

DIP-Project2 Report

Student Name: 何诗雅 房煊梓 许逸桐

Student ID: 210110502 210010101 210110530

Content

Part1 : License plate segmentation	- 3 -
1.Problem description	- 3 -
2.Implementation ideas	- 3 -
3.Result analysis	- 6 -
 Part2 : Split Character	 - 8 -
1.Problem description	- 8 -
2.Implementation ideas	- 8 -
3.Result analysis	- 11 -
 Part3 : Template Matching Recognized Characters	 - 13 -
1.Problem description	- 13 -
2.Implementation ideas	- 13 -
3.Result analysis	- 16 -
 Part4 : Summary	 - 18 -
1.Group division of labor	- 18 -
2.Model Analysis	- 18 -
3.What we learned from the project	- 18 -

● Part1 : License plate segmentation

1.Problem description

License plate recognition first needs to locate the license plate, extract the location of the license plate, segment the license plate from the figure, and then recognize it.

2.Implementation ideas

(1) Whole idea

Some image preprocessing is carried out to facilitate the subsequent recognition, including grayscale, Gaussian smoothing, binarization, Sobel operator edge detection, morphological operation, etc.

We need to determine whether the overall area formed by the extracted contour is a license plate through some parameters, such as whether the rectangular proportion after normalization is close to the length and width ratio of the license plate, and whether the color is blue, yellow and green.

(2) Key steps

➤ Step1: Read the image and convert it to grayscale

Load the image from the specified path; The gray scale conversion (cv2.COLOR_BGR2GRAY) is carried out to simplify image processing.

➤ Step2: Edge detection

cv2.GaussianBlur: Perform Gaussian blur processing on the image to reduce image noise and prepare for the subsequent edge detection.

cv2.Sobel: Perform the Sobel operator to calculate the first derivative of the x direction of the image to highlight the edge of the image and convert it to absolute value (cv2.convertScaleAbs), and then convert the image data back to 8 bits for easy display and processing.

➤ Step3: Image binarization

cv2.threshold(image, 0, 255, cv2.THRESH_OTSU) : Using Otsu's adaptive threshold algorithm, the image can be converted into a binary image, which is convenient for subsequent contour detection.

cv2.morphologyEx (image, cv2.MORPH_CLOSE, kernelX, iterations=1) : Carry out the closure operation using structural elements, fill the holes in the image, and connect the adjacent white areas, which helps to form a more complete license plate area.

```
# 自适应阈值处理--获得二值化图
ret, image = cv2.threshold(image, 0, 255, cv2.THRESH_OTSU)

# 闭运算, 白色部分连成整体
kernelX = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 5))
image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernelX, iterations=1)
```

➤ Step4

cv2.GetStructuringElement: Create structural elements.

cv2.dilate and cv2.erode: Be used to fill small gaps and smooth the borders of larger objects so that small white noise or objects are removed.

cv2.medianBlur: Perform median filtering to smooth the image and remove image noise without blurring the edges of the image.

```
kernelX = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 1))
kernelY = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 20))

# x方向进行闭操作 (抑制暗细节)
image = cv2.dilate(image, kernelX) # 膨胀
image = cv2.erode(image, kernelX) # 腐蚀

# y方向的开操作
image = cv2.erode(image, kernelY) # 腐蚀
image = cv2.dilate(image, kernelY) # 膨胀

# 中值滤波去除噪点
edge2 = cv2.medianBlur(image, 15)
```

➤ Step5: Obtain the outline and filter it

cv2.findContours: Get all the contours in the image.

The outline is converted to a rectangle, and the outline is screened for the aspect ratio between 2 and 6, because usually the license plate aspect ratio is in this range.

```

# 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
contours, hierarchy = cv2.findContours(edge2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# 将轮廓调整为长方形
rectangles = []
for c in contours:
    x = []
    y = []
    for point in c:
        y.append(point[0][0])
        x.append(point[0][1])
    r = [min(y), min(x), max(y), max(x)]
    rectangles.append(r)

# 排除形状不对的
car_contours = []
for r in rectangles:
    area_width = r[2]-r[0]
    area_height = r[3]-r[1]
    if area_width < area_height:
        area_width, area_height = area_height, area_width
    wh_ratio = area_width / area_height
    # print(wh_ratio)
    # 要求矩形区域长宽比在2到6之间，2到6是车牌的长宽比，其余的矩形排除
    if wh_ratio > 2 and wh_ratio < 6:
        car_contours.append(r)

