

# 图像识别小项目

## 车道线检测

步骤：

1. 读取图片
2. Canny边缘检测
3. roi\_mask获取感兴趣区域
4. 霍夫变换(只用于灰度图，常用来获取圆或者是直线的算法)获取直线
5. 离群值过滤
6. 最小二乘拟合
7. 绘制直线

### 1.边缘检测

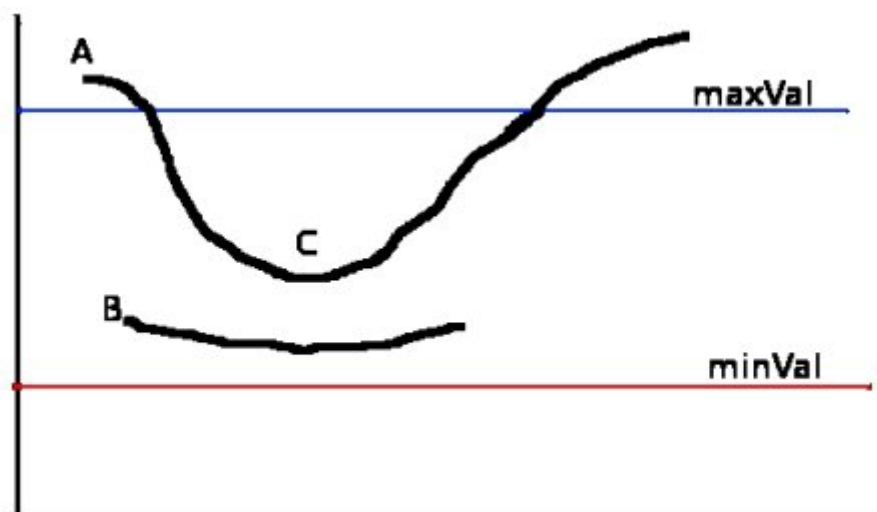
```
1 def get_edge_img(img, canny_threshold1=50, canny_threshold2=100):
2     """
3     灰度化, canny变换, 提取边缘
4     :param img: 彩色图
5     :param canny_threshold1:
6     :param canny_threshold2:
7     :return:
8     """
9     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10    edges_img = cv2.Canny(gray_img, canny_threshold1, canny_threshold2)
11    return edges_img
```

先将彩色图像转换为灰度图。

#### Canny()方法的步骤：

1. 利用高斯模糊来消除噪声。使用5\*5的高斯滤波器来去除噪声。
2. 计算图像梯度。对平滑后的图像使用Sobel算子计算水平方向和竖直方向的一阶导数（图像梯度）（Gx和Gy）。梯度的方向一般总是与边界垂直，一般被归为四类：垂直、水平和两个对角线。
3. 非极大值抑制。获得梯度的方向和大小之后，遍历整个图像，去除非边界上的点。检查每个像素点，看这个点的梯度是不是周围梯度方向相同的点中最大的。
4. 滞后阈值（确定真正的边界）。首先设置两个阈值：minVal和maxVal。当图像灰度梯度高于maxVal时被认为是真正的边界，低于minVal的边界被剔除。介于

二者之间的话，如果他和真正的边界点相连，那么他就是真正的边界点，不是就剔除。



## 2.获取ROI

```
1 def ROI_mask(gray_img):
2     """
3     对gray_img进行掩膜
4     :param gray_img:
5     :return:
6     """
7     poly_pts = np.array([[[0, 368], [300, 210], [340, 210], [640, 368]]])
8     mask = np.zeros_like(gray_img)
9     mask = cv2.fillPoly(mask, pts=poly_pts, color=255)
10    img_mask = cv2.bitwise_and(gray_img, mask)
11    return img_mask
```

### 步骤:

1. 四个点的坐标通过windows自带的画图就可以得到。顺序是（左下，左上，右上，右下）。
2. np.zeros\_like的作用是生成和gray\_img形状一样的矩阵，其元素全部为0。
3. fillpoly()用来绘制多边形并且进行填充
  - mask是把多边形画在mask上面
  - pts=poly\_pts是多边形的顶点集
4. 利用bitwise\_and()方法把原图像和mask进行与操作即可得到ROI

### 3.获取图像中线段

```
1 def get_lines(edge_img):
2     """
3     获取edge_img中的所有的线段
4     :param edge_img: 标记边缘的灰度图
5     :return:
6     """
7
8     def calculate_slope(line):
9         """
10        计算线段line的斜率
11        :param line: np.array([x_1, y_1, x_2, y_2])
12        :return:
13        """
14        x_1, y_1, x_2, y_2 = line[0]
15        return (y_2 - y_1) / (x_2 - x_1)
16
17    def reject_abnormal_line(lines, threshold=0.2):
18        """
19        剔除斜率不一致的线段
20        :param lines: 线段集合, [np.array([x_1, y_1, x_2, y_2]), np.array([x_1, y_1,
21        x_2, y_2]), ..., np.array([x_1, y_1, x_2, y_2])]
22        :param threshold: 斜率阈值, 如果差值大于阈值, 则剔除
23        :return:
24        """
25
26        slope = [calculate_slope(line) for line in lines]
27        while len(lines) > 0:
28            mean = np.mean(slope)
29            diff = [abs(s - mean) for s in slope]
30            idx = np.argmax(diff)
31            if diff[idx] > threshold:
32                slope.pop(idx)
33                lines.pop(idx)
34            else:
35                break
36        return lines
37
38    def least_squares_fit(lines):
39        """
40        将lines中的线段拟合成一条线段
41        :param lines: 线段集合, [np.array([x_1, y_1, x_2, y_2]), np.array([x_1, y_1,
42        x_2, y_2]), ..., np.array([x_1, y_1, x_2, y_2])]
```

```

41 :return: 线段上的两点,np.array([[xmin, ymin], [xmax, ymax]])
42 """
43
44 # 获取所有的x,y值, 转化为一维的数组
45 x_coors = np.ravel([[line[0][0], line[0][2]] for line in lines])
46 y_coors = np.ravel([[line[0][1], line[0][3]] for line in lines])
47
48 poly = np.polyfit(x_coors, y_coors, deg=1)
49 point_min = (np.min(x_coors), np.polyval(poly, np.min(x_coors)))
50 point_max = (np.max(x_coors), np.polyval(poly, np.max(x_coors)))
51
52 return np.array([point_min, point_max], dtype=np.int)
53
54 # 进行霍夫变换获取所有的直线
55 lines = cv2.HoughLinesP(edge_img, 1, np.pi / 180, 15, minLineLength=40,
56 maxLineGap=20)
57
58 # 按照斜率区分车道线
59 left_lines = [line for line in lines if calculate_slope(line) > 0]
60 right_lines = [line for line in lines if calculate_slope(line) < 0]
61
62 # 剔除离群线段
63 left_lines = reject_abnormal_line(left_lines)
64 right_lines = reject_abnormal_line(right_lines)
65
66 return least_squares_fit(left_lines), least_squares_fit(right_lines)

