杭州電子科技大学

HANGZHOU DIANZI UNIVERSITY

数字图像期末大作业

——基于 OpenCV 的车牌识别

学　院：　计算机学院

专　业：　计算机科学与技术

学　号：　22050102

姓　名：　赵妍

目　录

一. 程序简介

车牌识别系统是一种应用计算机视觉技术来自动识别车辆车牌号码的系统。这类系统广泛应用于交通监控、停车场管理、高速公路收费以及城市交通管理等领域。本程序结合数字图像处理的相关知识，基于 python 与 OpenCV 实现了对图片中车牌的简单识别，并将识别到的车牌进行了矫正处理，最终实现一个可视化界面，获得车牌识别过程并且打印出车牌识别结果及其背景颜色。

二. 程序设计思路

2.1 图像预处理

本程序首先根据路径读取输入图像，接着对输入的图像进行基本的预处理，预处理是指对采集的车牌图像进行大小规范化、降噪、彩色图像灰度化、腐蚀膨胀等，使图片牌照区域质量得到改善，保留车牌区域信息，去除噪声。在本程序中，限定图像的宽度不能超过 1000 像素，若图像大小超出了限制的大小，则等比地对图像进行缩放处理。获得符合条件大小的图像后，再使用高斯模糊函数 *cv2.GaussianBlur*对图像进行降噪处理，减少噪声对后续操作的干扰。最后，将得到的图像进行灰度化，并保存到固定路径，方便后续操作的进行。预处理的效果如下图显示：

原始图像

预处理后的图像

2.2 图像轮廓提取

在预处理完图像后，下一步是对图像轮廓的提取。为使车牌轮廓特征更加明显，先对图像进行形态学处理，即先对图像进行一次开运算，去除噪声，除去孤立的小点，毛刺和小桥， 再将所得到的图像与原先的灰度图像进行加权融合，强调原图像中的某些特征，最终得到如下图所示的结果：

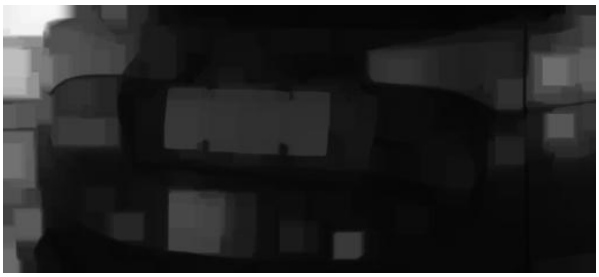


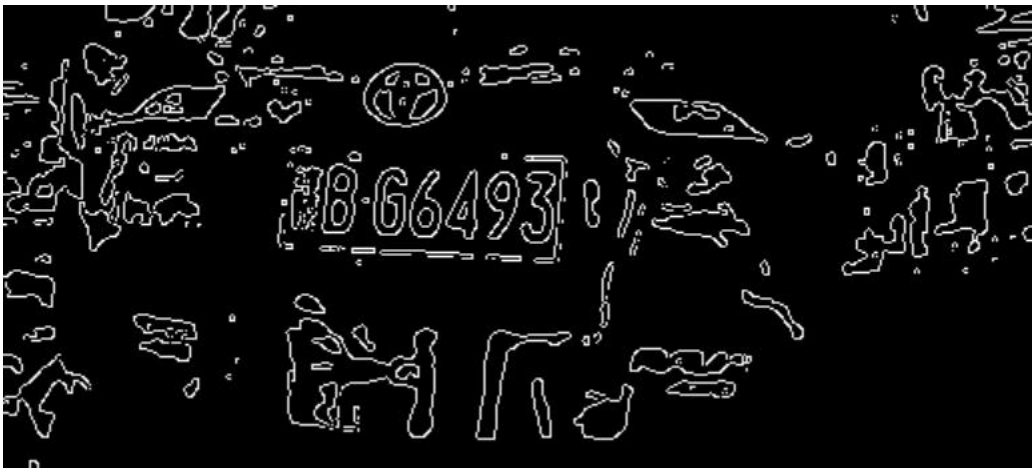
图1

经过开运算后的图像

图2

融合后的图像

接下来我们要提取图像的边缘。首先使用Otsu方法对图像进行阈值处理，将图像转换为二值图像（img_thresh），像素值是0或255。接下来，对阈值处理后的图像应用 Canny 边缘检测算法。边缘检测结果如下图所示。



边缘检测后的图像

经过边缘检测后的图像，我们再次对其进行形态学处理，先进行一次闭运算填补小孔和裂缝，再进行一次开运算除去小点及毛刺，最终得到较为明显的区域块。形态学处理实现效果如下：、

形态学处理后图像

2.3 车牌的提取

接下来任务是在所得到的轮廓中获得车牌的轮廓。首先定义蓝色、黄色和绿色在 HSV 颜色空间中的阈值范围。这是因为车牌通常是这些颜色。然后将图像从 BGR 颜色空间转换到 HSV。利用cv2.inRange函数和设定的颜色阈值过滤出特定颜色。用于识别图像中对应颜色的区域。接下来进行图像融合和处理，通过cv2.bitwise_and操作结合掩膜，提取出这些颜色的区域，并将提取结果转换为灰度图像。结果如下图：

