检测实现圆形识别并进行追踪，系统框图如下：



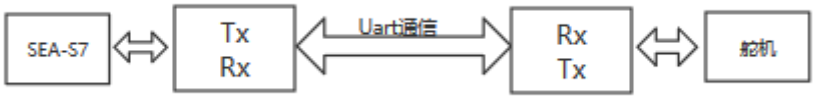
原颜色识别沿用得出小球中心坐标后的模块及摄像头得到 RGB 部分的模块。

2. 增加舵机控制功能实现追踪小球：

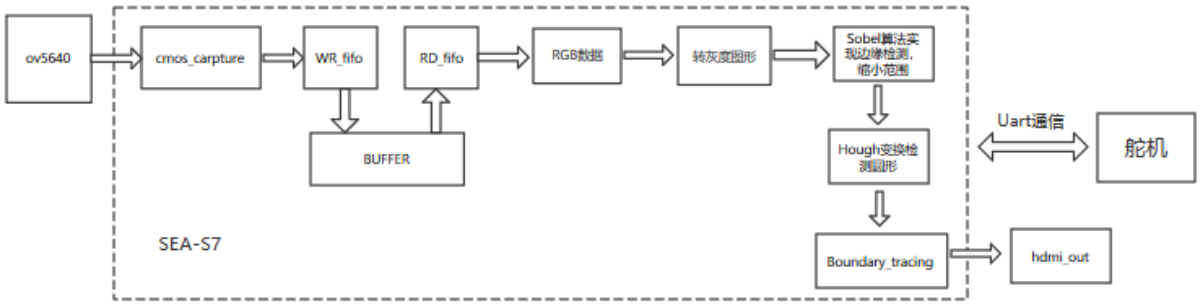
1>.最初为实现追踪，需要水平方向、竖直方向各一个舵机以组成二自由度平台。图像每一帧都会得到一组小球中心点的横坐标，在帧同步信号 vsync_in 的驱动下根据 ADC 采样值计算出当前的舵机角度，同时计算出小球中心点坐标与画面中心点坐标的距离，将两个数值的单位统一换算成 pwm，通过舵机当前位置和小球在图像中与中心点的偏差，计算下一时刻舵机的位置，舵机实现 pwm 输出。但是该方案只适用于四线舵机，在此次培训中，我们使用三线舵机，缺少反馈信号线，无法实现以电压的形式返回舵机当前的位置。



2>.之后改用串口通信，通过将 arduino 的库文件转换为适用于 FPGA 的库文件，通过串口发送指令给舵机，从控制板接收信息，通过调整屏幕中追踪到的小球的位置和屏幕中心的坐标差距，实现舵机的控制。



系统总框图如下所示：



第四部分

完成情况及性能参数 /Final Design & Performance Parameters

（作品完成情况，附照片/视频链接）

1. 实现 FPGA 通过 uart 通信控制舵机
2. 在 HLS 中实现 sobel 边缘检测和 hough 检测圆

第五部分

项目总结 /Conclusions

(项目中所用到的知识点，项目收获及心得)

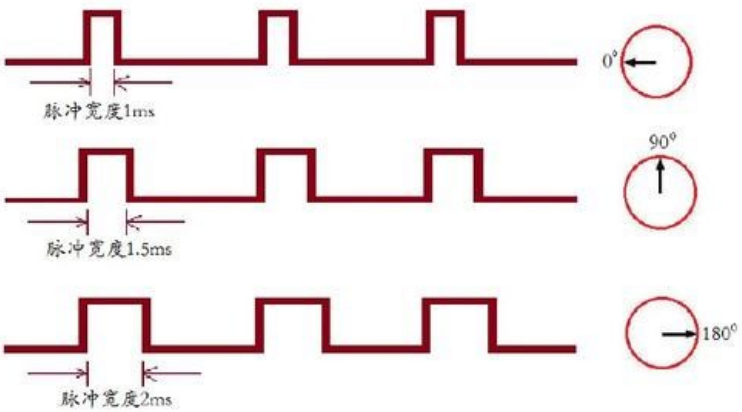
1. ADC 采样控制 PWM 波原理：

参考网站：http://www.51hei.com/bbs/forum.php?mod=viewthread&tid=48655

2. PWM 波控制舵机原理：

通过控制线发送可变宽度或脉冲宽度调制（PWM）的电脉冲来控制舵机。存在最小脉冲，最大脉冲和重复率。舵机通常只能在任一方向上转 90°，共转动 180°。电机的中性位置定义为伺服在顺时针或逆时针方向上具有相同的潜在旋转量的位置。发送到电动机的 PWM 确定轴的位置，并且基于通过控制线发送的脉冲的持续时间；该转子将转向所需的位置。舵机希望每 20 毫秒（ms）看到一个脉冲，脉冲长度将决定电机转动的距离。例如，1.5ms 脉冲将使电机转到 90°位置。短于 1.5ms 将其沿逆时针方向移向 0°位置，任何长于 1.5ms 的位置都会使舵机构顺时针方向朝 180°位置转动。

