上海大学

SHANGHAI UNIVERSITY

毕业设计（论文）

UNDERGRADUATE PROJECT (THESIS)

装订线

题 目： 车牌识别系统设计与实现

学 院 计算机工程与科学

专 业 计算机科学与技术

学 号 1*1*****

学生姓名 * * *

指导教师 宋 波

起讫日期 20**.02.** –20**.0*.0*

目　　录

车牌识别系统设计与实现

装订线

摘要

作为智能交通系统的核心技术，车牌识别系统是交通管理系统的重要课题，由于中国车牌的特殊性，在实现车牌识别的同时如何提高识别率和适用范围成为车牌识别系统的重要研究目标。论文对车牌识别系统中车牌定位、车牌分割和车牌识别三个模块进行了分析研究和优化改进。针对传统车牌定位模块所用的边缘检测算法缺乏识别率和适用范围低等缺点，论文采用将颜色匹配算法与边缘检测算法相结合的方式增加了车牌定位的效率，在此基础上结合人工智能的机器学习的方法可有效提高了识别率；鉴于字符分割中中文字符的非连通特性，在研究了中国车牌字符特性基础上，通过省市简称字符的位置分割定位来提高识别的准确率；针对传统字符识别的缺乏补充性，通过加载模板和模板补充的方式提高了模板容量以及字符识别率。通过优化和改进，系统提高了车牌识别的准确率和效率并且能够恢复图像角度偏转，使得系统可识别在监控摄像和移动设备拍摄的车牌图像。

关键词：车牌识别；机器学习；模式匹配；形态学；边缘检测

ABSTRACT

As the core of intelligent transportation systems technologies, license plate recognition systems is an important subject. Due to the characteristics of China plates, how to improve recognition rates and scope are important research targets for a license plate recognition system while implementing vehicle license plate.This thesis studies and analysis vehicle license plate recognition system in vehicle license plate location and license plate segmentation, license plate recognition three modules` implementation and optimization. To solve the problem of low recognition rate and scope caused by edge detection algorithm only used in traditional license plate location module, this thesis studies and implement the color matching algorithm combined with edge detection algorithm to improve the efficiency of vehicle license plate location and uses machine learning to improve recognition accuracy; To solve the non-connectivity characteristics of Chinese characters in the character segmentation, the thesis studies using a particular character position to reverse position of Chinese character, according to the characteristics of Chinese plates, so as to improve the segmentation accuracy; To solve the traditional character recognition`s lack of complementarity, the thesis studies to increase template capacity and character recognition rate through loading the template and the template extension.Through optimizing and improving, system improves the vehicle license plate recognition accuracy and efficiency and able to restore the angle deflection of an image, which enables system to identify image taken in surveillance cameras, and mobile devices.

Keywords: License Plate Recognition; Machine Learning; Pattern-matching; Morphology; Edge Detection

装 订 线

第一章 绪论

在人工智能和大数据时代的信息技术潮流下，智能化交通管理系统已经成为了当下交通管理的主要方式。作为其核心课题，车牌识别系统技术直接关系到智能化交通管理系统的效率和成本[1]。

第一节 课题研究背景及意义

随着社会经济的不断发展，汽车数量不断增加，道路交通问题也随之增多，早期的由交通员站岗巡查记录违规车辆的方式早已无法适应现在的网络大数据时代,并且准确性和效率都较低。智能交通系统也就应运而生，该系统能够智能及时记录交通状况，不仅可减少人工成本，同时能提高解决交通问题的效率，缓解交通压力。

解决交通管理中存在的一系列问题都需要以获得车牌为前提，因为车牌号码是确定汽车身份的唯一标识[2]，也是智能化交通管理系统的信息来源和直接处理对象，所以如何快速、准确地识别车牌是智能化交通系统研究的热点问题[3]。

因此，作为智能交通系统中重要环节的车牌识别系统，是实现交通管理智能化的关键因素。该技术通过计算机和外设辅助设备对车辆进行识别，可不用人工进行识别记录，能提高公共交通利用效率以及对快速移动车辆的识别正确性，尤其是针对车辆交通问题，可提高道路交通稽查的全面性和准确率，无需人工站岗查处。该设备能通过对行车车牌进行自动拍摄，提取车牌信息进行识别查询，从而完成对车辆、交通的智能化、自动化管理。

车牌识别系统具有广泛的应用范围：

（1）收费口车辆管理系统：随着车辆的增多，道路交通压力越来越大，如何提高收费口效率，提高车辆流通率是缓解交通压力的重要课题，车牌识别系统能够在车辆驶入收费口时由外设捕捉车辆图片信息后，提取车牌信息，从而减少人工处理时间，帮助实现不停车收费。

（2）道路交通事故管理系统：随着道路的扩展以及车辆的增多，交通事故越来越多，为了能够实时了解道路状况交通摄像头也随之增多，车牌识别系统可以即时根据视频截图获取车牌信息方便交通事故处理，免去了人工站岗以及单个人工简单交通事故处理的成本。轻交通事故可以配合管理系统根据车牌信息直接开罚单并完成罚款工作。

（3）道路监控系统：为了方便交通管理，需要获取交通大数据信息，人工处理工程量大容易出错，车牌识别系统可以结合交通摄像头获取车流量、特殊车牌车辆监控、临时车牌数量等信息，不仅提高了效率也减少了成本。

第二节 国内外研究现状

作为智能交通系统核心地位的车牌识别技术，融合了图像处理、人工智能、机器学习等多门学科，是现代智能交通系统中很重要的一项研究课题，吸引了国内外众多科研所的关注[4]，国外虽然有较成熟的技术，但是由于我国车牌含有汉字，且具有不同汽车的车牌尺寸、颜色均不相同等特征，车牌识别技术识别率仍只有 97%[2]，在恶劣的环境下识别率还会降低。目前国内车牌识别技术模式结构大致相同，比如在图像预处理方面，采用技术大致相同，利用图像灰度化、灰度拉伸和图像腐蚀等对图像感兴趣部分进行突显，但每个模块所用技术各有不同。

比如在操作平台方面，目前有基于 MATLAB 车牌识别系统和基于 OpenCV 的车牌识别系统。前者利用 MATLAB 中自带的函数对图像进行腐蚀膨胀，字符归一化等处理，但是执行速度较慢难以高效的承载复杂的算法结构[1]，而 OpenCV 作为一个完全开源并且跨平台的计算机视觉库，其视觉处理算法丰富并且执行速度较快[4]。

针对不同类型图像，有多种不同的增加处理方法。如针对运动模糊，有一种基于模糊度的自适应迭代盲解卷积去模糊算法和采用 Lucy-Richardson 复原方法对运动模糊图像进行复原[5]；针对较暗图像，有根据平均灰度值来决策是否加强边缘检测算子的方法；针对多雾霾天气下车牌图像，有一种基于暗原色先验的方法对有雾图像进行去雾处理[6]。

在车牌定位方面，常用的方法有基于文理特征分析的定位方法、基于数学形态学的定位方法、基于边缘检测的定位方法、基于小波分析的定位以及基于图像颜色信息的定位方法等[4]。针对不同的场景定位方法也有不同程度的优化，比如为了弥补由于光照变化以及拍摄角度等原因带来的光照不均匀以及阴影问题，有一种基于光照补偿校正车牌图像二值化算法。

在车牌字符分割方面：有基于垂直投影的字符分割方法、基于模板匹配的字符分割方法、基于连通域思想的垂直切分法以及基于模板匹配和垂直投影相结合的字符分割方法等方法。基于垂直投影的字符分割方法较为简单并能在大多情况下较好的分割，但在车牌区域字符有粘连的情况会出现错误；基于模板匹配的字符分割方法，能解决字符粘连问题，但对于变形图像分割会产生错误；基于连通域思想的垂直切分法，能解决噪声问题，但是对汉字分割错误较大；结合前两种方法根据其互补性，有基于模板匹配和垂直投影相结合的字符分割方法[4]。

在车牌字符识别方面，有基于模板匹配的字符识别方法以及基于神经网络的字符识别方法。前者算法简单直接易于理解并且有较快的识别速度，通过分割得到的车牌字符图像与事先准备好的模板库中的字符图像逐个对比来实现。后者多采用基于 BP 神经网络的方法，适应性和分类能力强[4]。

第三节 发展趋势

由于现阶段车牌识别系统多安装于 PC 终端，所以迫于终端计算机硬件水平的限制以及车牌识别系统实时性的要求[7]，那些识别率很高同时算法较为复杂的算法仍然难以广泛应用，使得识别率仍不能满足需求。针对这一现状，基于云计算技术的车牌识别系统应运而生，于 2013 年科学前沿杂志第三期便有学者开始提出并研究基于云计算技术的车牌识别系统，随着大数据时代的来临，基于云计算技术的车牌识别系统将会是未来的发展趋势。目前移动终端的相对较少，而无线移动的智能化交通系统正在不断发展[7]，所以将来在车牌识别的各个模块中，算法针对不同环境等因素还将不断优化从而提高复杂环境下的车牌识别率，并且基于云计算技术和移动终端的车牌识别系统也将不断发展和优化。

除了功能上发展有待提高，性能上也仍需改进。车牌识别系统的识别率随着算法的优化正在不停的提高且对原图片的要求也越来越低，但是识别率仍然无法做到接近 100%的水平，仍处于 90%左右的识别率，所以算法的再优化也将成为未来的重要研究课题。

第四节 论文结构安排

本论文一共含有八个部分，如下：

第一部分：绪论。主要论述了车牌识别系统研究的背景以及其社会经济价值和应用领域；分析了车牌识别系统各技术的国内外现状以及其中蕴含的问题和发展趋势；阐述了本文结构安排。

第二部分：车牌定位。本章分析和比较基于图像色彩信息定位方法和基于边缘检测的定位方法的优势和局限，由于两者具有良好互补性本文采用两者算法相结合的定位方式。

第三部分：图像形态学处理。本章主要分析和研究车牌定位模块中所运用的图像腐蚀和膨胀算法；针对原始图像中车牌区域可能存在偏转和倾斜，采用旋转和扭正操作对图像进行复原。

第四部分：车牌判断。本章分析和研究机器学习，通过运用 SVM 训练模型判断将车牌定位模块输出的候选车牌图像是否是车牌，从而提高识别率。

第五部分：车牌字符分割和识别。本章车牌字符分割部分主要分析中文字符的特性以及传统分割的局限性，本文采用基于城市字符的车牌字符分割方式。车牌字符识别部分主要分析和模式匹配算法。

第六部分：系统设计与实现。本章主要分析车牌识别系统的可行性分析以及需要分析。阐述该系统整体构架以及功能模块划分和实现，并附带给出系统的界面展示，以及系统出错分析。

第七部分：总结与展望。总结整个工作，对有待改善的地方提出展望。

第二章 车牌定位

车牌定位是车牌识别系统中的重要模块，也是车牌识别系统最重要的第一步。车牌定位的准确性直接关系和决定了之后环节的运作，所以车牌定位环节需要具有较高的鲁棒性和准确性。传统的车牌定位方法为仅基于图像色彩信息的定位方法和仅基于边缘检测的定位方法虽然能够实现车牌初步定位但是其鲁棒性并不高，由于两种定位方式具有较好的互补性，本文采用基于两者结合的车牌初步定位方式。

第一节 中国车牌特征

在图像处理中常以图像的特征来描述图像，所以车牌的特征是车牌定位的关键参考依据，充分的了解和分析车牌的特征是车牌定位的前提。所以基于车牌的某一种形态特征即可形成一种车牌定位的方式。如基于图像色彩信息的车牌定位方式即运用车牌的颜色特征进行定位。常用到的车牌特征有：

（1）形状特征：我国车牌由国家统一标准尺寸，前后车牌都为有规律的矩阵，前车牌标准外轮廓尺寸为 440x140，小型汽车后车牌与前车牌尺寸相同，大型汽车后车牌标准外轮廓尺寸为 440x220[8]。即使车牌因图像离外设的距离大小产生变化，但是车牌的长宽比相对统一，当然也会因外设与车牌的角度而产生失真。

（2）颜色特征：我国车牌的颜色有着明显的特征，车牌的颜色分区一共分为底板颜色和文字颜色两块。其中底板颜色在图像中色块大且颜色特殊所以常用车牌的底板颜色作为车牌定位的依据。我国车牌底板颜色有：蓝色、黄色、白色和黑色[8]。

（3）字符特征：车牌的字符主要由中英文字符和数字字符组成，每个字符宽度为 45，高度为 90，间隔符宽 10，字符间隔 12。标准车牌（小型汽车和大型汽车）首位为省名的简称，共有 31 个字符；次位为英文字符共 25 个字符，不包含字母“I”；后五位为英文字符或数字字符共 34 个字符，不包含字母“I”和字母“0”；字符颜色有白色、红色和黑色三种[8]。

（4）边缘特征：由于字符颜色和底板颜色相差明显，所以在图像中的字符边缘处会形成灰度突变边界，因此容易辨别字符边缘；并且底板和背景之间也存在颜色差异，容易辨别边缘。

第二节 基于边缘检测的车牌定位方式

一、边缘

边缘检测定位方式即利用了图像的边缘信息进行定位，因为物体内部灰度的变化是相对平缓的，而在物体的边界处灰度会发生明显的变化，所以边缘即为物体周围图像灰度急剧变化的像素的集合，它是图像分割最重要的依

据。边缘有方向和幅度两个特征。沿边缘走向，像素变化较平缓；而垂直于边缘走向，则像素变化较剧烈。剧烈程度分为两种：阶跃状边缘（边缘两边像素灰度值明显不同）和屋顶状边缘（边缘处像素灰度值由小到大或由大到小的转折点）[9]。

二、边缘检测

根据边缘上像素灰度值一阶导数较大和二阶导数在边缘处灰度值为零的特点，对图像进行基于搜索（对图像各像素点进行微分）和基于零交叉（对图像各像素点进行二阶微分）的边缘检测方式。

边缘检测的主要工具是边缘算子即基于梯度的滤波器。常见的一阶边缘算子有 Roberts Cross 算子、Prewitt 算子、Sobel 算子、 Kirsch 算子、罗盘算子；二阶边缘算子有 Marr-Hildreth，在梯度方向的二阶导数过零点，Canny 算子，Laplacian 算子[10]。

由于导数的计算对噪声很敏感，所以在使用边缘检测器之前会对图像进行滤波处理。但由于同时也导致了边缘强度的损失，因此需要对图像先进行增强边缘操作和降低噪声操作两个步骤。

三、车牌边缘检测

车牌定位时多采用 Sobel 算子，虽然 Sobel 算子只能检测水平和垂直边缘，但是在对细纹理不作要求时，其效率更高并且算法简单。

由于车牌底板颜色和字符颜色明显与背景差异较大，所以可以使用边缘检测的方式来定位车牌。定位原理为：将车牌视为闭合边缘区域为矩阵且长宽比一定的区域。

四、优劣

基于边缘检测的车牌定位方式算法简单容易实现并且效率高，但是对于复杂环境的车牌识别准确率却不高。因为边缘检测的定位方式对于原图的要求高，当图像中出现轮廓边缘亦是矩阵的情况或者垂直边缘交错的情况将无法排除筛选。如图 1- 1,图像中车牌区域周围有很多边缘，所以在识别的时候将无法将车前杠和车牌区域区分开。并且当图像中车牌部分因为外设或自然情况边缘不清晰的情况将无法识别。

图 1- 1 车牌周围边缘交错的车辆图像

第三节 基于图像色彩信息的车牌定位方式

一、 颜色定位 HSV 模型

按照车牌的颜色特征，车牌的底板颜色分为蓝色、黄色、黑色和白色四种。以最常见的蓝色车牌为例，车牌的定位的主要原理是寻找图中蓝色的长宽比一定的矩形区域。普通定位则为在图像中依次寻找蓝色、黄色、黑色和白色的长宽比一定的矩形区域。