```

### 步骤:

1. 对Canny变换后ROI部分进行霍夫变换获取到图中所有的线段。
2. 按照斜率区分车道线
3. 剔除离群线段(利用阈值)
4. 返回最后确定的两条线段

### 方法注解:

1. **np.ravel()**: 将多维数组转换成一维数组。
2. **np.polyfit(x\_coors, y\_coors, deg=1)**: 对一组数据进行多项式拟合。  
x\_coors, y\_coors是图像的x和y坐标的数组, deg是阶数 (自变量的最高次方)
3. **np.polyval(p, x)**: 计算多项式的函数值。返回在x处的多项式的值, p为多项式系数。
4. **cv2.HoughLinesP(edge\_img, 1, np.pi / 180, 15, minLineLength=40, maxLineGap=20)**:

方法原型：HoughLinesP(image, rho, theta, threshold, lines=None, minLineLength=None, maxLineGap=None)

- image：必须是二值图像，推荐使用Canny边缘检测后的图像。
- rho：线段以像素为单位的距离精度，double类型的，推荐用1.0。
- theta：线段以弧度为单位的角度精度，推荐用numpy.pi/180。
- threshold：累加平面的阈值参数，int类型，超过设定阈值才被检测出线段，值越大，基本上意味着检出的线段越长，检出的线段个数越少。
- lines：
- minLineLength：线段以像素为单位的最小长度。
- maxLineGap：同一方向上两条线段被判定为一条线段的最大允许间隔（断裂），小于了设定值，则把两条线段当成一条线段。

## 4.绘制线段

```
1 def draw_lines(img, lines):
2     """
3     在img上面绘制lines
4     :param img:
5     :param lines: 两条线段: [np.array([[xmin1, ymin1], [xmax1, ymax1]]),
6     np.array([[xmin2, ymin2], [xmax2, ymax2]])]
7     :return:
8     """
9     left_line, right_line = lines
10    cv2.line(img, tuple(left_line[0]), tuple(left_line[1]), color=(0, 0, 255),
11    thickness=5)
12    cv2.line(img, tuple(right_line[0]), tuple(right_line[1]), color=(0, 255, 0),
13    thickness=5)
```

cv2.line中传入的端点坐标必须是tuple格式的。

## 完整程序

```
1 #!/usr/bin/env python3
2 # -*- coding:utf-8 -*-
3 # @author Dinglong Zhang
4 # @date 2022/10/24
5 # @file line-detect.py
6
7
8 import cv2
9 import numpy as np
10
```

```

11
12 def get_edge_img(img, canny_threshold1=50, canny_threshold2=100):
13     """
14     灰度化, canny变换, 提取边缘
15     :param img: 彩色图
16     :param canny_threshold1:
17     :param canny_threshold2:
18     :return:
19     """
20     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
21     edges_img = cv2.Canny(gray_img, canny_threshold1, canny_threshold2)
22     return edges_img
23
24
25 def ROI_mask(gray_img):
26     """
27     对gray_img进行掩膜
28     :param gray_img:
29     :return:
30     """
31     poly_pts = np.array([[[0, 368], [300, 210], [340, 210], [640, 368]]])
32     mask = np.zeros_like(gray_img)
33     mask = cv2.fillPoly(mask, pts=poly_pts, color=255)
34     img_mask = cv2.bitwise_and(gray_img, mask)
35     return img_mask
36
37
38 def get_lines(edge_img):
39     """
40     获取edge_img中的所有的线段
41     :param edge_img: 标记边缘的灰度图
42     :return:
43     """
44
45     def calculate_slope(line):
46         """
47         计算线段line的斜率
48         :param line: np.array([x_1, y_1, x_2, y_2])
49         :return:
50         """
51         x_1, y_1, x_2, y_2 = line[0]
52         return (y_2 - y_1) / (x_2 - x_1)
53
54     def reject_abnormal_line(lines, threshold=0.2):
55         """

```

```

56     剔除斜率不一致的线段
57     :param lines: 线段集合, [np.array([[x_1, y_1, x_2, y_2]]),np.array([[x_1, y_1,
x_2, y_2]]),...,np.array([[x_1, y_1, x_2, y_2]])]
58     :param threshold: 斜率阈值,如果差值大于阈值, 则剔除
59     :return:
60     """
61
62     slope = [calculate_slope(line) for line in lines]
63     while len(lines) > 0:
64         mean = np.mean(slope)
65         diff = [abs(s - mean) for s in slope]
66         idx = np.argmax(diff)
67         if diff[idx] > threshold:
68             slope.pop(idx)
69             lines.pop(idx)
70         else:
71             break
72     return lines
73
74 def least_squares_fit(lines):
75     """
76     将lines中的线段拟合成一条线段
77     :param lines: 线段集合, [np.array([[x_1, y_1, x_2, y_2]]),np.array([[x_1, y_1,
x_2, y_2]]),...,np.array([[x_1, y_1, x_2, y_2]])]
78     :return: 线段上的两点,np.array([xmin, ymin], [xmax, ymax])
79     """
80
81     # 获取所有的x,y值, 转化为一维的数组
82     x_coors = np.ravel([line[0][0], line[0][2]] for line in lines)
83     y_coors = np.ravel([line[0][1], line[0][3]] for line in lines)
84
85     poly = np.polyfit(x_coors, y_coors, deg=1)
86     point_min = (np.min(x_coors), np.polyval(poly, np.min(x_coors)))
87     point_max = (np.max(x_coors), np.polyval(poly, np.max(x_coors)))
88
89     return np.array([point_min, point_max], dtype=np.int)
90
91     # 进行霍夫变换获取所有的直线
92     lines = cv2.HoughLinesP(edge_img, 1, np.pi / 180, 15, minLineLength=40,
maxLineGap=20)
93
94     # 按照斜率区分车道线
95     left_lines = [line for line in lines if calculate_slope(line) > 0]
96     right_lines = [line for line in lines if calculate_slope(line) < 0]
97