颜色过滤车牌

灰度图像

然后使用闭运算和开运算进行形态学变换，轮廓检测，使用img_findContours函数在处理后的图像上查找轮廓。找出可能包含车牌的区域。对于每个检测到的轮廓，获取其最小外接矩形，并在原始图像上绘制轮廓。结果如下图：

定位到的车牌

2.4 车牌的矫正

车牌位置和角度不同会造成畸变，为了后续的字符识别步骤有更准确的输入图像，我们需要进行车牌矫正。首先，对每个车牌轮廓进行角度处理处理，然后获取矩形的四个角点。接着根据车牌的倾斜角度（正或负），选择不同的角点集合来计算仿射变换矩阵。然后使用cv2.warpAffine函数应用仿射变换矩阵，矫正车牌图像。

图(1)原图像

图(2)仿射变换后的图像

2.5 确定车牌颜色和裁剪车牌图像

通过分析图像的颜色分布来识别车牌的颜色，并据此裁剪图像。
对每个输入的车牌图像，首先将其从BGR颜色空间转换为HSV颜色空间来进行车牌识别。HSV颜色空间更适合于颜色识别，因为它将颜色和亮度信息分离。然后统计颜色像素，基于颜色像素的计数，确定每个车牌图像的主要颜色。如果某种颜色的像素数达到总像素数的一定比例，那么这种颜色就被认为是车牌的主要颜色。例如，如果黄色像素占总像素数的一半以上，则车牌颜色被认为是黄色。确定主要颜色后，代码会根据颜色类型进行裁剪。对于非绿色车牌，会进一步裁剪图像，以便更好地聚焦于车牌区域。裁剪的具体方式依赖于颜色识别的结果。初始裁剪不准确或车牌边缘不清晰时，会进行更精确的车牌定位。最后得到包含每个车牌图像的颜色类别（例如“yellow”、“green”、“blue”），和包含相应的裁剪后的车牌图像。

图(1) 裁剪后车牌图像1

图(2) 裁剪后车牌图像2

2.6 消除裁剪后车牌图像的误差图像并生成更清晰的灰度图

首先，根据之前生成的车牌图像的颜色类别，检查每张车牌图像的颜色（"blue", "yellow", "green"），消除上述裁剪后的车牌图像2，来得到正确的车牌图像。选定的车牌图像接着被转换为灰度图像，灰度图像简化了图像数据，只保留亮度信息，这对于后续的处理步骤非常有用。对于黄色和绿色车牌，由于字符通常比背景暗，与蓝色车牌相反，所以灰度图像会进行二进制位反转操作。这有助于在接下来的步骤中更清晰地区分字符和背景。然后，应用Otsu's method进行二值化处理，将图像转换成只有黑色和白色的形式，以便更容易地识别出图像中的文字。最后得到的灰度图对比上述灰度图，字符更清晰，质量更高。

图(1) 最后得到的车牌图像

图(2)最后得到的灰度图

2.7 定位车牌具体字符

首先确定水平方向上的最大波峰（即车牌字符所在的区域），根据这个波峰的起始和结束位置来裁剪灰度图像。然后通过垂直直方图来进

一步定位字符的垂直位置。这样的裁剪有助于聚焦于车牌的主要部分，也就是包含字符的区域。然后通过计算垂直直方图，确定垂直阈值，进行垂直波峰检测，来进行波峰分析，如果波峰数量小于6个，则跳过当前图像，处理下一个，否则计算距离最远的两个波峰的距离，并处理左侧边缘的波峰，为下一步字符识别打下基础。

裁剪后的灰度图

2.8 提取车牌具体字符

我们最终的目的是得到车牌号，所以要对车牌进行分割，提取车牌具体字符，并通过机器学习模型进行识别。我们先对相邻且距离较近的波峰，进行合并，以正确地识别组合在一起的字符，例如汉字。根据确定的波峰，将车牌上的每个字符分割为独立的图像块。对每个字符图像进行处理，包括调整大小和应用预处理。然后使用机器学习模型进行字符识别。对于第一个字符（通常是省份标识），使用专门的中文识别模型。对于其他字符，使用通用的字符识别模型。并进行特殊情况处理，检查并排除车牌上的非字符元素，如铆钉或车牌边缘。最后将识别出的字符组合成完整的车牌号码。下图为字符分割后的图像。

