**基于OpenCV的车牌检测系统**

**摘要：**车牌识别是指带有车牌的图片经过特定系统的处理以字符形式输出车牌信息，是计算机视觉领域的一部分。本文研究内容的便是实现这样一个系统，主要包括车牌位置检测，字符识别两个部分。OpenCV是一个广受欢迎的开源计算机视觉库，提供了完善的图像处理函数，使用OpenCV可以大大节省开发时间加快开发效率，本系统的便是在OpenCV的基础上进行开发。系统在位置检测与字符分割部分取得了极高的准确度，字符识别部分准确率还有待提高。

**关键字：车牌识别，OpenCV，计算机视觉**

**Abstract：**Vehicle license plate Recognition is a part of computer vision field, which is a kind of picture with license plate, which is processed by special system to output license plate information in character form. The content of this paper is to realize such a system, mainly including license plate location detection, character recognition two parts. OpenCV is a popular open source computer Vision Library, providing a perfect image processing function, using OPENCV can greatly save development time and speed up development efficiency, this system is based on the OpenCV development. The system obtains the extremely high accuracy in the position detection and the character segmentation part, the character recognition part accuracy rate still needs to improve.

**Keywords: vehicle license plate recognition, OpenCV, computer vision**

**目 录**

[1 前言 4](#_Toc515895579)

[1.1 车牌识别介绍 4](#_Toc515895580)

[1.2 车牌识别应用领域 4](#_Toc515895581)

[1.3 实现方法简介 4](#_Toc515895582)

[1.4 开发环境 5](#_Toc515895583)

[2 方案论证 6](#_Toc515895584)

[2.1 车牌定位方案论证 6](#_Toc515895585)

[2.1.1 基于车牌颜色分布的定位方法 6](#_Toc515895586)

[2.1.2 基于Radon变换的车牌定位方法 6](#_Toc515895587)

[2.1.3 基于边缘检测的车牌定位方法 7](#_Toc515895588)

[2.2 字符识别方案论证 8](#_Toc515895589)

[2.2.1 基于神经网络的字符识别 8](#_Toc515895590)

[2.2.2 基于KNN最邻近算法的字符识别 9](#_Toc515895591)

[2.3 方案比较与选择 10](#_Toc515895592)

[3 OpenCV 11](#_Toc515895593)

[3.1 OpenCV简介 11](#_Toc515895594)

[3.2 OpenCV常用模块介绍 11](#_Toc515895595)

[3.2.1 CV模块 12](#_Toc515895596)

[3.2.2 MLL统计分类器 15](#_Toc515895597)

[3.2.3 HIGH GUI 16](#_Toc515895598)

[4 车牌位置检测 18](#_Toc515895599)

[4.1 图像预处理 18](#_Toc515895600)

[4.1.1 RGB空间与HSV空间的转换 18](#_Toc515895601)

[4.1.2 图像滤波 19](#_Toc515895602)

[4.1.1 图像二值化 20](#_Toc515895603)

[4.2 轮廓特征提取 21](#_Toc515895604)

[4.2.1 边缘检测 21](#_Toc515895605)

[4.2.2 轮廓筛选 22](#_Toc515895606)

[4.2.3 轮廓提取与分组 23](#_Toc515895607)

[4.3 车牌切割 25](#_Toc515895608)

[5 基于KNN的字符识别 26](#_Toc515895609)

[5.1 KNN简介 26](#_Toc515895610)

[5.2 字符预处理 26](#_Toc515895611)

[5.2.1 字符分割 27](#_Toc515895612)

[5.2.2 图像归一化 27](#_Toc515895613)

[5.3 样本训练 27](#_Toc515895614)

[5.3.1 贪心算法 27](#_Toc515895615)

[5.3.2 训练过程 28](#_Toc515895616)

[5.4 KNN算法的字符识别 31](#_Toc515895617)

[6 系统功能、指标参数 33](#_Toc515895618)

[6.1 系统功能 33](#_Toc515895619)

[6.2 指标参数 34](#_Toc515895620)

[7 总结与展望 35](#_Toc515895621)

[8 致谢 37](#_Toc515895622)

[9 参考文献 38](#_Toc515895623)

[附录：关键源代码 39](#_Toc515895624)

# 前言

## 车牌识别介绍

随着大数据与人工智能领域的发展，计算机视觉技术逐步得到应用，车牌识别作为其中一个炙手可热的应用项目，天生拥有广阔的市场与发展前景，从而吸引了大量企业从事相关研发。

车牌识别一般指车牌识别系统（vehicle license plate recognition，VLPR），即通过摄像头采集车辆信息，通过对视频图像的处理得到车牌信息，从而达到识别车辆的目的，是计算机视觉在图像处理识别方面的一种应用。

## 车牌识别应用领域

作为使用最为广泛的交通工具，汽车的身影无处不在，车牌识别的应用同样无处不在，以下例举了几个常见的应用。

不停车电子收费系统（Electronic Toll Collection，ETC），通过在高速公路出入口安装ETC系统大大加快了车辆通行速度，有效缓解高速公路收费处的拥堵问题，节省大量人力，并能通过系统记录行程信息，便于随时查证。

停车场出入管理系统，传统停车场出入由人工给卡并人工计算停车时间，不仅耗费人力，速度还很慢。应用停车出入管理系统自动识别车辆，配合缴费系统便可以实现无人操作。

超速违章处罚，通过在高速路安装超速监测系统，能够自动监测超速车辆并识别车辆信息，从而加大威慑力减少违法行为。通过遍布的摄像头能够迅速定位犯罪分子的车辆位置加快破案效率。

总之，车牌识别已经成为人类离不开的一项技术，使得车辆的监督管理变得极为便利，因此提高车牌识别的准确率，加快识别速度，降低成本等变得极为重要，虽然这一领域的实际应用已经比较成熟，但依然存在较大的进步空间，依旧存在很大的研究价值，本设计便是本着这样的态度进行研究希望能够理清车牌识别的逻辑并寻求突破。

## 实现方法简介

车牌识别系统的实现主要是两个方面，车牌检测与字符识别。车牌检测即车牌图像定位，在一副完整图像中找到车牌位置并进行分割。车牌检测部分即对分割得到的车牌部分进行字符识别得到车牌号。

车牌图像定位是车牌识别中至关重要的一步，由于字符识别的输入正好是车牌图像定位的输出结果，所以车牌图像定位的准确与否直接关系到最后结果的正确与否，并且需要考虑到周围环境的影响。车牌图像定位主要用到图像变换与图像形态学处理等知识进行处理。对图像进行预处理实现灰度变换以及滤波除去噪声等操作，利用车牌部分明显的方形轮廓作为特征从而实现车牌位置的查找。

车牌识别部分使用K最邻近（K-NearestNeighbor）分类算法，为数据挖掘分类算法中的一种，通过先验知识提取车牌中各个字符特征记录与XML文件中，在识别时提取上一步车牌图像定位得到切割图像的特征进行特征对比，选取与模板特征最邻近的特征群所对应的字符为结果字符。

## 开发环境

本设计代码使用Visual Studio2012进行编辑，以下简称VS。VS是微软公司推出的开发工具集，所编写的代码适用于Windows所有平台，具有极其完善的功能，包括了整个软件生命周期中所需要的大部分工具，开发环境如图 1.1所示。

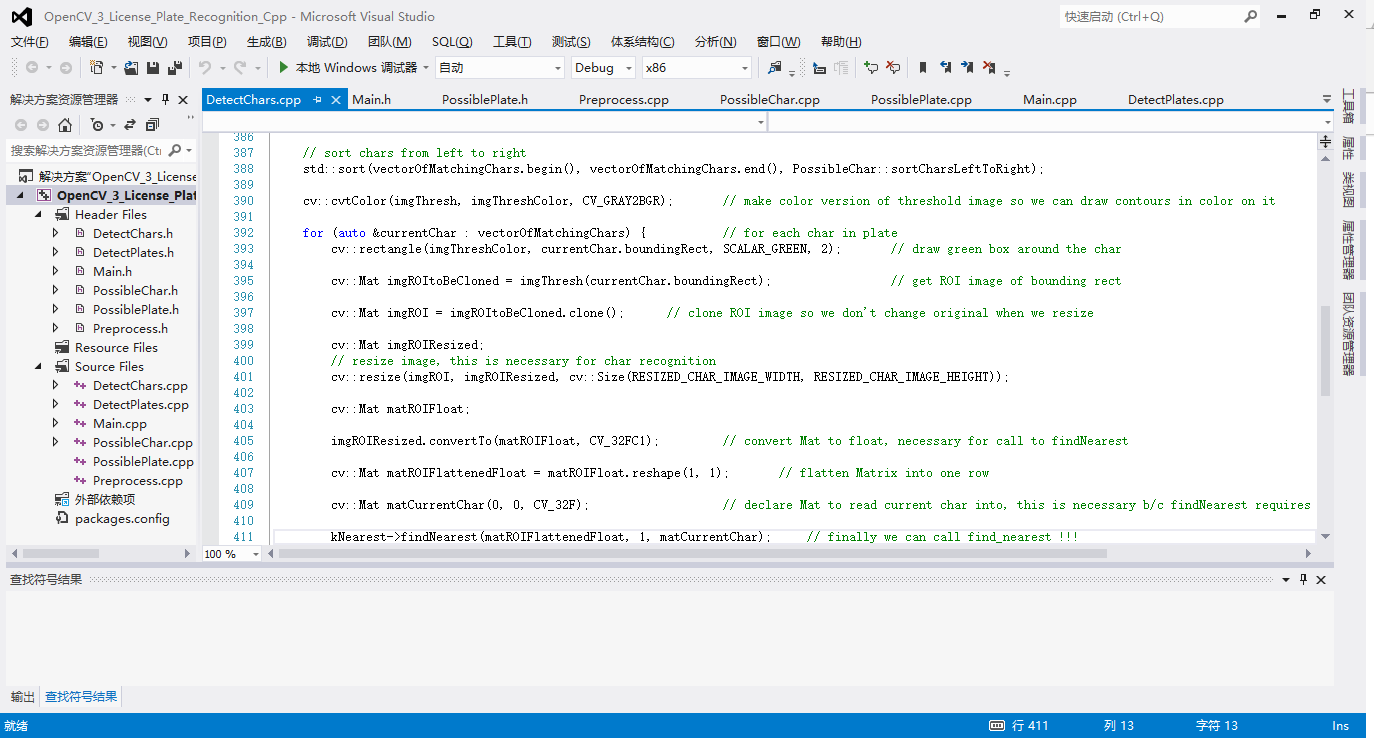


图 1.1 开发环境界面

在Visual Studio2012环境下配置OpenCV3.0的主要步奏包括：

1. OpenCV库的下载与安装
2. Windows环境变量的添加
3. Visual Studio调用文件链接的添加

# 方案论证

## 车牌定位方案论证

### 基于车牌颜色分布的定位方法

由于汽车车牌颜色对比度很高且颜色相对固定，例如普通车牌大多是白底黑字或者蓝底黄字，车牌与字符，车牌与车声的颜色差异较大，即可以采用区域生长法的方法对彩色图像进行分割。

