实验六 形态学图像处理

一、 实验目的

（1）学习例如腐蚀，膨胀，开运算，闭运算,边缘检测等操作，掌握 cv2.erode()，cv2.dilate()，cv2.morphologyEx()函数的使用；

（2）学习全局全局门限值分割，自适应阈值分割及 Otsu’s 二值化等，掌握 cv2.threshold，cv2.adaptiveThreshold 等的使用；

（3）掌握对图像进行颜色空间转换，比如从 BGR 到灰度图，或者从 BGR 到 HSV 等；

（4）掌握根据颜色进行视频跟踪的初步方法。

二、实验室原理与实验步骤

3.1 形态学转换

形态学操作是根据图像形状进行的简单操作。一般情况下对二值化图像进行的操作。需要输入两个参数，一个是原始图像，第二个被称为结构化元素或核，它是用来决定操作的性质的。两个基本的形态学操作是腐蚀和膨胀。他们的变体构成了开运算，闭运算，梯度等。

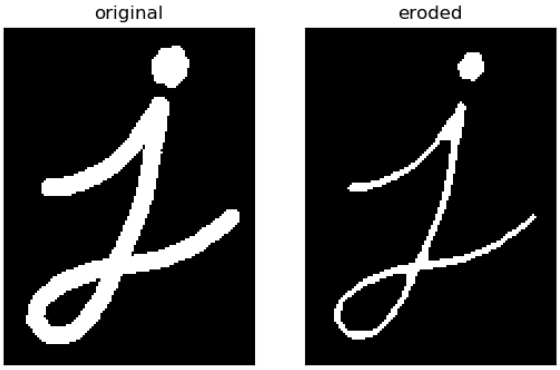
3.1.1 腐蚀

结构元素沿着图像滑动，如果与结构元素对应的原图像的所有像素值都是 1，那么中心元素就保持原来的像素值，否则就变为零。（注意：这是不是正式定义。）

这里我们有一个例子，使用一个 5x5 的结构元素，其中所有的值都是 1。

```
import cv2
import numpy as np
img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
```

结果：

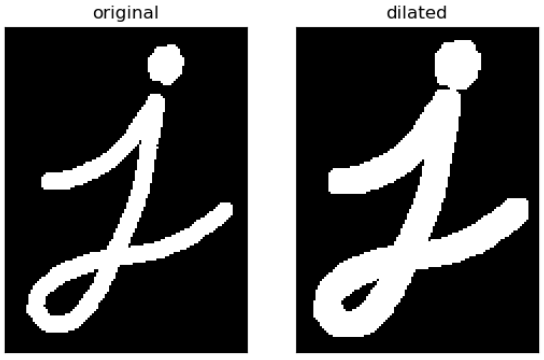


3.1.2 膨胀

与腐蚀相反，与结构元素对应的原图像的像素值中只要有一个是 1，中心元素的像素值就是 1。所以这个操作会增加图像中的白色区域（前景）。

```
dilation = cv2.dilate(img,kernel,iterations = 1)
```

结果：



3.1.3 开运算

进行腐蚀再进行膨胀就叫做开运算。就像我们上面介绍的那样，它被用来去除噪声。这里我们用到的函数是 **cv2.morphologyEx()**。

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

结果：

3.1.4 闭运算

先膨胀再腐蚀。它经常被用来填充前景物体中的小洞，或者前景物体上的小黑点。

```
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

结果：

3.1.5 形态学梯度

其实就是一幅图像膨胀与腐蚀的差别。结果看上去就像前景物体的轮廓。

