



OpenCV

介绍

C
O
N
T
E
N
T
S

目
录

[1] 概述

[2] 核心操作介绍

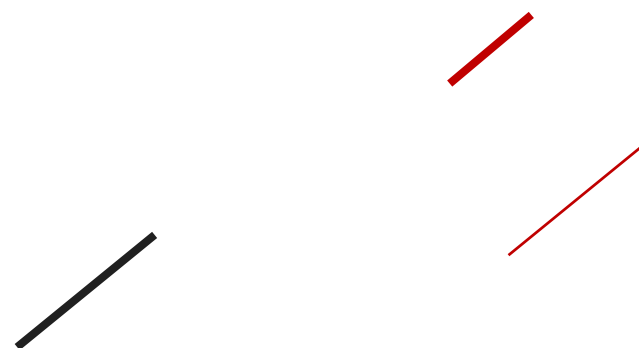
[3] 应用实例

1

A cluster of five small triangles in light red, dark red, black, and grey, arranged in a circular pattern.

PART

概述



功能

OpenCV是一个计算机视觉库,它实现了图像处理和计算机视觉方面的很多通用算法

Image processing



通用图像处理



变形



匹配

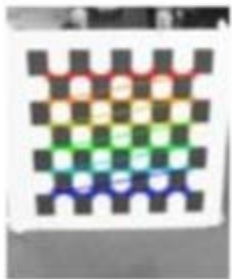


光流



分割

Video, Stereo, and 3D



相机标定



姿态估计



特征提取



深度图



物体检测



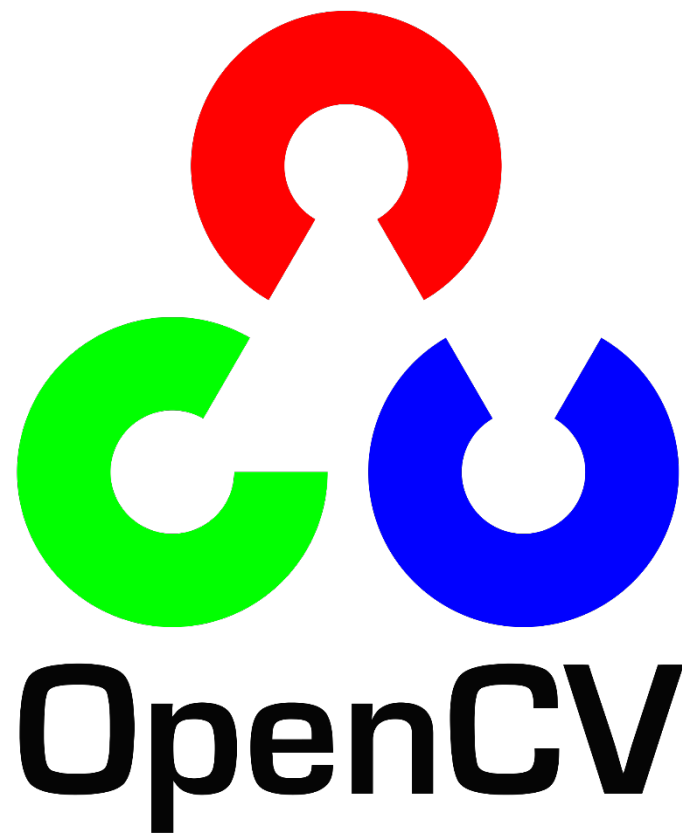
使用许可

BSD开源协议

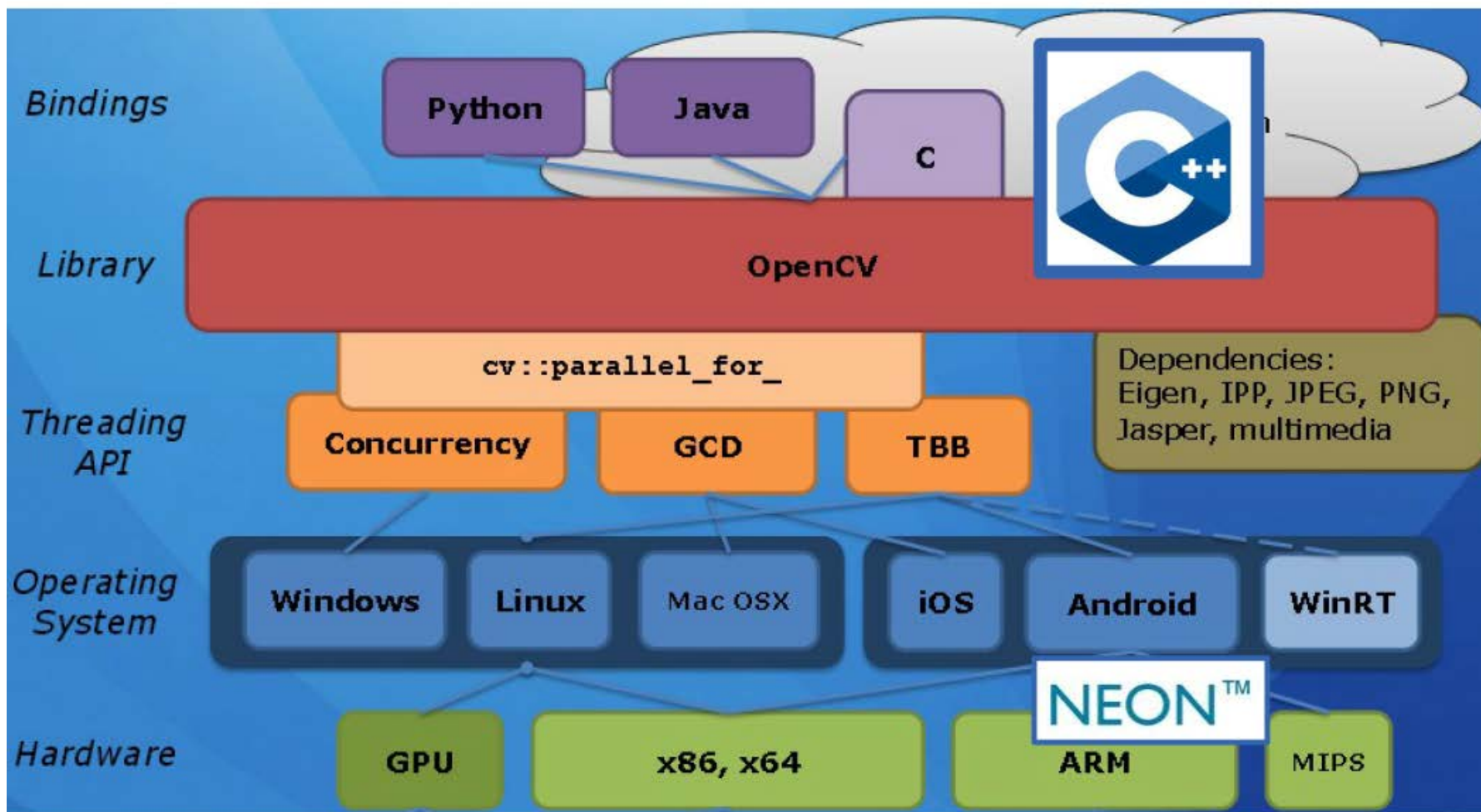
可随意使用、修改源码

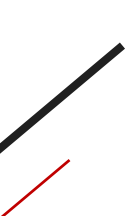
修改源码后，可闭源

可以商业使用



运行环境





python安装OpenCV

安装python和python包管理工具pip

将python安装路径加入环境变量

输入命令：

```
1 pip install numpy
2 pip install Matplotlib
3 pip install opencv-python
```