获取的汽车图像大多是RGB格式，处理起来比较麻烦，所以需要将图片从RGB空间转换到HSV空间。在HSV空间中只需要色度（Hue）和饱和度（Saturation）两个分量便可以对车牌颜色分布进行计算。基于车牌颜色分布的定位流程如图 2.1所示。



图 2.1 基于车牌颜色分布定位流程

此方法减少了车牌的漏检情况，在车牌倾斜情况下也有很高的检测准确度，提高了检测的准确性与稳定性。但是不适合极端天气下的检测，在车牌与背景对比度较低的情况下检测的准确性与速度都有所下降，而且由于车牌的颜色不一致，比如小轿车是蓝色背景，货车是黄色背景，除此之外还有很多其它不同的车牌，所以情况较多加大了检测的难度。

### 基于Radon变换的车牌定位方法

Radon变换在数学上指一种积分变换，在图像处理领域主要用于对倾斜的图象进行复原。现实生活中车牌都是矩形的，拥有固定不变的形状，边缘检测后车牌的长宽相对集中，其中长宽之间的距离相对固定，边缘都由一定比例的线段组成。但由于拍摄角度的原因，导致成像后的车牌发生变形扭曲，并非得到正面严格垂直角度的图像，与预定的特征不符，使用Radon变换算法首先将图像在各个方向进行投影得到投影后的图像信息，通过对投影后的图像做特定的分析得到原始图像的偏移角度，然后通过对图像进行旋转，拉伸，变形等操作对偏移角度进行纠正得到矩形或者近似矩形的车牌图像。基于Radon算法的车牌角度纠正定位方法流程如图 2.2所示。



图 2.2基于Radon变换的车牌定位流程图

### 基于边缘检测的车牌定位方法

边缘指图像中色彩变化较大的分界部分，灰度变换之后的图像边缘即为灰度值急剧变化的像素区域，边缘是图像的基本特征，车牌图像更是明显，车牌的字符与车牌背景之间，车牌与车声部分都存在明显的边缘，所以边缘检测是车牌检测中最常用的方法之一，在车牌图像定位中占据重要的地位。边缘检测的实现方法是通过使用某种算子来提取图像中的物体与边缘交界部分，常见的算子有sobel算子，拉普拉斯边缘检测算子，prewitt算子，roberts算子等[5]。

边缘检测的基本思想是基于边缘部分与背景具有较大的对比度，即差分值较大，通过检测发现当前点在临近区域为局部最大值且差值大于所规定的阈值，则判定当前点为边界点，众多点连在一起便形成了物体边缘的轮廓。车牌边缘检测便是通过这样的原理找到车牌位置，实现定位。基于边缘检测的车牌定位流程如图 2.3所示。



图 2.3 基于边缘检测的车牌定位流程

此方法具有检测速度快，精准度与效率高，且对噪声有很好的适应能力等优点，所以适用范围广，对于一张图片中有多个车牌的情况有也有很好的效果。但是对于车牌破损，边缘有污渍的车辆检测效果大大下降，对于背景复杂的情况，可能存在有类似于车牌的边缘，错把一些具有类似性质的其他标志物当做车牌导致定位错误。

## 字符识别方案论证

### 基于神经网络的字符识别

作为机器学习中的热门算法，神经网络模拟人类大脑的工作原理，以神经元为基本单位建立复杂的神经网络系统。人工神经网络结构如图 2.4[5]所示，由细胞体，树突，轴突构成的细胞体组成。

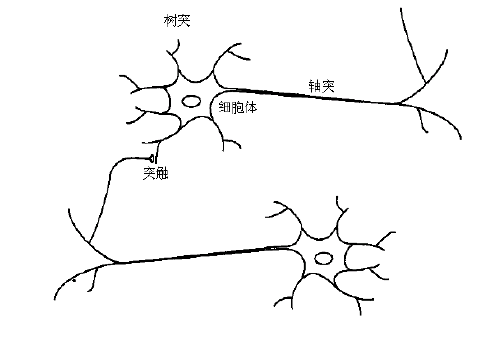


图 .4神经网络结构

抽象后的神经元数学模型如图2.5所示，输入信号P经过w的加权后进过累加器与b相加，进过函数f的处理最终输出a。图 2.5[5]表示的为单层神经网络模型，实际运用中常常使用单层神经网络的级联形式即多层神经网络模型。

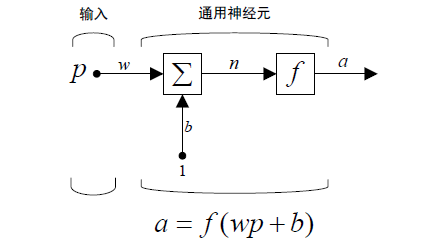


图 .5神经元数学模型

多层神经网络分为三层，输入层，隐含层，输出层。通常来说，隐含层层数越多，神经网络的分析能力越强，训练的数据也更多，多层神经网络模型如图 2.6所示。神经网络的训练即为对模型中各个参数的不断修正的过程，首先批量输入学习样本并对输入量进行归一化处理，并对最大训练次数，学习精度，隐含层节点数，初始权值，阈值，学习速率进行初始化，然后计算各层已经最后的输出情况，计算输出层误差从而作为反馈对参数进行调整直到误差减小到允许的范围[8]。

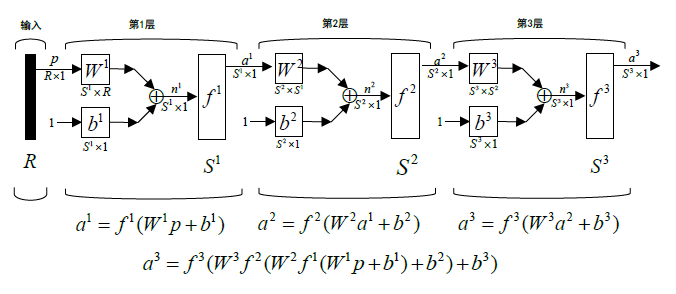


图 .6 多层神经网络模型

### 基于KNN最邻近算法的字符识别

KNN（K-Nearest Neighbor）属于非参数方法，在非参数估计（nonparametric estimation）中，我们只假设相似的输入具有相似的输出，这是一种合理的假设：世界是平稳的，并且无论密度，判别式还是回归函数都是缓慢地变化[2]。KNN算法则是这种思想的具体体现，通过计算判定样本与训练样本集的距离，在指定距离内涵盖最多样本所属的类别即为需要判定样本所属类别，如图 2.7选取k值，计算以欧氏距离k为半径的圆内其他类别的个数，图中中心小红点以k为半径的圆内三角形个数最多，则判定中心小红点为三角形。



图 .7 KNN示意图

KNN算法为惰性算法，不需要和神经网络一样训练参数得到判别函数，而是直接将训练集记录下来在判别的时候依次比较，所以此算法的训练成本较低，但是在数据量很大的时候使用knn算法有很大的计算量，所以一般用在训练量不大的地方。KNN算法可以归纳为以下几步：

1. 选择邻近的数量K和距离度量方法[1]。
2. 找到待分类样本的K个最近邻居[1]。
3. 根据最邻近的类标进行多数投票[1]。

## 方案比较与选择

通过对以上主流车牌方法识别的研究学习，发现各种方法均有相应的优点与缺点，考虑到车牌普遍拥有一样的方形结构并且车牌内有比较规律的字符，所以最终决定采用基于轮廓提取的定位方法查找车牌的位置。本次车牌识别主要对象限于只有字母与数字的车牌，所以仅有26个字母与10个数字，也就是在36个类别中进行匹配，所以类别并不是很多，若采用神经网络的方法进行识别会加大训练难度，所以最终选择了KNN算法实现车牌字符识别。综上，最终选择的方案为轮廓提取+K-邻近算法。

# OpenCV

## OpenCV简介

OpenCV(Open Source Computer Vision Library)是一个开源的计算机视觉与机器学习库。其建立的目的是为了提供一个通用的计算机视觉库与加速机器学习在通用商业产品中应用[6]。由于其使用BSD产品协议（一个非常宽松的协议），使得在商业应用中可以自由地使用与修改OpenCV。最开始有因特尔公司出资赞助开发，如今由Willow Garage提供支持。

OpenCV有超过2500中可操作算法，其中包括了全套的经典与最新的计算机视觉与机器学习算法[6]。这些算法可以用于检测，识别人脸，或者其他的物体，在视频中识别人类的行为，在相机功能中追踪移动的物体，提取物体的3D模型，操控立体视频中的点，拼割图像从而产生高分辨率的全景图像，在图像库中找到相同的图像，使用flash功能达到消除红眼的目的。OpenCV有着超过四万七千的开发者社区与超过一千四百万的下载量，被大量地用于公司，研发团队与政府部门。

