2018

条形码识别程序

程序说明文档 V1.0

孙浩翔

目录

1. 介绍

条形码识别程序，利用 OpenCV 和 ZBar 插件实现对一张包含多张条形码的图片框选的功能。

2. 使用软件

2.1 Microsoft Visual Studio 2013

https://www.visualstudio.com/

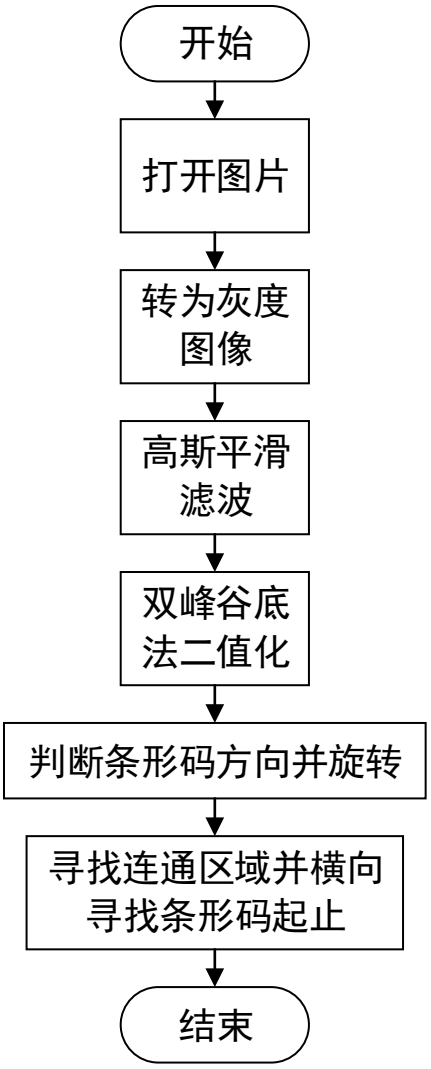
2.2 OpenCV 2.13.4

https://opencv.org/

2.3 ZBar 0.10

http://zbar.sourceforge.net/

3. 软件流程


开始
打开图片
转为灰度图像
高斯平滑滤波
双峰谷底法二值化
判断条形码方向并旋转
寻找连通区域并横向寻找条形码起止
结束

4. 效果

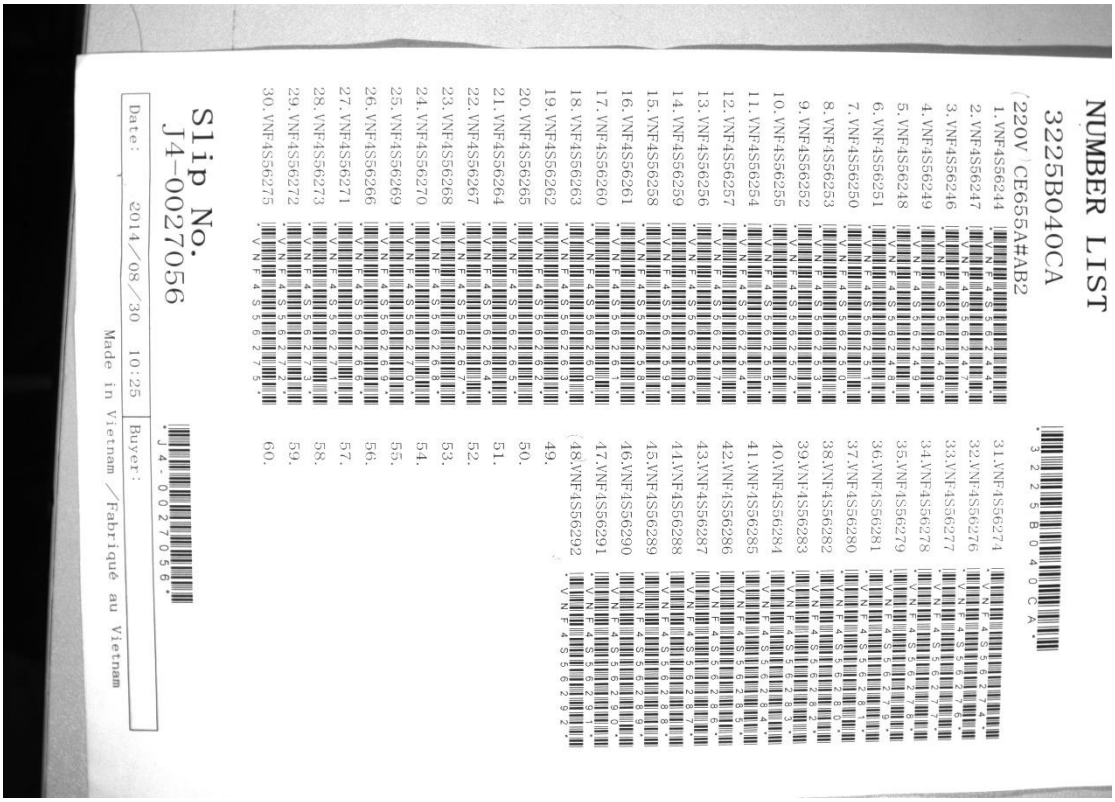

NUMBER LIST
3225B040CA
(220V) CE655A#AB2
Slip No.
J4-0027056
Date: 2014/08/30 10:25
Buyer:
Made in Vietnam /Fabriqué au Vietnam