所以在颜色定位中最重要的依据就是颜色模型。最常见的颜色模型是 RGB 颜色模型，将颜色分为 R（红）、G（绿）、B（蓝），任何色光都可以用 R、G、B 三色不同分量相加混合而成[11]。模型如图 1-2：图为 RGB 颜色模型，每一个颜色都可以有 R、G、B 三个分量来表示。

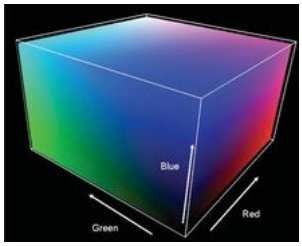

图 1-2 RGB 模型

但是 RGB 模型却不适用定位，以蓝色车牌为例。自然拍摄情况下拍摄的蓝色车牌中的蓝色各不相同，无法对于 B（蓝）分量进行明确的阈值划分，并且即使 B 分量确定，R、G 分量的不同值也会改变色相属性，所以无法采用 RGB 模型。

HSV 模型解决了这一系列问题。HSV 模型中每一种颜色都是由色相（H）、饱和度（S）和明度（V）所表示。模型如图 1- 3 :图为 HSV 模型,H 的取值范围为 0-360 代表颜色，V、H 的取值范围为 0-1 代表饱和度和亮度[11]。由于 H 分量是模型中唯一和颜色相关的分量。只要 H 的值固定不变，当 S、V 分量不太小使图像偏向于黑色时颜色便相对稳定。以蓝色车牌为例，只要将 H 分量在 200-280 之间来决定蓝色信息，而 S、V 分量在 0.35-1 之间即可。

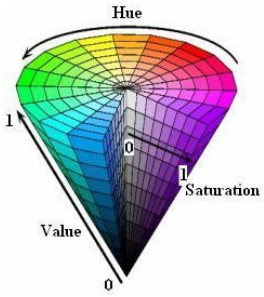

图 1- 3 HSV 模型

二、优劣

基于图像颜色信息的车牌定位方式原理直接易懂，但是颜色定位还是会有不足，当光线不足和车身颜色也为蓝色时，车牌便难以定位。如图 1-4 蓝色车辆所示，此时颜色定位将无法分辨车辆本身和车牌区域。

图 1-4 蓝色车辆

第四节 基于边缘检测和图像色彩信息定位的车牌定位方式

在分析了基于边缘检测和基于图像色彩信息的两种定位方式之后，不难发现两种方式都存在缺陷，但具有良好的互补性。在边缘检测时边缘算子的处理方式需要先将图像转变为灰度图像，如此便舍弃了图像重要的颜色信息，造成了信息损失，而图像色彩信息的定位方式正好可以弥补这一点。

本论文采用两种方式相结合的定位方式以相对稳定的图像色彩信息定位方式为主，当图像色彩信息的定位方式定位出的结果数量少于 5 时，认定其定位失败，此时采用边缘检测定位方式再进行定位，同时把两次定位的结果都输出给下一个模块。该定位方法弥补图像颜色信息的损失以及因颜色相近而无法定位的错误，提高系统的鲁棒性。

在基于图像色彩信息的定位方式时，定义蓝、黄和白三种颜色匹配模型（由于黑色车牌较少，先不考虑此情况），依次在图像中检索三种类型的颜色车牌，并根据所获取区域的长宽比来筛选是否属于车牌区域，当未检索到车牌区域时调用边缘检测定位方式。

在采用边缘检测的定位方式时，本文采用的检索方式不是寻找车牌边缘而是寻找车牌字符边缘，系统将车牌定义为仅包含字符的外接最小矩阵。利用 Sobel 算子检测垂直边缘找出字符边缘，并运用形态学操作使得所有字符连通以此获得该连通区域的最小外接矩阵。为了获得更好的鲁棒性进行 Sobel 算子边缘检测时对图像进行两次检索，对第一次检索后得到的区域再进行一次更精细的 Sobel 运算以此来增加准确性，但是会降低速度。

利用相结合的车牌定位方式，可以定位两种方式单独无法定位的车牌，适用的场合更加丰富，使得系统的性能和效率得到显著的提高。

第三章 形态学操作和图像处理

在车牌定位中，除了查找可能的车牌区域之外，还需要对图像进行一系列形态学操作和图像处理。如利用边缘检测检索字符边缘后需要对字符边缘进行形态学操作使得车牌字符连接成一个连通区域便于去轮廓；又如在车牌区域获得之后，需要对图像进行一系列图像处理排除非车牌区域和使得车牌尺寸形态统一，使得后续的操作能够更方便执行。

第一节 形态学操作——闭操作

闭操作通常是为了消除狭窄的间断和细长的鸿沟，消除小的空洞，并填补轮廓线中的断裂，为了使得字符连接成一个连通区域需要对图像进行闭操作，即对图像进行先膨胀后腐蚀的操作[12]。

腐蚀和膨胀操作都针对二值化的图像进行的操作。

一、 膨胀

膨胀操作即是将图像中亮的区域增多。如果周围有图像，延展该点，放大图像边界，以此对图像的边界进行膨胀。具体操作原理是：用一个 m*n 的矩形模板，遍历图像中的每一个像素点，对模板中的所有像素进行从小到大排序，将模板中心的像素点的像素值为该模板中所有像素值中最大的值。如此便可将图像外围的突出点连接并向外延伸。

二、 腐蚀

腐蚀操作与膨胀操作相反，是将图像中暗的区域增多。如果周围有空隙，腐蚀该点，缩小图像边界，以此对图像的边界进行腐蚀。具体操作原理和膨胀大致相同唯一不同的是中心像素点的像素值为所有像素值中最小的值。如此便可将图像外围的突出点腐蚀缩进。

三、 闭操作

闭操作的作用是连接图像内部断掉的部分。闭操作先对图像进行膨胀操作后对膨胀后的图像进行腐蚀操作，从而把各独立相近的图块相连成一个无突起的连通域。在车牌定位中需要将字符图块相连成一个连通区域才能获得外接最小矩阵（取轮廓）找到车牌位置，所以对原始图像进行边缘检测后的图像进行闭操作，使得车牌区域连通。

第二节 图像处理

一、 尺寸判断和角度判断

仅仅依靠边缘检测和形态学处理所检测到的图块中仍有极大的可能存在不属于车牌的情况，所以需要对图像进行尺寸判断和角度判断从而排除一些不属于车牌区域的图像。

首先，对图像进行尺寸判断。其原理是：根据车牌的形状特征，对外接矩形进行判断，以判断它们是否是可能的车牌区域。我国车牌的一般尺寸为440*140mm，长宽比约为3.14。然而由于实际拍摄时会产生误差，所以设置一个误差值，并计算的最大的长宽比和最小的长宽比以及最大的面积值和最小的面积值。当定位后的图块的长宽比和面积都在误差值范围内时，判定该图块为可能的车牌区域。

其次，对图像进行角度判断。其原理是：根据传统情况，根据图块的下边缘离水平位置的角度筛选可能的车牌区域。由于图像中的车牌区域都不会有太大的倾斜，所以设定若外接矩形的偏斜角大于60度时，认定为非车牌区域。

当然经过这两步判断之后也无法证明筛选后的图像是车牌区域。

二、 截取

为了防止在筛选过程中将可能的车牌区域排除，所以在角度判断时阈值的划定比较大，可是这也容易使得可能的车牌区域增多。如果对整个图像进行旋转操作，由于其数据量过大会使得系统的运行速度大大降低。为了降低数据计算量，对图像进行感兴趣部分截取即把可能的车牌区域截取其外接正方形。

三、 旋转和扭转

筛选后的可能的车牌区域是需要输出车牌判断的图像，称其为可能的车牌区域。可是可能的车牌区域尺寸、偏斜角都不统一，这就增加之后车牌字符分割模块的难度。所以需要对候选车牌尺寸统一，并使得车牌摆正水平。

由于拍摄情况的未知，道路摄像头拍摄的车牌会有三种情况：正视角水平车牌、正视角倾斜车牌和斜视角车牌。若为正视角水平车牌情况，获取后只需统一尺寸即可。所以只需要讨论另两种情况，由于拍摄角度的关系，如果物体与摄像头之间存在角度不是正视角时，物体的形状会因为角度而产生变化，所以对于车牌而言，正视角的车牌将为矩形，而斜视角的车牌则为四边形，以此便可区分这两种情况。

（1）正视角倾斜车牌，需要对车牌区域进行旋转使其水平。

将截取下的正方形部分，根据倾斜的角度进行旋转。可是如果直接旋转，可能会造成数据丢失，比如旋转45度后，图像的宽度和长度都超过了原始的

长度，所以需要先进行安全旋转范围计算，即将图像扩充到旋转后信息不丢失的大小，系统设定新图像长宽设置为原图像旋转 45 度后的长宽的长度（长宽为原来的 1.5 倍），新增的像素点值为 0 即黑色，此时再进行旋转时，图像的信息便不会丢失。旋转后，重新获取图像中车牌区域的二值化图像，即对该图像进行一次车牌定位以及闭操作。由于角度偏转带来的信息缺失，斜视角车牌的白色图块将不是矩形，所以如果白色图块为矩形，则为正视角倾斜车牌，而若白色图块为四边形时则为斜视角车牌。

（2）斜视角车牌，需要对车牌区域进行扭转操作。

如何判断白色图块为矩形还是四边形的方法，则是通过计算其斜边到边框的距离。可以通过统计每一行从头开始像素值为 0 的个数，来表示斜边到外接矩形的距离，当每一行的距离都很小很接近时，判断为矩形，当距离为从大到小或从小到大的趋势时，判断为四边形，并且可以由距离计算得该斜边的斜率，因为之后的扭正过程中会需要斜率计算偏移值。

当判定为斜视角车牌时，需要对车牌进行扭正操作，其原理是通过将原始图像和变换后图像的左上角、左下角和右上角的位置进行一一对应后，系统找到整个图像的对应关系，从而对图像进行扭正操作。由斜率可知车牌区域的左上角与外接矩形框的距离，若将扭正后图像的外接矩形框左上角位置定义为原左上角位置向左偏移该距离后的位置，并且固定左下角和右上角的位置，车牌区域便会被扭正如图 2- 1 所示。所以在扭正操作时，左下角和右上角的位置原始图像和变换后图像位置一致，若原始图像左上角的位置为（0,height），变换后图像的左上角的位置为（0-n ，height），其中 n 表示偏移距离。三个点坐标的位置对应后，图像便会被扭正，将斜视角的车牌转换为正视角的车牌。

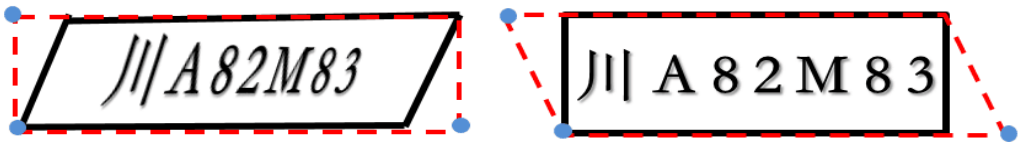

图 2- 1 扭正过程

第四章 车牌判断

即使经过了基于色彩信息和边缘检测两种车牌定位方法进行定位车牌并对于初步定位后的车牌进行了简单的筛选，但是仍然无法确定截取下来的图像便是车牌图块，亦有可能在背景中存在同样特征但是不属于车牌的情况。为了排除这种情况的干扰，则需要对所有经过车牌定位和图像处理的可能是车牌的图像进行一个判断，这里便需要用到机器学习的相关原理。

第一节 机器学习

由于之前通过算法和指令仍然不能让计算机正确的识别出车牌区域，所以需要借助机器学习帮助判断之前截取的部分是否真正是车牌区域。

机器学习顾名思义就是让计算机进行训练学习从而可以对新的数据进行预测，和我们人思考的方式相同，通过一系列数据经验所建立起某种规则，当新的事件发生的时候，根据这种规则进行预测结果。所以机器学习是利用数据获得数据，从而帮助我们解决指令和算法无法解决的问题。

机器学习可以分为两个部分，训练部分和预测部分。训练部分指的是将历史数据输入计算机，然后通过机器学习算法处理从而得到一个训练模型。预测部分指的是当新的数据输入时，计算机将其放入之前的训练模型进行判断，最后输出预测的结果。

机器学习中最重要的两个依据就是数据和模型。数据是指在计算机训练模型时所用到的历史数据，数据越多训练出的模型也就越精确，就如同人一样，经验越是丰富的人提出的想法越有价值。在如今的大数据时代，如何能利用好数据的价值十分的重要，机器学习是一种十分有效的数据分析方法，它可以将庞大的数据进行整合并从中归纳出一种规则，从而能够对新的数据进行预测。机器学习另一个重要的依据是模型，在计算机获得大量数据进行训练时，会对数据进行分析从而得出一个模型。以车牌判断为例，计算机在训练之前先收集许多车牌图像和非车牌图像并存储入计算机，之后利用机器学习算法对图像进行分析后得出一个模型即分类界线，界线的一侧为车牌图像，一侧为非车牌图像，当新的数据输入时计算机便根据分类界线预测出是否为车牌图像。

第二节 SVM 分类器

支持向量机（SVM）是机器学习算法的一种，它是线性分类器的一种强化[13]。所谓线性分类器，即数据可以用一条直接进行分类，举一个简单的例子：假设在价格和使用时间的二个维度上，汽车是否可以回收的一些数据如图 3-1 可回收车辆的分类数据图所示，可回收汽车为白色三角，不可回收汽车为黑色三角，当前输入新数据为斜线三角，图片为在价格和使用时间两个维度上汽车的分布，很明显是否可以回收的汽车数据可以被一条直线分开，当又

新的汽车数据输入使用时间和价格时便可预测该汽车为不可回收汽车。

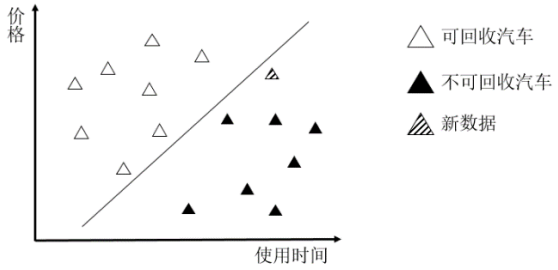

图 3-1 可回收汽车的分类数据图

但是该直线并不唯一，如何确定该分类直线是最佳分类直线遵循分开的两类的间隙越大越好原则，具体数学论证本文不作详细分析。此分类方法是线性分类器方法，但是实际情况会相对更加复杂并且维度也会增多，便会出现直线无法分类的情况，如图 3- 2 可回收汽车无法直线分类数据图所示，图中可回收汽车和不可回收汽车数据交错，无法简单的用直线进行分割，所以如何用正确的曲线来进行分类将变的比较难定义，此时运用到了 SVM 分类器。

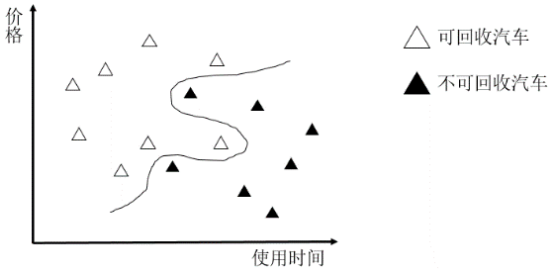

图 3-2 可回收汽车无法直线分类数据图

SVM 分类器的原理是在当前维度无法进行线性分割的时候，将数据原来所在的线性空间提高到一个更高维度的空间，并在高维度的空间下利用超平面进行分割[13]。当维度不断增加时，终会有一种情况使得数据可以进行分割，具体数学论证本文不作详细论述。