```

➤ Step6: Color recognition

Color recognition is carried out on the original drawing where the selected contour area is located, and the license plate color (blue, yellow or green) is recognized through the HSV color space and color threshold. The mean (cv2.mean) is calculated for each region, and the region with the largest mean is selected as the most likely license plate region.

```

# 用颜色识别出车牌区域
dist_r = []
max_mean = 0
for r in car_contours:

    block = origin_image[r[1]:r[3], r[0]:r[2]]
    hsv = cv2.cvtColor(block, cv2.COLOR_BGR2HSV)

    for i in range(3):
        if i == 0: # 蓝色
            low = np.array([100, 55, 55])
            up = np.array([125, 255, 255])

        if i == 1: # 黄色
            low = np.array([20, 55, 55])
            up = np.array([40, 255, 255])

        if i == 2: # 绿色
            low = np.array([35, 55, 55])
            up = np.array([80, 255, 255])

    result = cv2.inRange(hsv, low, up)
    # 用计算均值的方式找出符合指定颜色的区域
    mean = cv2.mean(result)
    if mean[0] > max_mean:
        color = i
        max_mean = mean[0]
    dist_r = r

```

➤ Step7: Display and return the result

cv2.imshow: Display the segmented license plate area.

3.Result analysis

This code successfully sections the license plate.

The following figure shows the processing of origin.jpg in successive steps to achieve the segmentation of the blue license plate:



The division of the green license plate:



The division of the yellow license plate:



It can be seen from the running results that although the colors of some license plates in the picture are biased, they can still be positioned and selected, indicating that the color range of color recognition is appropriate.

For some scenarios, the license plate location segmentation fails. The parameters of each operation need to be carefully modified to set a better and more appropriate parameter method; The code for this license plate segmentation is not robust and an exception handling mechanism, such as a try-except block should be added to capture and deal with any errors that may occur during image processing.

As shown in the figure below, the bright areas are scattered and similar to the license plate, and step2 binarization is not handled properly. OTSU method may have poor effect in complex background, and it cannot distinguish the license plate area, resulting in all sticking together, resulting in error reporting and program termination.



● Part2 : Split Character

1.Problem description

This task requires further segmentation of each character in the license plate image based on the segmented license plate image given in the previous task.

2.Implementation ideas

(1) Whole idea

Due to the possibility of distortion in the obtained license plate image, it is necessary to perform correction processing.

For character judgment and segmentation, it is necessary to perform dilation operations to make each character a connected whole, and then use contours for specific judgment and segmentation.

(2) Key steps

➤ Step1 : Denoise the image

denoise the original image to facilitate subsequent processing.

```
# 图像去噪  
image_denoise = cv2.GaussianBlur(image, (3, 3), 0)
```

➤ Step2 : Correct the image

Due to the possibility that the license plate segmented from the previous task may be skewed, it needs to be corrected.

```
# 将可能歪斜的车牌校正  
image_corrected = correct_plate_orientation(image_denoise)
```

Firstly, obtain the grayscale image and use Canny edge detection to obtain the

edges of the image. Then perform a Hough transform on these edges to obtain a set of line segments.

If the set of line segments is not empty, calculate the corresponding angle, rotate the image, obtain the rotated image, which is the corrected image, and then return it. If the set of line segments is empty, there is no need for correction and the original image can be returned directly.

```
# 将可能歪斜的车牌校正
def correct_plate_orientation(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150)
    lines = cv2.HoughLines(edges, 1, np.pi/180, 100)

    if lines is not None:
        #print(lines[0])
        for rho, theta in lines[0]:
            a = np.cos(theta)
            b = np.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 1000*(-1)*b), int(y0 + 1000*a))
            pt2 = (int(x0 - 1000*(-1)*b), int(y0 - 1000*a))
            angle = np.arctan2(pt2[1] - pt1[1], pt2[0] - pt1[0]) * 180.0 / np.pi

            rotated_image = image.copy()
            (h, w) = image.shape[:2]
            center = (w // 2, h // 2)
            M = cv2.getRotationMatrix2D(center, angle, 1.0)
            rotated_image = cv2.warpAffine(rotated_image, M, (w, h), flags=cv2.INTER_CUBIC, borderMode=cv2.BORDER_REPLICATE)

        return rotated_image
    return image
```

➤ Step3 : Obtain the processed binary image

Obtain the corresponding grayscale image after correction and convert it into a binary image.