图 4-1 原图像

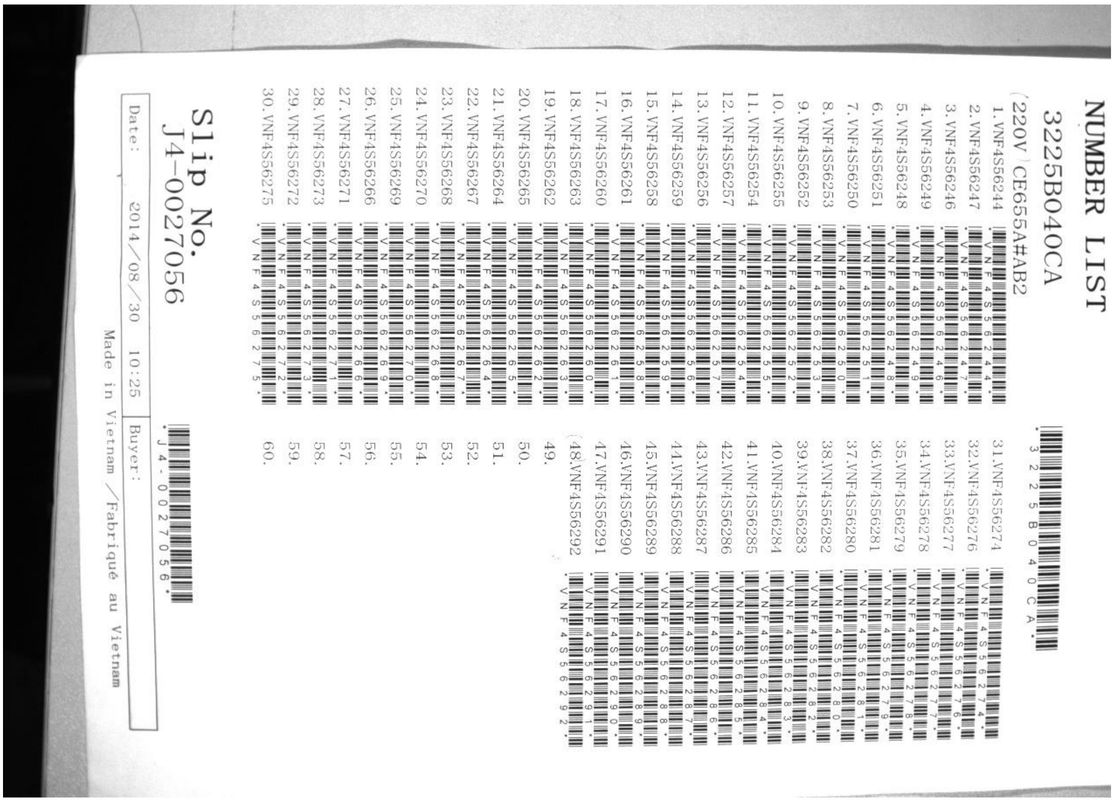


图 4-2 灰度图

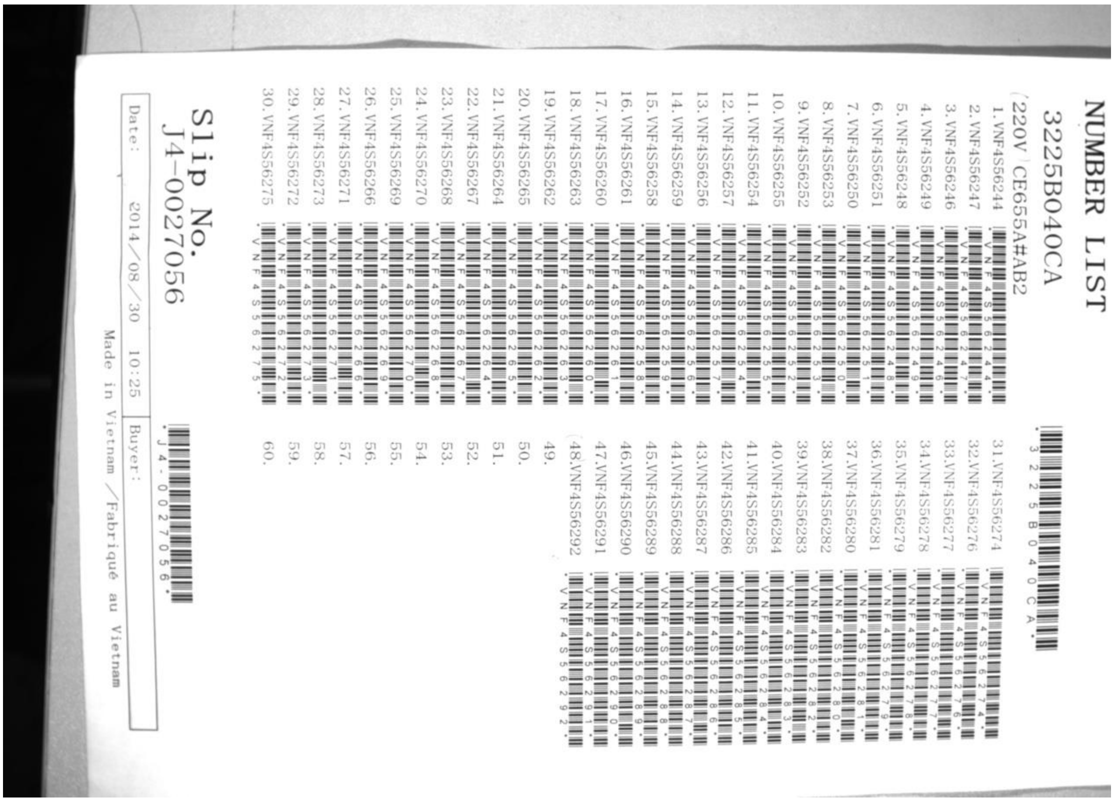


图 4-3 高斯平滑滤波后图像

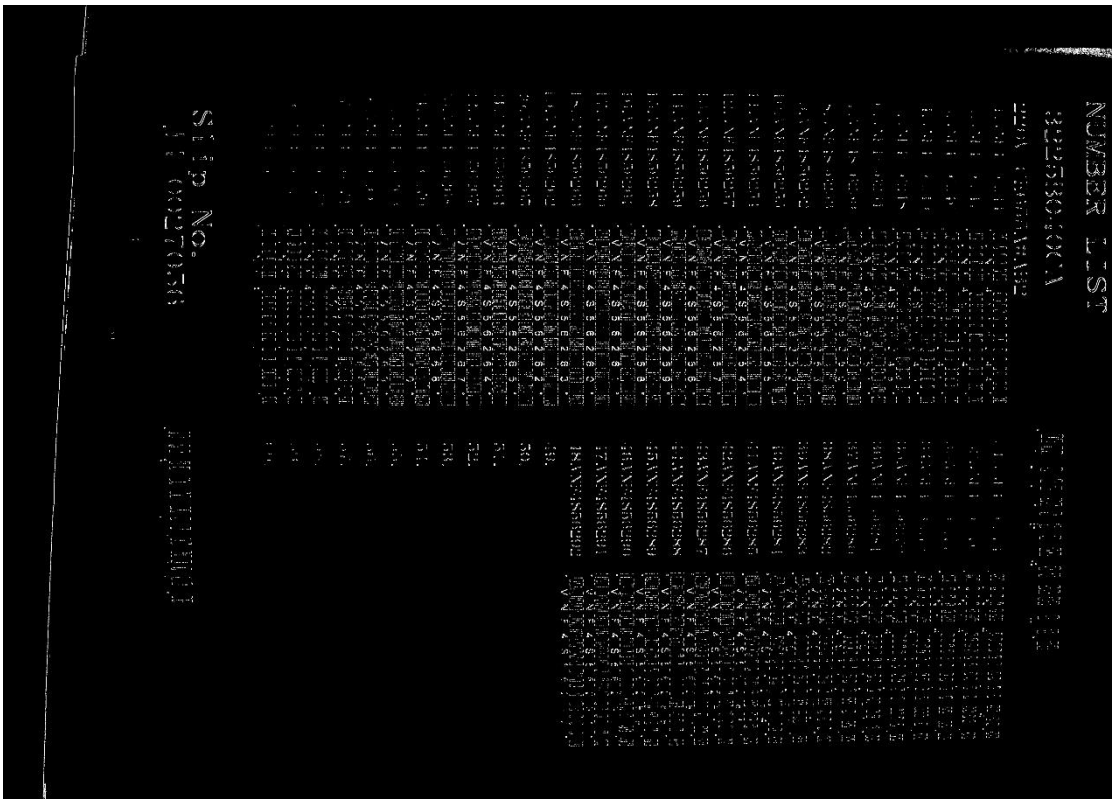


图 4-4 X 方向梯度图像

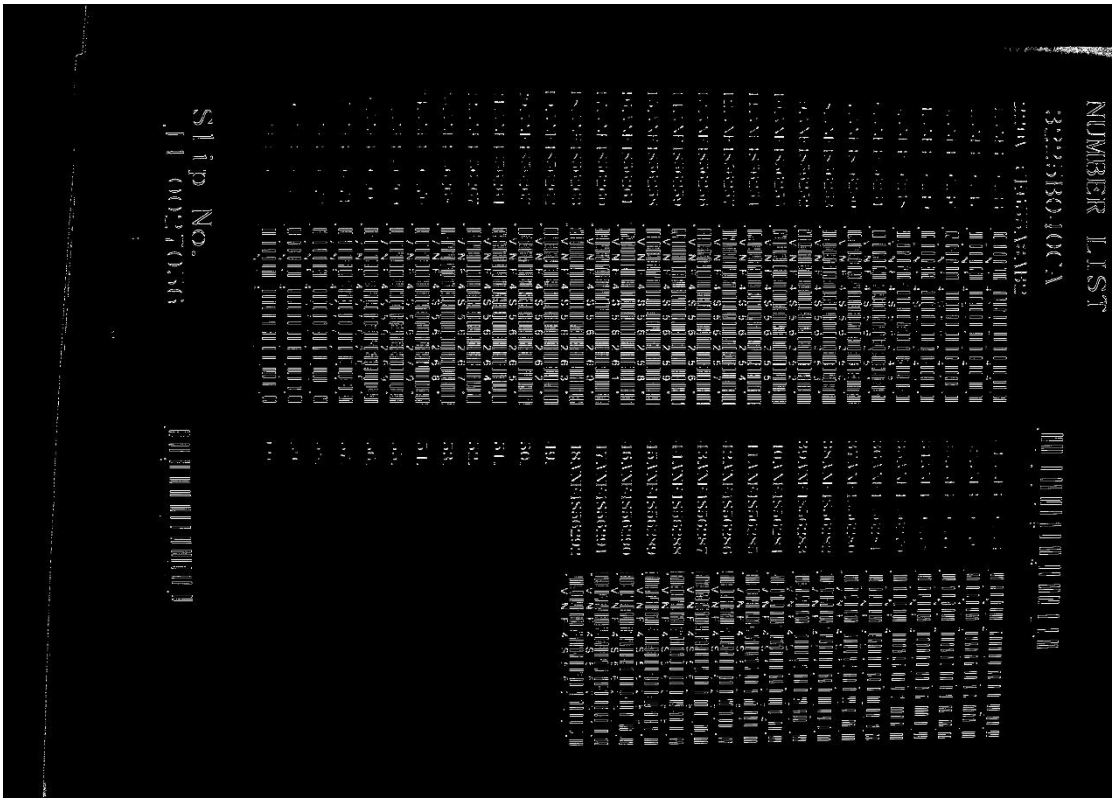


图 4-5 Y 方向梯度图

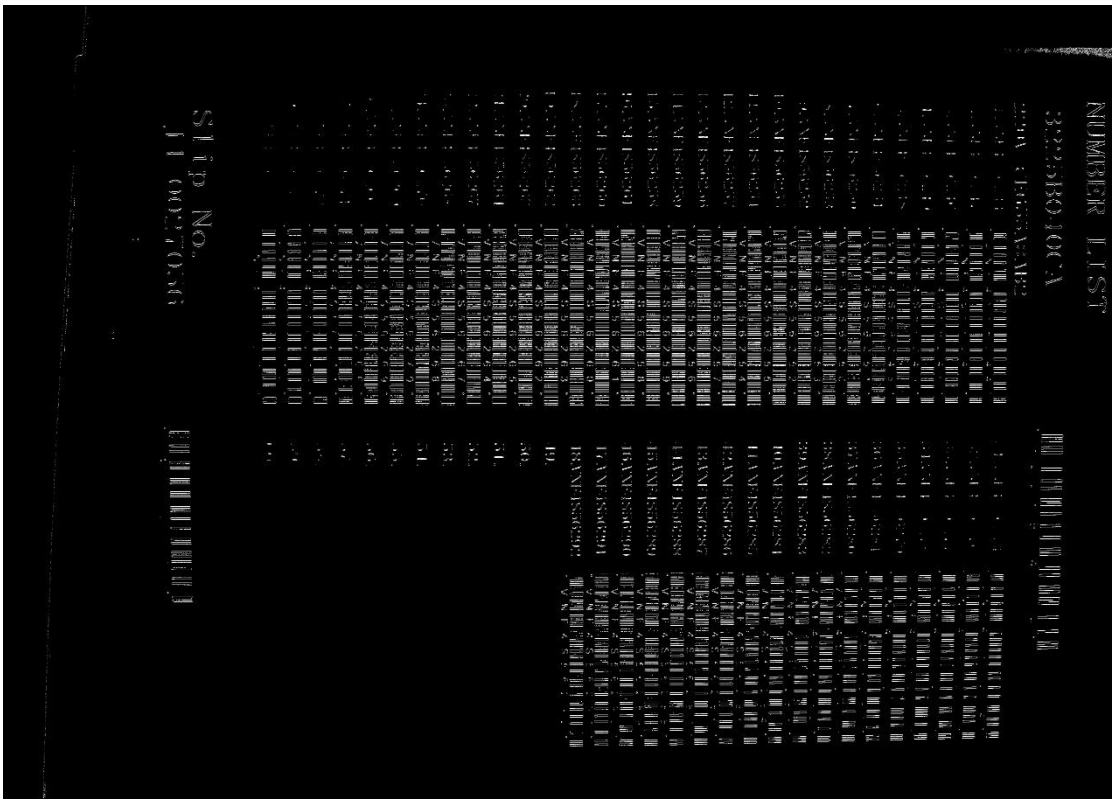


图 4-6 梯度方向图像

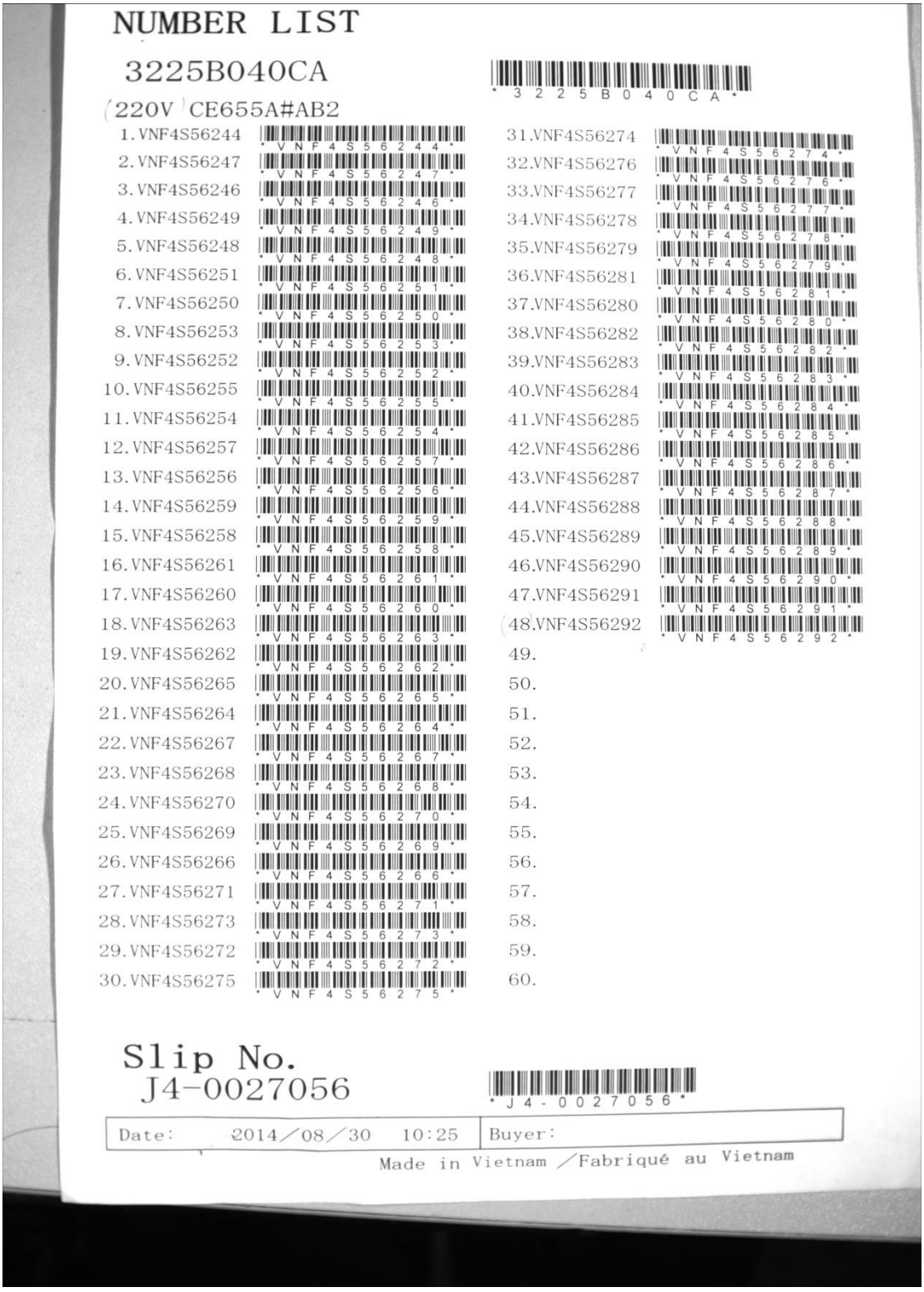

NUMBER LIST
3225B040CA
(220V) CE655A#AB2
1.VNF4S56244
