**网络空间安全学院**

2021秋季学期

《数字图像处理》课程设计报告

车牌自动识别算法

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名1： | 胡亿豪 | 姓名2： | 洪彬 |
| 学号1： | 201941404107 | 学号2： | 201941404106 |
| 日期： |  | | |

目录

**[一、设计任务说明](#_Toc41913821)** [1](#_Toc41913821)

[1.1 课程设计任务 1](#_Toc41913822)

[1.2 开发该系统软件环境及使用的技术说明 1](#_Toc41913823)

[1.3 开发计划 1](#_Toc41913824)

**[二、系统设计](#_Toc41913825)** [1](#_Toc41913825)

[2.1 概要设计 1](#_Toc41913826)

[2.2 详细设计 1](#_Toc41913827)

**[三、系统实施及结果](#_Toc41913828)** [1](#_Toc41913828)

[3.1 系统测试结果 1](#_Toc41913829)

[3.2 算法性能 1](#_Toc41913830)

[3.3 github提交日志 1](#_Toc41913831)

**[四、课程设计总结](#_Toc41913832)** [1](#_Toc41913832)

**一、设计任务说明**

1.1 课程设计任务

利用OpenCV库和其他开源工具，设计并实现车牌自动识别算法，基本功能要求：

1. 对给定的包含有汽车车牌的照片进行处理，利用图像分割算法将目标从背景中分离出来。
2. 对目标图像进行合适的处理，然后利用Tesseract库实现车牌号码的识别，将结果输出。
3. 要求提供比较友好的用户接口，可以对新的图片导入到系统中进行处理，并将结果返回给用户。
4. 要求处理过程的自动化，即输入图像，自动输出车牌信息，无需人去干预。

1.2 开发该系统软件环境及使用的技术说明

Pycharm:python IDE 用于代码主要实现环境还有git使用，上传代码到GitHub中。

Open-cv：是一个开源的计算机视觉库，它提供了很多函数，这些函数非常高效地实现了计算机视觉算法（最基本的滤波到高级的物体检测皆有涵盖）

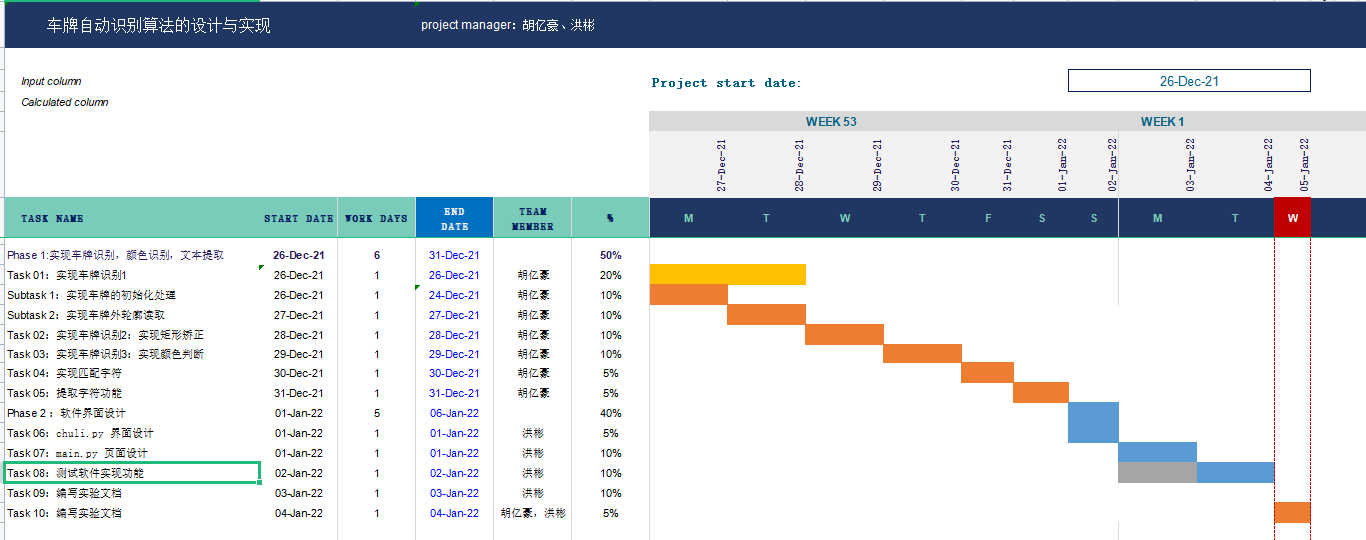
Numpy：是Python语言的一个扩展程序库。支持高阶大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。