伴随着许多高科技公司，例如谷歌，雅虎，微软，因特尔，IBM，宏达，Toyota对OpenCV技术人员的招聘，从而产生了许多类似Applied Minds，Zeitera，和VideoSurf之类的应用。使得OpenCV的社区更为活跃，OpenCV的应用从对街景图片的拼接到以色列国防安全的监测，到中国的矿井监测，又或者是Willow Garage公司的机器人项目，欧洲的泳池安全监测工作等等，OpenCV的应用遍布世界各处各个领域。

OpenCV同时具有C++,Python，Java，Matble的接口，支持Windows，Linux，Android以及Mac OS等平台。其应用主要依靠于实时视觉应用，必要的时候也会使用MMX以及SSE指令。全功能的CUDA和OPENCL接口正在开发之中，有超过500种算法以及10倍多的函数及组件用于支持这些算法。

## OpenCV常用模块介绍

OpenCV的基本结构主要包括5个模块，分别为图像处理和视觉算法，统计分类器（MLL），图像和水平输入输出，基本机构与算法，CnAux。其结构关系如图 3.1所示[5]。



图 .1OpenCV结构关系

首先，CV模块是图像处理时最常用的基本算法，用于对图像进行一般的数字形态学处理。MLL（Machine Learning Library）即统计分类器，随着大数据与计算机硬件速度的发展，机器学习也取得了重大的突破，OpenCV3.0新加入了很多机器学习算法用于图像识别与分类，这也是OpenCV更新最为频繁的一个模块。HighGUI则是针对视频图像的读取显示部分的，能够方便地读取与显示视频图像[5]。CXCORE包含了OpenCV基本的数据结构，动态数据库，绘图函数，数组操作相关函数，辅助功能与系统函数的宏[5]。结构图2.1中没有包括第五个模块CnAux，该模块的作用是包含即将淘汰的算法与结构。

从OpenCV出现至今已经更新了多个版本，本实验采用OpenCV3.0进行操作，相对于OpenCV2.X大体保留了OpenCV2经典的C++和Python编程接口风格，添加了Python3.X的支持，优化了Java的支持，并新接入了MATLAB的支持。对图像存储结构进行了调整，添加了一些新的图像处理算法与机器学习模型，新版本包括了目标检测算法，鱼眼镜头模型等全新算法。还增加了许多更高层次的高级封装，比如车牌检测，人脸检测等。相比之前新优化了更多指令集，值得一提的是不仅对Intel平台的处理器指令进行了优化，还增加了对ARM指令集的支持，使得在嵌入式设备进行处理时速度有了一定的提升。

### CV模块

CV模块可以说是OpenCV中使用最为广泛也是最基础的一个模块，在OpenCV对C++的支持库中单独以CV命名空间用于存储相关算法与数据结构，其中包含了大量的函数与类。

CV模块以namespace cv的方式给出，顾名思义，所有的计算机视觉算法都包含于此，这也是更新最为频繁的一个模块。从命名空间的角度可以将整个OpenCV库进行一个整体的归纳，可以说整个OpenCV库正是由这些命名空间组成得到。OpenCV命名空间分布结构如表3‑1。

表‑1 OpenCV命名空间分布结构

|  |  |
| --- | --- |
| 主命名空间（所有） | 子命名空间（部分） |
| cv | [bgsegm](https://docs.opencv.org/3.0.0/df/d5d/namespacecv_1_1bgsegm.html)，[bioinspired](https://docs.opencv.org/3.0.0/d2/d81/namespacecv_1_1bioinspired.html)，[bridge](https://docs.opencv.org/3.0.0/dd/d0e/namespacecv_1_1bridge.html)，[ccalib](https://docs.opencv.org/3.0.0/d7/d94/namespacecv_1_1ccalib.html)，[cuda](https://docs.opencv.org/3.0.0/d1/d1a/namespacecv_1_1cuda.html)，[cudacodec](https://docs.opencv.org/3.0.0/d0/de6/namespacecv_1_1cudacodec.html)，[cudev](https://docs.opencv.org/3.0.0/df/d1d/namespacecv_1_1cudev.html)，[datasets](https://docs.opencv.org/3.0.0/d0/d44/namespacecv_1_1datasets.html)，[detail](https://docs.opencv.org/3.0.0/d2/daf/namespacecv_1_1detail.html)，[directx](https://docs.opencv.org/3.0.0/d9/d12/namespacecv_1_1directx.html)，[Error](https://docs.opencv.org/3.0.0/d1/d0d/namespacecv_1_1Error.html)，[face](https://docs.opencv.org/3.0.0/d4/d48/namespacecv_1_1face.html)，[fisheye](https://docs.opencv.org/3.0.0/d2/d14/namespacecv_1_1fisheye.html)…………. |
| cvflann | [anyimpl](https://docs.opencv.org/3.0.0/d8/d7d/namespacecvflann_1_1anyimpl.html)，[lsh](https://docs.opencv.org/3.0.0/d4/daf/namespacecvflann_1_1lsh.html) |
| cvv | [impl](https://docs.opencv.org/3.0.0/d7/d84/namespacecvv_1_1impl.html) |
| matlab | 无 |
| NcvCTrep | 无 |
| tinyxml2 | 无 |

由表3-1可以清楚地看到cv模块是整个库中最庞大的部分，cv命名空间中又包含了大量子命名空间，为了方便管理，许多新增的计算机视觉算法便以这种新的形式加入，例如前面提到的鱼眼算法便以新的命名空间fishey加入到库中。

如果说cv模块是OpenCV库中使用最频繁的一个模块，那么mat类则是cv模块中使用最频繁的部分，说到OpenCV则不得不提及mat类，无论是对图像的读取，处理，传递都离不开mat类。mat类用于存储图像，其以高度抽象的结构可以涵盖n维图像，无论是单通道图片或者多通道图像都能直接读取。其子类信息如图 3.2所示。

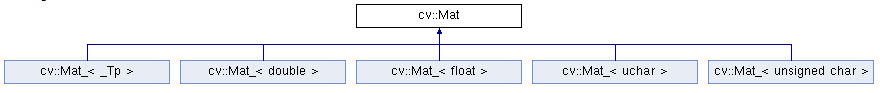


图 .2 mat类继承关系

数字图像都是由像素组成，以灰度图像为例，则是由二维矩阵表示，彩色（多通道）图像由三维矩阵表示。即图像由很多点组成，每个点可以由数字表示其颜色的深浅，mat类即是使用了这种思想用矩阵的方式存储图像。相比于早期OpenCV中使用Ipllmage或者Cvmat结构来存储图像，mat类新增了内存管理功能，开发者不用关心内存的申请与释放过程，这使得开发者的精力更能集中在功能的实现上，大大加快了开发的效率。Mat类的定义与关键的属性如下面代码所示。

1. **class** CV\_EXPORTS Mat
2. {
3. **public**:
4. //一系列通用的公有函数
5. ......
6. /\* flag参数中包含许多关于矩阵的信息，如：
7. -Mat 的标识
8. -数据是否连续
9. -深度
10. -通道数目
11. \*/
12. **int** flags;
13. //矩阵的纬度，取值通常为2
14. **int** dims;
15. //用于存储矩阵的行列信息，如果矩阵超过二维，则直接将取值设为-1
16. **int** rows,cols;
17. //指向数据的指针
18. uchar\* data;
20. //指向引用计数的指针
21. //如果数据是由用户分配的，则为NULL
22. **int**\* refcount;
23. //其他成员变量和成员函数
24. ...
25. };

以下为mat类常用的构造函数：

* Mat::Mat() 无参数构造方法
* Mat::Mat(int rows,int cols,int type)

创建行数为rows，列数为cols，类型为type的图像

* Mat::Mat(Size size,int type) 创建大小为size，类型为type的图像
* Mat::Mat(int rows,int cols,int type,const Scalar& s)

新建图像，rows表示图像的行数，cols表示图像的列数，type表示图像的类型，s为图像的初始值；

* Mat::Mat(Size size,int type,const Scalar& s)

新建图像，Size表示图像的大小，type表示图像的类型，s为图像的初始值；

* Mat::Mat(const Mat& m)

图像赋值，将原图像m赋值给新图像，此操作不对图像进行复制，新图像与原图像共用原图像空间；

* Mat::Mat(int rows,int cols,int type,void\* data,size\_t step=AUTO\_STEP)

新建图像，rows表示图像的行数，cols表示图像的列数，data表示图像数据位置，step表示图像的步长；

* Mat::Mat(Size size,int type,void\* data,size\_t step=AUTO\_STEP)
* Mat::Mat(const Mat& m,const Range& rowRange,const Range& colRange)

创建的图像源自已有图像m，图像范围由rowRange与colRange指定，此操作不对图像进行复制，新图像与原图像共用原图像空间[6]；

* Mat::Mat(const Mat& m,const Rect& roi)

创建新图像，新图像来自m图像的一部分，roi指定新图像范围，此操作不对图像进行复制，新图像与原图像共用原图像空间；

### MLL统计分类器

Machine learning模块作为新添加的模块越来越占据着重要的位置，其由一系列的函数与类组成，用于统计学分类，线性回归，簇等机器学习算法。OpenCV不断在更新对机器学习算法的支持，OpenCV中关于机器学习提供的支持如表3‑2所示。

