# 《数字图像处理》课程报告

**课题名称： 车牌定位2021**

课题负责人名（学号）：高婷婷（2019141460270）

同组成员名单（角色）： 无

指导教师： 李新胜

**评阅成绩：**

评阅意见：

提交报告时间：2021 年 1 月 3 日

**车牌定位2021**

人工智能 专业

**学生** 高婷婷

**[摘要]** 本文依据较为流行有效的车牌定位方法，针对2021年最新的车牌定位数据集，设计了新的车牌定位方法。主要采用了基于颜色特征的图像预处理方法与轮廓寻找筛选方法，达到在复杂背景下定位车牌的目的，最终定位准确率达到86.7%。

**关键词**：车牌定位 图像处理 opencv

目录

[《数字图像处理》课程报告 0](#_Toc91865426)

[1. 课题背景 1](#_Toc91865427)

[2. 实验方案 2](#_Toc91865428)

[2.1 数据集 2](#_Toc91865429)

[2.2 实验流程 2](#_Toc91865430)

[2.3 方法设计 3](#_Toc91865431)

[2.3.1 图像预处理 3](#_Toc91865432)

[2.3.2 轮廓寻找与筛选 6](#_Toc91865433)

[3. 实验结果与分析 6](#_Toc91865434)

[4. 结论 9](#_Toc91865435)

[附录(源码，更多的实验结果等)： 9](#_Toc91865436)

# 课题背景

近年来，人工智能的研究不断深入，智慧城市，智慧交通也引起了大家的关注，车牌识别作为智慧交通的重要一环，被广泛应用在停车场管理，违章处理，高速收费等领域。车牌定位作为车牌识别最开始的一环，也是必不可少的一环，同样受到了重视，车牌定位的方法也在不断的更新迭代中。

由于车辆牌照相对于车辆本身以及周围环境,具有其独有的特征,所以在车牌的定位过程中,一般采取的办法是是利用车牌区域的特征来判断牌照,将车牌区域从整幅车辆图像中分割出来。现有的车牌定位流程基本可以总结如下,首先将图片进行灰度转换，通过对灰度图像进行二值化、边缘检测等处理获得较理想的供定位图像,然后对该图像进行垂直和水平方向的闭运算,结合车辆牌照特点,进行颜色识别,对车辆牌照进行定位。然后通过边缘检测技术，分割出定位的车辆牌照区域。

根据本次的具体实验，本文主要采用基于颜色特征的定位方法，并利用几何大小比例筛选所有轮廓，最终得出定位结果。

# 实验方案

## 2.1 数据集

本次实验使用车牌定位2021的数据集，共75张汽车图片。其中部分图片大小为1080×1440，其余则为1440×1080大小。仅一张图片为绿色车牌，其余为1至4张左右的蓝色车牌，部分图片拍摄与夜晚。

## 2.2 实验流程

根据传统车牌定位的流程，本文主要分为两个部分实现对数据集的定位：图片预处理与轮廓处理。

其中，图片预处理主要采用了基于边缘特征的与基于颜色特征的，而根据部分图片特性，在基于颜色特征处理图像时又针对性的对绿色车牌、蓝色车牌以及较暗的图片设计了不同的方法。

图片预处理成功后，则可以对二值图像进行轮廓识别与原图的车牌圈定。其中，对于识别错误的轮廓，则利用几何大小比例的方法进行筛选。最后，在进行定位时只需调用相应的函数即可。以下是本次实验的流程图：

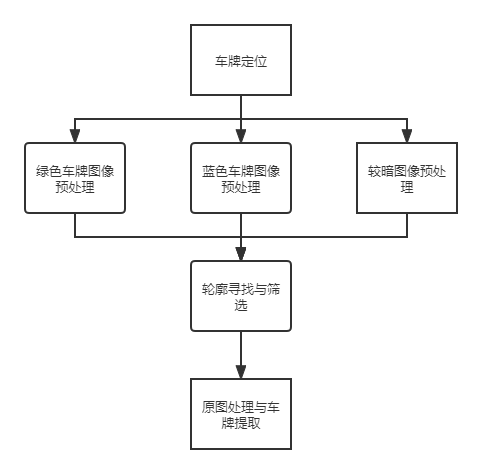


图2.2.1 车牌定位流程图

## 2.3 方法设计

### 2.3.1 图像预处理

在图像预处理环节，流行的方法有基于边缘特征与基于车牌颜色特征的。

基于边缘特征的图像处理方法先利用高斯模糊进行降噪，再对灰度图利用sobel边缘检测方法，使车牌边缘更加突出，然后利用阈值分割获得二值图像，并对二值图像进行分别进行一次闭运算与开运算操作获得更好结果。在调用该方法时，只需输入需要处理的图片，便可以获得处理结果的二值图。

由于测试的图片都有比较复杂的背景，单独使用该方法并不能获得较好的结果，而若能与基于颜色特征的处理方法结合，则会达到两者都不能达到的更好效果。以下是其中一张图片的测试结果：