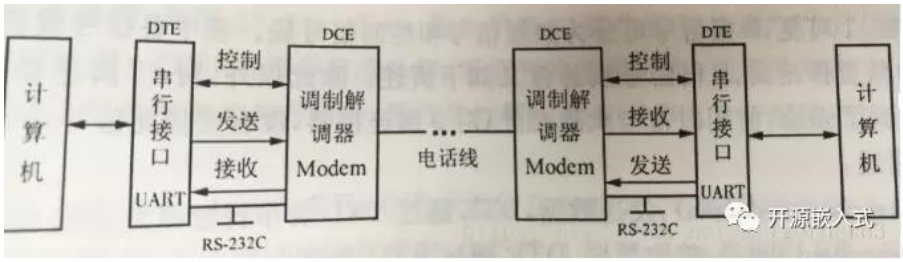
当命令这些舵机移动时，它们将移动到该位置并保持该位置。如果在舵机构保持位置时外力推压伺服机构，舵机构将不会移出该位置。伺服可以施加的最大力量称为伺服的扭矩额定值。悍马不会永远保持他们的位置；必须重复位置脉冲以指示伺服系统保持在原位。



3. 串口通信原理：

串行是与并行想对应的，并行通信是指数据的各位同时被传送。串行通信是将要传送的数据一位位的依次顺序发送。

串行通信实现的是两个对象之间的数据传递，对象通常是单片机。通信实际上是在两个单片机上连上线，通过线路来传递信息。



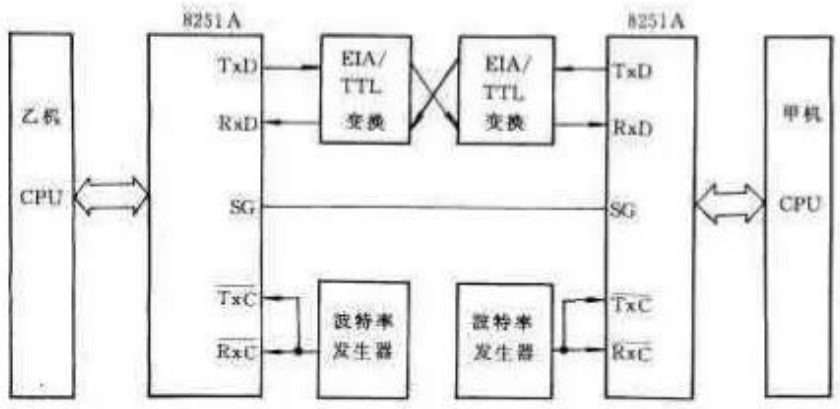


图 10-4 双机串行通信接口

4. Hu 不变矩边缘检测原理：

Hu 不变矩是形状不变矩的一种计算方式，具有平移、比例和旋转不变性，可减少特测图像的大小归一化与位置居中等预处理步骤，降低运算成本，提高运算速率，该方法需调用图像特征提取程序，获得各特测图像的 Hu 不变矩阵值，并将各图像矩阵值与数据库内模板数值进行欧式距离计算，通过将所得欧式距离进行排序，获得不同形状的识别结果。

5. 霍夫变换算法：

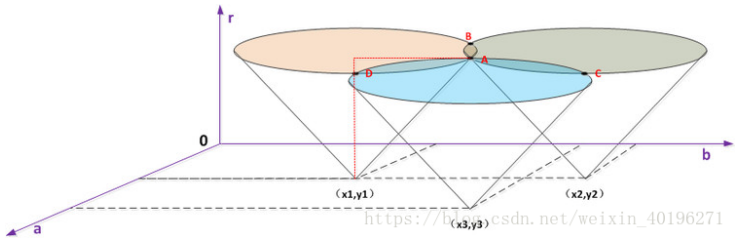
参考网站：https://blog.csdn.net/yuyuntan/article/details/80141392

霍夫检测是通过霍夫变化。

检测直线时，将直角坐标与极坐标之间做变换，通过点映射直线，通过遍历所有像素点，极坐标曲线交点是检测到的可能直线。使用（r,theta）来表示一条直线。其中 r 为该直线到原点的距离，theta 为该直线的垂线与 x 轴的夹角。假设有 N 个点，我们要检测其中的直线，也就是我们要找到具体的 r 和 theata。对于上面所说的每个点可以通过无数条直线，这里我们设为 n 条（通常 n = 180），则我们一起可以找到 Nn 个（r, theata），对这 Nn 个（r,theata），我们可以利用统计学，统计到在 theta=某个值 theta_i 时，多个点的 r 近似相等于 r_i。也就是说这多个点都在直线（r_i,theta_i）上。在实际的直线检测情况中，如果超过一定数目的点拥有相同的（r,theta）坐标，那么就可以判定此处有一条直线。

检测圆时，可知经过一个点可以作出无数个圆，假设某个点平面坐标为(xi,yi)，使用的参数为（ai,bi,ri）则经过此点的圆的表达式为（xi-ai）^2 +（yi-bi） ^ 2=ri ^ 2。对于点（xj,yj），必定存在（aj,bj,rj）使得 近似计算中 ai=aj,bi= bj, ri = rj，即两个点在同一个圆上；同理如果三个点在同一个圆上，则也必须存在 ai=aj =ak = a, bi=bj=bk = b, ri=rj=rk = r 的情况。假设 r 确定，此时点（x,y）又已知，根据（x-a）^2 +（y-b） ^ 2=r ^ 2 ，则（a,b）的轨迹在几何上则变成

