哈爾濱工業大學

Harbin Institute of Technology

视听觉信号处理
实验报告

实验三

课程名称：	视听觉信号处理
学院：	计算学部
专业：	计算机科学与技术
学号：	**1180300811**
姓名：	孙骁
指导老师：	姚鸿勋
学期：	**2020** 秋季学期

2020 年 **11** 月 **18** 日

目录

一、 实验目标和内容

1.1 实验目标

1. 综合运用图像处理的知识解决实际问题，以及可能出现的多种多样的情况。

1.2 实验内容

1. 对给定的静止状态下的一辆汽车图像进行车牌定位与检测，框出车牌保存结果图像，描述清楚整个算法流程；
2.（选做）在现实情况下，我们可能存在多种多样的情况。高速移动下的车牌；晚上夜景下的车牌；车牌的某些字符部分遮挡；图片中含有多张车牌；车牌倾斜情况。可以从中挑出一个感兴趣的去尝试进行定位（不仅限于上述情况）。

二、 实验环境

1. Anaconda 4.8.4
2. Python 3.7.4
3. PyCharm 2019.1 (Professional Edition)
4. Windows 10 2004

三、 实验原理

3.1 腐蚀

腐蚀是经典的形态学操作。假设 A 和 B 是 Z^2 中的两个集合，B 对 A 的腐蚀定义为

$$A \ominus B = \{z | (B)_z \subseteq A\}, \tag{1}$$

式中，A 是前景像素的一个集合，B 是一个结构元，z 项是前景像素值。公式 (1) 指出 B 对 A 的腐蚀是所有点 z 的集合，条件是平移后的 B 包含于 A。

腐蚀可以用于消除物体边界点，使目标缩小，可以消除小于结构元素的噪声点。

3.2 膨胀

膨胀是腐蚀的对偶运算。假设 A 和 B 是 Z^2 中的两个集合，B 对 A 的膨胀定义为

$$A \oplus B = \left\{z | (\hat{B})_z \cap A \neq \varnothing\right\}, \tag{2}$$

类似于腐蚀，式 (2) 是以 B 相对于其原点反射并将这一反射平移 z 为基础的。则 B 对 A 的膨胀就是所有位移 z 的集合，条件是 $\hat{B}$ 的前景元素与 A 的至少一个元素重叠。

膨胀可以用于将与物体接触的所有背景点合并到物体中，使目标增大，可添补目标中的空洞。

3.3 开运算与闭运算

结构元 B 对集合 A 的开运算定义为

$$A \circ B = (A \ominus B) \oplus B \tag{3}$$

因此，结构元 B 对集合 A 的开运算是：首先 B 对 A 腐蚀，接着 B 对腐蚀结果膨胀。

类似地，结构元 B 对集合 A 的闭运算定义为

$$A \bullet B = (A \oplus B) \ominus B \tag{4}$$

即，结构元 B 对集合 A 的闭运算是：首先 B 对 A 膨胀，接着 B 对膨胀结果腐蚀。

对开运算和闭运算的通俗解释为：B 对 A 的开运算是 B 的所有平移的并集，以便 B 完全拟合 A，条件是 B 完全拟合于 A，可以用公式写为

$$A \circ B = \cup \{(B)_z \,|\, (B)_z \subseteq A\} \tag{5}$$

式中，$\cup$ 表示大括号内所有集合的并集。

类似地，闭运算是 B 的所有不与 A 重叠的平移的并集的补集。所以可以把 B 对 A 的闭运算写为

$$A \bullet B = \left[\cup \left\{ (B)_z \,|\, (B)_z \cup A = \varnothing \right\} \right]^c \tag{6}$$

开运算用于平滑物体轮廓、断开狭窄的狭颈、消除细长的突出物；闭运算同样平滑轮廓，但是与开运算相反的是，闭运算弥合狭窄的断裂和细长的沟壑，消除小孔，填补轮廓中的缝隙。