Os：os 模块提供了非常丰富的方法用来处理文件和目录。

Tkinter： Tkinter 模块(Tk 接口)是 Python 的标准 Tk GUI 工具包的接口 .Tk 和 Tkinter 可以在大多数的 Unix 平台下使用,同样可以应用在 Windows 和 Macintosh 系统里。提供GUI，使软件有比较好的交互。

1.3 开发计划

这次代码分工比较简单，胡亿豪同学负责有关识别代码的开发，洪彬同学负责有关接口的设计，界面设计，还有相关的交互。



另外附件甘特图.xlsx

**二、系统设计**

2.1 概要设计

img\_function：主要是是实现识别

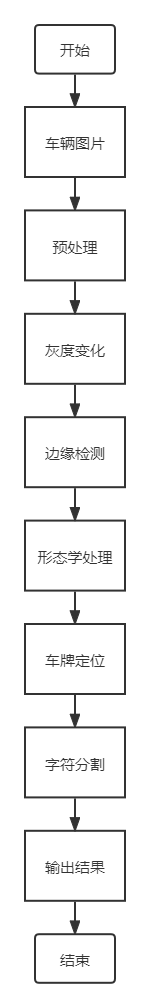
chuli.py与main.py：GUI的设计，一个是详细处理过程的GUI，一个是主页面的GUI。

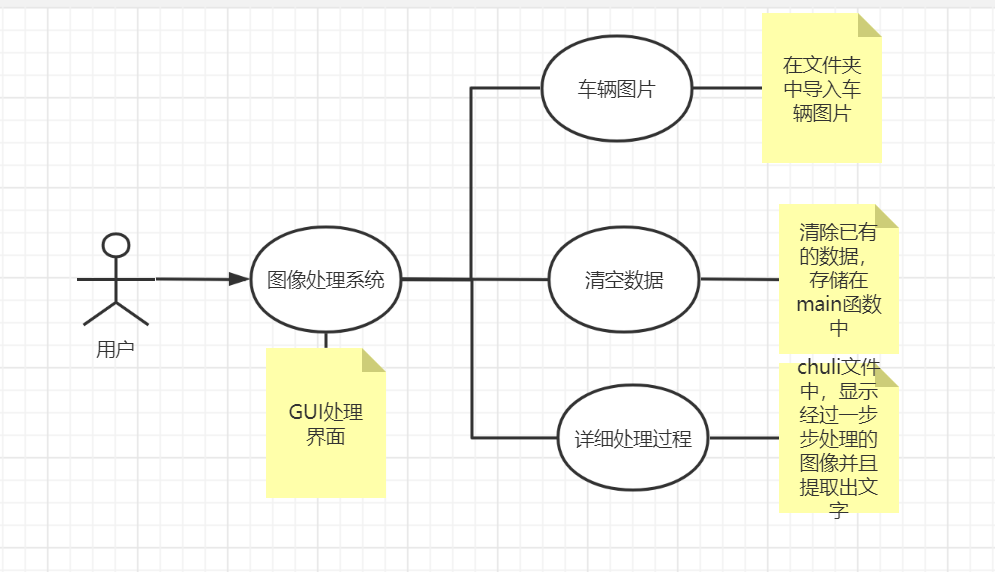
img\_math.py：提供了内置的矩阵转换，颜色转换

img\_regonization.py：文字匹配函数

img\_function.py：识别的核心代码

如果实现车辆车牌的基本检测，可以采用下面的方法，预处理灰度变化可以放在用一个文件中，使用边缘处理可以作为函数进行检测，返回高宽比在一定比例下的图片，在定位到车牌的同时，可以采用先处理车牌上的文字，也是通过进行边缘检测出文字，在进行字符的提取，也可以使用白字黑底识别出文字，还有就是车牌颜色的提取，把这些提取过程合成为一个软件测试界面，用友好的接口导入图像，并且输出结果，同时做一个按钮输出作为图像在每个阶段中作为tmp文件的处理过程。用图像表示如下：  
车辆的基本识别：



GUI的设计：

2.2 详细设计  
int\_fuction文件中，def img\_first\_pre：实现图像的基本处理，去噪还有实现先腐蚀在膨胀的处理找出白色区域。

def img\_first\_pre(self, car\_pic\_file):  
 *"""* ***:param*** *car\_pic\_file: 图像文件* ***:return****:已经处理好的图像文件 原图像文件  
 """* if type(car\_pic\_file) == type(""):  
 img = img\_math.img\_read(car\_pic\_file) # 读取文件  
 else:  
 img = car\_pic\_file  
  