了以（x,y）为圆心，r 为半径的圆；而 r 不确定时，（a,b，r）的轨迹变成了以（x,y）为顶点的一个圆锥。则（ai,bi,ri）,(aj,bj,rj)，(ak,bk,rk)的圆为下图中圆锥面的角点 A。



6. sobel 边缘检测算法：

Sobel 算子是像素图像边缘检测中最重要的算子之一，该算子包含两组 3x3 的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如下图，Gx 和 Gy 分别是在横向及纵向的灰度偏导的近似值。

$$G_x=\begin{pmatrix}-1 & 0 & +1\\ -2 & 0 & +2\\ -1 & 0 & +1\end{pmatrix}*A \qquad G_y=\begin{pmatrix}+1 & +2 & +1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{pmatrix}*A$$

对于每一个点我们可以获得两个方向的梯度，我们可以通过下面这个公式算出梯度的估计值。

$$\mathbf{G}=\sqrt{\mathbf{G}_x^{\,2}+\mathbf{G}_y^{\,2}}$$

定义一个阈值 Gmax（这里定义 Gmax = 150），如果 G 比 Gmax 大可以认为该点是一个边界值.则设置这个点为白色否则该店为黑色，这样就得到了通过边缘检测的图像。

第六部分

源代码/Source code

（作品源码 Github 链接，github 操作步骤见 *Github 项目上传简单教程*）

第七部分（技术总结/心得/笔记等分享）

（可分享的技术文档附件或已发表的文章链接）

技术总结：

1. Hu 不变矩原理

几何矩是由 Hu(Visual pattern recognition by moment invariants)在 1962 年提出的，具有平移、旋转和尺度不变性。在连续情况下，图像函数为 $f(x,y)$，那么图像的 p+q 阶几何矩（标准矩）定义为：

$$m_{pq}=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}x^p y^q f(x,y)dxdy \qquad p,q=0,1,2\ldots$$

p+q 阶中心距定义为：

$$\mu_{pq}=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\bar{x})^p(y-\bar{y})^q f(x,y)dxdy \qquad p,q=0,1,2\ldots$$

其中 $\bar{x}$ 和 $\bar{y}$ 代表图像的重心，

$$\bar{x} = m_{10} / m_{00}$$

$$\bar{y} = m_{01} / m_{00}$$

对于离散的数字图像，采用求和号代替积分：

$$m_{pq} = \sum_{y=1}^{N} \sum_{x=1}^{M} x^p y^q f(x, y) \quad p, q = 0,1,2\ldots$$

$$\mu_{pq} = \sum_{y=1}^{N} \sum_{x=1}^{M} (x - \bar{x})^p (y - \bar{y})^q f(x, y) \quad p, q = 0,1,2\ldots$$

N 和 M 分别是图像的高度和宽度；

归一化的中心距定义为：

$\eta_{pq} = \mu_{pq} / (\mu_{00}{}^{\rho})$ ；其中 $\rho = (p+q)/2+1$

利用二阶和三阶归一化中心矩构造了 7 个不变矩 $M1 \sim M7$ ：

$$M1 = \eta_{20} + \eta_{02}$$

$$M2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}{}^2$$

$$M3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$M4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$M5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2)$$
$$+ (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2)$$

$$M7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2)$$
$$- (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2)$$