表‑2 OpenCV MLL模块总览

|  |  |
| --- | --- |
| 功能及算法 | 简述 |
| 训练数据 | OpenCV专用的训练数据结构，位于cv::ml::TrainData |
| 朴素贝叶斯分类器 | 机器学习算法之一，位于[cv::ml::NormalBayesClassifier](https://docs.opencv.org/3.0.0/d4/d8e/classcv_1_1ml_1_1NormalBayesClassifier.html) |
| K-邻近算法 | 机器学习算法之一，位于[cv::ml::KNearest](https://docs.opencv.org/3.0.0/dd/de1/classcv_1_1ml_1_1KNearest.html) |
| 支持向量机 | 机器学习算法之一，位于[[cv::ml::SVM](https://docs.opencv.org/3.0.0/d1/d2d/classcv_1_1ml_1_1SVM.html)](https://docs.opencv.org/3.0.0/dd/de1/classcv_1_1ml_1_1KNearest.html) |
| 决策树 | 机器学习算法之一，位于[cv::ml::DTrees](https://docs.opencv.org/3.0.0/d8/d89/classcv_1_1ml_1_1DTrees.html) |
| 聚类 | 机器学习算法之一，位于[cv::ml::Boost](https://docs.opencv.org/3.0.0/d6/d7a/classcv_1_1ml_1_1Boost.html) |
| 随机树 | 机器学习算法之一，位于[cv::ml::RTrees](https://docs.opencv.org/3.0.0/d0/d65/classcv_1_1ml_1_1RTrees.html) |
| 期望最大化 | 机器学习算法之一，位于[cv::ml::EM](https://docs.opencv.org/3.0.0/d1/dfb/classcv_1_1ml_1_1EM.html) |
| 神经网络 | 机器学习算法之一，位于[cv::ml::ANN\_MLP](https://docs.opencv.org/3.0.0/d0/dce/classcv_1_1ml_1_1ANN__MLP.html) |
| 逻辑回归 | 机器学习算法之一，位于[cv::ml::LogisticRegression](https://docs.opencv.org/3.0.0/d6/df9/classcv_1_1ml_1_1LogisticRegression.html) |

与上表对应，其中每个算法都对应着一个类，用于对相关数据的存储，以及配应着相应的处理函数，MLL模块包含的类如下：

1. **class**   cv::ml::ANN\_MLP
2. Artificial Neural Networks - Multi-Layer Perceptrons.
4. **class**   cv::ml::Boost
5. Boosted tree classifier derived from DTrees.
7. **class**   cv::ml::DTrees
8. The **class** represents a single decision tree or a collection of decision trees.
10. **class**   cv::ml::EM
11. The **class** implements the Expectation Maximization algorithm.
13. **class**   cv::ml::KNearest
14. The **class** implements K-Nearest Neighbors model.
16. **class**   cv::ml::LogisticRegression
17. Implements Logistic Regression classifier.
19. **class**   cv::ml::NormalBayesClassifier
20. Bayes classifier **for** normally distributed data.
22. **class**   cv::ml::ParamGrid
23. The structure represents the logarithmic grid range of statmodel parameters.
25. **class**   cv::ml::RTrees
26. The **class** implements the random forest predictor.
28. **class**   cv::ml::StatModel
29. Base **class** **for** statistical models in OpenCV ML.
31. **class**   cv::ml::SVM
32. Support Vector Machines.
34. **class**   cv::ml::TrainData
35. Class encapsulating training data.

### HIGH GUI

当OpenCV被定位为一个通用计算机视觉库被设计之初就曾考虑是否需要设计兼容市面上通用的UI框架（比如QT,WinForms或者Cocoa），或者完全舍弃对UI界面的设计。由于在调试程序时经常需要快速得到可视化的处理结果，这便是HIGH GUI模块被设计的原因。

HIGH GUI为一下功能提供了简易的接口：

* 创建可以显示图像的计算机窗口，并记录对窗口图像的操作
* 追踪来自鼠标与键盘的操作并作出相应的回应

HIGH GUI模块下面又分三个模块，依次为[OpenGL support](https://docs.opencv.org/3.0.0/df/d24/group__highgui__opengl.html)，[Qt New Functions](https://docs.opencv.org/3.0.0/dc/d46/group__highgui__qt.html)，[C API](https://docs.opencv.org/3.0.0/d0/d28/group__highgui__c.html)。由各个模块名字很容易地联想到各个模块的功能及用处，由于本次设计都在Windows 10环境中使用，所以用到的都是[OpenGL support](https://docs.opencv.org/3.0.0/df/d24/group__highgui__opengl.html)中提供的功能，另外两个分别是针对QT环境与只支持C语言的环境中使用而建立的。[OpenGL support](https://docs.opencv.org/3.0.0/df/d24/group__highgui__opengl.html)中包含的函数如下：

1. **void**    cv::imshow (**const** String &winname, **const** ogl::Texture2D &tex)
2. **void**    cv::setOpenGlContext (**const** String &winname)
3. Sets the specified window as current OpenGL context.
5. **void**    cv::setOpenGlDrawCallback (**const** String &winname, OpenGlDrawCallback onOpenGlDraw, **void** \*userdata=0)
6. Sets a callback function to be called to draw on top of displayed image.
8. **void**    cv::updateWindow (**const** String &winname)
9. Force window to redraw its context and call draw callback ( setOpenGlDrawCallback ).

其中最常用到的自然是cv::imshow（）函数，方便的图像显示函数使得试验者能够轻松地观察处理后的图像结果，使得功能的调试效率极大提高。

# 车牌位置检测

## 图像预处理

图像预处理是车牌提取中的重要环节，通过输入设备输入的通常为多通道的彩色图像，数据量过大不利于图像的处理，图像的预处理包括图像格式空间转换，图像的二值化，图像的滤波。

### RGB空间与HSV空间的转换

RGB色彩模式主要面向硬件描述，是基于三原色原理的一种描述方式，通过红，黄，蓝三种颜色按不同比例混合可以合成各式各样的颜色，实际应用中输入设备采集的图像正是这种格式，但不便于人感官的辨认，比如我们看一种颜色，并不知道它由什么什么比例的三原色混合得到。HSV(hue，saturation，value)色彩空间则是面向用户 的一种颜色空间，也称作六角椎体模型，由色调H，饱和度S，强度V组成。颜色模型如图 4.1所示。

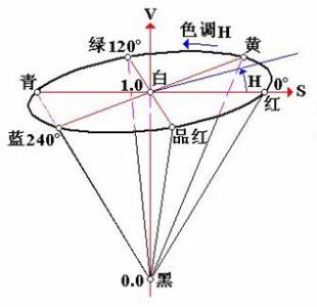


图 .1 HSV颜色模型（图像来源于网络）

当图像处于RGB空间时数据量太大且不利于对图像的处理，转化为HSV空间后直接取V(Value)通道便得到对应的灰度图像，图像处理时主要对灰度图像进行处理，转换后的结果如图 4.2所示。



图 .2 灰度变换后的汽车图像

### 图像滤波

图像滤波指在尽量保留图片特征信息的条件下对目标的噪声进行滤除，是图像处理中重要的部分。由于灰度图像中存在很多细小的颗粒噪声，如果直接进行二值化处理不仅会增加计算量，还会提取到很多噪声信息不利于后面的轮廓提取，进行滤波处理后可以滤除细小的颗粒物与复杂背景中的细微信息，利于后续对图像的处理。

数学形态学是图像处理中最常用也是最基本的算法，此处理方法的基本思想为用具有一定形态或形状的结构元素去提取和度量图像中对应的形状，从而实现分析图片、识别图片与处理图片的目的[16]。其数学基础为集合论，完备的数学基础是形态学用于图像分析处理，形态滤波器等领域的坚实后盾。数学形态学主要有四种基本操作：膨胀，腐蚀，开运算，闭运算。由这四种基本运算延伸出了很多新的运算，利于骨架抽取，击中与击不中变换，灰值形态学梯度，top-hat变换，颗粒分析，流域变换等。形态学运算的主要功能为：（1）图像的预处理（去除噪声，简化形状）；（2）图像增强（改善图像质量，丰富图像信息）；（3）物体检测（从背景中分割兴趣区域）；（4）物体量化描述（周长，面积，投影）[5]。

膨胀操作使得物体轮廓向外扩张，若物体中存在细小空洞或者某些边缘有细小的断口，可以使用膨胀操作对断口进行填补[5]。腐蚀操作是膨胀的反操作，会去掉物体的边缘点，细小物体会直接被认定为边缘点而被消除。腐蚀操作将图像轮廓边缘缩小，从而达到除去孤立的细小噪声点，使得轮廓平滑[5]。

对图像进行先膨胀再腐蚀的操作称为闭运算，相反，对图像进行先腐蚀再膨胀的操作称为开运算。开运算用于消除孤立的小物体，在纤细点处分离物体，平滑物体轮廓并不明显改变其面积[5]。顶帽操作是原图像与开运算结果之差，黑帽操作为闭运算结果与开运算之差。噪声无处不在，通常来说，在噪声的影响下，二值化后的图像边缘往往并不平滑，存在着由噪声引起的断层或者小孔，灵活地使用开闭运算能够对此类情况进行改善。本文真是使用了此类方法进行滤波处理，通过顶帽与黑帽操作达到降低鼓励噪声，平滑轮廓曲线，填补细小空洞，链接邻域的目的。

高斯滤波是一种线性平滑滤波，用于消除高斯噪声，本实验中采取的滤波方式正是高斯滤波，其一般有两种实现方式，一种是通过傅里叶变换去除高频分量，另外一种是用离散化窗口滑窗卷积，最常见的是用滑窗实现，本文使用高斯函数对灰度图像进行处理从而达到滤除高频噪声的目的，对于零均值的高斯函数如式4-1所示。

(式 ‑1)

### 图像二值化

进行轮廓提取前需要找到合适的阈值对图像二值化，二值化后图像中就只有黑色与白色两种颜色，便于进行轮廓提取，所以找到合适的阈值便显得及其重要，一般采取试探法尝试不同阈值观察效果，当车牌颜色与车牌内文字颜色能清晰分开时，便认定为合格。为了加大图像中黑白区域对比度使用图像处理形态学方法（灰度+闭运算-开运算）处理，二值化后的结果如图 4.3所示。观察图像易知进过二值化处理后图像中的关键信息（车牌与字符）并未损失反而更加突出，接下来的处理便基于此二值图像，由于仅有黑色与白色，即灰度值仅有两种0与255，所以处理起来的信息量大大降低，同时二值化后的图像进行轮廓提取能取得更好的效果。



图 .3 二值化后图像

## 轮廓特征提取

轮廓提取指在包含目标与背景的数字图像中，忽略图像背景，噪声等不需要的信息，对目标进行提取检测的技术。目前轮廓提取主要有两种方式，第一种利用传统的形态学方法使用边缘检测算子对边缘进行检测，另外一种方法是使用从人类视觉中提取出来的数学模型提取边缘信息，这儿使用传统的形态学方法进行边缘检测。

