案例三：基于颜色阈值分割和边缘检测的车牌定位

1. 案例目的

• 掌握车牌定位的基本流程；

• 熟练应用图像分割技术。

2. 案例内容

本案例主要采用颜色阈值分割和边缘检测进行车牌定位。车牌定位在工程实际项目上一般分为粗定位和精定位两部分。先通过颜色信息进行车牌感兴趣区域的粗略定位，然后基于边缘信息，根据车牌上下和左右明显的边框进行精确定位。车牌定位的精度直接影响后续车牌字符分割的效果。

3. 案例知识点

• 彩色图像灰度化；

• 彩色空间转换：RGB转HSV；

• 二值化；

• 边缘检测；

• 直方图

4. 案例时长

共4学时，具体安排如下：

• 获取车牌感兴趣区域（1学时）

• 车牌感兴趣区域粗定位（1学时）

• 车牌感兴趣区域精定位（1.5学时）

• 在原图画出车牌定位结果（0.5学时）

5. 案例实验环境

**•软件环境：**

1）Python 3

2）opencv 3.x

**•开发环境与工具：**

1）Jupyter Notebook

6. 案例分析

本案例主要分为以下4个步骤：

1）获取车牌感兴趣区域；

• 先对图像进行预处理，主要包括高斯平滑滤波，彩色图像转灰度图，RGB图像转HSV图像

• 根据蓝色车牌颜色进行阈值化（本案例仅以蓝色车牌为例，不考虑其他颜色车牌）

2）车牌感兴趣区域粗定位；

3）车牌感兴趣区域精定位；

• 对第一部分获取的粗略位置区域进行适当扩充

• 在扩充区域内，计算边缘图像

• 根据边缘图像，利用车牌边框边缘长的信息，通过边缘像素直方图确定车牌精确的上下和左右边界

4）在原图画出车牌定位结果。

车牌定位的流程图如图1所示。

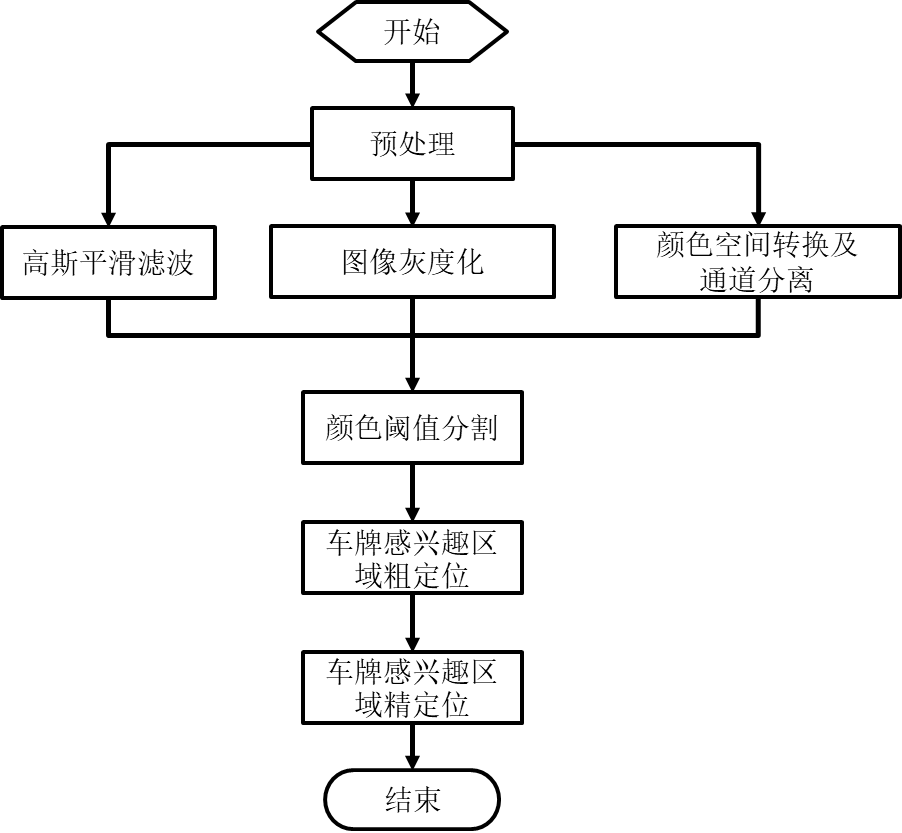


图1 车牌定位流程图

7. 案例实验过程

7.1开发准备

7.1.1 获取数据

本案例需要1个数据集，是car.jpeg。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/2f9a51190eaa42ecba2056fc09f6535b.zip下载数据集。将下载后的数据集解压得到名为DIP实验数据集的文件夹。文件夹下存储着本案例用到的数据car.jpeg。