2.VNF4S56247
3.VNF4S56246
4.VNF4S56249
5.VNF4S56248
6.VNF4S56251
7.VNF4S56250
8.VNF4S56253
9.VNF4S56252
10.VNF4S56255
11.VNF4S56254
12.VNF4S56257
13.VNF4S56256
14.VNF4S56259
15.VNF4S56258
16.VNF4S56261
17.VNF4S56260
18.VNF4S56263
19.VNF4S56262
20.VNF4S56265
21.VNF4S56264
22.VNF4S56267
23.VNF4S56268
24.VNF4S56270
25.VNF4S56269
26.VNF4S56266
27.VNF4S56271
28.VNF4S56273
29.VNF4S56272
30.VNF4S56275
31.VNF4S56274
32.VNF4S56276
33.VNF4S56277
34.VNF4S56278
35.VNF4S56279
36.VNF4S56281
37.VNF4S56280
38.VNF4S56282
39.VNF4S56283
40.VNF4S56284
41.VNF4S56285
42.VNF4S56286
43.VNF4S56287
44.VNF4S56288
45.VNF4S56289
46.VNF4S56290
47.VNF4S56291
48.VNF4S56292
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
Slip No.
J4-0027056
Date: 2014/08/30 10:25
Buyer:
Made in Vietnam /Fabriqué au Vietnam

