# PaperTableRecognition项目开题报告

摘 要

工业上很多场合需要纸质文档记录生产数据，然后再通过人工手动录入电子表格。虽然现在都向自动化生产，但是纸质记录还是必不可少的手段。纸质记录数据有很多优势，灵活性高，方便小量数据采集，可以针对个体用户，不需要系统集成。

PaperTableRecognition项目和其他项目有很多共同技术，比如车牌识别，电子抄表，OCR等。所以该项目不仅可以解决实际工业应用问题，还可以研究相关技术。

下面举例说明应用实例，某标准马达有限公司，研发人员需要对新设计的马达做很多实验，操作人员需要记录实验数据，然后再把记录的纸档实验数据交由研发工程师，然后由工程师助理将纸档实验数据手工录入电子档，然后再提交给工程师。有的时候，录入电子档的工作是由工程师自己完成的，需要耗费相当的时间成本。

关键技术：OpenCV,Tesseract-OCR,Machine learning,ImageProcess,ZBar;

## 方案设计

该系统的使用是针对于程式化的纸质表格，也就是说，待识别的表格是预先在系统中有记录的，系统预先知道该文档的存在，在表格的左上角有一个文档编号，文档编号的形式可以是条形码，二维码，数字序列等。使用的时候，将纸档表格扫描成图片，系统识别到文档编号后，系统就可以获知该表格各个单元格所在位置，数据形式等信息，系统可以针对性的对ROI进行识别。同时，系统在识别到文档编号后，会根据文档的类型，创建一个表格文档（EXCEL），在对ROI识别的数据会填入到EXCEL相应ROI所在位置。从而实现纸档表格的数据识别。系统对ROI的识别采用OpenCV，对字符的识别采用Tesseract-OCR，界面开发可采用MFC等。

这种方案重点是识别程式化的表格，针对ROI进行字符识别并填入相应的EXCEL中，可有限提高识别准确率和正确率。

### 程式化表格设计

表格的图像元素主要有文档编号（条形码或者二维码），单元格，边框，单元格内的说明文字。如下图。



其中文档编号最好以条形码或者二维码的方式存在，目的是为了在图像识别时，可以自动识别到原图中唯一的条形码，从而可以确认该文档的类型。如果仅仅是数字序列，系统会遇到2个问题：1.系统无法锁定哪一个数字序列才是文档编号；2.如果固定数字序列所在图片上的坐标位置，貌似可以解决1中的问题，但是系统如何精确定位数字序列的坐标位置是一个难题。

这里需要说明的是，虽然Office具有添加条形码的功能，但是添加后会启用宏，导致不能正常打开文档，条形码不可视等问题。通过实践证明，在制作程式化表格时，可以用第三方软件生成条形码，然后截图插入到文档中。

文档必须要一个边框，这个边框用来识别图片整体坐标，来匹配上面条形码指定的表格的坐标尺寸，这样才能确保图片中的表格可以完全映射到指定的表格中。我把这个边框命名为定位边框，定位边框越大越好，误差越小，定位越精确。所以我把文档最外围的边框作为定位边框。

空白文档中的文字信息是不需要识别的，因为通过文档编号条形码和定位边框，已经可以精确确定图片中需要识别的区域。并且知道识别的ROI内容将要填到哪一个单元格内。

制作完成后，文件以条形码序列号命名，并存储在软件系统中。

### 程式化表格配置文件

设计好程式化表格后，系统可以通过识别条形码来确定文档的编号，但是仅仅是锁定了表格编号还是不够的。系统还需要知道程式化表格定位边框的尺寸坐标以及每一个单元格的尺寸坐标，然后才能实现图片和表格的坐标映射，也就是Image\_size和Sheet\_size的坐标映射。其中Image\_size是识别的表格图片定位边框的尺寸，Sheet\_size是程式化表格定位边框的尺寸，我们定义一个比例参数Ips，定义如下：

Image\_size是可以自动识别的，但是其他单元格的坐标尺寸是很难识别地，即表示是识别到了，在没有《程式化表格配置文件》的条件下，也是无法和程式化表格单元格一一映射的。下面先给出坐标映射的方法，然后给出《程式化表格配置文件》的具体形式。Sheet\_Cell\_1是程式化表格第一个单元格的坐标尺寸，ROI\_Cell\_1是表格图片中第一个单元格的ROI，其中：

系统得到ROI\_Cell\_1后，就截取ROI\_Cell\_1的ROI，然后识别其中的数据，识别后填入Sheet\_Cell\_1的单元格，到此为止一个单元格识别完成。以此类推，识别Sheet\_Cell\_2，Sheet\_Cell\_X。