7.1.2 新建实验目录

在桌面上新建一个目录，名字叫SY03。在该目录下新建input目录用于存放待处理的数据。将图像car.jpeg复制到input目录下。

打开Jupyter Notebook，此时input目录的样式如图2所示。

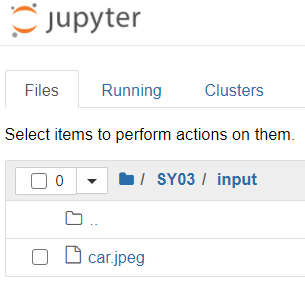


图2 input目录

7.1.3 新建代码文件

使用Jupyter Notebook新建plate\_locate.ipynb文件。

当前文件目录结构如图3所示。

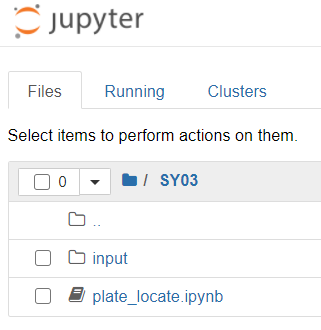


图3 案例目录结构

7.2获取车牌感兴趣区域

|  |
| --- |
|  |
| （a）车辆图片 |
|  |
| （b）根据颜色信息筛选出车牌感兴趣区域 |

图4 车牌感兴趣区域粗定位效果图

蓝色车牌最关键的信息是它的颜色和边缘。首先，根据颜色信息能够粗略的确定其位置。

效果图如图4所示。

打开plate\_locate.ipynb文件。按如下四个步骤逐步实现：

7.2.1导入相关模块

主要用到cv2，numpy和matplotlib.pyplot三个库。cv2主要用于读取图像，numpy用于构建矩阵，matplotlib.pyplot主要用于图像显示。

【代码7-2-1】plate\_locate.ipynb

# -\*- coding: utf-8 -\*-

'''

车牌定位

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

执行该代码块。

7.2.2车辆图像预处理

车辆图像预处理主要对原图像进行高斯模糊，减少噪声干扰。对RGB三个通道进行分离；获取彩色图像的灰度图；将RGB图像转成HSV图像。这些是下一步进行颜色分析的准备工作。

首先读入input目录下的图片car.jpeg，调用opencv的GaussianBlur接口函数对车辆图片进行高斯模糊，高斯模糊用到的模板大小为5\*5。

【代码7-2-2-1】plate\_locate.ipynb

path = 'input/car.jpeg'

img = cv2.imread(path)

img\_GaussBlur = cv2.GaussianBlur(img,(5,5),0)# 高斯模糊

执行该代码块。

然后，调用opencv的split函数，将模糊后的图像img\_GaussBlur的BGR三个通道分离，每个通道作为一幅图。

【代码7-2-2-2】plate\_locate.ipynb

# RGB通道分离

img\_B = cv2.split(img\_GaussBlur)[0]

img\_G = cv2.split(img\_GaussBlur)[1]

img\_R = cv2.split(img\_GaussBlur)[2]

执行该代码块。

调用opencv的cvtColor函数，将彩色图img\_GaussBlur转化为灰度图img\_gray和HSV彩色空间图img\_HSV。

【代码7-2-2-3】plate\_locate.ipynb

# 读取灰度图和HSV空间图

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

img\_HSV = cv2.cvtColor(img\_GaussBlur, cv2.COLOR\_BGR2HSV)

执行该代码块。

7.2.3获取车牌感兴趣区域二值图

根据案例一介绍的方法，打印车牌图像RGB和HSV各个通道的像素值，如图5所示。从中能够找到一些规律。通过观察发现：车牌蓝色区域的R通道像素值普遍在40以下，B通道在70以上，HSV通道的第一个通道值在100~115范围内。

|  |
| --- |
|  |
| （a）车牌图像B通道部分区域像素值截图 |
|  |
| （b）车牌图像R通道部分区域像素值截图 |
|  |
| （c）车牌图像HSV第一个通道部分区域像素值截图 |
| 图5 车牌区域图像各个通道像素值 |

根据以上的规律，可以用来进行基于颜色的阈值分割，从而得到车牌感兴趣区域。编写如下代码。

首先，定义获取车牌感兴趣区域函数getPlateROI，输入参数为灰度图img\_gray，HSV彩色图img\_HSV，B通道图img\_B，R通道图img\_R。

【代码7-2-3-1】plate\_locate.ipynb

def getPlateROI(img\_gray, img\_HSV, img\_B, img\_R):

获取图像的宽高信息，构造2层for循环遍历图像。设置阈值化的条件：HSV图像中H通道像素值在100~115范围内，且B通道像素值大于70，R通道像素值小于40，所有条件均满足时，将该位置处的像素值设为255，反之设为0。

【代码7-2-3-2】plate\_locate.ipynb

height = img\_gray.shape[0] # 获取图像的高

width = img\_gray.shape[1] # 获取图像的宽

for row in range(height):

for col in range(width):