It should be noted that due to the possible differences in the colors of various license plates, such as blue, green, and yellow, the corresponding character colors are white, black, and black. In order to unify the binary image with a black background and a white foreground, it is necessary to process the binary image based on the color value.

```
# 获得灰度图
image_gray = cv2.cvtColor(image_corrected, cv2.COLOR_RGB2GRAY)
# 获得二值化图
ret, image_bi = cv2.threshold(image_gray, 0, 255, cv2.THRESH_OTSU)
# print(color)
if(color!=0):
    image_bi = 255 - image_bi
```

➤ Step4 : Dilation

Prepare to retrieve the characters. Due to the fact that the first character of Chinese license plate is a Chinese character that may not be connected, while other characters are letters or numbers that are connected, it is necessary to perform dilation to make each character a separate whole, facilitating subsequent character segmentation. Here, rectangular convolution kernels are used for operation.

```
# 膨胀操作, 采用矩形卷积核
kernel_dilate = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
image_dilate = cv2.dilate(image_bi, kernel_dilate)
```

➤ Step5

Retrieve the contours, find the smallest rectangular box corresponding to each contour, and arrange them in order.

```
# 获取轮廓
contours, hierarchy = cv2.findContours(image_dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 根据轮廓找到对应的最小的矩形框, 并且进行排序
chars = list()
for contour in contours:
    rect = cv2.boundingRect(contour)
    chars.append(rect)
chars.sort()
```

➤ Step6 : Recognize characters

For each rectangular box, determine whether it is a character based on its height and width. If so, add it to the image_char list. Images_chars is a list of segmented characters that can be used for the next task of template matching characters to identify license plate numbers.

```
images_chars = list()
for rect in chars:
    # rect的[0],[1],[2],[3]分别是矩阵左上点的横坐标, 纵坐标, 矩阵的宽和矩阵的高
    x,y,w,h = rect[0],rect[1],rect[2],rect[3]
    # 根据w和h判断是否为一个字符, 若是, 则将对应的矩阵加入列表
    if (w*1.5<h<w*3.5)and(w>10):
        images_chars.append(image_dilate[y:y+h,x:x+w])
```

➤ Step7 : Display and return the result

Finally, display the character segmentation result and return images_chars.

3.Result analysis

The overall operation is relatively simple, and character segmentation can be successfully performed on the front license plate image.

The following images show some results of successful character segmentation.

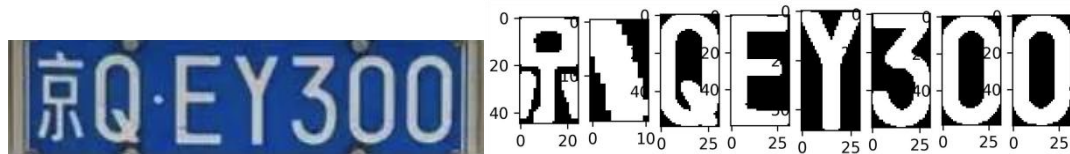


After correcting the skewed license plates, the characters were segmented to make them easier to recognize and match. The following images are the result images of character segmentation obtained after successfully correcting some skewed license plates.

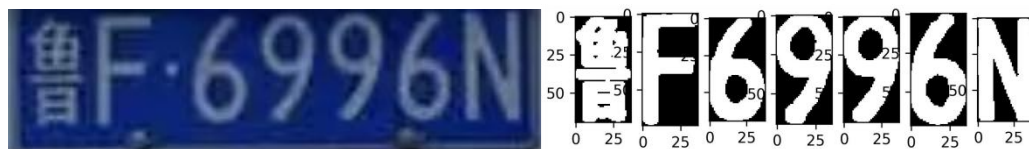


It can be seen that the originally slightly skewed characters have been corrected .

However, the dilation may not ensure that all characters are formed as a whole, leading to recognition errors. For example, scattered glyphs such as "云" and "京" may not be recognized as a single character due to not expanding into a whole. The following image is an example of misidentifying "京".



Also, for similar glyphs, the difference after dilation is not significant, which may lead to recognition errors. For example, the characters "鲁" and "贵" are incorrectly recognized due to their dilation shapes being relatively similar. The following figure is an example of incorrectly identifying "鲁" as "贵".



['贵', 'F', '6', '9', '9', '6', 'N']
贵F6996N

● Part3 : Template Matching Recognized Characters

1.Problem description

In order to achieve license plate recognition, after extracting the location of the license plate from the graph and segmenting the license plate characters, the last step that needs to be done is to use template matching to recognize the characters.