这 7 个不变矩构成一组特征量，Hu.M.K 在 1962 年证明了他们具有旋转，缩放和平移不变性。

实际上，在对图片中物体的识别过程中，只有 $M1$ 和 $M2$ 不变性保持的比较好，其他的几个不变矩带来的误差比较大，有学者认为只有基于二阶矩的不变矩对二维物体的描述才是真正的具有旋转、缩放和平移不变性（ $M1$ 和 $M2$ 刚好都是由二阶矩组成的）。不过我没有证明是否是真的事这样的。

```
double M[7] = {0};              //HU 不变矩
 bool HuMoment(IplImage* img)
 {
 int bmpWidth = img->width;
 int bmpHeight = img->height;
 int bmpStep = img->widthStep;
 int bmpChannels = img->nChannels;
       uchar*pBmpBuf = (uchar*)img->imageData;

double m00=0,m11=0,m20=0,m02=0,m30=0,m03=0,m12=0,m21=0;   //中心矩
double x0=0,y0=0;      //计算中心距时所使用的临时变量（x-x'）
double u20=0,u02=0,u11=0,u30=0,u03=0,u12=0,u21=0;//规范化后的中心矩
//double M[7];      //HU 不变矩
double t1=0,t2=0,t3=0,t4=0,t5=0;//临时变量，
//double Center_x=0,Center_y=0;//重心
int Center_x=0,Center_y=0;//重心
int i,j;                  //循环变量

//   获得图像的区域重心(普通矩)
double s10=0,s01=0,s00=0;   //0 阶矩和 1 阶矩
for(j=0;j<bmpHeight;j++)//y
      {
for(i=0;i<bmpWidth;i++)//x
            {
                  s10+=i*pBmpBuf[j*bmpStep+i];
                  s01+=j*pBmpBuf[j*bmpStep+i];
                  s00+=pBmpBuf[j*bmpStep+i];
            }
      }
      Center_x=(int)(s10/s00+0.5);
      Center_y=(int)(s01/s00+0.5);

//   计算二阶、三阶矩(中心矩)
      m00=s00;
for(j=0;j<bmpHeight;j++)
      {
for(i=0;i<bmpWidth;i++)//x
            {
                  x0=(i-Center_x);
                  y0=(j-Center_y);
                  m11+=x0*y0*pBmpBuf[j*bmpStep+i];
                  m20+=x0*x0*pBmpBuf[j*bmpStep+i];
                  m02+=y0*y0*pBmpBuf[j*bmpStep+i];
                  m03+=y0*y0*y0*pBmpBuf[j*bmpStep+i];
                  m30+=x0*x0*x0*pBmpBuf[j*bmpStep+i];
```

```
                m12+=x0*y0*y0*pBmpBuf[j*bmpStep+i];
                m21+=x0*x0*y0*pBmpBuf[j*bmpStep+i];
            }
        }

    //  计算规范化后的中心矩: mij/pow(m00,((i+j+2)/2)
        u20=m20/pow(m00,2);
        u02=m02/pow(m00,2);
        u11=m11/pow(m00,2);
        u30=m30/pow(m00,2.5);
        u03=m03/pow(m00,2.5);
        u12=m12/pow(m00,2.5);
        u21=m21/pow(m00,2.5);

    //  计算中间变量
        t1=(u20-u02);
        t2=(u30-3*u12);
        t3=(3*u21-u03);
        t4=(u30+u12);
        t5=(u21+u03);

    //  计算不变矩
        M[0]=u20+u02;
        M[1]=t1*t1+4*u11*u11;
        M[2]=t2*t2+t3*t3;
        M[3]=t4*t4+t5*t5;
        M[4]=t2*t4*(t4*t4-3*t5*t5)+t3*t5*(3*t4*t4-t5*t5);
       M[5]=t1*(t4*t4-t5*t5)+4*u11*t4*t5;
        M[6]=t3*t4*(t4*t4-3*t5*t5)-t2*t5*(3*t4*t4-t5*t5);

    return true;
    }
```

相似性准则：

```
//  计算相似度 1
double dbR =0; //相似度
double dSigmaST =0;
double dSigmaS =0;
double dSigmaT =0;
double temp =0;
    {for(int i=0;i<7;i++)
    {
        temp = fabs(Sa[i]*Ta[i]);
        dSigmaST+=temp;
        dSigmaS+=pow(Sa[i],2);
        dSigmaT+=pow(Ta[i],2);
```

```
    }}
    dbR = dSigmaST/(sqrt(dSigmaS)*sqrt(dSigmaT));
```

2. 灰度处理：

浮点算法实现灰度图，先将数据放大，然后再缩小。

```
always   @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        red_r1    <= 0 ;
        green_r1 <= 0 ;
        blue_r1   <= 0 ;
    end
    else begin
        red_r1    <= red    * 77 ;          //放大后的值
        green_r1 <= green * 150;
        blue_r1   <= blue   * 29 ;
    end
end

always   @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
         Gray <= 0;      // 三个数之和      end
    else begin
         Gray <= red_r1 + green_r1 + blue_r1;
    end
end

always   @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        post_data_in <= 0;   //输出的灰度数据      end
    else begin
        post_data_in <= { Gray[13:9], Gray[13:8], Gray[13:9] };//将 Gray 值赋值给 RGB 三个通道      end
end
```