第三节 SVM 车牌判断应用

OpenCV 中自带了一个类 CvSVM，它是机器学习类用于使用 SVM 进行训练预测的功能。在对车牌进行判断时，需要先载入预先训练好的 SVM 模型中，然后将在车牌定位模块中输出的可能是车牌的图像进行直方图统计，并将统计的结果作为其特征输入模型，系统会自动预测出该图像是否为车牌图像，本系统直接采用已经训练好的模型并没有真正对数据进行训练，下文仅简单对训练模型进行分析。

模型的训练过程可以分为获取数据、数据分类、数据训练和测试这四个部分。获取数据即是获取大量的车牌和非车牌图像，它便是上文提到在机器学习中的历史数据部分，它可以由大量图像经过车牌定位模块后获得，在此

环节数据量越大后面训练的模型便会越准确。第二步数据分类便是对已有的车牌和非车牌图像进行分类，分到两个文件夹中分别为车牌图像文件夹和非车牌图像文件夹。第三步数据训练，利用 CvSVM 类 train 方法，先载入车牌图像并给图像打上标签车牌图像为 1，非车牌图像为 0。之后配置 SVM 模型训练参数，进行 SVM 模型训练。最后一步便是测试，利用新的车牌图像和非车牌图像对模型进行测试，计算模型的准确性。

在训练的过程中，可以进行一些优化。

（1）例如在载入历史数据时，可以载入图像的所有像素作为特征进行训练，也可以采用图像的直方图统计数据作为特征进行训练。本系统采用的模型利用直方图统计数据作为特征进行训练，所以新数据载入时将新数据的直方图统计数据进行的预测。

（2）又例如在 SVM 训练时最重要的就是核函数即如何将低维度的空间转换到高维度的空间，OpenCV 支持的核函数类型有 4 种，liner 核和 rbf 核运用最多。liner 核表示无核，即没有向高维度进行转变，适用于数据维度本身多但数据容量少的情况，比如将图像的每一个像素作为一个特征维度便可以采用 liner 核；rbf 核表示利用高斯函数作为核函数，其进行了维度变换，转换的维度与总数据容量一致，适用于数据本身维度少，但是数据容量多的情况，比如将图像的直方图统计数据作为图像本身维度，利用大数据进行训练时便可以采用 rbf 核。

第五章 车牌字符分割和识别

第一节 车牌字符分割

定位并截取出车牌区域后，需要对车牌中的每一个字符进行字符分割，获得单一字符的图像，从而可以对每一个字符进行下一步的识别工作。字符分割有很多种方法，本文采用字符取轮廓法。

在分割的过程中，英文字符和数字字符因为字符本身具有连通性所以获得外接最小矩形时能够获取字符所在位置，而中文字符因为有偏旁结构其本身缺乏连通性，获取外接最小矩形时可能会存在将中文字符拆分的情况。如中文字符“川”，其为左中右结构，在获取轮廓时会将“川”分割为三个部分，从而无法获得中文字符的位置，鉴于中文字符的特殊情况，本文采用城市字符反推法，获得中文字符的位置。

一、中文字符

由于中文字符缺乏连通性，所以无法直接使用取轮廓的方法获取，直接获取很可能出现断节的情况。但是根据我国车牌的特征，我国车牌的第二位字符为英文字符代表城市代码，且七位字符的字符宽度相同，于是规定第二位英文字符为城市字符。由于英文字符具有连通性，所以容易获取其外接矩形，并且城市字符的位置大致约在整个车牌的七分之一到七分之二的位置，所以城市字符的位置很容易获得。在获取得到城市字符的位置之后，将城市字符的外接矩形向左进行推移，从而获取中文字符的位置。

既然字符的间隔宽度一定，字符宽度也一定为何不采用滑动窗口来分割字符呢？滑动窗口的方式的确可以实现车牌字符的分割，但是需要在车牌定位的时候将车牌的区域截取的正好只有车牌部分，才能根据滑动窗口分割车牌，这便对车牌定位模块增加了难度，如果没有正好截取车牌部分，比如在左部多了一部分非车牌区域，此时便无法判定最左边的中文字符的位置，如果还是根据固定的滑动窗口移动，便会出现分割错误的情况。所以先找到城市字符位置再反推这个算法的效率和效果更好一些，算法也简单实现。

二、普通字符

普通的英文字符和数字字符可以直接通过二值化处理后用获取轮廓得到单个字符的最小外接矩形得到个字分割进行分割。但是其中也会出现一些主要注意的问题：

（1）边框错认成字符。由于车牌定位后的车牌图像可能会包含车牌边框，所以在车牌分割的时候，很可能将边框错认为数字字符“1”被分割出来。这里可以根据车牌的字符特征即车牌中所有字符为 7 位，所以只要划分出中文字符的位置后往后取 6 位最小外接矩形，如此便排除了边框错认的情况。

（2）去铆钉。由于我国车牌上会有铆钉，在识别时会和字符混合造成错误操作，所以在这一环节需要将铆钉去除。这里可以根据铆钉处于车牌顶部和车牌底部一般不与字符在同一行，即二值化操作后先扫描车牌的每一行像素点，字符行像素跳变次数多，而非字符行像素跳变次数少，所以判定跳变次数较少的行为铆钉行，并将该行所有像素点赋值为 0，这样便大大消除了铆钉的影响。

（3）在获得了全部车牌字符的外接矩形位置后，需要对车牌字符进行分割并且进行归一化操作将字符图像归一化到统一格式，便于下一步车牌字符识别模块的运行。同时系统在后台保存下单个字符的图块，以便之后车牌字符识别出错时使用。

第二节 车牌字符识别

车牌识别系统的最后一个环节便是车牌字符识别模块，利用预先系统后台储存好的模板进行模式匹配识别车牌中的每一个字符。

车牌字符识别的原理是遍历模板中的每一类字符包括中文字符模板、英文字符模板和数字字符模板。以第一个中文字符为例，将字符分割后所得的中文字符图像和中文字符模板中的每一个模板图像进行各像素匹配，当图像和模板图像中每一类图像最相似时，认定所要识别的中文字符为该类字符。同理，第二位字符在英文字符模板中进行匹配，后五位字符在英文字符和数字字符模板中进行匹配。

由于使用简单的像素点模板比较识别，所以容易出现识别错误的情况。仅在理想状态下即模板通过大数据建立、容量足够大的情况下，出错率会逐渐减少，但是数据量增大的同时会降低效率增加运行时间。

这里使用伪机器学习的方式稍稍优化车牌字符识别模板，由于在车牌字符分割模块，系统在后台自动保存了分割后的单个字符图像，所以当识别出错时，可以手动将出错的字符图像保存到正确的字符模板中。如此便可增加模板容量提高识别率，亦可纠错使得在下一次运行时该车牌的识别便会正确。

第六章 系统设计与实现

本章节将主要针对本系统的设计与实现进行分析介绍和展示。

第一节 可行性分析

一、 系统要求

本系统先要求用户加载车牌字符模板，再要求用户输入原始车牌图像。然后用户可以要求系统对车牌图像进行车牌定位操作和图像操作提取车牌部分并显示截取的车牌正视角图像；可以要求系统对定位后的车牌字符进行分割并显示分割后每一个单个字符；可以要求系统将车牌号将以文字的形式显示；用户也可以要求系统自动完整整套车牌识别操作，直接显示车牌区域、分割后车牌字符和文字形式显示的车牌号。

二、 条件、假定和限制

车牌识别系统准确性除了与图像定位、字符分割、字符识别算法的优化有关还与原始图像质量有关，所以会受到原始图像拍摄时的物理环境和物理外设的影响，并且由于我国车牌含有汉字，且具有不同汽车的车牌尺寸、颜色均不相同等特征，车牌识别技术识别率仍只有 97%[2]，在恶劣的环境下识别率还会降低。

三、 对现有系统的分析

本文已经对现有系统中的各个模块算法和实现进行了分析，现在对于现有系统的整体框架进行分析。

现有系统的主要技术部分的数据流程图如图 5- 1 现有系统的数据流程图所示：图表示了每一模块的输入和输出数据以及模块划分。

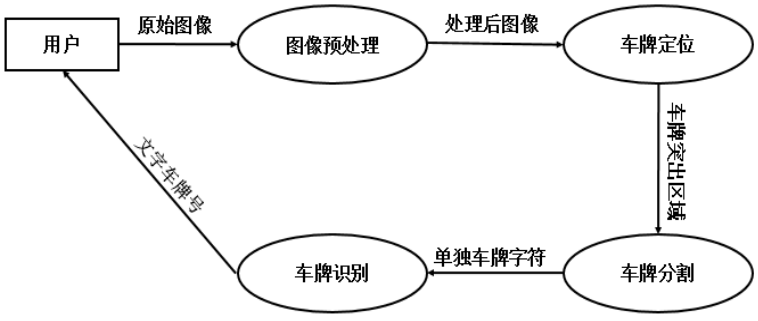

图 5- 1 现有系统的数据流程图

现有系统能够很好的处理类似于收费口车牌识别系统的功能，因为车辆进入收费口时速度缓慢且相对于摄像头的角度相对固定，此时拍摄的车牌图像属于正视角并且图像清晰，图像背景简单。现有车牌识别系统的局限性在

于，运行速度不高并且针对多样化的物理环境下拍摄的车牌图像处理困难，并且容易报错，识别率低。因为相对于监控摄像和移动设备拍摄的图像车辆的角度随意并且图像的质量不定，此时便需要对现有车牌识别系统进行一系列的算法优化，使得其能够满足复杂的拍摄情况。

四、 社会经济因素可行性分析

（1）社会因素可行性分析。随着车辆的增多，交通事故发生的频率也逐渐增高，政府对于如何高效及时的处理交通问题变得越来越关注，人们对于交通事件处理的要求越来越高，原有的人工处理方法已经远远不能满足现有交通量的需求。由此信息化的交通系统已是现在以及将来交通系统的主力系统，人工排查将由于信息化系统的发展而退居二线。随着交通监控摄像设备的增多和提升，简便易行的车牌识别系统可以降低人工的工作量也提高了交通处理的效率。

（2）经济因素可行性分析。车牌识别系统可以结合现有各交通管理系统，帮助系统快速识别各类车牌，帮助解决各类交通管理问题，解决了人工识别的慢速、不联网没有时效性和高成本的问题，从而降低了时间、空间和费用上的经济成本。

第二节 需求分析

一、 业务需求

为了能够更好的与现有的其他交通管理系统相结合，更好地完成车牌识别的任务，社会对车牌识别准确率的要求越来越高，对识别场景的要求也越来越严格。由于现有的车牌识别系统无法满足复杂的拍摄情况下拍摄的车牌图像，本系统主要针对能够还原和识别监控摄像和移动设备拍摄的车牌图像，能够将偏转的车牌、斜视角的车牌进行角度恢复和车牌识别。本系统在现有系统的基础上进行优化和改进，优化模块接口，使得系统能够更好地和其他交通管理系统相结合。

二、 用户需求

本系统在车牌识别正确的情况下，对用户没有特殊需求，用户可以输入不同大小不同格式的图像作为原始图像，但是原始图像需要是较清晰的含有车牌的图像。在车牌识别出错的情况下，需要用户将分割后的单独字符图像保存到系统后台的模板中，重新加载模板纠正错误识别。

三、 功能需求

本系统识别功能一共划分为四个功能模块分别为：车牌定位、车牌判断、车牌字符分割和车牌字符识别。车牌定位部分和车牌判断部分在界面中同属车牌定位环节，其功能是定位图像中车牌位置并显示车牌区域。车牌字符分

割其功能是将车牌区域中的每一个字符进行分割并以黑白图像进行显示。车牌字符识别部分其功能是识别出车牌区域中的每一个字符并以文字的形式显示供用户使用。用户可以分步看到车牌识别的每一个功能模块的实现结果也可以一步直接看到最后显示的文字车牌号结果。

四、 运行需求

本系统对于运行环境并没有特殊的运行需求，可以运行在 windows 平台上，并且本系统中代码透明，可以嵌入其它交通管理系统，嵌入时编译平台需要是 VS 2010 搭载 OpenCV 2.4.8，如果使用 OpenCV 其它版本可能会存在兼容问题，尤其是 3.0 以上版本。

第三节 系统总体构架

本系统开发环境:Windows 操作系统 VS2010 开发工具搭载 OpenCV2.4.8。

本系统总共分为五个模块：车牌定位模块、车牌判断模块、车牌字符分割模块、车牌字符识别模块和界面设计模块。

车牌定位模块设计思路：通过基于边缘检测和基于图像色彩信息相结合的检测方式对原始图像中的车牌区域进行初次检测，并将检测出的车牌图像进行简单的筛选和统一格式的操作。

车牌判断模块设计思路：通过加载已经训练好的 SVM 模型，对车牌定位模块中输出的车牌图像进行预测，选择出只包含车牌的图像。

车牌字符分割模块设计思路：将车牌判断模块预测出的车牌图像，利用边缘检测取轮廓的方式进行分割，对于中文字符采用利用城市字符倒推法获得其位置进行分割。

车牌字符识别模块设计思路：利用后台模板和分割下的每一个字符进行像素点比较，并将像素差别最小的模板作为该字符的识别结果。

界面设计模块设计思路：在界面中显示“载入图像”、“车牌定位”、“车牌分割”、“车牌识别”、“加载模板”、“直接识别”和“退出系统”七个按钮，并且能够显示原始图像、车牌图像、车牌分割后图像和文字对话框识别结果。