```

```

98     # 剔除离群线段
99     left_lines = reject_abnormal_line(left_lines)
100    right_lines = reject_abnormal_line(right_lines)
101
102    return least_squares_fit(left_lines), least_squares_fit(right_lines)
103
104
105    def draw_lines(img, lines):
106        """
107        在img上面绘制lines
108        :param img:
109        :param lines: 两条线段: [np.array([[xmin1, ymin1], [xmax1, ymax1]]),
110        np.array([[xmin2, ymin2], [xmax2, ymax2]])]
111        :return:
112        """
113        left_line, right_line = lines
114        cv2.line(img, tuple(left_line[0]), tuple(left_line[1]), color=(0, 0, 255),
115        thickness=5)
116
117        cv2.line(img, tuple(right_line[0]), tuple(right_line[1]), color=(0, 255, 0),
118        thickness=5)
119
120
121    def show_line(color_img):
122        """
123        在color_img上面画出车道线
124        :param color_img:
125        :return:
126        """
127        edge_img = get_edge_img(color_img)
128        mask_gray_img = ROI_mask(edge_img)
129        lines = get_lines(mask_gray_img)
130        draw_lines(color_img, lines)
131        return color_img
132
133
134    # 识别图片
135    color_img = cv2.imread('img.jpg')
136    result = show_line(color_img)
137    cv2.imshow('output', result)
138    cv2.waitKey(0)

```



## 信用卡识别

### 步骤:

如图所示分为四个组来处理。

### 对模板图像的处理:

1. 读取一个模板图像，获取灰度图。
2. 计算二值图像。
3. 计算和绘制轮廓。

### 对需识别图像的处理:

1. 读取图像，并获取灰度图。
2. 进行礼帽操作，突出原图像中更加明亮的区域。
3. 使用sobel算子进行边缘检测。
4. 做一次闭操作来把数字连在一起。
5. 利用OTSU(大津二值化算法)来获得二值图像。
6. 数字之间空隙比较大，再进行一次闭操作。
7. 计算轮廓，对轮廓进行遍历，把属于卡号部分的留下来。
8. 把卡号的每一个数字和模板中的数字进行比对，利用matchTemplate()进行模板比对操作。
9. 把获得的结果显示在原图像中。

## 对模板图像的处理

```
1 # 读取一个模板图像
2 img = cv2.imread('./images/ocr_a_reference.png')
3 cv_show('img', img)
4 # 灰度图
5 ref = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
6 cv_show('ref', ref)
7 # 二值图像
8 ref = cv2.threshold(ref, 10, 255, cv2.THRESH_BINARY_INV)[1]
9 cv_show('ref', ref)
10
11 # 计算轮廓
12 # cv2.findContours()函数接受的参数为二值图，即黑白图像（不是灰度图）
13 # cv2.RETR_EXTERNAL只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE只保留终点坐
   标
14 # 返回的list中的每个元素都是图像中的一个轮廓
15 refCnts, hierarchy = cv2.findContours(ref.copy(), cv2.RETR_EXTERNAL,
   cv2.CHAIN_APPROX_SIMPLE)
16
```

```

17 # 绘制轮廓
18 # -1表示绘制所有的
19 cv2.drawContours(img, refCnts, -1, (0, 0, 255), 3)
20 cv_show('img', img)
21
22 print(np.array(refCnts).shape) # 一共有10个轮廓
23 refCnts = myutils.sort_contours(refCnts, method="left-to-right")[0]
24 digits = {}
25
26 # 遍历每一个轮廓
27 for (i, c) in enumerate(refCnts):
28     # 计算外接矩形并且resize成合适的大小
29     (x, y, w, h) = cv2.boundingRect(c)
30     # 获取感兴趣的区域
31     roi = ref[y:y + h, x:x + w]
32     roi = cv2.resize(roi, (57, 88))
33     # 每一个数字对应一个模板
34     digits[i] = roi

```

### 方法注解：

1.cv2.findContours(image, mode, method, contours=None, hierarchy=None, offset=None):

- **image**：单通道图像，最好是二值图像。一般是经过Canny、拉普拉斯等边缘检测算子处理过的二值图像。
- **mode**：定义轮廓的检索模式。有几个模式可选：
  - **CV\_RETR\_EXTERNAL**只检测最外围轮廓，包含在外围轮廓内的内围轮廓被忽略。
  - **CV\_RETR\_LIST** 检测所有的轮廓，包括内围、外围轮廓，但是检测到的轮廓不建立等级关系，彼此之间相互独立，没有等级关系，这就意味着**这个检索模式下不存在父轮廓或内嵌轮廓**，所以hierarchy向量内所有元素的第3、第4个分量都会被置为-1。
  - **CV\_RETR\_CCOMP** 检测所有的轮廓，但所有轮廓只建立两个等级关系，外围为顶层，若外围内的内围轮廓还包含了其他的轮廓信息，则内围内的所有轮廓均归属于顶层。
  - **CV\_RETR\_TREE** 检测所有轮廓，所有轮廓建立一个等级树结构。外层轮廓包含内层轮廓，内层轮廓还可以继续包含内嵌轮廓。
- **method**：定义轮廓的近似方法。
  - **CV\_CHAIN\_APPROX\_NONE** 保存物体边界上所有连续的轮廓点到contours向量内。
  - **CV\_CHAIN\_APPROX\_SIMPLE** 仅保存轮廓的拐点信息，把所有轮廓拐点处的点保存入contours向量内，拐点与拐点之间直线段上的信息点不予保

留。

- **CV\_CHAIN\_APPROX\_TC89\_L1, CV\_CHAIN\_APPROX\_TC89\_KCOS** 使用teh-Chinl chain 近似算法。
- **contours**: 是一个双重向量, 向量内每个元素保存了一组由连续的point点构成的点的集合向量, 每一组point点就是一个轮廓。有多少轮廓, contours中就有多少元素。
- **hierarchy**: 向量hierarchy内的元素和轮廓向量contours内的元素是一一对应的, 向量的容量相同。hierarchy向量内每一个元素的4个int型变量——`hierarchy[i][0] ~ hierarchy[i][3]`, 分别表示第i个轮廓的后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号。如果当前轮廓没有对应的后一个轮廓、前一个轮廓、父轮廓或内嵌轮廓的话, 则`hierarchy[i][0] ~ hierarchy[i][3]`的相应位被设置为默认值-1。
- **offset**: 偏移量, 所有的轮廓信息相对于原始图像对应点的偏移量, 相当于在每一个检测出的轮廓点上加上该偏移量, 并且Point还可以是负值。

