

opencv

单个任务

配置环境

颜色识别

去噪

边缘检测

模板匹配

霍夫线检测、圆检测

霍夫线检测

霍夫圆检测

角点

harris角点检测

shi-tomas角点检测

SIFT特征点匹配

视频读取

meanshift、camshift自动追踪运动物体

实战项目

银行卡号识别——形态学提取&模板匹配

文本扫描OCR识别——边缘检测&坐标点透射变换

全景拼接——描述子匹配

人像抠图——背景掩膜

往届竞赛项目

红圈识别——颜色阈值分割&霍夫圆检测

巡检机器人：二维码识别——调包&透射变换

巡线机器人：异物条形码识别——形态学提取条形码

巡线机器人：飞机起降点检测——模板匹配

植保飞行器：飞机起降点检测——模板匹配

空地协同消防系统：火源识别——颜色掩膜

【黑马程序员人工智能教程_10小时学会图像处理OpenCV入门教程】

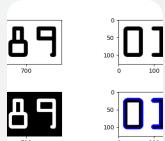


黑马程序员人工智能教程_10小时学会图像处理OpenCV入门教程_哔哩哔哩_bilibili

黑马程序员人工智能教程_10小时学会图像处理OpenCV入门教程共计67条视频，包括：01.课程介绍、...

https://www.bilibili.com/video/BV1Fo4y1d7JL?vd_source=47dedd4e022eac07e87bc4dc59e42a00

实战项目



Opencv图像处理（全） -CSDN博客

文章浏览阅读5.6w次，点赞310次，收藏2.5k次。备注：以下源码均可运行，不同项目涉及的函数均有...

<https://blog.csdn.net/shinuone/article/details/126022763>

单个任务

配置环境

ubuntu18.04

Bash |

```
1 conda create --name cv python=3.8
2 conda activate cv
3 pip install opencv-python==4.9.0.80
4 pip install matplotlib
5 pip install opencv-contrib-python==4.9.0.80
```

颜色识别

	黑	灰	白	红	橙	黄	绿	青	蓝	紫	
hmin	0	0	0	0	156	11	26	35	78	100	125
hmax	180	180	180	10	180	25	34;	77	99	124	155
smin	0	0	0	43	43	43	43	43	43	43	
smax	255	43	30	255	255	255	255	255	255	255	
vmin	0	46	221	46	46	46	46	46	46	46	
vmax	46	220	255	255	255	255	255	255	255	255	

去噪

均值滤波 (Mean Filter) :

cv2.blur 或 cv2.boxFilter

简介：将图像中每个像素周围的像素值取平均，用于平滑图像和去除噪声。

高斯滤波 (Gaussian Filter) :

cv2.GaussianBlur

简介：应用高斯函数对图像进行模糊处理，可有效去除高斯噪声，保留图像的整体细节。

中值滤波 (Median Filter) :

cv2.medianBlur

简介：对图像中每个像素周围的像素值进行排序，然后取中间值作为该像素的值，适用于去除椒盐噪声。

边缘检测

算子	优缺点比较
Roberts	对具有陡峭的低噪声的图像处理效果较好，但利用 Roberts 算子提取边缘的结果是边缘比较粗，因此边缘定位不是很准确。
Sobel	对灰度渐变和噪声较多的图像处理效果比较好，Sobel 算子对边缘定位比较准确。
Kirsch	对灰度渐变和噪声较多的图像处理效果较好。
Prewitt	对灰度渐变和噪声较多的图像处理效果较好。
Laplacian	对图像中的阶跃性边缘点定位准确，对噪声非常敏感，丢失一部分边缘的方向信息，造成一些不连续的检测边缘。
LoG	LoG 算子经常出现双边缘像素边界，而且该检测方法对噪声比较敏感，所以很少用 LoG 算子检测边缘，而是用来判断边缘像素是位于图像的明区还是暗区。
Canny	此方法不容易受噪声的干扰，能够检测到真正的弱边缘。在 edge 函数中，最有效的边缘检测方法是 Canny 方法。该方法的优点在于使用两种不同的阈值分别检测强边缘和弱边缘，并且仅当弱边缘与强边缘相连时，才将弱边缘包含在输出图像中。因此，这种方法不容易被噪声“填充”，更容易检测出真正的弱边缘。

模板匹配

```
res =cv.matchTemplate(img,template,method)
```

参数:

img: 要进行模板匹配的图像

Template: 模板

method: 实现模板匹配的算法, 主要有:

1. 平方差匹配(CV_TM_SQDIFF):利用模板与图像之间的平方差进行匹配, 最好的匹配是0, 匹配越差, 匹配的值越大。

2. 相关匹配(CV_TM_CCORR):利用模板与图像间的乘法进行匹配, 数值越大表示匹配程度较高, 越小表示匹配效果差。

3. 利用相关系数匹配(CV_TM_CCOEFF):利用模板与图像间的相关系数匹配, 1表示完美的匹配, -1表示最差的匹配。

完成匹配后, 使用cv.minMaxLoc()方法查找最大值所在的位置即可。如果使用平方差作为比较方法, 则最小值位置是最佳匹配位置。

```
1 import cv2
2 import numpy as np
3
4 # 读取输入图像和模板图像
5 input_image = cv2.imread('input_image.jpg')
6 template = cv2.imread('template_image.jpg')
7
8 # 获取模板图像的宽度和高度
9 template_height, template_width = template.shape[:2]
10
11 # 选择匹配方法, 这里使用 TM_CCOEFF_NORMED 方法
12 match_method = cv2.TM_CCOEFF_NORMED
13
14 # 使用模板匹配方法
15 result = cv2.matchTemplate(input_image, template, match_method)
16
17 # 获取匹配结果的最小值、最大值, 以及它们的位置
18 min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
19
20 # 如果使用的方法是 TM_SQDIFF 或 TM_SQDIFF_NORMED, 则取最小值位置, 否则取最大值位置
21 if match_method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
22     top_left = min_loc
23 else:
24     top_left = max_loc
25
26 # 确定匹配到的区域的右下角位置
27 bottom_right = (top_left[0] + template_width, top_left[1] + template_height)
28
29 # 在原始图像上绘制矩形框
30 cv2.rectangle(input_image, top_left, bottom_right, (0, 255, 0), 2)
31
32 # 显示结果图像
33 cv2.imshow('Template Matching Result', input_image)
34 cv2.waitKey(0)
35 cv2.destroyAllWindows()
36
```