3. 均值滤波

```
reg [5:0]p_11,p_12,p_13;   // 3 * 3  卷积核中的像素点
reg [5:0]p_21,p_22,p_23;
reg [5:0]p_31,p_32,p_33;
reg [8:0]mean_value_add1,mean_value_add2,mean_value_add3;//每一行之和


always   @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
         {p_11,p_12,p_13} <= {5'b0,5'b0,5'b0}    ;
         {p_21,p_22,p_23} <= {15'b0,15'b0,15'b0};
         {p_31,p_32,p_33} <= {15'b0,15'b0,15'b0};
    end
    else   begin
```

```
        if(per_href_ff0==1&&flag_do==1)begin
            {p_11,p_12,p_13}<={p_12,p_13,row_1};
            {p_21,p_22,p_23}<={p_22,p_23,row_2};
            {p_31,p_32,p_33}<={p_32,p_33,row_3};
        end
        else begin
             {p_11,p_12,p_13}<={5'b0,5'b0,5'b0};
             {p_21,p_22,p_23}<={5'b0,5'b0,5'b0}
             {p_31,p_32,p_33}<={5'b0,5'b0,5'b0}
        end
    end
end



always   @(posedge clk or negedge rst_n)begin
      if(rst_n==1'b0)begin
            mean_value_add1<=0;
            mean_value_add2<=0;
            mean_value_add3<=0;
      end
      else if(per_href_ff1)begin
            mean_value_add1<=p_11+p_12+p_13;
            mean_value_add2<=p_21+     0     +p_23;
            mean_value_add3<=p_31+p_32+p_33;
      end
end

wire [8:0]mean_value;//8 位数之和
wire [5:0]fin_y_data; //平均数，除以 8，相当于左移三位。

assign mean_value=mean_value_add1+mean_value_add2+mean_value_add3;
assign fin_y_data=mean_value[8:3];
4.  Sobel 边缘检测
reg [8:0] p_x_data ,p_y_data ;   // x  和  y  的正值之和
reg [8:0] n_x_data ,n_y_data ; // x  和  y  的负值之和
reg [8:0] gx_data   ,gy_data   ; //最终结果

always   @(posedge clk or negedge rst_n)begin
      if(rst_n==1'b0)begin
          p_x_data <=0;
          n_x_data <=0;
          gx_data     <=0;
      end
      else if(per_href_ff1==1) begin
```

```
            p_x_data <= p_13 + (p_23<<1) + p_33 ;
            n_x_data <= p_11 + (p_12<<1 )+ p_13 ;
            gx_data     <= (p_x_data >=n_x_data)? p_x_data - n_x_data : n_x_data - p_x_data ;
        end
        else begin
             p_x_data<=0;
             n_x_data<=0;
             gx_data <=0;
        end
end

always   @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
            p_y_data <=0;
            n_y_data <=0;
            gy_data     <=0;
        end
        else if(per_href_ff1==1) begin
            p_y_data <= p_11 + (p_12<<1) + p_13 ;
            n_y_data <= p_31 + (p_32<<1) + p_33 ;
            gy_data     <= (p_y_data >=n_y_data)? p_y_data - n_y_data : n_y_data - p_y_data ;
        end
        else begin
            p_y_data <=0;
            n_y_data <=0;
            gy_data     <=0;
    end
end

//求平方和,调用 ip 核开平方
reg [16:0] gxy; // Gx 与 Gy 的平方和
always   @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
            gxy<=0;
        end
        else begin
            gxy<= gy_data* gy_data + gx_data* gx_data ;
        end
end

wire [8:0] squart_out ;
altsquart   u1_altsquart (        //例化开平方的 ip 核
        .radical (gxy),
        .q          (squart_out),   //输出的结果
        .remainder()
```

```
                                    );

//与阈值进行比较
reg [15:0] post_y_data_r;
always    @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
                post_y_data_r<=16'h00;
        end
        else if(squart_out>=threshold)
                 post_y_data_r<=16'h00    ;
        else
                 post_y_data_r<=16'hffff    ;

end
```

5. 几何形状识别：

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <math.h>

using namespace cv;
using namespace std;


int main(int argc, char** argv)
{
        Mat src = imread("3 input.bmp", IMREAD_GRAYSCALE);
        Mat binary, dst = Mat::zeros(src.size(), CV_8UC3);
        Mat Triangle = dst.clone(), Rect1 = dst.clone(), BigCircle = dst.clone(), SmallCircle = dst.clone();

        if (src.empty()) {
                printf("Could not load image...");
                return -1;
        }
        src = ~src;//取反
        imshow("原图", src);
        //二值化
        threshold(src, binary, 0, 255, THRESH_BINARY | THRESH_OTSU);

        //发现轮廓
        vector<vector<Point>> contours;
        vector<Vec4i> hireachy;
        findContours(binary, contours, hireachy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point());

        //面积删选
        for (size_t t = 0; t < contours.size(); t++)
```

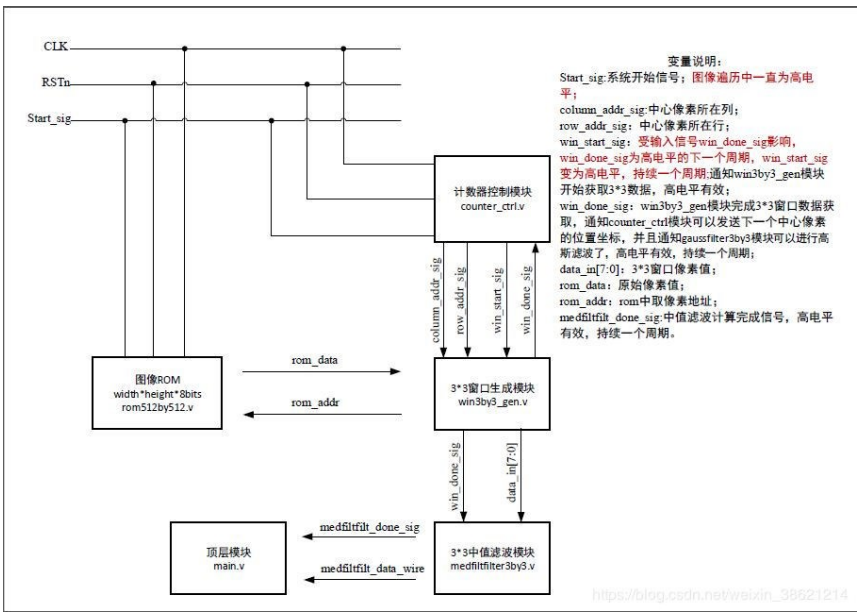
```
{
    double area = contourArea(contours[t]);
    if (area < 40000) continue;//将面积小于 40000 的去掉
    drawContours(Triangle, contours, t, Scalar(0, 0, 255), 2, 8, Mat(), 0, Point());
}

for (size_t t = 0; t < contours.size(); t++)
{
    double area = contourArea(contours[t]);
    if (area > 40000 || area<20000) continue;//将面积小于 40000 的去掉
    drawContours(BigCircle, contours, t, Scalar(0, 0, 255), 2, 8, Mat(), 0, Point());
}

for (size_t t = 0; t < contours.size(); t++)
{
    double area = contourArea(contours[t]);
    if (area > 20000 || area<15000) continue;//将面积小于 40000 的去掉
    drawContours(Rect, contours, t, Scalar(0, 0, 255), 2, 8, Mat(), 0, Point());
}

for (size_t t = 0; t < contours.size(); t++)
{
    double area = contourArea(contours[t]);
    if (area > 15000) continue;//将面积小于 40000 的去掉
    //其他过滤方法