2.cv2.boundingRect(array):计算外接矩形。

## 对需识别图像的处理

### 预处理

```
1  # 初始化卷积核
2  rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 3))
3  sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
4
5  # 读取输入图像, 并且进行预处理
6  image = cv2.imread('./images/credit_card_01.png')
7  cv_show('image', image)
8  image = myutils.resize(image, width=300)
9  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10 cv_show('gray', gray)
11
12 # 礼帽操作 (原图像-开运算后的图像), 可以得到原图像中的噪声, 突出更加明亮的区域
13 tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, rectKernel)
14 cv_show('tophat', tophat)
15
16 # Sobel算子
17 gradX = cv2.Sobel(tophat, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=-1) # -1是按照 (3*3) 来进行计算
18 gradX = np.absolute(gradX)
19 (minVal, maxVal) = (np.min(gradX), np.max(gradX))
20 gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))
```

```

21 gradX = gradX.astype("uint8")
22
23 print(np.array(gradX).shape)
24 cv_show('gradX', gradX)
25
26 # 开操作把通不过的都断开，闭操作把进不去的都填上
27
28 # 通过闭操作（先膨胀，再腐蚀）将数字连在一起
29 gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE, rectKernel)
30 cv_show('gradX', gradX)
31 # THRESH_OTSU(大津二值化算法)会自动寻找合适的阈值，适合双峰，需把阈值参
    数设置为0
32 thresh = cv2.threshold(gradX, 0, 255, cv2.THRESH_BINARY |
    cv2.THRESH_OTSU)[1]
33 cv_show('thresh', thresh)
34 # 再来一个闭操作
35 thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, sqKernel)
36 cv_show('thresh', thresh)
37
38 # 计算轮廓
39 threshCnts, hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
40 cur_image = image.copy()
41 cv2.drawContours(cur_image, threshCnts, -1, (0, 0, 255), 3)
42 cv_show('img', cur_image)

```

方法注解：

**1.cv2.getStructuringElement(int shape, Size esize, Point anchor = Point(-1, -1))**返回指定形状和尺寸的结构元素。

- shape：内核的形状，有三种：
  - MORPH\_RECT 矩形
  - MORPH\_CROSS 交叉形
  - MORPH\_ELLIPSE 椭圆形
- esize：内核的尺寸，矩形的宽、高格式为(width,height)
- anchor：锚点的位置。默认值Point (-1,-1)，表示锚点位于中心点。

**2.cv2.morphologyEx(src,op,kernel,anchor,iterations,borderType,borderValue)** 形态学操作

- src：输入的图像矩阵，二值图像
- op：形态学操作类型

- cv2.MORPH\_OPEN 开运算，先**腐蚀**后**膨胀**，主要用来去除一些较**亮**的部分，即先腐蚀掉不要的部分，再进行膨胀。
- cv2.MORPH\_CLOSE 闭运算，先**膨胀**后**腐蚀**，主要用来去除一些较**暗**的部分。
- cv2.MORPH\_GRADIENT 形态梯度，膨胀运算结果减去腐蚀运算结果，可以拿到轮廓信息。
- cv2.MORPH\_TOPHAT 顶帽运算，原图像减去开运算结果。
- cv2.MORPH\_BLACKHAT 底帽运算，原图像减去闭运算结果。
- kernel: 进行腐蚀操作的核，可以通过getStructuringElement()获得。
- anchor: 锚点，默认为(-1,-1)
- iterations:腐蚀操作的次数，默认为1
- borderType: 边界种类
- borderValue:边界值

## 比对, 识别

```

1  # 用来存储外接矩形
2  locs = []
3  # 遍历轮廓
4  for (i, c) in enumerate(threshCnts):
5      # 计算矩形
6      (x, y, w, h) = cv2.boundingRect(c)
7      ar = w / float(h)
8      # 筛选合适的区域留下来
9      if 2.5 < ar < 4.0:
10         if (40 < w < 55) and (10 < h < 20):
11             # 符合条件的留下来
12             locs.append((x, y, w, h))
13
14  locs = sorted(locs, key=lambda x:x[0]) # 利用key来进行排序, key为每个x的第一个元素
15
16  # 用来储存最后结果
17  output = []
18  # 遍历每一个轮廓中的数字
19  for (i, (gX, gY, gW, gH)) in enumerate(locs):
20      GroupOutput = []
21
22      # 根据坐标来提取每一个组
23      group = gray[gY - 5:gY + gH + 5, gX - 5:gX + gW + 5]
24      cv_show('group', group)
25

```

```

26 # 预处理
27 group = cv2.threshold(group, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)[1]
28 cv_show('group', group)
29
30 # 计算每一组的轮廓
31 digitCnts, hierarchy = cv2.findContours(group.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
32
33 digitCnts = contours.sort_contours(digitCnts, method="left-to-right")[0]
34
35 # 计算每一组中的每一个数值
36 for c in digitCnts:
37     # 找到当前数值的轮廓, resize成合适的大小
38     (x, y, w, h) = cv2.boundingRect(c)
39     roi = group[y:y + h, x:x + w]
40     roi = cv2.resize(roi, (57, 88))
41     cv_show('roi', roi)
42
43     # 计算匹配得分
44     scores = []
45     # 在模板中计算每一个得分
46     for (digit, digitROI) in digits.items(): # digits里面保存的是10个模板
47         # 模板匹配
48         result = cv2.matchTemplate(roi, digitROI, cv2.TM_CCOEFF)
49         print(result)
50         (_, score, _, _) = cv2.minMaxLoc(result)
51         scores.append(score)
52
53     # 找到最合适的数字
54     GroupOutput.append(str(np.argmax(scores)))
55
56 # 画出来
57 cv2.rectangle(image, (gX - 5, gY - 5), (gX + gW + 5, gY + gH + 5), (0, 0,
255), 1)
58 cv2.putText(image, "".join(GroupOutput), (gX, gY - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)
59
60 # 得到结果
61 output.extend(GroupOutput)
62
63 # 打印结果
64 print("Credit Card Type: {}".format(FIRST_NUMBER[output[0]]))
65 print("Credit Card #: {}".format("".join(output)))
66 cv2.imshow("Image", image)