### 边缘检测

边缘检测属于计算机视觉与图像处理中的基本问题，其目的是检测图像中亮度变化明显的点，在二值化图像中即是提取白色与黑色交界处的点。通过边缘检测大幅度地剔除了与目标无关的信息，利用OpenCV中的cv::findContours函数可以轻易得获取图像的边缘信息，二值化图像进过边缘检测函数后输出图像的轮廓信息，每个轮廓为一个整体。提取轮廓后的结果如图 4.4所示。



图 .4 提取轮廓信息后的图像

对比二值化图像与灰度图像容易发现，提取轮廓后的图像由线条构成，以黑色为背景，白色即为提取之后的轮廓。

### 轮廓筛选

由轮廓提取结果可知，轮廓提取后拥有大量的背景信息，所以需要对轮廓进行筛选去除大部分没必要的轮廓信息。通过计算轮廓的半径，周长等信息对轮廓进行筛选除去太大或者太小的轮廓从而达到筛选轮廓的目的。使用OpenCV中possibleChar.boundingRect.area()函数得到轮廓的面积，使用OpenCV中内置函数possibleChar.boundingRect.width得到轮廓的最大宽度，使用内置函数possibleChar.boundingRect.height获取轮廓最大高度，设定最小area为80，最小width为2，最小height为8，筛选后的轮廓如图 4.5所示。

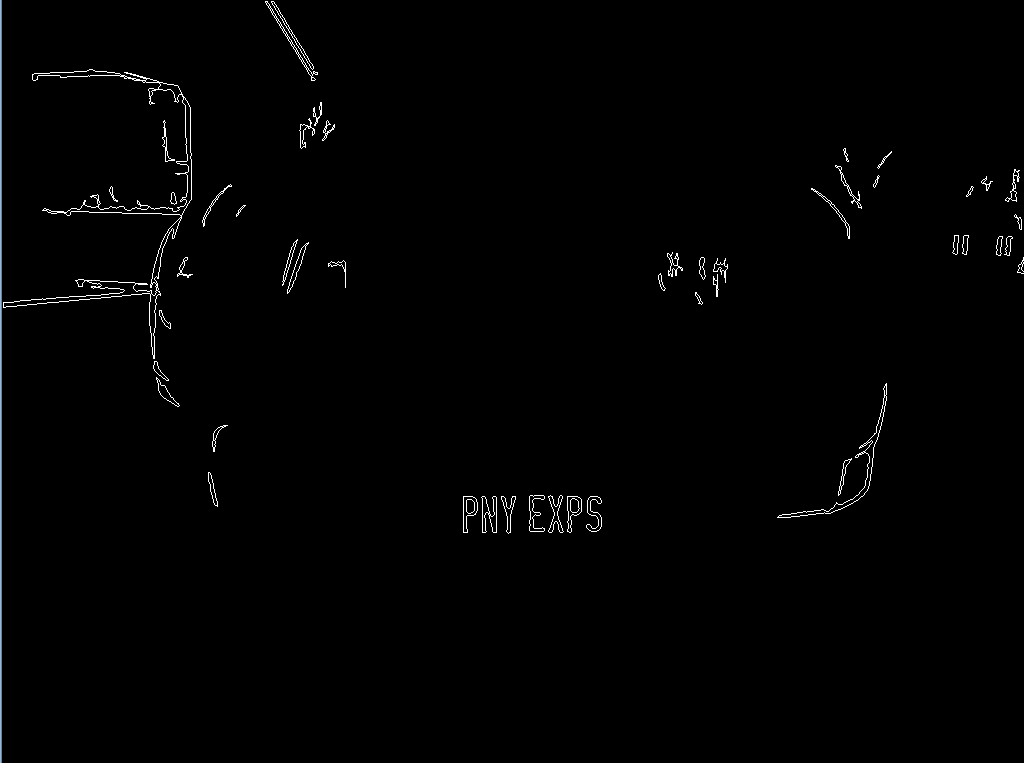


图 .5 轮廓筛选结果

进过筛选，去除了大量不相关的轮廓，并且完整保留了关键信息，大大地减少了后续计算量。

### 轮廓提取与分组

由图 4.6 轮廓分组流程可知，筛选后的轮廓依旧包含与车牌无关的部分，利用车牌字符轮廓之间的关联性可以有效提取有用轮廓与多余轮廓，车牌部分字符中心间距较窄，并且分布规律，对轮廓规律的提取流程如图4.6所示。将满足相关性的轮廓分到同一个组别，相关性的条件正是为了满足车牌的字符而量身定制，也就是说，车牌字符的轮廓都会被分到同一个类别并且成为一个整体，通过对现存轮廓的分组使得所要提取的所有信息都包含于此，后续处理便只需要针对组别进行便可。处理至此便完成了车牌检测的大部分工作，用红色的框将每个组括起来便能清楚地观察到处理后的结果，如图 4.7 轮廓分组筛选结果。



图 .6 轮廓分组流程

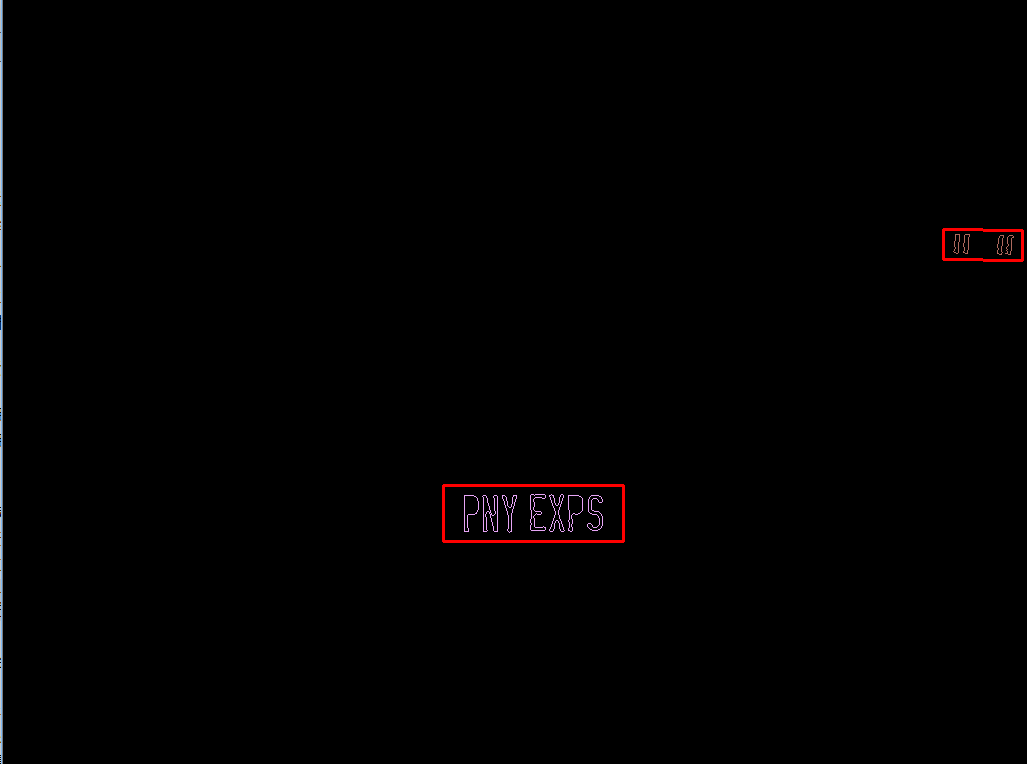


图 .7 轮廓分组筛选结果

对分组后的轮廓用红色的方框包含，计算方框的旋转角度，对不满足条件的组别直接剔除。

## 车牌切割

之前的所有处理均是在原图像上进行，目的都是为了提取图像中的车牌部分信息以便后续识别部分的处理，进过预处理与轮廓提取操作后已经成功提取出车牌部分信息，现在只需要对兴趣区域定位裁剪即可，即裁剪轮廓提取部分的红色区域。裁剪结果如图 4.8所示。

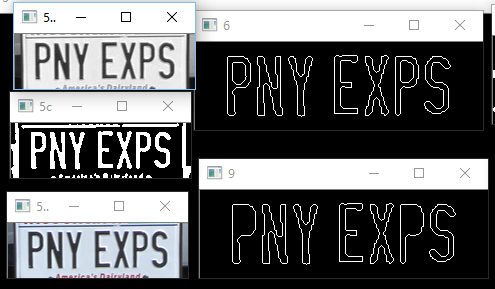


图 .8 车牌切割后结果

# 基于KNN的字符识别

## KNN简介

所谓最邻近算法，即使给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例，这K个实例的多数属于某个类，就把该输入实例分类到此类中。

KNN算法虽然为最简单的机器学习方法之一，但是应用得能得到极好的识别效果，本实验使用精简的最邻近算法。非参数方法的时间和空间复杂度正比于训练集的大小，目前已经提出了一些精简算法，以减少存放的实例数而不降低其性能。其基本思想是选择X的最小子集Z使得用Z代替X时，误差不增加（Dasarathy 1991）。最著名的方法是精简的最邻近（condensed nearest neighbor），其使用1-nn作为分类的非参数估计（Hart 1968）。1-nn以分段线性的方式近似判别式，并且只需要保存定义判别式的实例。类区域内部的实例不必作为它的同一类的最邻近存放，并且它的缺失不会导致（训练集上的）任何错误。这样的子集称为相容子集，并且我们希望找出最小的相容子集。

早在二十世纪Hart提出一种找到Z的贪心算法。即找到当前最佳是配置，训练集充空集开始，以空集去识别整个训练实例，每当出现一个错误时，便将错误实例添加至训练集中，直到没有错误出现为止，这样便将训练集有效地压缩并且实现误差不增加，此方法大大减小了训练样本的冗余度，尤其是在大样本的情况下大大降低了计算量提高了计算速度，压缩的最邻近算法的训练过程如图 5.1所示。

