第三章 完整芯片载板的芯片检测

3.1 图像预处理和分水岭算法进行图像分割

图像分割是进行更高层次的图像处理和图像分析的重要基础。第一节的目标就是将如下图所示的图片的下半部分完整的芯片载板区域分割出来方便后续处理。基于形态学的分水岭算法则是进行图像分割的一种重要方法。此处采用分水岭算法对图像进行分割。

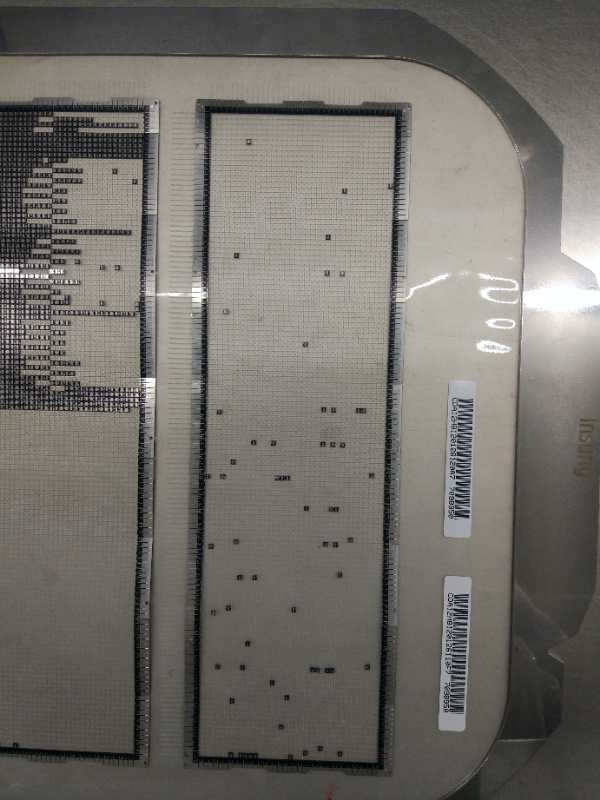


图 3-1.芯片载板图片

但分水岭算法容易产生过度分割的问题，原因是图像成像时存在拍摄系统中的噪声、光照、角度以及位置等干扰因素的影响，使目标的灰度并不均匀，而是由一些灰度相似的区域组成，如果不消除干扰因素的影响, 那么在组成目标的这些极值区域、噪声区域以及中间区域在梯度图像中都会变成极小区域而被分割出来。以下为直接对未经预处理的原始图像的梯度图像直接进行分水岭变换的效果图：

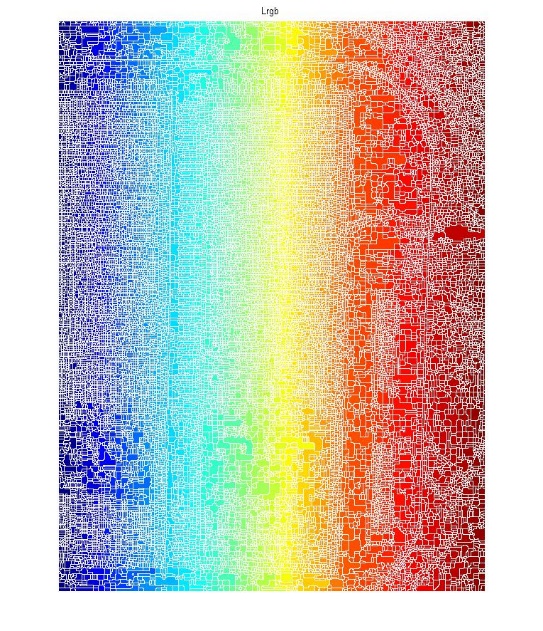
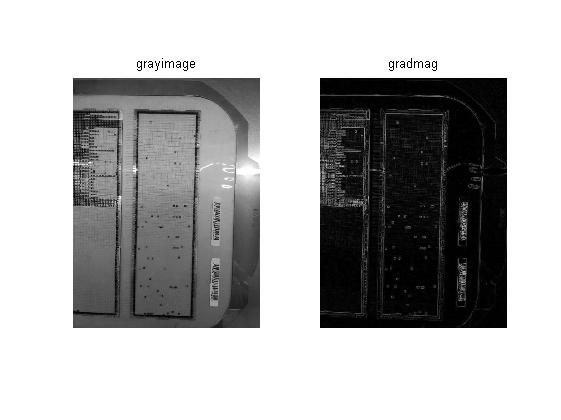


图 3-2. 直接使用分水岭变换的图像

如上图3-2所示，对梯度图像直接使用分水岭算法产生了明显的过度分割。事实上，过分割问题一直是分水岭算法研究的重点,目前主要有两种的解决方法:一是在使用分水岭算法之前,使用标记等预处理方法,抑制由于噪声和细纹理而产生的伪边界,许多文献都是利用这种提取标记的方法,文献[5-6]都是利用形态学重构算子等技术进行预处理,再提取标记,进行分水岭的图像分割;二是在使用分水岭算法之后,利用灰度图的纹理等特征,进行相似区域的合并,但是分水岭变换后区域数目很多,合并过程很复杂,如一部分文献利用分水岭算法与区域合并的方法相结合。在第一种解决方法中,标记的修正很重要,因为标记直接影响分割的效果。第二种方法比第一种方法复杂度高很多。

本文采用第一种消除过度分割的方法，即先对芯片载板图像进行形态学梯度运算,然后再利用变换提取区域极大值并分析极大值的位置及其边缘,提取新的标记,利用强制最大标定技术进行重标记,最后,对标记后的梯度图像进行分水岭算法的图像分割。

于是本文首先读取图像，将彩色图像转化为灰度图像，之后使用sobel算子进行边缘检测，即前文提到的分别采用两组3x3的矩阵，分别为横向及纵向模板，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，图像的每一个像素的横向及纵向梯度近似值可结合来计算梯度的大小，效果如图3-1右：



