**基于OTSU的芯片薄膜合格检测技术**

**摘 要**

21世纪无疑是半导体计算机的世界，其中芯片半导体工业对于信息技术产业革命的发展起着决定性的作用。现代计算机硬件的发展决定了信息革命的瓶颈，作为构成这座科技大厦的沙石——芯片，无疑是这座大厦最基本的部分。而芯片封装的合格性对于芯片保存和运输至关重要。本文主要运用OTSU（大津法）实现对泰科公司的HTU21芯片薄膜封装的合格检测，芯片封装不合格包括左右边缘和上下边缘超出。本文采用opencv和C++语言进行项目实战，首先运用图像分割技术将同一张图片中的芯片分割成单独的芯片，之后采用边缘检测和漫水技术将芯片从背景中提取出来，运用仿射变换将当前芯片调整至水平位置，然后运用OTSU大津法对处理后的芯片进行薄膜封装合格性检测。

关键词：芯片薄膜、OTSU、图像分割、opencv、C++。

**Qualification inspection technology of chip based on OTSU Transmission**

Abstract

The 21st century is undoubtedly the world of semiconductor computers. The chip semiconductor industry plays a decisive role in the development of the information technology industry revolution. The development of modern computer hardware has determined the bottleneck of the information revolution. As the sandstone of this technology building, the chip, is undoubtedly the most basic part of the building. The qualification of the chip package is very important for chip storage and transportation. This paper mainly uses the OTSU (Otsu method) to achieve the qualified inspection of Tyco's HTU21 chip thin film package. The chip package failure includes left and right edges and top and bottom edges exceeding. This article first uses image segmentation technology to divide the chip in the same picture into separate chips, then uses edge detection and flooding technology to extract the chip from the background, uses affine transformation to adjust the current chip to a horizontal position, and then uses OTSU The Otsu method performs thin-film packaging qualification testing on the processed chips.

Keywords: Chip film, OTSU, image segmentation, opencv,C++.

1 相关介绍

1.1 程序流程框图

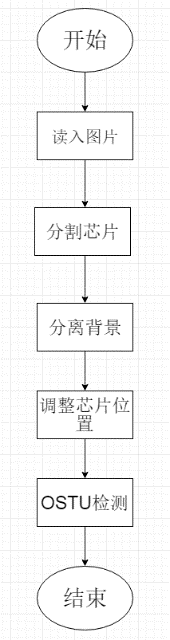


图1 程序流程图

本次项目主要分为4个步骤进行，第一步首先根据芯片自身特点将原图中的多个芯片分别分离出来；第二步运用边缘检测技术以及种子填充技术将芯片从背景中提取出来；第三步运用仿射变换技术将提取出来的芯片调整至水平位置；第四步采用OTSU法对调整后的芯片进行合格性检测。

**1.2 认识OpenCV**

OpenCV是一个开源的计算机视觉和机器学习软件库。OpenCV主要为开发计算机视觉程序提供一组公共的底层结构和加强商业产品中机器的感知能力。OpenCV使用BSD许可证,它可以被商用并可以修改源代码。OpenCV库有超过2500个已优化的算法，既包含了经典的也包含了最先进的计算机视觉和机器学习算法。这些算法可以被用来检测和识别人脸，跟踪移动的物体，提取物体的三维模型,从立体相机中产生3D point clouds,可以将图片拼接来生成一张高解析度的全景图像，从图像数据库中找到相似性的图像等等等等。OpenCV已经有超过4万7千的用户群体，并且估计下载数量超过1400万。使用OpenCV的公司有谷歌、雅虎、微软、IBM、索尼、本田、丰田等。