图 4-7 旋转图像

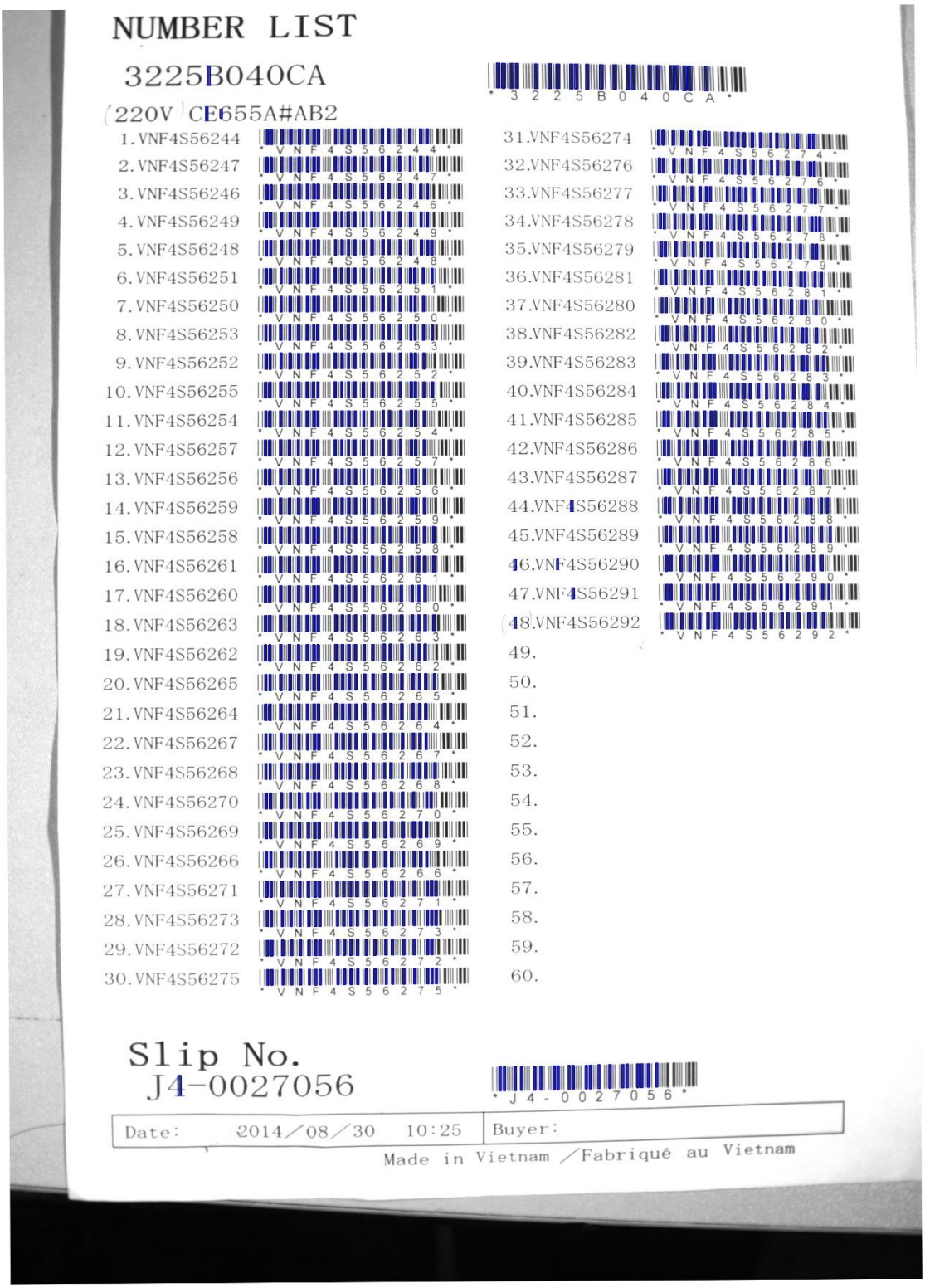


图 4-8 找到的所有连通区域示意图

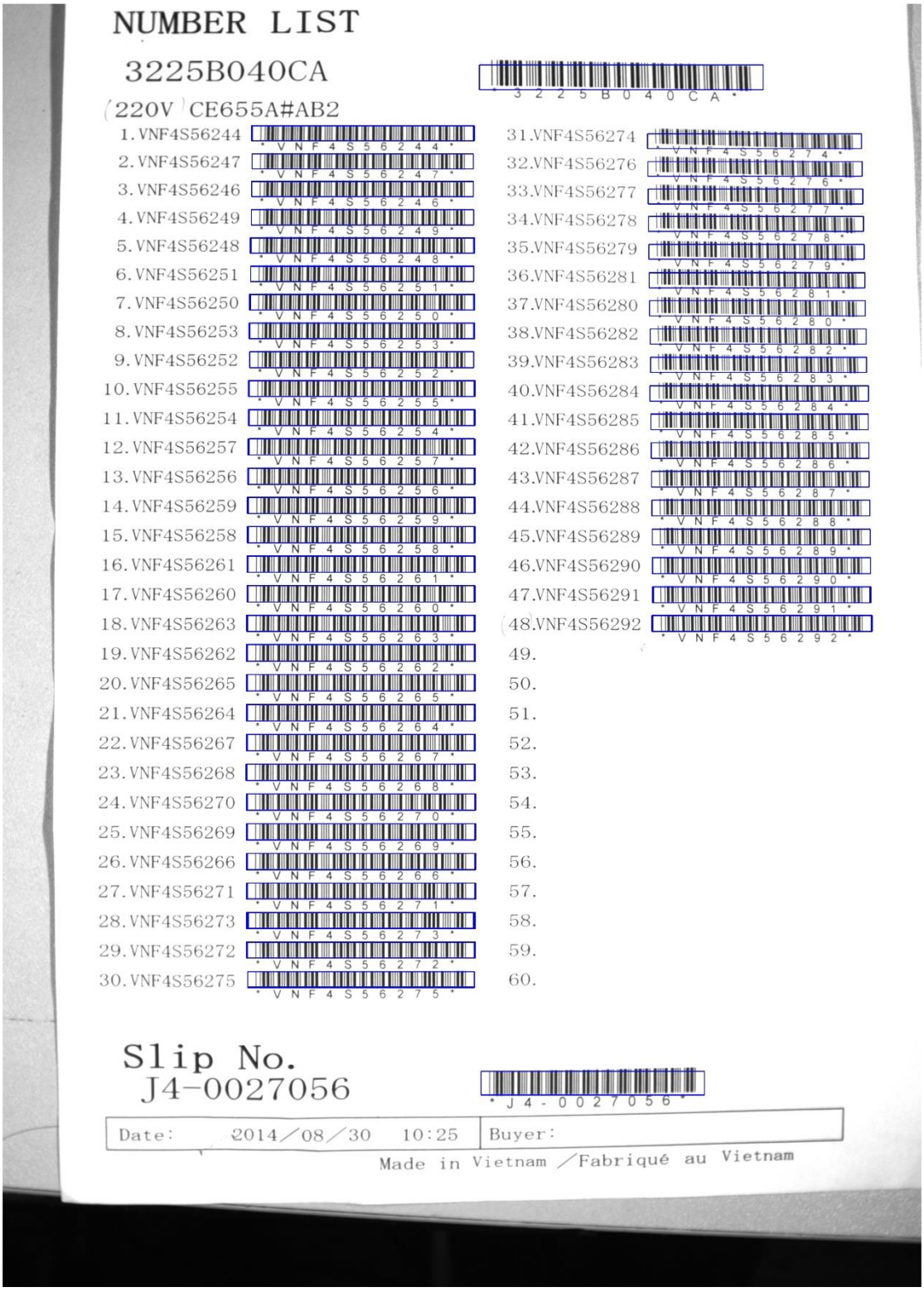


图 4-9 条形码区域查找结果

5. 文件

5.1 main.cpp

```
#include "core/core.hpp"
#include "highgui/highgui.hpp"
#include "imgproc/imgproc.hpp"
#include <math.h>
#include "source.h"
#include "zbar.h"

#define PI 3.1415926

using namespace std;
using namespace cv;
using namespace zbar;

int main(int argc, char *argv[])
{
    Mat image, imageGray, imageGuussian, imageEqualize;

    /*----------------------------------------
    1. 打开图像
    ----------------------------------------*/
    image = imread(argv[1]);
    myImShow("1.原图像", image,ZIP,1);

    /*----------------------------------------
    2.转化为灰度图
    ----------------------------------------*/
    cvtColor(image, imageGray, CV_RGB2GRAY);
    //为了方便，对图片压缩、保存进行了封装
    myImShow("2.灰度图", imageGray,ZIP,1);

    /*----------------------------------------
    3. 高斯平滑滤波
    ----------------------------------------*/
    GaussianBlur(imageGray, imageGuussian, Size(9, 9), 0);
    myImShow("3.高斯平衡滤波", imageGuussian,ZIP,1);

    /*----------------------------------------
```

```
    4. 双峰谷底法寻找阈值二值化
    ----------------------------------------*/
    //计算直方图
    MatND histG = myCalcHist(imageGuussian, 0);
    //双峰谷底法寻找阈值
    int vally = findThresholdVally(histG);
    printf("二值化阈值：%d\r\n", vally);
    //二值化
    Mat imageThreshold;
    threshold(imageGray, imageThreshold, vally, 255, CV_THRESH_BINARY);
    myImShow("二值化", imageThreshold,ZIP,1);

    /*----------------------------------------
    5. 判断条形码方向并旋转
        使用Sobel算子分别计算X、Y方向梯度
        根据两个方向上梯度余弦的计算，统计出变化最频繁方向
        根据条形码特征，条形码水平因为黑白交叉，梯度变换频繁
        根据统计信息旋转图像
    ----------------------------------------*/
    //求得水平和垂直方向灰度图像的梯度,使用Sobel算子
    Mat imageX16S, imageY16S;
    Mat imageSobelX, imageSobelY;
    Mat imageDirection;
    Sobel(imageThreshold, imageX16S, CV_16S, 1, 0, 3, 1, 0, 4);
    Sobel(imageThreshold, imageY16S, CV_16S, 0, 1, 3, 1, 0, 4);

    //计算每个像素点梯度方向，统计峰值（除0外）
    findDirection(imageX16S, imageY16S, imageDirection);
    int max = hist16S(imageDirection);
    printf("最大值位置：%d\r\n", max);

    //梯度图像显示
    //sobel计算后每个像素是short类型，需要转换为无符号数
    convertScaleAbs(imageX16S, imageSobelX, 1, 0);
    convertScaleAbs(imageY16S, imageSobelY, 1, 0);
    convertScaleAbs(imageDirection, imageDirection, 1, 0);
    myImShow("X方向", imageSobelX,ZIP,1);
    myImShow("Y方向", imageSobelY,ZIP,1);
    myImShow("5. 方向", imageDirection,ZIP,1);

    //旋转图像
    double angle = max / 255.0 * 90;
    //计算旋转后的大小，扩充旋转
    cv::Point2f center(image.cols / 2.0f, image.rows / 2.0f);
```

```
cv::Mat rot = cv::getRotationMatrix2D(center, angle, 1);
cv::Rect bbox = cv::RotatedRect(center, image.size(), angle).boundingRect();
//中心旋转
rot.at<double>(0, 2) += bbox.width / 2.0 - center.x;
rot.at<double>(1, 2) += bbox.height / 2.0 - center.y;

Mat imageRotate;
Mat imageGrayRotate;
Mat imageThresholdRotate;
Scalar borderColor = Scalar(255,255,255);
//因为默认旋转后填充黑色，做颜色反转
imageThreshold = 255 - imageThreshold;
warpAffine(image, imageRotate, rot, bbox.size(), INTER_LINEAR, BORDER_CONSTANT,borderColor);
warpAffine(imageThreshold, imageThresholdRotate, rot, bbox.size());
warpAffine(imageGray, imageGrayRotate, rot, bbox.size());
myImShow("5.旋转图像", imageGrayRotate, ZIP, 1);

/*----------------------------------------
6. 找连通区域
    contours记录所有找到的区域