# 普通蓝色车牌，同时排除透明反光物质的干扰

if (img\_HSV[:, :, 0][row, col] > 100) and \

(img\_HSV[:, :, 0][row, col]<115) and \

(img\_B[row, col] > 70) and \

(img\_R[row, col] < 40):

img\_gray[row, col] = 255

else:

img\_gray[row, col] = 0

得到初步的二值图img\_gray后，对其进行两次高斯平滑和固定阈值化，这样做的目的是消除部分离散的孤立的点，对车牌区域内部的“孔状”区域进行一定的填充，有助于下一步感兴趣区域的定位。最后，将最终的二值图img\_sec\_binary返回。

【代码7-2-3-3】plate\_locate.ipynb

img\_gray\_blur = cv2.GaussianBlur(img\_gray, (9, 9), 0) # 高斯平滑

# 二值化

\_, img\_binary = cv2.threshold(img\_gray\_blur, 100, 255, cv2.THRESH\_BINARY)

img\_sec\_blur = cv2.GaussianBlur(img\_binary, (9, 9), 0)# 高斯平滑

# 二值化

\_, img\_sec\_binary = cv2.threshold(img\_sec\_blur, 100, 255, cv2.THRESH\_BINARY)

return img\_sec\_binary

完整的代码参见代码7-2-3。

【代码7-2-3】plate\_locate.ipynb

# 获取车牌感兴趣区域二值图

def getPlateROI(img\_gray, img\_HSV, img\_B, img\_R):

height = img\_gray.shape[0] # 获取图像的高

width = img\_gray.shape[1] # 获取图像的宽

for row in range(height):

for col in range(width):

# 普通蓝色车牌，同时排除透明反光物质的干扰

if (img\_HSV[:, :, 0][row, col] > 100) and \

(img\_HSV[:, :, 0][row, col]<115) and \

(img\_B[row, col] > 70) and \

(img\_R[row, col] < 40):

img\_gray[row, col] = 255

else:

img\_gray[row, col] = 0

img\_gray\_blur = cv2.GaussianBlur(img\_gray, (9, 9), 0) # 高斯平滑

# 二值化

\_, img\_binary = cv2.threshold(img\_gray\_blur, 100, 255, cv2.THRESH\_BINARY)

img\_sec\_blur = cv2.GaussianBlur(img\_binary, (9, 9), 0)# 高斯平滑

# 二值化

\_, img\_sec\_binary = cv2.threshold(img\_sec\_blur, 100, 255, cv2.THRESH\_BINARY)

return img\_sec\_binary

执行该代码块。

7.2.4显示车牌感兴趣区域

调用getPlateROI函数，并显示结果。

【代码7-2-4】plate\_locate.ipynb

# 显示车牌感兴趣区域

img\_binary = getPlateROI(img\_gray, img\_HSV, img\_B, img\_R)

# 显示车牌感兴趣区域二值图

plt.imshow(img\_binary,cmap ='gray')

plt.show()

得到的车牌感兴趣区域如图6所示。



图6 车牌感兴趣区域

7.3车牌感兴趣区域粗定位

得到图6所示的车牌感兴趣区域后，接下来，需要计算出车牌感兴趣区域的坐标位置。这里，我们直接利用图像水平方向和垂直方向投影直方图进行区域定位。原理比较简单，如图7所示。

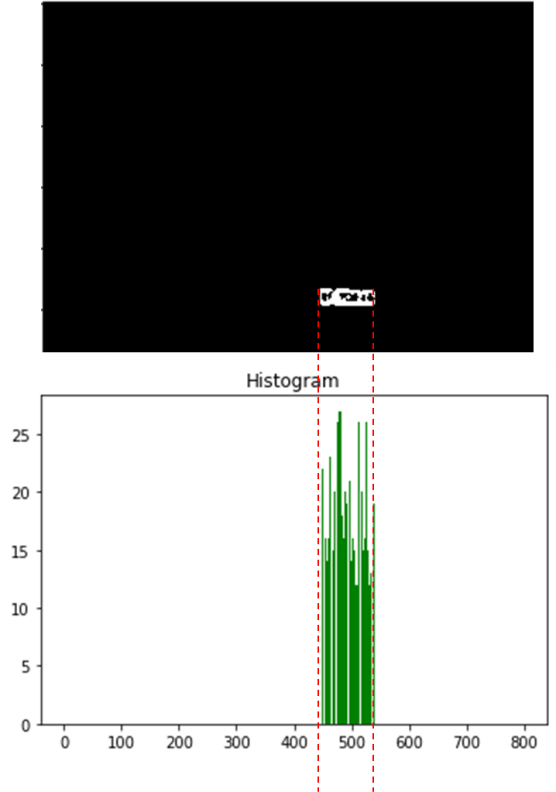


图7 二值图像垂直方向投影直方图

对二值图像按列统计像素值为255的像素个数，得到垂直方向的投影直方图。直方图的起始和结束对应着车牌感兴趣区域的左右边界位置坐标。同理，对二值图做水平方向的投影直方图，可以得到车牌感兴趣区域的上下边界位置坐标。

首先定义函数getRoughPosition，二值图像img\_binary作为函数输入。

【代码7-3-1】plate\_locate.ipynb

def getRoughPosition(img\_binary):