OpenCV有C++、Java、Python和Matlab四个版本，支持Windows、Linux、Android和MacOS平台。OpenCV主要倾向于实时的视觉应用程序，当MMX和SSE(MMX和SSE是因特尔开发的单指令多数据流指令集，后者是前者的扩展，具体不展开，有兴趣可自查)可用的时候OpenCV能够很好的去使用这两者的特性。一个全功能(full-feature)的CUDA和OpenCL接口已被开发，有超过500个算法和大约10倍支持这些算法的组建或支持。OpenCV用native C++开发，支持模板，可与STL容器无缝对接[1]。

**2 项目实战**

2.1分割芯片

2.1.1 形态学开操作

形态学一般指生物学中研究动物和植物结构的一个分支。用数学形态学（也称图像代数）表示以形态为基础对图像进行分析的数学工具。基本思想是用具有一定形态的结构元素去度量和提取图像中的对应形状以达到对图像分析和识别的目的。形态学图像处理的数学基础和所用语言是集合论。形态学图像处理的应用可以简化图像数据，保持它们基本的形状特性，并除去不相干的结构。形态学图像处理的基本运算有：膨胀、腐蚀、开操作和闭操作，击中与击不中变换，TOP-HAT变换，黑帽变换等。

膨胀和腐蚀操作的核心内容是结构元素。一般来说结构元素是由元素为1或者0的矩阵组成。结构元素为1的区域定义了图像的领域，领域内的像素在进行膨胀和腐蚀等形态学操作时要进行考虑。膨胀是腐蚀运算的对偶运算，其作用是在结构元素的约束下将与目标区域相接触的背景合并到该目标物中，使目标边界向外部扩张，物体的面积增大了相应数量的点。腐蚀是一种在结构元素约束下消除目标图形的部分边界点，使其边界向内部收缩的的算法，具有收缩目标区域的作用[2]。

形态学开操作是先腐蚀后膨胀，用来消除图像中细小对象，在纤细点处分离物体和平滑较大物体的边界而有不明显改变其面积和形状。开运算的几何解释如图2所示：

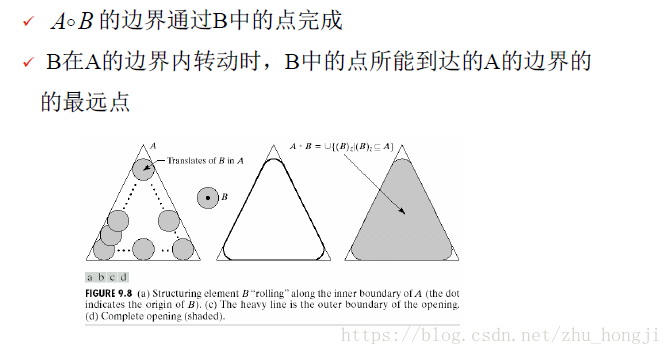


图2 开操作几何解释图

2.1.2 边缘轮廓检测

如图3所示，cv::findCountour()的基本功能，图的上部是一幅测试图像，其背景为白色，并含有数个彩色的的区域（标签A到E）。图中也绘制出了由cv::findContours()所确定的轮廓。这些轮廓被标记为cX或hX,其中c代表“contour(轮廓)”，h代表“洞（hole）”,而X是一些数字。有些轮廓使用虚线表示的，他们表示白色区域（即非零区域）的外部边界。OpenCV和cv::findContour()对这些外部边界和图中的点线，即内部边界或者是洞的外部边界，进行区分的。

OpenCV可以将找到的轮廓组织成轮廓树，表示其轮廓结构的包围关系。对于测试图像中的轮廓，我们将根节点处的轮廓称为c0，而“洞”h00和h01是其子节点。反过来这些子节点又会包含新的子节点以此类推。