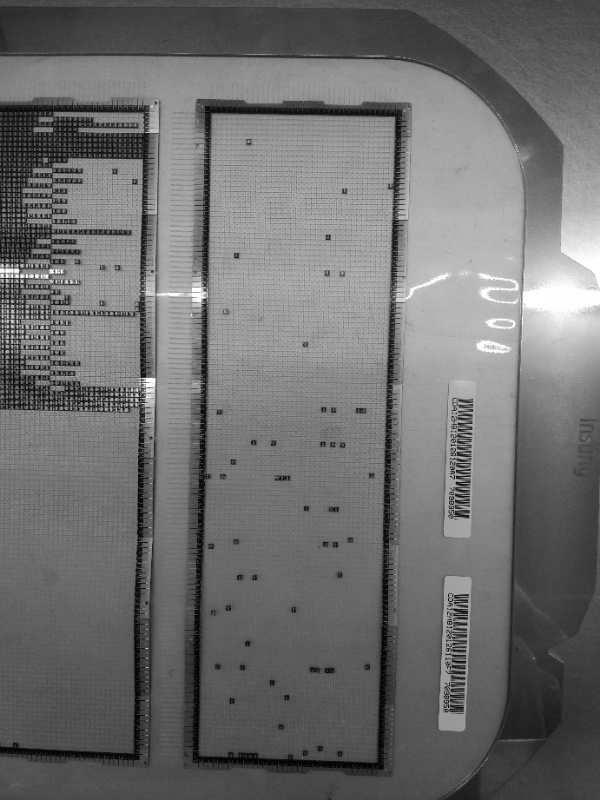


图 3-3. 芯片载板区域灰度图（左）、芯片载板区域梯度图（右）

接下来对原图进行腐蚀和重建。腐蚀与膨胀涉及的一个重要概念就是核，也可以称之为模板或者掩码。核具有几个重要的属性，形状（圆形、方形、十字架甚至椭圆），大小（3x3，5x5等）以及参考点。多数情况下，模板比较简单的是一个圆形，大小3x3，参考点在圆心，如下图所示：

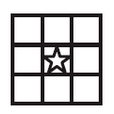


图 3-4. 3×3的方形结构元素（核）

不同的腐蚀与膨胀算法，本质的区别在于核的设计，如果核的设计与图像场景以及相关目的比较吻合时，效果也许就会好。本文选用的是一个半径为20的圆形结构元素。

腐蚀的具体机理并没有很复杂，确定好选用的核以后，与原图像进行卷积。原图像的每个像元的灰度值等于以该像元为参考点的核所覆盖的范围内，所有像元的灰度最小值，这样就完成了腐蚀操作。与腐蚀相反，膨胀操作就是取所有像元的灰度最大值。

本文先对图像进行腐蚀，之后对图像进行重建，重建的效果图如图3-4左图。此后本文再次对图像进行膨胀重建，效果图如3-4右图。

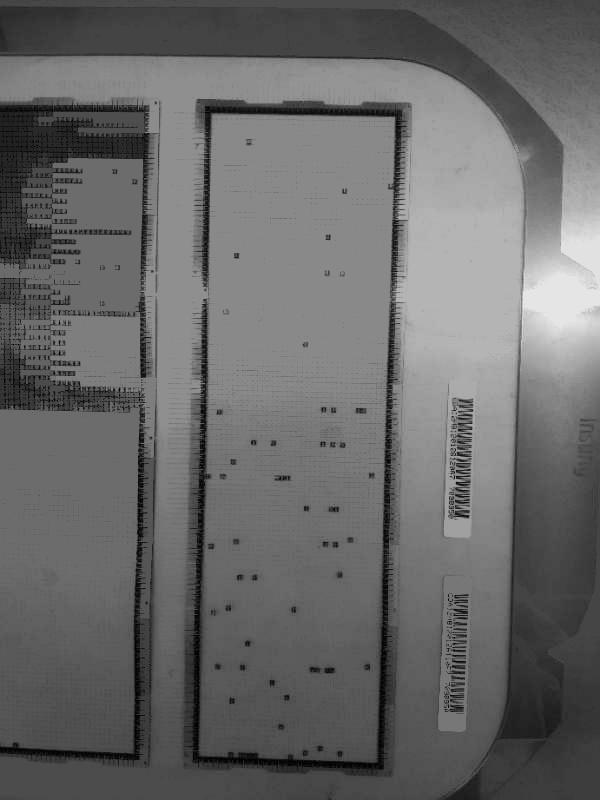


图 3-5. 腐蚀重建图（左）、膨胀重建图（右）

先腐蚀后膨胀称为开运算，而先膨胀后腐蚀称为闭运算。开和闭这两种运算可以除去比结构元素小的特定图像细节，同时保证不产生全局几何失真。之前上文进行的是基于重建的开运算，事实上，在移除小污点同时不影响对象全局形状的应用下，基于重建的开闭操作要比标准的开闭重建更加有效。之后本文通过计算Iobrcbr（原图像进行基于重建的开操作之后的图像）的局部极大来得到更好的前景标记，一般来说图像的主体为前景，其余部分称之为背景，例如人物图像，前景就是人物，而其他的就叫背景。前景标记的结果如下图所示。

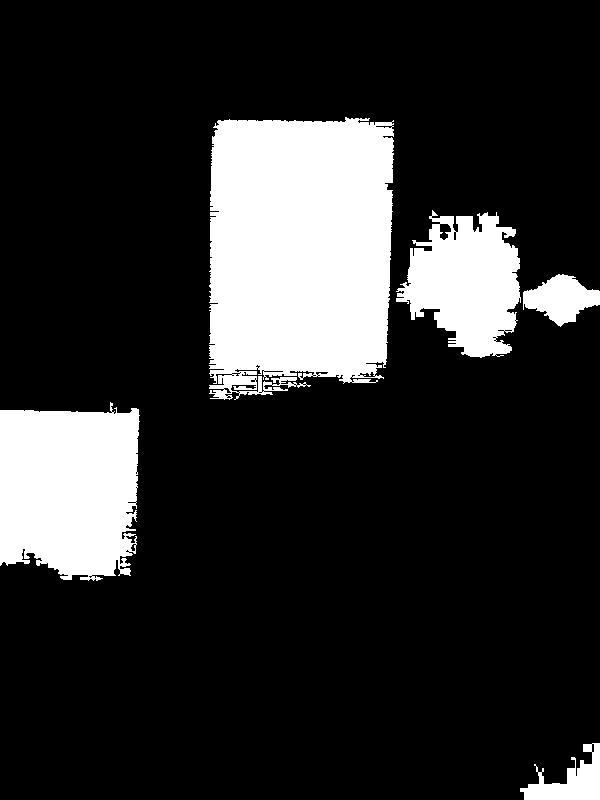


图 3-6. 局部极大前景标记（左图）、原图与前景标记的叠加

注意到大多闭塞处和阴影对象没有被标记，这就意味着这些对象在结果中将不会得到合理的分割。而且，一些对象的前景标记会一直到对象的边缘。这就意味着应该清理标记斑点的边缘，然后收缩它们。可以通过闭操作和腐蚀操作，此处本文采用的是5×5大小的方形模板来完成，得到修改后的局部极大区域，将修改后的极大区域显示在原图上如图3-6左所示。之后利用修改后的局部最大值区域对之前得到的芯片载板区域的梯度图像gradmag（图3-1右）进行形态重建修改得到gradmag2。现在直接对gradmag2进行分水岭变换得到L，为了清楚起见本文将L转化为伪彩色图像如下图3-6右所示。

