数字图像处理课程设计报告

目录

一、 课程设计任务

本次课程设计没有明确的要求，运用数字图像处理的相关知识，用编程实现一个功能即可。在一开始我想做现在应用很广泛的二维码识别，然而在查阅资料的过程中，我认识到二维码译码的复杂性，因此我选择降低难度，做条形码的识别。条形码技术是计算机应用实践中产生和发展起来的一种自动识别技术，它是集条形码理论、光电技术、计算机技术、通信技术、条形码引致技术于一体的综合性技术。现代设备对条形码的识别不但快速准确，而且能提供可靠性很高的数据，误码率可达百万分之一，首读率可达 98%，因而被广泛运用在计算机管理领域之中，如图书管理、商品流通管理等。为了降低图片预处理的工作量，我选择用 opencv 作为辅助工具，用 MFC 写一个对话框程序。。

二、 课程设计原理及设计方案

1. EAN-13 条形码的构成

条形码技术是计算机应用实践中产生和发展起来的一种自动识别技术，它是集条形码理论、光电技术、计算机技术、通信技术、条形码引致技术于一体的综合性技术。现代设备对条形码的识别不但快速准确，而且能提供可靠性很高的数据，误码率可达百万分之一，首读率可达 98%，因而被广泛运用在计算机管理领域之中，如图书管理、商品流通管理等。

条形码是由宽度不同、反射率不同的条和空，按照一定的编码规则编制而成，用来表示一组数字或者字母符号信息的图形标识符。简单来说，条形码是一组粗细不同，按照一定规则安排平行安排间距的平行线条图形。一般条形码的组成如下：

静区：静区也叫空白区，分为左空白区和右空白区，左空白区是让扫描设备做好扫描准备，右空白区是保证扫描设备正确识别条码的结束标记。

为了防止左右空白区（静区）在印刷排版时被无意中占用，可在空白区加印一个符号（左侧没有数字时印<号，右侧没有数字时加印>号）这个符号就叫静区标记。主要作用就是防止静区宽度不足。只要静区宽度能保证，有没有这个符号都不影响条码的识别。

起始字符：第一位字符，具有特殊结构，当扫描器读取到该字符时，便开始正式读取代码了。

数据字符：条形码的主要内容。

校验字符：检验读取到的数据是否正确。不同编码规则可能会有不同的校验规则。

终止字符：最后一位字符，一样具有特殊结构，用于告知代码扫描完毕，同时还起到只是进行校验计算的作用。

在这次课程设计中我选择以 EAN-13 码为对象，这也是比较通用的条形码协议和标准。EAN13 码标准码共 13 位数，系由「国家代码」3 位数，「厂商代码」4 位数，「产品代码」5 位数，以及「校正码」1 位数组成。其排列如下：

