|  |
| --- |
| 实验成绩 |
|  |



实验报告

课程名称： 暑期生产实习

实验项目： 车牌识别

所在院系： 数学学院数学与应用数学

学生姓名： 桑浩鑫

学生学号： 1171000221

授课学期： 2020年夏季学期

完成时间： 2020年7月27日

1. **背景和意义**

项目背景：在我国，车辆牌照是机动车唯一的管理标识符号，因而在交通管理中具有不可替代的作用，因此车辆牌照识别系统应具有非常高的识别正确率，对环境光照条件、拍摄位置和车辆行驶速度等因素的影响应有较大的容阈，而且要求满足实时性要求。

车牌识别系统 (Vehicle License Plate Recognition，VLPR) 是计算机视频图像识别技术在车辆牌照识别中的一种应用，能够将运动中的车辆牌照信息（含汉字字符、英文字母、阿拉伯数字及号牌颜色）从复杂背景中提取并识别出来，通过车牌提取、图像预处理、特征提取、车牌字符识别等技术，识别车辆牌号、颜色等信息，目前最新的技术水平为字母和数字的识别率均可达到 99% 以上。

车牌识别是现代智能交通系统中的重要组成部分之一，应用十分广泛。它以数字图像处理、模式识别、计算机视觉等技术为基础，对摄像机所拍摄的车辆图像或者视频序列进行分析，得到每一辆汽车唯一的车牌号码，从而完成识别过程

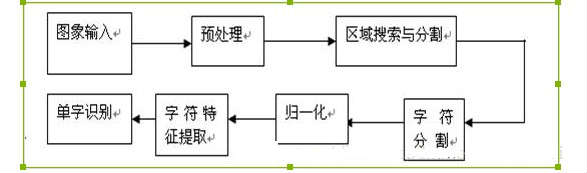
项目意义：当前，车牌识别技术已经广泛应用于停车管理、称重系统、静态交通车辆管理、公路治超、公路稽查、车辆调度、车辆检测等各种场合，对于维护交通安全和城市治安，防止交通堵塞，实现交通自动化管理有着现实的意义。

随着车牌识别技术应用场景逐渐细分，智慧停车场、智慧车服、智慧工地、智慧加油站等各大落地场景都对车牌识别技术有各自的场景特定需求，智能车牌识别相机也在不断发展以适应市场变化。

1. 实验内容及原理

开发工具：利用Microsoft Visual Studio 2010旗舰版为平台，基于C++语言OpenCV 2.49。

实验内容：整个车牌识别系统可以划分为以下几个步骤：



限于时间有限，本实验主要实现上述步骤中的“图像输入”，“预处理”以及“区域搜索与分割”三个步骤。剩余的四个步骤“字符分割”，“归一化”，“字符特征提取”以及“单字识别”在此仅作适当介绍，具体算法及代码实现，本人将在实习结束后进行进一步地深入了解和学习。

实验原理：

一、图像采集和转换

考虑到现有牌照的字符与背景的颜色搭配一般有蓝底白字、黄底黑字、白底红字、绿底白字和黑底白字等几种，利用不同的色彩通道就能够将区域与背景明显地区分出来。

二、边缘提取

边缘是指图像局部亮度变化显著的部分，是图像纹理特征提取和形状特征提取等图像分析的重要基础。所以在此我们要对图像进行边缘检測。图像增强处理对图像牌照的可辨认度的改善和简化后续的牌照字符定位和切割的难度都是非常有必要的。

2.1 灰度矫正

因为牌照图像在拍摄时受到种种条件的限制和干扰，图像的灰度值往往与实际景物不完全匹配，这将直接影响到图像的后续处理。造成这样的影响的原因可能是因为被摄物体的远近不同，使得图像中央区域和边缘区域的灰度失衡，或是因为摄像头在扫描时各点的灵敏度有较大的差异而产生图像灰度失真，或是因为曝光不足而使得图像的灰度变化范围非常窄。这时可以采用灰度校正的方法来处理，增强灰度的变化范围、丰富灰度层次，以达到增强图像的对照度和分辨率。