霍夫线检测、圆检测

霍夫线检测

```
cv.HoughLines(img,rho,theta,threshold)
```

参数：

img：检测的图像，要求是二值化的图像，所以在调用霍夫变换之前首先要进行二值化，或者进行Canny边缘检测

rho、theta： ρ 和 θ 的精确度

threshold：阈值，只有累加器中的值高于该阈值时才被认为是直线

```
▼ 霍夫线检测 Python |  
1 import cv2  
2 import numpy as np  
3  
4 # 读取图像  
5 image = cv2.imread('road.jpg')  
6 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  
7 edges = cv2.Canny(gray, 50, 150, apertureSize=3)  
8  
9 # 进行霍夫线变换  
10 lines = cv2.HoughLines(edges, 1, np.pi / 180, 150, threshold=100)  
11  
12 # 绘制检测到的线  
13 if lines is not None:  
14     for rho, theta in lines[:, 0]:  
15         a = np.cos(theta)  
16         b = np.sin(theta)  
17         x0 = a * rho  
18         y0 = b * rho  
19         x1 = int(x0 + 1000 * (-b))  
20         y1 = int(y0 + 1000 * (a))  
21         x2 = int(x0 - 1000 * (-b))  
22         y2 = int(y0 - 1000 * (a))  
23         cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)  
24  
25 # 显示结果  
26 cv2.imshow('Hough Lines Detection', image)  
27 cv2.waitKey(0)  
28 cv2.destroyAllWindows()  
29
```

霍夫圆检测

```
cv.Houghcircles(image, method, dp, minDist, param1=100, param2=100,  
minRadius=0,maxRadius=0 )
```

参数：

image：输入图像，应输入灰度图像

method：使用霍夫变换圆检测的算法，它的参数是CVHOUGH GRADIENT

dp：霍夫空间的分辨率，dp=1时表示霍夫空间与输入图像空间的大小一致，dp=2时霍夫空间是输入图像空间的一半，以此类推

minDist：圆心之间的最小距离，如果检测到的两个圆心之间距离小于该值，则认为它们是同一个圆

param1：边缘检测时使用Canny算子的高值，低值是高值的一半

param2：检测圆心和确定半径时所共有的值

minRadius和maxRadius：所检测到的圆半径的最小值和最大值

返回：

circles：输出圆向量，包括三个浮点型的元素--圆心横坐标，圆心纵坐标和圆半径

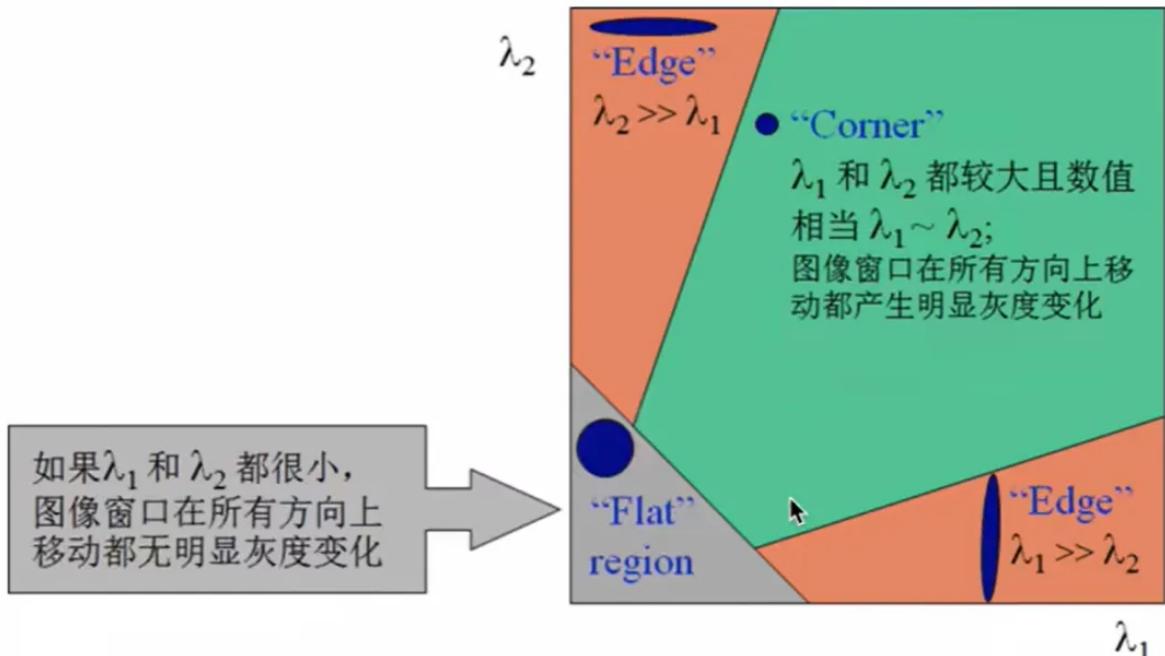
```
1 import cv2
2 import numpy as np
3
4 # 读取图像
5 image = cv2.imread('road.jpg', cv2.IMREAD_COLOR)
6 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
7 gray_blurred = cv2.medianBlur(gray, 5)
8
9 # 霍夫圆检测
10 circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, dp=1, minDist
=20, param1=50, param2=30, minRadius=0, maxRadius=0)
11
12 # 绘制检测到的圆
13 if circles is not None:
14     circles = np.uint16(np.around(circles))
15     for i in circles[0, :]:
16         center = (i[0], i[1])
17         radius = i[2]
18         cv2.circle(image, center, radius, (0, 255, 0), 2)
19
20 # 显示结果
21 cv2.imshow('Hough Circles Detection', image)
22 cv2.waitKey(0)
23 cv2.destroyAllWindows()
24
```

角点

- 灵敏度：Harris角点检测可能对灰度变化更敏感，Tomasi角点检测相对保守一些。
- 方向信息：Tomasi角点检测考虑了角点的方向信息，能够更准确地判断角点。

harris角点检测

椭圆函数特征值与图像中的角点、直线（边缘）和平面之间的关系如下图所示。



共可分为三种情况：

- 图像中的直线。一个特征值大，另一个特征值小， $\lambda_1 >> \lambda_2$ 或 $\lambda_2 >> \lambda_1$ 。椭圆函数值在某一方向上大，在其他方向上小。
- 图像中的平面。两个特征值都小，且近似相等；椭圆函数数值在各个方向上都小。
- 图像中的角点。两个特征值都大，且近似相等，椭圆函数在所有方向都增大

```
dst=cv.cornerHarris(src, blockSize, ksize, k)
```