国家号码由国际商品条码总会授权，我国的「国家号码」为「690-699」。

厂商代码由国家商品条码策进会核发给申请厂商，占四个码，代表申请厂商的号码。

产品代码占五个码，系代表单项产品的号码，由厂商自由编定。

校正码占一个码，系为防止条码扫描器误读的自我检查。

条码符号组成如下：

左侧空白区：位于条码符号最左侧与空的反射率相同的区域，其最小宽度为 11 个模块宽。

起始符：位于条码符号左侧空白区的右侧，表示信息开始的特殊符号，由 3 个模块组成。

左侧数据符：位于起始符右侧，表示 6 位数字信息的一组条码字符，由 42 个模块组成。

中间分隔符：位于左侧数据符的右侧，是平分条码字符的特殊符号，由 5 个模块组成。

右侧数据符：位于中间分隔符右侧，表示 5 位数字信息的一组条码字符，由

35 个模块组成。

校验符:位于右侧数据符的右侧,表示校验码的条码字符,由 7 个模块组成。

终止符：位于条码符号校验符的右侧，表示信息结束的特殊符号，由 3 个模块组成。

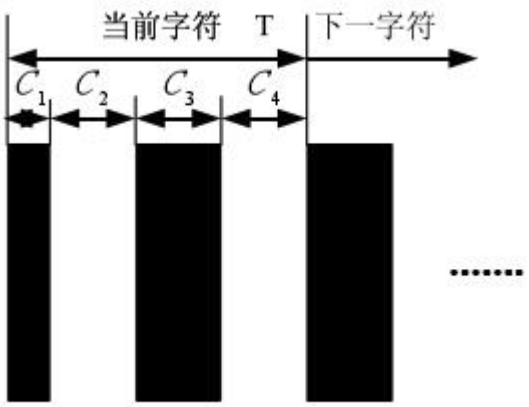
右侧空白区：位于条码符号最右侧的与空的反射率相同的区域，其最小宽度为 7 个模块宽。为保护右侧空白区的宽度，可在条码符号右下角加“>”符号。

供人识读字符：位于条码符号的下方，是与条码字符相对应的供人识别的 13 位数字，最左边一位称前置码。供人识别字符优先选用 OCR-B 字符集，字符顶部和条码底部的最小距离为 0.5 个模块宽。标准版商品条码中的前置码印制在条码符号起始符的左侧。

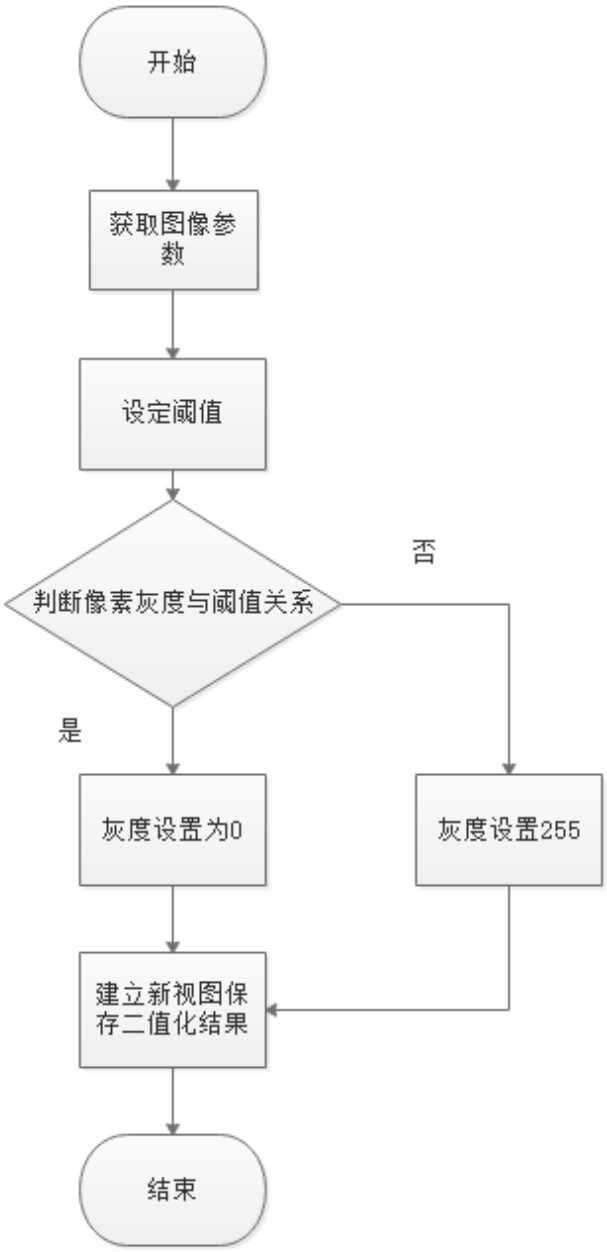
2. 设计的思路及主要内容

主要分三步完成。第一步是写出图形界面，图片能够很好地显示在程序上。第二步是从图片中找出条形码所在矩形区域,通过形态学的处理和 opencv 函数，可以很好地检测出边界。第三步则是确定准确边界译码实现，在这里采用整体阈值的方法获得二值化图像，译码则采用平均值法译码。

平均值译码法先测量出从起始符到终止符的像素数 M，这一段的像素数是（95=3+7*6+5+7*6+3)个单位长度,则可以计算出单位长度的像素数(unit=M/95)，根据单位长度的像素数，就可以计算出每个 bar 或者 space 的标准宽度，根据其标准宽度就可以查表译出相应的数字字符。