字符分割后图像

2.9 提供可视化界面

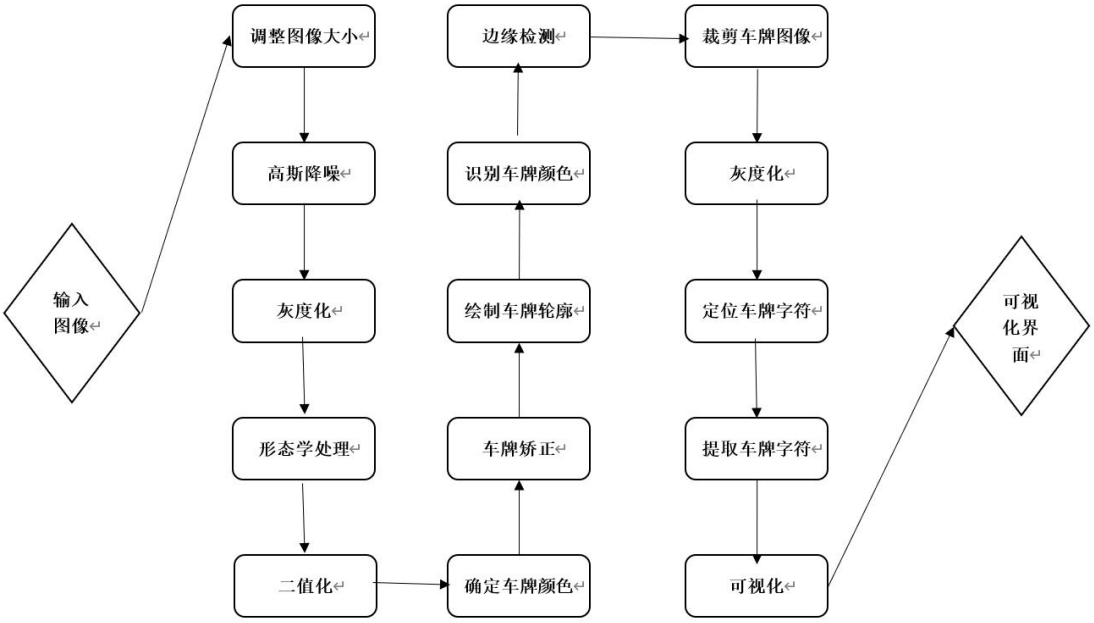
实现车牌识别系统的图形用户界面（GUI），使用 Python 的 Tkinter 库。这个 GUI 应用程序展示了车牌识别过程的不同阶段，包括灰度变化、边缘检测、形态学处理、车牌定位和字符分割。从本地读取图像，并给出定位车牌的图像和识别结果和颜色结果。并允许清理数据，重新进行车牌识别。