```

先把所有的轮廓都遍历一遍，然后通过长宽的比值来留下符合要求的轮廓。

对留下来的轮廓进行排序

找出四组卡号所对应的轮廓，然后计算每一组中的每个数值，进行模板匹配，在原图像上面画出来。

## 完整代码

```
1  #!/usr/bin/env python3
2  # -*- coding:utf-8 -*-
3  # @author Dinglong Zhang
4  # @date 2022/10/17
5  # @file ocr-template-py.py
6
7  import cv2
8  import numpy as np
9  import myutils
10 from imutils import contours
11
12 # 指定信用卡类型
13 FIRST_NUMBER = {
14     "3": "Americian Express",
15     "4": "Visa",
16     "5": "MasterCard",
17     "6": "Discover Card"
18 }
19
20 # 绘图展示
21 def cv_show(name, img):
22     cv2.imshow(name, img)
23     cv2.waitKey(0)
24     cv2.destroyAllWindows()
25
26
27 # 读取一个模板图像
28 img = cv2.imread('./images/ocr_a_reference.png')
29 cv_show('img', img)
30 # 灰度图
31 ref = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
32 cv_show('ref', ref)
33 # 二值图像
34 ref = cv2.threshold(ref, 10, 255, cv2.THRESH_BINARY_INV)[1]
35 cv_show('ref', ref)
```

```
36
37 # 计算轮廓
38 # cv2.findContours()函数接受的参数为二值图，即黑白图像（不是灰度图）
39 # cv2.RETR_EXTERNAL只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE只保留终点
    坐标
40 # 返回的list中的每个元素都是图像中的一个轮廓
41 refCnts, hierarchy = cv2.findContours(ref.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
42
43 # 绘制轮廓
44 cv2.drawContours(img, refCnts, -1, (0, 0, 255), 3)
45 cv_show('img', img)
46
47 print(np.array(refCnts).shape) # 一共有10个轮廓
48 refCnts = myutils.sort_contours(refCnts, method="left-to-right")[0]
49 digits = {}
50
51 # 遍历每一个轮廓
52 for (i, c) in enumerate(refCnts):
53     # 计算外接矩形并且resize成合适的大小
54     (x, y, w, h) = cv2.boundingRect(c)
55     # 获取感兴趣的区域
56     roi = ref[y:y + h, x:x + w]
57     roi = cv2.resize(roi, (57, 88))
58     # 每一个数字对应一个模板
59     digits[i] = roi
60
61 # 初始化卷积核
62 rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 3))
63 sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
64
65 # 读取输入图像，并且进行预处理
66 image = cv2.imread('./images/credit_card_01.png')
67 cv_show('image', image)
68 image = myutils.resize(image, width=300)
69 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
70 cv_show('gray', gray)
71
72 # 礼帽操作（原图像-开运算后的图像），可以得到原图像中的噪声，突出更加明亮
    的区域
73 tophat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT, rectKernel)
74 cv_show('tophat', tophat)
75
76 # Sobel算子
```



```
77 gradX = cv2.Sobel(tophat, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=-1) # -1
    是按照 (3*3) 来进行计算
78 gradX = np.absolute(gradX)
79 (minVal, maxVal) = (np.min(gradX), np.max(gradX))
80 gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))
81 gradX = gradX.astype("uint8")
82
83 print(np.array(gradX).shape)
84 cv_show('gradX', gradX)
85
86 # 开操作把通不过的都断开，闭操作把进不去的都填上
87
88 # 通过闭操作（先膨胀，再腐蚀）将数字连在一起
89 gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE, rectKernel)
90 cv_show('gradX', gradX)
91 # THRESH_OTSU(大津二值化算法)会自动寻找合适的阈值，适合双峰，需把阈值参
    数设置为0
92 thresh = cv2.threshold(gradX, 0, 255, cv2.THRESH_BINARY |
    cv2.THRESH_OTSU)[1]
93 cv_show('thresh', thresh)
94 # 再来一个闭操作
95 thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, sqKernel)
96 cv_show('thresh', thresh)
97
98 # 计算轮廓
99 threshCnts, hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
100 cur_image = image.copy()
101 cv2.drawContours(cur_image, threshCnts, -1, (0, 0, 255), 3)
102 cv_show('img', cur_image)
103
104 # 用来存储外接矩形
105 locs = []
106 # 遍历轮廓
107 for (i, c) in enumerate(threshCnts):
108     # 计算矩形
109     (x, y, w, h) = cv2.boundingRect(c)
110     ar = w / float(h)
111     # 筛选合适的区域留下来
112     if 2.5 < ar < 4.0:
113         if (40 < w < 55) and (10 < h < 20):
114             # 符合条件的留下来
115             locs.append((x, y, w, h))
116
```

```
117 locs = sorted(locs, key=lambda x:x[0]) # 利用key来进行排序, key为每个x的第
    一个元素
118
119 # 用来储存最后结果
120 output = []
121 # 遍历每一个轮廓中的数字
122 for (i, (gX, gY, gW, gH)) in enumerate(locs):
123     GroupOutput = []
124
125     # 根据坐标来提取每一个组
126     group = gray[gY - 5:gY + gH + 5, gX - 5:gX + gW + 5]
127     cv_show('group', group)
128
129     # 预处理
130     group = cv2.threshold(group, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)[1]
131     cv_show('group', group)
132
133     # 计算每一组的轮廓
134     digitCnts, hierarchy = cv2.findContours(group.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
135
136     digitCnts = contours.sort_contours(digitCnts, method="left-to-right")[0]
137
138     # 计算每一组中的每一个数值
139     for c in digitCnts:
140         # 找到当前数值的轮廓, resize成合适的大小
141         (x, y, w, h) = cv2.boundingRect(c)
142         roi = group[y:y + h, x:x + w]
143         roi = cv2.resize(roi, (57, 88))
144         cv_show('roi', roi)
145
146         # 计算匹配得分
147         scores = []
148         # 在模板中计算每一个得分
149         for (digit, digitROI) in digits.items(): # digits里面保存的是10个模板
150             # 模板匹配
151             result = cv2.matchTemplate(roi, digitROI, cv2.TM_CCOEFF)
152             print(result)
153             (_, score, _, _) = cv2.minMaxLoc(result)
154             scores.append(score)
155
156         # 找到最合适的数字
157         GroupOutput.append(str(np.argmax(scores)))
158
```

```

159     # 画出来
160     cv2.rectangle(image, (gX - 5, gY - 5), (gX + gW + 5, gY + gH + 5), (0, 0,
255), 1)
161     cv2.putText(image, "".join(GroupOutput), (gX, gY - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)
162
163     # 得到结果
164     output.extend(GroupOutput)
165
166 # 打印结果
167 print("Credit Card Type: {}".format(FIRST_NUMBER[output[0]]))
168 print("Credit Card #: {}".format("".join(output)))
169 cv2.imshow("Image", image)
170 cv2.waitKey(0)

```

## myutils.py

```

1  import cv2
2
3  def sort_contours(cnts, method="left-to-right"):
4      reverse = False
5      i = 0
6
7      if method == "right-to-left" or method == "bottom-to-top":
8          reverse = True
9
10     if method == "top-to-bottom" or method == "bottom-to-top":
11         i = 1
12     boundingBoxes = [cv2.boundingRect(c) for c in cnts] #用一个最小的矩形,
把找到的形状包起来x,y,h,w
13     (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes),
14                                         key=lambda b: b[1][i], reverse=reverse))
15
16     return cnts, boundingBoxes
17 def resize(image, width=None, height=None, inter=cv2.INTER_AREA):
18     dim = None
19     (h, w) = image.shape[:2]
20     if width is None and height is None:
21         return image
22     if width is None:
23         r = height / float(h)
24         dim = (int(w * r), height)
25     else:
26         r = width / float(w)
27         dim = (width, int(h * r))

```

```
28     resized = cv2.resize(image, dim, interpolation=inter)
29     return resized
```

# 文档识别

步骤：

1. 读取图像，获取灰度图。
2. 利用高斯滤波消除噪声。
3. 进行Canny边缘检测。
4. 计算轮廓，遍历所有的轮廓，利用approxPolyDP()来对图像轮廓点进行多边形拟合，判断出原图像中属于菜单的部分。
5. 对图像进行透视变换。
6. 对透视变换后的图像利用OTSU(大津二值化)算法来获得二值图像。
7. 利用pytesseract这个包来进行文档的识别的操作。