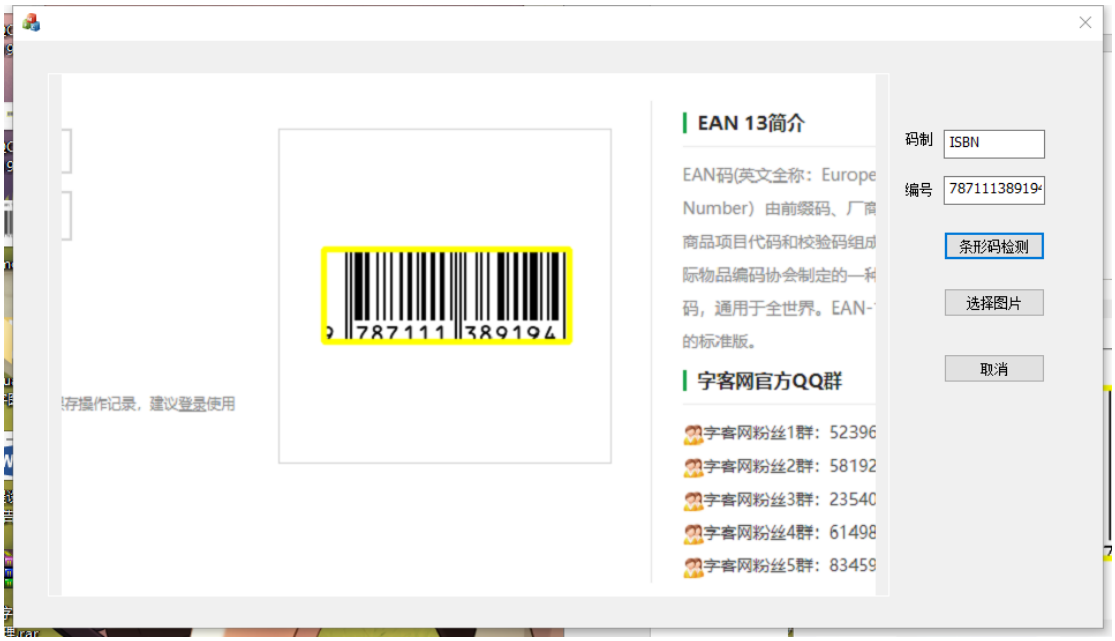
平均值译码法易于理解，编写简单，但是有一个致命的缺点，那就是对条形码图片的质量要求太高。效果差强人意。


开始
获取图像参数
设定阈值
判断像素灰度与阈值关系
否
是
灰度设置为0
灰度设置255
建立新视图保存二值化结果
结束

二值化过程

三、 最终结果

效果截图



效果不太好，对于手机拍出的图片不能识别成功，用条形码生成器生成的图片部分能识别成功。对于大多数图片能够框选处正确的条形码范围，但译码程序无法得到正确的译码结果。程序能够拉伸，控件能够自适应大小。将译码的结果（后 12 位）和码制送至编辑框显示。

四、 个人总结

第一步在图片控件显示图片时，用 opencv 的显示函数显示出图片再作为子窗口与图片控件相连接。然而这种方法只能填充整个图片控件大小，对于不同比例的图片由于长宽拉伸比例不同会使得图片看起来很怪。因此我选择将 Mat 类的图像转化为 MFC 图片控件支持的 CImage 类显示，使得图片能够合适的显示在程序中。

第二步选择合适的内核大小进行腐蚀膨胀的操作，找到图中最大的矩形部分选取出来。将处理过后的图送到程序上显示来判断是否得到正确的条形码区域，而选择出来的那部分作为返回值以便进行下一步的译码。

第三部译码采用平均值法译码，在调试时我发现单位模块的宽度只有两个像素甚至更少，并且因为在译码过程中如果图片质量不符合要求得不到正确的 top 和 bottom，程序会提示未初始化。