3.4 顶帽变换

灰度级图像 $f(x,y)$ 的顶帽变换定义为 $f(x,y)$ 减去其开运算，即

$$T_{hat}(f(x,y)) = f(x,y) - (f(x,y) \circ B) \tag{7}$$

式中，B 为结构元，T_{hat} 表示顶帽运算的结果。

在车牌识别的场景中，特别针对黑夜的情况，会出现背景亮度不均匀的情况，而开运算可以用于补偿不均匀的背景亮度，所以用一个大的结构元素做开运算后。做顶帽变换，就得到了背景均衡的图像。

四、 实验步骤

实验中核心代码在附录可见，车牌检测流程中对图片的处理是一系列连续的操作，所以在一个类下完成。对车牌检测主要流程的流程图如图 (1) 所示。

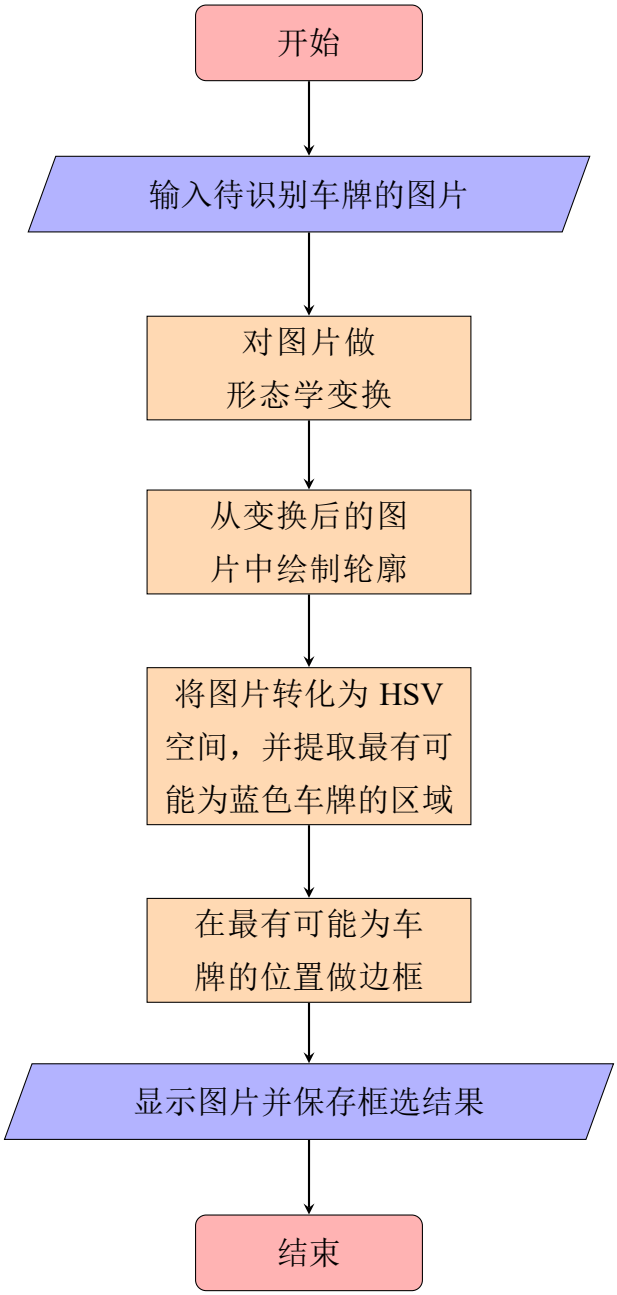


图 **1**　车牌检测算法的流程

下面对“对图片做形态学变换”的环节做详细说明。流程图如图 (2) 所示。

下面对各个操作做详细解释。

顶帽变换　顶帽变换的作用在章节3.4中做了说明，第一步使用顶帽变换可以消除照片光照不均匀的情况，特别是针对黑夜的情况，可以补偿不均匀的背景亮度，使用大的结