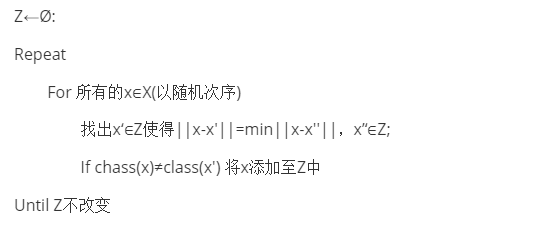


图 .1 压缩的最邻近算法

## 字符预处理

在对图片进行训练之前有必要对图像进行预处理使之满足统一的规则，从网上得到的图片各式各样，进过预处理之后的图像才能用于计算相互之间的距离。首先经过与4.1章相同的预处理得到车牌字符的二值化图像，之后进行字符分割与归一化处理。

### 字符分割

字符分割是车牌识别系统中的重要部分，车牌识别的时候也是对单个字符进行识别，所以字符识别的准确与否直接影响了最终的识别结果。

本实验采取计算字符轮廓大小的方法实现字符分割，直接计算字符轮廓大小然后以方形对图形进行分割。这样的方法分割的结果十分准确并且具有自动适配性，不会因为车牌的型号不同而受影响，处理之后的结果如图 5.2所示



图 .2 字符分割结果

计算出字符的轮廓之后，按照轮廓对车牌进行分割并将分割结果存入新的数组中，以便后续训练。

### 图像归一化

归一化是一种简化的计算方式，用以简化量纲以便统一计算，使物体系统绝对值的计算方式变为相对值的计算方式。

截取之后的字符图片大小大小各异，没法直接进行欧式距离的计算，所以需要对图像进行归一化处理，在综合考虑归一化后的速度与效率之后选择了200×300作为归一化的结果，将所有切割后的字符图像统一重新缩放至200×300的分辨率。

## 样本训练

为了减少样本个数，提高识别速度，训练过程中采取了贪心算法的思想，并将训练过后的结果数字化之后保存于xml文件中。

### 贪心算法

贪心算法又称贪婪算法，指在对某一问题进行求解时，只考虑适配与当前样本的最佳结果而不从整体最优解上考虑取值，所得的结果为局部最优解。精简的最邻近算法就是一种贪心算法，其目的是最小化训练误差与复杂度。可以用一个增广误差函数表示如式5-1所示：

E(Z|X)=E(X|Z)+λ|Z| (式 ‑1)

其中E(X|Z)用于存放Z在X上的误差。|Z|是z的基数，第二项为对复杂度的惩罚。与其他的正则化方案相同，λ体现了误差与复杂度的折中，使对于较小的λ，误差变得更为重要，随着λ的逐渐增大，对模型的惩罚则会更大。

贪心算法的一般训练流程如图 5.3所示。

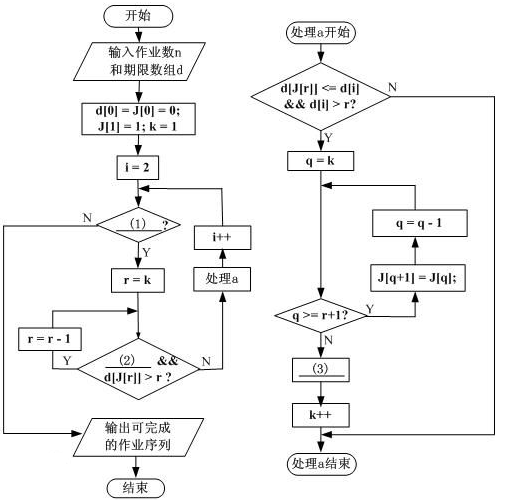


图 .3 贪心算法一般流程

### 训练过程

因为KNN是非参数化模型，所以其训练过程与神经网络，SVM（支持向量机），核机器等模型的训练方法不同，不用对未知参数进行训练。简单来说，作为惰性学习算法，KNN的训练过程就是对样本的存储过程，同时因为采用了贪心算法，样本的存储与筛选便成了训练过程中的首要问题。

首先从训练样本集中读出第一个样本，此时没有任何数据对样本进行识别，所以识别结果自然错误，将此样本值保存到用于识别未筛选样本的识别文件中。如此不断往复，使用识别文件中的数据对未筛选内容进行识别，若识别成功则舍弃此样本，若识别失败，将识别失败的图片数据存入识别文件中，直到达到要求的正确率为止，训练流程如图 5.4所示。



图 .4 KNN训练流程

训练用的样本来自互联网，训练的时候只需要对图片就行归一化后便可以直接使用。样本如图 5.5所示（裁取部分训练样本）。

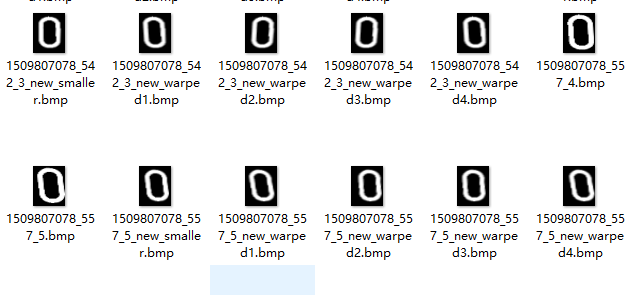


图 .5 训练样本

训练之后的数据以XML的格式保存起来，两个XML文件分别保存图片数据与数据对应的分类。本次设计以classification.xml文件保存对应的分类，images.xml与classification.xml数据一一对应，用于保存每个训练样本数据所处的分类。本次车牌识别仅仅针对只有字母与数字的车牌，所以车牌字符类别共有26个字母与10个数字一共36个类别，每个类别有5组不同的数据与之对应。判断类别的数据如图 5.6所示。

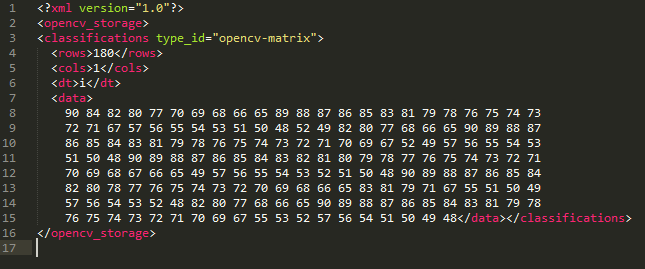


图 .6 KNN车牌字符类别数据

由图可以看出，数据均为数字，这是因为所有的类别均用相应的ASCII码表示，字符0-9对应着ASCII码的48-57，字符A-Z对应着ASCII码的65-90。这就是上图中数字所表示的意义。上面说到，每一个分类都有对应的数据，数据即是用作与待识别字符进行欧式距离计算的，距离最近的数据所对应的类别即被判定为待识别字符的类别。其部分数据如图 5.7（数据太大只截取了开头与结尾处数据）所示。由图可以看出，样本图片的数据为180\*600的矩阵，180与classification.xml中的180行对应，对应着36\*5=180个类别，600则是字符图像归一化后的分辨率20\*30=600，即对应的图片数据，目前为止，KNN模型的训工作练便完成了。

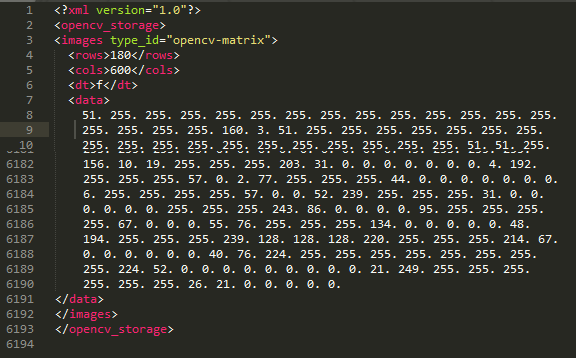


图 .7 字符图片数据（部分）

## KNN算法的字符识别

关于KNN算法的字符识别过程前面有过简要的介绍，即通过计算待识别字符图片与训练样本中的已分类数据之间的距离得到相似度，根据相似度来对未分类数据进行判定，字符识别的流程如图 5.8所示，识别开始时读取待识别数据，然后用待分类数据依次与已训练数据计算欧式距离，进行遍历过后比较得到最小距离，最小距离对应数据所在的分类即是我们需要得到的分类结果。OpenCV中内置的机器学习算法中直接包含了KNN算法，利用内置的KNearest->setDefaultK(1)设置K的初始值为1；使用kNearest->train()读取5.3节中训练好的样本数据；使用kNearest->findNearest()读取待识别数据并且与样本数据进行对比并返回对应类别的ASCII编码。为了将识别后的结果更清晰地显示出来，直接将字符写入原图像车牌下面进行对比。识别后的结果如图 5.9所示。



图 .8 识别流程图



图 .9 识别结果

# 系统功能、指标参数

## 系统功能

此系统实现了图像预处理，图像滤波，图像阈值化，轮廓提取，车牌分割，字符分割，KNN字符识别等功能。且接口很好地进行了封装，最后识别得到的车牌直接以字符形式返回，返回结果如图 6.1所示，接口的调用十分方便，可用于多种实用场合。

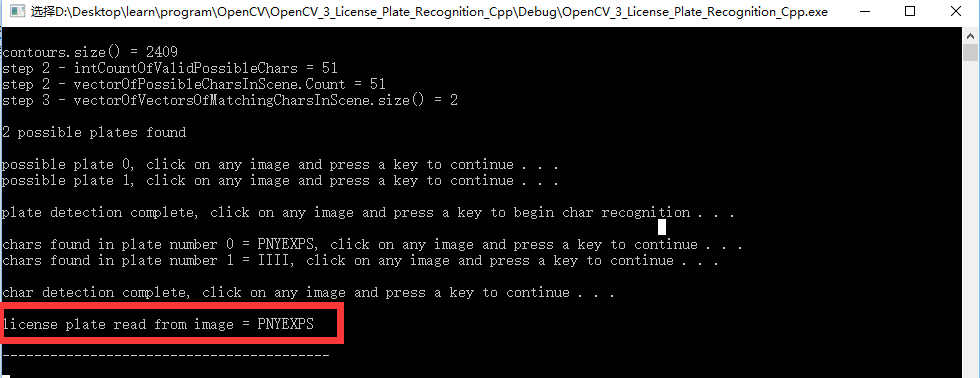


图 .1 系统最终返回结果

由于系统具有良好的封装性，所以整个系统实现中的任何一个过程都可以单独提取出来作为一个单独的功能使用。接口的调用都十分方便，都以函数或类的方式进行了良好的封装。系统的各个功能封装信息如表6‑1所示。