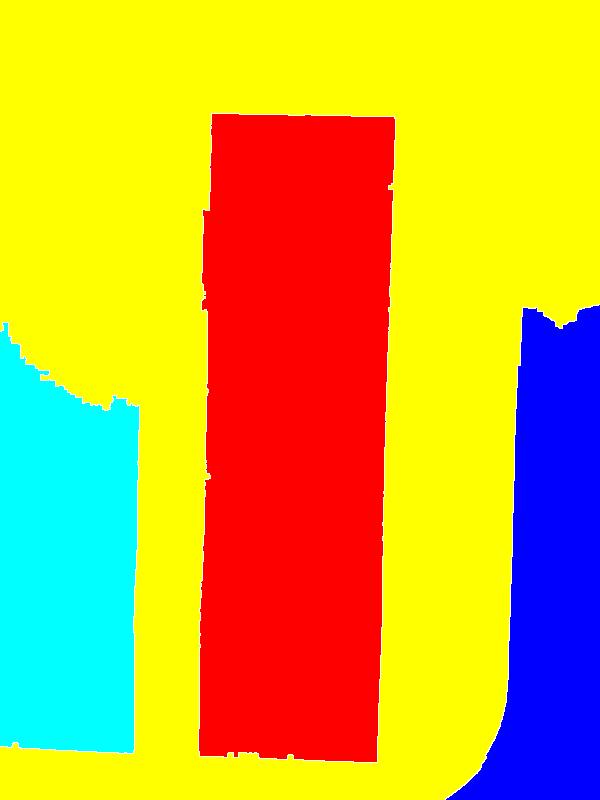


图 3-7. 原图与修改后的前景标记的叠加（左）、伪彩色图像（右）

之后为了显示方便，可以在原图上显示伪彩色图像，如下图左所示，可以看到，使用分水岭算法标记图像右侧的完整芯片载板区域取得了较好的效果。然后我们将完整的芯片载板区域单独分割出来，如下图右所示

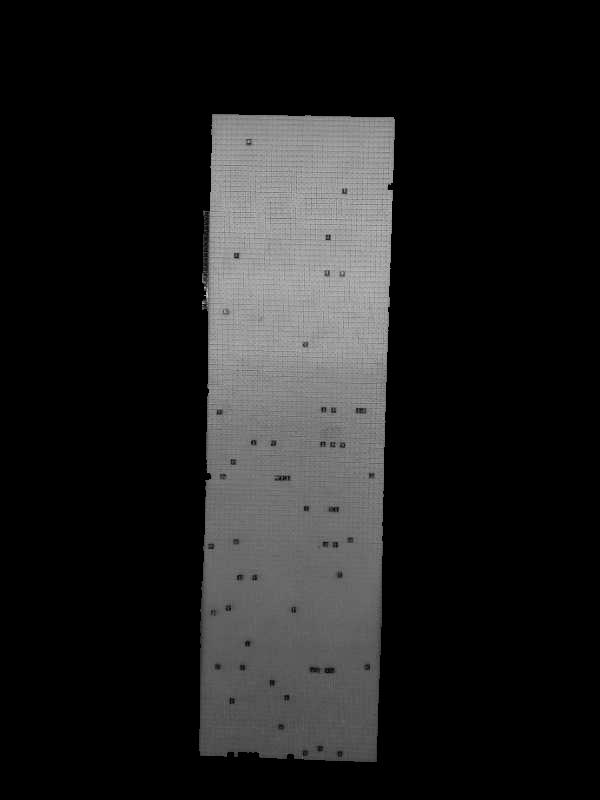
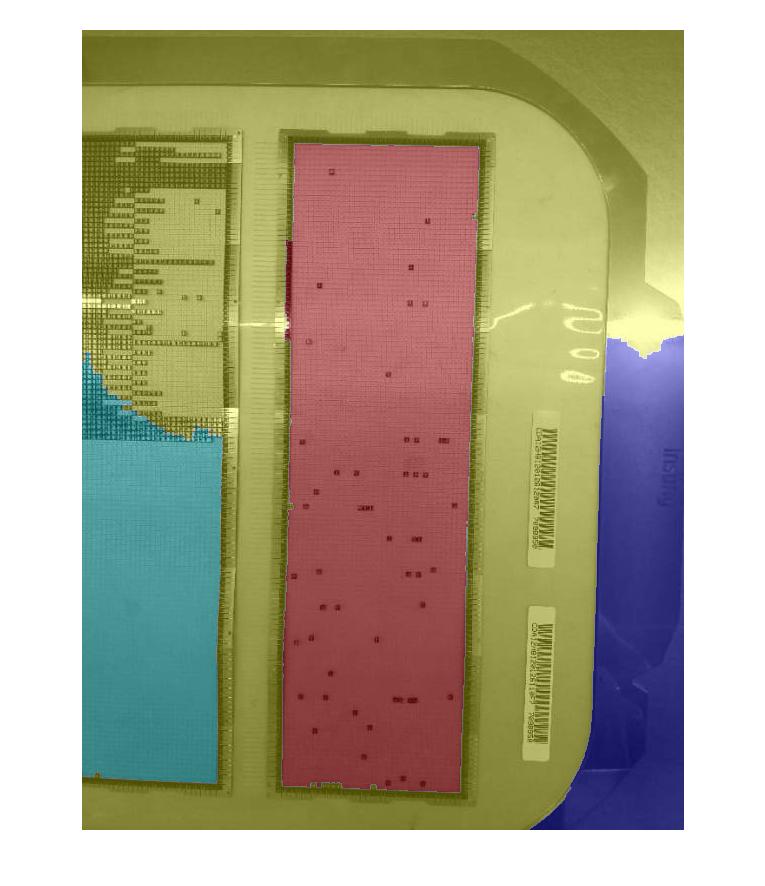


图 3-8. 原图伪彩像（左）、分割后的结果

3.2 芯片载板区域的透视变换

由于图片拍摄角度造成的形变，芯片载板区域以一个不规则类四边形（图像边缘充满了锯齿状的空隙）呈现在图像上，为了接下来处理的方便，当务之急是将不规则的类四边形区域转化为四边形，之后再将四边形区域通过透视变换转化为矩形。

为了将不规则的类四边形转化为矩形，我们先将这个区域的轮廓进行提取，之后根据轮廓寻找四个角的位置，将四个角分别相互连接即可得到一个四边形。目前本文使用的寻角算法是根据四个角的位置特征来进行遍历寻找，例如左上角为轮廓点集中x值最小的点组成的点集中y值最小的点，右上角为轮廓点集中x值最大的点组成的点集中y值最小的点。以此类推，这样可以得到关于四个角位置的初步推测，之后再以每个点为圆心，2像元为半径画一个圆。我们用此范围内的点依次对最初的角进行替换进行拟合度计算，如果拟合度高于此时的角的位置，则选取新的像元作为矩形的顶点，对矩形顶点进行更迭。