 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。  
 # cv2.INTER\_CUBIC和 cv2.INTER\_LINEAR 更适合放大图像，其中INTER\_LINEAR为默认方法。  
  
 img = cv2.GaussianBlur(img, (5, 5), 0)  
 # 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  
 # 0 是指根据窗口大小（ 5,5 ）来计算高斯函数标准差  
  
 oldimg = img  
 # 转化成灰度图像  
 # 转换颜色空间 cv2.cvtColor  
 # BGR ---> Gray cv2.COLOR\_BGR2GRAY  
 # BGR ---> HSV cv2.COLOR\_BGR2HSV  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
 cv2.imwrite("tmp/img\_gray.jpg", img)  
  
 # ones()返回一个全1的n维数组  
 Matrix = np.ones((20, 20), np.uint8)  
  
 # 开运算:先进性腐蚀再进行膨胀就叫做开运算。它被用来去除噪声。 cv2.MORPH\_OPEN   
 img\_opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, Matrix)  
  
 # 图片叠加与融合  
 # 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  
 img\_opening = cv2.addWeighted(img, 1, img\_opening, -1, 0)  
 # cv2.imwrite("tmp/img\_opening.jpg", img\_opening)  
 # 创建20\*20的元素为1的矩阵 开操作，并和img重合  
  
 # Otsu’s二值化  
 ret, img\_thresh = cv2.threshold(img\_opening, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
 # Canny 边缘检测  
 # 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  
 # 较小的阈值1用于将这些间断的边缘连接起来  
 img\_edge = cv2.Canny(img\_thresh, 100, 200)  
 cv2.imwrite("tmp/img\_edge.jpg", img\_edge)  
  
 Matrix = np.ones((4, 19), np.uint8)  
 # 闭运算:先膨胀再腐蚀  
 img\_edge1 = cv2.morphologyEx(img\_edge, cv2.MORPH\_CLOSE, Matrix)  
 # 开运算  
 img\_edge2 = cv2.morphologyEx(img\_edge1, cv2.MORPH\_OPEN, Matrix)  
 cv2.imwrite("tmp/img\_xingtai.jpg", img\_edge2)  
 return img\_edge2, oldimg

img\_math.py中的img\_findCountours是找出外轮廓为width/high大于2小于5.5的白色区域一般可以看作是车牌的位置。

def img\_findContours(img\_contours):  
 # 查找轮廓  
 # cv2.findContours()   
 # 有三个参数，第一个是输入图像，第二个是轮廓检索模式，第三个是轮廓近似方法。参数为二值图，即黑白的（不是灰度图）  
 # 返回值有三个，第一个是图像，第二个是轮廓，第三个是（轮廓的）层析结构。  
 # 轮廓（第二个返回值）是一个 Python列表，其中存储这图像中的所有轮廓。  
 # 每一个轮廓都是一个 Numpy 数组，包含对象边界点（x，y）的坐标。  
 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  
 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]  
 if width < height:  
 width, height = height, width  
 ration = width / height  
  
 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  
  
 return car\_contours

img\_math.py中对车牌位置不是摆正的进行了矩阵偏移的处理

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))  
 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)  
  
 return car\_imgs

img\_function,img\_math.py：def img\_color()还有def img\_only\_color()可以判断出图像颜色，在def img\_only\_color()中提供了不同颜色的阈值，def img\_color()负责计算还有比较那种颜色占比大。

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  
 xl, xr, yh, yl = accurate\_place(card\_img\_hsv, limit1, limit2, color)  
 if yl == yh and xl == xr:  
 continue  
 need\_accurate = False  
 if yl >= yh:  
 yl = 0  
 yh = row\_num  
 need\_accurate = True  
 if xl >= xr:  
 xl = 0  
 xr = col\_num  
 need\_accurate = True  
  
 if color == "green":  
 card\_imgs[card\_index] = card\_img  
 else:  
 card\_imgs[card\_index] = card\_img[yl:yh, xl:xr] if color != "green" or yl < (yh - yl) // 4 else card\_img[  
 yl - (  
 yh - yl) // 4:yh,  
 xl:xr]  
 if need\_accurate:  
 card\_img = card\_imgs[card\_index]  
 card\_img\_hsv = cv2.cvtColor(card\_img, cv2.COLOR\_BGR2HSV)  
 xl, xr, yh, yl = accurate\_place(card\_img\_hsv, limit1, limit2, color)  
 if yl == yh and xl == xr:  
 continue  
 if yl >= yh:  
 yl = 0  
 yh = row\_num  
 if xl >= xr:  
 xl = 0  
 xr = col\_num  
 if color == "green":  
 card\_imgs[card\_index] = card\_img  
 else:  
 card\_imgs[card\_index] = card\_img[yl:yh, xl:xr] if color != "green" or yl < (yh - yl) // 4 else card\_img[  
 yl - (  
 yh - yl) // 4:yh,  
 xl:xr]  
 return colors, card\_imgs

img\_math.py中实现车牌字符的分离，def seperate\_card()