hello world.py

```
import cv2 as cv
pic = cv.imread('test.jpg')          读取一张图片
grey_pic = cv.cvtColor(pic, cv.COLOR_BGR2GRAY)  将图像转换成灰度图像
box_color = (0, 255, 0)               设置边框的颜色
classifier=cv.CascadeClassifier("haarcascade_frontalface_default.xml")  载入OpenCV提供的人脸识别分类器
faceRects = classifier.detectMultiScale(grey_pic, scaleFactor = 1.2, minNeighbors = 3, minSize = (32, 32))进行分类, 输出坐标
if len(faceRects) > 0:
    for faceRect in faceRects:
        x, y, w, h = faceRect
        cv.rectangle(pic, (x - 10, y - 10), (x + w + 10, y + h + 10), box_color, 2)  如果检测到人脸, 则根据坐标用矩形框圈出人脸
cv.namedWindow('pic')                 创建名为"pic"的窗口
cv.imshow('pic',pic)                 在窗口中显示图片
cv.waitKey(0)&0xFF                    等待键盘输入
cv.destroyAllWindows()               关闭所有窗口
```


hello world.py




2



PART

核心操作



图像表示

图像的基础操作

255	0	0	0	255	0	0	0	255
-----	---	---	---	-----	---	---	---	-----

公路堵车二

--	--	--

```
import cv2
```



```
img2=cv2.imread('101.jpg')  
head=img2[15:85,45:125]  
img2[10:80,345:425]=head
```

#读取
像素点
[0,0]
变像
形状
(3, 3)
大小



#获取头部区域矩阵
#将目标区域用头部区域覆盖

图像加法

OpenCV: `cv2.add()`, 大于255取255

Numpy: `"+"`, 大于255取模的大小

上述运算均要求两张图像的大小必须一致,
或者第二个图像可以是一个简单的标量值

```
opencv = cv.imread('opencv.png')
python = cv.imread('python.jpg')
plus_by_opencv = cv.add(python, opencv)
plus_by_numpy = python + opencv
```



图像混合

`cv2.addWeighted()`

两幅图像的权重不同的加法，给人一种混合或者透明的感觉。计算公式如下：

$$g(x) = (1 - \alpha) f_0(x) + \alpha f_1(x)$$

```
opencv = cv.imread('opencv.png')
python = cv.imread('python.jpg')
plus = cv.addWeighted(opencv, 0.5, python, 0.5, 0)
cv.imshow('addWeighted', plus)
```



按位运算：AND, OR, NOT, XOR

bitwise_and():

对图像（灰度图像或彩色图像均可）每个像素值进行二进制“与”操作

bitwise_or():

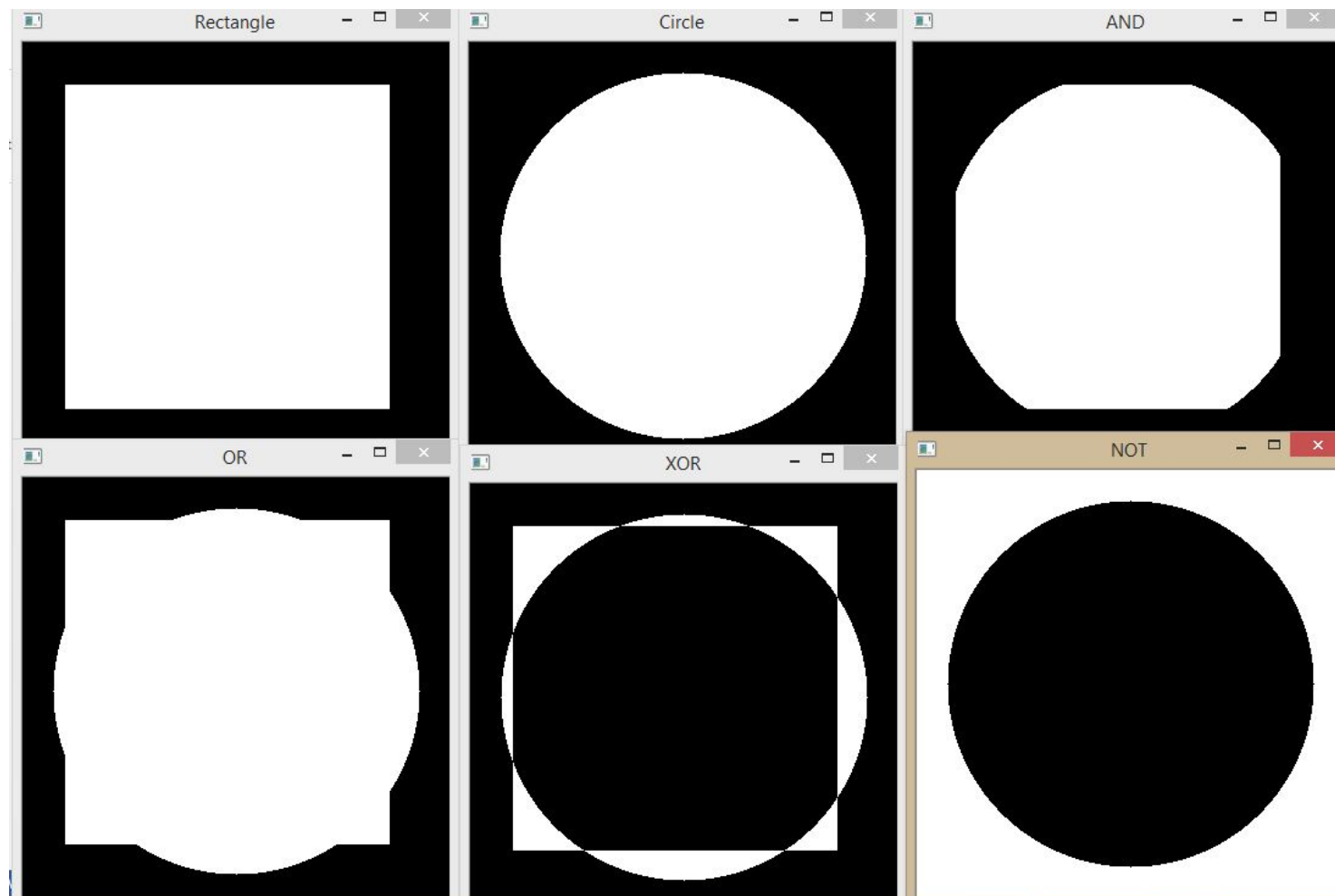
对图像（灰度图像或彩色图像均可）每个像素值进行二进制“或”操作

bitwise_xor():

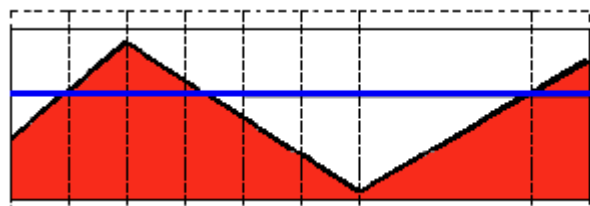
对图像（灰度图像或彩色图像均可）每个像素值进行二进制“异或”操作

bitwise_not():

对图像（灰度图像或彩色图像均可）每个像素值进行二进制“非”操作



图像上的算术运算

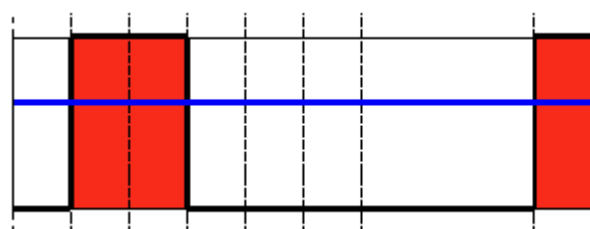


255

150

#通过threshold函数将图像二值化

```
ret,mask = cv.threshold(opencv_grey,150,255,cv.THRESH_BINARY)
```