可改进部分：

形态学处理的方法对于不同背景的图片适用性并不是太好，有些图片会因为太多的腐蚀膨胀操作最终选择出错误的矩形部分。另外本程序未考虑倾斜条码和校正。倾斜的图片会加大平均值译码的误差。译码部分采用相似边距离译码法也能有效减少误差（ZXING 和 ZBar 使用此方法），因为时间原因我选择了更为简单的平均值译码。另外平均值译码过程中的 IF 语句的判断条件的参数可能修改后会有更好结果。整体阈值法对于直方统计有两个峰的图像效果比较好，如果用其他方法根据背景条件的不同动态选取二值化图像的方法也能改善边缘检测的成功率。

五、 心得体会

通过本次课程设计，我想对数字图像处理所学知识进行巩固和扩充，对分析问题、解决问题的能力以及动手操作能力进行提高。最主要的还是想对自己的 C++编程能力进行进一步的提升。通过做出一个能够识别一维条形码的有图形化界面的基于 MFC 的程序来提高面向对象的编程能力。本次课程设计中会使用

opencv 中的一些基础类和函数，通过学习 opencv 相关库函数和 zxing 等项目解码的实现以及自己的编程以达到更熟练地使用 C++编程的目的，同时也是对数据结构，算法等学过的内容的回顾和进一步深入学习。一开始是想做二维 QR 码的识别，但 QR 码标准较多，且支持汉字等特性使得定位和译码过程相对来说都困难得多。Opencv4.0 提供了 QR 码识别的新功能，但处于 beta 阶段支持的还不太好。在学习 Windows 图形编程的过程中也要学 Windows 原理和 MFC 一些控件和一些常用的类，增强 C++的理解。