```
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
```

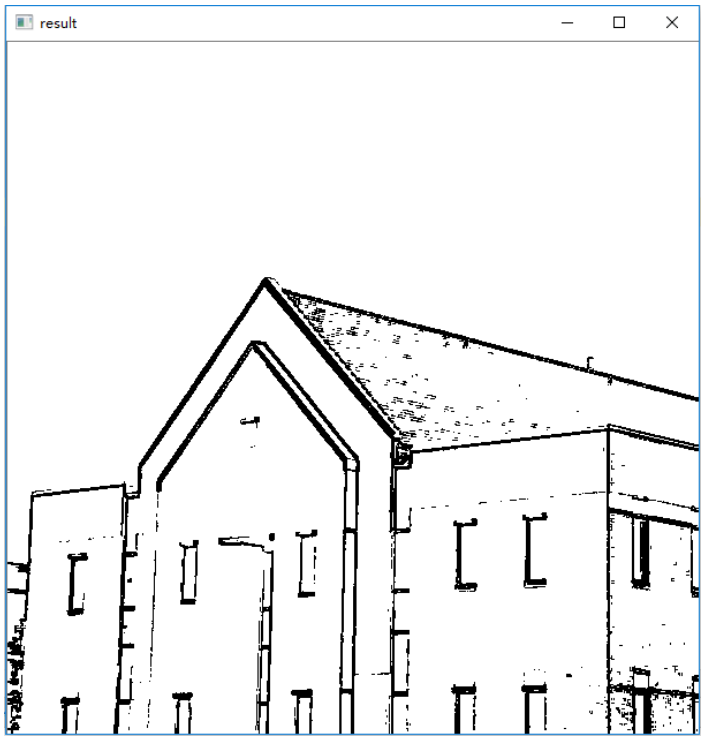
结果：

3.1.6 边缘检测

形态学检测边缘的原理很简单，在膨胀时，图像中的物体会想周围“扩张”；腐蚀时，图像中的物体会“收缩”。比较这两幅图像，由于其变化的区域只发生在边缘。所以这时将两幅图像相减，得到的就是图像中物体的边缘。

```
import cv2
import numpy

image = cv2.imread("house.tif",0);
#构造一个 3×3 的结构元素
element = cv2.getStructuringElement(cv2.MORPH_RECT,(3, 3))
dilate = cv2.dilate(image, element)
erode = cv2.erode(image, element)

#将两幅图像相减获得边，第一个参数是膨胀后的图像，第二个参数是腐蚀后的图像
result = cv2.absdiff(dilate,erode);