此种根据位置特征找角的算法在此存在一定问题，即当角的位置存在芯片时，寻角的偏差将会比较大。好在对于此张图片，目前的角的定位效果可以接受。之后就可以连接四个角生成一个规则的四边形区域如下图3-9所示，从图中可以看到，原来边缘位置的锯齿已经被消除。

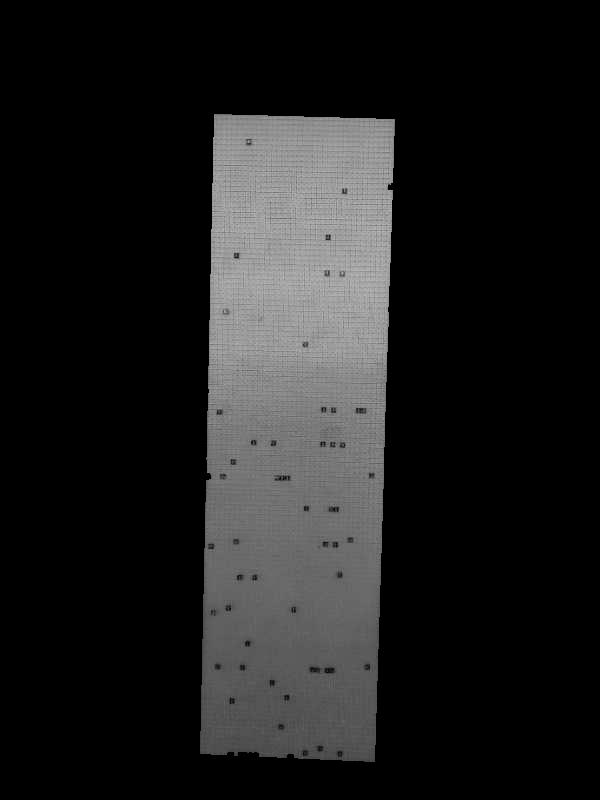


图 3-9.规则四边形区域

接下来需要通过透视变换将四边形区域转化为矩形。本文使用的变换方法即前文提到透视变换。在透视变换的过程中，使用的插值方法为最近邻域插值，根据目标图像的像素点找到原始图像中的4个像素点，取距离该像素点最近的一个原始像素值作为该点的值，也可酌情使用双线性插值等高级插值方法。使用透视变换之后的结果如下图3-10所示：

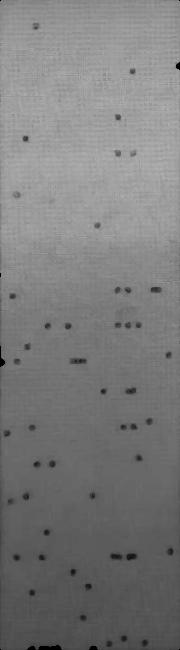


图 3-10 透视变换之后的矩形区域

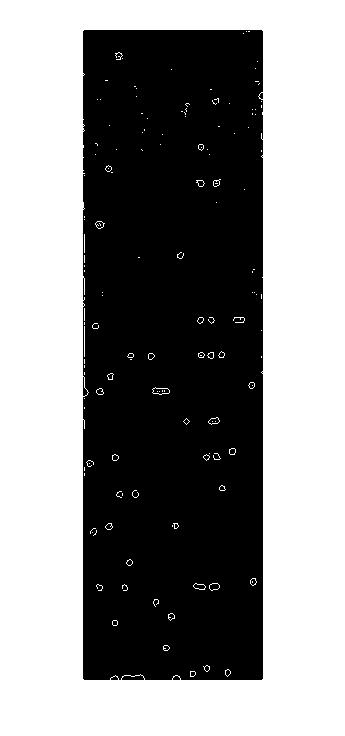
从上图可以看到，在实现了对芯片载板区域的透视变换之后，目前芯片载板区域是一个标准的矩形，且边缘等敏感位置处理效果也较为理想。

3.3 芯片位置的确定

为了实现芯片位置的自动输出，接下来需要将图3-10进行二值化处理，在最终的效果图上实现芯片为黑色像素点集，背景位置为白色像素点集。

由于图像拍照时候光照的影响，图片中像素明暗分布不均，对于整个区域无法找到一个固定的阈值来进行二值化。处于芯片和其周围区域的亮度差别，因此本文仍然考虑通过边缘检测的方式来进行边缘检测。

以下本文分别选用sobel算子和canny算子进行处理，处理效果图分别如下图3-11左图和3-11右图所示：



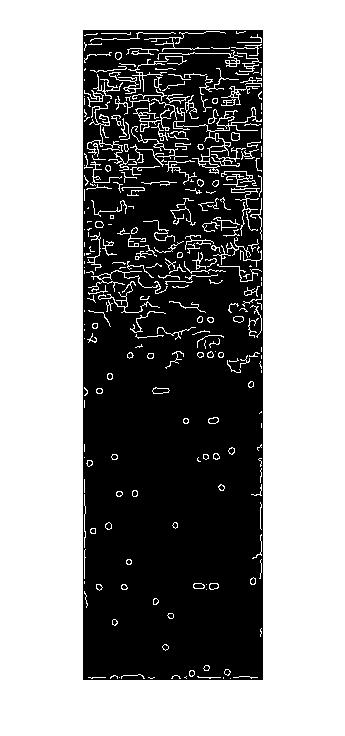


图 3-11 sobel边缘检测的效果图（上）、canny边缘检测的效果图（下）

从图中可以看到，Sobel算子检测方法对灰度渐变和噪声较多的图像处理效果较好，但sobel算子对边缘定位不是很准确，图像的边缘不止一个像素。而如图所示，canny算子不容易受噪声干扰，能够检测到真正的弱边缘，但在此图中，这个特性反而成了缺点，canny算子检测的图像输出了太多的无关的信息。因此本文选择sobel算子边缘检测的结果进行下一步计算。

接下来先对图3-11左进行二值化，阈值采用最大类间方差法获取。是一种自适合于双峰情况的自动求取阈值的方法,又叫大津法,简称Otsu。它是按图像的灰度特性,将图像分成背景和目标两部分。背景和目标之间的类间方差越大,说明构成图像的两部分的差别越大,当部分目标错分为背景或部分背景错分为目标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。