将二值图img\_binary转换为0，1二值图binary\_0\_1，目的是为了便于后面统计计数。

【代码7-3-2】plate\_locate.ipynb

binary\_0\_1 = (img\_binary > 0) \* 1.0

按列统计二值图binary\_0\_1每一列非零像素个数，得到统计数组hist\_col。

【代码7-3-3】plate\_locate.ipynb

hist\_col = binary\_0\_1.sum(axis=0)

按行统计二值图binary\_0\_1每一行非零像素个数，得到统计数组hist\_row。

【代码7-3-4】plate\_locate.ipynb

hist\_row = binary\_0\_1.sum(axis=1)

根据hist\_row找到值在60到120之间的元素下标索引,记录在row\_range中。60和120的选取是依据经验确定。

【代码7-3-5】plate\_locate.ipynb

row\_range = np.where((hist\_row>60)&(hist\_row<120))

根据row\_range确定车牌的上边界。因为row\_range中保存了二值图binary\_0\_1中满足条件的行下标，按顺序搜索row\_range中的值。车牌上边界应该满足如下规律：

找到第一个水平投影值在60到120之间的行，且临近的行也符合该条件，防止“毛刺噪声”直方图带来的干扰。

反映在row\_range中的值就是当前行和后面几行的下标值比较接近，也即如果当前行是车牌边界，那么车牌边界的后的连续几行（比如3行）应当也在row\_rang中。举例说明，比如row\_range=[21,22,35,36,37,38,39,40,41,42,43,…,65,66,67,68]，表明原图像中第21，22，35，36，37，…,65,66,67,68行非零元素统计值满足要求。车牌上边界应该是第几行？按照规律，应当是第35行。第21行和它后面的几个行下标值相差太多，说明21行和22行是噪声数据。用同样的方式确定车牌下边界。确定下边界需要从row\_range的尾部逆向搜索。

逻辑代码如下。

【代码7-3-6】plate\_locate.ipynb

if row\_range:#row\_range 不为空

if len(row\_range[0])> 5:

for y\_index in range(len(row\_range[0])-1):

cur\_value = row\_range[0][y\_index]

next\_value = row\_range[0][y\_index+1]

next2\_value = row\_range[0][y\_index+2]

next3\_value = row\_range[0][y\_index+3]

if next\_value-cur\_value<3 and \

next2\_value-cur\_value<4 and \

next3\_value-cur\_value<5:

y1 = cur\_value #确定车牌上边界

break

for y\_index in range(len(row\_range[0])-1,-1,-1):

cur\_value = row\_range[0][y\_index]

last\_value = row\_range[0][y\_index-1]

last2\_value = row\_range[0][y\_index-2]

last3\_value = row\_range[0][y\_index-3]

if last\_value-cur\_value<3 and \

last2\_value-cur\_value <4 and \

last3\_value-cur\_value<5:

y2 = cur\_value #确定车牌下边界

break

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

车牌左右边界遵循同样的处理方法。根据hist\_col找到值在10到60之间的元素下标索引,记录在col\_range中。根据col\_range可得到车牌左右边界对应的列下标。

【代码7-3-7】plate\_locate.ipynb

col\_range = np.where((hist\_col>10)&(hist\_col<60))

if col\_range:

if len(col\_range[0])>10:

for x\_index in range(len(col\_range[0])-1):

cur\_value = col\_range[0][x\_index]

next\_value = col\_range[0][x\_index+1]

next2\_value = col\_range[0][x\_index+2]

next3\_value = col\_range[0][x\_index+3]

if next\_value-cur\_value<3 and \

next2\_value-cur\_value <4 and \

next3\_value-cur\_value<5:

x1 = cur\_value #确定车牌左边界

break

for x\_index in range(len(col\_range[0])-1,-1,-1):

cur\_value = col\_range[0][x\_index]

last\_value = col\_range[0][x\_index-1]

last2\_value = col\_range[0][x\_index-2]

last3\_value = col\_range[0][x\_index-3]

if last\_value-cur\_value<3 and \

last2\_value-cur\_value <4 and \

last3\_value-cur\_value<5:

x2 = cur\_value #确定车牌右边界

break

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

return x1,y1,x2,y2

调用粗定位的代码，获得车牌感兴趣区域粗略坐标位置，并保存车牌感兴趣区域粗略子图。

【代码7-3-8】

# 执行车牌粗定位

x1,y1,x2,y2 = getRoughPosition(img\_binary)

# 显示车牌感兴趣粗略子图

rough\_plateImg = img[y1:y2, x1:x2]

# opencv中的通道顺序是BGR，与Python的RGB刚好相反。因此此处需要调整顺序。

rough\_plateImg = rough\_plateImg[:,:,(2,1,0)]

plt.imshow(rough\_plateImg)

plt.show()