2.2图像平滑处理

对于受噪声干扰严重的图像，因为噪声点多在频域中映射为高频分量，因此能够在通过低也能够直接在空域中用求邻域平均值的方法来通滤波器来滤除噪声，但实际中为了简化算法，削弱噪声的影响，这样的方法称为图像平滑处理。

    同时为了克服平均化引起的图像模糊现象，可以给中心点像素值与其邻域平均值的差值设置某一固定的阈值，仅仅有大于该阈值的点才干替换为邻域平均值，而差值不大于阈值时，仍保留原来的值。

三、牌照的定位和切割

牌照的定位和切割是牌照识别系统的关键技术之中的一个，其主要目的是在经图像预处理后的原始灰度图像中确定牌照的详细位置，并将包括牌照字符的一块子图像从整个图像中切割出来，供字符识别子系统识别之用，切割的准确与否直接关系到整个牌照字符识别系统的识别率。

    3.1牌照区域定位

    ​牌照图像经过了以上的处理后，牌照区域已经十分明显，并且其边缘得到了勾勒和加强。此时可进一步确定牌照在整幅图像中的准确位置。可以选用数学形态学的方法，其基本思想是用具有一定形态的机构元素去量度和提取图像中的相应形状以达到对图像分析和识别的目的。数学形态学的应用能够简化图像数据，保持它们主要的形态特征，并除去不相干的结构。

  3.2牌照区域切割    ​

对车牌的切割能够有非常多种方法，例如可以利用车牌的彩色信息来进行切割。依据车牌底色，采用彩色像素点统计的方法切割出合理的车牌区域，确定车牌底色蓝色RGB相应的各自灰度范围，然后行方向统计在此颜色范围内的像素点数量，设定合理的阈值，确定车牌在行方向的合理区域。然后，在切割出的行区域内，统计列方向蓝色像素点的数量，终于确定完整的车牌区域。

3.3统一处理    ​

经过上述方法切割出来的车牌图像中存在目标物体、背景还有噪声，要想从图像中直接提取出目标物体，最经常使用的方法就是设定一个阈值T，然后对图像二值化。

四、字符切割和识别

4.1字符切割

在汽车牌照自己主动识别过程中，字符切割起着承前启后的作用。它在前期牌照定位的基础上进行字符的切割，然后再利用切割的结果进行字符识别。

4.2字符归一化

一般切割出来的字符要进行进一步的处理，以满足下一步字符识别的需要。此时需要进行归一化处理，以便于后期处理。

4.3字符识别

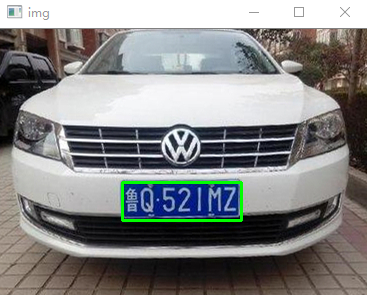
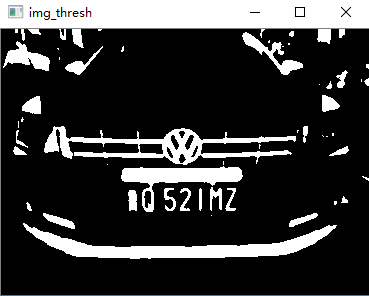
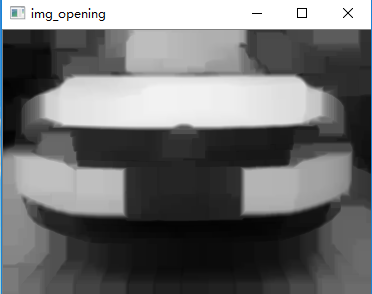
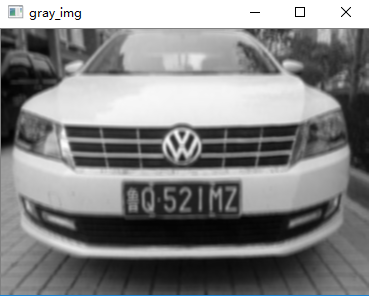
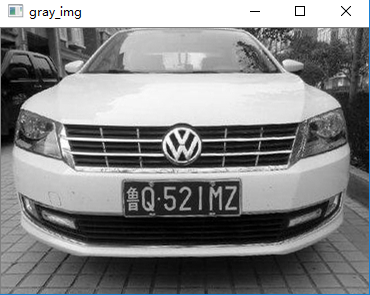
字符的识别眼下用于车牌字符识别(OCR)中的算法主要有基于模板匹配的OCR算法以及基于人工神经网络的OCR算法。