使用最大类间方差法获取的阈值对图像进行二值化之后使用区域生长法填充芯片轮廓。区域生长的基本思想是将具有相似性的像素集合起来构成区域。首先对每个需要分割的区域找出一个种子像素作为生长的七点，然后将种子像素周围邻域中与种子有相同或相似性质的像素（根据事先确定的生长或相似准则来确定）合并到种子像素所在的区域中。而新的像素继续作为种子向四周生长，直到再没有满足条件的像素可以包括进来，一个区域就生长而成了。区域生长法的结果如下图3-12所示

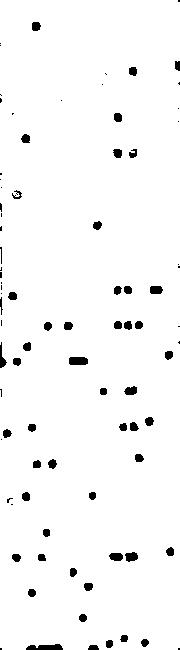


图 3-12 完整芯片载板区域最终处理结果

上图为芯片载板区域的最终处理结果，取得了较为精确的显示效果。但是在左侧边缘位置存在一定误差，在载板上也存在某两个芯片显示不是很清楚，这些可能会对本文下一步的芯片坐标的输出造成误差。

之后对芯片的定位采用按区域遍历，以7×7的方形区域作为遍历的基本单位，即作为最终输出的一个单位“坐标点”，当此单位内的黑色像素点个数超过我们所设的阈值θ时，我们即认为此区域有芯片。

注意，本文中的横坐标和纵坐标指的并不是芯片的像素坐标，而是如下规则，采用实例说明，例如芯片载板左上角如果有芯片则其坐标为（1,1），这个芯片的右边紧邻位置的芯片的坐标为（2,1），左上角芯片下方紧邻位置芯片的坐标为（1,2）以此类推。根据此规则，本文将芯片的坐标值保存在一个矩阵中输出如表格3-1下，

表格 1 完整芯片载板芯片坐标识别结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 纵坐标Y | 横坐标X | 纵坐标Y | 横坐标X | 纵坐标Y | 横坐标X |
| 6 | 4 | 25 | 51 | 25 | 79 |
| 26 | 10 | 1 | 52 | 3 | 80 |
| 20 | 11 | 3 | 52 | 6 | 80 |
| 17 | 17 | 11 | 52 | 17 | 80 |
| 4 | 20 | 12 | 52 | 18 | 80 |
| 17 | 22 | 13 | 52 | 19 | 80 |
| 14 | 33 | 19 | 56 | 20 | 80 |
| 17 | 42 | 20 | 56 | 11 | 82 |
| 18 | 42 | 19 | 57 | 13 | 84 |
| 19 | 42 | 18 | 61 | 5 | 85 |
| 22 | 42 | 22 | 61 | 12 | 89 |
| 23 | 42 | 5 | 62 | 16 | 92 |
| 2 | 43 | 18 | 62 | 18 | 92 |
| 7 | 47 | 20 | 66 | 21 | 92 |
| 10 | 47 | 21 | 66 | 5 | 93 |
| 17 | 47 | 6 | 67 | 7 | 93 |
| 18 | 47 | 8 | 67 | 8 | 93 |
| 19 | 47 | 4 | 71 | 9 | 93 |
| 20 | 47 | 14 | 71 | 14 | 93 |
| 4 | 50 | 4 | 72 | 16 | 93 |
| 5 | 50 | 7 | 77 |  |  |

3.4 结果分析

由于原始芯片图像质量过差，无法通过观察得到芯片的真实坐标，只能通过原始的图像和识别得到的图像通过观察进行大致比对，为了方便对结果进行评估，本文将上面表3-1中的坐标信息显示在图片上，如下图3-13所示

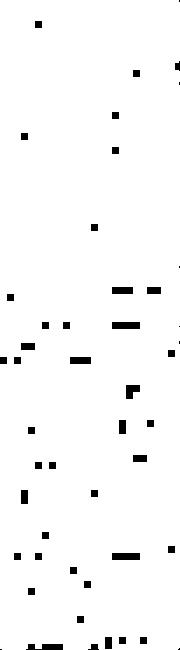


图 3-13 芯片识别结果

实际上原图中芯片载板区域共有59枚芯片，如图3-1所示，本文将图3-1与图3-13的芯片识别结果进行对比。之后发现输出结果中有8个点属于在原图中没有，在最终输出效果中有，属于多输出误差，如下图3-14所示

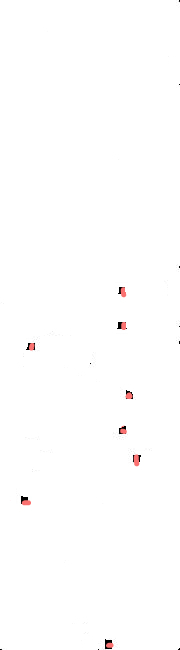


图 3-14 多输出芯片

接下来将多输出的芯片坐标进行统计，统计结果如表2所示

表格 2 多输出的芯片坐标

|  |  |  |  |
| --- | --- | --- | --- |
| 横坐标X | 纵坐标Y | 横坐标X | 纵坐标Y |
| 17 | 42 | 18 | 62 |
| 18 | 47 | 21 | 66 |
| 5 | 50 | 4 | 72 |
| 19 | 57 | 16 | 93 |

实际图片中有五个芯片在最终的识别结果图中并没有得到输出，本文对未输出的芯片进行了标识，如下图3-15所示

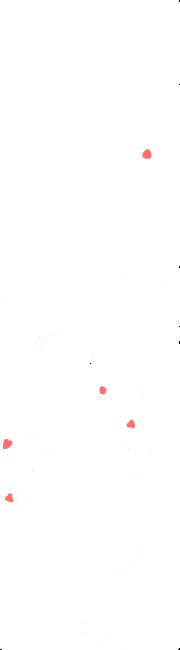


图 3-15 少输出芯片

接下来本文对少输出的芯片的坐标信息进行了统计，少输出的芯片的坐标如下表3所示

|  |  |
| --- | --- |
| 横坐标X | 纵坐标Y |
| 22 | 22 |
| 15 | 56 |
| 19 | 61 |
| 2 | 64 |
| 2 | 72 |

本文定义识别率和误识别率的公式如下：

26)

27)

原始图中芯片数量为59个，本文正确识别的数量为54个，则由公式26可计算得到识别率为91.525%，而本文误识别的芯片数量为8个，则由公式27可得误识别率为13.559%。

以上数据的准确性十分有限，由于没有芯片的真实坐标，只是通过原始的图像和识别得到的图像通过观察进行大致比对，因此识别率一定相比实际偏高，误识别率一定相比实际偏低，而且偏差应该较大。以上数据只是在图像质量较差的情况下的一个大致的对比和说明。

从以上数据可以看出，目前当图片素质较低时的识别效果很不理想。而工业生产需要对各种环境良好的适应，需要鲁棒性强的解决方案，因此仍需要较大的改进方可能应用到工业生产。