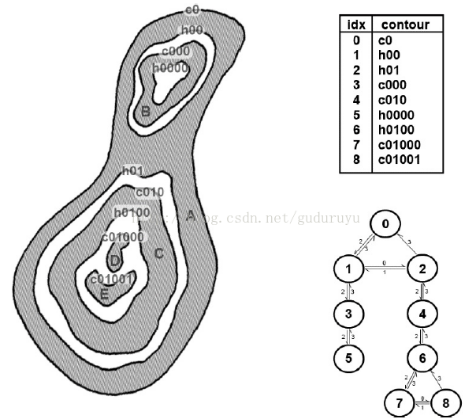


图3 轮廓查找示意图

表示这种树的方式有很多种，OpenCV中使用数组（尤其是vectors）来表示这种树，其中数组中的每个条目都代表一个特定的轮廓，每个条目包含一个由4个整数组成的集合（通常表示为cv :: Vec4i类型的元素，就像四通道数组中的条目一样）。对于每个节点来说，四个元素所表示的含义分别如下：0号元素表示下一个轮廓（同一层级）；1号元素表示前一个轮廓（同一层级）；2号元素表示第一个子轮廓（下一层级）；3号元素表示父轮廓（上一层级）[3]。

2.1.3 分割芯片步骤

现在伴随着深度学习和计算机硬件的飞速发展，图像分割已经越来越复杂和精准化。与运用图像特征值的深度学习图像分割相比，本次项目抓住了芯片的多边形特性和背景的简单性，大大简化分割图像的步骤。本次项目的芯片摆放如图2所示：

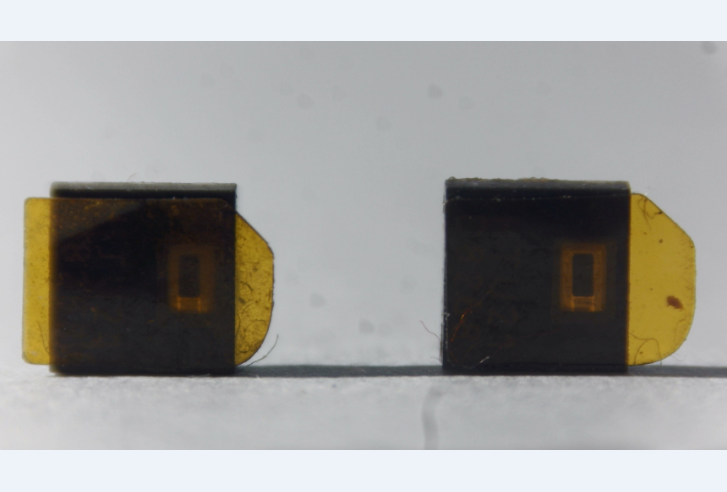


图4 芯片示例图

分割芯片步骤如下：

1、将芯片图三通道彩色图像转换成灰度图像，并选定阈值150进行图像二值化。

2、由于背景中会有少许的杂乱像素以及芯片照相影子阴影影响，所以对转化后的二值图像进行开操作，腐蚀膨胀芯片周围的干扰像素，同时使得不同芯片之间分得更清楚。

3、对进行开操作后的图像找寻轮廓，并对找到的轮廓进行长方形轮廓拟合，对于拟合后的长方形长度和宽度不满足要求的进行舍弃操作，本次项目中的长方形长度需超过图像的一半。将符合条件的长方形保存在一个vector<rect\_vec>中。

4、对于第3步中得到的长方形轮廓外围，取出其左顶点以及长宽即可得到在原图像中芯片需要切割的坐标点，依据坐标点将芯片从原图中复制出来，便实现了芯片的简单分割。

主要程序代码如下：

void spilt\_product(Mat pic) //分离产品

{

Mat grayImage, binPic;

vector<int> cols;

vector<vector<Point>> contours;

vector<Vec4i> hireachy;

vector<Rect> Rect\_vector;

cvtColor(pic, grayImage, CV\_BGR2GRAY);

threshold(grayImage, binPic, 150, 255, 1);

Mat element = getStructuringElement(MORPH\_ELLIPSE, Size(30, 30));

Mat openMat;

Rect rect\_temp;

morphologyEx(binPic, openMat, MORPH\_OPEN, element);

findContours(openMat, contours, hireachy, CV\_RETR\_EXTERNAL, CHAIN\_APPROX\_SIMPLE, Point());

cout << "contours.size:" << contours.size()<<endl;

for (size\_t i = 0; i < contours.size(); i++)