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

img\_function中的def img\_only\_color()中包含着车牌图像返回值还有分离后车牌的图片，各自采用了math.py中的方法进行实现，具体代码如下：

def img\_only\_color(self, filename, oldimg, img\_contours):  
 *"""* ***:param*** *filename: 图像文件* ***:param*** *oldimg: 原图像文件* ***: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  
  
 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  
  
 output = cv2.cvtColor(output, cv2.COLOR\_BGR2GRAY)  
 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  
  
 card\_contours = img\_math.img\_findContours(img\_edge2)  
 card\_imgs = img\_math.img\_Transform(card\_contours, oldimg, pic\_width, pic\_hight)  
 colors, car\_imgs = img\_math.img\_color(card\_imgs)  
  
 predict\_result = []  
 predict\_str = ""  
 roi = None  
 card\_color = None  
  
 for i, color in enumerate(colors):  
  
 if color in ("blue", "yello", "green"):  
 card\_img = card\_imgs[i]  
  
 try:  
 gray\_img = cv2.cvtColor(card\_img, cv2.COLOR\_BGR2GRAY)  
 except:  
 print("gray转换失败")  
  
 # 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  
 if color == "green" or color == "yello":  
 gray\_img = cv2.bitwise\_not(gray\_img)  
 ret, gray\_img = cv2.threshold(gray\_img, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
 x\_histogram = np.sum(gray\_img, axis=1)  
  
 x\_min = np.min(x\_histogram)  
 x\_average = np.sum(x\_histogram) / x\_histogram.shape[0]  
 x\_threshold = (x\_min + x\_average) / 2  
 wave\_peaks = img\_math.find\_waves(x\_threshold, x\_histogram)  
  
 if len(wave\_peaks) == 0:  
 # print("peak less 0:")  
 continue  
 # 认为水平方向，最大的波峰为车牌区域  
 wave = max(wave\_peaks, key=lambda x: x[1] - x[0])  
  
 gray\_img = gray\_img[wave[0]:wave[1]]  
 # 查找垂直直方图波峰  
 row\_num, col\_num = gray\_img.shape[:2]  
 # 去掉车牌上下边缘1个像素，避免白边影响阈值判断  
 gray\_img = gray\_img[1:row\_num - 1]  
 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 = img\_math.find\_waves(y\_threshold, y\_histogram)  
 if len(wave\_peaks) < 6:  
 # print("peak less 1:", len(wave\_peaks))  
 continue  
  
 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  
 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  
 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]  
 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)  
  
 # 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）  
  
 part\_cards = img\_math.seperate\_card(gray\_img, wave\_peaks)  
  
 for i, part\_card in enumerate(part\_cards):  
 # 可能是固定车牌的铆钉  
  
 if np.mean(part\_card) < 255 / 5:  
 # print("a point")  
 continue  
 part\_card\_old = part\_card  
  
 w = abs(part\_card.shape[1] - SZ) // 2  
  
 part\_card = cv2.copyMakeBorder(part\_card, 0, 0, w, w, cv2.BORDER\_CONSTANT, value=[0, 0, 0])  
 part\_card = cv2.resize(part\_card, (SZ, SZ), interpolation=cv2.INTER\_AREA)  
  
 part\_card = img\_recognition.preprocess\_hog([part\_card])  
 if i == 0:  
 resp = self.modelchinese.predict(part\_card)  
 charactor = img\_recognition.provinces[int(resp[0]) - PROVINCE\_START]  
 else:  
 resp = self.model.predict(part\_card)  
 charactor = chr(resp[0])  
 # 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  
 if charactor == "1" and i == len(part\_cards) - 1:  
 if part\_card\_old.shape[0] / part\_card\_old.shape[1] >= 7: # 1太细，认为是边缘  
 continue  
 predict\_result.append(charactor)  
 predict\_str = "".join(predict\_result)  
  
 roi = card\_img  
 card\_color = color  
 break  
  
 cv2.imwrite("tmp/img\_caijian.jpg", roi)  
 return predict\_str, roi, card\_color # 识别到的字符、定位的车牌图像、车牌颜色