    /*//横纵比过滤
    Rect rect= boundingRect(contours[t]);//返回最小外接矩形
    float ratio = float(rect.width) / float(rect.height);//计算横纵比
    if (ratio<1.1&&ratio>0.9) {}

    //周长过滤
    float length = arcLength(contours[t], true);//计算轮廓长度

    */


    drawContours(SmallCircle, contours, t, Scalar(0, 0, 255), 2, 8, Mat(), 0, Point());
}
imshow("Triangle", Triangle);
imshow("BigCircle", BigCircle);
imshow("Rect", Rect1);
imshow("SmallCircle", SmallCircle);

waitKey(0);
```

```
    return 0;
}
```

6. verilog 实现中值滤波

整个中值滤波模块分为几个小的模块：3*3 窗口生成模块、计数器控制模块、3*3 中值滤波模块、顶层模块以及最后的测试模块 testbench 的编写。

整个框架的设计如下图所示：



1>.3*3 窗口生成模块，用于生成滤波的滑动窗口，得到窗口内的所有元素数据。

功能：

(1)根据中心像素点得到所在其所在的行、列位置；

(2)根据该模块的开始信号设计得到获取数据的有效时间序列；

(3)在读取数据的有效时序内，得到窗口内的所有元素数据；

(4)窗口数据的获取按照一定的时序顺序来获得。

(5)根据中心像素点的行、列位置信息得到每个窗口元素的 ROM 地址，根据某一时刻 ROM 地址，下一时刻调用 ROM 模块得到对应的元素数据，下一时刻将数据锁存，然后再读取该地址的数据；所以要注意地址和数据的获取不是在同一时刻，而是需要延迟两个时刻。

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    09:27:48 05/18/2016
// Design Name:
// Module Name:    win3by3_gen
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module win3by3_gen(
  CLK,
  RSTn,
  center_pix_sig,
  cols,   // the column numbers of the input image
  rows,
  rom_data_win,   //input-from U1;
  column_addr_sig,    //input-from U3; //output [9 : 0] addra;
  row_addr_sig,         //input-from U3; //output [9 : 0] addra;
  rom_addr_sig,            //output-to U1;
  data_out0,           //output-to U4;
  data_out1,
  data_out2,
  data_out3,
  data_out4,
  data_out5,
  data_out6,
  data_out7,
  data_out8,
  win_data_done_sig            //output-to U4/U3;complete the win data;
    );

  input CLK;
  input RSTn;
  input [7:0] rom_data_win;
  input [9:0] cols;
  input [9:0] rows;
  input center_pix_sig;  //
  input [9:0] column_addr_sig;
  input [9:0] row_addr_sig;

  output [7:0] data_out0;           //output-to U4;
  output [7:0] data_out1;
  output [7:0] data_out2;
  output [7:0] data_out3;
  output [7:0] data_out4;
  output [7:0] data_out5;
  output [7:0] data_out6;
```

```
    output [7:0] data_out7;
    output [7:0] data_out8;
    output [17:0] rom_addr_sig;
    output win_data_done_sig;

/*****************************************************************************************/

    reg [9:0] m;

    always @ ( posedge CLK or negedge RSTn )
      if ( !RSTn )
        m <= 10'd1;
       else if (  center_pix_sig )
        m <= row_addr_sig[9:0];


/*****************************************************************************************/

    reg [9:0] n;

    always @ ( posedge CLK or negedge RSTn )
      if ( !RSTn )
        n <= 10'd1;
       else if (  center_pix_sig )
        n <= column_addr_sig[9:0];


/*****************************************************************************************/

    reg [3:0] i;
    reg isWinDone;
    reg [17:0] rom_addr;
    reg [7:0] a11;
    reg [7:0] a12;
    reg [7:0] a13;
    reg [7:0] a21;
    reg [7:0] a22;
    reg [7:0] a23;
    reg [7:0] a31;
    reg [7:0] a32;
    reg [7:0] a33;

/*****************************************************************************************/

reg get_9point_vld;
```

```
always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
          get_9point_vld <= 1'b0;
     else if ( center_pix_sig )
           get_9point_vld <= 1'b1;
     else if ( i==4'd10 )
           get_9point_vld <= 1'b0;


always @ ( posedge CLK or negedge RSTn )
    if ( !RSTn )
          isWinDone <= 1'b0;
     else if ( i==4'd10 )
           isWinDone <= 1'b1;
     else
           isWinDone <= 1'b0;



always @ ( posedge CLK or negedge RSTn )
    if ( !RSTn )
          i <= 4'd0;
     else if (i == 4'd10)
           i <= 4'd0;
     else if ( get_9point_vld )
           i <= i + 1'b1;




always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
          rom_addr <= 0;
     else if ( get_9point_vld)
       case (i)
          4'd0:
            if(!(m==1 || n==1)) rom_addr <= (m-2)*cols + (n-1) -1;

          4'd1:
          if(!(m==1 )) rom_addr <= (m-2)*cols + n -1;

          4'd2:
            if(!(m==1 || n==cols)) rom_addr <= (m-2)*cols + (n+1) -1;
```

```
            4'd3:
              if(!(n==1)) rom_addr <= (m-1)*cols + (n-1) -1;

            4'd4:
              rom_addr <= (m-1)*cols + n -1;

            4'd5:
              if(!(n==cols)) rom_addr <= (m-1)*cols + (n+1) -1;

            4'd6:
              if(!(m==cols || n==1)) rom_addr <= m*cols + (n-1) -1;

            4'd7:
              if(!(m==cols)) rom_addr <= m*cols + n -1;

            4'd8:
              if(!(m==cols || n==cols)) rom_addr <= m*cols + (n+1) -1;

            default:;

          endcase

always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
       begin
          a11 <= 0;
          a12 <= 0;
          a13 <= 0;
          a21 <= 0;
          a22 <= 0;
          a23 <= 0;
          a31 <= 0;
          a32 <= 0;
          a33 <= 0;
        end
     else if ( get_9point_vld )

       case (i)

          4'd2:
          if ( m==1 || n==1 )
                a11 <= 0;
          else
                a11 <= rom_data_win;
```

```
            4'd3:
            if ( m==1 )  a12 <= 0;
            else a12 <= rom_data_win;

            4'd4:
            if ( m==1 || n==cols ) a13 <= 0;
            else a13 <= rom_data_win;

            4'd5:
            if ( n==1 ) a21 <= 0;
            else  a21 <= rom_data_win;

            4'd6:
            a22 <= rom_data_win;

            4'd7:
            if ( n==cols ) a23 <= 0;
            else a23 <= rom_data_win;

            4'd8:
            if ( m==cols || n==1 ) a31 <= 0;
            else a31 <= rom_data_win;

            4'd9:
            if ( m==cols ) a32 <= 0;
            else a32 <= rom_data_win;

            4'd10:
            if ( m==cols || n==cols ) a33 <= 0;
            else a33 <= rom_data_win;

            default:;

        endcase

/*********************************************************************************************/

  assign win_data_done_sig = isWinDone;
  assign rom_addr_sig = rom_addr;

  assign data_out0 = a11;
  assign data_out1 = a12;
  assign data_out2 = a13;
  assign data_out3 = a21;
  assign data_out4 = a22;
```

```
  assign data_out5 = a23;
  assign data_out6 = a31;
  assign data_out7 = a32;
  assign data_out8 = a33;

/*********************************************************************************************/

endmodule
```