1. 实验步骤

实验方法及过程：

1. 首先进行彩色灰度处理；同时为保证输入图像的大小不能过大，还需要限制图像的最大宽度。
2. **int** maxWindowWidth=400;
3. **int** maxWindowHeight=300;
4. //转换颜色空间
5. cvtColor(img, grayImg, COLOR\_BGR2GRAY);
7. Mat  grayImgResize= ReSize(grayImg, maxWindowWidth, maxWindowHeight);
9. imshow("gray\_img", grayImgResize);
10. Mat ReSize(Mat imgArr,**int** maxWidth,**int** maxHeight) {
11. Mat imgReSize;
12. **double** changeXRate = 0.0;
13. **double** changeYRate = 0.0;
14. Size dsize =Size(maxWidth, maxHeight);//尺寸类
16. **int** width = imgArr.cols;
17. **int** height = imgArr.rows;
18. **if** (width > maxWidth) {
19. changeXRate = maxWidth / width;
20. }
21. resize(imgArr, imgReSize, dsize, changeXRate, changeYRate);
23. **return** imgReSize;
24. }
25. 基于二维离散卷积的高斯平滑对灰度图像进行降噪处理。
26. Mat imgGauss;
27. GaussianBlur(reSizeImg, imgGauss, kSize, 0, 0, BORDER\_DEFAULT);
28. 为了后面更加准确的提取车牌的轮廓，需要对图像进行形态学处理，这里对图像进行开运算处理。
29. Point mPoint = Point(-1, -1);//二维图像的点类
30. Mat kernel = getStructuringElement(MORPH\_RECT, mSize, mPoint);//获取形态学处理所需的结构化元素
31. Mat imgOpening;
32. //开运算（先腐蚀后膨胀）
33. morphologyEx(imgGauss, imgOpening, MORPH\_OPEN, kernel, mPoint,2, BORDER\_DEFAULT, morphologyDefaultBorderValue());
34. addWeighted(reSizeImg,1, imgOpening,-1,0, imgOpening,-1);//将两张相同大小，相同类型的图片融合
35. imshow("img\_opening", imgOpening);
36. 对图像进行阈值分割，这里采用 OTSU阈值处理。
37. threshold(imgOpening, thresholdImg, 0, 255, THRESH\_BINARY + THRESH\_OTSU);//阈值化函数
38. imshow("img\_thresh", thresholdImg);
39. 对图像进行边缘检测，这里采用的是 Canny 边缘检测。
40. Canny(thresholdImg, edgesImg, 100, 200, 3, **false**);//边缘检测函数
41. imshow("img\_edge", edgesImg);
42. 再进行一次闭运算和开运算，填充白色物体内细小黑色空洞的区域并平滑其边界。
43. //使用闭运算和开运算让图像边缘成为一个整体
44. Size mSize1 = Size(7, 7);
45. Point mPoint1 = Point(-1, -1);
46. Mat kernel1 = getStructuringElement(MORPH\_RECT, mSize1, mPoint1);
47. morphologyEx(edgesImg, closingImg, MORPH\_CLOSE, kernel1, mPoint1, 2, BORDER\_DEFAULT, morphologyDefaultBorderValue());
48. Size mSize2 = Size(7, 7);
49. Mat kernel2 = getStructuringElement(MORPH\_RECT, mSize2, mPoint1);
50. morphologyEx(closingImg, openingImg, MORPH\_OPEN, kernel2, mPoint1, 2, BORDER\_DEFAULT, morphologyDefaultBorderValue());
51. imshow("img\_edge2", openingImg);
52. 筛选出车牌所在的那个轮廓，这里利用车牌宽和高的比例是固定的这一形态特征进行筛选，并使用绿色的线条将得到的车牌框选出来。
53. //查找图像边缘整体形成的矩形区域，可能有很多，车牌就在其中一个矩形区域中
54. vector<vector<Point> > contoursArrs;
55. vector<Vec4i> hierarchy;
56. findContours(openingImg, contoursArrs, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);
57. //"CHAIN\_APPROX\_SIMPLE": 仅保存轮廓的拐点信息
59. vector<vector<Point> > tempContoursArrs(contoursArrs.size());
61. **int** k = 0;
62. **for** (**size\_t** i = 0; i < contoursArrs.size(); i++) {
64. **if** (contourArea(contoursArrs[i], **false**) > 2000) {
65. tempContoursArrs[k] = contoursArrs[i];
66. k++;
67. cout << contourArea(contoursArrs[i], **false**)<<endl;
69. }
70. }
71. vector<RotatedRect> minRotatedRect(k);
72. vector<Rect>  minRect(k);
73. **for** (**size\_t** i = 0; i < k; i++) {
74. minRotatedRect[i]=minAreaRect(tempContoursArrs[i]);
76. cout << minRotatedRect[i].size<< endl;
77. **double** ratio = minRotatedRect[i].size.width / minRotatedRect[i].size.height;
78. **int** m = 0;
79. vector<vector<Point> > PlateContoursArrs(k);
80. **if** ((ratio > 2.5 && ratio < 4)) {
81. RNG rng(12345);
82. PlateContoursArrs[m] = tempContoursArrs[i];
83. vector<RotatedRect>   plateRotatedRect(k);
84. plateRotatedRect [m]= minAreaRect(PlateContoursArrs[m]);
86. Scalar color = Scalar(rng.uniform(0, 256), rng.uniform(0, 256), rng.uniform(0, 256));
87. drawContours(reSizeImg, PlateContoursArrs, (**int**)0, color);
88. Point2f rect\_points[4];
89. minRotatedRect[i].points(rect\_points);
90. **for** (**int** j = 0; j < 4; j++)
91. {
92. line(reSizeImg, rect\_points[j], rect\_points[(j + 1) % 4], color);
93. }
94. imshow("ContoursLine", reSizeImg);
95. //提取图片
96. Rect brect = plateRotatedRect[m].boundingRect(); //返回包含旋转矩形的最小矩形
97. rectangle(reSizeImg, brect, Scalar(0, 255, 0), 2);
98. imshow("RectImg'", reSizeImg);
99. m++;
100. }
101. }
102. 实验结果