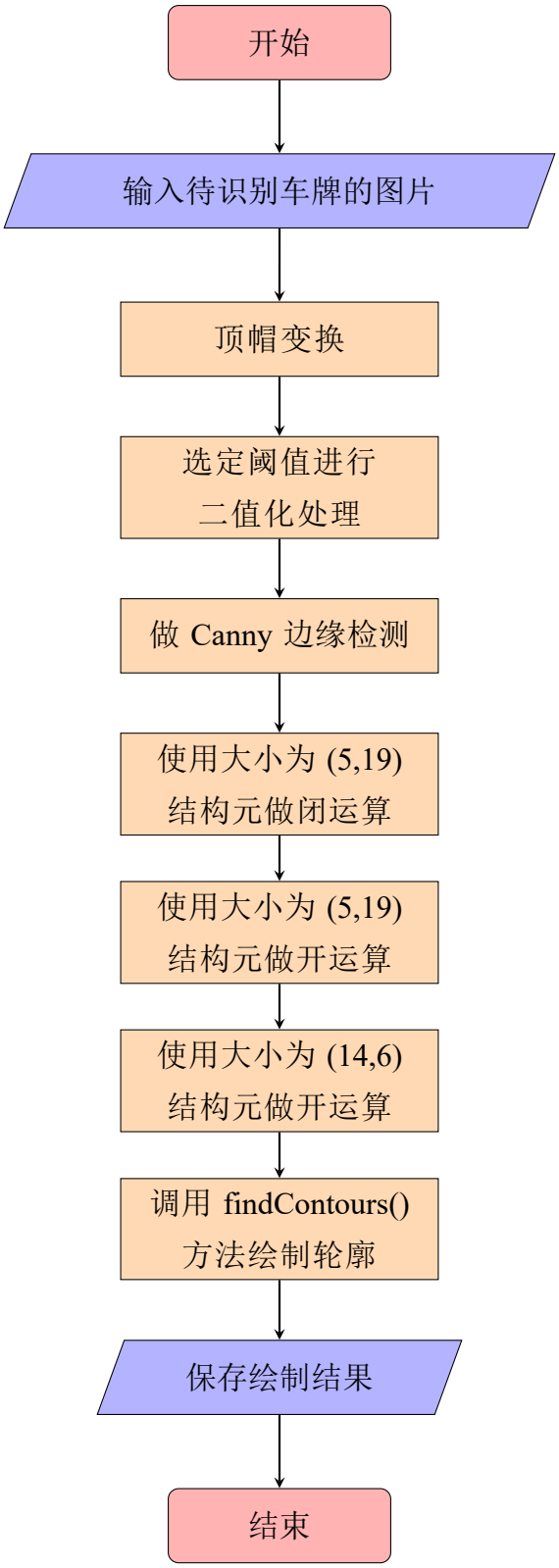


图 **2**　对图片做形态学变换的详细流程

构元做顶帽变换，可以得到背景均匀的图像。

选定阈值做二值化　我们需要根据顶帽变换后的灰度图提取车牌边缘，做二值化的目的是为了后面依据字符边缘提取边框时，可以较好的提取出车牌字符的边缘，可以便于下一步的操作。

Canny 边缘检测　Canny 边缘检测的目的是根据车牌字符边缘细节较多的特点进行边缘提取，进而便于分离出车牌的范围。

结构元为 (5,19) 的闭运算和开运算　选用“竖形”结构元对 Canny 边缘基恩侧的结果进行闭运算和开运算，为的是将提取出的边缘进行连接，连接成范围较大的区域，便于后续步骤对相应大小区域的特征提取。经过此步骤之后，可以将车牌的字符区域进行提取，大致的区分出车牌的位置。

结构元为 (14,6) 的开运算　选用与车牌形状相似的结构元素对依字符扩大的边缘进行再一次的形态学操作，可以将车牌的范围处理成矩形，既便于后续的画边框处理，又有将倾斜边框做归正的功能，即可以使用水平铅直的边框框出车牌范围。