完整的代码参见代码7-3。

【代码7-3】plate\_locate.ipynb

# 根据车牌感兴趣区域，得到车牌粗略定位

def getRoughPosition(img\_binary):

binary\_0\_1 = (img\_binary > 0) \* 1.0

hist\_col = binary\_0\_1.sum(axis=0) #按列统计非零像素个数

hist\_row = binary\_0\_1.sum(axis=1) #按行统计非零像素个数

row\_range = np.where((hist\_row>60)&(hist\_row<120))

if row\_range:#row\_range 不为空

if len(row\_range[0])> 5:

for y\_index in range(len(row\_range[0])-1):

cur\_value = row\_range[0][y\_index]

next\_value = row\_range[0][y\_index+1]

next2\_value = row\_range[0][y\_index+2]

next3\_value = row\_range[0][y\_index+3]

if next\_value-cur\_value<3 and \

next2\_value-cur\_value<4 and \

next3\_value-cur\_value<5:

y1 = cur\_value #确定车牌上边界

break

for y\_index in range(len(row\_range[0])-1,-1,-1):

cur\_value = row\_range[0][y\_index]

last\_value = row\_range[0][y\_index-1]

last2\_value = row\_range[0][y\_index-2]

last3\_value = row\_range[0][y\_index-3]

if last\_value-cur\_value<3 and \

last2\_value-cur\_value <4 and \

last3\_value-cur\_value<5:

y2 = cur\_value #确定车牌下边界

break

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

col\_range = np.where((hist\_col>10)&(hist\_col<60))

if col\_range:

if len(col\_range[0])>10:

for x\_index in range(len(col\_range[0])-1):

cur\_value = col\_range[0][x\_index]

next\_value = col\_range[0][x\_index+1]

next2\_value = col\_range[0][x\_index+2]

next3\_value = col\_range[0][x\_index+3]

if next\_value-cur\_value<3 and \

next2\_value-cur\_value <4 and \

next3\_value-cur\_value<5:

x1 = cur\_value #确定车牌左边界

break

for x\_index in range(len(col\_range[0])-1,-1,-1):

cur\_value = col\_range[0][x\_index]

last\_value = col\_range[0][x\_index-1]

last2\_value = col\_range[0][x\_index-2]

last3\_value = col\_range[0][x\_index-3]

if last\_value-cur\_value<3 and \

last2\_value-cur\_value <4 and \

last3\_value-cur\_value<5:

x2 = cur\_value #确定车牌右边界

break

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

return x1,y1,x2,y2

# 执行车牌粗定位

x1,y1,x2,y2 = getRoughPosition(img\_binary)

# 显示车牌感兴趣粗略子图

rough\_plateImg = img[y1:y2, x1:x2]

# opencv中的通道顺序是BGR，与Python的RGB刚好相反。因此此处需要调整顺序。

rough\_plateImg = rough\_plateImg[:,:,(2,1,0)]

plt.imshow(rough\_plateImg)

plt.show()

执行该代码块。

得到粗定位的车牌感兴趣区域子图如图8所示。



图8 粗定位车牌子图

7.4车牌感兴趣区域精定位

根据颜色信息已经可以定位出车牌的粗略位置，为了保证后续车牌图像二值化，车牌号码分割和识别等操作的效果，还需要对车牌进行精准定位。精定位的理想效果是对粗定位的区域进行裁剪，切割掉不是车牌区域的部分。

基本思路是根据粗定位，获取车牌粗略区域，提取该区域内的图像。在此基础上，利用图像边缘信息精调边界。

首先，定义函数getPrecisePosition，输入参数是原图像srcImg和车牌粗略区域坐标x1,y1,x2,y2。

【代码7-4-1】plate\_locate.ipynb

def getPrecisePosition(srcImg, x1,y1,x2,y2):

获取原图像的宽高信息。

【代码7-4-2】plate\_locate.ipynb

height = srcImg.shape[0]

width = srcImg.shape[1]

先对车牌粗略区域进行扩充，这里对车牌区域上下左右各扩充了5个像素，防止颜色信息不造成车牌感兴趣区域不完整。得到四周有一定扩展的车牌区域图像plateImg。

【代码7-4-3】plate\_locate.ipynb

#对车牌区域进行适当扩展，避免颜色偏色带来的定位误差

x1 = x1-5

x1 = max(0,x1)

x2 = x2+5

x2 = min(x2,width)

y1 = y1-5

y1 = max(y1,0)

y2 = y2+5

y2 = min(y2, height)

plateImg = srcImg[y1:y2, x1:x2]

对扩展后的车牌图像plateImg进行灰度化得到灰度图像plateImg\_gray。这里调用了opencv的cvtColor函数。CvtColor函数前几章案例已经介绍过，这里不再赘述。

【代码7-4-4】plate\_locate.ipynb

plateImg\_gray = cv2.cvtColor(plateImg,cv2.COLOR\_RGB2GRAY)

然后提取车牌图像的边缘图Canny\_img。这里调用了opencv的canny边缘检测算子。Canny()函数的调用形式为cv2.Canny(grayImg,thresh1,thresh2)。grayImg是输入的图片，要求是灰度图。thresh1和thresh2是阈值，用于删选边缘信息。