{

rect\_temp = boundingRect(Mat(contours[i]));

if ((rect\_temp.height >(binPic.rows / 3)) && (rect\_temp.width < (binPic.cols / 2)))

Rect\_vector.push\_back(rect\_temp);

}

copy\_chip2\_separated\_pic(pic, Rect\_vector,600,400);

cout << "size:" << pic\_mat\_vector.size() << endl;

}

程序运行结果如图3所示：

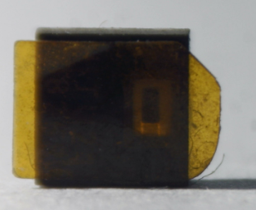
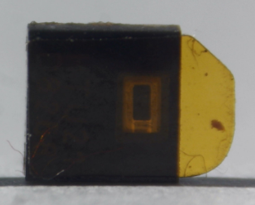
 

图5 分割芯片程序运行图

此时已经初步从原来图像中将芯片分离出来，但用于后期的合格检测还远远不够，因为在实际工业中还存在背景的杂乱和影子的影响问题，还需要对分离出的芯片进行进一步的处理，才能用于进一步的合格检测。

2.2 分离背景

为了进一步更精准的判别芯片合格性，需要将芯片从背景中提取出来。考虑到芯片像素明显与背景像素不同，采用边缘检测与种子填充技术实现芯片的进一步提取。

2.2.1种子填充

种子填充算法是从多边形区域内部的一点开始，由此出发找到区域内的所有像素。种子填充算法采用的边界定义是区域边界上所有像素具有某个特定的颜色值，区域内部所有像素均不取这一特定颜色，而边界外的像素则可具有与边界相同的颜色值。具体算法步骤：

1、标记种子（x,y）的像素点 ;

2、检测该点的颜色，若他与边界色和填充色均不同，就用填充色填充该点，否则不填充 ;

3、检测相邻位置，继续 2。这个过程延续到已经检测区域边界范围内的所有像素为止。

当然在搜索的时候有两种检测相邻像素：四向连通和八向连通。四向连通即从区域上一点出发，通过四个方向上、下、左、右来检索。而八向连通加上了左上、左下、右上、右下四个方向。这种算法的有点是算法简单，易于实现，也可以填充带有内孔的平面区域[4]。

2.2.2 分离背景步骤

此时芯片已经被独立分割出来，但由于还存在少许背景，仍有可能对后续的识别操作造成影响，需要将芯片从背景中提取出来，具体步骤如下：

1、将芯片图转换成二值图像，找出芯片的边缘轮廓。

2、根据芯片的边缘轮廓进行种子填充，使得芯片部分成为单一像素。

3、根据步骤2生成的填充图，在原图上比对像素复制出芯片。

主要程序代码如下：

Mat extract\_chip(Mat Soupic)

{

Mat result\_pic;

Mat temppic;

vector<vector<Point>> contours;

vector<Vec4i> hireachy;

Mat grayImage, binPic;

cvtColor(Soupic, grayImage, CV\_BGR2GRAY);

threshold(grayImage, binPic, 150, 255, 1);

imshow("binPic", binPic);

findContours(binPic, contours, hireachy, CV\_RETR\_EXTERNAL, CHAIN\_APPROX\_SIMPLE, Point());

Mat hole(binPic.size(), CV\_8U, Scalar(0));

drawContours(hole, contours, -1, Scalar(255), CV\_FILLED);

for (int i = 0; i < Soupic.rows; i++) {

for (int j = 0; j < Soupic.cols; j++) {

if (hole.ptr<uchar>(i)[j] == 0)

{

Soupic.ptr<uchar>(i)[Soupic.channels()\*j] = 255;

Soupic.ptr<uchar>(i)[Soupic.channels()\*j + 1] = 255;