2.Implementation ideas

(1) Whole idea

First, prepare templates for all the characters that can appear on the license plate. Divide the templates into those that contain only Chinese characters (for the first character of the license plate), those that contain only letters (for the second character of the license plate) and those that contain both letters and numbers (for the last five/six characters of the license plate). Finally, each segmented character image is matched for recognition and the results are printed out.

(2) Key functions

➤ get_chinese_words_list()

Read all the pictures of Chinese characters from the template file, return a list whose elements are a list of all the pictures of a character.

```
# 获得中文模板列表（只匹配车牌的第一个字符）
def get_chinese_words_list():
    chinese_words_list = []
    for i in range(34,64):
        #将模板存放在字典中
        c_word = read_directory('D:/HITSZer_juanlife/CS/2024_Spring/DIP/Project2/car/refer1/'+ template[i])
        chinese_words_list.append(c_word)
    return chinese_words_list
```

➤ get_eng_words_list()

Reads all pictures of English characters from a template file and returns a list whose elements are a list of all pictures of a character.

```
# 获得英文模板列表（只匹配车牌的第二个字符）
def get_eng_words_list():
    eng_words_list = []
    for i in range(10,34):
        e_word = read_directory('D:/HITSZer_juanlife/CS/2024_Spring/DIP/Project2/car/refer1/'+ template[i])
        eng_words_list.append(e_word)
    return eng_words_list
```


➤ **get_eng_num_words_list()**

Read all the pictures of English characters and pictures of numeric characters from the template file, and return a list whose elements are a list of all the pictures of a character.

```
# 获得英文和数字模板列表（匹配车牌后面的字符）
def get_eng_num_words_list():
    eng_num_words_list = []
    for i in range(0,34):
        word = read_directory('D:/HITSZer_juanlife/CS/2024_Spring/DIP/Project2/car/refer1/'+ template[i])
        eng_num_words_list.append(word)
    return eng_num_words_list
```

➤ **template_score(template,image)**

template is a template image and image is the character image to be recognized. The template is first converted to a format and then thresholded to obtain a black and white image. Then the template is resized to the same size as the image, and finally the cv2 library is used to match the template and return the matching score.

```
# 读取一个模板地址与图片进行匹配，返回得分
def template_score(template,image):
    #将模板进行格式转换
    template_img=cv2.imdecode(np.fromfile(template,dtype=np.uint8),1)
    template_img = cv2.cvtColor(template_img, cv2.COLOR_RGB2GRAY)
    #模板图像阈值化处理——获得黑白图
    ret, template_img = cv2.threshold(template_img, 0, 255, cv2.THRESH_OTSU)
    # height, width = template_img.shape
    # image_ = image.copy()
    # image_ = cv2.resize(image_, (width, height))
    image_ = image.copy()
    #获得待检测图片的尺寸
    height, width = image_.shape
    # 将模板resize至与图像一样大小
    template_img = cv2.resize(template_img, (width, height))
    # 模板匹配，返回匹配得分
    result = cv2.matchTemplate(image_, template_img, cv2.TM_CCOEFF)
    return result[0][0]
```

➤ **template_matching(images_chars,chinese_words_list,eng_words_list,eng_num_words_list)**

images_chars is the list of character images to be recognized and xxx_words_list is the list of template images obtained through the get_xxx_words_list function.

For each character image to be recognized, according to its index, select the template image list to be used for comparison. Match the character images with each set of templates in the template list one by one, and call the template_score() function to calculate the matching score. For each set of templates, select the one with the highest match score as the match score for this character. For all possible characters, select the one with the highest match score as the character recognized from the character image. And add this character to the result set. After all the character images are recognized, return the result set.

```
# 对分割得到的字符逐一匹配
def template_matching(images_chars, chinese_words_list, eng_words_list, eng_num_words_list):
    results = []
    for index,word_image in enumerate(images_chars):
        if index==0:
            best_score = []
            for chinese_words in chinese_words_list:
                score = []
                for chinese_word in chinese_words:
                    result = template_score(chinese_word,word_image)
                    score.append(result)
                best_score.append(max(score))
            i = best_score.index(max(best_score))
            # print(template[34+i])
            r = template[34+i]
            results.append(r)
            continue
        if index==1:
            best_score = []
            for eng_word_list in eng_words_list:
                score = []
                for eng_word in eng_word_list:
                    result = template_score(eng_word,word_image)
                    score.append(result)
                best_score.append(max(score))
            i = best_score.index(max(best_score))
            # print(template[10+i])
            r = template[10+i]
            results.append(r)
            continue
        else:
            best_score = []
            for eng_num_word_list in eng_num_words_list:
                score = []
                for eng_num_word in eng_num_word_list:
                    result = template_score(eng_num_word,word_image)
                    score.append(result)
                best_score.append(max(score))
            i = best_score.index(max(best_score))
            # print(template[i])
            r = template[i]
            results.append(r)
            continue
    return results
```

➤ **match(images_chars):**

This is the main function in the section. It calls function `get_chinese_words_list()`, function `get_eng_words_list()` and function `get_eng_num_words_list()` to get different template lists for specific indexes. Then it calls `template_matching(images_chars_,chinese_words_list,eng_words_list,eng_num_words_list)` to fetch result list. At last, it prints the result.