六、 源代码（部分）

工程全部代码请见：https://github.com/BrokenGzh/Opencv-EAN13_barcode_reader

```
/条形码边缘检测函数，框选出条形码
Mat BarcodeScan(Mat &img)
{
    Mat greyimg;
    Mat resultImage;
    cvtColor(img,greyimg, CV_RGB2GRAY); //转化为灰度图
    img.convertTo(img, CV_8U);//深度变为 8 位，便于进行接下来运算
    Mat greyXimg, greyYimg;
    Sobel(greyimg, greyXimg, CV_32F, 1,0,-1);
    Sobel(greyimg, greyYimg, CV_32F, 0,1,-1); //xy 方向的梯度
    Mat imgsobel,blurImage, thresholdImage, morphImage,angleMat;
    cartToPolar(greyXimg, greyYimg, imgsobel, angleMat);
    convertScaleAbs(imgsobel, imgsobel); //再转化为八位图像
    blur(imgsobel, blurImage, Size(5, 5));
    threshold(blurImage, thresholdImage, 160, 255, THRESH_BINARY);
//二值化
    Mat kernel = getStructuringElement(MORPH_RECT, Size(21, 7));//
寻找内核
```

```
    morphologyEx(thresholdImage, morphImage, MORPH_CLOSE,
kernel);//闭运算
    erode(morphImage, morphImage, getStructuringElement(MORPH_RECT,
Size(3, 3)), Point(-1, -1), 4);
    dilate(morphImage, morphImage,
getStructuringElement(MORPH_RECT, Size(3, 3)), Point(-1, -1), 4);//
腐蚀过后再膨胀
    vector<vector<Point2i>>contours;
    vector<float>contourArea;//创建二维浮点向量储存坐标
    findContours(morphImage, contours, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE); //找到边缘四个点
    //计算轮廓的面积并且存放
    for (int i = 0; i < contours.size(); i++)
    {
        contourArea.push_back(cv::contourArea(contours[i]));
    }
    //找出面积最大的轮廓
    double maxValue; Point maxLoc;
    minMaxLoc(contourArea, NULL, &maxValue, NULL, &maxLoc);
    //计算面积最大的轮廓的最小的外包矩形
    RotatedRect minRect = minAreaRect(contours[maxLoc.x]);
    //为了防止找错,要检查这个矩形的偏斜角度不能超标
    //如果超标,那就是没找到
    if (minRect.angle < 2.0)
    {
        //找到了矩形的角度,但是这是一个旋转矩形,所以还要重新获得一个
外包最小矩形
        Rect myRect = boundingRect(contours[maxLoc.x]);
        //把这个矩形在源图像中画出来
```

```
        rectangle(img,myRect,Scalar(0,255,255),3,LINE_AA);
        //看看显示效果,找的对不对
        //将扫描的图像裁剪下来,并保存为相应的结果,保留一些 X 方向的边界,所以对 rect 进行一定的扩张
        myRect.x = myRect.x - (myRect.width / 20);
        myRect.width = myRect.width*1.1;
        myRect.height = myRect.height*1.1;
        resultImage = Mat(img, myRect);
    }
    return resultImage;
};

//判断码值的程序
int JudgNum(int w1, int w2, int w3, int w4, double mx)
{
    double a1, a2, a3;
    int ia1, ia2, ia3;
    a1 = (double)(w1 + w2) / mx;
    a2 = (double)(w2 + w3) / mx;
    a3 = (double)(w3 + w4) / mx;
    ia1 = (int)(a1 + 0.5);
    ia2 = (int)(a2 + 0.5);
    ia3 = (int)(a3 + 0.5);

    //判断该码值
    if ((ia1 == 5 && ia2 == 3 && ia3 == 2) || (ia1 == 2 && ia2 == 3 && ia3 == 5))
        return 0;
```

```
    if ((ia1 == 4 && ia2 == 4 && ia3 == 3) || (ia1 == 3 && ia2 == 4
&& ia3 == 4))
    {
        if (ia1 == 4)
        {
            double dw2 = (double)w2 / mx;
            if (dw2 < 2.4)
                return 1;
            else if (dw2 > 2.6)
                return 7;
            else return -1;
        }

        if (ia1 == 3)
        {
            double dw3 = (double)w3 / mx;
            if (dw3 < 2.4)
                return 1;
            else if (dw3 > 2.6)
                return 7;
            else return -1;
        }
    }

    if ((ia1 == 3 && ia2 == 3 && ia3 == 4) || (ia1 == 4 && ia2 == 3
&& ia3 == 3))
    {
        if (ia1 == 3)
        {
```

```
            double dw4 = (double)w4 / mx;
            if (dw4 < 2.4)
                return 2;
            else if (dw4 > 2.6)
                return 8;
            else return -1;
        }

        if (ia1 == 4)
        {
            double dw1 = (double)w1 / mx;
            if (dw1 < 2.4)
                return 2;
            else if (dw1 > 2.6)
                return 8;
            else return -1;
        }
    }