【代码7-4-5】plate\_locate.ipynb

Canny\_img = cv2.Canny(plateImg\_gray, 85, 150)

对边缘图像Canny\_img进行阈值化，得到边缘二值图plate\_edge，该二值图只有0和255两个值。

【代码7-4-6】plate\_locate.ipynb

ret, plate\_edge = cv2.threshold(Canny\_img, 127, 255, cv2.THRESH\_BINARY)

为了便于后续统计计数，将plate\_edge调整为0-1二值图binary\_0\_1。对binary\_0\_1进行处理，分别按行和按列统计像素值为1的个数，即得到图像的水平投影直方图hist\_row和垂直方向直方图his\_col。

【代码7-4-7】plate\_locate.ipynb

binary\_0\_1 = (plate\_edge > 0) \* 1.0

hist\_row = binary\_0\_1.sum(axis=1) #按行统计非零像素个数

hist\_col = binary\_0\_1.sum(axis=0) #按列统计非零像素个数

根据水平方向投影直方图hist\_row筛选出大于15的行下标，放置在数组row\_index中。

【代码7-4-8】plate\_locate.ipynb

row\_index = np.where(hist\_row > 15) #统计像素数总和超过阈值的行下标

将row\_index的第一个元素和最后一个元素的值作为车牌的上下边界。

【代码7-4-9】plate\_locate.ipynb

if row\_index: # row\_range 不为空

if(len(row\_index[0])>1):

dy1= row\_index[0][0]

y1 = y1 + dy1 + 1 #车牌上边界精调

dy2 = plate\_edge.shape[0] - row\_index[0][-1] #车牌下边界

y2 = y2 - dy2 - 1

else: #缺失一个边界，不再进行精调

y1 = y1

y2 = y2

else: #row\_range 为空

y1 = y1

y2 = y2

采用同样的处理方法，确定车牌的左右边界。

【代码7-4-10】plate\_locate.ipynb

col\_index = np.where(hist\_col > 15) #统计像素数总和超过阈值的列下标

if col\_index:

if(len(col\_index[0])>1):

dx1 = col\_index[0][0]

x1 = x1 + dx1 + 1 ##车牌左边界精调

dx2 = plate\_edge.shape[1] - col\_index[0][-1]

x2 = x2 - dx2 - 1 #车牌右边界精调

else:

x1 = x1

x2 = x2

else:

x1 = x1

x2 = x2

return x1,y1,x2,y2

调用getPrecisePosition函数，获得车牌精定位坐标。

【代码7-4-11】plate\_locate.ipynb

# 执行车牌精定位

new\_x1,new\_y1,new\_x2,new\_y2 = getPrecisePosition(img, x1,y1,x2,y2)

根据精定位得到的车牌位置坐标，从原车辆图像得到车牌区域子图并显示。

【代码7-4-12】plate\_locate.ipynb

# 显示车牌子图

plateImg = img[new\_y1:new\_y2, new\_x1:new\_x2]

# opencv中的通道顺序是BGR，与Python的RGB刚好相反。因此此处需要调整顺序。

plateImg = plateImg[:,:,(2,1,0)]

plt.imshow(plateImg)

plt.show()

完整的代码参见代码7-4。

【代码7-4】plate\_locate.ipynb

# 定义车牌精定位函数

def getPrecisePosition(srcImg, x1,y1,x2,y2):

height = srcImg.shape[0]

width = srcImg.shape[1]

#对车牌区域进行适当扩展，避免颜色偏色带来的定位误差

x1 = x1-5

x1 = max(0,x1)

x2 = x2+5

x2 = min(x2,width)

y1 = y1-5

y1 = max(y1,0)

y2 = y2+5

y2 = min(y2, height)

plateImg = srcImg[y1:y2, x1:x2]

plateImg\_gray = cv2.cvtColor(plateImg,cv2.COLOR\_RGB2GRAY)

Canny\_img = cv2.Canny(plateImg\_gray, 85, 150)

ret, plate\_edge = cv2.threshold(Canny\_img, 127, 255, cv2.THRESH\_BINARY)

binary\_0\_1 = (plate\_edge > 0) \* 1.0

hist\_row = binary\_0\_1.sum(axis=1) #按行统计非零像素个数

hist\_col = binary\_0\_1.sum(axis=0) #按列统计非零像素个数

row\_index = np.where(hist\_row > 15) #统计像素数总和超过阈值的行下标

# 车牌上下边界精调

if row\_index: # row\_range 不为空

if(len(row\_index[0])>1):

dy1= row\_index[0][0]

y1 = y1 + dy1 + 1 #车牌上边界精调

dy2 = plate\_edge.shape[0] - row\_index[0][-1] #车牌下边界

y2 = y2 - dy2 - 1

else: #缺失一个边界，不再进行精调

y1 = y1

y2 = y2

else: #row\_range 为空

y1 = y1

y2 = y2

# 车牌左右边界精调

col\_index = np.where(hist\_col > 15) #统计像素数总和超过阈值的列下标

if col\_index:

if(len(col\_index[0])>1):

dx1 = col\_index[0][0]

x1 = x1 + dx1 + 1 ##车牌左边界精调

dx2 = plate\_edge.shape[1] - col\_index[0][-1]

x2 = x2 - dx2 - 1 #车牌右边界精调

else:

x1 = x1

x2 = x2

else:

x1 = x1

x2 = x2

return x1,y1,x2,y2

# 执行车牌精定位

new\_x1,new\_y1,new\_x2,new\_y2 = getPrecisePosition(img, x1,y1,x2,y2)