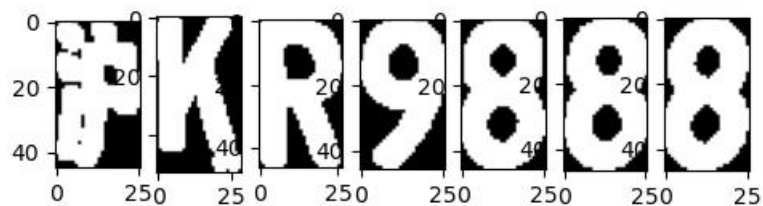
```
def match(images_chars):
    chinese_words_list = get_chinese_words_list()
    eng_words_list = get_eng_words_list()
    eng_num_words_list = get_eng_num_words_list()

    images_chars_ = images_chars.copy()
    # 调用函数获得结果
    result = template_matching(images_chars_, chinese_words_list, eng_words_list, eng_num_words_list)
    print(result)
    # "".join(result)函数将列表转换为拼接好的字符串，方便结果显示
    print("".join(result))
```

3.Result analysis

Overall result: the algorithm can basically correctly recognize characters from segmented character images.

Give two examles:



segmented character images

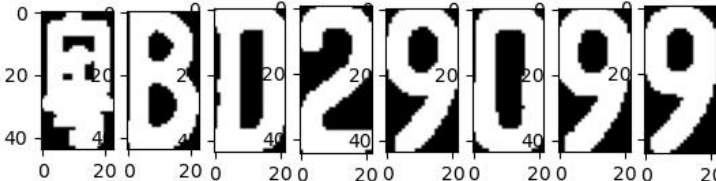
```
['沪', 'K', 'R', '9', '8', '8', '8']
```

```
沪KR9888
```

```
|
```

```
Process finished with exit code 0
```

result



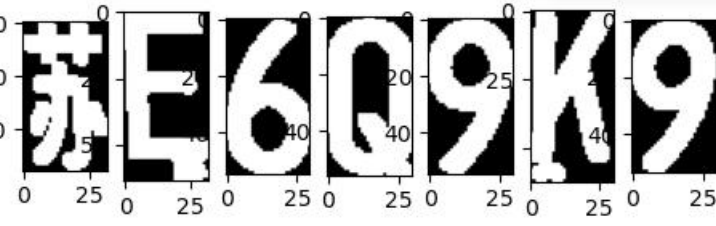
segmented character images

```
['B', 'D', '2', '9', '0', '9', '9']
BD29099
```

Process finished with exit code 0

result

However, when encountering characters with more similar shapes, such as '0' and 'Q', this algorithm can also confuse similar characters, resulting in failure to recognize the correct result. For example, in the example below, 'Su E6Q9K9' is recognized as 'Su E609K9'.



segmented character images

```
['苏', 'E', '6', '0', '9', 'K', '9']
苏E609K9
```

Process finished with exit code 0

result

● Part4 : Summary

1.Group division of labor

Group leader : 房煊梓

Part1 : 何诗雅

Part2 : 房煊梓

Part3 : 许逸桐

2.Model Analysis

Our model first segments blue, yellow, and green license plates. Then it corrects license plates that may be skewed and uses dilation and contour to separate each character. Finally, the final recognition result is obtained through template matching of characters.

Our model has basically achieved correct recognition of license plate numbers. Our model can still locate and select certain license plates in the image even if there is color deviation. It can correct skewed license plates and accurately recognize characters from segmented character images. However, our model may not perform well in complex backgrounds, with character segmentation errors occurring for more dispersed glyphs or closer glyphs.

3.What we learned from the project

In the process of completing this project, we not only learned to apply the knowledge learned in the class to practical programming and thus gaining a deeper understanding of knowledge and stronger coding ability, but also learned how to divide tasks into groups and communicate and cooperate with team members. In addition, we improved our English writing skills during the process of writing the report.