3.5 本章小结

本章首先对原始图像进行预处理以消除分水岭算法出现的过度分割问题，之后对预处理后的图像进行分水岭变换得到单独的完整芯片载板区域。我们对不规则的芯片载板区域进行处理，将其转换成规则的四边形，之后对四边形区域进行透视变换，将其映射到矩形区域。使用sobel算子对矩形区域进行边缘检测得到梯度图像之后使用最大类间方差法对梯度图像进行二值化。二值化后我们使用区域生长法对芯片轮廓进行填充，最终我们按固定大小的区域进行遍历确定芯片坐标。

目前来看实际识别效果可以接受，但是距离工业应用精度有较大距离，有很多地方可以加以改进。如拍照的时候的角度和光照可以进行调整，免除不必要的误差；边缘检测算子可以尝试除了sobel和canny之外的更高级的方法等等。

第四章 不完整芯片载板的芯片检测

4.1 图像处理

本章第一节的目标就是将如下图4-1左所示的图片的左半部分完整的芯片载板区域分割出来。下图右边的芯片载板区域照片下方有水印遮挡，故选用左边进行处理，而且事实上两者基本一致，选哪一个进行处理对于最后的识别的结果区别不大。

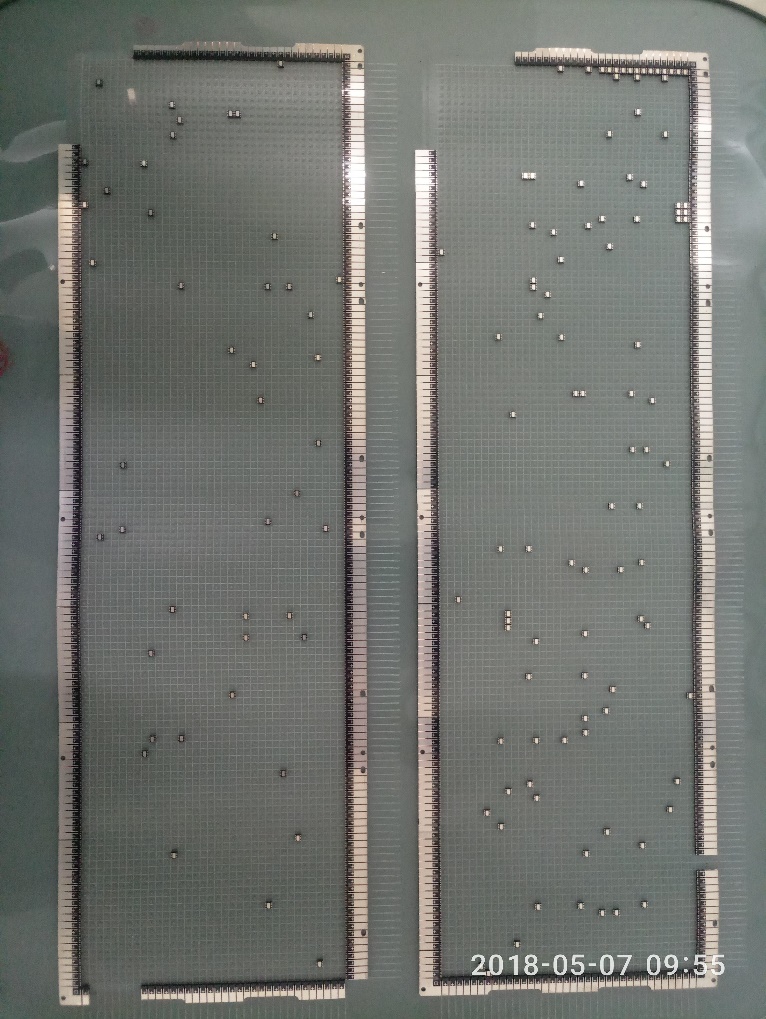


图 4-1 不完整芯片载板图片

对于不完整的芯片载板区域无法使用分水岭算法，因为分水岭算法针对的是闭合的区域分割，对于左图这样左上角和左下有缺口的图像若使用分水岭误差将十分严重，无法进行分割，因此本章考虑使用其它方式来进行处理。

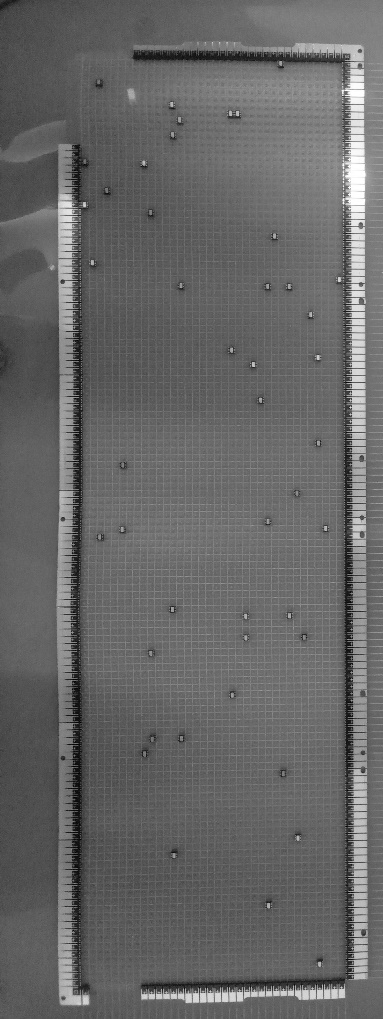


图 4-2 不完整芯片载板图片

首先将图片一分为二只保留左半部分，如上图4-2所示，由于此张图片分辨率太高，并因此导致图片后续处理速度过慢，考虑对图片进行重采样。我们的重采样方法是以上下左右四个点为一个区域，选取其中灰度值最大的点作为采样点，从而将新的图片的长宽分别缩小到原来的1/2。之后对图片进行处理，使用sobel算子进行边缘检测，效果图如下图4-3上所示。之后本文对边缘检测得到的梯度图像通过最大类间方差法进行二值化。