255

150

```
background = cv.bitwise_and(python,python,mask=mask)
```

#合并前后景

```
plus = cv.add(foreground,background)
```

```
,cv.COLOR_BGR2GRAY)
```

```
cv,opencv,mask=mask_inv)
```




3



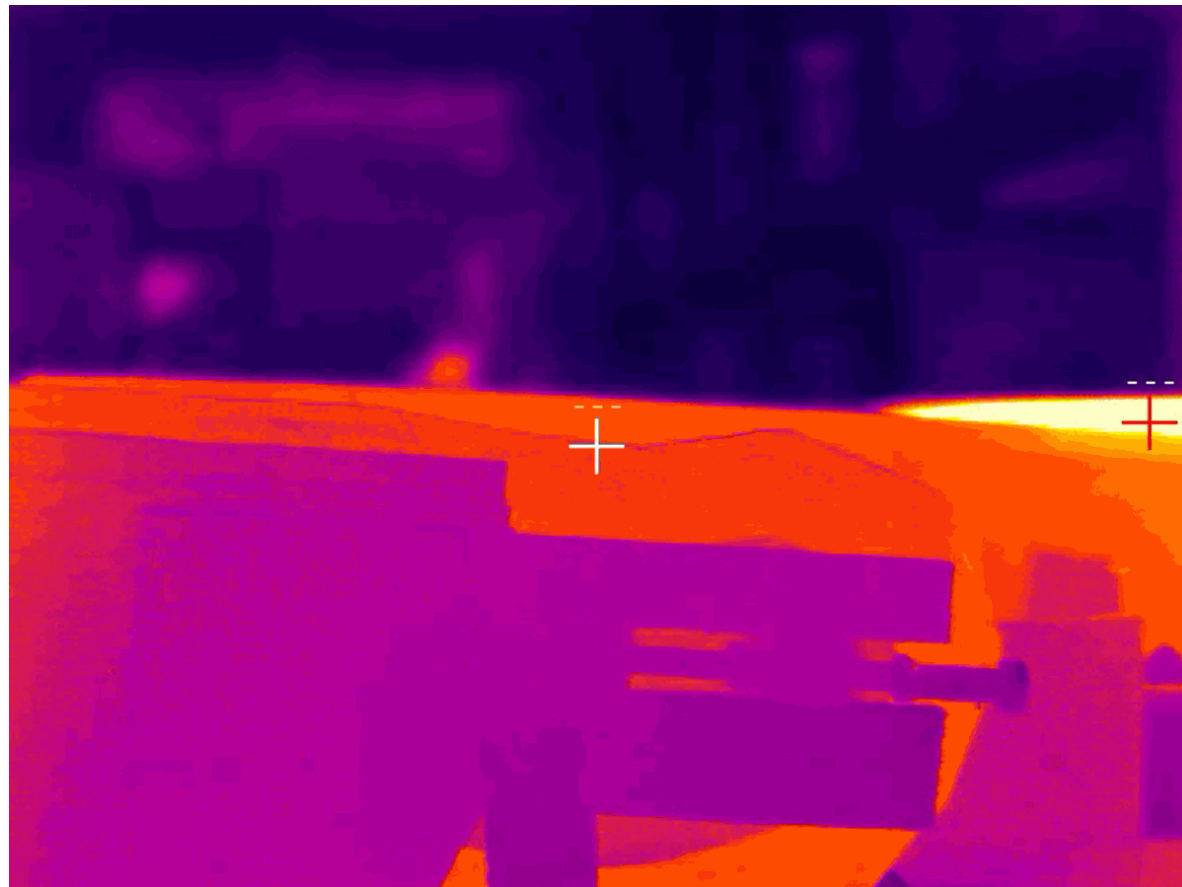
PART

项目实例



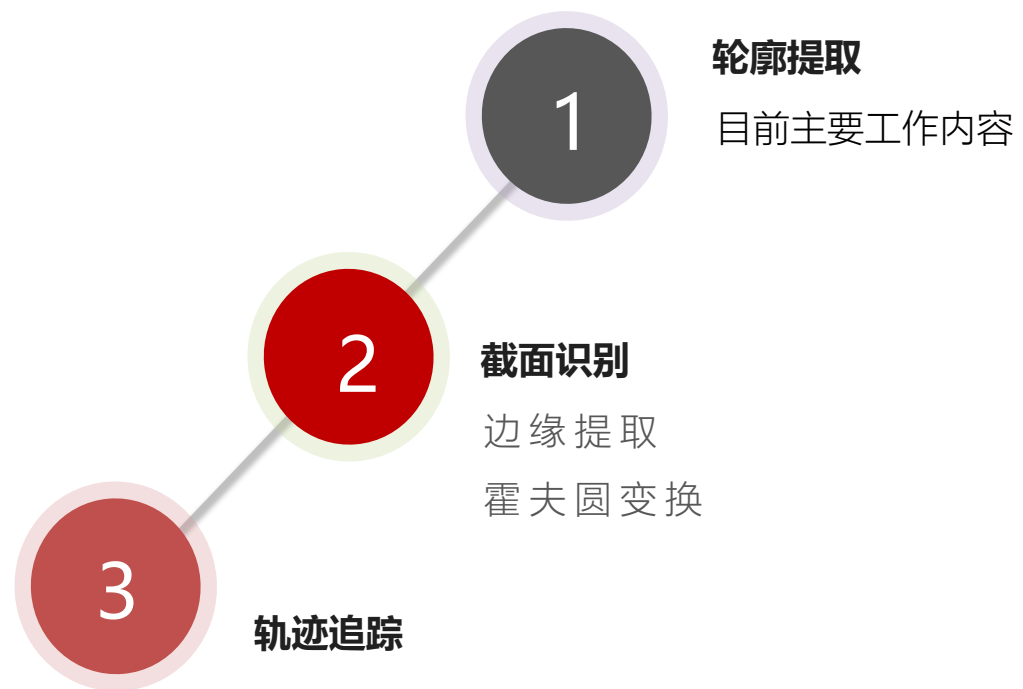
出厂钢筋自动计数

实现对热感摄像头的监视画面进行识别
要求实现对视频中传送带送出钢筋的自动
计数





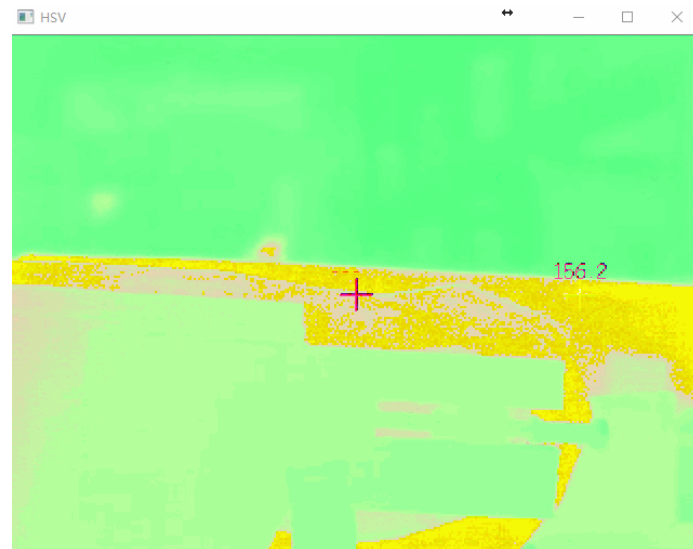
流程分析



色彩空间转换

将BGR空间转为HSV和灰度空间

```
#读取视频文件
cap = cv2.VideoCapture('vtest.mp4')
while(1):
    #获取单帧
    ret, frame = cap.read()
    #ret值表示该帧是否存在
    if ret:
        #转化为HSV空间
        HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        #转化为灰度空间
        GRAY = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow('HSV', HSV)
        cv2.imshow('Gray', GRAY)
```