《程式化表格配置文件》的具体表现形式可以很多，比如.log，.ini，.xml文件，这里我们采用.xml文件格式，因为它是应用程序交换数据和传输数据最佳的选择，具体相关.xml文件格式的说明，这里不在赘述。《程式化表格配置文件》和《程式化表格》文件名保持一致，只是后缀不一样，下面给出《程式化表格配置文件》具体内容样式表如下：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <sheet>  <fileName>1521611168303.xml</fileName>    <border>  <x>0</x>  <y>0</y>  <width>888</width>  <height>420</height>  </border>    <cells>  <cell\_1>  <x>0</x>  <y>81</y>  <width>148</width>  <height>34</height>  <cellNum>A4</cellNum>  </cell\_1>  <cell\_2>  <x>148</x>  <y>81</y>  <width>148</width>  <height>34</height>  <cellNum>B4</cellNum>  </cell\_2>  </cells>  </sheet> |

配置文件尽量详细地描述了程式化表格，通过配置文件可以精确定位每个单元格。

### 识别文档编号

每一个程式化文档都有且仅有一个文档编号，文档编号用来识别该文档的样式，在系统中该编号是唯一的，相当于个人身份证。系统首先扫描图片中的条形码，可以使用基于ZBar的条形码识别，只要图片中存在条形码即可识别。识别结果是一个数字序列号，识别后立即从系统数据库中查询该序列号的文档，然后从数据库中复制一份文档到指定目录下，作为识别生成的空白文档。

识别文档主要过程如下，OpenCV读取并简单处理原始图片，ZBar识别图像中的条形码得到序列号，根据序列号从系统中找到对应的文档，找到后将系统中的文档复制到用户数据目录下，重命名该文档。

### 识别定位边框

定位边框实际就是一个文档的最外边框，作用只要有2个，一是用于确定文档的方向，二是确定文档图片的实际尺寸和坐标。以上2个确定条件的目的是为了和系统数据库中的程式化文档进行约束映射，从而精确确定图片中的任意一个ROI对应的单元格。

识别边框主要步骤如下，首先用OpenCV对图片简单处理，比如去噪，形态学处理，阈值化，然后利用cvApproxPoly()，以及cvFindContours()等方法进行边框轮廓检测。实践验证方法可行，实际使用时需要调整相关的参数。最终可以得到定位边框4个点的坐标，可用于后续的坐标映射。

### 读取配置文件

识别出条形码之后，便可以找到该文档的配置文件。配置文件是XML文件，所以我们采用as-is提供的TinyXML解析器来解析XML文件，TinyXML小巧开源，使用简单方便，关于TinyXML详细说明，这里不在赘述。

C++读取XML文件分为2个层次：

1. 读取XML节点元素，这部分工作由TinyXML解析器来完成，TinyXML解析得到的是节点元素字符串，而我们通常还需要要使用元素数据，所以我们需要解析节点元素。
2. 解析节点元素，也就是将XML数据封装成C++识别的数据。要实现这一点，我们要明白XML文件数据结构的语义，然后我们在C++中定义一样的数据结构来保存XML数据即可。

根据前文提出的《程式化表格配置文件》样式表，我们需要的就是定位边框尺寸Sheet\_Size和单元格坐标尺寸Sheet\_Cell\_X两种数据，所以我们定义两个结构体来存储这两种数据，其中Sheet\_Cell\_X采用可变向量Vector来存储。

设计好结构体后，我们可以遍历XML树形节点，树形节点主要分为2种Element和Text，我们在设计《程式化表格配置文件》时，只有最底层的数据是Text类型的节点，其他节点均是Element类型，在遍历到底层数据时，层层查询父亲节点，从而确认该底层数据是归属哪一个父亲节点，最终将底层数据存储在结构体中。

### 坐标映射

有了定位边框尺寸Image\_Size和Sheet\_Size可以得到Ips（图表比），详细描述参见程式化表格配置文件。从Image\_Size中可以得到偏置量offSetX和offSetY，其中：

通常我们会将Sheet\_Size.x和Sheet\_Size.y设为0。

加上偏置量后，我们重新计算ROI\_Cell\_X：

其中Ips具体算法公式如下：

通过坐标映射，我们可以得到每一个单元格的ROI，通过Tesseract-OCR识别后，得到的识别结果会填入相应的Sheet\_Cell\_x.cellNum，到此为止，完成了坐标映射。

### ROI识别

在提取ROI之前，需要将原图中的单元格边框擦除，否则提取到的ROI可能会包含单元格边框，从而影响ROI识别。另一种方案是将ROI认为的缩小，从而避免包含到单元格边框，但是由于坐标映射会存在误差，所以这种方式并不能够彻底的解决上面的问题。

擦除原图中单元格边框，采用OpenCV自带的累计概率霍夫变换HoughLinesP()函数，它可以找到原图中的直线，并返回直线矢量Lines。然后我们将Lines绘制成背景色，即实现了单元格边框的擦数。