下图从左到右，从上到下，依次对应实验步骤中各步骤的实现结果。其中最后一个是是步骤（7）完成后的最终结果。



1. **实验总结**

通过本次实验，首先我熟悉了本次实验所使用的开发工具，增强了自己的编程能力与实践能力。同时，也在短短的几个星期内，简单涉猎了有关图像处理的基本知识，也了解到了一些常用方法，为自己日后的工作或是实践打下了基础。

但在这次实验中我也遇到了许多问题。一些小问题诸如环境配置失败，函数参数理解错误，头文件引用缺少等等。但也还有一些由于自己的编程能力不足，以及资料查阅不够所引发的问题，例如程序实现结果并不理想，识别成功率不高。后来，在对这一问题进行仔细分析后，个人认为是在“车牌定位”这一步骤上，所采取的方法有些过于简单，翻阅网上的资料后，发现在定位车牌时，除了利用车牌的形态特征，还可以利用车牌的色彩特征加以辅助，具体来说就是：依据车牌底色等有关的先验知识，采用彩色像素点统计的方法切割出合理的车牌区域，确定车牌底色蓝色RGB相应的各自灰度范围，然后行方向统计在此颜色范围内的像素点数量，设定合理的阈值，确定车牌在行方向的合理区域。然后，在切割出的行区域内，统计列方向蓝色像素点的数量，最终确定出完整的车牌区域。但是由于时间有限，这一部分尚未在代码中体现，也算是一个遗憾。

本次实验是一个快速但完整的学习过程，为我今后的学习有着很大的鼓舞作用，使得自己有信心能够在一个较短的时间内学习一个新工具、掌握一套新方法、解决一系列新问题。同时，图像处理在日常生活中也是一个很常见的问题，本次实验所学的方法，在将来走上工作岗位也有着潜在的作用，拓展了视野丰富了基础知识。总而言之，我在本次实验中，无论是在学习能力方面或是知识方面，都有着较大的收获。