img\_regonization.py：实现匹配还有字符识别

GUI的设计：省略

**三、系统实施及结果**

## 3.1 系统测试结果

系统测试步骤，文字描述加截图

## （1）登陆首页



1. 点击车牌图片，导入需要识别的车牌，识别照片的结果  
   如下：
2. 定位车辆的车牌；
3. 提取字符；
4. 提取颜色；



1. 图像处理过程

打开查看图像处理过程能够看到详细的图片处理过程

## e0654d36e67fe16e1ccf853c8ccf1b7

## 3.2 算法性能

给出算法处理的效率（每次处理所需要的时间）和算法的准确率的统计。

## 3.3 github提交日志

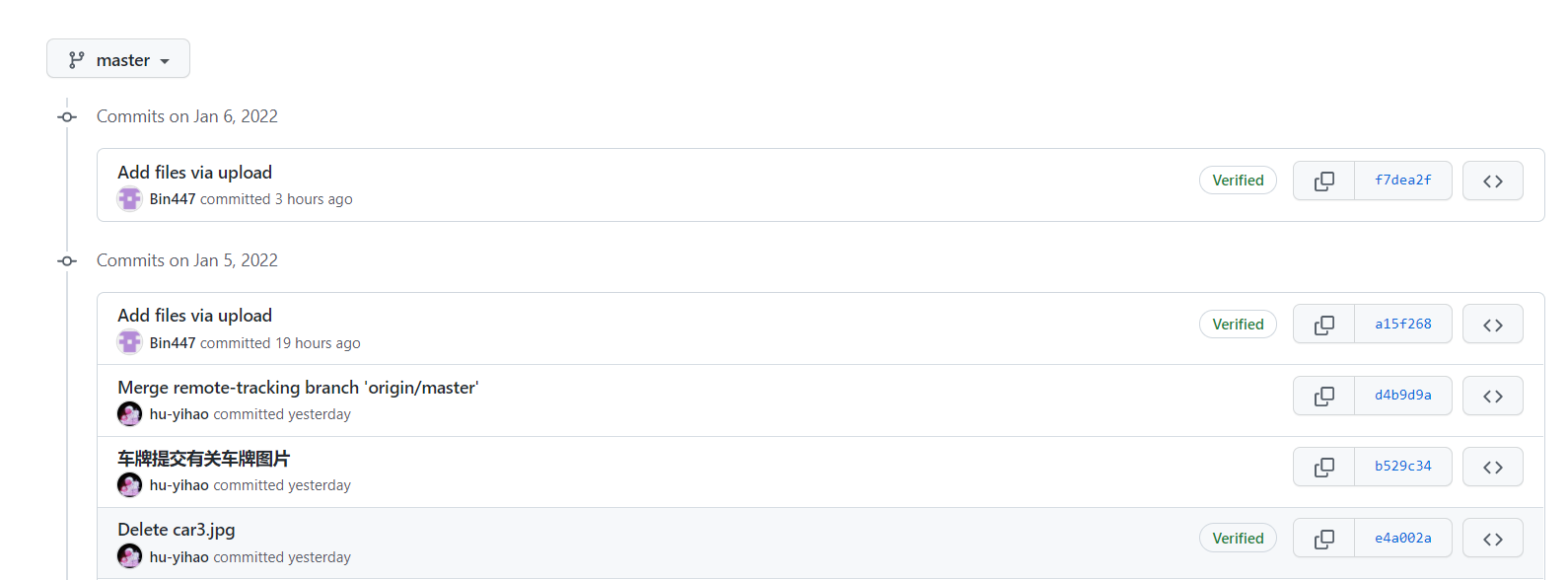
胡亿豪：







洪彬：



**四、课程设计总结**

此次课程设计，我们的系统本身还存在许多不足，距离具体实用的要求仍有很大差距，但我们却在这次课程设计中学到了很多知识。

这个课程设计让我们学到了很多东西，涉及到方方面面的知识，在这整个过程中我们查阅了大量的资料，在这期间遇到了很多困难，我知道做什么都不容易，只能静下心来，一步—个脚印的去完成才行。

这学期我们学习了数字图像处理这门课程 ，在这个课程设计应用到了很多其中的知识。理论只有应用到实际中才能学着更有意义。在整个设计中我们懂得了许多东西，也培养了我们团结合作工作的能力，树立了信心，相信会对今后的学习工作生活有非常重要的影响。同样,此次课程设计也提高了我们的动手的能力，使我们充分体会到了在创造过程中探索的艰难和成功时的喜悦。虽然这个设计的缺陷是并非对所以车牌都合适，但是在设计过程中所学到的学习方法是我们最大收获和财富，相信定会使我们受益终身。