表‑1 系统功能封装信息

|  |  |
| --- | --- |
| 系统功能 | 封装情况 |
| 图像预处理 | 函数preprocess() |
| 车牌相关信息 | 类class possibleplate |
| 字符相关信息 | 类class possiblechar |
| 图像分割 | 函数ExtractPlate() |
| 字符识别 | 函数recognizeCharsInPlate() |

## 指标参数

系统的主要任务大致分为车牌位置检测与字符识别两个部分，车牌位置检测的结果直接影响字符识别的准确度。所以以下指标分为两个部分，车牌位置检测以随机的带有车辆的图片进行识别，字符识别的指标建立在车牌位置识别正确的前提下进行指标测试。

* 识别特征：车牌号码
* 输出结果：车牌号码
* 车牌倾斜角度范围：-10°-- +10°
* 车牌位置检测准确率：100%
* 字符识别准确率：98.7%

# 总结与展望

随着计算机芯片的快速发展，计算机的处理速度越来越快，许多之前受限于硬件的技术得到快速的发展。计算机视觉便是这样一个领域，在大数据的背景下机器学习得以发展，车牌识别由于巨大的市场需求成为最早落地的项目之一。本次设计的实现过程中通过互联网了解了很多相关知识，收获颇丰，虽然过程中遇到了很多问题，例如OpenCV库的不熟悉，图像处理相关知识的匮乏，C++语法及思想的不充分理解，机器学习相关的知识更是接触甚少。图书馆丰富的藏书量与互联网的便利很大程度地解决了这些问题，当静心潜力地投入到对这个领域的探索中时收获的知识与快乐都是不可言喻的。

本次设计针对目前计算机视觉领域的热门问题车牌识别给出了边缘检测+KNN最邻近算法的识别方案，旨在提出一种快捷简单的方法实现车牌识别的工作。整个项目的开发环境为Visual Studio2012，使用开源的计算机视觉库OpenCV。结合前人经验与个人的实践成功实现了车牌的识别工作。

首先，使用图像形态学领域的知识结合OpenCV便利的接口函数，通过图像空间变换，灰度变换，滤波，阈值化，轮廓提取等操作定位车牌位置，确定位置后切割车牌，则车牌位置检测部分工作基本完成；字符识别使用KNN算法，作为惰性算法，KNN的训练过程即使对训练样本的加工与分类存储，整个过程包括字符预处理，字符分割，图像归一化，计算欧式距离得到待分类字符所属类别，基本完成了预定的目标，实现的车牌字符的提取。

该系统在车牌识别中通用性强，全程使用C++进行开发，可以方便地移植到移动设备与嵌入式系统上工作。本次设计虽然完成了车牌识别的工作，但是整个过程中依然存在很多不足的地方有待提高。

1. 对于车牌角度纠正的问题没有给出合适的方案，所以对于倾斜的车辆图片不能很好地进行识别，此处还有较大的改进空间。
2. 系统对于复杂背景与复杂环境的抗干扰能力不足，当图像背景复杂（比如有类似车牌的标志，多个车牌的情况）时，识别效果并不理想。
3. 在夜晚或是光线不均匀的环境中，阈值化的效果不佳，造成车牌定位错误的情况，或是由于反光导致识别过程中分类错误，不能有效地提取到车牌字符。
4. 识别的准确率与实时性有待进一步地提高，车牌识别的实时性至关重要，正确识别车牌只是最基本的要求，当输入设备的分辨率较高，输入图像的实时性要求高的时候，系统的识别能力下降，有卡顿的现象发生，算法依然需要完善。

通过此次毕业设计我收获到了很多新的知识，也更清楚了不足的地方，需要改进的地方还有很多，路漫漫其修远兮，吾将上下而求索。

# 致谢

在熬过了漫长的两个月后，毕业论文终于接近尾声，毕业设计能够正常完工离不开老师同学们的默默支持，在这里首先要对我的指导老师卿曹进教授表示诚挚的感谢，卿老师为毕业设计的方向与实施方案给出了指导性的意见。以及CSDN论坛上的博主们，你们的博客解决了我遇到的很多难题。

同时，我还要感谢微软公司，感谢你们开发出visual studio这么好用的IDE以及Microsoft Word，没有这些我是完成不了代码的调试以及论文的编写工作；感谢Willow Garage一直以来对OpenCV的开发维护，使得我们能免费用上如此好用的计算机视觉库，没有你们的支持，此次毕业设计就要更换题目了；感谢Image-Net网站维护人员十多年来的搜集素材努力，是你们为机器学期提供了大量的训练样本，极大地减少了此次论文的工作量；感谢学校及时地装上了空调，使得同学们能在炎炎夏日睡个好觉，否则是不会有如此精力持续地接收毕业论文的摧残；感谢学校二食堂年复一年坚持不懈地做出难以下咽的饭菜，让我时刻体验到生活的艰辛与不易，给了我们不断奋斗的强大动力，让我们明白吃得苦中苦方位人上人的祖传真理；感谢少年JUMP一直以来推出的优秀漫画，让我能够在论文的重压下找到一处喘息的净土，重新发现生活的美好之处；感谢篮球，好的身体是完成毕业设计不可缺少的重要支撑，每周两次的篮球运动使得身体一直保持着不错的状态。感谢维普论文检测系统，你精准地检测使我清晰地了解到论文的重复内容，论文查重能过你功不可没。一路走来需要感谢的人和事太多太多，怀有一颗感恩的心，才能更好地向前。

# 参考文献

1. Sebastian Raschka.Python机器学习[M].机械工业出版社，2017.3
2. Ethem Alpaydin.机器学习导论[M].机械工业出版社，2015.11
3. 王志明.数字图像处理与分析[M].清华大学出版社，2012.2
4. 邬延辉、王小权、陈叶芳.C++程序设计教程基于案例与试验驱动[M]，2010.6
5. 杨思源. 基于OPENCV的车辆牌照识别系统研究[D]. 西安电子科技大学, 2013.
6. 滕俊 王弟林 文汉云. 基于OpenCV下的Visual C数字图像处理方法[D]. 现代计算机(专业版), 2012.
7. 刘耀军 张姗梅 冯娟. 基于有限自动机的二值图像膨胀[D]. 西北师范大学学报(自然科学版), 2012.
8. 宋佳星. 基于机器视觉的插件机定位系统的研究与应用[D]. 大连理工大学, 2014.
9. 彭晓庆. 道路交通标志图像处理[D]. 西安电子科技大学, 2010.
10. 程成. 基于电磁扫描的高密度电路板故障器件检测技术[D]. 西安电子科技大学, 2011.
11. 吴亮. 运动目标提取、阴影和鬼影检测及去除算法研究[D]. 国防科学技术大学, 2007.
12. 张鑫. 智能视频监控中运动目标的检测与跟踪[D]. 杭州电子科技大学, 2012.
13. 韩志远. 基于粒子滤波的运动行人跟踪方法的研究[D]. 苏州大学, 2012.
14. 孙红霞. 地下输油管道红外图像的分析研究[D]. 天津大学, 2004.
15. 刘峰. 基于面部特征的性别鉴别方法研究[D]. 计算机光盘软件与应用, 2013.
16. 吕联盟. 基于云计算的人脸识别系统研究与设计[D]. 长安大学, 2014.
17. 陈筱. 基于区域生长和数学形态学的MRI图像处理研究[D]. 中南民族大学, 2012.
18. 闫海霞 刘岩俊. 基于顺序形态学自适应边缘检测方法[D]. 计算机应用研究, 2011.
19. 钟华. 水火弯板机系统中大尺寸钢板三维重建过程研究[D]. 广东工业大学, 2014.
20. 苏超 彭泽荣 黄志文. 基于图像处理的绝缘子缺陷检测[D]. 广东电网有限责任公司清远供电局, 2016.
21. 莫国影. CCD图像识别技术在疲劳裂纹检测中的应用基础研究[D]. 南京航空航天大学, 2008.