本系统的数据流程图如图 5- 2 本系统的数据流程图：图中表示了本系统各模块的输入输出数据以及模块划分。

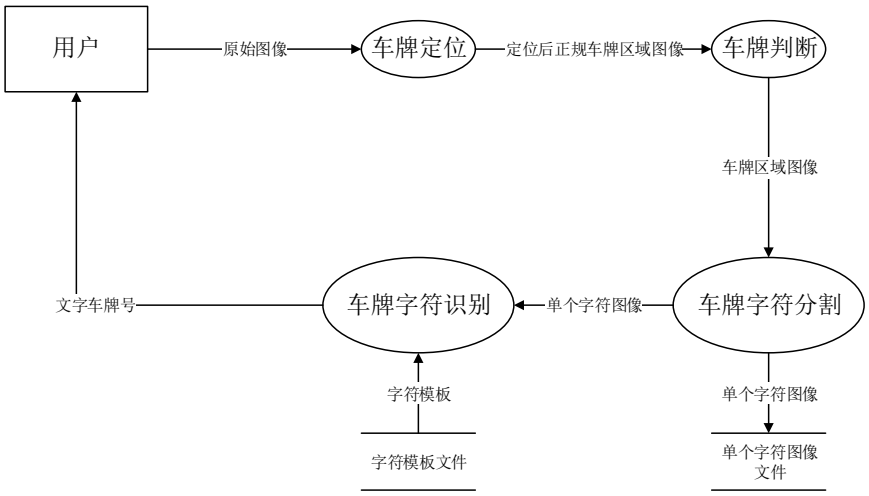

图 5- 2 本系统的数据流程图

主要模块的程序流程图如图 5-3 本系统的程序流程图：

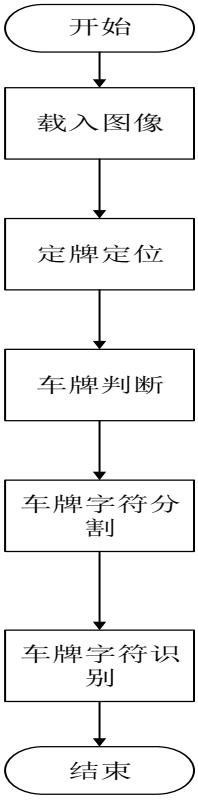

图 5- 3 本系统的程序流程图

第四节 功能模块划分与实现

本系统的主要功能模块为：车牌定位模块、车牌判断模块、车牌字符分割模块、车牌字符识别模块和界面设计模块。

一、 车牌定位模块

车牌定位模块主要是将原始图像中可能是车牌的区域进行截取。该模块分为车牌定位和图像处理两个子模块，分别完成车牌区域定位和车牌区域归一化的工作。

车牌定位子模块主要是基于色彩图像信息的车牌定位和基于边缘检测的车牌定位操作的结合操作，能够最完全化的将原始图像中可能的车牌区域进行定位。

如图 5- 4 基于图像色彩信息的车牌定位的程序流程图：表示了基于图像色彩信息的车牌定位的实现过程。

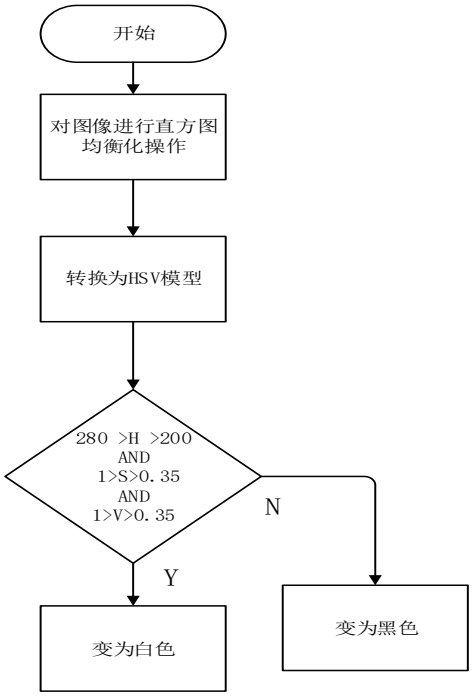

图 5- 4 基于图像色彩信息的车牌定位的程序流程图

该例为蓝色车牌，实际系统运行时针对蓝牌、黄牌、白牌均各自运行一遍，如果统一在一个模板中运行，会使得结果混乱并且降低了定位的效率，如黄色的汽车蓝色的车牌会一起被定位出。该过程中，第一步将图像进行直方图均衡化操作从而降低光照带来的影响。第二步将图像从 RGB 模型转换到 HSV 模型从而可以用 HSV 模型对图像进行定位。第三步遍历图像中所有像素点，当 H 值在 200 到 280 之间，S、V 的值在 0.35 到 1 之间时，该像素点的值变为白色，反之则变为黑色，经过此步后，图像变为车牌区域为白色，非车牌区域为黑色的图像，当然也包括白色杂质即原始图像中也为蓝色的部分，如此车牌部分便被定位出。

如图 5- 5 基于边缘检测的车牌定位的程序流程图所示：表示了基于边缘检测的车牌定位的实现过程。

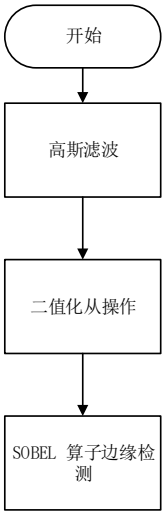

图 5- 5 基于边缘检测的车牌定位的程序流程图

该过程第一步为高斯滤波，由于之后要进行基于 Sobel 算子的边缘检测，需要对图像进行高斯滤波去除噪点。第二步为将图像变为灰度图像，该步同样是为了 Sobel 算子的边缘检测所进行的准备工作。第三步为 Sobel 算子边缘检测，求其水平方向的一阶导数，即寻找垂直边缘，该步进行两次从而提高定位准确率。第四步为将检测后的图像进行二值化操作。

图像操作子模块主要是对定位后的图像进行初步筛选和归一化统一格式的操作，目的是尽可能输出正视角统一大小的车牌区域图像。

如图 5- 6 图像操作子模块程序流程图：表示了图像操作子模块的实现过程。

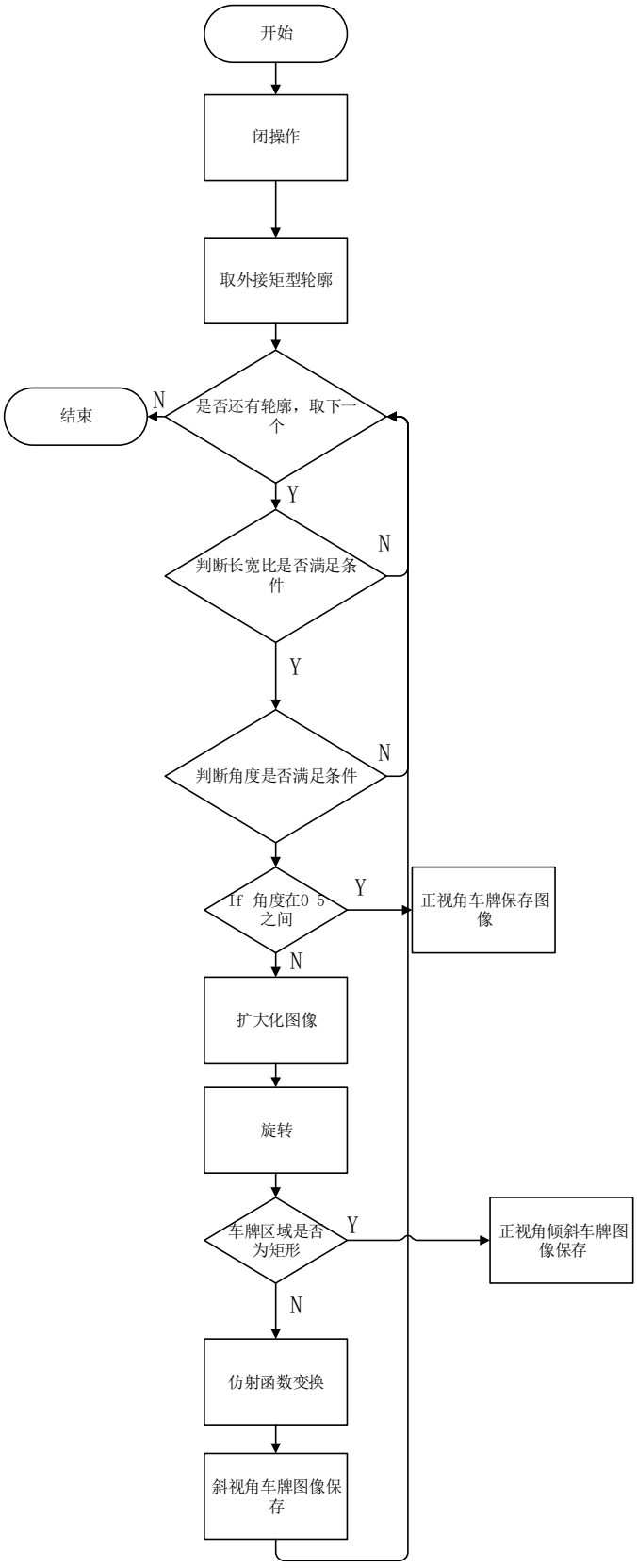

图 5- 6 图像操作子模块程序流程图

该过程第一步为闭操作，将图像中车牌区域的内部进行紧密的连接，使得车牌区域全为白色。第二步为取外接矩形轮廓，获取图像中白色图块的外接矩形轮廓，即车牌部分的外接矩形轮廓。如图像中没有到获取一个外接矩形轮廓则判定为没有定位出车牌。第三步为选取每一个外接矩形轮廓，判断其长宽比尺寸是否满足条件，满足条件则进行下一步操作，不满足条件则将该轮廓舍弃。第四步为判断倾斜角度，角度过大则舍弃，角度满足条件则进行下一步操作。第五步为判断是否为正视角车牌，当倾斜角度很小时认定为正视角车牌，不满足则舍弃。第六步为斜视角车牌判断，其余满足条件的倾斜车牌先截取包含外接矩形的正方形区域，其目的是为了减少之后的数据运算量，然后扩大正方形区域的画布面积，使其旋转 45 度后，有信息部分仍然在画布上，其目的是防止旋转时信息丢失。最后进行旋转使得外接矩形与水平线平行。第七步为斜视角判断，若旋转后原图像中车牌的白色区域为矩形时则判定为正视角倾斜车牌，若为平行四边形时则判定为斜视角车牌。第八步为扭转，对斜视角车牌进行扭正操作。第十步，将之前所有满足条件的车牌统一尺寸并保存。

在车牌定位模块输出的图像，可以用作 SVM 训练的数据进行使用，从而提高了训练的效率，也可以验证车牌定位模块的准确性。

二、 车牌判断模块

车牌判断模块主要是将车牌定位模块中定位的所有可能是车牌的区域进行 SVM 训练和预测，使得最终定位出车牌区域图像。主要判断过程分为三个阶段，分别为加载模板、单幅图像判断和多幅图像判断。

加载模板。系统中加载已经训练好的 SVM 模型，并将车牌定位模块输出的可能是车牌的图像进行直方图均衡化操作。

单幅图像判断。从车牌定位模块输出的结果中抽取一个图像，计算其直方图统计结果，具体过程为：先将图像进行二值化操作，并统计其每一行像素值为 1 的个数以及每一列像素值为 1 的个数，将其结果作为特征，也就是直方图统计结果输入到 SVM 模型中进行预测，该图像是否为车牌区域图像，返回值为 1 为是，返回值为 0 为不是。

多幅图像判断。依次判断车牌定位模块输出的可能是车牌的图像，进行单幅图像判断，当返回值为 1 时，将该图像保存进车牌判断模块结果图像栈。当返回值为 0 时，截取图像中间区域再进行一次单幅图像判断，从而解决可能在车牌定位模块截取时包含区域不仅仅含有车牌的情况，此时同样根据结果进行进栈和舍弃的操作。

由于拍摄图像可能不仅仅含有一张车牌，所以在车牌判断模块是可以将图像中车牌进行预测和定位，只要在车牌定位模块该区域被截取下来。但是本系统仅针对单一车牌的识别，所以虽然可以将所有车牌进行定位，但是仅将最先定位的车牌区域进行结果显示。如何在同一界面下选择性地识别多张车牌将是之后的研究方向。

车牌判断模块程序流程图如图 5- 7 车牌判断模块程序流程图：表示了车牌判断模块的实现过程。

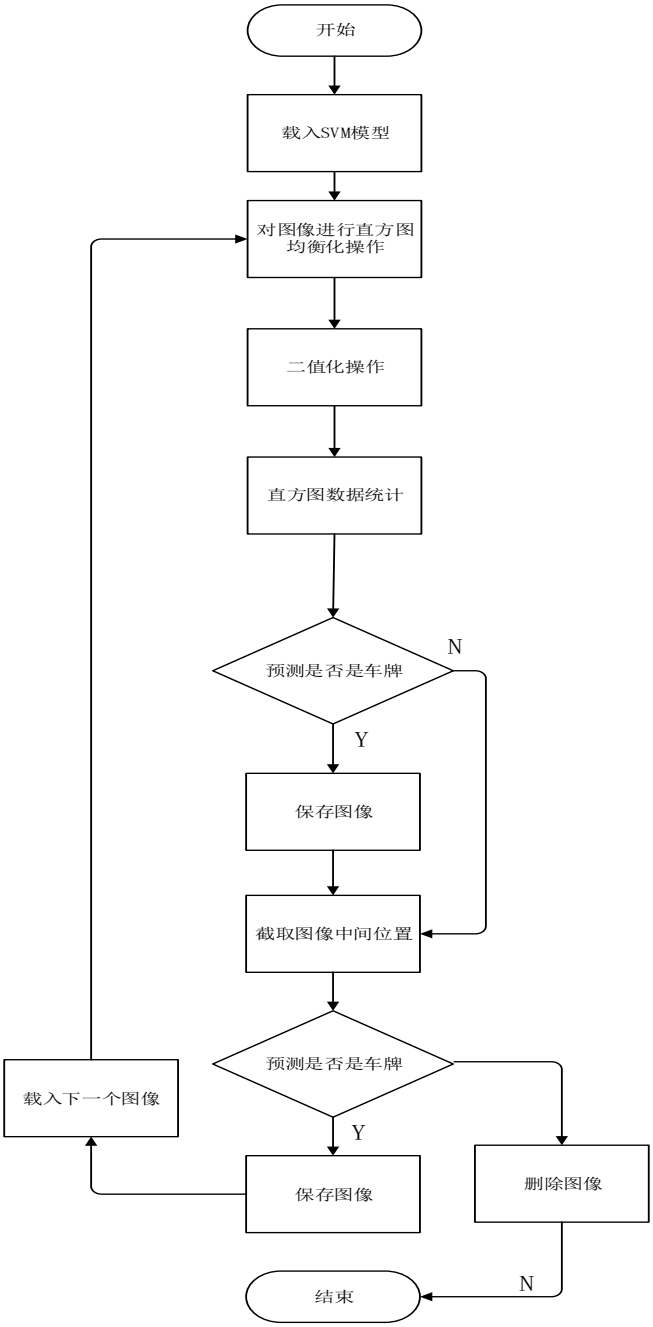

图 5- 7 车牌判断模块程序流程图

三、 车牌字符分割模块

车牌字符分割模块主要是将车牌区域中的每一个字符进行分割，即在车牌区域图像中，把字符分割成一个个只包含一个字符的小图像块。主要分割过程分为四个阶段，分别为图像处理，获取城市字符外接矩形，获取中文字符外接矩形和获取所有字符外接矩形并分割。

图像处理。在进行定位分割之前，需要对车牌判断模块输入的图像进行图像处理，以便之后的操作。在车牌判断模块中，输出的返回值结果为彩色的车牌区域图像，为了能够进行二值化操作为后续的取轮廓操作进行准备，需要对图像进行灰度化操作，因为二值化操作只能针对灰度化图像进行。之后便是对图像进行二值化操作，此时图像的字符部分为白色，背景为黑色。由于车牌部分很可能包含铆钉，所以在二值化操作之后，需要对图像进行去铆钉操作，此时图像仅包含字符，并且字符为白色，背景为黑色，也有一定可能包含线框。

获取城市字符外接矩形。对于之前图像处理后的图像进行 findConours 操作，找到图像中白色部分的外接矩形轮廓。此时找到的所有轮廓并不一定便是字符轮廓，也可能是杂质线框的轮廓，并且由于中文字符缺乏连通性，中文字符找到的轮廓可能会存在断节的情况。所以此时需要对轮廓进行筛选，由于车牌字符尺寸一定，所以将大小不一致的矩形排除，得到的所有外接矩形进行从左到右排序，并将位置在车牌的七分之一到七分之二的区域的外接矩形判定为城市字符。

获取中文字符。由于中文字符无法使用外接矩形直接获取，所以采用城市字符倒推法，具体过程为：将城市字符外接矩形轮廓向前偏移，偏移值为外接矩形宽度的 1.15 倍，并判断偏移后的外接矩形是否误将城市字符包含进去，即判断中文字符外接矩形左边界 x 加上矩形宽度 width 是否大于城市字符左边界，如果大于将城市字符左边界作为中文字符右边界。

获取所有字符外接矩形并分割。获取中文字符的位置后，依次再获取之前从左到右排序的外接矩形 6 个的位置，从而排除了将车牌右边框作为字符输出的情况，并将字符根据外接矩形截取保存。