参数：

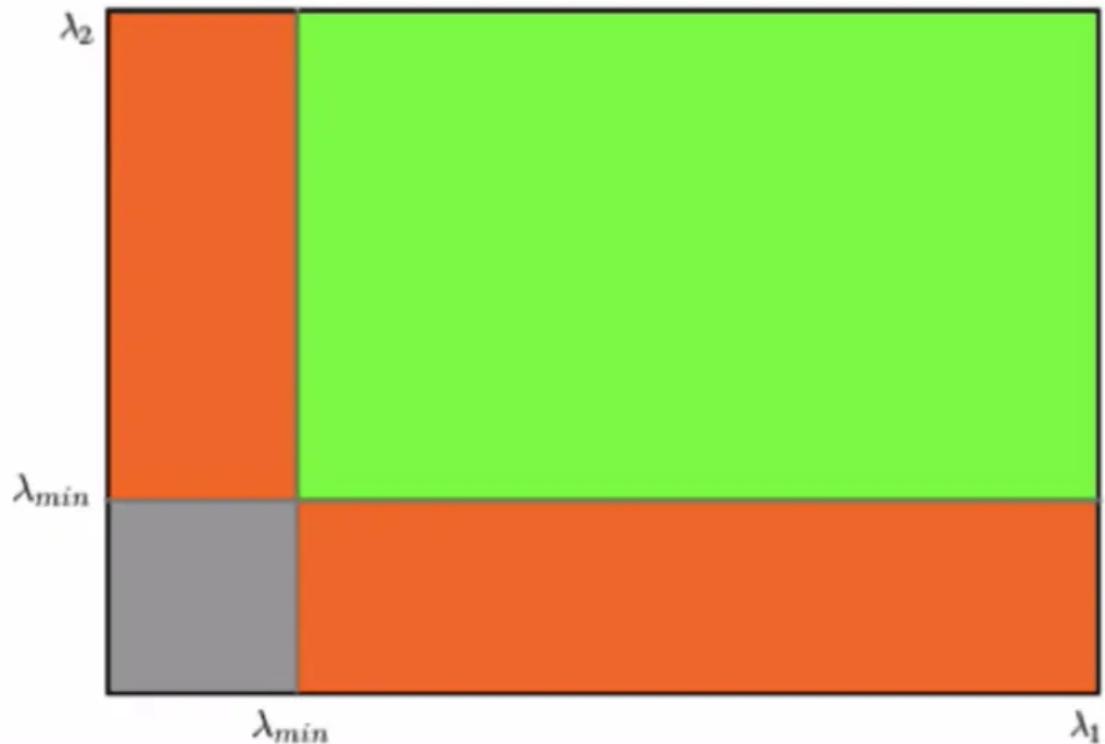
img：数据类型为 float32 的输入图像

blockSize：角点检测中要考虑的邻域大小

ksize：sobel求导使用的核大小

k：角点检测方程中的自由参数，取值参数为[0.04, 0.06]

shi-tomas角点检测



```
corners =cv2.goodFeaturesToTrack( image, makcorners, qualityLevel, minDistance )
```

参数：

Image：输入灰度图像

maxCorners：获取角点数的数目。

qualityLevel：该参数指出最低可接受的角点质量水平，在0–1之间

minDistance：角点之间最小的欧式距离，避免得到相邻特征点。

SIFT特征点匹配

```
1 import cv2
2 import numpy as np
3
4 # 读取两张输入图像
5 img1 = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)
6 img2 = cv2.imread('image2.jpg', cv2.IMREAD_GRAYSCALE)
7
8 # 初始化SIFT检测器
9 sift = cv2.xfeature2d.SIFT_create()
10
11 # 在两个图像上检测关键点和计算描述子
12 keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
13 keypoints2, descriptors2 = sift.detectAndCompute(img2, None)
14
15 # 创建一个暴力匹配器对象
16 bf = cv2.BFMatcher()
17
18 # 使用KNN匹配器找到最佳匹配
19 matches = bf.knnMatch(descriptors1, descriptors2, k=2)
20
21 # 应用比率测试，以筛选出最好的匹配
22 good_matches = []
23 for m, n in matches:
24     if m.distance < 0.75 * n.distance:
25         good_matches.append(m)
26
27 # 绘制匹配结果
28 img_matches = cv2.drawMatches(img1, keypoints1, img2, keypoints2, good_matches, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
29
30 # 显示结果
31 cv2.imshow('Matches', img_matches)
32 cv2.waitKey(0)
33 cv2.destroyAllWindows()
34
```

视频读取

- propId: 从0到18的数字，每个数字表示视频的属性

常用属性有：

索引	flags	意义
0	cv2.CAP_PROP_POS_MSEC	视频文件的当前位置 (ms)
1	cv2.CAP_PROP_POS_FRAMES	从0开始索引帧，帧位置
2	cv2.CAP_PROP_POS_AVI_RATIO	视频文件的相对位置 (0表示开始, 1表示结束)
3	cv2.CAP_PROP_FRAME_WIDTH	视频流的帧宽度
4	cv2.CAP_PROP_FRAME_HEIGHT	视频流的帧高度
5	cv2.CAP_PROP_FPS	帧率
6	cv2.CAP_PROP_FOURCC	编解码器四字符代码
7	cv2.CAP_PROP_FRAME_COUNT	视频文件的帧

1. 读取视频

读取视频: cap=cv.VideoCapture()

判断读取成功: cap.isOpened()

读取每一帧图像: ret,frame=cap.read()

获取属性: cap.get(propId)

设置属性: cap.set(propId,value)

资源释放: cap.release()

2. 保存视频

保存视频: out=cv.VideoWriter()

视频写入: out.write()

资源释放: out.release()

meanshift、camshift自动追踪运动物体

1. meanshift

原理：一个迭代的步骤，即先算出当前点的偏移均值，移动该点到其偏移均值，然后以此为新的起始点，继续移动，直到满足一定的条件结束。

API: cv.meanshift()

优缺点：简单，迭代次数少，但无法解决目标的遮挡问题并且不能适应运动目标的形状和大小变化

2.camshift

原理：对meanshift算法的改进，首先应用meanshit，一旦meanshift收敛，它就更新窗口的大小，还计算最佳拟合椭圆的方向，从而根据目标的位置和大小更新搜索窗口。

API：cv.camshift()