# 显示车牌子图

plateImg = img[new\_y1:new\_y2, new\_x1:new\_x2]

# opencv中的通道顺序是BGR，与Python的RGB刚好相反。因此此处需要调整顺序。

plateImg = plateImg[:,:,(2,1,0)]

plt.imshow(plateImg)

plt.show()

执行该代码块。

获取到的车牌子图如图9所示。



图9 精定位获取到的车牌子图

7.5在原图画出车牌定位结果

根据精定位得到的车牌位置坐标，在原图上画出车牌定位结果矩形框。

【代码7-5】plate\_locate.ipynb

# 在原图上画出车牌区域矩形框

ptLeftTop = (new\_x1, new\_y1)

ptRightBottom = (new\_x2, new\_y2)

point\_color = (0, 0, 255) # BGR

thickness = 2

lineType = 4

cv2.rectangle(img,ptLeftTop,ptRightBottom,point\_color,thickness,lineType)

# 在原图显示车牌框选效果

srcImg\_show = img[:,:,(2,1,0)]

plt.imshow(srcImg\_show)

plt.show()

执行该代码块。

在原图上画出车牌定位结果矩形框,如图10所示。

|  |
| --- |
|  |
| 图10 车牌精定位结果 |

8. 案例代码

【案例代码】plate\_locate.ipynb

# -\*- coding: utf-8 -\*-

'''

车牌定位

'''

import cv2

import numpy as np

import matplotlib.pyplot as plt

path = 'input/car.jpeg'

img = cv2.imread(path)

img\_GaussBlur = cv2.GaussianBlur(img,(5,5),0)# 高斯模糊

# RGB通道分离

img\_B = cv2.split(img\_GaussBlur)[0]

img\_G = cv2.split(img\_GaussBlur)[1]

img\_R = cv2.split(img\_GaussBlur)[2]

# 读取灰度图和HSV空间图

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

img\_HSV = cv2.cvtColor(img\_GaussBlur, cv2.COLOR\_BGR2HSV)

# 获取车牌感兴趣区域二值图

def getPlateROI(img\_gray, img\_HSV, img\_B, img\_R):

height = img\_gray.shape[0] # 获取图像的高

width = img\_gray.shape[1] # 获取图像的宽

for row in range(height):

for col in range(width):

# 普通蓝色车牌，同时排除透明反光物质的干扰

if (img\_HSV[:, :, 0][row, col] > 100) and \

(img\_HSV[:, :, 0][row, col]<115) and \

(img\_B[row, col] > 70) and \

(img\_R[row, col] < 40):

img\_gray[row, col] = 255

else:

img\_gray[row, col] = 0

img\_gray\_blur = cv2.GaussianBlur(img\_gray, (9, 9), 0) # 高斯平滑

# 二值化

\_, img\_binary = cv2.threshold(img\_gray\_blur, 100, 255, cv2.THRESH\_BINARY)

img\_sec\_blur = cv2.GaussianBlur(img\_binary, (9, 9), 0)# 高斯平滑

# 二值化

\_, img\_sec\_binary = cv2.threshold(img\_sec\_blur, 100, 255, cv2.THRESH\_BINARY)

return img\_sec\_binary

# 显示车牌感兴趣区域

img\_binary = getPlateROI(img\_gray, img\_HSV, img\_B, img\_R)

# 显示车牌感兴趣区域二值图

plt.imshow(img\_binary,cmap ='gray')

plt.show()

# 根据车牌感兴趣区域，得到车牌粗略定位

def getRoughPosition(img\_binary):

binary\_0\_1 = (img\_binary > 0) \* 1.0

hist\_col = binary\_0\_1.sum(axis=0) #按列统计非零像素个数

hist\_row = binary\_0\_1.sum(axis=1) #按行统计非零像素个数

row\_range = np.where((hist\_row>60)&(hist\_row<120))

if row\_range:#row\_range 不为空

if len(row\_range[0])> 5:

for y\_index in range(len(row\_range[0])-1):

cur\_value = row\_range[0][y\_index]

next\_value = row\_range[0][y\_index+1]

next2\_value = row\_range[0][y\_index+2]

next3\_value = row\_range[0][y\_index+3]

if next\_value-cur\_value<3 and \

next2\_value-cur\_value<4 and \

next3\_value-cur\_value<5:

y1 = cur\_value #确定车牌上边界

break

for y\_index in range(len(row\_range[0])-1,-1,-1):

cur\_value = row\_range[0][y\_index]

last\_value = row\_range[0][y\_index-1]