如图 5- 8 车牌字符分割模块：

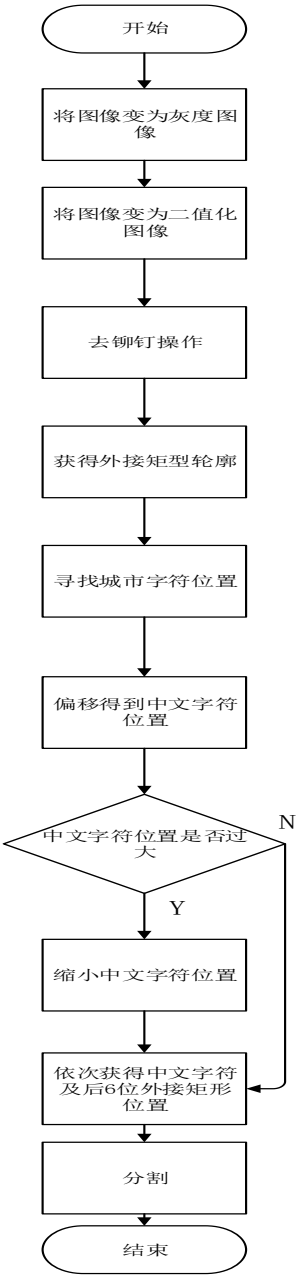

图 5- 8 车牌字符分割模块

四、 车牌字符识别模块

车牌字符识别模块主要是将车牌区域的字符进行识别。主要识别过程分为四个阶段，输入图像归一化，中文字符识别，城市代码识别和后 5 位英文数字字符识别。

输入图像归一化。车牌字符识别模块的输入项为车牌字符分割模块的返回值结果，即 7 个矩形图像，由于在分割时截取的图像和模板格式有一定的差异，所以需要在识别前，对图像进行归一化处理，使其仅仅只包含字体区域。主要操作为先对图像进行灰度化和二值化的操作，使得图像变为字体部分为白色，背景为黑色的图像，然后寻找图像的上下左右边界，以寻找左边

界为例，其设计思路为：从头遍历图像中的每一列，当第一次搜索到列所有行的像素和大于 0.1 时，即该列有白色字体部分时，便认定该列为左边界。以此类推可以找到上下右边界。如图 5- 9 寻找左边界程序流程图：其中 I 表示列 ，J 表示行

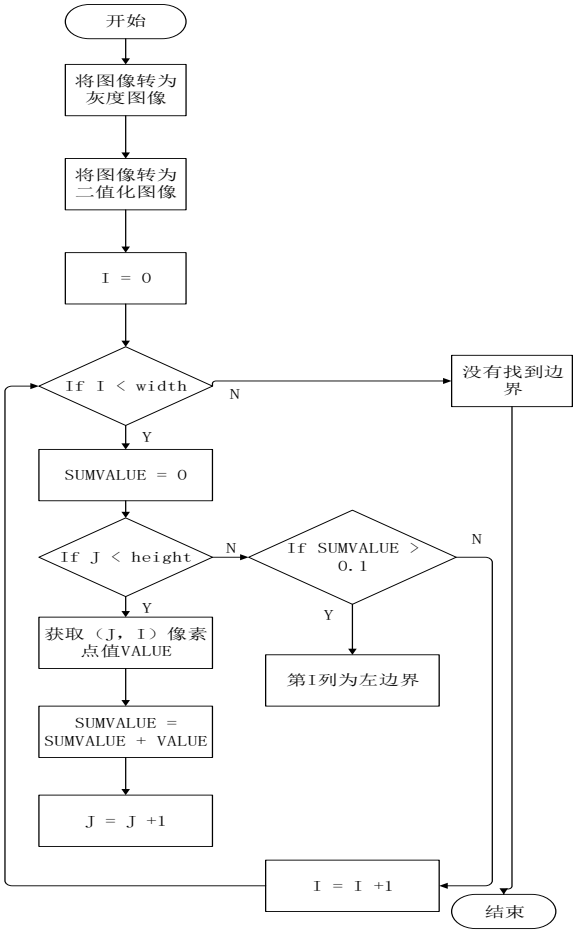

图 5- 9 寻找左边界程序流程图

中文字符识别、城市字符识别和后五位英文字符和数字识别操作过程相同，不同的是遍历匹配的模板不同，中文字符识别只需匹配中文模板，城市字符只需英文模板，后五位则是数字字符模板和英文字符模板，其设计思维为：逐个比较模板和字符图像中的每一个像素点，当像素点不一致时，差异值加 1，边遍历模板边将差异值最小的值和模板类型保存，将模板全部遍历完之后，将差异值最小的模板的种类作为该字符的识别结果输出。其程序流程图如图 5- 10 中文字符识别程序流程图：以中文字符识别为例

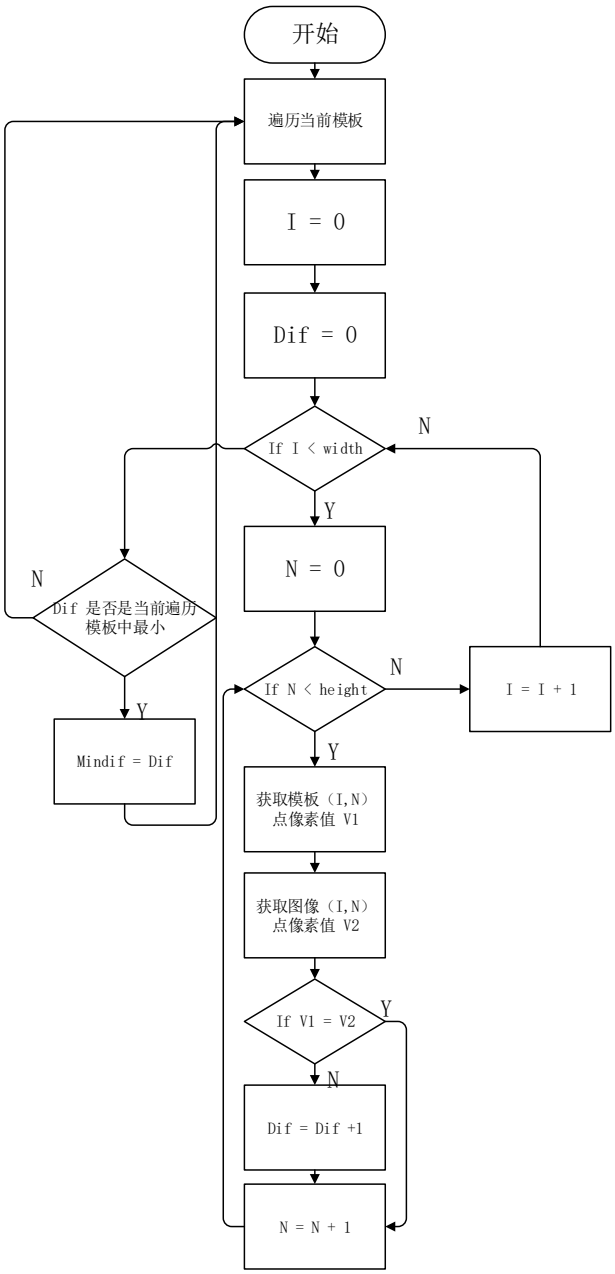

图 5- 10 中文字符识别程序流程图

五、 界面设计模块

界面设计模块主要是 MFC 界面设计以及将所有功能实现模块进行整合在界面上进行展示，程序流程图如图 5- 11 界面设计模块程序流程图：流程图中不包含报错处理。

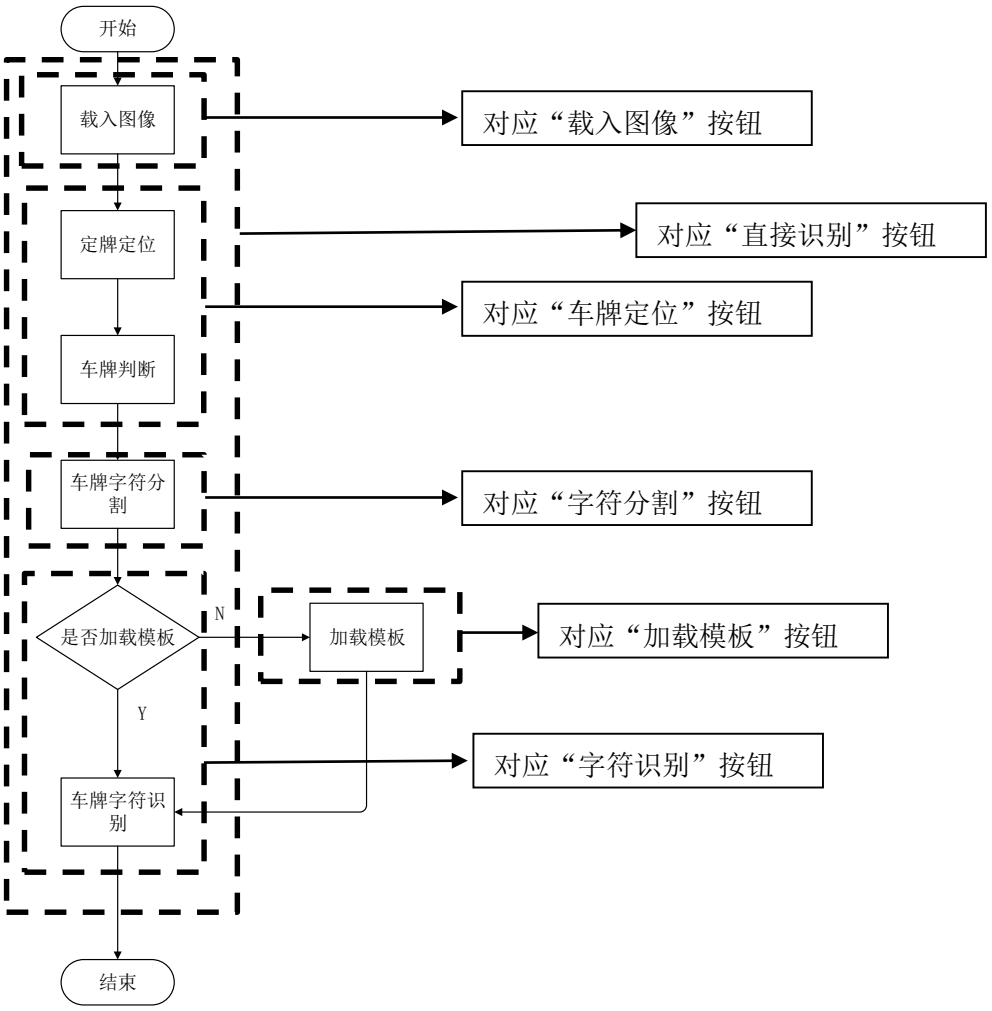

图 5- 11 界面设计模块程序流程图

“载入图像”、“车牌定位”、“字符分割”和“字符识别”四个按钮分别有各自的显示结果区域，前三者为图像显示控件，“字符识别”为文本框显示结果。并且在操作执行完毕后，清空列表释放内存。

界面设计模块共有七个报错提示，其中四个是功能错误报错提示，分别是当系统未能定位车牌即原始图像通过车牌定位模块后，返回值为 0 没有定位到图像时，报错为“没有定位出车牌!”；当系统在车牌判断模块后，筛选去所有可能车牌区域，返回值为 0 时，报错为“SVM 没有定位出车牌!”；当系统未能找到载入图片位置时，报错为“载入图像失败!”；当系统未能找到字符模板时，报错为“加载模板失败!”。另外三个为按钮提示报错，分别是当用户没有进行载入图像便先按车牌定位时，提示“请先载入图像!”；当用户没有进行车牌定位便先进行字符分割时，提示“请先车牌定位!”；当用户没有进行字符分割便先进行字符识别时，提示“请先字符分割!”；当用户没有加载模板便进行字符识别时，提示“请先加载模板!”。

第五节 系统测试和界面展示

一、 界面显示

本系统的界面展示结果如图 5- 12 空界面展示图：此界面为进入系统时的界面展示界面。

图 5- 12 空界面展示图

运行结果界面展示如图 5- 13 运行结果展示界面：此界面为输入图像后运行结果界面。

图 5- 13 运行结果展示界面

定牌定位出错界面显示如图 5- 14 出错提示界面：此界面为当系统运行出错时的提示界面。

图 5- 14 出错提示界面

未加载模板界面显示如图 5- 15 未加载模板提示界面：此界面为用户操作出错没有加载模板时显示的提示界面（其余用户出错提示界面类似）。

图 5- 15 未加载模板提示界面

二、 系统测试

针对不同的情况，对本系统进行一系列的系统测试，以判断系统是否对于现有系统进行了优化。

（1）基于边缘检测无法完成的多框架车牌情况，测试结果如图 5- 16 多框架车辆的车牌识别测试结果图：图中输入的原始车辆框架多，仅仅使用基于边缘检测的车牌定位方法无法定位的情况，可以从图中得知本系统可以对其进行定位识别。

图 5- 16 多框架车辆的车牌识别测试结果图

（2）基于图像色彩信息定位无法完成的蓝色汽车情况，测试结果如图 5- 17 蓝色车辆的车牌识别测试结果图：图中输入的原始车辆为蓝色与车牌区域颜色相同，仅仅使用基于图像色彩信息的车牌定位方式将无法定位的情况，可以从图中得知本系统可以对其进行定位识别。

图 5- 17 蓝色车辆的车牌识别测试结果图

（3）含有正视角平行车牌图像，测试结果如图 5- 18 正视角平行车辆的车牌识别测试结果图：图中输入图像为正视角车辆后部拍摄的车辆图像，其为最普通的输入图像，可以从图中得知本系统可以对其进行定位识别。

图 5- 18 正视角平行车辆的车牌识别测试结果图

（4）含有正视角倾斜车牌图像，测试结果如图 5- 19 正视角倾斜车牌测试结果图：图中输入图像为正视角的倾斜车牌车辆，其可能因为拍摄设备的倾斜导致车牌区域倾斜，由图中车牌图像可以系统将其进行了旋转摆正，并可以对其进行定位识别。

图 5- 19 正视角倾斜车牌测试结果图

（5）含有斜视角车牌图像，测试结果如图 5- 20 斜视角车辆的车牌识别测试结果图：图中输入图像为斜视角的车辆，即拍摄角度与车牌角度并不正对，可能为监控拍摄或移动设备拍摄的情况，由图中车牌图像可知系统对车牌区域进行了扭正，并且可以对其进行定位识别。

图 5- 20 斜视角车辆的车牌识别测试结果图

（6）夜间拍摄，测试结果如图 5- 21 夜间拍摄车辆的车牌识别测试结果图：图中输入图像为夜间拍摄的车辆图像，由图可知本系统可以对其进行了定位识别。

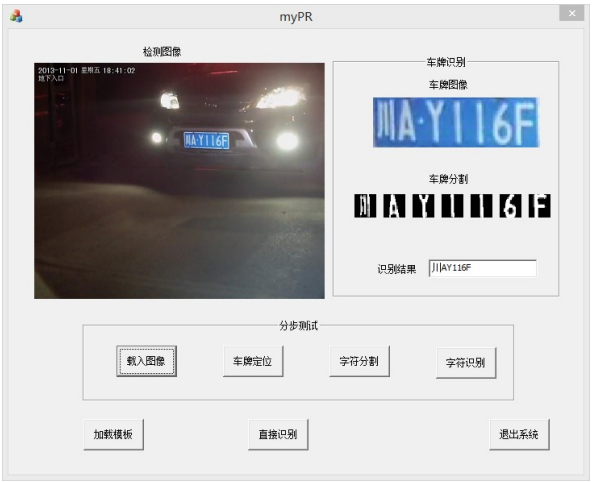

图 5- 21 夜间拍摄车辆的车牌识别测试结果图

三、系统错误结果及分析

（1）系统出错情况 1：由于偏转角度实在过大的情况如图 5- 22 车辆对拍摄视角偏转角度过大车辆的车牌识别测试结果图，图中输入图像为偏转角度过大的车辆图像，由图可知系统对其定位时进行扭正但是扭正效果不佳，所以无法准确识别。