优缺点：可适应运动目标的大小形状的改变，具有较好的跟踪效果，但当背景色和目标颜色接近时，容易使目标的区域变大，最终有可能导致目标跟踪丢失

meanshift

Python |

```
1 import cv2
2
3 # 读取视频文件
4 cap = cv2.VideoCapture('video.mp4')
5
6 # 设置初始窗口位置 (左上角点、宽度、高度)
7 ret, frame = cap.read()
8 x, y, w, h = 300, 200, 100, 50 # 初始跟踪窗口位置
9
10 # 设置初始跟踪窗口
11 track_window = (x, y, w, h)
12
13 # 设置追踪算法的参数
14 term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)
15
16 while True:
17     ret, frame = cap.read()
18     if not ret:
19         break
20
21     # 在当前帧中定位ROI区域 (Region of Interest)
22     roi = frame[y:y+h, x:x+w]
23
24     # 将ROI区域转换为HSV颜色空间
25     hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
26
27     # 计算ROI区域的直方图
28     roi_hist = cv2.calcHist([hsv_roi], [0], None, [180], [0, 180])
29
30     # 归一化直方图
31     roi_hist = cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
32
33     # 设置MeanShift追踪器
34     dst = cv2.calcBackProject([cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)], [0],
35                               roi_hist, [0, 180], 1)
36
37     # 应用MeanShift算法进行目标追踪
38     ret, track_window = cv2.meanShift(dst, track_window, term_crit)
39
40     # 绘制跟踪窗口
41     x, y, w, h = track_window
42     img2 = cv2.rectangle(frame, (x, y), (x+w, y+h), 255, 2)
43     cv2.imshow('img2', img2)
44
45     # 按下ESC键退出
```

```
45     k = cv2.waitKey(30) & 0xff
46     if k == 27:
47         break
48
49 cap.release()
50 cv2.destroyAllWindows()
51
```

```
1 import cv2
2 import numpy as np
3
4 # 读取视频文件
5 cap = cv2.VideoCapture('video.mp4')
6
7 # 设置初始窗口位置 (左上角点、宽度、高度)
8 ret, frame = cap.read()
9 x, y, w, h = 300, 200, 100, 50 # 初始跟踪窗口位置
10
11 # 设置初始跟踪窗口
12 track_window = (x, y, w, h)
13
14 # 设置追踪算法的参数
15 term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)
16
17 while True:
18     ret, frame = cap.read()
19     if not ret:
20         break
21
22     # 在当前帧中定位ROI区域 (Region of Interest)
23     roi = frame[y:y+h, x:x+w]
24
25     # 将ROI区域转换为HSV颜色空间
26     hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
27
28     # 计算ROI区域的直方图
29     roi_hist = cv2.calcHist([hsv_roi], [0], None, [180], [0, 180])
30
31     # 归一化直方图
32     roi_hist = cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
33
34     # 设置CamShift追踪器
35     dst = cv2.calcBackProject([cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)], [0],
36                               roi_hist, [0, 180], 1)
37
38     # 应用CamShift算法进行目标追踪
39     ret, track_window = cv2.CamShift(dst, track_window, term_crit)
40
41     pts = cv2.boxPoints(ret)
42     pts = np.int0(pts)
43
44     # 绘制追踪结果
```

```

45     img2 = cv2.polyline(frame, [pts], True, 255, 2)
46     cv2.imshow('img2', img2)
47
48     # 按下ESC键退出
49     k = cv2.waitKey(30) & 0xff
50     if k == 27:
51         break
52
53 cap.release()
54 cv2.destroyAllWindows()
55

```

实战项目

环境配置

Bash |

```

1 conda create opencv45
2 conda activate opencv45
3 conda install python=3.9
4 pip install opencv-python==4.5.1.18
5 pip install opencv-contrib-python==4.5.1.18
6 pip install numpy==1.23

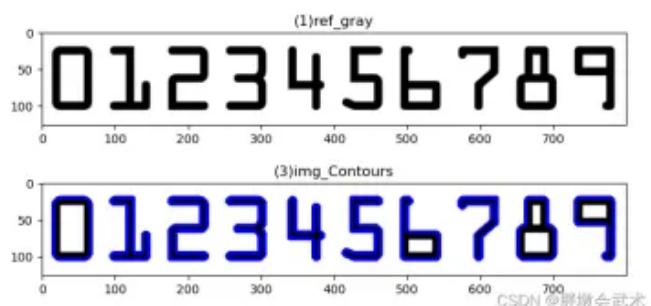
```