图1　主界面

图2 图像处理关键过程

三. 程序流程图

本程序的流程图如下图所示：



四. 程序代码

本程序的代码如下：

```
import os
import cv2
from PIL import Image
import numpy as np
import function
import img_recognition
SZ = 20  # 训练图片长宽
MAX_WIDTH = 1000  # 原始图片最大宽度
# Min_Area = 2000  # 车牌区域允许最大面积
PROVINCE_START = 1000


class StatModel(object):
    def load(self, fn):
        self.model = self.model.load(fn)

    def save(self, fn):
        self.model.save(fn)

#车牌识别中的机器学习任务
```

```
class SVM(StatModel):
    def __init__(self, C=1, gamma=0.5):
        self.model = cv2.ml.SVM_create()
        self.model.setGamma(gamma)
        self.model.setC(C)
        self.model.setKernel(cv2.ml.SVM_RBF)#核类型（RBF核）
        self.model.setType(cv2.ml.SVM_C_SVC)

    # 训练svm
    def train(self, samples, responses):
        self.model.train(samples, cv2.ml.ROW_SAMPLE, responses)

    # 字符识别
    def predict(self, samples):
        #预测输入样本的类别
        r = self.model.predict(samples)
        return r[1].ravel()


class CardPredictor:
    def __init__(self):
        pass

    def train_svm(self):
        # 识别英文字母和数字
        self.model = SVM(C=1, gamma=0.5)
        # 识别中文
        self.modelchinese = SVM(C=1, gamma=0.5)
        if os.path.exists("svm.dat"):
            self.model.load("svm.dat")
        if os.path.exists("svmchinese.dat"):
            self.modelchinese.load("svmchinese.dat")

    def img_first_process(self, car_pic_file):
        """
        :param car_pic_file: 图像文件
        :return:已经处理好的图像文件，二值图像 原图像文件
        """
        if type(car_pic_file) == type(""):
            img = function.img_read(car_pic_file)   #读取文件
        else:
            img = car_pic_file
        # 优化处理速度和效果，因为过大的图像可能不必要地增加计算负担
        pic_hight, pic_width = img.shape[:2]  #取彩色图片的高、宽
        if pic_width > MAX_WIDTH:
            resize_rate = MAX_WIDTH / pic_width
            # 缩小图片
            img = cv2.resize(img, (MAX_WIDTH, int(pic_hight * resize_rate)), interpolation=cv2.INTER_AREA)
        # 关于interpolation 有几个参数可以选择:
        # cv2.INTER_AREA - 局部像素重采样，适合缩小图片。
```

```
71.             # cv2.INTER_CUBIC和 cv2.INTER_LINEAR 更适合放大图像，其中INTER_LINEAR为
    默认方法。
72.
73.             img = cv2.GaussianBlur(img, (5, 5), 0)
74.             # 高斯滤波是一种线性平滑滤波，对于除去高斯噪声有很好的效果
75.             # 0 是指根据窗口大小（ 5,5 ）来计算高斯函数标准差
76.
77.
78.             oldimg = img
79.             # 转化成灰度图像
80.             # 转换颜色空间 cv2.cvtColor
81.             # BGR ---> Gray  cv2.COLOR_BGR2GRAY
82.             # BGR ---> HSV  cv2.COLOR_BGR2HSV
83.             img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
84.
85.             cv2.imwrite("./tmp/img_gray.jpg", img)
86.
87.             #ones()返回一个全1的n维数组
88.             Matrix = np.ones((20, 20), np.uint8)
89.
90.             # 开运算:先进行腐蚀再进行膨胀就叫做开运算。它被用来去除噪声
    。 cv2.MORPH_OPEN
91.             img_opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, Matrix)
92.             cv2.imwrite("./tmp/img_opening.jpg", img_opening)
93.             # 图片叠加与融合
94.             # g (x) = (1 − α)f0 (x) + αf1 (x)   a→（0，1）不同的a值可以实现不同的效
    果
95.             # 图像融合
96.             img_opening = cv2.addWeighted(img, 1, img_opening, -1, 0)
97.             cv2.imwrite("./tmp/img_ronghe.jpg", img_opening)
98.             # 创建20*20的元素为1的矩阵 开操作，并和img重合
99.
100.
101.            # Otsu’s二值化
102.            ret, img_thresh = cv2.threshold(img_opening, 0, 255, cv2.THRESH_BINA
    RY + cv2.THRESH_OTSU)
103.            # Canny 边缘检测
104.            # 较大的阈值2用于检测图像中明显的边缘  一般情况下检测的效果不会那么完美，边
    缘检测出来是断断续续的
105.            # 较小的阈值1用于将这些间断的边缘连接起来
106.            img_edge = cv2.Canny(img_thresh, 100, 200)
107.            cv2.imwrite("./tmp/img_edge.jpg", img_edge)
108.
109.            Matrix = np.ones((4, 19), np.uint8)
110.            # 闭运算:先膨胀再腐蚀，闭运算可以填充图像中的小孔，同时也可以连接相邻的物体
111.            img_edge1 = cv2.morphologyEx(img_edge, cv2.MORPH_CLOSE, Matrix)
112.            # 开运算：开运算可以消除图像中的小孔，同时也可以分离相邻的物体。
113.            img_edge2 = cv2.morphologyEx(img_edge1, cv2.MORPH_OPEN, Matrix)
114.            cv2.imwrite("./tmp/img_xingtai.jpg", img_edge2)
115.            return img_edge2, oldimg
116.
117.    def img_only_color(self, filename, oldimg, img_contours):
```

```
        """
        :param filename: 图像文件
        :param oldimg: 原图像文件