图 5- 22 车辆对拍摄视角偏转角度过大车辆的车牌识别测试结果图

错误分析：在如图的情况中，由于车牌的偏转角度过大，在扭转的过程中会出现一定的信息缺失，所以在使用扭正的时候，并不能真正正确的将车牌区域扭转成正视角，车牌区域仍会有一定的倾斜，但是无法使用倾斜函数进行水平调整，因为车牌的底部是处于水平的，其倾斜是扭转造成的并不是因为拍摄时设备倾斜造成的。由于分割时的车牌字符是倾斜的，所以给识别时带来了难度，识别出错率便会增高，虽然能够用当前字符图像加入模板中，使得下次识别时正确，但是如何在大角度扭转时，还原车牌正视角信息将是之后的研究方向。

（2）系统出错情况 2：车牌图像在逆光中拍摄的情况如图 5- 23 逆光中拍摄的车辆的车牌识别测试结果图，图中输入图像为在逆光中拍摄的车辆图像，其中车牌区域的像素很低，由图可知系统对其无法进行定位。

图 5- 23 逆光中拍摄的车辆的车牌识别测试结果图

错误分析：在如图的情况中，系统并不能定位出车牌区域，并且可以知道是在车牌判断中 SVM 模型将所有车牌定位中筛选出的可能的车牌区域图像全部去除，其原因是因为在 SVM 预测时，是将车牌图像的直方图统计数据作为特征维度进行预测的，所以在逆光的情况下，车牌字符像素和车牌背景像素相近，导致在预测时出现错误，为了排除这种情况，可以在 SVM 模型训练时加入逆光车牌进行预测。但是在过度逆光的情况，此时车牌图像像素实在太低，便实在无法识别。

（3）系统出错情况 3：车牌区域污损太严重的情况如图 5- 24 车牌污损的车辆的车牌识别测试结果图，图中输入图像为车牌区域污损严重的车辆，其车牌中中文字符已经模糊看不清，背景也变成了灰色，由图可知系统无法对其进行准备的定位识别。

图 5- 24 车牌污损的车辆的车牌识别测试结果图

错误分析：在如图的情况中，车牌区域中中文字符的污损程度太严重，所以在识别时颜色无法识别，字符也无法识别，系统自动将这部分的车牌区域当作了车牌的背景被截去，这也导致了后来在分割的时候，系统先检测到的城市字符为“P”,“L”字符也因为被污损断节被忽略，在最后的车牌边缘被系统当作字符“X”识别出来。所以如何能够在污损的情况下正确的定位和分割出车牌将是未来的研究方向。

第七章 总结与展望

第一节 工作总结

车牌识别系统虽然早已实现，但依旧是计算机视觉和模式识别等领域的热门研究课题。随着信息智能化和大数据时代的广泛应用，智能化交通管理系统变的越来越普及，对车牌识别系统的要求也越来越高，不再满足于类似于收费口正面清晰的车牌识别系统，而是需要对道路监控、手机随拍的车牌原始图像进行车牌识别。本文从道路监控可能拍摄到的偏斜视角的车牌图像进行分析并优化传统识别系统，增加系统的鲁棒性和识别率。

本文所研究和实现的车牌识别系统工作共分为 5 个部分，分别为车牌定位、图像处理、车牌判断、车牌字符分割和车牌字符识别。

（1）车牌定位部分主要研究和分析传统车牌定位方式—基于边缘检测的定位方式和基于图像色彩信息的定位方式的原理，根据这两种的车牌定位方式的优劣和其互补性，分析和研究相结合的车牌定位方式。同时进行优化，将边框边缘识别改为字符边缘识别的方式增加准确性，并用二次 Sobel 算法边缘检测的方式增加系统的鲁棒性。

（2）图像处理部分主要研究和分析在车牌定位后对车牌进行一系列的操作，使得车牌能够统一尺寸并以正视角水平的格式输入到车牌判断的环节。其中针对道路交通摄像头或者移动拍摄设备拍摄到的非正视角的车牌进行了扭转操作以此来增加车牌识别系统的识别范围。

（3）车牌判断部分主要研究机器学习，利用 SVM 模型对车牌进行了判断，增加了车牌的识别准确性，减少了错误定位的几率。

（4）车牌字符分割部分主要研究和分析如何利用车牌的字符准确有效的对车牌进行分割，排除误割、多割和分割后包含杂质等错误情况。

（5）车牌字符识别部分主要研究对单个的字符进行识别，针对可能出现的错误识别的情况进行了分析并提出了可以改进的方案。

本文所研究的车牌识别系统，以 MFC 界面形式整合呈现，能够实现偏斜大视角的车牌的定位识别，也可以实现交通监控和移动设备拍摄等情况的车牌识别。

第二节 研究展望

虽然本文开发的车牌识别系统可以识别针对各种场合的车牌图像并且具有较高的识别率，但是系统在某些方面仍然存在一部分的问题，有待改进和优化，针对本系统中存在的不足提出以下几点建议：

（1）对于本系统车牌定位模块虽然通过两种定位相结合的方式对车牌位置进行定位提高了定位的准备性和识别率，减少了误识的情况也减少了之后

的工作量，但是由于对图像可能进行两次定位所以提高了系统运行的工作量同时也增加系统运行的速率。如何能够在多方位车牌定位的同时降低系统的运行时间和提高效率是一个有待进一步研究的问题。

（2）对于本系统中图像处理部分采用的扭正操作将斜视角的车牌转正的操作，虽然可以将车牌扭转到正视角但是扭转过来的图像仍会有一些失真和走形，增加了对于车牌字符识别部分的难度。大角度的偏转转正会造成之后的识别出错，所以如何在扭转视角的同时将图像进一步进行优化统一，方便车牌字符识别部分的工作是需要进一步研究的重点。

（3）对于本系统中字符识别的部分，因为识别率和字符模板的容量大小有关，但是大容量的模板又造成了运行速度的下降，在加载模板时一般需要1-2 秒，效率不高。若输入的字符图像为模糊和扭曲的情况还很容易出现错误，虽然能够手动在后台进行纠正，但是人机交互能力不强。所以系统可以在操作员的指示下自动纠正错误，而非手动在系统后台纠正错误，以及如何通过更优化快捷的方式进行字符识别的进一步需要研究的重点。

我国车牌识别系统的识别率仍然需要优化，本系统可以针对道路监控或移动设备的车牌图像进行识别，但是对于强光、大雾等自然情况下的车牌图像却无法正确识别。另外针对移动设备可能会造成的抖动等图像模糊也无法正确识别，所以如何能够将车牌识别系统的应用范围更加的广泛是进一步需要研究的重点。并且在增加系统性能的同时，如何可能降低时间成本提高系统效率也是进一步需要研究的重点。总体而言，车牌识别系统仍有广阔的研究空间和应用价值。

致　　谢

大学四年的时光马上就要进入了尾声，在大学的最后一段时光中我完成了这篇毕业论文和毕业设计，在这里我要感谢所有的老师和同学在这四年中带给我的指导以及在本次课题研究中带来的帮助。

尤其是最后毕业论文的阶段，我无数次遇到困难和瓶颈无法独自完成，导师都耐心的帮我解答并帮助我一起寻找可能的答案。从选题到最后的论文，导师都认真负责指出我存在的具体问题，循循善诱，给我带来了极大的帮助，在此我衷心感谢教导我的导师。

大学四年的时间里，在老师的悉心教导下和同学们的耐心帮助下，我在专业知识领域和生活的方方面面得到了提高和磨砺，他们不仅教会了我知识，更教会了我如何去学习知识，如何利用知识在社会中站稳脚跟。我知道我能有现在的一些成果和他们的教导和陪伴是分不开的，人无法独自一人成长，环境的磨砺以及陪伴者的浇灌，正是由老师们的谆谆教导和同学们的鼓励带给我最美好的雨露，帮助我成长。

最后，我要感谢我的同学、老师以及默默关心我的父母！

参考文献

【1】. 蒋钰,海云.基于 matlab 车牌识别系统[J]. 信息系统工程 , 2015.10.20;

【2】. 张涛，王剑魁，张国山等. 基于 Cortex-A8 的嵌入式车牌识别系统设计[J]. 电子设计工程. 2015，Vol.23 ,No.9;

【3】. 唐宏震. 车牌识别系统中字符识别的研究与实现[D]. 西北大学, 2008;

【4】. 冀国亮.基于 Android 平台的车牌识别系统研究与实现[D]. 2013..05.14;

【5】. 赵瑞.运动模糊车牌识别系统的设计与实现[D]. 2015.09.07;

【6】. 张涛，王剑魁，张国山等.多雾霾天气车牌识别系统图像预处理算法[N]. 山东理工大学学报. 2015, Vol.29 , No.1;

【7】. 朱江峰,刘苏,夏冰.基于云计算技术的车牌识别系统[J].科学前沿,2013, Vol.3;

【8】. 百度百科，车牌，http://baike.baidu.com/link?url=zRWsj8VyzBWRNOBiGUkBJxFpAOaiop7gTQs_4ZmW25RgihxliI_TLu9pg5YK1vBdxGI2EKJ4t9QmSRrpaNuz9_ ，2016.4.30（2016.5.1）;

【9】. 刘洋,张天翔,童亚拉.基于 Sobel 算子的图像边缘检测实验分析[J]. 计算机光盘软件与应用. 2012，Vol.7;

【10】. 百度文库. 边缘检测原理（内含三种算法）. http://wenku.baidu.com/link?url=cmG1-nMPz_q9xb36fQ8bWcveafhFrcevdPz5R6wxUqlrWXvRhojWQla4pd6ztx4bezuQeMdGOiL9LVWiOPJorBCTGUpLB2OczJ2COeIwe2i, 2013.06.19(2016.4.28);

【11】. 百度百科. 颜色模型，http://baike.baidu.com/link?url=s33pNw8dieV-J4UcJRgE5ADC3hFHPsEhh7LeSOIvEL8U8qrwNJTuMkxGaeXOzWE6nJVgJBQA_Yh9sDHzBxS38K， 2016.3.20（2016.5.3）;

【12】. 百度百科. 闭操作，http://baike.baidu.com/link?url=IHxTtbqfEcbYo-

YW2OhE_a9f72DZtigFz31v7KrqtUW9w4X7UKAIlgDLmiAh_Pg2G7z4KPWrRx7k-ns4i50xgq， 2016.3.21（2016.5.5）；

【13】. 百度文库. svm分类器原理. http://wenku.baidu.com/link?url=3el76V_63TKgY_sW1FDsPMFHvnX-PHg-oDngd4mev30h7qyfBHGV52rbPKxVp36gnvU_fx2NkCjEwxNSFuShxwPYJK_rb2fnCXXBtUy_sRy, 2012.03.05(2016.4.30);

【14】. LeftNotEasy. 机器学习中的算法（2）—支持向量机（SVM）基础, http://www.cnblogs.com/LeftNotEasy/archive/2011/05/02/basic-of-svm.html, 2011.05.02（2016.5.2）

【15】. 百度百科. 形态学, http://baike.baidu.com/link?url=yFT9wJFet_gvkTQYuuAMjQRsAXOrVECTrMUbmm_ZynAsQ49oaElLfVB-PUeKW6S18utfecF8sSxRiDgC5DuWua#3 , 2016.1.29（2016.5.5）；

【16】. 郭茹侠，李日财. 车牌识别系统[J]. 中文科技期刊数据库（文摘版）工程技术, 2015.07;

【17】. 尚晓波. 车牌识别系统中倾斜校正和字符识别的研究与实现[D]. 2015.03.13;

【18】. H Rajput ,T Som, S Kar. An Automated Vehicle License Plate Recognition System [J].Computer. 2015,Vol.48:56-61;

【19】. 石志享. 基于径向基函数神经网络的车牌识别技术研究[D]. 西安科技大学. 2014;

【20】. 郑雪. 基于图像处理的车牌识别研究[J]. 电脑知识与技术. 2015, Vol.19;

【21】. 傅建强. 复杂背景下车牌识别算法研究与应用[D]. 复旦大学. 2013;

【22】. 张献艺. 机动车车牌智能识别技术研究[D]. 山东理工大学. 2013;

【23】. 百度百科. cvFindContours, http://baike.baidu.com/link?url=E8s0D-

2zSzaW5Z08pr0uj2SqcDKVVqzq-
3L9_3Oex1j4Wo7UKEi9hrxwab6DgYQopDAQ3aQ8CCTDD1jvO0nkFK,2015.5.2
1(2016.3.28);

附录 1：部分源程序清单

第一节 车牌定位模块关键代码

一、基于颜色匹配的车牌定位

```
//! 根据一幅图像与颜色模板获取对应的二值图
//! 输入 RGB 图像, 颜色模板（蓝色、黄色）
//! 输出灰度图（只有 0 和 255 两个值，255 代表匹配，0 代表不匹配）
Mat CPlateLocate::colorMatch(const Mat& src, Mat& match, const Color
r, const bool adaptive_minsv) {
  const float max_sv = 255;
  const float minref_sv = 64;
  const float minabs_sv = 95;
  // blue 的 H 范围
  const int min_blue = 100;  // 100
  const int max_blue = 140;  // 140
  // yellow 的 H 范围
  const int min_yellow = 15;  // 15
  const int max_yellow = 40;  // 40
  // white 的 H 范围
  const int min_white = 0;   // 15
  const int max_white = 30;  // 40
  Mat src_hsv;
  cvtColor(src, src_hsv, CV_BGR2HSV);
  vector<Mat> hsvSplit;
  split(src_hsv, hsvSplit);
  equalizeHist(hsvSplit[2], hsvSplit[2]);
  merge(hsvSplit, src_hsv);
  //匹配模板基色,切换以查找想要的基色
  int min_h = 0;
  int max_h = 0;
  switch (r) {
    case BLUE:
      min_h = min_blue;
      max_h = max_blue;
      break;
    case YELLOW:
      min_h = min_yellow;
      max_h = max_yellow;
```

```
break;
   case WHITE:
     min_h = min_white;
     max_h = max_white;
     break;
   default:
     // Color::UNKNOWN
     break;
 }
 float diff_h = float((max_h - min_h) / 2);
 float avg_h = min_h + diff_h;
 int channels = src_hsv.channels();
 int nRows = src_hsv.rows;
 int nCols = src_hsv.cols * channels;
 if (src_hsv.isContinuous())
 {
   nCols *= nRows;
   nRows = 1;
 }
 int i, j;
 uchar* p;
 float s_all = 0;
 float v_all = 0;
 float count = 0;
 for (i = 0; i < nRows; ++i) {
   p = src_hsv.ptr<uchar>(i);
   for (j = 0; j < nCols; j += 3) {
     int H = int(p[j]);      // 0-180
     int S = int(p[j + 1]);  // 0-255
     int V = int(p[j + 2]);  // 0-255
     s_all += S;
     v_all += V;
     count++;
     bool colorMatched = false;
     if (H > min_h && H < max_h) {
       float Hdiff = 0;
       if (H > avg_h)
         Hdiff = H - avg_h;
       else
         Hdiff = avg_h - H;
       float Hdiff_p = float(Hdiff) / diff_h;
       float min_sv = 0;
    if (true == adaptive_minsv)
```

```
            min_sv =
                minref_sv -
                minref_sv / 2 *
                    (1 - Hdiff_p);
          else
            min_sv = minabs_sv;
          if ((S > min_sv && S < max_sv) && (V > min_sv && V < max_sv))
            colorMatched = true;
        }
        if (colorMatched == true) {
          p[j] = 0;
          p[j + 1] = 0;
          p[j + 2] = 255;
        } else {
          p[j] = 0;
          p[j + 1] = 0;
          p[j + 2] = 0;
        }
      }
    }
    Mat src_grey;
    vector<Mat> hsvSplit_done;
    split(src_hsv, hsvSplit_done);
    src_grey = hsvSplit_done[2];
    match = src_grey;
    return src_grey;
  }

  // !基于 HSV 空间的颜色搜索方法
  int CPlateLocate::colorSearch(const Mat& src, const Color r, Mat&
  out, vector<RotatedRect>& outRects, int index) {
    Mat match_grey;
    const int color_morph_width = 10;
    const int color_morph_height = 2;
    // 进行颜色查找
    colorMatch(src, match_grey, r, false);
    if (m_debug) {
         stringstream ss(stringstream::in | stringstream::out);
         ss << "image/tmp/debug_blue" <<".jpg";
         imwrite(ss.str(), match_grey);
    }
    Mat src_threshold;
threshold(match_grey, src_threshold, 0, 255, CV_THRESH_OTSU +
```

```
CV_THRESH_BINARY);
  Mat element = getStructuringElement(
      MORPH_RECT, Size(color_morph_width, color_morph_height));
  morphologyEx(src_threshold, src_threshold, MORPH_CLOSE, element);
  if (m_debug) {
     stringstream ss(stringstream::in | stringstream::out);
      ss << "image/tmp/color" <<".jpg";
      imwrite(ss.str(),src_threshold);
   //              utils::imwrite("resources/image/tmp/color.jpg",
src_threshold);
  }
  src_threshold.copyTo(out);
  // 查找轮廓
  vector<vector<Point>> contours;
  findContours(src_threshold,
               contours,
               CV_RETR_EXTERNAL,
               CV_CHAIN_APPROX_NONE);
  vector<vector<Point>>::iterator itc = contours.begin();
  while (itc != contours.end()) {
    RotatedRect mr = minAreaRect(Mat(*itc));
    //大小尺寸判断
    if (!verifySizes(mr))
      itc = contours.erase(itc);
    else {
      ++itc;
      outRects.push_back(mr);
    }
  }
  return 0;
}

// !基于颜色信息的车牌定位
int   CPlateLocate::plateColorLocate(Mat   src,   vector<CPlate>&
candPlates,
                                     int index) {
  vector<RotatedRect> rects_color_blue;
  vector<RotatedRect> rects_color_yellow;
  vector<CPlate> plates;
  Mat src_b;
  // 查找蓝色车牌
  // 查找颜色匹配车牌
  colorSearch(src, BLUE, src_b, rects_color_blue, index);
```

```
  // 进行抗扭斜处理
  deskew(src, src_b, rects_color_blue, plates);
  // 查找黄色车牌
  colorSearch(src, YELLOW, src_b, rects_color_yellow, index);
  deskew(src, src_b, rects_color_yellow, plates);
  for (size_t i = 0; i < plates.size(); i++) {
    candPlates.push_back(plates[i]);
  }
  return 0;
}
```