2>.计数器控制模块，主要用于获得中心像素点的地址信息。

(1)系统模块开始信号之后开始获取第一个中心像素点，注意初始化信号值和系统开始的信号值的区别；

(2)该时刻得到的的数据将在下一个时刻产生结果，该时刻的数据并没有改变；

(3)注意中心像素点的行、列位置信息的计算；

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    09:28:59 05/18/2016
// Design Name:
// Module Name:    counter_ctrl
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module counter_ctrl(
  CLK,
  RSTn,
  start_sig,  //input-from top
  nxt_pix_sig,     //input-from --start next center point pixel
  cols,
  column_addr_sig,  //output
  row_addr_sig,     //output-to
  pix_done_sig   //output-to
    );

  input CLK;
```

```
input RSTn;
input start_sig;
input nxt_pix_sig;
input [9:0] cols;

output pix_done_sig;
output [9:0] column_addr_sig;
output [9:0] row_addr_sig;

/*****************************************************************************************/

reg isCtrlDone;
//reg isWinStart;
reg [17:0] imk;   //The k-th pixel of the image
reg [9:0] row_addr;  // The row of the centeral pixel
reg [9:0] column_addr;   // The column of the centeral pixel

reg start_sig_d;

wire start_sig_rising_vld;

 always @ (posedge CLK or negedge RSTn)   //Asynchronous reset
  if (!RSTn)
      start_sig_d <= 0;
   else
      start_sig_d <= start_sig;

assign start_sig_rising_vld = start_sig & (~start_sig_d);

always @ (posedge CLK or negedge RSTn)   //Asynchronous reset
  if (!RSTn)
     begin
      imk <= 18'b0;
      column_addr <= 10'b0;
      row_addr <= 10'b0;
      isCtrlDone <= 1'b0;
      end
   else if (start_sig_rising_vld)
     begin
      imk <= 18'b1;
      column_addr <= 10'b1;
      row_addr <= 10'b1;
      isCtrlDone <= 1'b1;
      end
   else if ( nxt_pix_sig )
```

```
        begin
           imk <= imk + 1'b1;
           row_addr <= imk / cols + 1;
           column_addr <= imk % cols + 1;
         isCtrlDone <= 1'b1;
         end
     else isCtrlDone <= 1'b0;

/*****************************************************************************************/

  assign row_addr_sig = row_addr;
  assign column_addr_sig = column_addr;
  assign pix_done_sig = isCtrlDone;