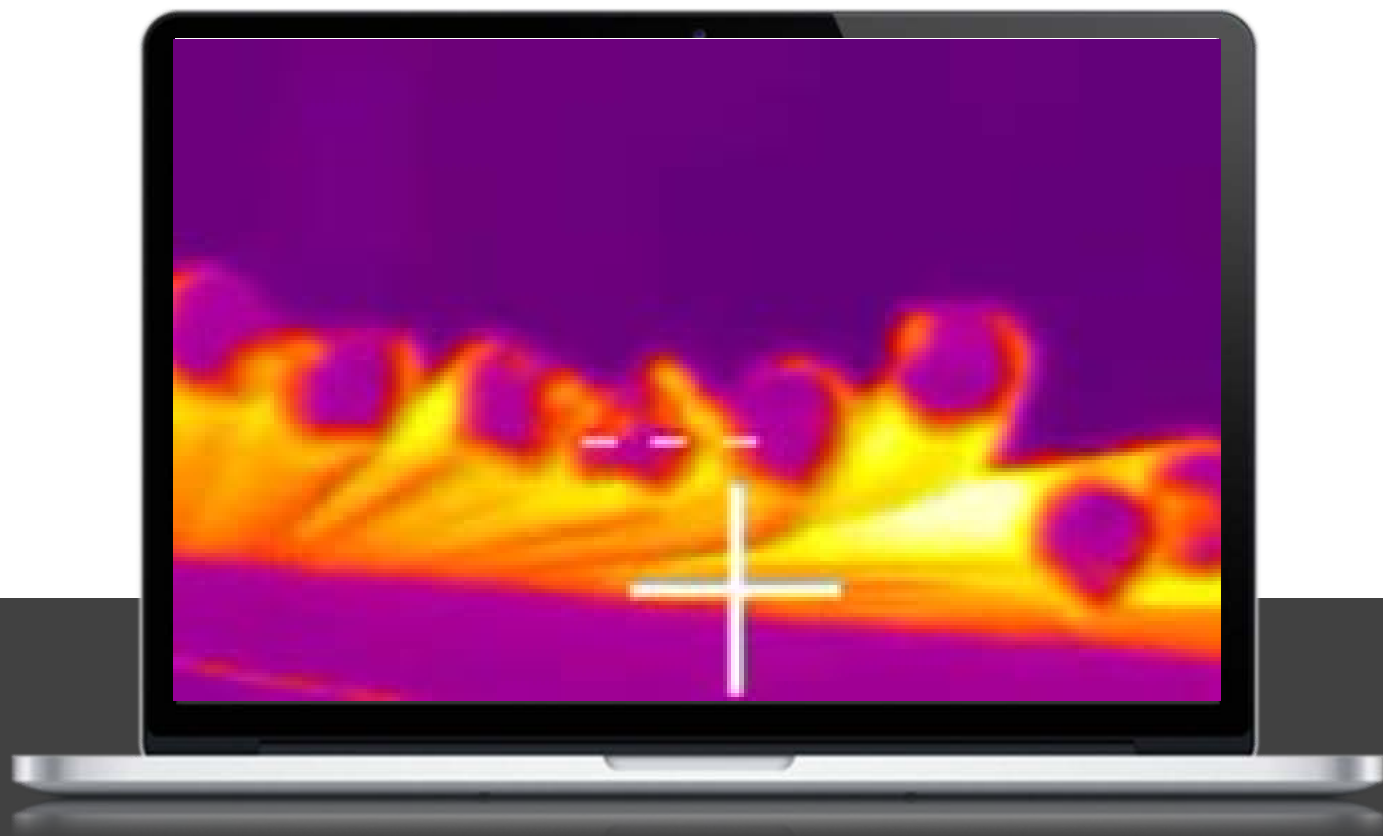


HSV 空间



灰度空间

轮廓提取目标



最理想的目标是提取到包括钢筋圆截面和钢筋筋身的轮廓

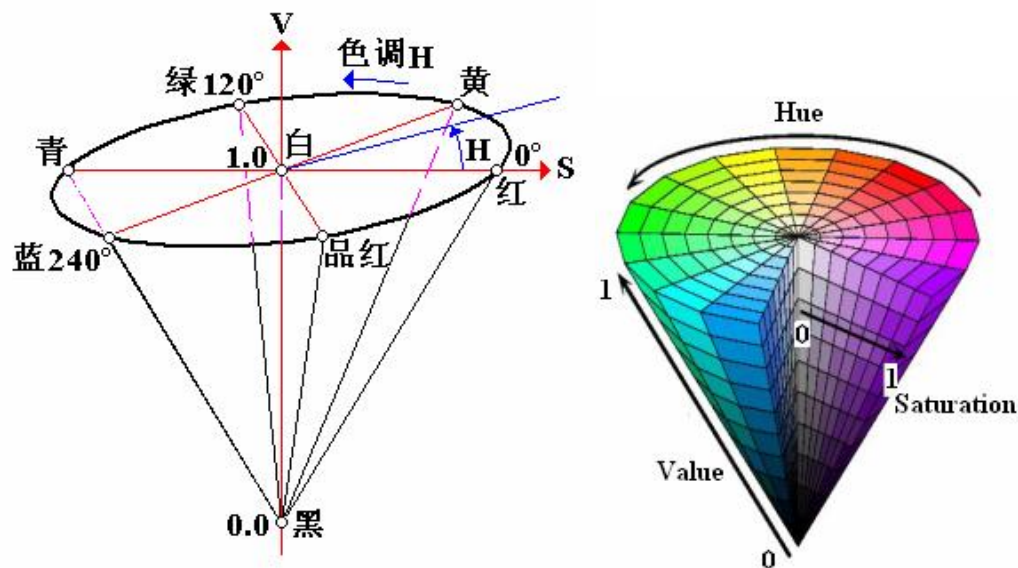
HSV空间简介

HSV(Hue, Saturation, Value)是根据颜色的直观特性由A. R. Smith在1978年创建的一种颜色空间, 也称六角锥体模型(Hexcone Model)。这个模型中颜色的参数分别是: 色调 (H), 饱和度 (S), 亮度 (V)。

H:表示色彩信息, 即所处的光谱颜色的位置。该参数用一角度量来表示

S:为一比例值, 范围从0到1, 它表示成所选颜色的纯度和该颜色最大的纯度之间的比率。S=0时, 只有灰度

V:表示色彩的明亮程度, 范围从0到1



HSV在OpenCV中的取值范围:

H: 0—180

S: 0—255

V: 0—255

单帧分析

中间黄色和红色部分：

H：15-35

S：250+（红色部分不考虑）

V：250+

钢筋截面的绿色部分

H：145+

S：250+

V：160+

剩余绿色部分

H：145+

S：250+

V：170-

中间黄色和红色部分提取范围

H：15-50

S：0-255

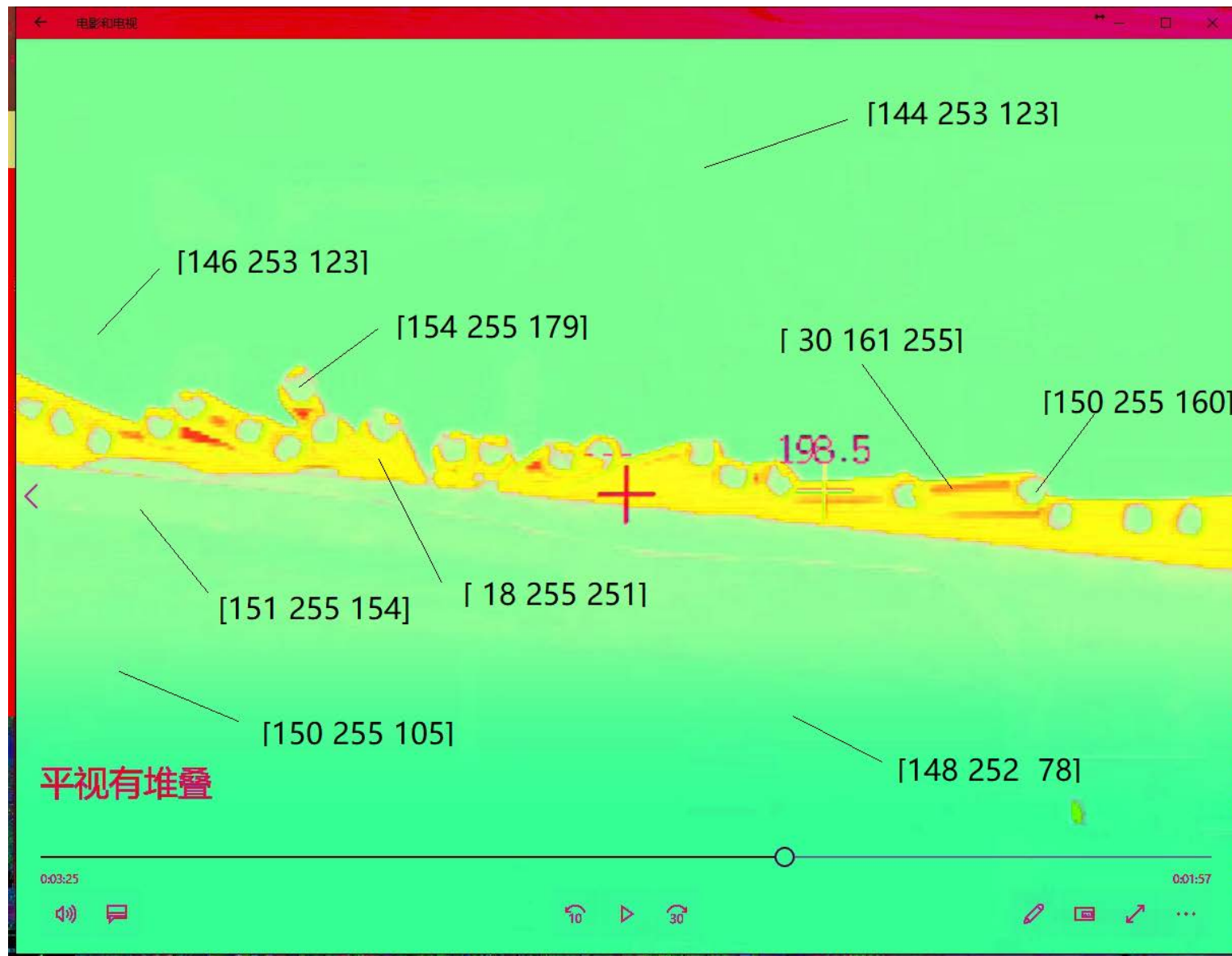
V：0-255

钢筋截面的绿色部分提取范围

H：140-255

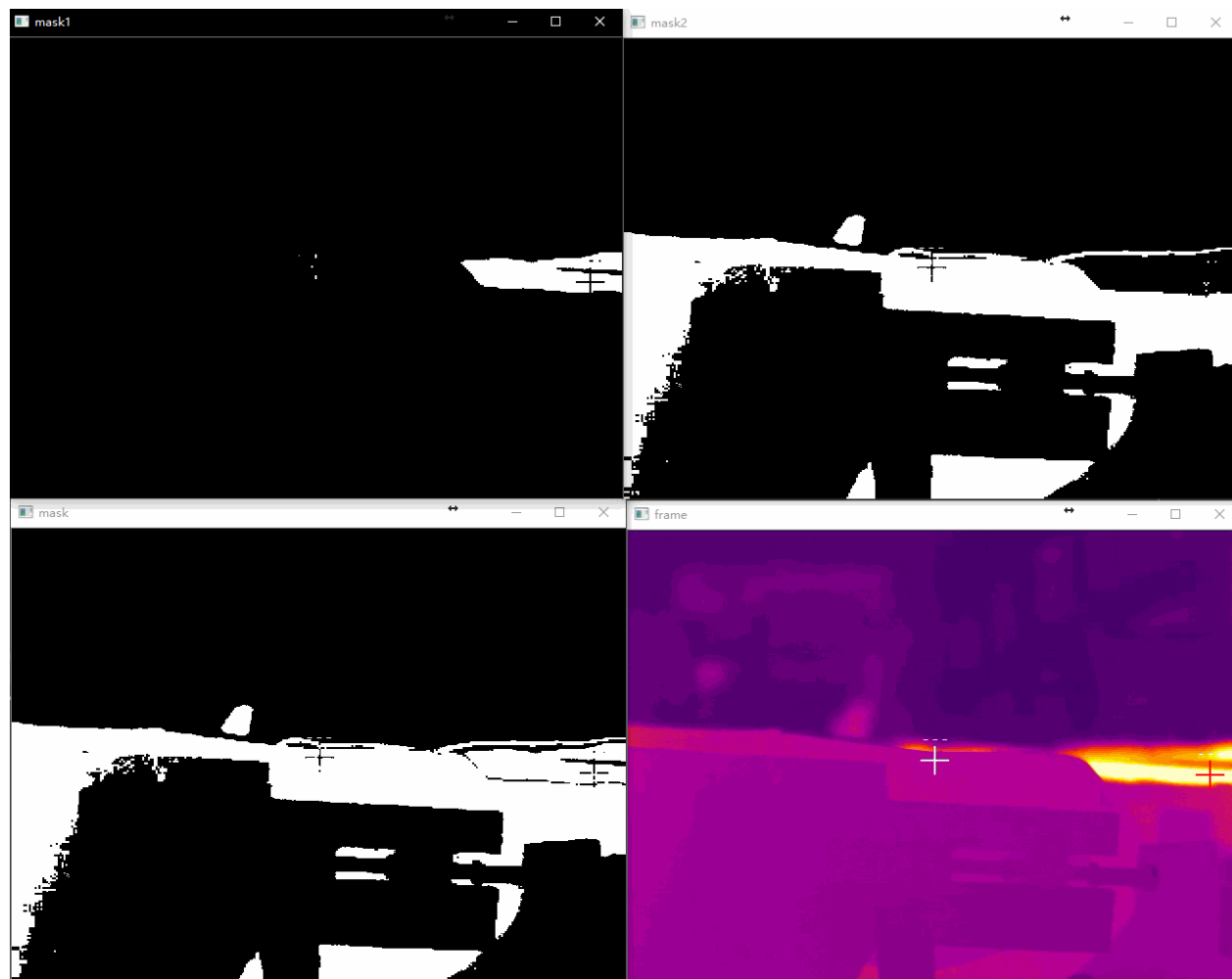
S：0-255

V：160-255



按颜色进行提取

```
cap = cv2.VideoCapture('vtest.mp4')
while(1):
    ret, frame = cap.read()
    HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    #提取钢筋筋身部分
    lower1 = np.array([15,0,0])
    upper1 = np.array([50,255,255])
    mask1 = cv2.inRange(HSV, lower1, upper1)
    #提取钢筋截面部分
    lower2 = np.array([140,0,160])
    upper2 = np.array([255,255,255])
    mask2 = cv2.inRange(HSV, lower2, upper2)
    #将两个矩阵进行与操作
    mask = cv2.bitwise_or(mask1, mask2)
```