二、基于边缘检测的车牌定位

```
//! Sobel 第一次搜索
//! 不限制大小和形状，获取的 BoundRect 进入下一步
int CPlateLocate::sobelFrtSearch(const Mat& src,
                                 vector<Rect_<float>>& outRects) {
  Mat src_threshold;
  sobelOper(src,         src_threshold,         m_GaussianBlurSize,
m_MorphSizeWidth,
            m_MorphSizeHeight);
  vector<vector<Point>> contours;
  findContours(src_threshold,
               contours,
               CV_RETR_EXTERNAL,
               CV_CHAIN_APPROX_NONE);
  vector<vector<Point>>::iterator itc = contours.begin();
  vector<RotatedRect> first_rects;
  while (itc != contours.end()) {
    RotatedRect mr = minAreaRect(Mat(*itc));
    //大小尺寸判断
    if (verifySizes(mr)) {
      first_rects.push_back(mr);
      float area = mr.size.height * mr.size.width;
      float r = (float)mr.size.width / (float)mr.size.height;
      if (r < 1) r = (float)mr.size.height / (float)mr.size.width;
      /*cout << "area:" << area << endl;
      cout << "r:" << r << endl;*/
    }
    ++itc;
  }
  for (size_t i = 0; i < first_rects.size(); i++) {
```

```
    RotatedRect roi_rect = first_rects[i];
    Rect_<float> safeBoundRect;
    if (!calcSafeRect(roi_rect, src, safeBoundRect)) continue;
    outRects.push_back(safeBoundRect);
  }
  return 0;
}

//! Sobel 第二次搜索,对断裂的部分进行再次的处理
//! 对大小和形状做限制，生成参考坐标
int CPlateLocate::sobelSecSearchPart(Mat& bound, Point2f refpoint,
vector<RotatedRect>& outRects) {
  Mat bound_threshold;
  sobelOperT(bound, bound_threshold, 3, 6, 2);
  Mat tempBoundThread = bound_threshold.clone();
  clearLiuDingOnly(tempBoundThread);
  int posLeft = 0, posRight = 0;
  if (bFindLeftRightBound(tempBoundThread, posLeft, posRight)) {
    if (posRight != 0 && posLeft != 0 && posLeft < posRight) {
      int posY = int(bound_threshold.rows * 0.5);
      for (int i = posLeft + (int)(bound_threshold.rows * 0.1);
           i < posRight - 4; i++) {
        bound_threshold.data[posY * bound_threshold.cols + i] = 255
      }
    }
    for (int i = 0; i < bound_threshold.rows; i++) {
      bound_threshold.data[i * bound_threshold.cols + posLeft] = 0;
      bound_threshold.data[i * bound_threshold.cols + posRight] =
0;
    }
  }
  vector<vector<Point>> contours;
  findContours(bound_threshold,
               contours,
               CV_RETR_EXTERNAL,
               CV_CHAIN_APPROX_NONE);
  vector<vector<Point>>::iterator itc = contours.begin();
  vector<RotatedRect> second_rects;
  while (itc != contours.end()) {
    RotatedRect mr = minAreaRect(Mat(*itc));
    second_rects.push_back(mr);
    ++itc;
  }
```

```
  for (size_t i = 0; i < second_rects.size(); i++) {
    RotatedRect roi = second_rects[i];
    if (verifySizes(roi)) {
      Point2f refcenter = roi.center + refpoint;
      Size2f size = roi.size;
      float angle = roi.angle;
      RotatedRect refroi(refcenter, size, angle);
      outRects.push_back(refroi);
    }
  }
  return 0;
}
```

```
//! Sobel 运算//对图像分割，腐蚀和膨胀的操作
//! 输入彩色图像，输出二值化图像
int CPlateLocate::sobelOper(const Mat& in, Mat& out, int blurSize,
int morphW,int morphH) {
  Mat mat_blur;
  mat_blur = in.clone();
  GaussianBlur(in,  mat_blur,  Size(blurSize,  blurSize),  0,  0,
BORDER_DEFAULT);
  Mat mat_gray;
  if (mat_blur.channels() == 3)
    cvtColor(mat_blur, mat_gray, CV_RGB2GRAY);
  else
    mat_gray = mat_blur;
  // equalizeHist(mat_gray, mat_gray);
  int scale = SOBEL_SCALE;
  int delta = SOBEL_DELTA;
  int ddepth = SOBEL_DDEPTH;
  Mat grad_x, grad_y;
  Mat abs_grad_x, abs_grad_y;
  Sobel(mat_gray,  grad_x,  ddepth,  1,  0,  3,  scale,  delta,
BORDER_DEFAULT);
  convertScaleAbs(grad_x, abs_grad_x);
  Mat grad;
  addWeighted(abs_grad_x, SOBEL_X_WEIGHT, 0, 0, 0, grad);
  Mat mat_threshold;
  double otsu_thresh_val =
      threshold(grad,  mat_threshold,  0,  255,  CV_THRESH_OTSU  +
CV_THRESH_BINARY);
```

```
// 腐蚀和膨胀
  Mat element = getStructuringElement(MORPH_RECT, Size(morphW,
morphH));
  morphologyEx(mat_threshold, mat_threshold, MORPH_CLOSE, element);
  out = mat_threshold;
  return 0;
}

//! Sobel 运算
//! 输入彩色图像，输出二值化图像
int CPlateLocate::sobelOperT(const Mat& in, Mat& out, int blurSize,
int morphW, int morphH) {
  Mat mat_blur;
  mat_blur = in.clone();
  GaussianBlur(in, mat_blur, Size(blurSize, blurSize), 0, 0,
BORDER_DEFAULT);
  Mat mat_gray;
  if (mat_blur.channels() == 3)
    cvtColor(mat_blur, mat_gray, CV_BGR2GRAY);
  else
    mat_gray = mat_blur;
  int scale = SOBEL_SCALE;
  int delta = SOBEL_DELTA;
  int ddepth = SOBEL_DDEPTH;
  Mat grad_x, grad_y;
  Mat abs_grad_x, abs_grad_y;
  Sobel(mat_gray, grad_x, ddepth, 1, 0, 3, scale, delta,
BORDER_DEFAULT);
  convertScaleAbs(grad_x, abs_grad_x);
  Mat grad;
  addWeighted(abs_grad_x, 1, 0, 0, 0, grad);
  Mat mat_threshold;
  double otsu_thresh_val =
      threshold(grad, mat_threshold, 0, 255, CV_THRESH_OTSU +
CV_THRESH_BINARY);
  Mat element = getStructuringElement(MORPH_RECT, Size(morphW,
morphH));
  morphologyEx(mat_threshold, mat_threshold, MORPH_CLOSE, element);
  out = mat_threshold;
  return 0;
}
```

```
// !基于垂直线条的车牌定位
int CPlateLocate::plateSobelLocate(Mat src, vector<CPlate>&
candPlates,int index) {
  vector<RotatedRect> rects_sobel;
  vector<RotatedRect> rects_sobel_sel;
  vector<CPlate> plates;
  vector<Rect_<float>> bound_rects;
  sobelFrtSearch(src, bound_rects);
  vector<Rect_<float>> bound_rects_part;
  for (size_t i = 0; i < bound_rects.size(); i++) {
    float fRatio = bound_rects[i].width * 1.0f /
bound_rects[i].height;
    if (fRatio < 3.0 && fRatio > 1.0 && bound_rects[i].height < 120)
{
      Rect_<float> itemRect = bound_rects[i];
      //宽度过小，进行扩展
      itemRect.x = itemRect.x - itemRect.height * (4 - fRatio);
      if (itemRect.x < 0) {
        itemRect.x = 0;
      }
      itemRect.width = itemRect.width + itemRect.height * 2 * (4 -
fRatio);
      if (itemRect.width + itemRect.x >= src.cols) {
        itemRect.width = src.cols - itemRect.x;
      }
      itemRect.y = itemRect.y - itemRect.height * 0.08f;
      itemRect.height = itemRect.height * 1.16f;
      bound_rects_part.push_back(itemRect);
    }
  }
  for (size_t i = 0; i < bound_rects_part.size(); i++) {
    Rect_<float> bound_rect = bound_rects_part[i];
    Point2f refpoint(bound_rect.x, bound_rect.y);
    float x = bound_rect.x > 0 ? bound_rect.x : 0;
    float y = bound_rect.y > 0 ? bound_rect.y : 0;
    float width =
        x + bound_rect.width < src.cols ? bound_rect.width :
src.cols - x;
    float height =
        y + bound_rect.height < src.rows ? bound_rect.height :
src.rows - y;
    Rect_<float> safe_bound_rect(x, y, width, height);
    Mat bound_mat = src(safe_bound_rect);
```

```
sobelSecSearchPart(bound_mat, refpoint, rects_sobel);
  }
  for (size_t i = 0; i < bound_rects.size(); i++) {
    Rect_<float> bound_rect = bound_rects[i];
    Point2f refpoint(bound_rect.x, bound_rect.y);
    float x = bound_rect.x > 0 ? bound_rect.x : 0;
    float y = bound_rect.y > 0 ? bound_rect.y : 0;
    float width =
        x + bound_rect.width < src.cols ? bound_rect.width :
src.cols - x;
    float height =
        y + bound_rect.height < src.rows ? bound_rect.height :
src.rows - y;
    Rect_<float> safe_bound_rect(x, y, width, height);
    Mat bound_mat = src(safe_bound_rect);
    sobelSecSearch(bound_mat, refpoint, rects_sobel);
  }
  Mat src_b;
  sobelOper(src, src_b, 3, 10, 3);
  deskew(src, src_b, rects_sobel, plates);
  for (size_t i = 0; i < plates.size(); i++)
  candPlates.push_back(plates[i]);
  return 0;
}
```