    rectVector中记录条形码区域
    判断contours中的元素是否在rectVector中
    一个条形码的多个部分不重复查找
----------------------------------------*/
//查找连通区域
vector<vector<Point>> contours;
vector<Vec4i> hiera;
findContours(imageThresholdRotate, contours, hiera, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
printf("counters:%d\r\n", contours.size());
int imageWidth = imageRotate.cols;
int j = 0;
Vector<Rect> rectVector;
//对连通区域遍历
for (int i = 0;i<contours.size();i++)
{
    //简单过滤
    //区域宽度不超过图像10%，区域高是宽的3倍以上即长方形，宽大于4个像素
    Rect rect = boundingRect((Mat)contours[i]);
    if (rect.width < imageWidth / 10)
    {
        if (rect.width * 3 < rect.height && rect.width > 4)
        {
            //判断是否在已经找到的条形码内
            int xCurent = rect.tl().x;
```

```
                int yCenter = rect.tl().y + rect.height;
                int rectI;
                for (rectI = 0; rectI < rectVector.size();rectI++)
                {
                    Rect rectT = rectVector[rectI];
                    if ((xCurent > rectT.tl().x) && (xCurent < rectT.tl().x + rectT.width) &&
(yCenter > rectT.tl().y) && (yCenter < rectT.tl().y + rectT.height))
                    {
                        break;
                    }
                }

                if (rectI == rectVector.size())
                {
                    Rect rectTem;

                    //横向过滤
                    if (findBloak(imageGrayRotate, rect, rectTem))
                    {
                        //条形码识别
                        ImageScanner scanner;
                        scanner.set_config(ZBAR_NONE, ZBAR_CFG_ENABLE, 1);
                        Mat imageCut = Mat(imageGrayRotate, rectTem);
                        Mat imageCopy = imageCut.clone();
                        uchar *raw = (uchar *)imageCopy.data;
                        Image imageZbar(imageCopy.cols, imageCopy.rows, "Y800", raw,
imageCopy.cols * imageCopy.rows);
                        scanner.scan(imageZbar);        //扫描条形码
                        Image::SymbolIterator sybmol = imageZbar.symbol_begin();
                        if (imageZbar.symbol_begin() == imageZbar.symbol_end())
                        {
                            continue;
                        }
                        //如果区域是可识别的条形码
                        rectVector.push_back(rectTem);
                        printf("height:%d;width:%d\r\n", rect.height, rect.width);
                        printf("x:%d,y:%d\r\n", xCurent, yCenter);
                        rectangle(imageRotate, rectTem, Scalar(255), 2);
                    }
                }
            }
        }
    }
    namedWindow("6. 找出二维码矩形区域",0);
```

```
    myImShow("6. 找出二维码矩形区域", imageRotate,ZIP,1);

    waitKey();
    return 0;
}
```

5.2 source.h

```
#ifndef _SOURCE_H
#define _SOURCE_H

#include "core/core.hpp"
#include "highgui/highgui.hpp"
#include "imgproc/imgproc.hpp"
#include <math.h>

#define PI 3.1415926

using namespace std;
using namespace cv;

#define ZIP 1
#define ZIPTIME 0.17


int findThresholdVally(MatND hist);
MatND myCalcHist(Mat imageGray, int isShow);
int findDirection(Mat &inputImageX, Mat &inputImageY, Mat &outputImage);
int eraseBackground(Mat &inputImage, Mat &outputImage, int threshold);
void myImShow(char *imageName, Mat &image, int isZip, int isSave);
bool findBloak(Mat &image, Rect &rect, Rect & rectOut);
int hist16S(Mat &image);


#endif // !_SOURCE_H
```