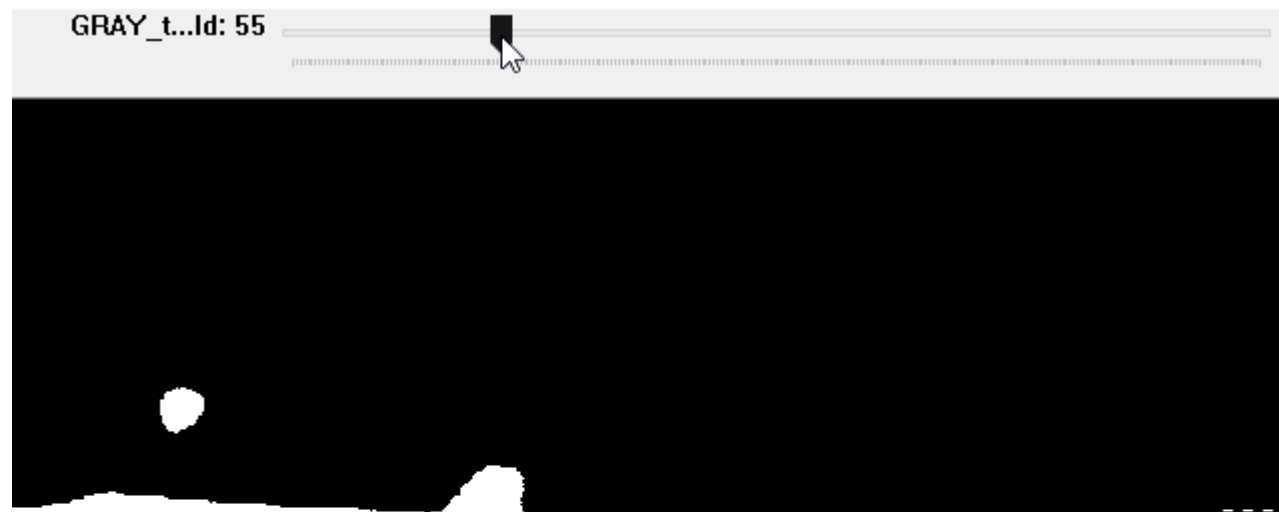


分析

1. 估计摄像头是基于热辐射来取像，所以当视频没有钢筋时图像颜色会产生变化，此时会出现比较大的误差。
2. 并不能将目标区域完全取到。如果调整阈值保证目标区域能取到，这时会取到较多的无关区域。

第二个问题，使用灰度特征作为分割依据的情况依然存在，如右图所示，找不到一个合适的分离出轮廓的阈值

总而言之单靠色彩或这灰度，或者两者结合的特征进行阈值图像分割无法得到较为理想的结果，还需要使用其他的方法





灰度二值化和形态学滤波

灰度二值化：通过OpenCV的threshold函数

数学形态学 (Mathematical morphology) 是一门建立在格论和拓扑学基础之上的图像分析学科，是数学形态学图像处理的基本理论。

其基本的运算包括：二值腐蚀和膨胀、二值开闭运算、骨架抽取、极限腐蚀、击中击不中变换、形态学梯度、Top-hat变换、颗粒分析、流域变换、灰值腐蚀和膨胀、灰值开闭运算、灰值形态学梯度等。