五、 实验结果

实验选用了 7 张正常光照情况、3 张倾斜的照片、2 张黑夜照片做测试，原图如图 (3) 所示，框选的结果如图 (4) 所示。可单击数字进行跳转查看。

六、 实验结论

1. 腐蚀操作可以消除物体边界点，消除小于结构元的噪声；
2. 膨胀操作可以填补目标中的空洞，使目标增大；
3. 开运算可以平滑物体轮廓，消除平滑物体表面的“毛刺”，闭运算可以弥合狭窄的断裂，填补缝隙；
4. 顶帽变换可以补偿光照不均匀的背景亮度，从而对整幅图片进行光照均匀化处理；
5. 对于图像的形态学操作处理需要自行选用合适的参数进行处理，参数不同，得到的结果也会有很大的差异，有时甚至与预期结果相差甚远。

参考文献

[1] Rafael C. Gonzalez & Richard E. Woods (2020). Digital Image Processing (4th ed.).

附录 A　检测车牌的主算法–locate_license_plate.py

```python
import numpy as np
import cv2 as cv
from src.read_file_from_dic import *


class LocatePlate(object):
    def __init__(self, img):
        self.img = img
        self.open_img = None
        self.hat_img = None

    def binaryzation(self):
        maxi = float(self.hat_img.max())
        mini = float(self.hat_img.min())
        x = maxi - ((maxi - mini) / 2)
        _, thresh = cv.threshold(self.hat_img, x, 255, cv.THRESH_BINARY)
        return thresh

    @staticmethod
    def find_rectangle(contour):
        x = []
        y = []
        for p in contour:
            y.append(p[0][0])
            x.append(p[0][1])
        return [min(x), max(x), min(y), max(y)]

    def filter_img(self):
        mask = 400 * self.img.shape[0] / self.img.shape[1]
        self.img = cv.resize(self.img, (400, int(mask)), interpolation=cv.INTER_AREA)
        gray_img = cv.cvtColor(self.img, cv.COLOR_BGR2GRAY)
        radius = 16
        height = width = radius * 2 + 1
        kernel = np.zeros((height, width), np.uint8)
        cv.circle(kernel, (radius, radius), radius, 1, -1)
        # 开运算
        open_img = cv.morphologyEx(gray_img, cv.MORPH_OPEN, kernel)
        self.open_img = open_img
        # show_img("open img", self.open_img)
        # 顶帽变换
        hat_img = cv.absdiff(gray_img, self.open_img)
        # show_img("hat img", hat_img)
        self.hat_img = hat_img
        # 二值化
```

```
        binary_img = self.binaryzation()
        # Canny边缘检测
        canny = cv.Canny(binary_img, binary_img.shape[0], binary_img.shape[1])
        # show_img("canny", canny)
        kernel = np.ones((5, 19), np.uint8)
        # 闭运算
        close_img = cv.morphologyEx(canny, cv.MORPH_CLOSE, kernel)
        # show_img("close img", close_img)

        # 开运算
        open_img = cv.morphologyEx(close_img, cv.MORPH_OPEN, kernel)
        # show_img("open img", open_img)

        # 开运算
        kernel = np.ones((14, 6), np.uint8)
        open_img = cv.morphologyEx(open_img, cv.MORPH_OPEN, kernel)
        self.open_img = open_img
        # show_img("open img", self.open_img)

    def detect_edge(self):
        contours, _ = cv.findContours(self.open_img, cv.RETR_EXTERNAL,
            cv.CHAIN_APPROX_SIMPLE)
        block = []
        for c in contours:
            r = self.find_rectangle(c)
            block.append(r)
        max_weight = 0
        max_index = -1
        for i in range(len(block)):
            b = self.img[block[i][0]: block[i][1], block[i][2]: block[i][3]]
            # 转化为hsv颜色空间
            hsv_img = cv.cvtColor(b, cv.COLOR_BGR2HSV)
            # 蓝色下界
            lower = np.array([100, 50, 50])
            # 蓝色上界
            upper = np.array([140, 255, 255])
            mask = cv.inRange(hsv_img, lower, upper)
            w1 = 0
            for m in mask:
                w1 += m / 255
            w2 = 0
            for n in w1:
                w2 += n
            if w2 > max_weight:
                max_index = i
                max_weight = w2
        # print(block)
```

```
        # print(max_index)
        if max_index == -1:
            return self.img
        rect = block[max_index]
        img = self.img.copy()
        cv.rectangle(img, (rect[3], rect[1]), (rect[2], rect[0]), (0, 0, 255), 2)
        # show_img("img", img)
        return img

    def locate_plate_main(self):
        self.filter_img()
        img = self.detect_edge()
        return img


def main():
    img_list = read_file_from_dic()
    for i in range(img_list.size):
        # show_img("i", img_list[i])
        print("detect img " + str(i + 1))
        locate = LocatePlate(img_list[i])
        img = locate.locate_plate_main()
        write_img_to_file(str(i + 1), img)


if __name__ == '__main__':
    main()
```