Soupic.ptr<uchar>(i)[Soupic.channels()\*j + 2] = 255;

}

}

}

return Soupic;

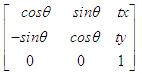
}

2.3 调整芯片位置

在现实的工业生产中，芯片检测不一定都会相对照相机水平位置摆放，为了增加程序的鲁棒性和更加贴近实际工业生产，还需对分离背景后的芯片图进行位置调整，本文主要运用仿射变换进行位置的调整。

2.3.1仿射变换

仿射变换是二维平面中一种重要的变换，在图像图形领域有广泛的应用。许多人对“仿射”没有一个感官的认识，我觉得很有必要先来说一下“仿射”。所谓的“仿射变换”就是一种简单的变换，它的变化包括旋转、平移、伸缩，原来的直线仿射变换后还是直线，原来的平行线经过仿射变换之后还是平行线，这就是仿射。仿射变换的矩阵是其次坐标形式的变换矩阵。



这个矩阵包含的变换有旋转和平移，其实是两个矩阵的混合体，仿射变换的数学公式里，是如何做到坐标点位置的平移呢？清楚这个才是弄明白仿射变换的关键 。

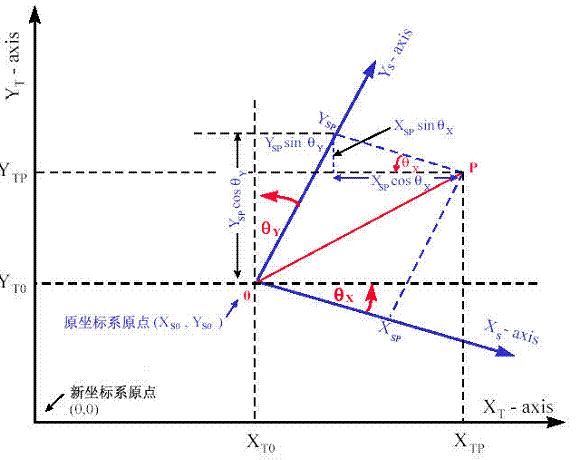
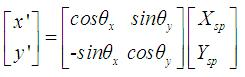


图6 仿射变换示意图

一个点P在原始坐标系下的坐标是(Xsp,Ysp)。然后要完成旋转操作，旋转操作是基于原点的，如何得到旋转之后的点的坐标，这里用到一个技巧，坐标系中某个点的旋转可以等价地去旋转坐标轴，所以有了上图中以(Xs0,Ys0)为中心的虚线与屏幕水平垂直的坐标系。在这个坐标系中确定P的坐标，和在蓝色坐标系中确定旋转之后P的坐标是等价的。基于这个结论，我们可以通过简单的立体几何知识确定P在新坐标系中的坐标。P在新坐标系中的X坐标和Y坐标分别是：

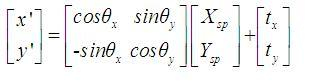


经典的仿射变换的模型呼之欲出了。整理上面两个式子得：



这就是仿射变换模型中旋转部分的原理，还有一步，就是平移。

旋转变换之后，我们确定了P点在新坐标系中的位置，然后在这个位置的基础上加上其在X轴和Y轴的偏移即可[5]



主要程序如下：

Mat rotate\_pic(Mat pic, float degree) // 旋转芯片

{

Mat out\_img;

double angle = 90 - degree;

int rotated\_width = ceil(pic.rows \* fabs(sin(angle \* CV\_PI / 180)) + pic.cols \* fabs(cos(angle \* CV\_PI / 180)));

int rotated\_height = ceil(pic.cols \* fabs(sin(angle \* CV\_PI / 180)) + pic.rows \* fabs(cos(angle \* CV\_PI / 180)));

// 计算仿射变换矩阵

Point2f center(pic.cols / 2, pic.rows / 2);

Mat rotate\_matrix = getRotationMatrix2D(center, angle, 1.0);