C OpenCV图像处理（全） -CSDN博客 中的代码可以直接使用。

银行卡号识别——形态学提取&模板匹配

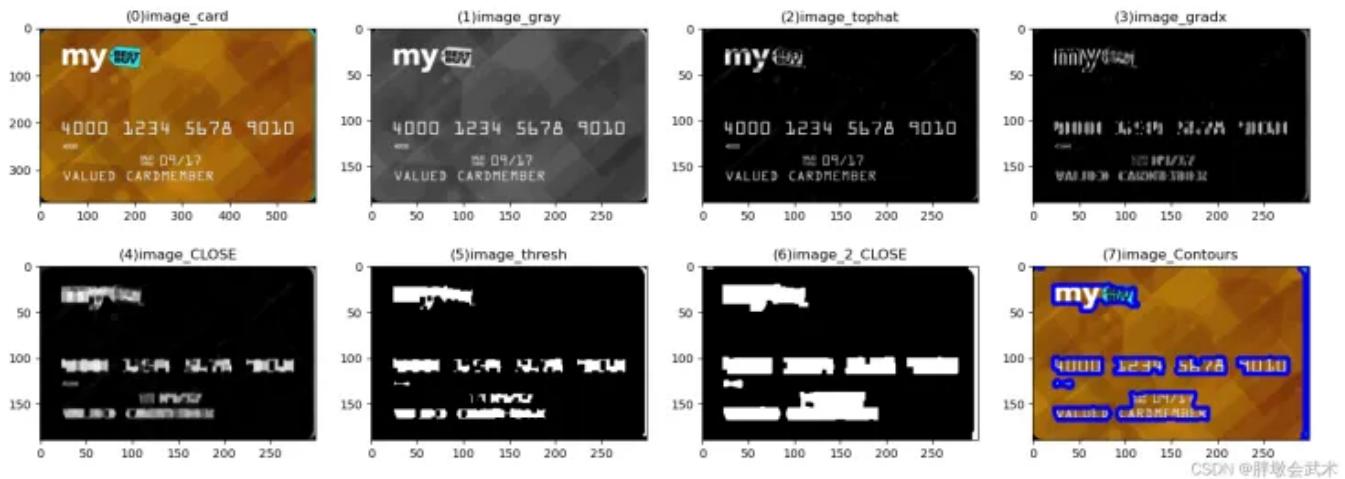
模板提取：

灰度图→二值化→轮廓提取→排序



银行卡数字提取：

读取卡图像、灰度图、顶帽运算、sobel算子、闭运算、二值化、二次膨胀+腐蚀；边缘检测

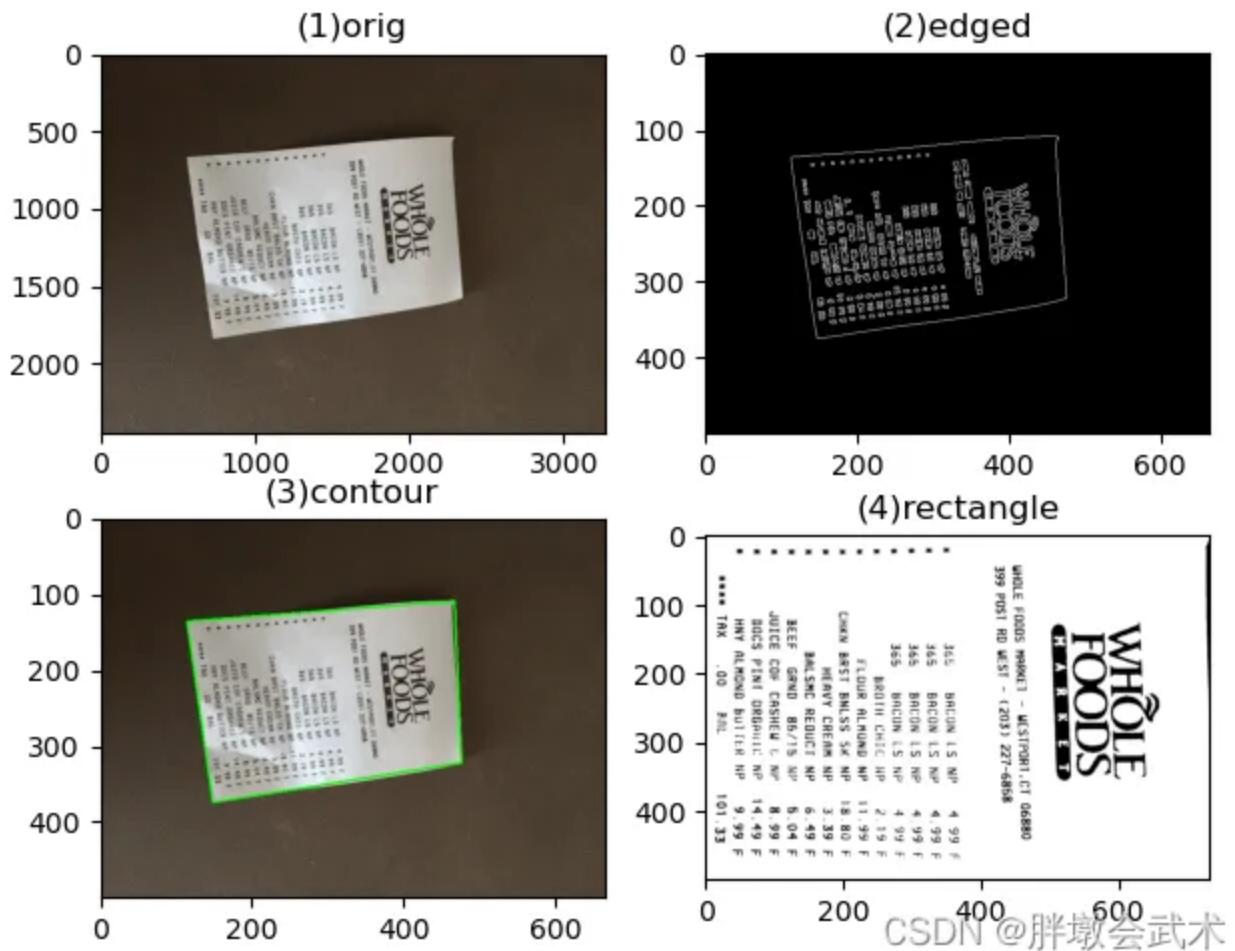


模板匹配和识别：

数字模板与每个银行卡数字进行模板匹配

文本扫描OCR识别——边缘检测&坐标点透射变换

边缘检测，找到方形曲线，透射变换

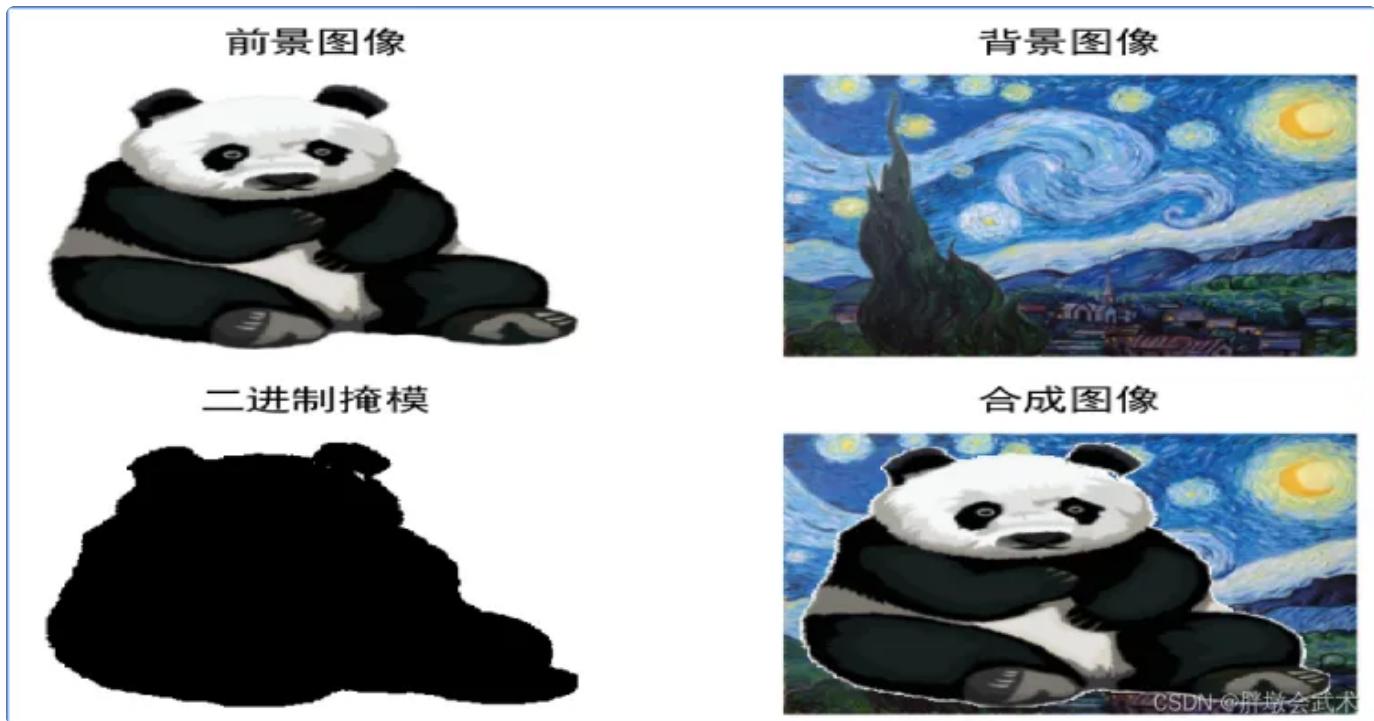


全景拼接——描述子匹配

SIFT特征点检测，匹配特征描述子，透射变换，拼接

人像抠图——背景掩膜

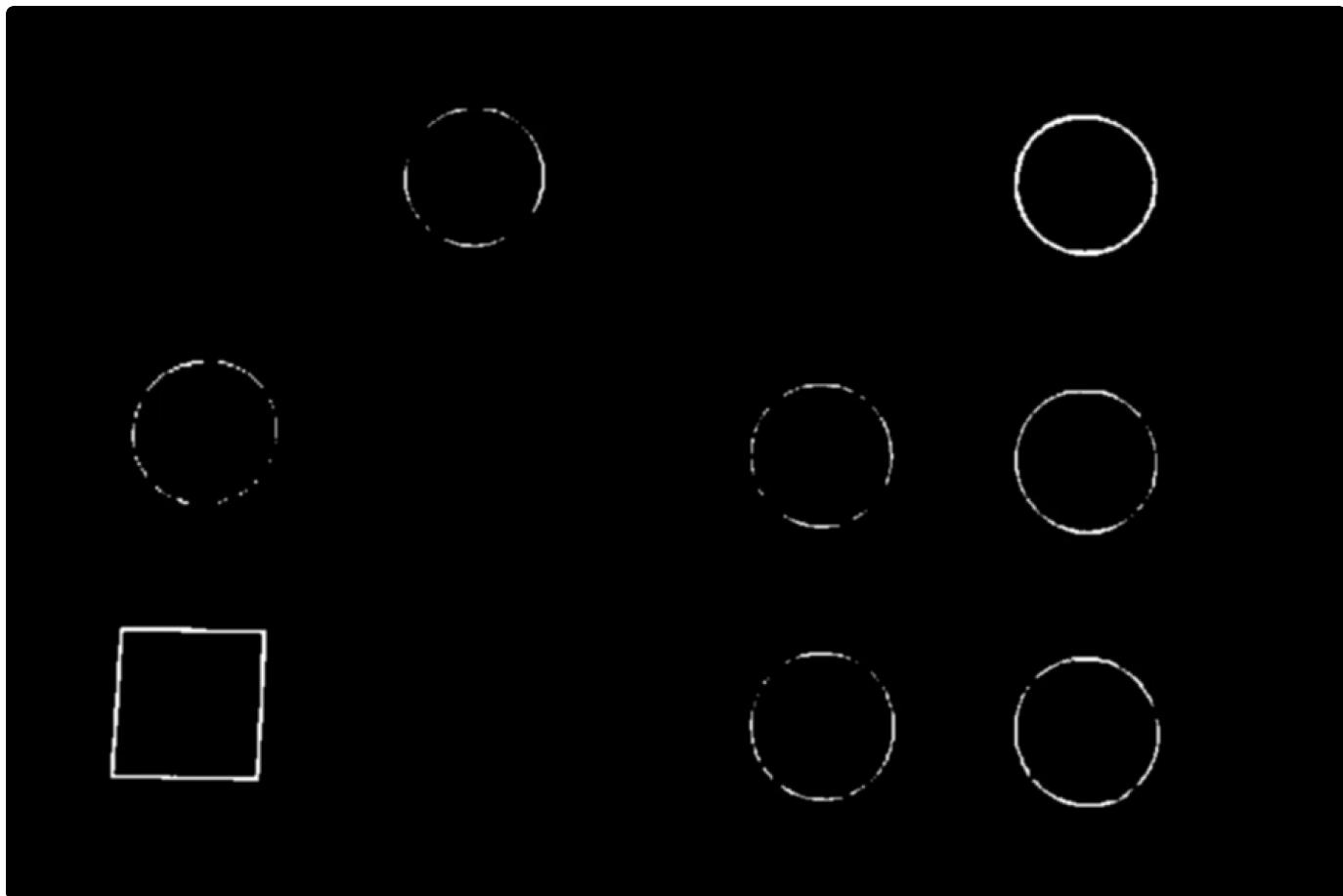
前景提取，二进制掩膜，合成



往届电赛项目

红圈识别——颜色阈值分割&霍夫圆检测

HSV阈值分割，形态学操作填补孔洞，霍夫圆检测圆心

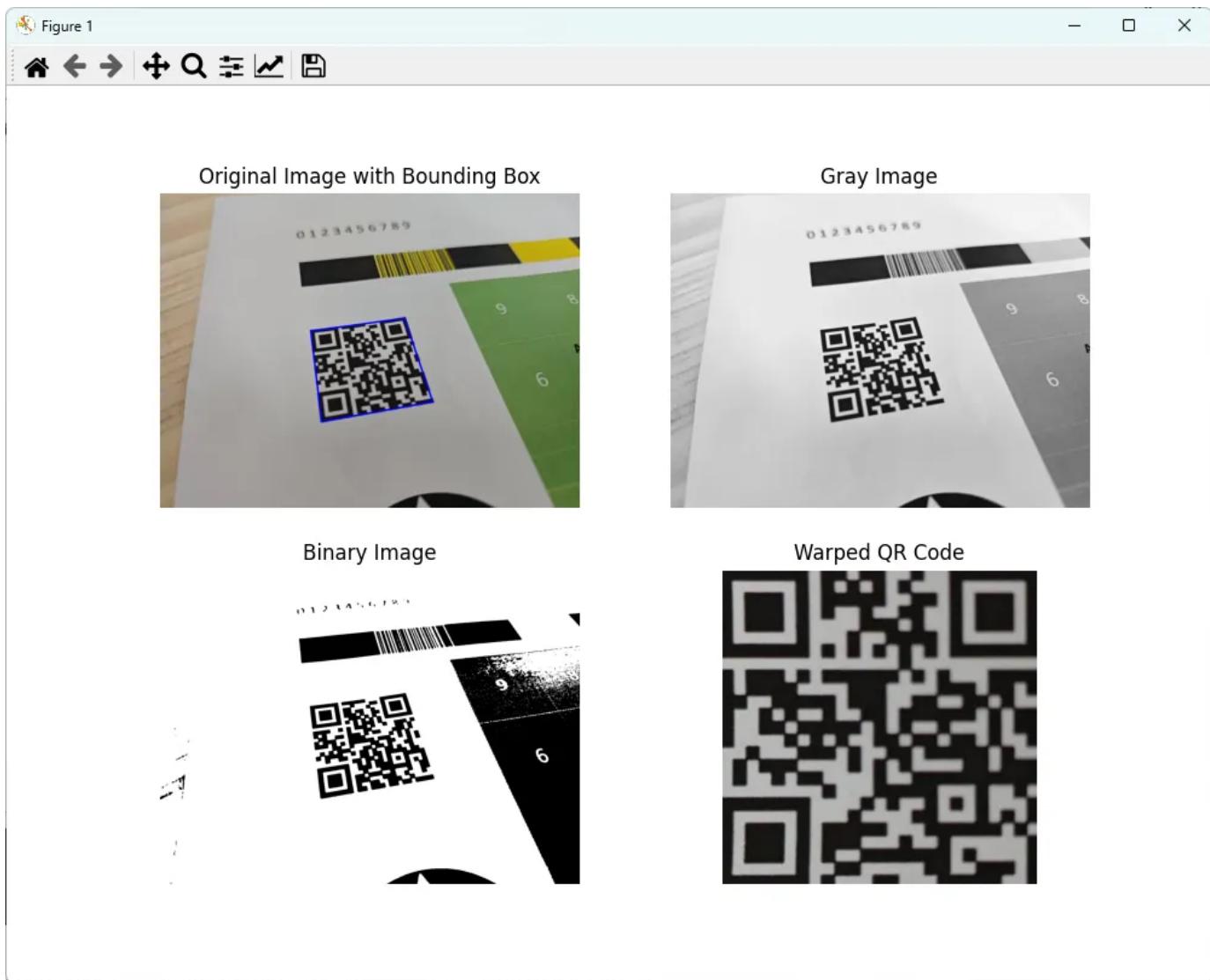


```
Detected circle at: (932, 650), radius: 74
Detected circle at: (630, 662), radius: 80
Detected circle at: (346, 668), radius: 86
Detected circle at: (934, 656), radius: 74
Detected circle at: (634, 672), radius: 75
Detected circle at: (926, 666), radius: 73
Detected circle at: (634, 402), radius: 69
Detected circle at: (358, 684), radius: 85
Detected circle at: (634, 676), radius: 77
Detected circle at: (930, 666), radius: 70
Detected circle at: (638, 408), radius: 64
Detected circle at: (358, 684), radius: 88
Detected circle at: (926, 676), radius: 74
Detected circle at: (636, 418), radius: 67
Detected circle at: (356, 674), radius: 71
Detected circle at: (642, 680), radius: 72
Detected circle at: (640, 684), radius: 73
Detected circle at: (926, 686), radius: 75
Detected circle at: (358, 690), radius: 74
Detected circle at: (368, 712), radius: 79
Detected circle at: (634, 706), radius: 78
```