附录 B　从系统中读取图片并显示保存结果–read_file_from_dic.py

```
import os
import numpy as np
import cv2 as cv


def read_file_from_dic():
    file_dir_path = '../data/'
    pic_list = []
    file_list = os.listdir(file_dir_path)
    for file in file_list:
        file_path = os.path.join(file_dir_path, file)
        print("open figure " + file_path)
        with open(file_path) as f:
            img = cv.imread(file_path)
```

```
        pic_list.append(img)
    return np.asarray(pic_list)


def show_img(title, img):
    cv.namedWindow(title, cv.WINDOW_FREERATIO)
    cv.imshow(title, img)
    cv.waitKey()
    cv.destroyAllWindows()


def write_img_to_file(file_name, img):
    file_path = '../result/' + file_name + '.jpg'
    cv.imwrite(file_path, img)
```

(a) 正常光照图片 1

(b) 正常光照图片 2

(c) 正常光照图片 3

(d) 正常光照图片 4

(e) 正常光照图片 5

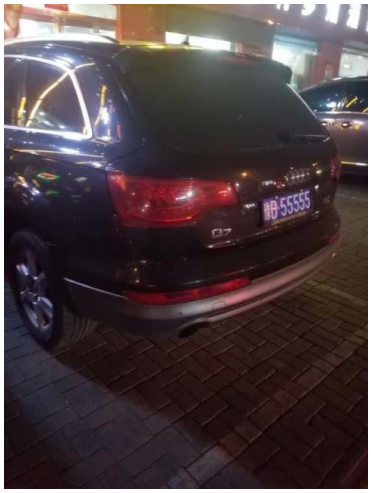
(f) 正常光照图片 6

(g) 正常光照图片 7

(h) 倾斜车牌图片 1

(i) 倾斜车牌图片 2

(j) 倾斜车牌图片 3



(k) 黑暗夜景图片 1

(l) 黑暗夜景图片 2

图 **3**　待检测车牌图片

(a) 正常光照图片 1 结果

(b) 正常光照图片 2 结果

(c) 正常光照图片 3 结果

(d) 正常光照图片 4 结果

(e) 正常光照图片 5 结果

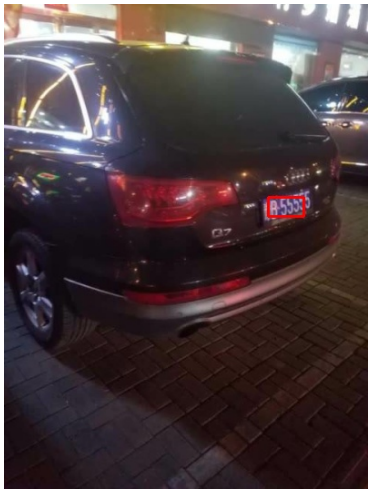
(f) 正常光照图片 6 结果

(g) 正常光照图片 7 结果

(h) 倾斜车牌图片 1 结果

(i) 倾斜车牌图片 2 结果

(j) 倾斜车牌图片 3 结果



(k) 黑暗夜景图片 1 结果

(l) 黑暗夜景图片 2 结果

图 **4**　车牌检测的结果