        img_contours: 图像轮廓
        :return: 识别到的字符、定位的车牌图像、车牌颜色
        """
        pic_hight, pic_width = img_contours.shape[:2] # #取彩色图片的高、宽
#三个颜色范围的阈值
        lower_blue = np.array([100, 110, 110])
        upper_blue = np.array([130, 255, 255])
        lower_yellow = np.array([15, 55, 55])
        upper_yellow = np.array([50, 255, 255])
        lower_green = np.array([50, 50, 50])
        upper_green = np.array([100, 255, 255])

        # BGR ---> HSV
        hsv = cv2.cvtColor(filename, cv2.COLOR_BGR2HSV)
        # 利用cv2.inRange函数设阈值，去除背景部分
        # 参数1：原图
        # 参数2：图像中低于值，图像值变为0
        # 参数3：图像中高于值，图像值变为0
        mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
        mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
        mask_green = cv2.inRange(hsv, lower_yellow, upper_green)

        # 图像算术运算  按位运算 按位操作有： AND， OR， NOT， XOR 等
        output = cv2.bitwise_and(hsv, hsv, mask=mask_blue + mask_yellow + mask_green)
        # 根据阈值找到对应颜色
        cv2.imwrite("./tmp/img_corlor.jpg", output)
        output = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)
        cv2.imwrite("./tmp/img_huidu.jpg", output)
        Matrix = np.ones((20, 20), np.uint8)
        #使用一个 20x20 的卷积核
        img_edge1 = cv2.morphologyEx(output, cv2.MORPH_CLOSE, Matrix)  #闭运算
        img_edge2 = cv2.morphologyEx(img_edge1, cv2.MORPH_OPEN, Matrix) #开运算
#使用function.img_findContours函数找到边缘轮廓
        card_contours = function.img_findContours(img_edge2)
        # for rect in card_contours :
        #     box = cv2.boxPoints(rect)  # 获取轮廓的四个角点
        #     box = np.int0(box)  # 转换为整数
        #     cv2.drawContours(oldimg, [box], 0, (0, 255, 0), 2)  # 在原始图像上绘制轮廓
        # cv2.imwrite("./tmp/img_contour.jpg",oldimg)
        # function.img_Transform函数对轮廓进行变换。
        card_imgs = function.img_Transform(card_contours, oldimg, pic_width, pic_hight)
        i=0
        # for item in card_imgs:
        #     i=i+1
```

```
165.          #     cv2.imwrite("./tmp/img_jiaozheng{}.jpg".format(i), item)
166.          # 使用function.img_color函数获取颜色信息。
167.          colors, car_imgs = function.img_color(card_imgs)
168.          # cv2.imwrite("./tmp/img_yansemay.jpg",car_imgs[0])
169.          # cv2.imwrite("./tmp/img_yansemay1.jpg", car_imgs[1])
170.          predict_result = []
171.          predict_str = ""
172.          roi = None
173.          card_color = None
174.          # 遍历颜色列表，找到蓝色、黄色和绿色的颜色，然后对对应的卡片图像进行灰度转换。如果灰度转换失败，会打印
175.          # "gray转换失败"。
176.          for i, color in enumerate(colors):
177.
178.              if color in ("blue", "yello", "green"):
179.                  card_img = card_imgs[i]
180.                  cv2.imwrite("./tmp/img_chepailast.jpg",card_img)
181.                  try:
182.                      gray_img = cv2.cvtColor(card_img, cv2.COLOR_BGR2GRAY)
183.                  except:
184.                      print("gray转换失败")
185.
186.                  # 黄、绿车牌字符比背景暗、与蓝车牌刚好相反，所以黄、绿车牌需要反向
187.                  if color == "green" or color == "yello":
188.                      gray_img = cv2.bitwise_not(gray_img)#二进制位运算
189.  #执行该代码后，ret表示阈值计算的结果，gray_img为经过二值化处理后的图像。
190.                  ret, gray_img = cv2.threshold(gray_img, 0, 255, cv2.THRESH_B
      INARY + cv2.THRESH_OTSU)
191.                  cv2.imwrite("./tmp/img_yansegray.jpg", gray_img)
192.                  x_histogram = np.sum(gray_img, axis=1)
193.                  x_min = np.min(x_histogram)
194.                  x_average = np.sum(x_histogram) / x_histogram.shape[0]
195.                  x_threshold = (x_min + x_average) / 2
196.                  wave_peaks = function.find_waves(x_threshold, x_histogram)
197.                  #如果波形峰值数量为0，则跳过当前循环，继续处理下一个车牌图像。
198.                  if len(wave_peaks) == 0:
199.                      # print("peak less 0:")
200.                      continue
201.                  # 认为水平方向，最大的波峰为车牌区域
202.                  # 这段代码的作用是找到wave_peaks列表中，元素值之差最大的元组。其中，wave_peaks是一个包含多个元组的列表，每个元组有两个元素，表示波形峰值的起始和结束位置。
203.                  wave = max(wave_peaks, key=lambda x: x[1] - x[0])
204.
205.                  gray_img = gray_img[wave[0]:wave[1]]
206.                  # 查找垂直直方图波峰
207.                  row_num, col_num = gray_img.shape[:2]
