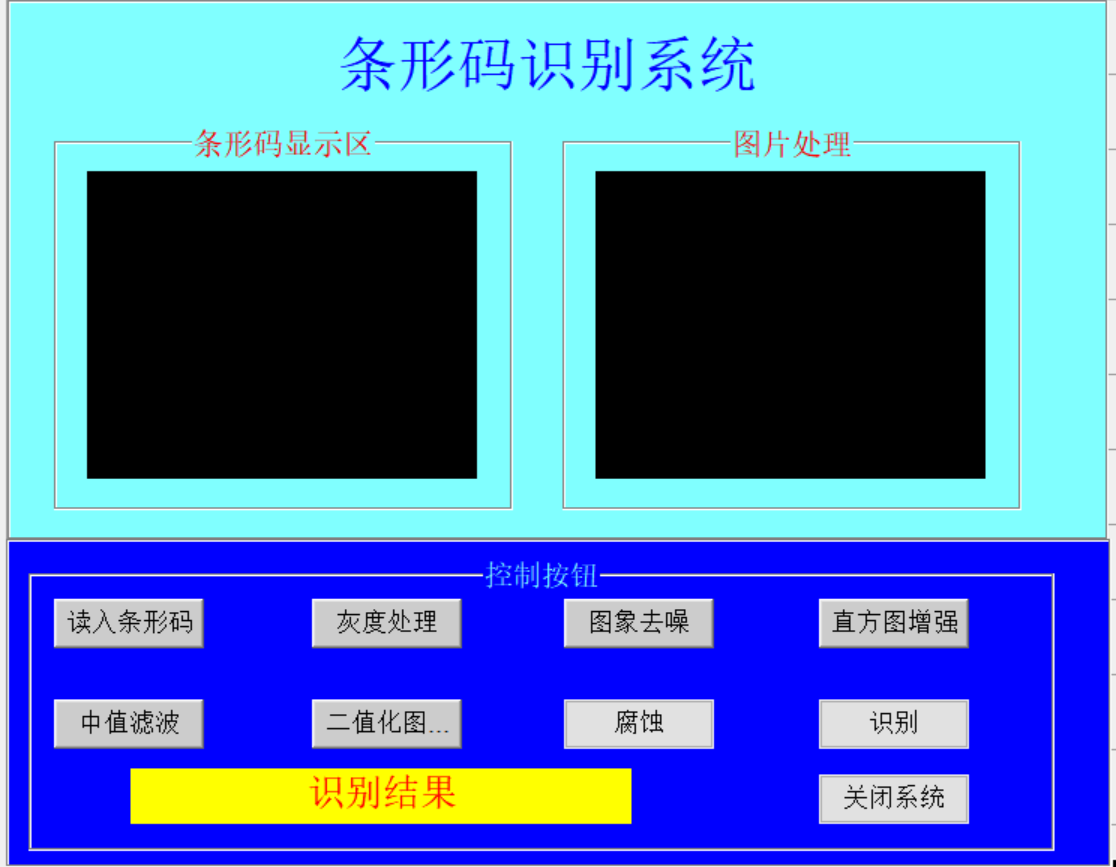
## 源码：https://download.csdn.net/download/weixin\_38719187/12678960

## 1.1 应用MATLAB识别条形码总体设计

本设计为基于MATLAB的条形码识别系统，带有一个GUI界面。



### 1.1.1 程序总体设计思路

在上节中我们简单的介绍了MATLAB的发展以及优点。在以后的章节中将阐述如何应用MATLAB编程实现对条形码的识别。在这一节中将主要介绍编程的总体思路。

首先将含有条形码信息的图片读入到MATLAB中去，这个图片将以矩阵的形式存储在MATLAB中。这样，我们就能通过对矩阵的运算来实现对图片的处理。由于条形码一般都是黑白的，这样我们的第一步就是要把图片行二值化处理，用于去除那些不需要的细节。然后就要确定条形码所在的位置。这一步至关重要，如果定位不准确，我们就不能获取条形码所有的信息，以至于不能准确的识别。找到条形码准确位置后。就要对条形码信息提取，然后在按照比例将提取到的条形码信息转换成标准模块组成的条形码。最后通过相应的译码得到条形码中的数字，判别是否正确后输出，这样就完成了条形码识别的整个过程。