#上面得到的结果是灰度图，将其二值化以便更清楚的观察结果
retval, result = cv2.threshold(result, 40, 255, cv2.THRESH_BINARY);
#反色，即对二值图每个像素取反
result = cv2.bitwise_not(result);
#显示图像
cv2.imshow("result",result);
cv2.waitKey(0)
cv2.destroyAllWindows()
```

结果：



实验图像

j.png　　腐蚀所用图　　膨胀所用图

house.tif

3.2 图像阈值

3.2.1 全局门限值方法

像素值高于阈值时，我们给这个像素赋予一个新值（可能是白色），否则我们给它赋予另外一种颜色（也许是黑色）。这个函数就是 cv2.threshhold()。这个函数的第一个参数就是原图像，原图像应该是灰度图。第二个参数就是用来对像素值进行分类的阈值。第三个参数就是当像素值高于（有时是小于）阈值时应该被赋予的新的像素值。OpenCV 提供了多种不同的阈值方法，这是有第四个参数来决定的。这些方法包括：

cv2.THRESH_BINARY

cv2.THRESH_BINARY_INV

cv2.THRESH_TRUNC

cv2.THRESH_TOZERO

cv2.THRESH_TOZERO_INV

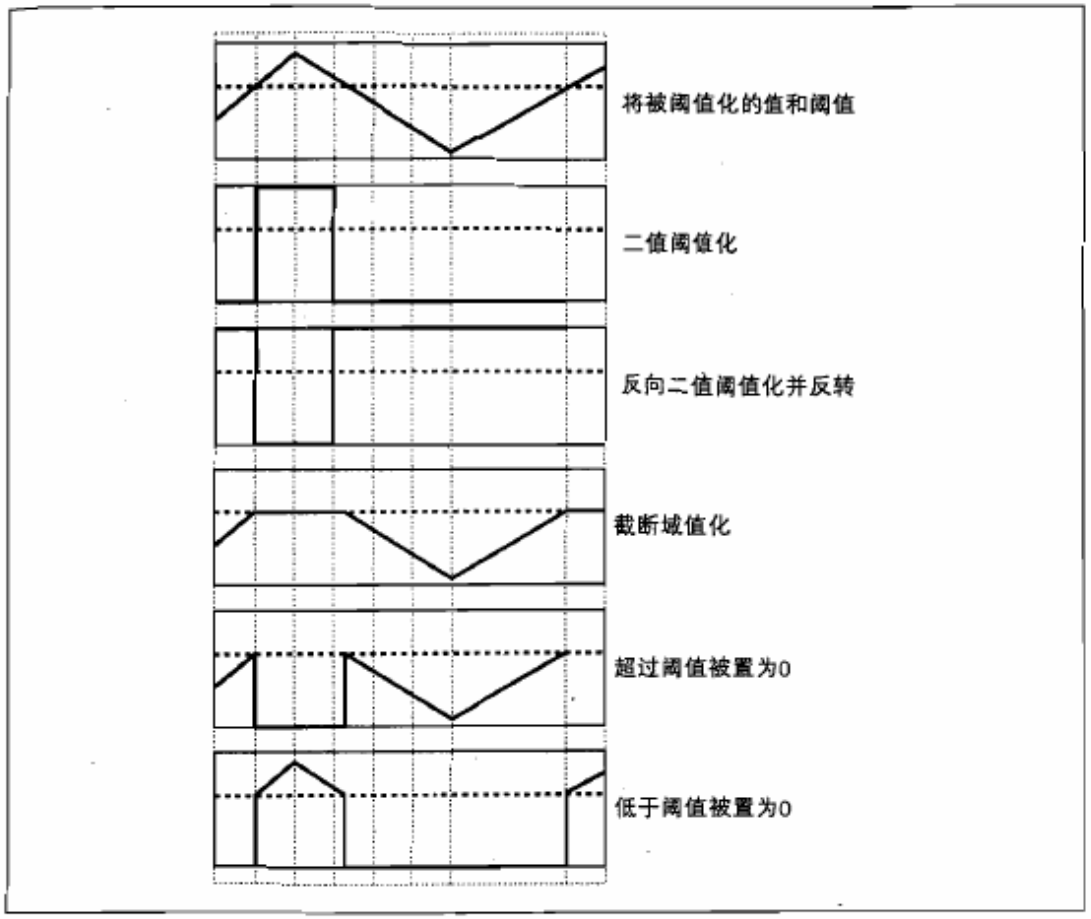


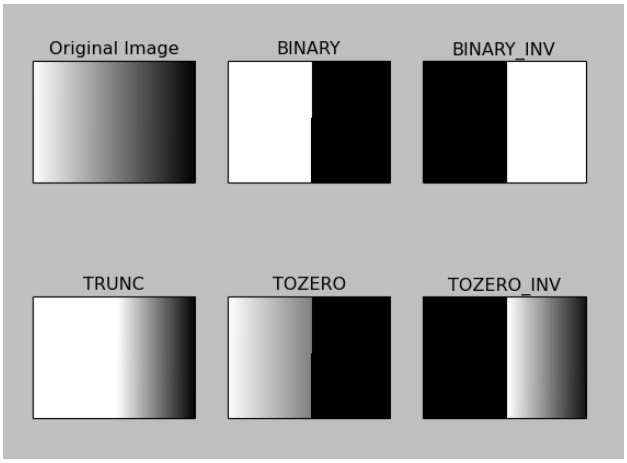
图 5-23：在 cvThreshold ()中不同阈值类型的操作结果。每个图表的水平虚线代表应用到最上方图的阈值，5 种阈值类型的操作结果列于其后 【136】

代码：

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('gradient.png',0)

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

```
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)

ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)

ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)

ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']

images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):

    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

    plt.title(titles[i])

    plt.xticks([]),plt.yticks([])