208.                  # 去掉车牌上下边缘1个像素，避免白边影响阈值判断
209.                  gray_img = gray_img[1:row_num - 1]
210.                  cv2.imwrite("./tmp/img_yansegray1.jpg", gray_img)
211.                  #计算子图像的垂直直方图，并找到直方图中的波峰。
212.                  y_histogram = np.sum(gray_img, axis=0)
```

```
                y_min = np.min(y_histogram)
                y_average = np.sum(y_histogram) / y_histogram.shape[0]
                y_threshold = (y_min + y_average) / 5  # U和0要求阈值偏小，否则U和0会被分成两半
                wave_peaks = function.find_waves(y_threshold, y_histogram)
                #如果波峰的数量小于6，那么跳过当前循环，继续处理下一个子图像。
                if len(wave_peaks) < 6:
                    # print("peak less 1:", len(wave_peaks))
                    continue
                # 如果波峰的数量大于等于6，那么找到波峰中距离最远的两个波峰，计算它们的距离，
                # 并将结果存储在变量max_wave_dis中。
                wave = max(wave_peaks, key=lambda x: x[1] - x[0])
                max_wave_dis = wave[1] - wave[0]
                # 判断是否是左侧车牌边缘
                # 通过判断第一个波形的宽度是否小于最大波形距离的三分之一且起始位置为0来判断是否是左侧车牌边缘
                if wave_peaks[0][1] - wave_peaks[0][0] < max_wave_dis / 3 and wave_peaks[0][0] == 0:
                    wave_peaks.pop(0)

                # 组合分离汉字
                # 代码使用一个循环来遍历剩余的波形。在循环中，它计算当前波形与前一个波形之间的距离，
                # 并与最大波形距离的60 % 进行比较。如果当前波形与前一个波形之间的距离大于最大波形距离的60 %，
                # 则跳出循环。否则，将当前波形与前一个波形的距离累加到cur_dis变量中。
                cur_dis = 0
                for i, wave in enumerate(wave_peaks):
                    if wave[1] - wave[0] + cur_dis > max_wave_dis * 0.6:
                        break
                    else:
                        cur_dis += wave[1] - wave[0]
                        # 如果循环中有至少一个波形被处理（即i > 0），
                        # 则将第一个波形与最后一个被处理的波形组合成一个新的波形，并将其插入到列表的开头。
                        # 这样，汉字就被正确地组合在一起了。
                if i > 0:
                    wave = (wave_peaks[0][0], wave_peaks[i][1])
                    wave_peaks = wave_peaks[i + 1:]
                    wave_peaks.insert(0, wave)

                point = wave_peaks[2]
                point_img = gray_img[:, point[0]:point[1]]
                if np.mean(point_img) < 255 / 5:
                    wave_peaks.pop(2)

                if len(wave_peaks) <= 6:
                    # print("peak less 2:", len(wave_peaks))
                    continue
                # print(wave_peaks)
```

```
257.
258.                 # wave_peaks  车牌字符 类型列表 包含7个（开始的横坐标，结束的横坐标）
259.
260.
261.
262.                 part_cards = function.seperate_card(gray_img, wave_peaks)
263.
264.                 for i, part_card in enumerate(part_cards):
265.                     # 可能是固定车牌的铆钉
266.
267.                     if np.mean(part_card) < 255 / 5:
268.                         # print("a point")
269.                         continue
270.                     part_card_old = part_card
271.
272.                     w = abs(part_card.shape[1] - SZ) // 2
273.
274.                     part_card = cv2.copyMakeBorder(part_card, 0, 0, w, w, cv2.BORDER_CONSTANT, value=[0, 0, 0])
275.                     part_card = cv2.resize(part_card, (SZ, SZ), interpolation=cv2.INTER_AREA)
276.
277.                     part_card = img_recognition.preprocess_hog([part_card])
278.                     if i == 0:
279.                         resp = self.modelchinese.predict(part_card)
280.                         charactor = img_recognition.provinces[int(resp[0]) - PROVINCE_START]
281.                     else:
282.                         resp = self.model.predict(part_card)
283.                         charactor = chr(resp[0])
284.                     # 判断最后一个数是否是车牌边缘，假设车牌边缘被认为是1
285.                     if charactor == "1" and i == len(part_cards) - 1:
286.                         if part_card_old.shape[0] / part_card_old.shape[1] >= 7:  # 1太细，认为是边缘
287.                             continue
288.                     predict_result.append(charactor)
289.                     predict_str = "".join(predict_result)
290.
291.                 roi = card_img
292.                 card_color = color
293.                 break
294.         # cv2.imwrite("./tmp/img_caijian.jpg", roi)
295.         # print(predict_str)
296.         return predict_str, roi, card_color  # 识别到的字符、定位的车牌图像、车牌颜色
```