## 1.预处理

**resize()函数：**对原图像的长和宽做等比例变换。

```
1  def resize(image, width=None, height=None, inter=cv2.INTER_AREA):
2      dim = None
3      (h, w) = image.shape[:2]
4      if width is None and height is None:
5          return image
6      if width is None:
7          r = height / float(h)
8          dim = (int(w * r), height)
9      else:
10         r = width / float(w)
11         dim = (width, int(h * r))
12     resized = cv2.resize(image, dim, interpolation=inter)
13     return resized
```

方法注解：

cv2.resize(InputArray src, OutputArray dst, Size, fx, fy, interpolation)

- InputArray src: 输入图片
- OutputArray dst: 输出图片
- Size: 输出图片尺寸
- fx, fy: 沿x轴, y轴的缩放系数
- interpolation: 插入方式

```
1  # 读取图像
```

```

2 image = cv2.imread('./images/receipt.jpg')
3 print(image.shape) # (3264, 2448, 3) 3264是height, 2448是width
4
5 # 坐标也会相同变化
6 ratio = image.shape[0] / 500
7 origin = image.copy()
8
9 image = resize(origin, height=500)
10
11 # 图像预处理
12 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
13 # 利用高斯滤波消除噪声
14 gray = cv2.GaussianBlur(gray, (5, 5), 0)
15 # Canny边缘检测
16 edged = cv2.Canny(gray, 75, 200)
17
18 # 展示预处理的结果
19 print("STEP1:边缘检测")
20 cv2.imshow('image', image)
21 cv2.imshow('edged', edged)
22 cv2.waitKey(0)
23 cv2.destroyAllWindows()

```

shape[0]是原图像的高度。

## 2.轮廓检测

```

1 # 轮廓检测
2 cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST,
3 cv2.CHAIN_APPROX_SIMPLE)[0]
4
5 cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]
6
7 # 遍历轮廓
8 for c in cnts:
9     # 计算轮廓近似
10     peri = cv2.arcLength(c, True)
11
12     # 参数1是源图像的某个轮廓，是一个点集
13     # 参数2是是一个距离值，表示多边形的轮廓接近实际轮廓的程度，值越小，得到的
14     # 多边形角点越多，对原图像的多边形近似效果越好。
15     # 参数3表示是否闭合
16     approx = cv2.approxPolyDP(c, 0.02 * peri, True)
17
18     # 如果是4个点的时候就拿出来
19     if len(approx) == 4:

```

```

17     screenCnt = approx
18     break
19
20 # 展示轮廓的结果
21 print("STEP2:获取轮廓")
22 cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
23 cv2.imshow('image', image)
24 cv2.waitKey(0)
25 cv2.destroyAllWindows()

```

### 方法注解:

- cv2.arcLength(cnt,True): 计算轮廓的周长（弧长），第二个参数是用来指定形状是闭合还是打开的。
- cv2.contourArea(cnt): 计算轮廓的面积。
- cv2.approxPolyDP(cnt,epsilon,True):
  - cnt: 是源图像的某个轮廓，是一个点集。
  - epsilon: 是从原始轮廓到近似轮廓的最大距离，它是一个准确率参数，值越小，得到的多边形角点越多，对原图像的多边形近似效果越好。
  - True: 表示闭合

## 3.透视变换

```

1 # 寻找原图像的四个坐标点
2 def order_points(pts):
3     # 一共有四个坐标点
4     rect = np.zeros((4, 2), dtype="float32")
5     # 按顺序0123找到四个坐标点为左上，右上，右下，左下
6     # 计算左上，右下(把x, y坐标相加，最小的是左上，最大是右下)
7     s = pts.sum(axis=1) # axis=1就是把每一行向量进行相加
8     rect[0] = pts[np.argmin(s)]
9     rect[2] = pts[np.argmax(s)]
10
11     # 计算右上，左下 (右上是y-x最小的，左下是y-x最大的)
12     diff = np.diff(pts, axis=1) # diff就是数组中a[n] - a[n-1]
13     rect[1] = pts[np.argmin(diff)]
14     rect[3] = pts[np.argmax(diff)]
15
16     return rect
17
18
19 def four_points_transform(image, pts):
20     # 获取输入坐标点

```

```

21     rect = order_points(pts)
22     (tl, tr, br, bl) = rect
23
24     # 取较大的
25     # 计算输入的w和h的值
26     widthA = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
27     widthB = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
28     maxwidth = max(int(widthA), int(widthB))
29
30     heightA = np.sqrt(((bl[0] - tl[0]) ** 2) + ((bl[1] - tl[1]) ** 2))
31     heightB = np.sqrt(((br[0] - tr[0]) ** 2) + ((br[1] - tr[1]) ** 2))
32     maxheight = max(int(heightA), int(heightB))
33
34     # 变换后对应的坐标位置
35     dst = np.array([
36         [0, 0],
37         [maxwidth - 1, 0],
38         [maxwidth - 1, maxheight - 1],
39         [0, maxheight - 1]],
40         dtype='float32'
41     )
42
43     # 计算变换矩阵
44     M = cv2.getPerspectiveTransform(rect, dst) # 通过原来的四个点和新的四个点
        来计算变换矩阵
45     warped = cv2.warpPerspective(image, M, (maxwidth, maxheight)) #
        (maxwidth, maxheight)是输出图像的大小
46
47     return warped

```

order\_points(pts)用来修正四个坐标的顺序，这个函数传入的pts本身数据是源图像四个坐标，但是顺序不正确。

- 计算左上、右下的方法：把x, y坐标相加，最小的是左上，最大是右下
- 计算右上、左下的方法：右上是y-x最小的，左下是y-x最大的

### 方法注解：

- np.diff(): 数组中 $a[n] - a[n-1]$ 。
- np.argmin(a, axis=None, out=None):给出axis方向最小值的**下标**。
  - a: INPUT ARRAY
  - axis: 默认是讲数组展平，否则，按照axis方向
  - RETURN: index\_array: 下标组成的数组。shape与输入数组a去掉axis的维度相同。

```

1 # 透视变换
2 warped = four_points_transform(origin, screenCnt.reshape(4, 2) * ratio) # 按照
   缩放的比例还原回去
3
4 # 二值处理
5 warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
6 ref = cv2.threshold(warped, 0, 255, cv2.THRESH_OTSU)[1]
7 cv2.imwrite('scan.jpg', ref)
8
9 # 展示结果
10 print("STEP3:变换")
11 cv2.imshow('Original', resize(origin, height=650))
12 cv2.imshow('Scanned', resize(ref, height=650))
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()