    if ((ia1 == 5 && ia2 == 5 && ia3 == 2) || (ia1 == 2 && ia2 == 5
&& ia3 == 5))
        return 3;

    if ((ia1 == 2 && ia2 == 4 && ia3 == 5) || (ia1 == 5 && ia2 == 4
&& ia3 == 2))
        return 4;

    if ((ia1 == 3 && ia2 == 5 && ia3 == 4) || (ia1 == 4 && ia2 == 5
&& ia3 == 3))
```

```
        return 5;

    if ((ia1 == 2 && ia2 == 2 && ia3 == 5) || (ia1 == 5 && ia2 == 2
&& ia3 == 2))
        return 6;

    if ((ia1 == 4 && ia2 == 2 && ia3 == 3) || (ia1 == 3 && ia2 == 2
&& ia3 == 4))
        return 9;

    return false;
};

//译码主体程序
bool PreProcess(Mat &img)
{
    img.convertTo(img, CV_8U);//深度变为8位，便于进行接下来运算
    cvtColor(img, img, CV_RGB2GRAY); //转化为灰度图
    //二值化
    threshold(img, img, 160, 255, THRESH_BINARY);
    int i, j;
    int tempMax;
    int tempArray[1000];
    long ImageHeight = img.rows;
    long ImageWidth = img.cols;
    int arWidth[200];           //保存宽度序列：从最左边的黑条开始
    int arDifference[1000];         //差分
    bool arMark[1000];              //标记
    int ImageLeft;                  //图像最左边的座标
```

```
int ImageRight;                    //图像最右边的座标
int ImageTop;                  //条形码顶部的座标
int ImageBottom;               //条形码下部的座标
int arPixelH[1000];                //为了水平方向直方图统计用
int arPixelV[1000];                //为了垂直方向直方图统计用
int arDelta[1000];
int arLeftEdge1[1000];
int arLeftEdge2[1000];
int arLeftEdge[1000];
//进行水平方向和垂直方向上的直方图统计
for (i = 0; i < ImageHeight; i++)
{
    for (j = 0; j < ImageWidth; j++)
    {
        if (img.at<uchar>(i, j) >= 180)
            ImageArray[i][j] = (BYTE)0;
        else
            ImageArray[i][j] = (BYTE)1;
    }
}
for (i = 0; i < ImageHeight; i++)
    arPixelV[i] = 0;
for (i = 0; i < ImageWidth; i++)
    arPixelH[i] = 0;
for (i = 0; i < ImageHeight; i++)
{
    for (j = 0; j < ImageWidth; j++)
    {
        if (ImageArray[i][j] == 1)
```

```
            {
                arPixelV[i] += 1;
                arPixelH[j] += 1;
            }
        }
    }

    //寻找包含条形码的区域，
    //线寻找水平方向上黑象素最大的行

    tempMax = 0;
    for (i = 0; i < ImageHeight; i++)
    {
        if (arPixelV[i] > tempMax)
            tempMax = arPixelV[i];
        arMark[i] = false;
    }

    for (i = 0; i < ImageHeight - 1; i++)
    {
        //计算差分
        arDifference[i] = arPixelV[i + 1] - arPixelV[i];

        //如果该行像素足够多且变化不大，标记为 true
        if ((abs(arDifference[i]) < 20) && (arPixelV[i] >
(0.75*tempMax)))
            arMark[i] = true;
    }
```

```
//确定包含条码的行
int iLengthThrehold = 40;
int iCount;
for (i = 0; i < ImageHeight - iLengthThrehold; i++)
{
    iCount = 0;
    for (j = 0; j < iLengthThrehold; j++)
    {
        if (arMark[i + j] == true)
            iCount++;
    }
    if (iCount >= 37)
    {
        ImageTop = i + 10;       //确定顶部
        break;
    }
}

for (i = ImageHeight - 1; i >= iLengthThrehold - 1; i--)
{
    iCount = 0;
    for (j = 0; j < iLengthThrehold; j++)
    {
        if (arMark[i - j] == true)
            iCount++;
    }
    if (iCount >= 37) //iLengthThrehold-3
    {
        ImageBottom = i - 10;       //确定底部
```

```
            break;
        }
    }

    //寻找左边缘，在已经确定的上下边界内全局搜索
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        for (j = 2; j < ImageWidth; j++)
        {
            if ((ImageArray[i][j - 1] == 0) && (ImageArray[i][j] ==
1))
            {
                arLeftEdge[i] = j;
                break;
            }
        }
    }

    //为消除噪声的干扰，下面确定准确的左边界
    tempMax = 0;
    int iMax = 0;
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        if (arLeftEdge[i] > tempMax)
        {
            tempMax = arLeftEdge[i];
            iMax = i;
        }
    }
```

```
    //倾斜度不能大于 1/10
    iCount = 0;