plt.show()
```

结果：



注意：为了同时在一个窗口中显示多个图像，我们使用函数 plt.subplot()。你可以通过查看 Matplotlib 的文档获得更多详细信息。

3.2.2 自适应阈值

在前面的部分我们使用是全局阈值，整幅图像采用同一个数作为阈值。当时这种方法并不适应与所有情况，尤其是当同一幅图像上的不同部分的具有不同亮度时。这种情况下我们需要采用自适应阈值。此时的阈值是根据图像上的每一个小区域计算与其对应的阈值。因此在同一幅图像上的不同区域采用的是不同的阈值，从而使我们能在亮度不同的情况下得到更好的结果。这种方法需要我们指定三个参数，返回值只有一个。

- Adaptive Method- 指定计算阈值的方法。
  - cv2.ADPTIVE_THRESH_MEAN_C：阈值取自相邻区域的平均值
  - cv2.ADPTIVE_THRESH_GAUSSIAN_C：阈值取值相邻区域的加权和，权重为一个高斯窗口。
- Block Size - 邻域大小（用来计算阈值的区域大小）。
- C - 这就是是一个常数，阈值就等于的平均值或者加权平均值减去这个常数。

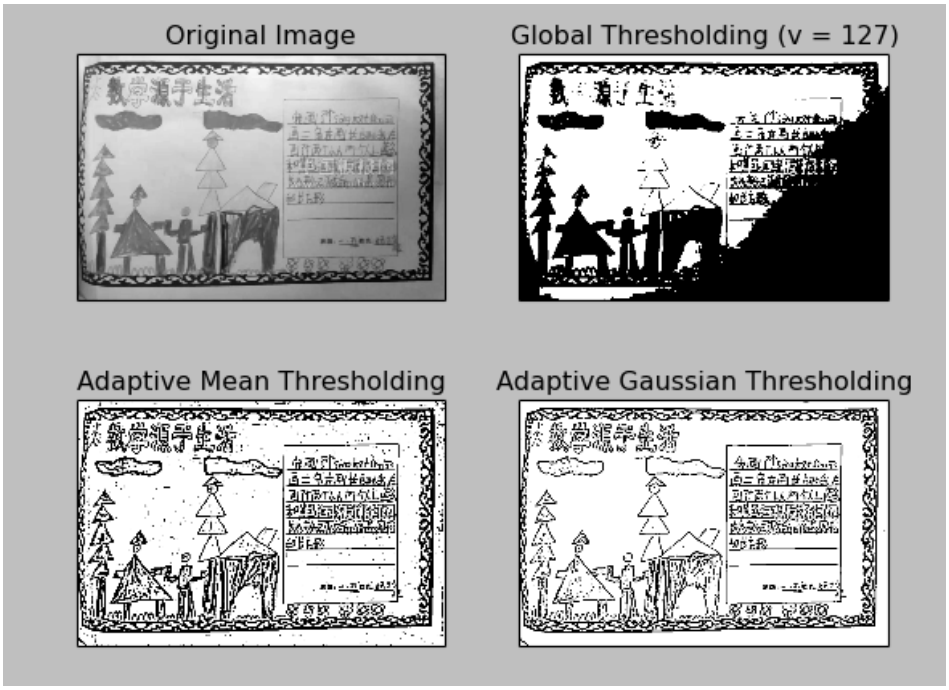
（参考：https://blog.csdn.net/on2way/article/details/46812121）

我们使用下面的代码来展示简单阈值与自适应阈值的差别：

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dave.jpg',0)
img = cv2.medianBlur(img,5)

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
            cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
```

```
            cv2.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

结果:



3.2.3 Otsu’s 二值化

在使用全局阈值时，我们就是随便给了一个数来做阈值，那我们怎么知道我们选取的这个数的好坏呢？答案就是不停的尝试。如果是一副双峰图像（简单来说双峰图像是指图像直方图中存在两个峰）呢？我们岂不是应该在两个峰之间的峰谷选一个值作为阈值？这就是 Otsu 二值化要做的。简单来说就是对一副双峰图像自动根据其直方图计算出一个阈值。(对于非双峰图像，这种方法得到的结果可能会不理想)。

这里用到到的函数还是 **cv2.threshold()**，但是需要多传入一个参数（flag）：**cv2.THRESH_OTSU**。这时要把阈值设为 0。然后算法会找到最优阈值，这个最优阈值就是返回值 **retVal**。如果不使用 Otsu 二值化，返回的 retVal 值与设定的阈值相等。

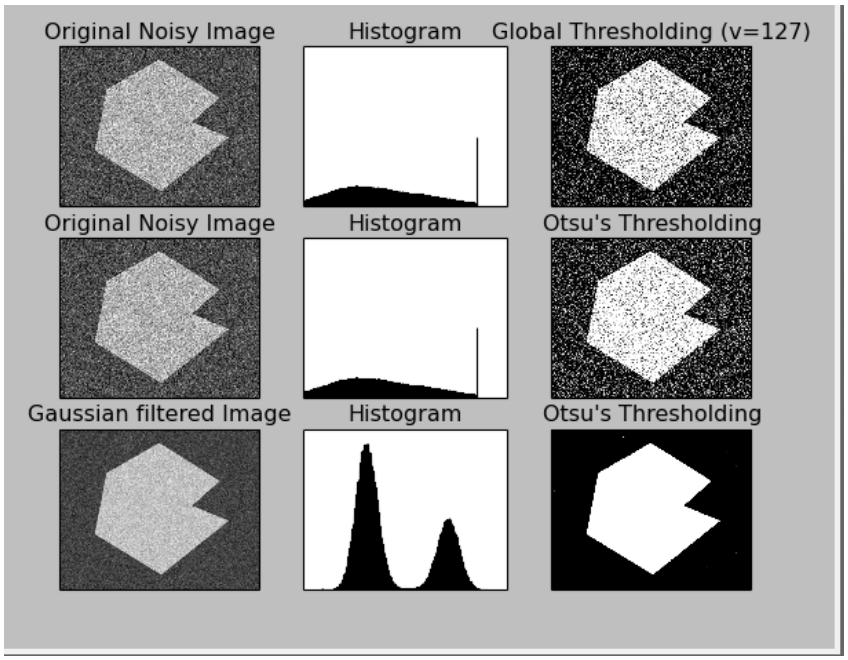
下面的例子中，输入图像是一副带有噪声的图像。第一种方法，我们设 127 为全局阈值。第二种方法，我们直接使用 Otsu 二值化。第三种方法，我们首先使用