5.3 source.cpp

```
#include "source.h"
```

```
//平均值法找二值化阈值
//参数： hist：直方图计算结果
//返回值：    灰度均值
int findThresholdAverage(MatND hist)
{
    double histMaxValue;
    Point histMaxLoc;
    minMaxLoc(hist, 0, &histMaxValue, 0, &histMaxLoc);

    double avr = 0;
    double sum = 0;
    for (int i = 0;i < 255;i++)
    {
        sum += hist.at<float>(i);
        avr += (double)hist.at<float>(i) * i;
    }

    return (int)(avr / sum);
}

//检测直方图是否为双峰的
//参数： HistGram[] 直方图数组
//返回值：    是否为双峰
bool IsDimodal(double HistGram[])
{
    // 对直方图的峰进行计数，只有峰数位2才为双峰
    int Count = 0;
    for (int Y = 1; Y < 255; Y++)
    {
        if (HistGram[Y - 1] < HistGram[Y] && HistGram[Y + 1] < HistGram[Y])
        {
            Count++;
            if (Count > 2) return false;
        }
    }
    if (Count == 2)
        return true;
    else
        return false;
}

//谷底最小值二值化阈值
//参数： hist 直方图
//返回值：    谷底灰度值
```

```
int findThresholdVally(MatND hist)
{
    int Y, Iter = 0;
    double HistGramC[256];          // 基于精度问题，一定要用浮点数来处理，否则得不到正确的结果
    double HistGramCC[256];         // 求均值的过程会破坏前面的数据，因此需要两份数据
    for (Y = 0; Y < 256; Y++)
    {
        HistGramC[Y] = hist.at<float>(Y);
        HistGramCC[Y] = hist.at<float>(Y);
    }

    // 通过三点求均值来平滑直方图
    while (IsDimodal(HistGramCC) == false)                                  // 判断是否已经是
双峰的图像了
    {
        HistGramCC[0] = (HistGramC[0] + HistGramC[0] + HistGramC[1]) / 3;                 // 第一点
        for (Y = 1; Y < 255; Y++)
            HistGramCC[Y] = (HistGramC[Y - 1] + HistGramC[Y] + HistGramC[Y + 1]) / 3;     // 中间的
点
        HistGramCC[255] = (HistGramC[254] + HistGramC[255] + HistGramC[255]) / 3;         // 最后一
点
        memcpy(HistGramC, HistGramCC, sizeof(HistGramCC));
        Iter++;
        if (Iter >= 1000)
            return -1;                                                  // 直方图无法平滑为双峰
的，返回错误代码
    }
    // 阈值极为两峰之间的最小值
    bool Peakfound = false;
    for (Y = 1; Y < 255; Y++)
    {
        if (HistGramCC[Y - 1] < HistGramCC[Y] && HistGramCC[Y + 1] < HistGramCC[Y]) Peakfound =
true;
        if (Peakfound == true && HistGramCC[Y - 1] >= HistGramCC[Y] && HistGramCC[Y + 1] >=
HistGramCC[Y])
            return Y - 1;
    }
    return -1;
}




//计算直方图
```

```
//参数:  imageGray 灰度图像
//参数:  isShow  -0 不绘制
//                    -1 绘制
//返回值:    灰度直方图数组
MatND myCalcHist(Mat imageGray, int isShow)
{
    //计算直方图
    int channels = 0;
    MatND dstHist;
    int histSize[] = { 256 };
    float midRanges[] = { 0,256 };
    const float *ranges[] = { midRanges };
    calcHist(&imageGray, 1, &channels, Mat(), dstHist, 1, histSize, ranges, true, false);

    if (isShow)
    {
        //绘制直方图,首先先创建一个黑底的图像，为了可以显示彩色，所以该绘制图像是一个8位的3通道图像
        Mat drawImage = Mat::zeros(Size(256, 256), CV_8UC3);
        //任何一个图像的某个像素的总个数有可能会很多，甚至超出所定义的图像的尺寸，
        //所以需要先对个数进行范围的限制，用minMaxLoc函数来得到计算直方图后的像素的最大个数
        double g_dHistMaxValue;
        minMaxLoc(dstHist, 0, &g_dHistMaxValue, 0, 0);
        //将像素的个数整合到图像的最大范围内
        for (int i = 1; i < 256; i++)
        {
            int value = cvRound(dstHist.at<float>(i) * 256 * 0.9 / g_dHistMaxValue);
            line(drawImage, Point(i, drawImage.rows - 1), Point(i, drawImage.rows - 1 - value),
Scalar(0, 0, 255));
        }
        line(drawImage, Point(0, drawImage.rows - 1), Point(0, drawImage.rows - 1 - 0), Scalar(0,
0, 255));
        imshow("hist", drawImage);
    }


    return dstHist;
}

//查找梯度最多方向
//参数:  inputImageX x方向梯度图像
//参数:  inputImageY y方向梯度图像
//参数:  outputImage 输出结果图像
//返回值:    0  - 正常
//           -1 - 异常
```

```
int findDirection(Mat &inputImageX, Mat &inputImageY, Mat &outputImage)
{
    if (inputImageX.cols != inputImageY.cols)
        return -1;
    if (inputImageX.rows != inputImageY.rows)
        return -1;

    outputImage.create(inputImageX.size(), inputImageX.type());

    short* dataX = inputImageX.ptr<short>(0);
    short* dataY = inputImageY.ptr<short>(0);
    short* data = outputImage.ptr<short>(0);

    int i, j;
    for (i = 0; i < inputImageX.rows;i++)
    {
        for (j = 0; j < inputImageX.cols;j++)
        {
            if (*dataY < 20 && *dataY > -20 && *dataX > -20 && *dataX < 20)
            {
                //梯度变化过小的剔除
                *data = 0;
            }
            else if (*dataX == 0)
            {
                if (*dataY != 0)
                {
                    *data = 255;
                }
                else
                {
                    *data = 0;
                }
            }
            else
            {
                *data = atan((float)*dataY / (float)*dataX) / PI * 2 * 254;
                //无意义数据/两个方向梯度都是0的数据，放在0里
                //结果小于1的取整为0，存为1
                if (*data == 0)
                    (*data)++;
            }
            data++;
            dataX++;
```

```
            dataY++;
        }

    }

    return 0;
}

//背景分离
//背景摸为全黑0，其他不变
//参数： inputImage 输入图像
//参数： outputImage 输出图像
//参数： threshold 阈值
//返回值：    0 - 正常
int eraseBackground(Mat &inputImage, Mat &outputImage, int threshold)
{
    outputImage.create(inputImage.size(), inputImage.type());

    uchar* dataIn = inputImage.ptr<unsigned char>(0);
    uchar* dataOut = outputImage.ptr<unsigned char>(0);

    for (int i = 0;i < inputImage.rows;i++)
    {