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        if (abs(tempMax - arLeftEdge[i]) < abs(i - iMax) / 6 + 1)
        {
            iCount++;
        }
    }

    if ((iCount / (ImageBottom - ImageTop)) < 0.6)
        return false;

    //调整起点
    for (i = iMax; i > ImageTop; i--)
    {
        if (abs(arLeftEdge[i] - arLeftEdge[i - 1]) >= 2)
        {
            if (ImageArray[i - 1][arLeftEdge[i]] - ImageArray[i -
1][arLeftEdge[i] - 1] == 1)
                arLeftEdge[i - 1] = arLeftEdge[i];
            else if (ImageArray[i - 1][arLeftEdge[i] - 1] -
ImageArray[i - 1][arLeftEdge[i] - 2] == 1)
                arLeftEdge[i - 1] = arLeftEdge[i] - 1;
            else if (ImageArray[i - 1][arLeftEdge[i] + 1] -
ImageArray[i - 1][arLeftEdge[i]] == 1)
                arLeftEdge[i - 1] = arLeftEdge[i] + 1;
            else
```

```
            arLeftEdge[i - 1] = arLeftEdge[i];
        }
    }

    for (i = iMax; i < ImageBottom; i++)
    {
        if (i == ImageBottom)
            break;

        if (abs(arLeftEdge[i] - arLeftEdge[i + 1]) >= 2)
        {
            if (ImageArray[i + 1][arLeftEdge[i]] - ImageArray[i +
1][arLeftEdge[i] - 1] == 1)
                arLeftEdge[i + 1] = arLeftEdge[i];
            else if (ImageArray[i + 1][arLeftEdge[i] - 1] -
ImageArray[i + 1][arLeftEdge[i] - 2] == 1)
                arLeftEdge[i + 1] = arLeftEdge[i] - 1;
            else if (ImageArray[i + 1][arLeftEdge[i] + 1] -
ImageArray[i + 1][arLeftEdge[i]] == 1)
                arLeftEdge[i + 1] = arLeftEdge[i] + 1;
            else
                arLeftEdge[i + 1] = arLeftEdge[i];
        }
    }

    int n;
    //搜索出所有的宽度
    for (n = 0; n < 29; n++)
    {
```

```
        //搜索条的右边缘
        for (i = ImageTop; i <= ImageBottom; i++)
        {
            for (j = arLeftEdge[i] + 1; j < ImageWidth; j++)
            {
                if ((ImageArray[i][j - 1] == 1) && (ImageArray[i][j]
== 0))
                {
                    arLeftEdge1[i] = j;
                    break;
                }
            }
            arDelta[i] = arLeftEdge1[i] - arLeftEdge[i];
        }

        //假定条和空的宽度最多为 11
        //排序，可以认为最中间的 5 个宽度是平均宽度
        for (i = ImageTop; i < ImageBottom; i++)
            tempArray[i] = arDelta[i];

        for (i = ImageTop; i < ImageBottom; i++)
        {
            for (j = ImageBottom; j > i; j--)
            {
                int tempSwap;
                if (tempArray[j] < tempArray[j - 1])
                {
                    tempSwap = tempArray[j];
                    tempArray[j] = tempArray[j - 1];
```

```
                tempArray[j - 1] = tempSwap;
            }
        }
    }

    if (tempArray[ImageTop + (ImageBottom - ImageTop) / 2 + 2] -
tempArray[ImageTop + (ImageBottom - ImageTop) / 2 - 2] > 1)
        return false;
    else
        arWidth[2 * n] = tempArray[ImageTop + (ImageBottom -
ImageTop) / 2];

    //调整下一列边缘
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        if (abs(arDelta[i] - arWidth[2 * n]) > 2)
            arLeftEdge1[i] = arLeftEdge[i] + arWidth[2 * n];
        arLeftEdge[i] = arLeftEdge1[i];
    }

    //搜索空的右边缘
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        for (j = arLeftEdge[i] + 1; j < ImageWidth; j++)
        {
            if ((ImageArray[i][j - 1] == 0) && (ImageArray[i][j]
== 1))
            {
                arLeftEdge1[i] = j;
```

```
                break;
            }
        }
        arDelta[i] = arLeftEdge1[i] - arLeftEdge[i];
    }

    //假定条和空的宽度最多为 11
    //排序，可以认为最中间的 5 个宽度是平均宽度
    for (i = ImageTop; i < ImageBottom; i++)
        tempArray[i] = arDelta[i];

    for (i = ImageTop; i < ImageBottom; i++)