一个 5x5 的高斯核除去噪音，然后再使用 Otsu 二值化。看看噪音去除对结果的影响有多大吧。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('noisy.tif',0)

# global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
```

```
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

结果：



3.2.4 Otsu’s 二值化是如何工作的？

Otsu 方法是一种全局化的动态二值化方法，又叫大津法，是一种灰度图像二值化的常用算法。该算法的基本思想是：设使用某一个阈值将灰度图像根据灰度大小分成目标部分和背景部分两类，在这两类的类内方差最小和类间方差最大的时候，得到的阈值是最优的二值化阈值。

在这一部分我们会演示怎样使用 Python 来实现 Otsu 二值化算法，从而告诉大家它是如何工作的。

因为是双峰图，Otsu 算法就是要找到一个阈值（t），使得同一类加权方差最小，需要满足下列关系式：$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$

其中：

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_1(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_1(t)]^2 \frac{P(i)}{q_2(t)}$$

其实就是在两个峰之间找到一个阈值 t，将这两个峰分开，并且使每一个峰内的方差最小。实现这个算法的 Python 代码如下：

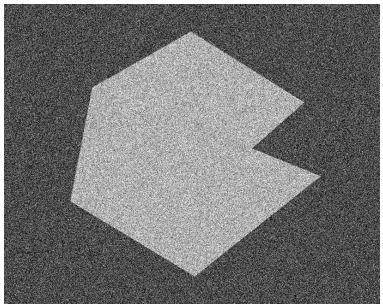
```
import cv2
import numpy as np
img = cv2.imread('noisy.tif',0)
blur = cv2.GaussianBlur(img,(5,5),0)
# find normalized_histogram, and its cumulative distribution function
# 计算归一化直方图
#CalcHist(image, accumulate=0, mask=NULL)
hist = cv2.calcHist([blur],[0],None,[256],[0,256])
hist_norm = hist.ravel()/hist.max()
Q = hist_norm.cumsum()
bins = np.arange(256)
fn_min = np.inf
thresh = -1
for i in range(1,256):
    p1,p2 = np.hsplit(hist_norm,[i]) # probabilities
    q1,q2 = Q[i],Q[255]-Q[i] # cum sum of classes
    b1,b2 = np.hsplit(bins,[i]) # weights
    # finding means and variances
    m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
```