三、图像处理

```
//! 对 minAreaRect 获得的最小外接矩形，用纵横比进行判断
bool CPlateLocate::verifySizes(RotatedRect mr) {
  float error = m_error;
  float aspect = m_aspect;
  int min = 34 * 8 * m_verifyMin;  // minimum area
  int max = 34 * 8 * m_verifyMax;  // maximum area
  // Get only patchs that match to a respect ratio.
  float rmin = aspect - aspect * error;
  float rmax = aspect + aspect * error;
  float area = mr.size.height * mr.size.width;
  float r = (float)mr.size.width / (float)mr.size.height;
  if (r < 1) r = (float)mr.size.height / (float)mr.size.width;
  if ((area < min || area > max) || (r < rmin || r > rmax))
    return false;
  else
```

```
 return true;
}

//! 旋转操作
bool CPlateLocate::rotation(Mat& in, Mat& out, const Size rect_size,
const Point2f center, const double angle) {
  Mat in_large;
  in_large.create(int(in.rows * 1.5), int(in.cols * 1.5),
in.type());
  float x = in_large.cols / 2 - center.x > 0 ? in_large.cols / 2 -
center.x : 0;
  float y = in_large.rows / 2 - center.y > 0 ? in_large.rows / 2 -
center.y : 0;
  float width = x + in.cols < in_large.cols ? in.cols : in_large.cols
- x;
  float height = y + in.rows < in_large.rows ? in.rows :
in_large.rows - y;
  /*assert(width == in.cols);
  assert(height == in.rows);*/
  if (width != in.cols || height != in.rows) return false;
  Mat imageRoi = in_large(Rect_<float>(x, y, width, height));
  addWeighted(imageRoi, 0, in, 1, 0, imageRoi);
  Point2f center_diff(in.cols / 2.f, in.rows / 2.f);
  Point2f new_center(in_large.cols / 2.f, in_large.rows / 2.f);
  Mat rot_mat = getRotationMatrix2D(new_center, angle, 1);
  Mat mat_rotated;
  warpAffine(in_large, mat_rotated, rot_mat, Size(in_large.cols,
in_large.rows), CV_INTER_CUBIC);
  Mat img_crop;
  getRectSubPix(mat_rotated,                Size(rect_size.width,
rect_size.height),
  new_center, img_crop);
  out = img_crop;
  return true;
}

//! 是否偏斜
//! 输入二值化图像，输出判断结果
bool CPlateLocate::isdeflection(const Mat& in, const double angle,
double& slope) {
  int nRows = in.rows;
  int nCols = in.cols;
  assert(in.channels() == 1);
```

```
int comp_index[3];
  int len[3];
  comp_index[0] = nRows / 4;
  comp_index[1] = nRows / 4 * 2;
  comp_index[2] = nRows / 4 * 3;
  const uchar* p;
  for (int i = 0; i < 3; i++) {
    int index = comp_index[i];
    p = in.ptr<uchar>(index);
    int j = 0;
    int value = 0;
    while (0 == value && j < nCols) value = int(p[j++]);
    len[i] = j;
  }
  double maxlen = max(len[2], len[0]);
  double minlen = min(len[2], len[0]);
  double difflen = abs(len[2] - len[0]);
  double PI = 3.14159265;
  double g = tan(angle * PI / 180.0);
  if (maxlen - len[1] > nCols / 32 || len[1] - minlen > nCols / 32)
{
    double slope_can_1 =
        double(len[2] - len[0]) / double(comp_index[1]);
    double   slope_can_2   =   double(len[1]   -   len[0])   /
double(comp_index[0]);
    double   slope_can_3   =   double(len[2]   -   len[1])   /
double(comp_index[0]);
    slope  =  abs(slope_can_1  -  g)  <=  abs(slope_can_2  -  g)  ?
slope_can_1: slope_can_2;
    return true;
  } else {
    slope = 0;
  }
  return false;
}

//! 扭变操作
void CPlateLocate::affine(const Mat& in, Mat& out, const double
slope) {
  Point2f dstTri[3];
  Point2f plTri[3];
  float height = (float)in.rows;
  float width = (float)in.cols;
```

```
  float xiff = (float)abs(slope) * height;
  if (slope > 0) {
    plTri[0] = Point2f(0, 0);
    plTri[1] = Point2f(width - xiff - 1, 0);
    plTri[2] = Point2f(0 + xiff, height - 1);
    dstTri[0] = Point2f(xiff / 2, 0);
    dstTri[1] = Point2f(width - 1 - xiff / 2, 0);
    dstTri[2] = Point2f(xiff / 2, height - 1);
  } else {
    plTri[0] = Point2f(0 + xiff, 0);
    plTri[1] = Point2f(width - 1, 0);
    plTri[2] = Point2f(0, height - 1);
    dstTri[0] = Point2f(xiff / 2, 0);
    dstTri[1] = Point2f(width - 1 - xiff + xiff / 2, 0);
    dstTri[2] = Point2f(xiff / 2, height - 1);
  }
  Mat warp_mat = getAffineTransform(plTri, dstTri);
  Mat affine_mat;
  affine_mat.create((int)height, (int)width, TYPE);
  if (in.rows > HEIGHT || in.cols > WIDTH)
    warpAffine(in, affine_mat, warp_mat, affine_mat.size(),
    CV_INTER_AREA);
  else
    warpAffine(in,   affine_mat,   warp_mat,   affine_mat.size(),
CV_INTER_CUBIC);
  out = affine_mat;
}

//! 计算一个安全的 Rect
//! 如果不存在，返回 false
bool CPlateLocate::calcSafeRect(const RotatedRect& roi_rect, const
Mat& src,Rect_<float>& safeBoundRect) {
  Rect_<float> boudRect = roi_rect.boundingRect();
  float tl_x = boudRect.x > 0 ? boudRect.x : 0;
  float tl_y = boudRect.y > 0 ? boudRect.y : 0;
  float br_x = boudRect.x + boudRect.width < src.cols ? boudRect.x
+ boudRect.width - 1 : src.cols - 1; float br_y = boudRect.y +
boudRect.height < src.rows? boudRect.y + boudRect.height - 1 :
src.rows - 1;
  float roi_width = br_x - tl_x;
  float roi_height = br_y - tl_y;
  if (roi_width <= 0 || roi_height <= 0) return false;
  safeBoundRect = Rect_<float>(tl_x, tl_y, roi_width, roi_height);
```

```
  return true;
}
```

第二节 车牌判断模块关键代码

```
CPlateJudge::CPlateJudge()
{
    m_path = "model/svm.xml";
    m_getFeatures = getHistogramFeatures;
    LoadModel();
}

void CPlateJudge::LoadModel(string s)
{
    svm.clear();
    svm.load(s.c_str(), "svm");
}

//! 对单幅图像进行 SVM 判断
int CPlateJudge::plateJudge(const Mat& inMat,int& result)
{
    if (m_getFeatures == NULL)
        return -1;
    Mat features;
    m_getFeatures(inMat, features);
    Mat p = features.reshape(1, 1);
    p.convertTo(p, CV_32FC1);
    int response = (int)svm.predict(p);
    result = response;
    return 0;
}

//! 对多幅车牌进行 SVM 判断
int CPlateJudge::plateJudge(const vector<CPlate>& inVec,
    vector<CPlate>& resultVec)
{
    int num = inVec.size();
  for (int j = 0; j < num; j++) {
    CPlate inPlate = inVec[j];
    Mat inMat = inPlate.getPlateMat();
    int response = -1;
    plateJudge(inMat, response);
```

```
    if (response == 1)
      resultVec.push_back(inPlate);
    else {
      int w = inMat.cols;
      int h = inMat.rows;
      Mat tmpmat = inMat(Rect_<double>(w * 0.05, h * 0.1, w * 0.9,
h * 0.8));
      Mat tmpDes = inMat.clone();
      resize(tmpmat, tmpDes, Size(inMat.size()));
      plateJudge(tmpDes, response);
      if (response == 1) resultVec.push_back(inPlate);
    }
  }
    return 0;
}
```

第三节 车牌字符分割模块关键代码

```
//! 找出指示城市的字符的 Rect
int CCharsSegment::GetSpecificRect(const vector<Rect>& vecRect)
{
    vector<int> xpositions;
    int maxHeight = 0;
    int maxWidth = 0;
    for (int i = 0; i < vecRect.size(); i++)
    {
        xpositions.push_back(vecRect[i].x);
        if (vecRect[i].height > maxHeight)
        {
            maxHeight = vecRect[i].height;
        }
        if (vecRect[i].width > maxWidth)
        {
            maxWidth = vecRect[i].width;
        }
    }
    int specIndex = 0;
    for (int i = 0; i < vecRect.size(); i++)
    {
        Rect mr = vecRect[i];
        int midx = mr.x + mr.width/2;
    if ((mr.width > maxWidth * 0.8 || mr.height > maxHeight *
```

```
0.8) &&(midx < int(m_theMatWidth / 7) * 2 && midx > int(m_theMatWidth
/ 7) * 1))
        {
            specIndex = i;
        }
    }
    return specIndex;
}

//! 根据特殊车牌来构造猜测中文字符的位置和大小
Rect CCharsSegment::GetChineseRect(const Rect rectSpe)
{
    int height = rectSpe.height;
    float newwidth = rectSpe.width * 1.15;
    int x = rectSpe.x;
    int y = rectSpe.y;
    int newx = x - int (newwidth * 1.15);
    newx = newx > 0 ? newx : 0;
    if(newx +newwidth > x )
    {
        newwidth = x-newx;
    }
    Rect a(newx, y, int(newwidth), height);
    return a;
}

//去除车牌上方的钮钉
//计算每行元素的阶跃数，如果小于 X 认为是铆钉，将此行全部填 0（涂黑）
Mat CCharsSegment::clearLiuDing(Mat img)
{
    const int x = m_LiuDingSize;
    Mat jump = Mat::zeros(1, img.rows, CV_32F);
    for(int i=0; i < img.rows; i++)
    {
        int jumpCount = 0;
        for(int j=0; j < img.cols-1; j++)
        {
            if (img.at<char>(i,j) != img.at<char>(i,j+1))
                jumpCount++;
        }
        jump.at<float>(i) = jumpCount;
    }
    for(int i=0; i < img.rows; i++)
```

```
    {
        if(jump.at<float>(i) <= x)
        {
            for(int j=0; j < img.cols; j++)
            {
                img.at<char>(i,j) = 0;
            }
        }
    }
    return img;
}

//! 字符分割与排序
int CCharsSegment::charsSegment(Mat input, vector<Mat>& resultVec)
{
    if( !input.data )
    { return -3; }
    int plateType = getPlateType(input);
    cvtColor(input, input, CV_RGB2GRAY);
    Mat img_threshold;
    if (1 == plateType)
        threshold(input,        img_threshold,        10,        255,
CV_THRESH_OTSU+CV_THRESH_BINARY);
    else
        threshold(input,        img_threshold,        10,        255,
CV_THRESH_OTSU+CV_THRESH_BINARY_INV);
    if(m_debug)
    {
        stringstream ss(stringstream::in | stringstream::out);
        ss << "image/tmp/debug_char_threshold" << ".jpg";
        imwrite(ss.str(), img_threshold);
    }
    clearLiuDing(img_threshold);
    if(m_debug)
    {
        stringstream ss(stringstream::in | stringstream::out);
        ss << "image/tmp/debug_char_clearLiuDing" << ".jpg";
        imwrite(ss.str(), img_threshold);
    }
    Mat img_contours;
    img_threshold.copyTo(img_contours);
    vector< vector< Point> > contours;
    findContours(img_contours,
```

```
    contours,
    CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_NONE);
vector<vector<Point> >::iterator itc= contours.begin();
vector<Rect> vecRect;
while (itc != contours.end())
{
    Rect mr = boundingRect(Mat(*itc));
    Mat auxRoi(img_threshold, mr);
    if (verifySizes(auxRoi))
    {
        vecRect.push_back(mr);
    }
    ++itc;
}

if (vecRect.size() == 0)
    return -3;
vector<Rect> sortedRect;
SortRect(vecRect, sortedRect);
int specIndex = 0;
specIndex = GetSpecificRect(sortedRect);
if(m_debug)
{
    if (specIndex < sortedRect.size())
    {
        Mat specMat(img_threshold, sortedRect[specIndex]);
        stringstream ss(stringstream::in | stringstream::out);
        ss << "image/tmp/debug_specMat" <<".jpg";
        imwrite(ss.str(), specMat);
    }
}
Rect chineseRect;
if (specIndex < sortedRect.size())
    chineseRect = GetChineseRect(sortedRect[specIndex]);
else
    return -3;
if(m_debug)
{
    Mat chineseMat(img_threshold, chineseRect);
    stringstream ss(stringstream::in | stringstream::out);
    ss << "image/tmp/debug_chineseMat" <<".jpg";
    imwrite(ss.str(), chineseMat);
```

```
    }
    vector<Rect> newSortedRect;
    newSortedRect.push_back(chineseRect);
    RebuildRect(sortedRect, newSortedRect, specIndex);
    if (newSortedRect.size() == 0)
        return -3;
    for (int i = 0; i < newSortedRect.size(); i++)
    {
        Rect mr = newSortedRect[i];
        Mat auxRoi(img_threshold, mr);
        if (1)
        {
            auxRoi = preprocessChar(auxRoi);
            if(m_debug)
            {
               stringstream ss(stringstream::in | stringstream::out);
                ss << "image/tmp/debug_char_auxRoi_" << i <<".jpg";
                imwrite(ss.str(), auxRoi);
            }
            resultVec.push_back(auxRoi);
        }
    }
    return 0;
}
```

第四节 车牌字符识别模块关键代码

```
//处理图像，得到归一化的图像
void procImage(IplImage* inimage,IplImage** outimage)
{
    IplImage* grayimg = NULL;
    rgb2gray(inimage,&grayimg);
    IplImage* bwimg= cvCreateImage(cvGetSize(grayimg), 8, 1);
    ThresholdOtsu(grayimg,bwimg);
    int left,right,top,bottom;
    //左边界
    for (int i=0;i<bwimg->width;i++)
    {
        double sumvalue = 0;
        for(int j=0;j<bwimg->height;j++)
        {
```

```
            double value = cvGetReal2D(bwimg,j,i);
            sumvalue= sumvalue+value;
        }
        if(sumvalue > 0.1)
        {
            left = i;//左边界
            break;
        }

    }
    //右边界
    for (int i=bwimg->width-1; i>=0; i--)
    {
        double sumvalue = 0;
        for(int j=0;j<bwimg->height;j++)
        {
            double value = cvGetReal2D(bwimg,j,i);
            sumvalue= sumvalue+value;
        }
        if(sumvalue > 0.1)
        {
            right = i;//右边界
            break;
        }
    }
    //上边界
    for (int i=0;i<bwimg->height;i++)
    {
        double sumvalue = 0;
        for(int j=0;j<bwimg->width;j++)
        {
            double value = cvGetReal2D(bwimg,i,j);
            sumvalue= sumvalue+value;
        }
        if(sumvalue > 0.1)
        {
            top = i;//上边界
            break;
        }

    }
    //下边界
    for (int i=bwimg->height-1; i>=0; i--)
```

```
    {
        double sumvalue = 0;
        for(int j=0;j<bwimg->width;j++)
        {
            double value = cvGetReal2D(bwimg,i,j);
            sumvalue= sumvalue+value;
        }
        if(sumvalue > 0.1)
        {
            bottom = i;//下边界
            break;
        }
    }
    CvRect r;
    r.x = left;
    r.y = top;
    r.height = bottom-top+1;
    r.width = right-left+1;
    cvSetImageROI(bwimg,r);
    *outimage = cvCreateImage( cvGetSize(bwimg),8, 1);
    cvCopy(bwimg,*outimage);
    cvResetImageROI(bwimg);
    CvSize siz;
    siz.width = DEST_WIDTH;
    siz.height = DEST_HEIGHT;
    imageResize(outimage,siz);
    IplImage* bwimgcrop = cvCreateImage(cvGetSize(*outimage), 8, 1);
    ThresholdOtsu(*outimage,bwimgcrop);
    imageReplace(bwimgcrop,outimage);
    cvReleaseImage(&grayimg);;
    cvReleaseImage(&bwimg);
}

//单个切割出的数字图像识别
//type 0 汉字识别
//type 1 英文字母识别
//type >=2 英文数字识别
int CmyPRDlg::imagerecogn(IplImage* image,int type)
{
    IplImage *procimg = NULL;
    procImage(image,&procimg);
    list<IMGFEATURE>::iterator iter;
    int minindex = -1;
```

```
    double minvalue = DBL_MAX;
    if (type == 0 || type >= 2)
    {
        if(type == 0)
        {
            for (iter=m_chineselist.begin(); iter != m_chineselist.end();iter++)
            {
                IMGFEATURE tmp = *iter;
                IplImage *mouldimage = tmp.image;
                double value = 0;
                for (int row=0;row<procimg->height;row++)
                {
                    for (int col=0;col<procimg->width;col++)
                    {
                        double value1 = cvGetReal2D(mouldimage, row, col);
                       double value2 = cvGetReal2D(procimg, row, col);
                        if(value1 != value2)
                        {
                            value++;
                        }
                    }
                }
                if(value < minvalue)
                {
                    minvalue = value;
                    minindex = tmp.typeindex;
                }
            }
        }
        else
        {
            for (iter=m_char2list.begin(); iter != m_char2list.end();iter++)
            {
                IMGFEATURE tmp = *iter;
                IplImage *mouldimage = tmp.image;
                double value = 0;
                for (int row=0;row<procimg->height;row++)
                {
                    for (int col=0;col<procimg->width;col++)
                    {
```

```
                    double value1 = cvGetReal2D(mouldimage, row,
col);
                   double value2 = cvGetReal2D(procimg, row, col);
                    if(value1 != value2)
                    {
                        value++;
                    }
                }
            }
            if(value < minvalue)
            {
                minvalue = value;
                minindex = tmp.typeindex;
            }
        }
    }

    }
    else if (type == 1)
    {
        for (iter=m_char2list.begin(); iter != 
m_char2list.end();iter++)
        {
            IMGFEATURE tmp = *iter;
            if(tmp.typeindex < 10)
            {
                continue;
            }
            IplImage *mouldimage = tmp.image;
            double value = 0;
            for (int row=0;row<procimg->height;row++)
            {
                for (int col=0;col<procimg->width;col++)
                {
                    double value1 = cvGetReal2D(mouldimage, row, col);
                    double value2 = cvGetReal2D(procimg, row, col);//
                    if(value1 != value2)
                    {
                        value++;
                    }
                }
            }
            if(value < minvalue)
```

```
    {
                minvalue = value;
                minindex = tmp.typeindex;
            }
        }
    }
    cvReleaseImage(&procimg);
    return minindex;
}
```