// 防止切边，对平移矩阵B进行修改

rotate\_matrix.at<double>(0, 2) += (rotated\_width - pic.cols) / 2;

rotate\_matrix.at<double>(1, 2) += (rotated\_height - pic.rows) / 2;

// 应用仿射变换

warpAffine(pic, out\_img, rotate\_matrix, Size(rotated\_width, rotated\_height), INTER\_LINEAR, 0, Scalar(255, 255, 255));

return out\_img;

}

仿射变换后的芯片图如下：

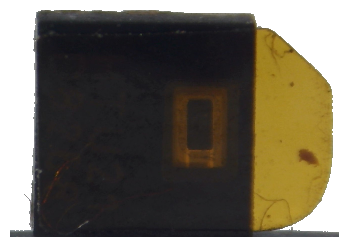
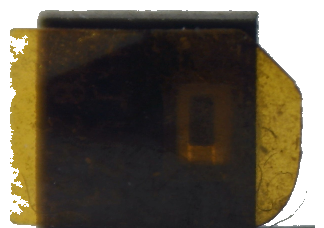
 

图7 芯片最终分离图

2.4 OTSU

经过上面的处理之后，芯片已经能够用于最后的检测。分离后的芯片图主要分为黑色芯片与黄色薄膜重叠部分和黄色薄膜超出芯片部分，像素比较单一。由像素分布直方图可以看出，像素主要集中在特定区间。于是可以采用OTSU法将黑色芯片与黄色薄膜重叠部分找出，再将黄色薄膜部分找出，查看两者是否在规定位置。

2.4.1 OTSU介绍

大津法又叫最大类间方差法、最大类间阈值法（OTSU）。它的基本思想是，用一个阈值将图像中的数据分为两类，一类中图像的像素点的灰度均小于这个阈值，另一类中的图像的像素点的灰度均大于或者等于该阈值。如果这两个类中像素点的灰度的方差越大，说明获取到的阈值就是最佳的阈值（方差是灰度分布均匀性的一种度量,背景和前景之间的类间方差越大,说明构成图像的两部分的差别越大,当部分前景错分为背景或部分背景错分为前景都会导致两部分差别变小。因此,使类间方差最大的分割意味着错分概率最小。）。则利用该阈值可以将图像分为前景和背景两个部分。而我们所感兴趣的部分一般为前景。该算法原理如下[6]：

1、背景像素占比 ， 代表背景像素，代表像素总数

2、前景像素占比= ， 代表前景像素

3、背景的平均灰度值

4、前景的平均灰度值

5、0~M灰度区间的灰度累计值

6、类间方差：

7、最终化简公式得：

OTSU算法程序如下：

int myOtsu(Mat & src)

{

int th;

const int GrayScale = 256; //单通道图像总灰度256级

int pixCount[GrayScale] = { 0 };//每个灰度值所占像素个数

int pixSum = src.cols \* src.rows;//图像总像素点

float pixPro[GrayScale] = { 0 };//每个灰度值所占总像素比例

float w0, w1, u0tmp, u1tmp, u0, u1, deltaTmp, deltaMax = 0;

for (int i = 0; i < src.cols; i++)

{

for (int j = 0; j < src.rows; j++)

{

pixCount[src.at<uchar>(j, i)]++;//统计每个灰度级中像素的个数

}

}

for (int i = 0; i < GrayScale; i++)

{

pixPro[i] = pixCount[i] \* 1.0 / pixSum;//计算每个灰度级的像素数目占整幅图像的比例

}

for (int i = 0; i < GrayScale; i++)//遍历所有从0到255灰度级的阈值分割条件，测试哪一个的类间方差最大

{

w0 = w1 = u0tmp = u1tmp = u0 = u1 = deltaTmp = 0;

for (int j = 0; j < GrayScale; j++)

{

if (j <= i)//背景

{

w0 += pixPro[j];

u0tmp += j \* pixPro[j];