图2.3.1.1 原图 图2.3.1.2 sobel（边缘检测）图

图2.3.1.3基于颜色(color)特征处理图 图2.3.1.4 sobel and color



图2.3.1.4 处理结果图

除了基于边缘的预处理方法，在车牌定位中最常使用的则是基于颜色的预处理方法，且事实证明，即使单独利用颜色进行车牌定位的图像预处理都可以得到比较精确的结果了。这里详细介绍蓝色车牌图像的预处理方法：首先还是进行高斯模糊去除噪声，然后可以直接利用opencv的inRange函数获得图像中偏蓝色部分的掩膜；再将原图转化为hsv图，获得颜色饱和度较大部分掩膜，与原颜色掩膜取交并膨胀减小空隙获得最终结果。对于绿色车牌，则只需要更改颜色掩膜的数据即可；对于部分拍摄环境较暗的图像，则先进行图像增强。在使用函数时，可选是否对绿色车牌与较暗环境处理，若不选择，则默认为蓝色车牌图像。以下是单独利用颜色处理图像的结果：

图2.3.1.4 基于颜色处理图 图2.3.1.4 处理结果图

### 2.3.2 轮廓寻找与筛选

轮廓寻找主要借助与opencv的findContours函数，可以找到二值图像中的全部轮廓，再利用矩形将其框出即可得到候选轮廓。

对于候选轮廓，首先根据其大小进行第一次筛选，图像中的车牌大小一般都大于1000，所以，可以首先将大部分较小的轮廓排除，个别较小车牌则单独设立数值处理；除此之外，利用车牌长宽比进行排除，由于实际图像中车牌有倾斜较大的情况，所以长宽比并不能完全按照标准数值进行排除比较，这里设置的为0.5至6.0是比较合适的范围。

# 实验结果与分析

大部分图像使用相同的参数、方法可以获得较好的定位结果：

图3.1 基于颜色处理图

但由于不同图像情况不同，所以无法完全使用相同的方法进行处理，所以，对于一些特殊图片可以进行单独处理，单独处理同样可以达到比较理想的处理结果。如绿色车牌使用不同的掩膜参数，黑暗环境下拍摄得到的图像先进行图像增强，部分有颜色干扰的图像使用边缘检测与颜色特征结合的方式进行预处理，有无法排除的干扰则可以利用图像裁剪先一步将干扰部分排除，仅留下车牌部分位置，或者调整部分参数从而排除影响。通过以上方法同样可以处理得到部分优良结果：

图3.2 绿色车牌处理图 图3.3 黑暗环境下处理图

图3.4 边缘检测与颜色特征结合处理图 图3.5 根据具体情况更改筛选参数

同时，存在以上方法也无法解决的情况。如下图在过于黑暗的情况下，即使进行图像增强也无法良好的与周围环境分离，利用以上方法不能获得成功定位的结果，除此之外，部分图像干扰较大或者车牌倾斜程度较大也会导致定位失败。

图3.6 部分识别不准图片

# 结论

利用以上方法对2021车辆图像定位，可以对简单图获得良好的定位结果，对于背景复杂图则可以利用提供函数自定义定位方法，同样可以获得定位结果。最终有10张仍然存在定位问题的图像，已在文件夹中显示出，敬请老师指教。

# 附录(源码，更多的实验结果等)：

实验完整源码与结果已上传至github: <https://github.com/gutu1/License_plate_location_2021.git>

同样可以根据以下代码了解：

import os

import numpy as np

import cv2

import random

path = './License\_plate\_data\_set\_2021/'

imgName = os.listdir(path)

#部分筛选条件

minArea = 1000

minPlateRatio = 0.5

maxPlateRatio = 6.0

#使用sobel（边缘检测）进行预处理

def img\_pro\_sobel(imgName):

    img\_gaus = cv2.GaussianBlur(imgName,(5,5),0)

    img\_gray = cv2.cvtColor(img\_gaus,cv2.COLOR\_BGR2GRAY)

    sobel = cv2.Sobel(img\_gray, cv2.CV\_8U, 1, 0)

    \_,img\_thresh = cv2.threshold(sobel,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

    img\_edge = cv2.Canny(img\_thresh,100,200)

    # 膨胀和腐蚀操作的核函数

    kernel = np.ones((10, 19), np.uint8)

    imgEdge = cv2.morphologyEx(img\_edge, cv2.MORPH\_CLOSE, kernel)

    imgEdge = cv2.morphologyEx(imgEdge, cv2.MORPH\_OPEN, kernel)

    return imgEdge

#基于车牌颜色特征进行预处理

def img\_pro\_color(imgName,green = False, enhance = False):

    '''

    :param imgName: 原图像

    :param green: 是否是绿色车牌预处理，可选参数，默认为False

    :param enhance: 是否是黑暗环境图像预处理，可选参数，默认为False

    '''

    if(green):

        img\_gaus = cv2.GaussianBlur(imgName,(5,5),0)

        mask\_color = cv2.inRange(img\_gaus,(0,100,0),(140,200,140))

    elif(enhance):

        kernel = np.array([[0, -1, 0], [0, 5, 0], [0, -1, 0]])

        img\_enhance = cv2.filter2D(imgName,-1,kernel=kernel)

        mask\_color = cv2.inRange(img\_enhance,(100,0,0),(255,140,140))

    else:

        img\_gaus = cv2.GaussianBlur(imgName,(5,5),0)

        mask\_color = cv2.inRange(img\_gaus,(100,0,0),(200,140,140))

    img\_hsv = cv2.cvtColor(imgName,cv2.COLOR\_BGR2HSV)