        for (int j = 0;j < inputImage.cols;j++)
        {
            if (*dataIn < threshold)
                *dataOut = *dataIn;
            else
                *dataOut = 255;

            dataIn++;
            dataOut++;
        }
    }

    return 0;
}

//图像显示，附带压缩显示和保存
//参数： imageName 图像名称
//参数： iamge 图像
//参数： isZip 是否压缩显示 1-压缩 0-不压缩
//参数： isSave 是否保存图片（不受上一参数影响，全分辨率保存） 1-保存 0-不保存
void myImShow(char *imageName, Mat &image, int isZip, int isSave)
```

```
{
    Mat imagZip;
    if (isZip)
    {
        resize(image, imagZip, Size(), ZIPTIME, ZIPTIME);
    }
    else
    {
        imagZip = image.clone();
    }

    if (isSave)
    {
        char * name = new char[strlen(imageName) + sizeof(char) * 4];
        memcpy(name, imageName, strlen(imageName));
        *(name + strlen(imageName)) = '.';
        *(name + strlen(imageName) + 1) = 'j';
        *(name + strlen(imageName) + 2) = 'p';
        *(name + strlen(imageName) + 3) = 'g';
        *(name + strlen(imageName) + 4) = 0;
        imwrite(name, image);
    }
    imshow(imageName, imagZip);
}


//验证是否是条形码区域
//参数: image 图像
//参数: rect 感兴趣区域
//参数: rectOut 条形码区域
//返回值:    是否是条形码
bool findBloak(Mat & image, Rect & rect,Rect & rectOut)
{
    int rectX, rectY, rectWidth, rectHeight;
    int rectEndX;

    int x = rect.tl().x;
    int y = rect.tl().y + rect.height / 2;

    int y0_0, y0_1, y0_2, y0_3;
    int y1_1, y1_2;
    int y2_1, y2_2;

    int i = 0;
```

```
if (x < rect.width || x > image.cols - rect.width)
    return false;

//x -= rect.width / 2;    //左移部分，保证监测到边沿

//x增大方向判断
int edge_last = x;
int edge_cur = 0;
int y1_sign = 0;          //一阶导方向
while (1)
{
    //零阶
    y0_0 = image.at<uchar>(y,x);
    y0_1 = image.at<uchar>(y,x-1 );
    y0_2 = image.at<uchar>(y,x-2);
    y0_3 = image.at<uchar>(y,x-3);

    //一阶导
    y1_1 = y0_1 - y0_2;
    y1_2 = y0_2 - y0_3;
    {
        if ((abs(y1_1) < abs(y1_2)) && ((y1_1 >= 0) == (y1_2 >= 0)))
            y1_1 = y1_2;
    }

    //二阶导
    y2_1 = y0_0 - (y0_1 * 2) + y0_2;
    y2_2 = y0_1 - (y0_2 * 2) + y0_3;

    //二阶导为0点，一阶导极大/极小值，可能是边沿
    if (!y2_1 || ((y2_1 > 0) ? y2_2 < 0 : y2_2>0))
    {
        if (!y1_sign && y1_1)
        {
            edge_last = edge_cur = x;
            y1_sign = y1_1;
        }
        //黑框后沿
        else if ((y1_sign < 0) && (y1_1 > 0))
        {
            edge_cur = x;
            edge_last = edge_cur;
            y1_sign = y1_1;
```

```
                i++;
            }
            //黑框前沿
            else if ((y1_sign > 0) && (y1_1 < 0))
            {
                edge_last = x;
                y1_sign = y1_1;
            }
        }


        x++;
        //黑框不超过感兴趣区域1.5倍宽
        //白色部分不超过感兴趣区域3倍宽
        if ((y1_sign > 0)?(x - edge_last > rect.width * 3):(x - edge_last > rect.width * 1.5) || (x
== image.cols))
        {
            if (i > 9)
            {
                //连续9个符合区域，是条形区域
                rectEndX = x;
                break;
            }
            else
                return false;
        }
    }

    //x减小方向判断
    x = rect.tl().x;
    edge_last = x;
    edge_cur = 0;
    y1_sign = 0;          //一阶导方向
    while (1)
    {
        //零阶
        y0_0 = image.at<uchar>(y, x);
        y0_1 = image.at<uchar>(y, x - 1);
        y0_2 = image.at<uchar>(y, x - 2);
        y0_3 = image.at<uchar>(y, x - 3);

        //一阶导
        y1_1 = y0_1 - y0_2;
        y1_2 = y0_2 - y0_3;
```

```
        {
            if ((abs(y1_1) < abs(y1_2)) && ((y1_1 >= 0) == (y1_2 >= 0)))
                y1_1 = y1_2;
        }

        //二阶导
        y2_1 = y0_0 - (y0_1 * 2) + y0_2;
        y2_2 = y0_1 - (y0_2 * 2) + y0_3;

        //二阶导为0点，一阶导极大/极小值，可能是边沿
        if (!y2_1 || ((y2_1 > 0) ? y2_2 < 0 : y2_2>0))
        {
            if (!y1_sign && y1_1)
            {
                edge_last = edge_cur = x;
                y1_sign = y1_1;
            }
            //黑框前沿
            else if ((y1_sign > 0) && (y1_1 < 0))
            {
                edge_cur = x;
                edge_last = edge_cur;
                y1_sign = y1_1;
            }
            //黑框后沿
            else if ((y1_sign < 0) && (y1_1 > 0))
            {
                edge_last = x;
                y1_sign = y1_1;
            }
        }

        x--;
        if ((y1_sign < 0) ? (edge_last - x > rect.width * 3) : (edge_last - x  > rect.width * 1.5)
|| (x == 5))
        {
            rectX = x;
            rectY = rect.tl().y;
            rectHeight = rect.height;
            rectWidth = rectEndX - rectX;
            rectOut.height = rectHeight;
            rectOut.width = rectWidth;
            rectOut.x = rectX;
            rectOut.y = rectY;
```

```
            return true;
        }
    }


    return false;
}

//16位图像找直方图最大值
//输入数据范围-255 -- +254
//0为无效数据
//参数： image 输入图像
//返回值：    直方图最大值
int hist16S(Mat &image)
{
    int maxLoc = 0;
    int maxValue = 0;
    double hist[512] = { 0 };

    short *data = image.ptr<short>(0);

    for (int i = 0; i < image.rows;i++)
    {
        for (int j = 0;j < image.cols;j++)
        {
            hist[*data + 255]++;
            if (hist[*data + 255] > maxValue && *data != 0)
            {
                maxLoc = *data;
                maxValue = hist[*data + 255];
            }
            data++;
        }
    }
    return maxLoc;
}
```