}

else//前景

{

w1 += pixPro[j];

u1tmp += j \* pixPro[j];

}

}

u0 = u0tmp / w0;

u1 = u1tmp / w1;

deltaTmp = (float)(w0 \*w1\* pow((u0 - u1), 2)); //类间方差公式 g = w1 \* w2 \* (u1 - u2) ^ 2

if (deltaTmp > deltaMax)

{

deltaMax = deltaTmp;

th = i;

}

}

return th;

}

2.4.2 芯片薄膜合格性判别标准

根据上节中的OTSU算法可将最佳前景与背景最佳阈值找出，根据最佳阈值对芯片图片进行二值化，可以得到只含有黑色芯片部分的阈值图片与只含有薄膜部分的二值图片，再分别比较两个二值图片间的坐标关系，即可得知该芯片薄膜是否合格。下图为OTSU最佳阈值后的二值化图片：

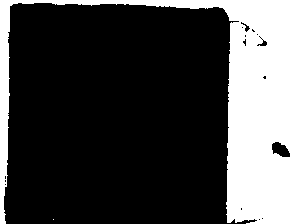
 

图8 OTSU取最佳阈值后芯片薄膜分离图

对于得到的二值化图片进行坐标比对检验薄膜是否合格，检验步骤如下：

1、首先将两个二值化图像的轮廓找出，由于没有背景干扰作用，轮廓即为黑色芯片和黄色薄膜部分的轮廓。

2、对两个轮廓求解出最小外接矩形，则对于最小外接矩形的顶点和几何参数信息都存储在求出的最小矩形容器中。

3、求出两个矩形的左下标，通过对两个矩形左下标的坐标比对，判断芯片薄膜是否上下超出。比对两个矩形的宽和高，判断芯片薄膜是否左右超出。

程序判断如下：

void judge\_product(Mat pic1, Mat pic2)//对二值化后的两幅图进行分析 pic1为黑色芯片二值图，pic2为有黄色薄膜二值图

{

vector<vector<Point>> contours1;

vector<vector<Point>> contours2;

vector<Vec4i> hireachy1;

vector<Vec4i> hireachy2;

Rect rect1;

Rect rect2;

int area;

int i = 0;

findContours(pic1, contours1, hireachy1, RETR\_TREE, CHAIN\_APPROX\_SIMPLE, Point(0, 0));

findContours(pic2, contours2, hireachy2, RETR\_TREE, CHAIN\_APPROX\_SIMPLE, Point(0, 0));

int area1\_max = 0;

int area2\_max = 0;

vector<Point> contours1\_max = contours1[0];

vector<Point> contours2\_max = contours2[0];

for (; i < contours1.size(); i++)

{

Rect rect = boundingRect(contours1[i]);

if (rect.height>pic1.rows - 5) continue;

area = contourArea(contours1[i]);

if (area > area1\_max)

{

area1\_max = area;

contours1\_max = contours1[i];

}

}

i = 0;

for (; i < contours2.size(); i++)

{

Rect rect = boundingRect(contours2[i]);

if (rect.height>pic2.rows - 5) continue;

area = contourArea(contours2[i]);

if (area > area2\_max)

{

area2\_max = area;

contours2\_max = contours2[i];

}

}

rect1 = boundingRect(Mat(contours1\_max));

rect2 = boundingRect(Mat(contours2\_max));

Mat rect\_pic1 = Mat::zeros(pic1.size(), CV\_8UC3);

Mat rect\_pic2 = Mat::zeros(pic2.size(), CV\_8UC3);

rectangle(rect\_pic1, rect1.tl(), rect1.br(), Scalar(0, 0, 255), 2, 8, 0);

rectangle(rect\_pic2, rect2.tl(), rect2.br(), Scalar(0, 0, 255), 2, 8, 0);

//imshow("rect1\_pic",rect\_pic1);