形态学操作一般作用于二值化图，来连接相邻的元素或分离成独立的元素。

此处用到了形态学方法中的一种最基本的操作：**膨胀 (Dilation)**

膨胀和腐蚀



原图



腐蚀



膨胀

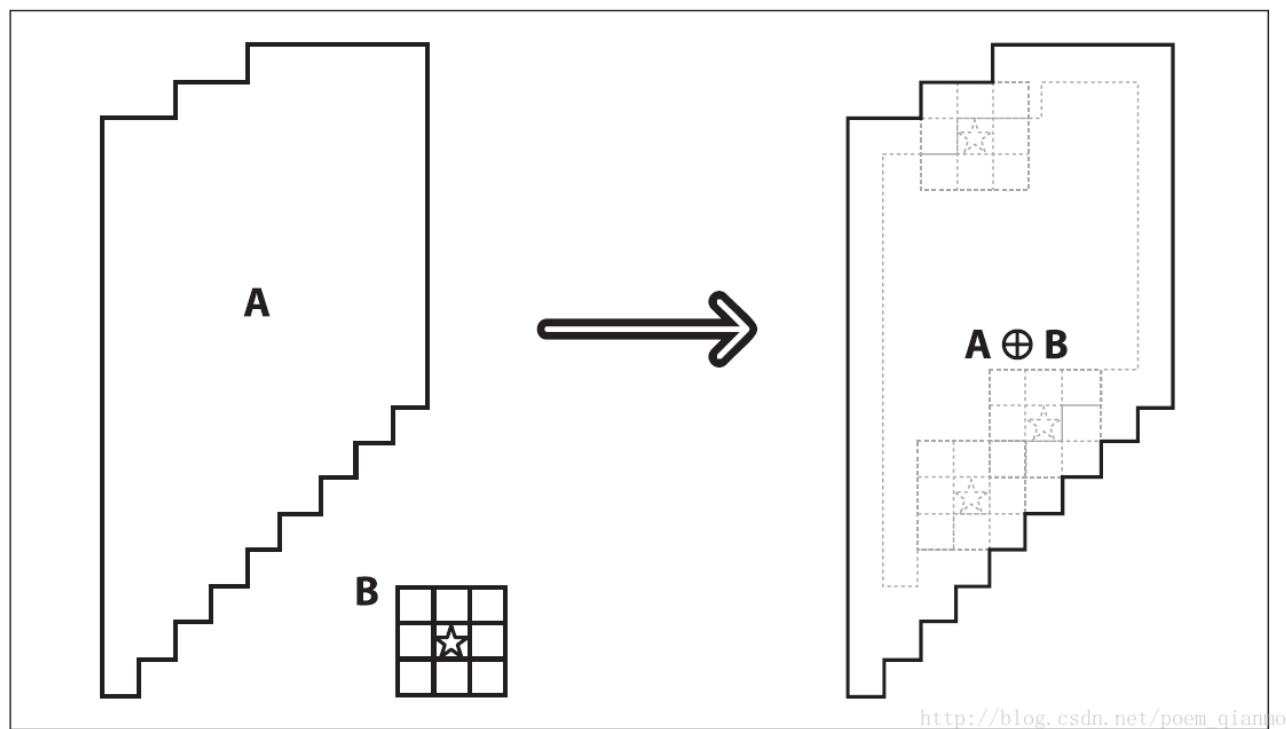
膨胀：图像中的高亮部分（白色部分）进行膨胀，使得效果图拥有比原图更大的高亮区域

腐蚀：图图像中的高亮部分（白色部分）被腐蚀，使得效果图拥有比原图更小的高亮区域。

膨胀原理

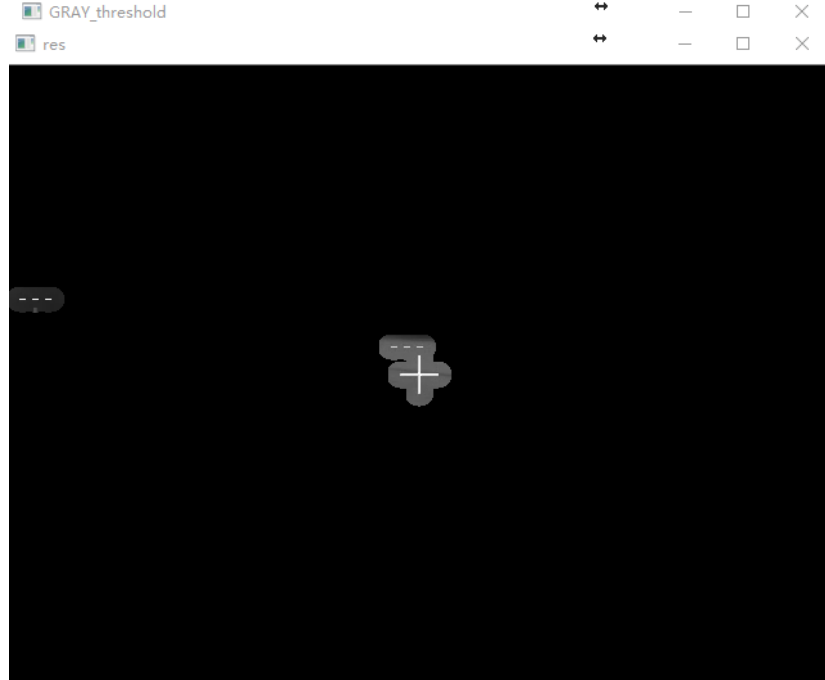
膨胀就是求局部最大值的操作。

具体的实现来说，膨胀或者腐蚀操作就是将图像A与核B进行卷积。即计算核B覆盖的区域的像素点的最大值，并把这个最大值赋值给参考点指定的像素。这样就会使图像中的高亮区域逐渐增长。



膨胀应用

```
cap = cv2.VideoCapture('vtest.mp4')
while(1):
    ret, frame = cap.read()
    if ret:
        # 将BGR图像转换为灰度图像
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # 设置阈值
        # 阈值为140
        # MORPH_ELLIPSE表示核结构为椭圆
        # 选择矩形核结构: cv2.MORPH_RECT
        # 与操作得到最后提取的轮廓
        res = cv2.bitwise_and(gray, gray, mask=DILATE)
```



背景减除

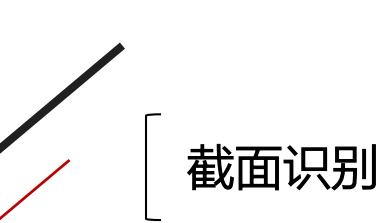
通常我们假定没有入侵物体的静态场景具有一些常规特性，可以用一个统计模型描述。

已知这个背景模型，一旦有入侵物体，入侵物体就能通过标出场景图像中不符合这一背景模型的部分来检测到。这一过程被称为背景减除 (Backgroundsubtraction)

MOG2

```
cap = cv2.VideoCapture('vtest.mp4')
fgbg = cv2.createBackgroundSubtractorMOG2()
while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame', frame)
    cv2.imshow('fgmask', fgmask)
```





还在尝试阶段

目前计划通过霍夫圆变换来实现截面识别

霍夫变换原理

【1】众所周知, 一条直线在图像二维空间可由两个变量表示. 如:

<1>在笛卡尔坐标系: 可由参数: 斜率和截距(m,b) 表示。

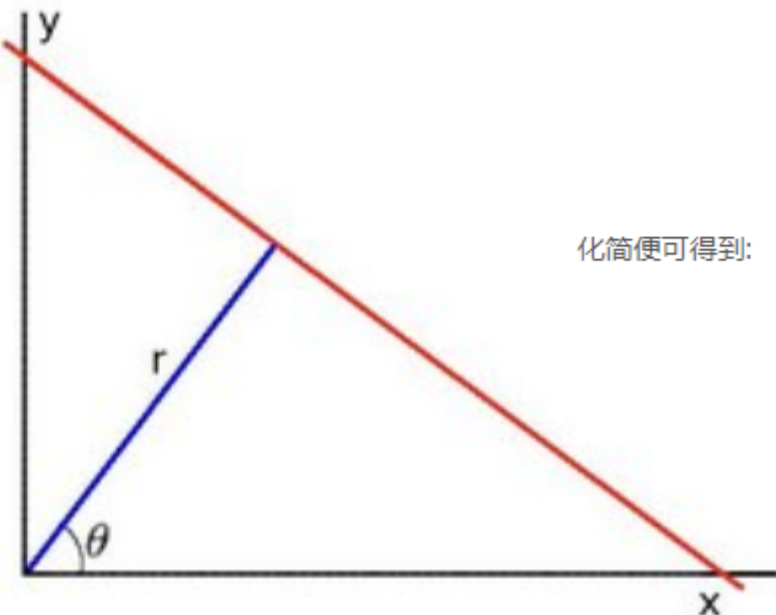
<2>在极坐标系: 可由参数: 极径和极角(r, θ)表示。

对于霍夫变换, 我们将采用第二种方式极坐标系来表示直线. 因此, 直线的表达式可为:

$$y = \left(-\frac{\cos \theta}{\sin \theta} \right) x + \left(\frac{r}{\sin \theta} \right)$$

化简便可得到:

$$r = x \cos \theta + y \sin \theta$$





霍夫变换原理

【2】一般来说对于点 (x_0, y_0) , 我们可以将通过这个点的一族直线统一定义为:

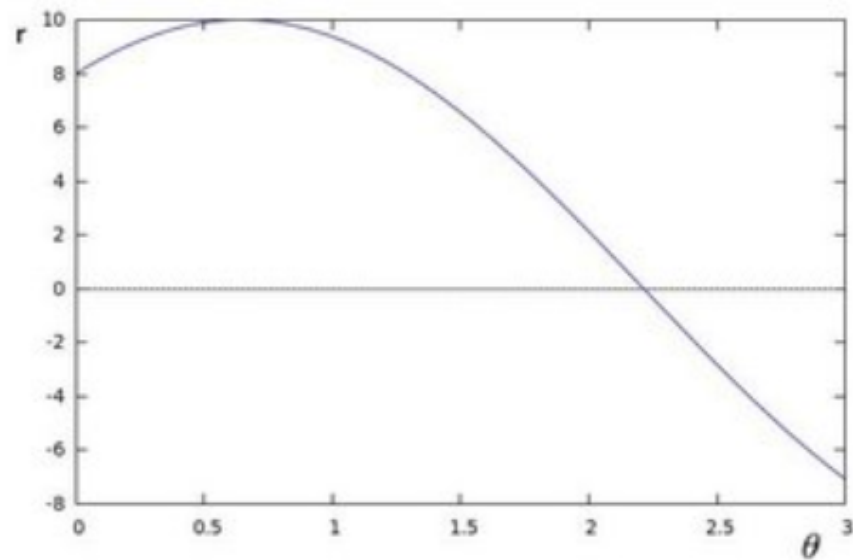
$$r = x \cos \theta + y \sin \theta$$

这就意味着每一对 (r_θ, θ) 代表一条通过点 (x_0, y_0) 的直线。

霍夫变换原理

【3】 如果对于一个给定点 (x_0, y_0) 我们在极坐标对极径极角平面绘出所有通过它的直线, 将得到一条正弦曲线. 例如, 对于给定点 $X_0 = 8$

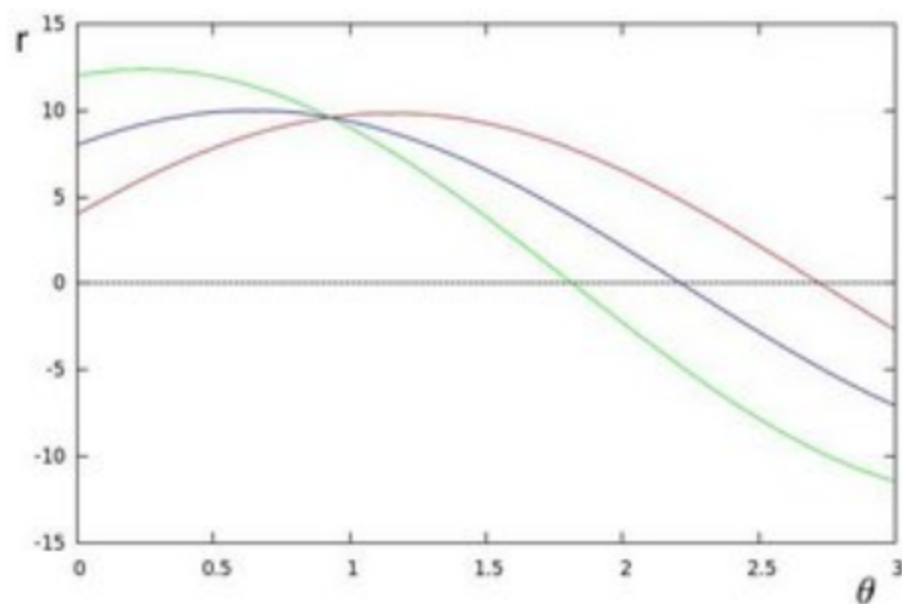
和 $Y_0 = 6$ 我们可以绘出下图 (在平面):



只绘出满足下列条件的点 $r > 0$ 和 $0 < \theta < 2\pi$.

霍夫变换原理

【4】 我们可以对图像中所有的点进行上述操作. 如果两个不同点进行上述操作后得到的曲线在平面 $\theta - r$ 相交, 这就意味着它们通过同一条直线. 例如, 接上面的例子我们继续对点 $x_1 = 9, y_1 = 4$ 和点 $x_2 = 12, y_2 = 3$ 绘图, 得到下图:



这三条曲线在平面相交于点 $(0.925, 9.6)$, 坐标表示的是参数对 $\theta - r$ 或者说点 (x_0, y_0) , 点 (x_1, y_1) 和点 (x_2, y_2) 组成的平面内的的直线。

霍夫变换原理

【5】 以上的说明表明, 一般来说, 一条直线能够通过平面 $\theta - r$ 寻找交于一点的曲线数量来检测。而越多曲线交于一点也就意味着这个交点表示的直线由更多的点组成。一般来说我们可以通过设置直线上点的阈值来定义多少条曲线交于一点我们才认为检测到了一条直线。

【6】 这就是霍夫线变换要做的。它追踪图像中每个点对应曲线间的交点。如果交于一点的曲线的数量超过了阈值, 那么可以认为这个交点所代表的参数对 (θ, r_θ) 在原图像中为一条直线。



霍夫圆变换

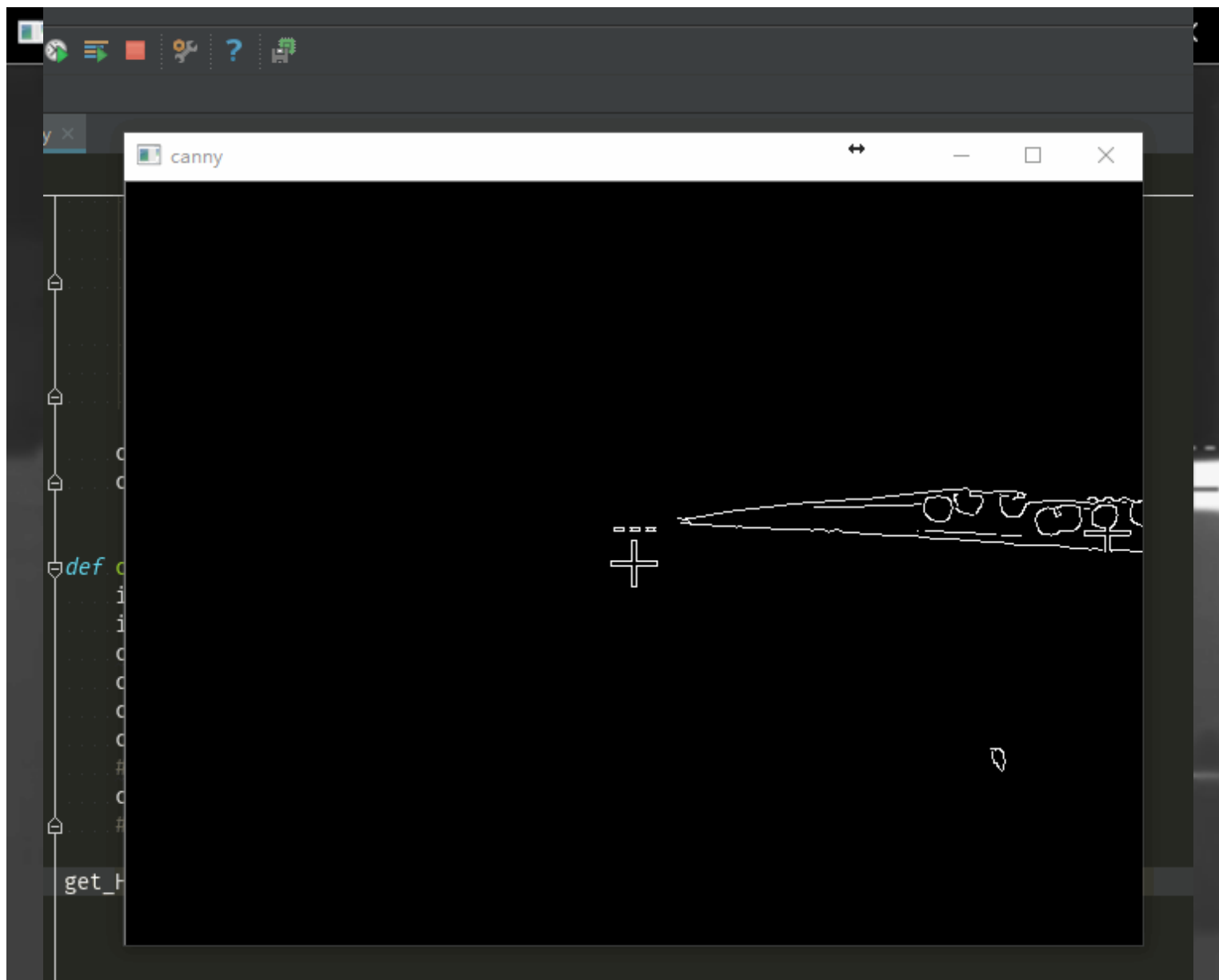
对直线来说，一条直线能由参数极径极角表示。而对圆来说，我们需要三个参数来表示一个圆，也就是：

$$C : (x_{center}, y_{center}, r)$$

霍夫圆变换的基本原理和上面讲的霍夫线变化大体上是很类似的，只是点对应的二维极径极角空间被三维的圆心点 x ， y 还有半径 r 空间取代。说“大体上类似”的原因是，如果完全用相同的方法的话，累加平面会被三维的累加容器所代替：在这三维中，一维是 x ，一维是 y ，另外一维是圆的半径 r 。这就意味着需要大量的内存而且执行效率会很低，速度会很慢。

效率优化：霍夫梯度法

测试效果





还在调研阶段

考虑使用模板匹配法



THANKS

请输入你的内容