```
        v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
        # calculates the minimization function
        fn = v1*q1 + v2*q2
        if fn < fn_min:
            fn_min = fn
            thresh = i
# find otsu's threshold value with OpenCV function
ret, otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print thresh,ret
```

实验图像

gradient.png

dave.jpg



noisy.tif

3.3 颜色空间转换

3.3.1 转换颜色空间

在 OpenCV 中有超过 150 中进行颜色空间转换的方法。但是你以后就会发现我们经常用到的也就两种：BGR ◄► Gray 和 BGR ◄► HSV。

颜色空间变换函数是：**cv2.cvtColor(input_image，flag)**，其中 flag 就是转换类型。

对于 BGR 和 Gray 的转换，我们要使用的 flag 就是 **cv2.COLOR_BGR2GRAY**。同样对于 BGR 和 HSV 的转换，我们用的 flag 就是 **cv2.COLOR_BGR2HSV**。

还可以通过下面的命令得到所有可用的 flag。

```
import cv2
flags=[i for in dir(cv2) if i startswith('COLOR_')]
print flags
```

注意：在 OpenCV 的 HSV 格式中，H（色彩/色度）的取值范围是[0，179]，S（饱和度）的取值范围[0，255]，V（亮度）的取值范围[0，255]。但是不同的软件使用的值可能不同。所以当需要拿 OpenCV 的 HSV 值与其他软件的 HSV 值进行对比时，一定要记得归一化。

3.3.2 物体跟踪

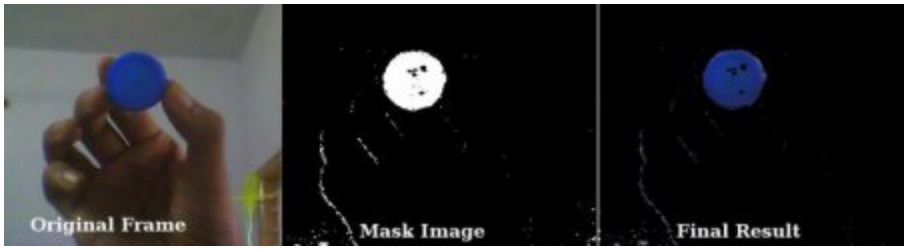
现在我们知道怎样将一幅图像从 BGR 转换到 HSV 了，可以利用这一点来提取带有某个特定颜色的物体。在 HSV 颜色空间中要比在 BGR 空间中更容易表示一个特定颜色。下列程序需要提取的是一个蓝色的物体。步骤如下：

- 从视频中获取每一帧图像
- 将图像转换到 HSV 空间
- 设置 HSV 阈值到蓝色范围。

• 获取蓝色物体，当然我们还可以做其他任何我们想做的事，比如：在蓝色物体周围画一个圈。代码如下：

```
import cv2
import numpy as np
cap=cv2.VideoCapture(0)
while(1):
# 获取每一帧
ret,frame=cap.read()
# 转换到 HSV
hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
# 设定蓝色的阈值
lower_blue=np.array([110,50,50])
upper_blue=np.array([130,255,255])
# 根据阈值构建掩模
mask=cv2.inRange(hsv,lower_blue,upper_blue)
# 对原图像和掩模进行位运算
res=cv2.bitwise_and(frame,frame,mask=mask)
# 显示图像
cv2.imshow('frame',frame)
cv2.imshow('mask',mask)
cv2.imshow('res',res)
k=cv2.waitKey(5)&0xFF
if k==27:
   break
# 关闭窗口
cv2.destroyAllWindows()
```

下图显示了追踪蓝色物体的结果：



注意：图像中仍然有一些噪音。如何使用前面讲的形态学方法消除空洞或多余的小区域?

3.3.3 怎样找到要跟踪对象的 HSV 值？

其实这真的很简单，函数 cv2.cvtColor() 也可以用到这里。但是现在你要传入的参数是（你想要的）BGR 值而不是一副图。例如，我们要找到绿色的 HSV 值，我们只需在终端输入以下命令：

```
import cv2
import numpy as np
green=np.uint8([0,255,0])  %这个地方有错误。
hsv_green=cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
error: /builddir/build/BUILD/opencv-2.4.6.1/
modules/imgproc/src/color.cpp:3541:
error: (-215) (scn == 3 || scn == 4) && (depth == CV_8U || depth == CV_32F)
```

在 cvtColor 函数中。

```
#scn (the number of channels of the source),
#i.e. self.img.channels(), is neither 3 nor 4.
#
#depth (of the source),
#i.e. self.img.depth(), is neither CV_8U nor CV_32F.
# 所以不能用[0,255,0]，而要用[[[0,255,0]]]
# 这里的三层括号应该分别对应于 cvArray，cvMat，IplImage
```

正确的输入方式为：

```
green=np.uint8([[[0,255,0]]])
hsv_green=cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
print hsv_green
[[[60 255 255]]]
```

除了这个方法之外，你可以使用任何其他图像编辑软件（例如 GIMP）或者在线转换软件找到相应的 HSV 值，但是最后别忘了调节 HSV 的范围。

四、 思考题

(1)由于实验条件时的约束，并不是每个人都能进行物体跟踪中的视频操作，所以思考如何实现对图片（图 1）进行物体跟踪中的视频操作（即获取蓝色物体），并进行去噪（膨胀、腐蚀操作），也可实现在蓝色物体周围画一个圈。

(2)请独自完成视频图像中蓝色物体的跟踪。请用矩形框在视频中标出给跟踪的目标。

图 1

资料参考网址：

http://opencv-python-tutroals.readthedocs.org/en/latest/index.html