/*****************************************************************************************/
endmodule
```

3>3*3 中值滤波模块

功能：得到某一中心像素点的 3*3 滑窗区域的灰度值的中值，作为中心像素点的值。

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    09:28:20 05/18/2016
// Design Name:
// Module Name:    medfilter3by3
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module medfilter3by3(
    CLK,
    RSTn,
    win_data_sig,       //input-from module of win3by3_gen;
    medfilt_done_sig,   //output-to top;
    data_in0,           //input-from module of win3by3_gen;
    data_in1,
```

```
data_in2,
data_in3,
data_in4,
data_in5,
data_in6,
data_in7,
data_in8,
medfilt_data_out    //output-to top;
);

input CLK;
input RSTn;
input win_data_sig;
input [7:0] data_in0;           //output-to ;
input [7:0] data_in1;
input [7:0] data_in2;
input [7:0] data_in3;
input [7:0] data_in4;
input [7:0] data_in5;
input [7:0] data_in6;
input [7:0] data_in7;
input [7:0] data_in8;

output medfilt_done_sig;
output [7:0] medfilt_data_out;

/*****************************************************************************/
reg [7:0] a11;
reg [7:0] a12;
reg [7:0] a13;
reg [7:0] a21;
reg [7:0] a22;
reg [7:0] a23;
reg [7:0] a31;
reg [7:0] a32;
reg [7:0] a33;

reg [7:0] b11;
reg [7:0] b12;
reg [7:0] b13;
reg [7:0] b21;
reg [7:0] b22;
reg [7:0] b23;
reg [7:0] b31;
reg [7:0] b32;
```

```
reg [7:0] b33;

reg [7:0] c11;
reg [7:0] c12;
reg [7:0] c13;
reg [7:0] c21;
reg [7:0] c22;
reg [7:0] c23;
reg [7:0] c31;
reg [7:0] c32;
reg [7:0] c33;

reg [2:0] i;
reg [7:0] medfilt_data;
reg filt_done;

reg cal_vld;


always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
        begin
            a11 <= 0;
            a12 <= 0;
            a13 <= 0;
            a21 <= 0;
            a22 <= 0;
            a23 <= 0;
            a31 <= 0;
            a32 <= 0;
            a33 <= 0;
        end
    else if (win_data_sig)
        begin
            a11 <= data_in0;
            a12 <= data_in1;
            a13 <= data_in2;
            a21 <= data_in3;
            a22 <= data_in4;
            a23 <= data_in5;
            a31 <= data_in6;
            a32 <= data_in7;
            a33 <= data_in8;
        end
```

```
always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
        i <= 3'd0;
    else if( cal_vld & ( i!=3 ) )
        i <= i + 1;
    else
        i <= 0;
always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
        cal_vld <= 1'b0;
    else if( win_data_sig )
        cal_vld <= 1'b1;
    else if( i==3'd3 )
        cal_vld <= 0;
always @ ( posedge CLK or negedge RSTn )
    if (!RSTn)
        begin
            filt_done <= 1'b0;
            b11 <= 0;
            b12 <= 0;
            b13 <= 0;
            b21 <= 0;
            b22 <= 0;
            b23 <= 0;
            b31 <= 0;
            b32 <= 0;
            b33 <= 0;
            c11 <= 0;
            c12 <= 0;
            c13 <= 0;
            c21 <= 0;
            c22 <= 0;
            c23 <= 0;
            c31 <= 0;
            c32 <= 0;
            c33 <= 0;
            medfilt_data <= 0;
        end
    else if( cal_vld )
        case(i)
            3'd0:
                begin
                    b11 <= max(a11, a21, a31);
                    b12 <= max(a12, a22, a32);
                    b13 <= max(a13, a23, a33);
```

```
                        b21 <= med(a11, a21, a31);
                        b22 <= med(a12, a22, a32);
                        b23 <= med(a13, a23, a33);
                        b31 <= min(a11, a21, a31);
                        b32 <= min(a12, a22, a32);
                        b33 <= min(a13, a23, a33);
                    end
                3'd1:
                    begin
                        c31 <= max(b31, b32, b33);
                        c22 <= med(b21, b22, b23);
                        c13 <= min(b11, b12, b13);
                    end
                3'd2:
                    begin
                        medfilt_data <= med(c13, c22, c31);
                        filt_done<=1'b1;
                    end

                3'd3:
                    filt_done <= 1'b0;
                default:;
            endcase
        function [7:0] max;//if the data is signed number, please add the char signed behind
key function;
            input [7:0] a, b, c;
            begin
                max = (((a >= b) ? a : b) >= c ) ?  ((a >= b) ? a : b) : c;
            end
        endfunctio
        function [7:0] med;
            input [7:0] a, b, c;
            begin
                med = a<b?(b<c?b:a<c?c:a) : (b>c?b:a>c?c:a);
            end
        endfunction
        function [7:0] min;
            input [7:0] a, b, c;
            begin
                min= (((a <= b) ? a : b) <= c ) ?  ((a <= b) ? a : b) : c;
            end
        endfunction
        assign medfilt_data_out = medfilt_data;
        assign medfilt_done_sig = filt_done;
    endmodule
```

心得：