其中用到的自己写的函数

```
1.     def img_findContours(img_contours):
2.         # 查找轮廓
3.         # 使用cv2.findContours
```

```
        # 函数查找图像中的轮廓。这个函数有三个参数：输入图像、轮廓检索模式（cv2.RETR_TREE
        # 表示建立轮廓的层次结构）和轮廓近似方法（cv2.CHAIN_APPROX_SIMPLE
        # 表示压缩轮廓点，仅保留端点）。
        contours, hierarchy = cv2.findContours(img_contours, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        # cv2.RETR_TREE建立一个等级树结构的轮廓
        #  cv2.CHAIN_APPROX_SIMPLE压缩水平方向，垂直方向，对角线方向的元素，
        #  只保留该方向的终点坐标，例如一个矩形轮廓只需4个点来保存轮廓信息

        # cv2.contourArea计算该轮廓的面积
        # 筛选出面积大于 Min_Area的轮廓
        # Min_Area应该是一个定义好的最小面积阈值，用于排除过小的轮廓
        contours = [cnt for cnt in contours if cv2.contourArea(cnt) > Min_Area]
        # print("findContours len = ", len(contours))

        # 面积小的都筛选掉
        car_contours = []
        for cnt in contours:
            ant = cv2.minAreaRect(cnt)# 得到最小外接矩形的（中心(x,y), (宽,高), 旋转角度）
            width, height = ant[1]
            #对矩形的宽高进行校正，确保宽大于高。
            if width < height:
                width, height = height, width
            ration = width / height
            # 计算宽高比
            if 2 < ration < 5.5:
                car_contours.append(ant)
                # box = cv2.boxPoints(ant) # 获得要绘制这个矩形的 4 个角点
        # cv2.imwrite("./tmp/img_dingwei.jpg", car_contours)
        # print(type(car_contours), car_contours)
        return car_contours


    def img_Transform(car_contours, oldimg, pic_width, pic_hight):
        """
        进行矩形矫正
        """
        car_imgs = []
        for car_rect in car_contours: #（中心(x,y), (宽,高), 旋转角度）
            if -1 < car_rect[2] < 1:
                angle = 1
                # 对于角度为-1 1之间时，默认为1
            else:
                angle = car_rect[2]
            car_rect = (car_rect[0], (car_rect[1][0] + 5, car_rect[1][1] + 5), angle)

            box = cv2.boxPoints(car_rect) # 获得要绘制这个矩形的 4 个角点

            heigth_point = right_point = [0, 0]
```

```
            left_point = low_point = [pic_width, pic_hight]

            for point in box:
                if left_point[0] > point[0]:
                    left_point = point
                if low_point[1] > point[1]:
                    low_point = point
                if heigth_point[1] < point[1]:
                    heigth_point = point
                if right_point[0] < point[0]:
                    right_point = point

            if left_point[1] <= right_point[1]:  # 正角度
                new_right_point = [right_point[0], heigth_point[1]]
                pts2 = np.float32([left_point, heigth_point, new_right_point])  # 字符只是高度需要改变
                pts1 = np.float32([left_point, heigth_point, right_point])
                #  仿射变换
                M = cv2.getAffineTransform(pts1, pts2)
                dst = cv2.warpAffine(oldimg, M, (pic_width, pic_hight))

                point_limit(new_right_point)
                point_limit(heigth_point)
                point_limit(left_point)

                car_img = dst[int(left_point[1]):int(heigth_point[1]), int(left_point[0]):int(new_right_point[0])]
                car_imgs.append(car_img)

            elif left_point[1] > right_point[1]:  # 负角度
                new_left_point = [left_point[0], heigth_point[1]]
                pts2 = np.float32([new_left_point, heigth_point, right_point])  # 字符只是高度需要改变
                pts1 = np.float32([left_point, heigth_point, right_point])
                M = cv2.getAffineTransform(pts1, pts2)
                dst = cv2.warpAffine(oldimg, M, (pic_width, pic_hight))
                cv2.imwrite("./tmp/img_jiaozheng.jpg", dst)
                point_limit(right_point)
                point_limit(heigth_point)
                point_limit(new_left_point)
                car_img = dst[int(right_point[1]):int(heigth_point[1]), int(new_left_point[0]):int(right_point[0])]
                car_imgs.append(car_img)
        # print(type(car_imgs), car_imgs)
        # cv2.imwrite("./tmp/img_jiaozheng.jpg", car_imgs)
        return car_imgs

    def img_color(card_imgs):
        """
        颜色判断函数
        """
        colors = []
```

```
    for card_index, card_img in enumerate(card_imgs):

        green = yello = blue = black = white = 0
        try:
            card_img_hsv = cv2.cvtColor(card_img, cv2.COLOR_BGR2HSV)
        except:
            print("矫正矩形出错, 转换失败")# 可能原因:上面矫正矩形出错

        if card_img_hsv is None:
            continue
        row_num, col_num = card_img_hsv.shape[:2]
        card_img_count = row_num * col_num

        for i in range(row_num):
            for j in range(col_num):
                H = card_img_hsv.item(i, j, 0)
                S = card_img_hsv.item(i, j, 1)
                V = card_img_hsv.item(i, j, 2)
                if 11 < H <= 34 and S > 34:
                    yello += 1
                elif 35 < H <= 99 and S > 34:
                    green += 1
                elif 99 < H <= 124 and S > 34:
                    blue += 1

                if 0 < H < 180 and 0 < S < 255 and 0 < V < 46:
                    black += 1
                elif 0 < H < 180 and 0 < S < 43 and 221 < V < 225:
                    white += 1
        color = "no"

        limit1 = limit2 = 0
        if yello * 2 >= card_img_count:
            color = "yellow"
            limit1 = 11
            limit2 = 34  # 有的图片有色偏偏绿
        elif green * 2 >= card_img_count:
            color = "green"
            limit1 = 35
            limit2 = 99
        elif blue * 2 >= card_img_count:
            color = "blue"
            limit1 = 100
            limit2 = 124  # 有的图片有色偏偏紫
        elif black + white >= card_img_count * 0.7:
            color = "bw"
        colors.append(color)
        card_imgs[card_index] = card_img

        if limit1 == 0:
            continue
```

```
151.         xl, xr, yh, yl = accurate_place(card_img_hsv, limit1, limit2, color)
152.         if yl == yh and xl == xr:
153.             continue
154.         need_accurate = False
155.         if yl >= yh:
156.             yl = 0
157.             yh = row_num
158.             need_accurate = True
159.         if xl >= xr:
160.             xl = 0
161.             xr = col_num
162.             need_accurate = True
163.
164.         if color == "green":
165.             card_imgs[card_index] = card_img
166.         else:
167.             card_imgs[card_index] = card_img[yl:yh, xl:xr] if color != "gree
    n" or yl < (yh - yl) // 4 else card_img[
168.
                                    yl - (
169.
                                            yh - yl) // 4:yh,
170.
                                    xl:xr]
171.
172.         if need_accurate:
173.             card_img = card_imgs[card_index]
174.             card_img_hsv = cv2.cvtColor(card_img, cv2.COLOR_BGR2HSV)
175.             xl, xr, yh, yl = accurate_place(card_img_hsv, limit1, limit2, co
    lor)
176.             if yl == yh and xl == xr:
177.                 continue
178.             if yl >= yh:
179.                 yl = 0
180.                 yh = row_num
181.             if xl >= xr:
182.                 xl = 0
183.                 xr = col_num
184.         if color == "green":
185.             card_imgs[card_index] = card_img
186.         else:
187.             card_imgs[card_index] = card_img[yl:yh, xl:xr] if color != "gree
    n" or yl < (yh - yl) // 4 else card_img[
188.
                                    yl - (
189.
                                            yh - yl) // 4:yh,
190.
                                    xl:xr]
191.     #colors 包含了每个车牌图像的颜色类别，card_imgs 包含了裁剪后的车牌图像
192.     return colors, card_imgs
193.
194. def seperate_card(img, waves):
```

```
    """
    分离车牌字符
    """
    h , w = img.shape
    part_cards = []
    i = 0
    for wave in waves:
        i = i+1
        part_cards.append(img[:, wave[0]:wave[1]])
        chrpic = img[0:h,wave[0]:wave[1]]

        #保存分离后的车牌图片
        cv2.imwrite('tmp/chechar{}.jpg'.format(i),chrpic)


    return part_cards