last2\_value = row\_range[0][y\_index-2]

last3\_value = row\_range[0][y\_index-3]

if last\_value-cur\_value<3 and \

last2\_value-cur\_value <4 and \

last3\_value-cur\_value<5:

y2 = cur\_value #确定车牌下边界

break

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

col\_range = np.where((hist\_col>10)&(hist\_col<60))

if col\_range:

if len(col\_range[0])>10:

for x\_index in range(len(col\_range[0])-1):

cur\_value = col\_range[0][x\_index]

next\_value = col\_range[0][x\_index+1]

next2\_value = col\_range[0][x\_index+2]

next3\_value = col\_range[0][x\_index+3]

if next\_value-cur\_value<3 and \

next2\_value-cur\_value <4 and \

next3\_value-cur\_value<5:

x1 = cur\_value #确定车牌左边界

break

for x\_index in range(len(col\_range[0])-1,-1,-1):

cur\_value = col\_range[0][x\_index]

last\_value = col\_range[0][x\_index-1]

last2\_value = col\_range[0][x\_index-2]

last3\_value = col\_range[0][x\_index-3]

if last\_value-cur\_value<3 and \

last2\_value-cur\_value <4 and \

last3\_value-cur\_value<5:

x2 = cur\_value #确定车牌右边界

break

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

else:

print('没有找到车牌感兴趣区域！')

return 0,0,0,0

return x1,y1,x2,y2

# 执行车牌粗定位

x1,y1,x2,y2 = getRoughPosition(img\_binary)

# 显示车牌感兴趣粗略子图

rough\_plateImg = img[y1:y2, x1:x2]

# opencv中的通道顺序是BGR，与Python的RGB刚好相反。因此此处需要调整顺序。

rough\_plateImg = rough\_plateImg[:,:,(2,1,0)]

plt.imshow(rough\_plateImg)

plt.show()

# 定义车牌精定位函数

def getPrecisePosition(srcImg, x1,y1,x2,y2):

height = srcImg.shape[0]

width = srcImg.shape[1]

#对车牌区域进行适当扩展，避免颜色偏色带来的定位误差

x1 = x1-5

x1 = max(0,x1)

x2 = x2+5

x2 = min(x2,width)

y1 = y1-5

y1 = max(y1,0)

y2 = y2+5

y2 = min(y2, height)

plateImg = srcImg[y1:y2, x1:x2]

plateImg\_gray = cv2.cvtColor(plateImg,cv2.COLOR\_RGB2GRAY)

Canny\_img = cv2.Canny(plateImg\_gray, 85, 150)

ret, plate\_edge = cv2.threshold(Canny\_img, 127, 255, cv2.THRESH\_BINARY)

binary\_0\_1 = (plate\_edge > 0) \* 1.0

hist\_row = binary\_0\_1.sum(axis=1) #按行统计非零像素个数

hist\_col = binary\_0\_1.sum(axis=0) #按列统计非零像素个数

row\_index = np.where(hist\_row > 15) #统计像素数总和超过阈值的行下标

# 车牌上下边界精调

if row\_index: # row\_range 不为空

if(len(row\_index[0])>1):

dy1= row\_index[0][0]

y1 = y1 + dy1 + 1 #车牌上边界精调

dy2 = plate\_edge.shape[0] - row\_index[0][-1] #车牌下边界

y2 = y2 - dy2 - 1

else: #缺失一个边界，不再进行精调

y1 = y1

y2 = y2

else: #row\_range 为空

y1 = y1

y2 = y2

# 车牌左右边界精调

col\_index = np.where(hist\_col > 15) #统计像素数总和超过阈值的列下标

if col\_index:

if(len(col\_index[0])>1):

dx1 = col\_index[0][0]

x1 = x1 + dx1 + 1 ##车牌左边界精调

dx2 = plate\_edge.shape[1] - col\_index[0][-1]

x2 = x2 - dx2 - 1 #车牌右边界精调

else:

x1 = x1

x2 = x2

else:

x1 = x1

x2 = x2

return x1,y1,x2,y2

# 执行车牌精定位

new\_x1,new\_y1,new\_x2,new\_y2 = getPrecisePosition(img, x1,y1,x2,y2)

# 显示车牌子图

plateImg = img[new\_y1:new\_y2, new\_x1:new\_x2]

# opencv中的通道顺序是BGR，与Python的RGB刚好相反。因此此处需要调整顺序。

plateImg = plateImg[:,:,(2,1,0)]

plt.imshow(plateImg)

plt.show()

# 在原图上画出车牌区域矩形框

ptLeftTop = (new\_x1, new\_y1)

ptRightBottom = (new\_x2, new\_y2)

point\_color = (0, 0, 255) # BGR

thickness = 2

lineType = 4

cv2.rectangle(img,ptLeftTop,ptRightBottom,point\_color,thickness,lineType)

# 在原图显示车牌框选效果

srcImg\_show = img[:,:,(2,1,0)]

plt.imshow(srcImg\_show)

plt.show()