//imshow("rect2\_pic", rect\_pic2);

int rect1\_y = rect1.y + rect1.height; //矩形1的左下坐标

int rect2\_y = rect2.y + rect2.height;

if (abs(rect1\_y - rect2\_y) > (rect1.height\*0.03))

cout << "bad product:Lower edge overflowed!" << endl;

else if (rect2.width\*0.95 > rect1.width)

cout << "bad product:left and right sides overflowed" << endl;

else

cout << "good product" << endl;

}

程序运行结果图如下：

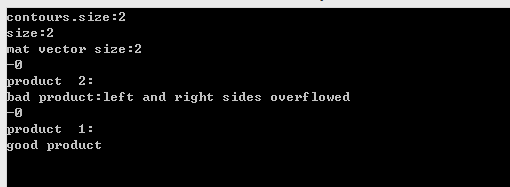


图9 程序运行结果图

3 总结展望

首先非常感谢泰科公司提供的这个芯片薄膜识别项目，让我获益匪浅。在这次项目中，不仅仅学到了实实在在的图像处理知识，让我对OpenCV有了更进一步的理解和运用，对于C++这门语言产生了更加强烈的兴趣，更为重要的是，本次项目极大的锻炼了我的项目实战能力，让我明白真正的工业生产实际中遇到的问题仅仅靠理论知识是解决不了的，必须参与到项目实战中去，才能够学以致用。

本次项目历时半个月左右，刚刚接到项目时，心里非常忐忑。因为鄙人之前从来没有接触过图像处理方面的项目，仅仅知道一张图片是由一个二维矩阵组成，更加谈不上去处理图像，还要分辨出芯片薄膜是否合格。但俗话说的好，只要思想不滑坡，办法总比困难多，你不试试怎么知道自己不行呢？接下项目后，我开始从更方面查找资料，对前期的芯片图片处理比较顺利，可对于后边的如何判断芯片薄膜合格性不知所措。与小组成员不停的讨论，不停举出方案，不停实验，不停的失败。最开始我们想到一个可行方案是通过找出芯片两侧边缘部分，根据两侧边缘的像素值内部变化是否一致来判断薄膜是否合格，但很快发现，这种方法不仅耗时，而且不靠谱，收到芯片打光和位置的影响，于是项目陷入了僵局。

期间，我们试过模板匹配、像素分布直方图等等方法，最后皇天不负有心人，曾礼虎同学突然间看到了OTSU法，想到我们也许能够利用薄膜和芯片本身之间的像素不同而将两者分离出来。于是，我们运用OTSU法，幸运的是，虽然这种方法原理简单，但对于此次项目却非常实用。

通过本次项目，我深刻认识到一定要多着眼于项目问题的细节所在，在对项目问题没有经过详细研究与仔细思考的前提下，贸然进行实战只会浪费更多的时间和心血。知己知彼，方能百战不殆。

19年的美国封杀中国科技的行为让我更加一步了解了科技必须自主创新，才能不被别人扼住咽喉。科技无小事，即使是一枚小小的芯片薄膜，关乎就是一个芯片的质量过关与否，那更加就是一个产品的质量过关与否，更是一个国家的科技发展水平的小小缩影。勿以事小而不为，一道大堤的溃败往往就是从最初的一个小小的蚁洞开始的。作为计算机专业的学生，无法在芯片设计上做到实质性的贡献，但对于后期的芯片合格性自动化检测上，运用自己的专业知识设计出严谨合格的程序便是职责所在。

参考文献：

[1] 王爱玲,叶明生.图像处理技术与应用.电子工业出版社.

[2] 周平,郑文刚,孙忠富.基于局部信息的图像滤波及边缘锐化算法[J].计算机应用,2009,(06).

[3] <https://www.cnblogs.com/henuliulei/p/10645109.html>

[4] <https://blog.csdn.net/keneyr/article/details/83753585>

[5] <https://blog.csdn.net/u011681952/article/details/98942207>

[6] <https://blog.csdn.net/weixin_40647819/article/details/90179953>