### 图像处理的结果

### 1.1.2 程序总体设计流程图

主程序的流程图可以用图4-1来表示。

效验

输出条码

结束

转换成标准模块的条码

读入条码中的第n行

提取条码信息

译码

开始

读入图片

二值化处理

定位条形码

n加1

图4-1主程序流程图

## 1.3 各部分子程序说明

上一节中我们分析了条形码识别的主体设计思路，在本节中我们将介绍程序各部分的作用以及实现方法。

### 1.3.1 读入图片并二值化处理

在这个步骤中，我们的主要任务是把含有条形码信息的图片文件放到MATLAB的工作区中去。当图片经过imread命令后，MATLAB会给图片开辟一个存储空间，把图片各像素点的值储存到一个矩阵中。读入图片格式一般都是 .jpg, . bmp，等格式，这样格式的图片每一个点都有其相应的R、G、B值，这不仅增大了MATLAB的运算量，而且还大大降低了识别的准确性。

这就要求我们先对图像进行二值化处理。因为条形码在正常情况下是黑白的。这种方法可以去除与条形码不相关的信息，这些信息对条形码的识别是不起作用的。这样既可以提高效率，又可以提高准确性。

图4-2和图4-3表示了二值化前后两个图像的对比。通过对比我们发现，二值化后图像更加简洁，我们需要的条形码符号更加清晰，容易进行识别。这也为我们以后的识别工作做了很好的铺垫。

### 1.3.2条形码的定位

上一个过程结束后，存储在MATLAB中的信息只有‘0’和‘1’，其中图像中黑色的部分用‘0’表示，白色部分用‘1’表示。



图4-2 二值化前的图片 图4-3 二值化后的图片

所谓条形码的定位就是找到条形码的确切位置，这样我们才能够到相应位置去提取条形码信息。由于条形码的前后都有9个模块的空白区，这些知识我们在第二章中我们已经做了详细的介绍。所以我们可以通过条形码前面的空白区来定位条形码，如果我们找到条形码前面空白区也就完成了对条形码的定位。

假设我们定义条形码中每个模块所占的像素数为4，那么条形码空白区在水平方向上总共占的像素数为36。在竖直方向上每个条形码的长度都是不同的，但大约都是水平宽度的4~5倍。考虑到边缘的噪声问题。我们选取了一个120×30的单位矩阵作为空白区域的特征矩阵。接下来，对整幅图像进行逐点扫描。扫描到一点时，以这一点为原点与特征矩阵比较，如果相等，程序则返回该点的值，这一点就是条形码左上角的点，如果不相等，继续扫描下一点。其主要的程序流程图如图4-4所示。经过提取条形码如图4-5所示条形码被置于整个图片左上方。

开始

定义空白区特征矩阵

逐点进行

扫描

是否与特征矩阵相同

返回该点值

Y

N

图4-4 定位条形码流程图



### 

图4-5 提取后的条形码

### 1.3.3条形码信息的提取

找到条形码的起始坐标后，我们就能对条形码的信息进行提取了。首先以条形码的起始坐标为依据提取一行含有条形码信息的数据。通过对起始符，结束符，以及左右空白区的判断，我们可以准确的找到条形码中间的数据区域。接下来我们将对条形码的数据区域进行进一步的处理，把条形码转换成标准模块的条形码，也就是每个模块都是有‘0’或‘1’表示。

### 1.3.4条形码单元的判断

条形码的单元指的是我们看到的条形码中的条和空，每一个单元都是有1~4个模块组成的。本步的主要工作是判断条形码中每个单元的数据(0或1)和其所占的像素数。其主要程序的流程图如图4-6所示。

### 1.3.5条形码模块的识别

模块是组成条形码最基本的单位，在识别的过程中识别出每个模块的值我们就能对条形码进行判别了。上一步我们得到了条形码每个单元的数据以及宽度(所占像素数)，把所有单元的像素数相加就可以得出整个条形码所占的总像素数了。在第二章中，我们在介绍了EAN-13码的基本构成，不包括条形码的左右空白区总空有95个模块组成。这样就可以知道每个模块所占的像素数了。进而可以计算出每个单元的模块数。主要程序如下：