```

五. 程序测试实例

	输入	输出（颜色|车牌号）
实例 1		blue	赣BG6493
实例 2		blue	吉AA266G
实例 3		blue	苏EC62N8
实例 4		None	皖A256R2

六. 程序后续改进

6.1 程序缺陷

本程序虽然在一定程度上实现了对车牌的识别，但在经过测试后发现程序存在一定的缺陷，主要缺陷如下：

识别效果依赖原始图像质量。本程序对于背景颜色和车牌颜色相近的图片识别颜色能力过低，不能识别出具体的颜色，定位到车牌。无法对所有的图片进行准确的识别。

车牌矫正函数不够精细。当矩形的旋转角度在 -1 到 1 度之间时，默认将角度设置为 1 度。这种做法不够精确，特别是对于接近于 0 度但不是完全水平的车牌。

参数有硬编码。代码中有一些硬编码的参数（如阈值、卷积核大小等），这会限制程序在不同场景下的适用性。

6.2 后续改进

后续考虑从以下方面进行改进：

为了进一步提升整个车牌识别系统的性能和准确度，可以考虑以下方面的改进：

1. 增强图像预处理

引入更高级的图像预处理技术，例如自适应阈值处理、直方图均衡化等，以提高图像质量。

2. 使用深度学习方法

可以考虑使用深度学习技术，如卷积神经网络（CNN）来进行车牌检测和字符识别，这可以显著提高识别的准确性。

3. 优化车牌定位算法：

使用更复杂的图像分割和特征提取技术来改进车牌定位的准确性和鲁棒性。

4. 多角度和多尺度识别：

对车牌进行多角度和多尺度的分析，以应对不同拍摄条件下的车牌识别问题。

5. 用户交互界面改进：

提高用户界面的友好性和互动性，例如提供车牌校正工具、结果预览和手动调整选项。

七. 个人感想

在完成数字图像处理大作业的过程中，我深刻地体会到了数字图像处理技术的重要性和实用性。通过对图像的预处理、特征提取、分割和识别等方面的学习，我对数字图像处理有了更加全面和深入的了解。以下是我在完成这次大作业过程中的一些感想：

1. 理论与实践相结合：在学习数字图像处理的过程中，我发现理论知识与实际操作是相辅相成的。通过阅读相关教材和论文，我掌握了数字图像处理的基本概念、原理和方法。而在实际操作中，我将所学的理论知识应用到实际问题中，加深了对理论知识的理解，同时也锻炼了自己的动手能力。

2. 编程能力的提升：在完成大作业的过程中，我主要使用了Python编程语言和OpenCV库进行图像处理。通过编写代码实现各种图像处理算法，我不仅提高了自己的编程能力，还学会了如何利用现有的开源库解决实际问题，节省了大量的时间和精力。

3. 创新思维的培养：在完成大作业的过程中，我遇到了许多实际问题，如图像质量差、噪声干扰等。面对这些问题，我学会了如何运用所学知识进行创新思考，寻找解决问题的方法。这种创新思维的培养对我今后的学习和工作具有重要的意义。

4. 对专业的兴趣和热情：通过完成数字图像处理大作业，我对计算机视觉、图像处理等专业领域产生了浓厚的兴趣。我认识到这些技术在现实生活中有着广泛的应用，如人脸识别、自动驾驶等。这激发了我继续深入学习和研究相关领域的动力。

总之，在完成数字图像处理大作业的过程中，我收获了丰富的知识和技能。这次大作业的完成，对我今后的学习和工作具有重要的指导意义。