    \_,s,\_ = cv2.split(img\_hsv)

    mask\_s = cv2.inRange(s,80,255)

    img\_result = mask\_color & mask\_s

    kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (15, 3))

    img\_dilate = cv2.dilate(img\_result, kernel, 3)   # 膨胀 ，减小车牌空洞

    return img\_dilate

#轮廓寻找与筛选

def find\_location(img,img\_dilate,minArea=1000,ra\_screen = True):

    '''

    :param img: 原图像

    :param img\_dilate: 预处理后的二值图像

    :param minArea: 筛选的最小矩形面积，可选参数，默认为1000

    :param ra\_screen: 是否进行长宽比筛选，可选参数，默认为True

    '''

    contours,\_ = cv2.findContours(img\_dilate,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

    region = []

    count = 0

    for i in range(len(contours)):

        contour = contours[i]

        area = cv2.contourArea(contour)

        rect = cv2.minAreaRect(contour)

        if(area < minArea):

            continue

        box = np.int32(cv2.boxPoints(rect))

        if(ra\_screen):

            width, height = box[1]

            if(height == 0):

                continue

            ratio = float(width) / float(height)

            if(ratio > maxPlateRatio or ratio < minPlateRatio):

                continue

        count = count + 1

        Xs = [i[0] for i in box]

        Ys = [i[1] for i in box]

        x1 = min(Xs)

        x2 = max(Xs)

        y1 = min(Ys)

        y2 = max(Ys)

        hight = y2 - y1

        width = x2 - x1

        plate = img[y1:y1+hight, x1:x1+width]

        img\_name = file.split('.')

        save\_path = "./plate/"+img\_name[0]+"("+str(count)+").jpg"

        cv2.imwrite(save\_path,plate)

        region.append(box)

    img\_result = np.copy(img)

    cv2.drawContours(img\_result, region, -1, (0, 255, 0), 2)

    save\_path = "./result/"+file

    cv2.imwrite(save\_path,img\_result)

    return img\_result

out = []

for file in imgName:

    img = cv2.imread(path+file)

    if(file == "IMG20211116093217.jpg" or

        file == "IMG20211116215933.jpg" or

        file == "IMG20211116215958.jpg" or

        file == "IMG20211117090658.jpg" or

        file == "IMG20211117120424.jpg" or

        file == "IMG20211117120435.jpg"):

        img = img[200:1000,200:1000]

    if(file == "IMG20211116093237.jpg"):

        img\_cut = img[400:700,400:700]

        img\_pro = img\_pro\_color(img\_cut,green=True)

        img\_pro = cv2.copyMakeBorder(img\_pro,400,740,400,380,cv2.BORDER\_REPLICATE)

        img\_result = find\_location(img,img\_pro)

    elif(file == "IMG20211117090817.jpg"):

        img\_cut = img[:,0:500]

        img\_pro = img\_pro\_color(img\_cut)

        img\_result = find\_location(img,img\_pro,ra\_screen=False)

    elif(file == "IMG20211117090832.jpg" or

        file == "IMG20211117120237.jpg" or

        file == "IMG20211117120312.jpg" or

        file == "IMG20211116093021.jpg" or

        file == "IMG20211117120437.jpg" or

        file == "IMG20211118180119.jpg" or

        file == "IMG20211118180054.jpg" or

        file == "IMG20211118142702.jpg" or

        file == "IMG20211116093058.jpg" or

        file == "IMG20211116093258.jpg" or

        file == "IMG20211117090658.jpg" or

        file == "IMG20211117120424.jpg"

        ):

        img\_color = img\_pro\_color(img)

        img\_sobel = img\_pro\_sobel(img)

        img\_pro = img\_color & img\_sobel

        img\_result = find\_location(img,img\_pro,500)

    else:

        img\_pro = img\_pro\_color(img)

        if(file == "IMG20211117090625.jpg" or file == "IMG20211117090832.jpg"):

            img\_result = find\_location(img,img\_pro,250)

        elif(file == "IMG20211118142540.jpg"):

            img\_result = find\_location(img,img\_pro,500)

        else:

            img\_result = find\_location(img,img\_pro)

            if((img\_result == img).all()):

                img\_pro = img\_pro\_color(img,enhance=True)

                img\_result = find\_location(img,img\_pro)

            out.append(img\_result)

#rand = random.randint(0,len(out))

#cv2.imshow("random img",np.array(out[rand]))

#cv2.waitKey(0)

参考文献

代码参考：

[1][afeiererer](https://blog.csdn.net/afeiererer).https://blog.csdn.net/afeiererer/article/details/86537935，2019

[2][weixin\_34185560](https://blog.csdn.net/weixin_34185560).https://blog.csdn.net/weixin\_34185560/article/details/88024806,2018