```

## 完整代码

```

1 #!/usr/bin/env python3
2 # -*- coding:utf-8 -*-
3 # @author Dinglong Zhang
4 # @date 2022/10/18
5 # @file scan.py
6
7 import cv2
8 import numpy as np
9
10
11 # 寻找原图像的四个坐标点(传入的pts数据的原图像的数据，只是顺序不对，
   order_points是用来改变顺序)
12 def order_points(pts):
13     print('pts', pts)
14     # 一共有四个坐标点
15     rect = np.zeros((4, 2), dtype="float32")
16     # 按顺序0123找到四个坐标点为左上，右上，右下，左下
17     # 计算左上，右下(把x, y坐标相加，最小的是左上，最大是右下)
18     s = pts.sum(axis=1)
19     print('s', s)
20     rect[0] = pts[np.argmin(s)]
21     print('rect0', rect[0])
22     rect[2] = pts[np.argmax(s)]
23
24     # 计算右上，左下 (右上是y-x最小的，左下是y-x最大的)
25     diff = np.diff(pts, axis=1)

```



```

26 rect[1] = pts[np.argmin(diff)]
27 rect[3] = pts[np.argmax(diff)]
28
29 return rect
30
31
32 def four_points_transform(image, pts):
33     # 获取输入坐标点
34     rect = order_points(pts)
35     print('rect', rect)
36     (tl, tr, br, bl) = rect
37
38     # 取较大的
39     # 计算输入的w和h的值
40     widthA = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
41     widthB = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
42     maxwidth = max(int(widthA), int(widthB))
43
44     heightA = np.sqrt(((bl[0] - tl[0]) ** 2) + ((bl[1] - tl[1]) ** 2))
45     heightB = np.sqrt(((br[0] - tr[0]) ** 2) + ((br[1] - tr[1]) ** 2))
46     maxheight = max(int(heightA), int(heightB))
47
48     # 变换后对应的坐标位置
49     dst = np.array([
50         [0, 0],
51         [maxwidth - 1, 0],
52         [maxwidth - 1, maxheight - 1],
53         [0, maxheight - 1]],
54         dtype='float32'
55     )
56
57     # 计算变换矩阵
58     M = cv2.getPerspectiveTransform(rect, dst) # 通过原来的四个点和新的四个
点来计算变换矩阵
59     warped = cv2.warpPerspective(image, M, (maxwidth, maxheight)) #
(maxwidth, maxheight)是输出图像的大小
60
61     return warped
62
63
64 def resize(image, width=None, height=None, inter=cv2.INTER_AREA):
65     dim = None
66     (h, w) = image.shape[:2]
67     if width is None and height is None:
68         return image

```

```
69     if width is None:
70         r = height / float(h)
71         dim = (int(w * r), height)
72     else:
73         r = width / float(w)
74         dim = (width, int(h * r))
75     resized = cv2.resize(image, dim, interpolation=inter)
76     return resized
77
78
79 # 读取图像
80 image = cv2.imread('./images/receipt.jpg')
81 print(image.shape) # (3264, 2448, 3) 3264是height, 2448是width
82
83 # 坐标也会相同变化
84 ratio = image.shape[0] / 500
85 origin = image.copy()
86
87 image = resize(origin, height=500)
88
89 # 图像预处理
90 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
91 # 利用高斯滤波消除噪声
92 gray = cv2.GaussianBlur(gray, (5, 5), 0)
93 # Canny边缘检测
94 edged = cv2.Canny(gray, 75, 200)
95
96 # 展示预处理的结果
97 print("STEP1:边缘检测")
98 cv2.imshow('image', image)
99 cv2.imshow('edged', edged)
100 cv2.waitKey(0)
101 cv2.destroyAllWindows()
102
103 # 轮廓检测
104 cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST,
105                          cv2.CHAIN_APPROX_SIMPLE)[0]
106 cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]
107
108 # 遍历轮廓
109 for c in cnts:
110     # 计算轮廓近似
111     peri = cv2.arcLength(c, True)
112     # 参数1是源图像的某个轮廓, 是一个点集
```

```

113     # 参数2是是一个距离值，表示多边形的轮廓接近实际轮廓的程度，值越小，得到
    的多边形角点越多，对原图像的多边形近似效果越好。
114     # 参数3表示是否闭合
115     approx = cv2.approxPolyDP(c, 0.02 * peri, True)
116
117     # 如果是4个点的时候就拿出来
118     if len(approx) == 4:
119         screenCnt = approx
120         break
121
122     # 展示轮廓的结果
123     print("STEP2:获取轮廓")
124     cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 2)
125     cv2.imshow('image', image)
126     cv2.waitKey(0)
127     cv2.destroyAllWindows()
128
129     # 透视变换
130     warped = four_points_transform(origin, screenCnt.reshape(4, 2) * ratio) # 按
    照缩放的比例还原回去
131
132     # 二值处理
133     warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
134     ref = cv2.threshold(warped, 0, 255, cv2.THRESH_OTSU)[1]
135     cv2.imwrite('scan.jpg', ref)
136
137     # 展示结果
138     print("STEP3:变换")
139     cv2.imshow('Original', resize(origin, height=650))
140     cv2.imshow('Scanned', resize(ref, height=650))
141     cv2.waitKey(0)
142     cv2.destroyAllWindows()

```

## 利用pytesseract进行OCR操作

```

1  #!/usr/bin/env python3
2  # -*- coding:utf-8 -*-
3  # @author Dinglong Zhang
4  # @date 2022/10/18
5  # @file test.py
6
7  # https://digi.bib.uni-mannheim.de/tesseract/
8  # 配置环境变量如E:\Program Files (x86)\Tesseract-OCR
9  # tesseract -v进行测试
10 # tesseract XXX.png 得到结果

```

```

11 # pip install pytesseract
12 # anaconda lib site-packages pytesseract pytesseract.py
13 # tesseract_cmd 修改为绝对路径即可
14 from PIL import Image
15 import pytesseract
16 import cv2
17 import os
18
19 preprocess = 'blur' # thresh
20
21 image = cv2.imread('14988.png')
22 # image = cv2.flip(image, -1)
23 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
24
25 # 阈值
26 if preprocess == "thresh":
27     gray = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY |
28 cv2.THRESH_OTSU)[1]
29
30 # 中值滤波
31 if preprocess == "blur":
32     gray = cv2.medianBlur(gray, 3)
33
34 filename = "{}.png".format(os.getpid()) # 获取当前进程的id, 这里叫什么名字都可以
35 cv2.imwrite(filename, gray)
36
37 text = pytesseract.image_to_string(Image.open(filename))
38 print(text)
39 os.remove(filename)
40
41 cv2.imshow("Image", image)
42 cv2.imshow("Output", gray)
43 cv2.waitKey(0)
44 cv2.destroyAllWindows()