    {
        for (j = ImageBottom; j > i; j--)
        {
            int tempSwap;
            if (tempArray[j] < tempArray[j - 1])
            {
                tempSwap = tempArray[j];
                tempArray[j] = tempArray[j - 1];
                tempArray[j - 1] = tempSwap;
            }
        }
    }

    if (tempArray[ImageTop + (ImageBottom - ImageTop) / 2 + 2] -
tempArray[ImageTop + (ImageBottom - ImageTop) / 2 - 2] > 1)
        return false;
    else
```

```
            arWidth[2 * n + 1] = tempArray[ImageTop + (ImageBottom -
ImageTop) / 2];

        //调整下一列边缘
        for (i = ImageTop; i <= ImageBottom; i++)
        {
            if (abs(arDelta[i] - arWidth[2 * n + 1]) > 2)
                arLeftEdge1[i] = arLeftEdge[i] + arWidth[2 * n + 1];
            arLeftEdge[i] = arLeftEdge1[i];
        }
    }

    //搜索最后一个条的右边缘
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        for (j = arLeftEdge[i] + 1; j < ImageWidth; j++)
        {
            if ((ImageArray[i][j - 1] == 1) && (ImageArray[i][j] ==
0))
            {
                arLeftEdge1[i] = j;
                break;
            }
        }
        arDelta[i] = arLeftEdge1[i] - arLeftEdge[i];
    }

    //假定条和空的宽度最多为 11
    //排序，可以认为最中间的 5 个宽度是平均宽度
```

```
    for (i = ImageTop; i < ImageBottom; i++)
        tempArray[i] = arDelta[i];

    for (i = ImageTop; i < ImageBottom; i++)
    {
        for (j = ImageBottom; j > i; j--)
        {
            int tempSwap;
            if (tempArray[j] < tempArray[j - 1])
            {
                tempSwap = tempArray[j];
                tempArray[j] = tempArray[j - 1];
                tempArray[j - 1] = tempSwap;
            }
        }
    }

    if (tempArray[ImageTop + (ImageBottom - ImageTop) / 2 + 2] -
tempArray[ImageTop + (ImageBottom - ImageTop) / 2 - 2] > 1)
        return false;
    else
        arWidth[2 * n] = tempArray[ImageTop + (ImageBottom -
ImageTop) / 2];

    //调整下一列边缘
    for (i = ImageTop; i <= ImageBottom; i++)
    {
        if (abs(arDelta[i] - arWidth[2 * n + 1]) > 2)
```

```
            arLeftEdge1[i] = arLeftEdge[i] + tempArray[ImageTop +
(ImageBottom - ImageTop) / 2];
        arLeftEdge[i] = arLeftEdge1[i];
    }
    //总共有 7×12＋3×2＋5＝ 95 个单位宽度
    //有 4×12＋3×2＋5＝59 个宽度，
    int result[12];
    double mx = 0.0; //平均宽度

    for (i = 0; i < 59; i++)
        mx += (double)arWidth[i];
    mx /= 95.0;

    //起始条文
    for (i = 0; i < 3; i++)
    {
        double dTemp = (double)arWidth[i] / mx;
        if (dTemp<0.6 || dTemp>1.4)
            break;
    }
    //起始码不符合要求
    //if(i<3)
    //  return false;

    //识别前 6 个
    for (i = 0; i < 6; i++)
    {
        result[i] = JudgNum(arWidth[i * 4 + 3], arWidth[i * 4 + 4],
arWidth[i * 4 + 5], arWidth[i * 4 + 6], mx);
```

```
    }
    //识别后 6 个
    for (i = 6; i < 12; i++)
    {
        result[i] = JudgNum(arWidth[i * 4 + 8], arWidth[i * 4 + 9],
arWidth[i * 4 + 10], arWidth[i * 4 + 11], mx);
    }

    //判断码制
    if (result[0] == 7 && result[1] == 7)
    {
        strCodeStyle = "ISSN";
    }
    else if (result[0] == 7 && result[1] == 8)
    {
        strCodeStyle = "ISBN";
    }
    else
        strCodeStyle = "Unknown!";

    //判断是否全部识别出来
    for (i = 0; i < 12; i++)
        if (result[i] == -1)
            return false;
    for (i = 0; i < 12; i++)
        if (result[i] == -1)
            return false;
    CString strTemp;
    for (i = 0; i < 12; i++)
```

```
    {
        strTemp.Format(_T("%d"), result[i]);
        strCodeNumber += strTemp;
    }
    return true;
};
```

七、 参考文献

1.《Visual C++ 数字图像模式识别典型案例详解》

2. OPENCV 条形码检测与识别 https://www.cnblogs.com/dengxiaojun/p/5278679.html

3.opecv 官方手册