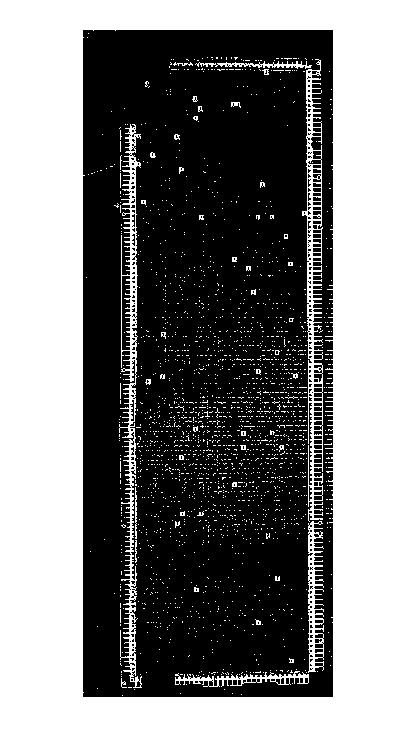
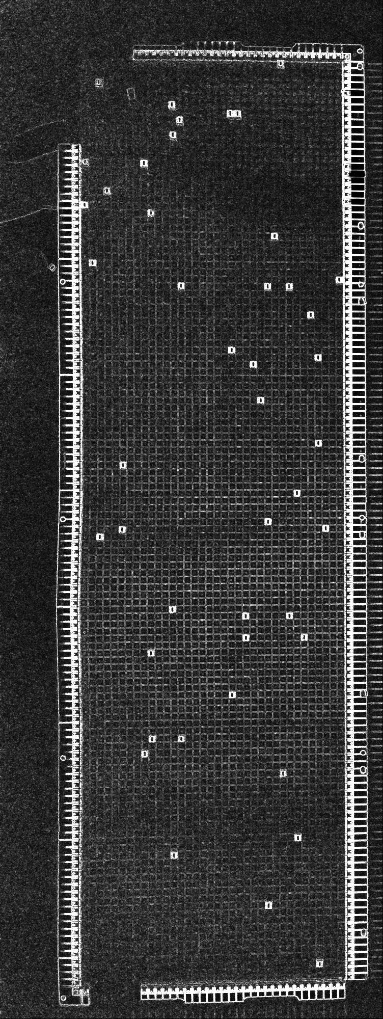
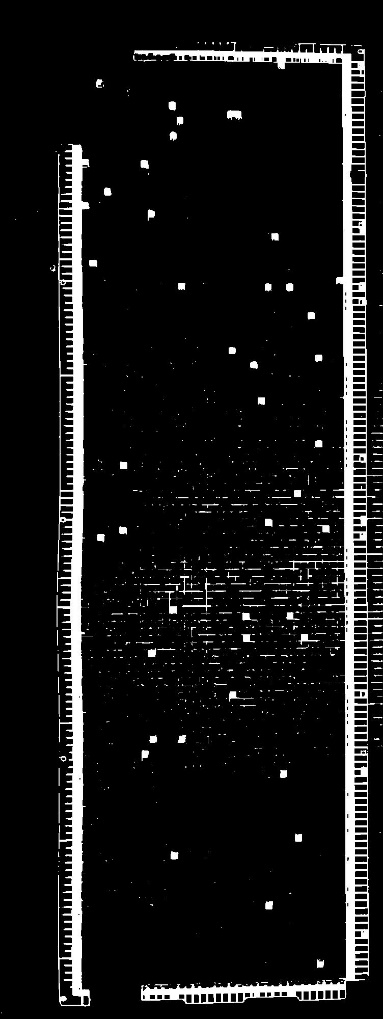


图 4-3 边缘检测结果（上）、二值化结果（下）

二值化的结果如图4-3（下）所示，图像中有很多白色的误差点。对图像进行遍历，将单独的白色像素的误差点消除，结果如下图4-4所示，此时周围连通域内没有其他白色像素点。



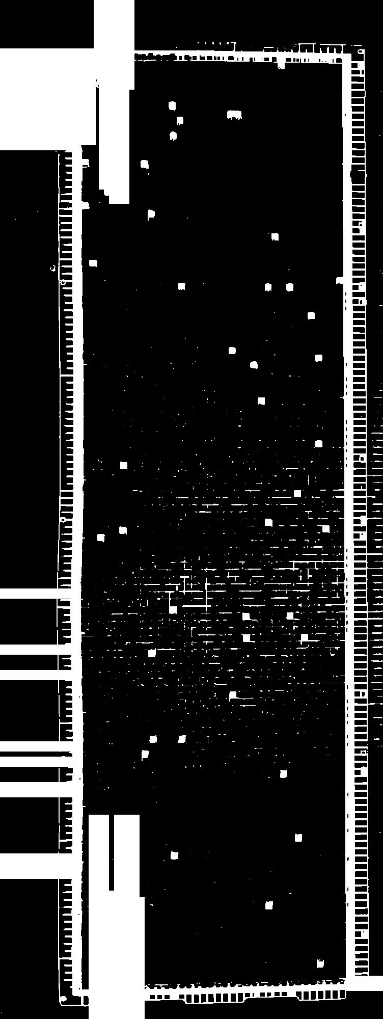
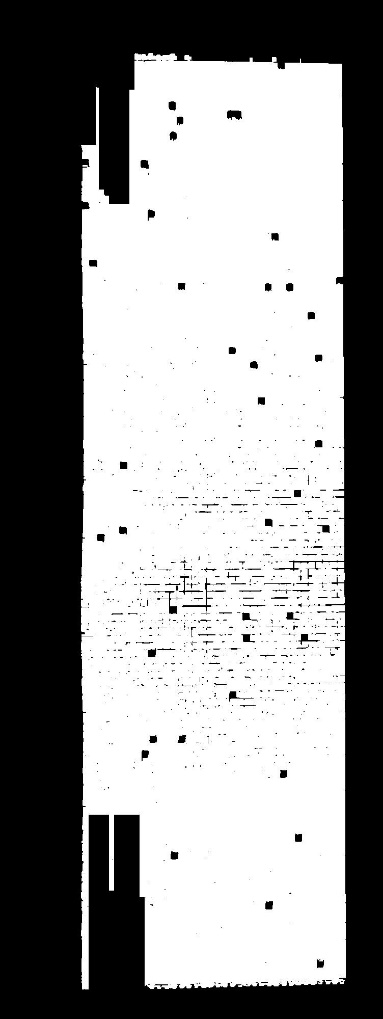


图 4-4 消除部分误差点之后的二值图像（上）、补上缺口的二值图像（下）

由于此芯片载板区域左上角和左下有缺口，考虑需要将其缺口部分补上，本文的修补思路是寻找X方向白色像素点数量超过阈值的区域将其继续向左延伸，Y方向白色像素点数量超过阈值的区域将其继续向上延伸，即可将缺口堵住。修补效果如图4-4下所示。

接下来本文使用区域生长法对图像进行处理，填补图中空缺，之后利用区域生长之后的图像（图4-5上）就可以寻找四边形的四个顶点的位置为透视变换创造条件。但是此处也与完整区域的芯片检测不同，本章对四边形区域顶点的定位不能再简单的从位置特征去进行定位。在此处本章直接采用对边缘进行拟合（在拟合之前我们对坏值进行剔除）然后选取两两直线的角点作为四边形区域的顶点。之后对前文边缘检测得到的结果（如图4-3上所示）进行透视变换将其转化为矩形，结果如图4-5下所示。