N

Y

N

Y

开始

输入含有条形码信息的一行C

i是否小于最大点

该点c（i）值是否与下点相同

单元长度m加1

m重新记录

指针j加1

Barlong（j）=m

Bardata（j）=c（i）

结束

图4-6 单元判别流程图

完成上述两个步骤后我们已经将各个模块单元化了。也就是说每个模块都是由一个单位的0或1表示的，转换成了条形码的逻辑值。经过处理后条形码的图像如图4-7所示。我们可以把这幅图像与图4-5做一比较，条形码明显清晰。

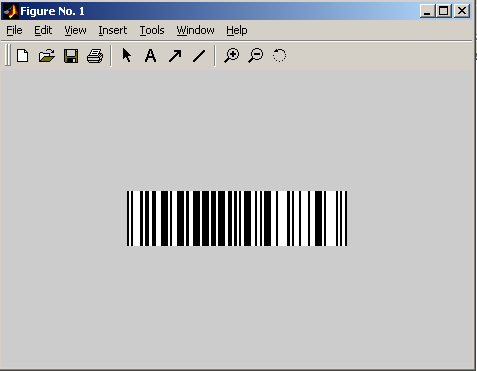


图4-7 经过整形后单位模块的条形码

### 1.3.6条形码数字判别

到现在为止，我们已经将条形码的图片信息转换成了二进制的数组。按照条形码的编码规则，整个数组包括起始符，结束符，中间分隔符，数据信息总共95位，前3位是起始符(010)，4~46位为左侧数据，46~51位为中间分隔符(01010)，51~92位为右侧数据，93~95位为结束符(010)。

程序清单：

j=1, k=1

for i=4:51 %从第四位开始跳过了起始符

evrybar(k ,j)=bar(i) %每7位数字存到一行里

j=j+1

if j==8 %判断是否存储了7位数字

k=k+1

j=1

end

end

j=1, k=7

for i=51:92 %从第51位开始跳过了中间分隔符

evrybar(k ,j)=bar(i)

j=j+1

if j==8

k=k+1

j=1

end

end

这样我们就能把每一个数字的逻辑值存到数组evrybar中。查看evrybar中的截图为图4-8。

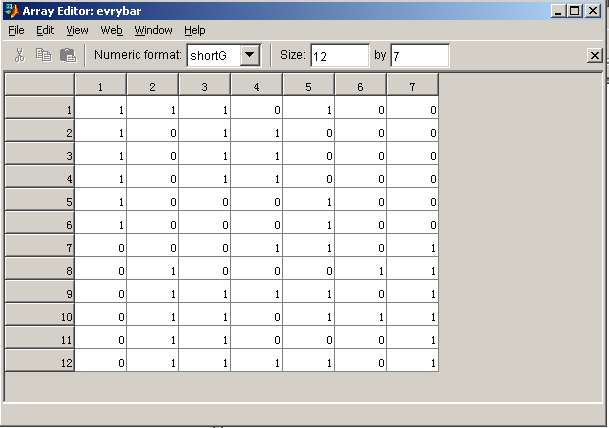


图4-8 数组evrybar存储的数据

数组evrybar中每一行表示一位数字的逻辑值，例如第一位数字的逻辑值为1110100。接下来的任务就是判别数字了，取出数组evrybar中的一行，与数据库中的数据做比较，相同时返回相应的数字。注意在判别左侧数字时，因为左侧数字存在A、B两种编码方式，所以还要创建一个存储编码方式的数组barab。如果A类编码把数组barab相应的位置设置为1，B类编码则设置为0。这样我们就能通过数组barab来判断前置字符了。相应程序见附录。

### 1.3.7校验输出结果

从代码位置序号2开始，所有偶数位的数字代码求和为a。 将上步中的a乘以3为a。从代码位置序号3开始，所有奇数位的数字代码求和为b。将a和b相加为c。取c的个位数d。用10减去d即为校验位数值。

例：234235654652的校验码的计算如表4-1。

表4-1 校验码计算示意表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 代码位置序号 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 数字码 | 2 | 3 | 4 | 2 | 3 | 5 | 6 | 5 | 4 | 6 | 5 | 2 | × |

步骤1：3+2+5+5+6+2=23

步骤2：23\*3=69

步骤3：2+4+3+5+4+5=23

步骤4：69+23=92

步骤5：10-2=8

步骤6：校验码为8

## 1.4 主程序执行结果

图4-9是程序运行后输出的结果，与条形码上的数字相同。准确判断出了条形码的数字，达到了预期的效果。