巡检机器人：二维码识别——调包&透射变换

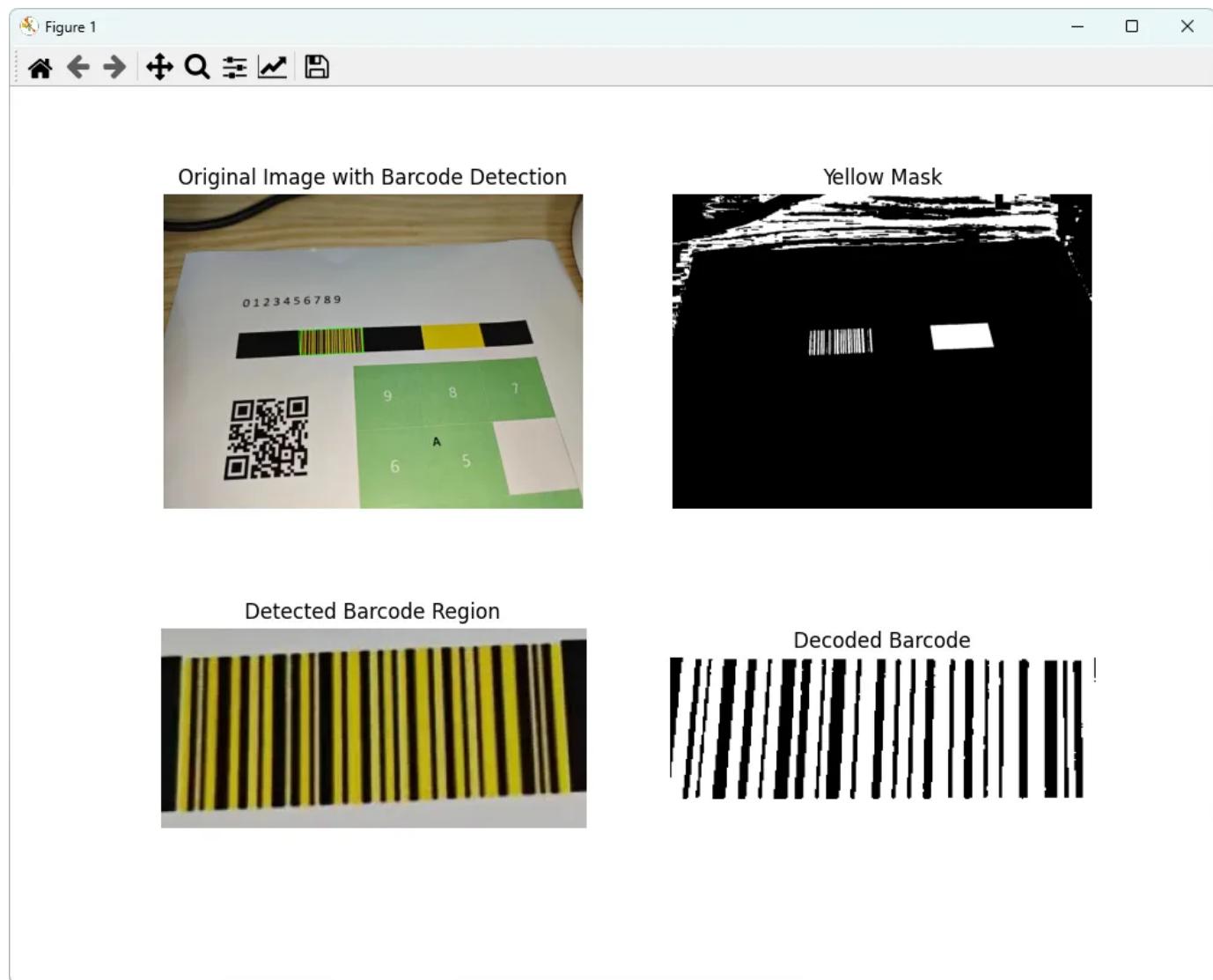
```
from pyzbar.pyzbar import decode
```



```
D:\wendy\study\others\cv>python qrcode.py  
二维码内容： 2019大学生电子设计竞赛
```

巡线机器人：异物条形码识别——形态学提取条形码

闭运算的差值找到条形码，透射变换，掩膜分割颜色，调包解析



```
D:\wendy\study\others\cv>python barcode.py
条形码类型：CODE128
条形码数据：b'20190807'
```