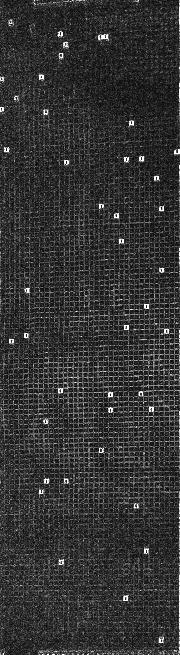


图 4-5 区域生长后的二值图像（上）、透视变换后的梯度图像（下）

随后继续对透视变换之后的梯度图像使用边缘检测算子canny再度进行处理，并于之后对结果进行二值化，得到的二值图像如下图4-6所示

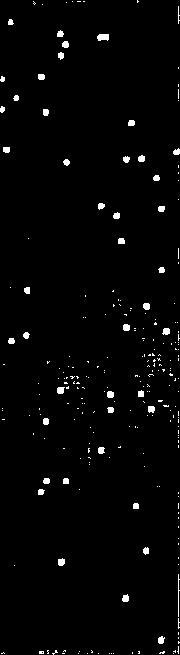


图 4-6芯片载板区域最终处理结果

4.2 芯片位置的确定和分析

图4-6为芯片载板区域的最终处理结果，取得了精确的显示效果。虽然图中有一些微小的白色误差点，但事实上不影响本章对芯片的定位识别。

对芯片的定位采用按区域（block）遍历，本文以5×5的方形区域作为遍历的基本单位，即作为最终输出的一个单位“坐标点”，当此单位内的黑色像素点个数超过本文所设的阈值θ时，即认为此区域有芯片。

由于本次不完整芯片载板区域的原始图像较为清楚，可以准确的得到芯片的真实坐标，因此更有利于本文的结果分析。

接下来将芯片的真实坐标值和识别得到的芯片坐标输出如表格3所示，此处坐标的定义和前文第三章的定义相同，之后将完全准确的输出结果标识为1，在表格的第五列输出：

表格 3 不完整芯片载板识别结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 真实横坐标X | 真实纵坐标Y | 识别横坐标X | 识别纵坐标Y | 是否正确 |
| 3 | 4 | 3 | 4 | 1 |
| 13 | 6 | 13 | 7 | 0 |
| 13 | 7 | 13 | 7 | 1 |
| 21 | 7 | 21 | 8 | 0 |
| 22 | 7 | 22 | 8 | 0 |
| 14 | 9 | 14 | 9 | 1 |
| 13 | 11 | 13 | 11 | 1 |
| 9 | 15 | 9 | 15 | 1 |
| 4 | 19 | 4 | 19 | 1 |
| 10 | 22 | 10 | 22 | 1 |
| 27 | 24 | 27 | 25 | 0 |
| 2 | 29 | 2 | 29 | 1 |
| 36 | 30 | 36 | 31 | 0 |
| 14 | 32 | 14 | 32 | 1 |
| 26 | 32 | 26 | 32 | 1 |
| 32 | 35 | 32 | 36 | 0 |
| 21 | 41 | 21 | 41 | 1 |
| 33 | 41 | 33 | 42 | 0 |
| 33 | 42 | 33 | 42 | 1 |
| 24 | 43 | 24 | 43 | 1 |
| 25 | 48 | 25 | 48 | 1 |
| 33 | 54 | 33 | 54 | 1 |
| 6 | 58 | 6 | 57 | 0 |
| 30 | 61 | 30 | 61 | 1 |
| 26 | 65 | 26 | 65 | 1 |
| 34 | 66 | 34 | 67 | 0 |
| 6 | 67 | 6 | 67 | 1 |
| 3 | 68 | 3 | 68 | 1 |
| 13 | 78 | 13 | 78 | 1 |
| 29 | 78 | 29 | 79 | 0 |
| 23 | 79 | 23 | 79 | 1 |
| 29 | 79 | 29 | 79 | 1 |
| 31 | 81 | 31 | 82 | 0 |
| 23 | 82 | 23 | 82 | 1 |
| 31 | 82 | 31 | 82 | 1 |
| 10 | 84 | 10 | 84 | 1 |
| 21 | 90 | 21 | 90 | 1 |
| 10 | 96 | 10 | 96 | 1 |
| 14 | 96 | 14 | 96 | 1 |
| 9 | 98 | 9 | 98 | 1 |
| 28 | 101 | 27 | 101 | 0 |
| 30 | 110 | 29 | 110 | 0 |
| 13 | 112 | 13 | 112 | 1 |
| 26 | 119 | 25 | 119 | 0 |
| 33 | 128 | 33 | 128 | 1 |

原图上共有芯片46个，此处识别得到了46个芯片，其中坐标完全正确的有33个，坐标不完全正确的有13个，由前文定义的公式26可计算得到得识别率71.739%，由公式27计算可得误识别率为28.261%。

看似识别率比第三章的完整芯片载板区域的芯片位置检测的结果要差，其实不然，之前第三章由于图像质量较差，无法得到芯片的真实坐标。本文第三章识别结果的比对只是通过对原图和结果图的大致比对得到。因此和实际识别率偏差极大。事实上图片质量的提升对芯片位置的识别效果带来的提升是显著的，从数据明显可以看到得到的芯片坐标位置和原本的位置基本一致，大多数错误的坐标也只有1格的误差。而且这个误差很有可能是图像质量问题造成的，从最初的图边缘部分的折痕可以看到图片的拍摄存在一定问题，板子没有完全放平，因此在透视变换之后不可避免的存在误差。事实上，本文对不完整芯片载板进行芯片坐标检测的误差属于可以接受的范围内，在后续的工业生产中可以考虑通过严格规范芯片载板的图片拍摄等措施来进行消除。

4.3 本章小结

本章对不完整区域采用了全新的处理思路，先是对不规则的芯片载板区域使用sobel算子进行边缘检测，之后对梯度图像进行二值化，随后将二值化的图像中的一部分白色像素的误差点消除。紧接着将芯片载板区域的左上角和左下角缺口部分补上，随后使用区域生长法对补好的图像进行处理，在得到基本完整的图像之后接下来我们就可以使用边缘拟合的方法对边缘进行拟合，而边缘相交的位置即为四边形区域的顶点。得到四边形顶点之后对第四章一开始边缘检测得到的梯度图像进行透视变换将其投影到矩形区域。之后对得到的结果本文使用canny算子进行边缘检测，紧接着对其进行二值化即得到不完整芯片载板的最终识别结果。

目前来看实际识别效果较为理想，但是仍存在一定误差，在后续的工业生产中可以通过严格规范芯片载板的图片拍摄或是采用其它消除误差的办法来改善检测结果。