```

## 二维码、条形码识别

### 步骤:

1. 获取摄像头资源
2. 读取授权文件
3. 遍历decode之后的图像中的码

4.判断识别出来的二维码是否已经被授权。若被授权，显示“authorized”，并且颜色为绿色；若被授权，显示“un-authorized”，并且颜色为红色。

## 完整代码

```
1  #!/usr/bin/env python3
2  # -*- coding:utf-8 -*-
3  # @author Dinglong Zhang
4  # @date 2022/10/26
5  # @file QRbartest.py
6
7  from pyzbar.pyzbar import decode
8  import cv2
9  import numpy as np
10
11 # 获取摄像头资源
12 capture = cv2.VideoCapture(0)
13
14 with open('MyDataFile.txt') as f:
15     MyDataList = f.read().splitlines() # 读取文件内容，没有\n
16 print(MyDataList)
17
18 while True:
19
20     success, img = capture.read()
21
22     for barcode in decode(img):
23         # barcode中包含data（码中存储的信息），type（码的类型），rect（左上角
24         # 坐标和宽、高），polygon（外界多边形框的四个顶点的坐标）
25
26         # print(barcode.data) # b代表的是byte,
27         mydata = barcode.data.decode('utf-8')
28         print(mydata)
29
30         if mydata in MyDataList:
31             output = 'authorized'
32             mycolor = (0, 255, 0)
33         else:
34             output = 'un-authorized'
35             mycolor = (0, 0, 255)
36
37         pts = np.array([barcode.polygon], np.int32)
38         pts = pts.reshape((-1, 1, 2)) # -1表示自动计算，shape为(4, 1, 2)。导入
        polyLines之前都要做这个操作(-1,1,2)
        cv2.polyLines(img, [pts], True, mycolor, 4)
```

```

39     pts2 = barcode.rect # barcode的外界矩形
40     # (pts2[0], pts2[1])是左上角顶点的坐标。0.9是字体大小
41     cv2.putText(img, output, (pts2[0], pts2[1]), cv2.FONT_HERSHEY_SIMPLEX,
    0.9, mycolor, 2)
42
43     cv2.imshow('result', img)
44     cv2.waitKey(1)

```

### 方法注解：

- decode()之后的barcode中包含四个信息：barcode中包含data（码中存储的信息），type（码的类型），rect（左上角坐标和宽、高），polygon（外界多边形框的四个顶点的坐标）。
- f.readlines()和f.read().splitlines()都是返回一个list，f.readlines()后面有加\n,f.read().splitlines()没有\n
- cv2.waitKey(delay):
  - 参数delay:
    - delay <= 0: 一直等待按键。
    - delay取得正整数：等待按键的时间，比如cv2.waitKey(25)，就是等待25毫秒（视频中一帧数据显示（停留）的时间）
  - 返回值:
    - 等待期间有按键：返回按键的ASCII码（比如：Esc的ASCII码为27）。
    - 等待期间没有按键：返回 -1。

## 目标追踪

H（色调）、S（饱和度）、V（明度）

HSV色彩分离的基本步骤：

1. 转换HSV表示
2. 设定目标阈值
3. 设置掩膜
4. 过滤目标颜色

### 实验步骤：

1. 导入原视频
2. 将原视频转换到HSV颜色空间
3. 求得掩膜
4. 过滤目标颜色
5. 目标追踪

# 1.导入原视频

```
1 cap = cv2.VideoCapture('green.mp4') #打开同一目录下的视频
2 while(cap.isOpened()):
3     ret, frame = cap.read() #frame保存视频每一帧
4     if ret==True: #当读取成功时
5
6         cv2.imshow('frame',frame)
7         if cv2.waitKey(10) & 0xFF == ord('q'):
8             break
9     else:
10        break
11
12 cap.release()
13 cv2.destroyAllWindows()
14
```

# 2.将原视频转换到HSV颜色空间

```
1 hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

# 3.求掩膜

计算掩膜，首先根据需要确定出颜色的阈值，然后使用cv2.inRange()来设置掩膜，只保留橙色部分。

```
1 lower = np.array([11, 43, 46])
2 upper = np.array([25, 255, 255])
3 mask2 = cv2.inRange(hsv, lower, upper)
```

# 4.过滤目标颜色

求出掩膜之后，使用cv2.bitwise\_and()操作把掩膜和原图像进行“与”操作来过滤出橙色。

```
1 res = cv2.bitwise_and(frame, frame, mask=mask2)
```

# 5.目标追踪

具体思路：

1. 使用形态学中的开运算，去除视频中橙色噪点。
2. 根据掩膜得到的（0-255）矩阵，得到物体的范围。

### 3. 根据物体的范围绘制矩形框。

```
1 kernel = np.ones((10, 10), np.uint8) # 设置开运算所需核
2 opening = cv2.morphologyEx(mask2, cv2.MORPH_OPEN, kernel) # 对得到的
mask进行开运算
3 print(opening)
4 rectangle = np.where(opening == 255) # 找出开运算后矩阵中为255的部分，即物
体范围
5 cv2.rectangle(frame, (min(rectangle[1]), min(rectangle[0])), (max(rectangle[1]),
max(rectangle[0])),
6 (0, 0, 255), 3) # 根据每一帧中物体的左上角坐标以及右下角坐标绘制矩形框
```

## 完整代码

```
1 #!/usr/bin/env python3
2 # -*- coding:utf-8 -*-
3 # @author Dinglong Zhang
4 # @date 2022/10/27
5 # @file HSV目标追踪.py
6
7 import cv2
8 import numpy as np
9
10 cap = cv2.VideoCapture('orange1.mp4')
11
12 while cap.isOpened():
13     ret, frame = cap.read()
14
15     if ret==True:# 当读取成功时
16         # 转换为HSV颜色空间
17         hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
18         # 橙色阈值范围
19         lower = np.array([11, 43, 46])
20         upper = np.array([25, 255, 255])
21         # 计算掩膜
22         mask2 = cv2.inRange(hsv, lower, upper)
23
24         res = cv2.bitwise_and(frame, frame, mask=mask2)
25
26         kernel = np.ones((10, 10), np.uint8) # 设置开运算所需核
27         opening = cv2.morphologyEx(mask2, cv2.MORPH_OPEN, kernel) # 对得
到的mask进行开运算
28         print(opening)
```



```
29     rectangle = np.where(opening == 255) # 找出开运算后矩阵中为255的部分，即物体范围
30     cv2.rectangle(frame, (min(rectangle[1]), min(rectangle[0])),
31                    (max(rectangle[1]), max(rectangle[0])),
32                    (0, 0, 255), 3) # 根据每一帧中物体的左上角坐标以及右下角坐标绘制矩形框
33     cv2.imshow('frame', hsv)
34     if cv2.waitKey(1) & 0xff == ord('q'):
35         break
36     else:
37         break
38
39 cap.release()
40 cv2.destroyAllWindows()
```