巡线机器人：飞机起降点检测——模板匹配

掩膜分割，多模板匹配，跳帧匹配

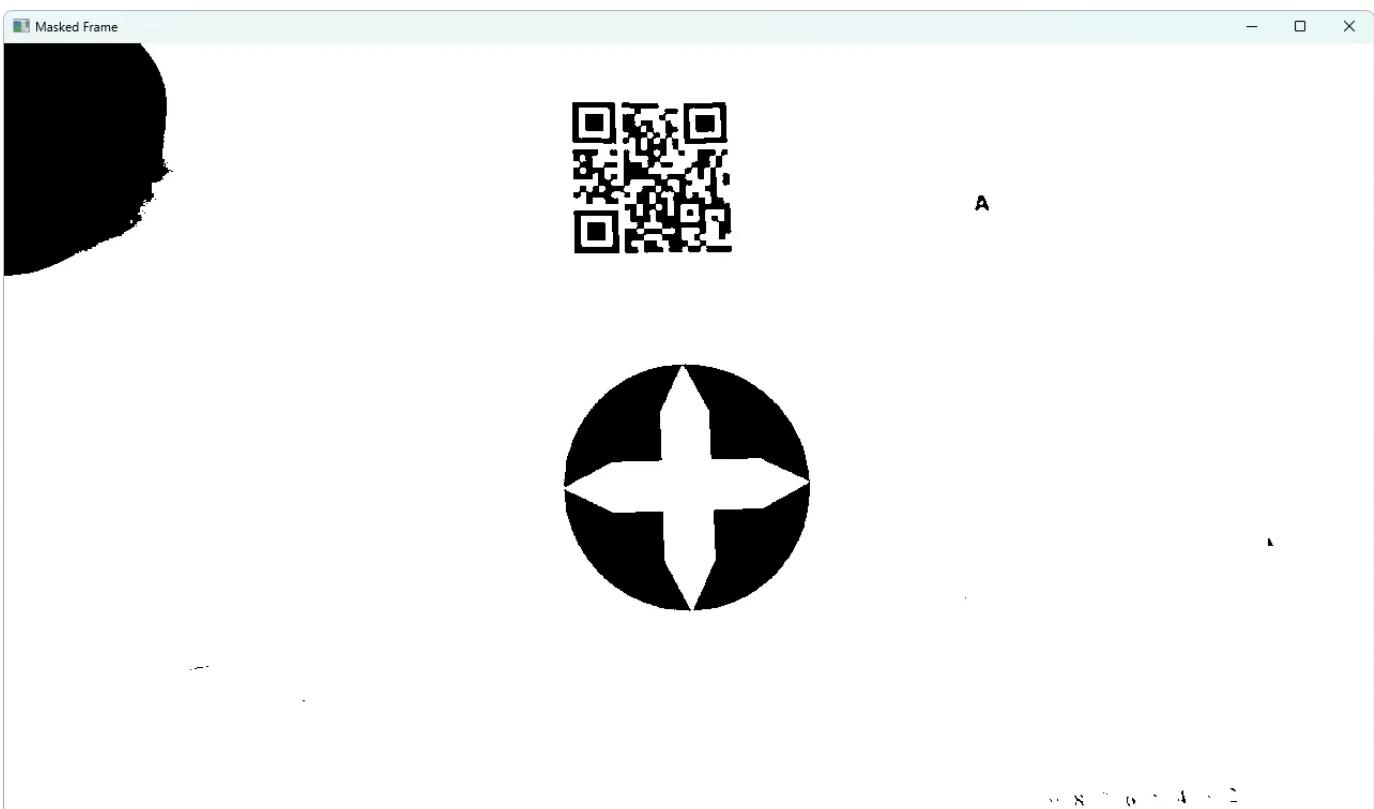
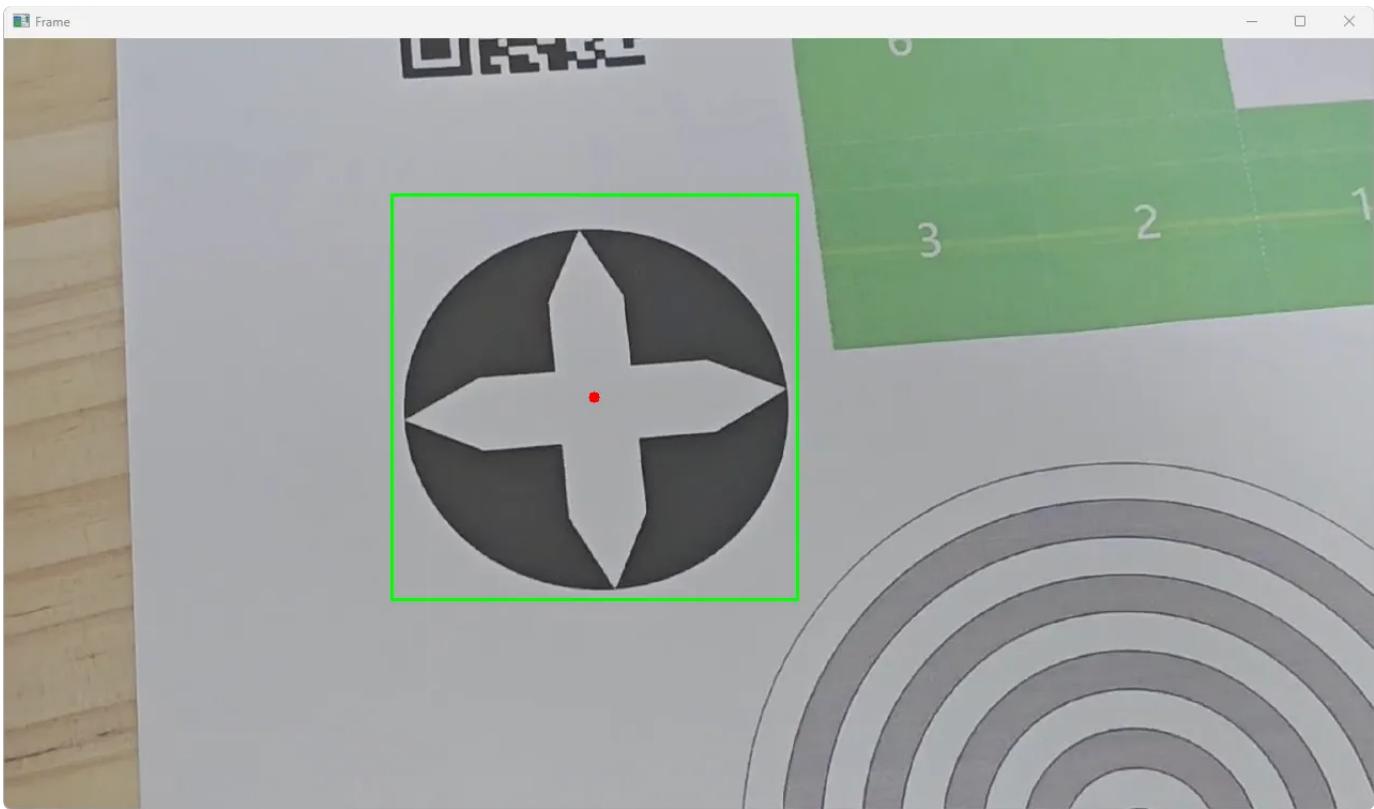


```
Frame 354: Max confidence: 0.26706191897392273
Frame 354: Center of the box: (341, 645)
Frame 360: Max confidence: 0.23933418095111847
Frame 366: Max confidence: 0.24603517353534698
Frame 372: Max confidence: 0.22562956809997559
Frame 378: Max confidence: 0.24825870990753174
Frame 384: Max confidence: 0.3039780259132385
Frame 384: Center of the box: (345, 746)
Frame 390: Max confidence: 0.3402646780014038
Frame 390: Center of the box: (334, 761)
Frame 396: Max confidence: 0.3177894651889801
Frame 396: Center of the box: (325, 760)
Frame 402: Max confidence: 0.3260599672794342
Frame 402: Center of the box: (364, 755)
Frame 408: Max confidence: 0.3147903084754944
Frame 408: Center of the box: (412, 751)
Frame 414: Max confidence: 0.25542110204696655
Frame 414: Center of the box: (402, 736)
```

植保飞行器：飞机起降点检测——模板匹配

掩膜分割，二值化处理，多模板匹配，跳帧匹配

```
Frame 508: Center of the box: (659, 394)
Frame 512: Center of the box: (675, 401)
Frame 516: Center of the box: (686, 418)
Frame 520: Center of the box: (698, 434)
Frame 524: Center of the box: (694, 439)
Frame 528: Center of the box: (661, 449)
Frame 532: Center of the box: (602, 461)
Frame 536: Center of the box: (562, 457)
```



空地协同消防系统：火源识别——颜色掩膜