附录：关键源代码

1. // Main.cpp
3. // add-self-round
4. **double** round(**double** val)
5. {
6. **return** (**double**)val;
7. }
8. #include "Main.h"
10. ///////////////////////////////////////////////////////////////////////////////////////
11. **int** main(**void**) {
12. **bool** blnKNNTrainingSuccessful = loadKNNDataAndTrainKNN();           // attempt KNN training
14. **if** (blnKNNTrainingSuccessful == **false**) {                            // if KNN training was not successful
15. // show error message
16. std::cout << std::endl << std::endl << "error: error: KNN traning was not successful" << std::endl << std::endl;
17. **return**(0);                                                      // and exit program
18. }
20. cv::Mat imgOriginalScene;           // input image
22. imgOriginalScene = cv::imread("image9.png");         // open image
24. **if** (imgOriginalScene.empty()) {                             // if unable to open image
25. std::cout << "error: image not read from file\n\n";     // show error message on command line
26. \_getch();                                               // may have to modify this line if not using Windows
27. **return**(0);                                              // and exit program
28. }
29. // Main.h
31. #ifndef MY\_MAIN         // used MY\_MAIN for this include guard rather than MAIN just in case some compilers or environments #define MAIN already
32. #define MY\_MAIN
34. #include<opencv2/core/core.hpp>
35. #include<opencv2/highgui/highgui.hpp>
36. #include<opencv2/imgproc/imgproc.hpp>
37. #include<math.h>
39. #include "DetectPlates.h"
40. #include "PossiblePlate.h"
41. #include "DetectChars.h"
43. #include<iostream>
44. #include<conio.h>           // may have to modify this line if not using Windows
46. //#define SHOW\_STEPS            // un-comment or comment this line to show steps or not
48. // global constants ///////////////////////////////////////////////////////////////////////////////
49. **const** cv::Scalar SCALAR\_BLACK = cv::Scalar(0.0, 0.0, 0.0);
50. **const** cv::Scalar SCALAR\_WHITE = cv::Scalar(255.0, 255.0, 255.0);
51. **const** cv::Scalar SCALAR\_YELLOW = cv::Scalar(0.0, 255.0, 255.0);
52. **const** cv::Scalar SCALAR\_GREEN = cv::Scalar(0.0, 255.0, 0.0);
53. **const** cv::Scalar SCALAR\_RED = cv::Scalar(0.0, 0.0, 255.0);
55. // function prototypes ////////////////////////////////////////////////////////////////////////////
56. **int** main(**void**);
57. **void** drawRedRectangleAroundPlate(cv::Mat &imgOriginalScene, PossiblePlate &licPlate);
58. **void** writeLicensePlateCharsOnImage(cv::Mat &imgOriginalScene, PossiblePlate &licPlate);
59. # endif // MAIN
60. // PossiblePlate
61. std::vector<PossiblePlate> vectorOfPossiblePlates = detectPlatesInScene(imgOriginalScene);          // detect plates
63. vectorOfPossiblePlates = detectCharsInPlates(vectorOfPossiblePlates);                               // detect chars in plates
65. cv::imshow("imgOriginalScene", imgOriginalScene);           // show scene image
67. **if** (vectorOfPossiblePlates.empty()) {                                               // if no plates were found
68. std::cout << std::endl << "no license plates were detected" << std::endl;       // inform user no plates were found
69. }
70. **else** {                                                                            // else
71. // if we get in here vector of possible plates has at leat one plate
73. // sort the vector of possible plates in DESCENDING order (most number of chars to least number of chars)
74. std::sort(vectorOfPossiblePlates.begin(), vectorOfPossiblePlates.end(), PossiblePlate::sortDescendingByNumberOfChars);
76. // suppose the plate with the most recognized chars (the first plate in sorted by string length descending order) is the actual plate
77. PossiblePlate licPlate = vectorOfPossiblePlates.front();
79. cv::imshow("imgPlate", licPlate.imgPlate);            // show crop of plate and threshold of plate
80. cv::imshow("imgThresh", licPlate.imgThresh);
82. **if** (licPlate.strChars.length() == 0) {                                                      // if no chars were found in the plate
83. std::cout << std::endl << "no characters were detected" << std::endl << std::endl;      // show message
84. **return**(0);                                                                              // and exit program
85. }
87. drawRedRectangleAroundPlate(imgOriginalScene, licPlate);                // draw red rectangle around plate
89. std::cout << std::endl << "license plate read from image = " << licPlate.strChars << std::endl;     // write license plate text to std out
90. std::cout << std::endl << "-----------------------------------------" << std::endl;
92. writeLicensePlateCharsOnImage(imgOriginalScene, licPlate);              // write license plate text on the image
94. cv::imshow("imgOriginalScene", imgOriginalScene);                       // re-show scene image
96. cv::imwrite("imgOriginalScene.png", imgOriginalScene);                  // write image out to file
97. }
99. cv::waitKey(0);                 // hold windows open until user presses a key
101. **return**(0);
102. }
104. ///////////////////////////////////////////////////////////////////////////////////////////////////
105. **void** drawRedRectangleAroundPlate(cv::Mat &imgOriginalScene, PossiblePlate &licPlate) {
106. cv::Point2f p2fRectPoints[4];
108. licPlate.rrLocationOfPlateInScene.points(p2fRectPoints);            // get 4 vertices of rotated rect
110. **for** (**int** i = 0; i < 4; i++) {                                       // draw 4 red lines
111. cv::line(imgOriginalScene, p2fRectPoints[i], p2fRectPoints[(i + 1) % 4], SCALAR\_RED, 2);
112. }
113. }
114. ////////////////////////////// 车牌处理部分
115. **void** writeLicensePlateCharsOnImage(cv::Mat &imgOriginalScene, PossiblePlate &licPlate) {
116. cv::Point ptCenterOfTextArea;                   // this will be the center of the area the text will be written to
117. cv::Point ptLowerLeftTextOrigin;                // this will be the bottom left of the area that the text will be written to
119. **int** intFontFace = CV\_FONT\_HERSHEY\_SIMPLEX;                              // choose a plain jane font
120. **double** dblFontScale = (**double**)licPlate.imgPlate.rows / 30.0;            // base font scale on height of plate area
121. **int** intFontThickness = (**int**)round(dblFontScale \* 1.5);             // base font thickness on font scale
122. **int** intBaseline = 0;
124. cv::Size textSize = cv::getTextSize(licPlate.strChars, intFontFace, dblFontScale, intFontThickness, &intBaseline);      // call getTextSize
126. ptCenterOfTextArea.x = (**int**)licPlate.rrLocationOfPlateInScene.center.x;         // the horizontal location of the text area is the same as the plate
128. **if** (licPlate.rrLocationOfPlateInScene.center.y < (imgOriginalScene.rows \* 0.75)) {      // if the license plate is in the upper 3/4 of the image
129. // write the chars in below the plate
130. ptCenterOfTextArea.y = (**int**)round(licPlate.rrLocationOfPlateInScene.center.y) + (**int**)round((**double**)licPlate.imgPlate.rows \* 1.6);
131. }
132. **else** {                                                                                // else if the license plate is in the lower 1/4 of the image
133. // write the chars in above the plate
134. ptCenterOfTextArea.y = (**int**)round(licPlate.rrLocationOfPlateInScene.center.y) - (**int**)round((**double**)licPlate.imgPlate.rows \* 1.6);
135. }
137. ptLowerLeftTextOrigin.x = (**int**)(ptCenterOfTextArea.x - (textSize.width / 2));           // calculate the lower left origin of the text area
138. ptLowerLeftTextOrigin.y = (**int**)(ptCenterOfTextArea.y + (textSize.height / 2));          // based on the text area center, width, and height
140. // write the text on the image
141. cv::putText(imgOriginalScene, licPlate.strChars, ptLowerLeftTextOrigin, intFontFace, dblFontScale, SCALAR\_YELLOW, intFontThickness);
142. }
143. // PossibleChar.cpp
145. #include "PossibleChar.h"
147. ///////////////////////////////////////////////////////////////////////////////////////////////////
148. PossibleChar::PossibleChar(std::vector<cv::Point> \_contour) {
149. contour = \_contour;
151. boundingRect = cv::boundingRect(contour);
153. intCenterX = (boundingRect.x + boundingRect.x + boundingRect.width) / 2;
154. intCenterY = (boundingRect.y + boundingRect.y + boundingRect.height) / 2;
156. dblDiagonalSize = sqrt(pow(boundingRect.width, 2) + pow(boundingRect.height, 2));
158. dblAspectRatio = (**float**)boundingRect.width / (**float**)boundingRect.height;
159. }
161. // PossibleChar.h
163. #ifndef POSSIBLE\_CHAR\_H
164. #define POSSIBLE\_CHAR\_H
166. #include<opencv2/core/core.hpp>
167. #include<opencv2/highgui/highgui.hpp>
168. #include<opencv2/imgproc/imgproc.hpp>
170. ///////////////////////////////////////////////////////////////////////////////////////////////////
171. **class** PossibleChar {
172. **public**:
173. // member variables ///////////////////////////////////////////////////////////////////////////
174. std::vector<cv::Point> contour;
176. cv::Rect boundingRect;
178. **int** intCenterX;
179. **int** intCenterY;
181. **double** dblDiagonalSize;
182. **double** dblAspectRatio;
184. ///////////////////////////////////////////////////////////////////////////////////////////////
185. **static** **bool** sortCharsLeftToRight(**const** PossibleChar &pcLeft, **const** PossibleChar & pcRight) {
186. **return**(pcLeft.intCenterX < pcRight.intCenterX);
187. }
189. ///////////////////////////////////////////////////////////////////////////////////////////////
190. **bool** operator == (**const** PossibleChar& otherPossibleChar) **const** {
191. **if** (**this**->contour == otherPossibleChar.contour) **return** **true**;
192. **else** **return** **false**;
193. }
195. ///////////////////////////////////////////////////////////////////////////////////////
196. **bool** operator != (**const** PossibleChar& otherPossibleChar) **const** {
197. **if** (**this**->contour != otherPossibleChar.contour) **return** **true**;
198. **else** **return** **false**;
199. }
200. // function prototypes ////////////////////////////////////////////////////////////////////////
201. PossibleChar(std::vector<cv::Point> \_contour);
202. };
203. #endif  // POSSIBLE\_CHAR\_H


207. // PossiblePlate.h
209. #ifndef POSSIBLE\_PLATE\_H
210. #define POSSIBLE\_PLATE\_H
212. #include <string>
214. #include<opencv2/core/core.hpp>
215. #include<opencv2/highgui/highgui.hpp>
216. #include<opencv2/imgproc/imgproc.hpp>
218. ///////////////////////////////////////////////////////////////////////////////////////////////////
219. **class** PossiblePlate {
220. **public**:
221. // member variables ///////////////////////////////////////////////////////////////////////////
222. cv::Mat imgPlate;
223. cv::Mat imgGrayscale;
224. cv::Mat imgThresh;
226. cv::RotatedRect rrLocationOfPlateInScene;
228. std::string strChars;
229. ///////////////////////////////////////////////////////////////////////////////////////
230. **static** **bool** sortDescendingByNumberOfChars(**const** PossiblePlate &ppLeft, **const** PossiblePlate &ppRight) {
231. **return**(ppLeft.strChars.length() > ppRight.strChars.length());
232. }
234. };
235. #endif      // end #ifndef POSSIBLE\_PLATE\_H