

Detecção de objetos coloridos

June 17, 2018

1 -----

2 Introdução

Vimos que no espaço de cor HSV, o canal H (*hue*, matiz) apresenta certa independência em relação à iluminação de um objeto. Essa característica pode ser explorada para a detecção de objetos. Teste o código seguinte.

```
In [ ]: # Adaptado de http://docs.opencv.org/3.2.0/d9d/tutorial_py_colorspaces.html
import numpy as np
import cv2

frame = cv2.imread("imgOut_frame_2_bUp.jpg") #obtida de http://blogs.mathworks.com/image

# Convert BGR to HSV
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

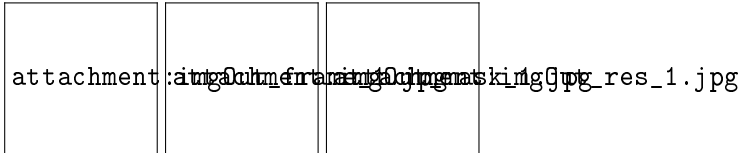
# -----
# define range of blue color in HSV
blue=100 #importante: faixa da componente H no OpenCV: [0, 179]
thres=100
lower_blue = np.array([blue-thres,50,50])
upper_blue = np.array([blue+thres,255,255])

# -----
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv, lower_blue, upper_blue)
# Bitwise-AND mask and original image
res = cv2.bitwise_and(frame,frame, mask= mask)

# -----
cv2.imshow('frame',frame)
cv2.imshow('mask',mask)
cv2.imshow('res',res)
```

```
cv2.imwrite( "imgOut_frame_1.jpg", frame);
cv2.imwrite('imgOut_mask_1.jpg', mask)
cv2.imwrite( "imgOut_res_1.jpg", res);
```

```
# -----
cv2.waitKey(0)
cv2.destroyAllWindows()
```



1. Na função *cvtColor*, que outras conversões estão disponíveis?

RGB -> HSV (CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB)
https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor

2. O que faz a função *inRange*?

Atribui o valor branco aos píxeis de uma matriz com índices iguais aos píxeis de uma imagem analisada no qual os valores HSV estejam dentro de uma faixa definida. Para os demais píxeis da matriz é atribuído valor zero.

Agora, vamos testar o mesmo procedimento usando um sinal de vídeo (webcam).

```
In [ ]: import cv2
import numpy as np
```

```
# -----
color_to_find=90 #importante: faixa da componente H no OpenCV: [0, 179]
thres=20
cap = cv2.VideoCapture(0)
```

```
while(1):
    # Take each frame
    _, frame = cap.read()
    # Convert BGR to HSV

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # define range of blue color in HSV
    lower_blue = np.array([color_to_find-thres,50,50])
    upper_blue = np.array([color_to_find+thres,255,255])
```

```
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv, lower_blue, upper_blue)
# Bitwise-AND mask and original image
res = cv2.bitwise_and(frame,frame, mask= mask)
```

```
# -----
```

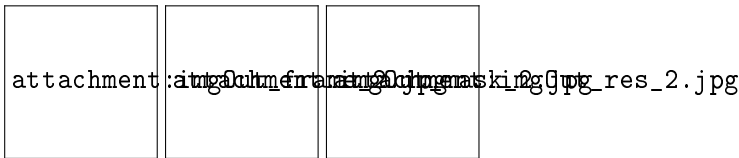
```

cv2.imshow('frame',frame)
cv2.imshow('mask',mask)
cv2.imshow('res',res)

# -----
k = cv2.waitKey(5)
if k == 27:
    cv2.imwrite( "imgOut_frame_2.jpg", frame);
    cv2.imwrite('imgOut_mask_2.jpg', mask)
    cv2.imwrite( "imgOut_res_2.jpg", res);
    print("Frame saved\n")
    break

# -----
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()

```



Você deve ter observado que há muito ruído na máscara. Uma forma de fazer uma "limpeza" é através de operações morfológicas.

3 -----

4 Operações morfológicas

São operações baseadas no formato da imagem. Usam um *kernel* ou elemento estruturante que define a operação. Os operadores básicos são erosão e dilatação.

4.1 Erosão

A ideia básica da erosão é retirar os pixels que estão nas bordas do objeto (objeto é o que está em branco, ou seja, com intensidade 1).

O elemento estruturante desliza sobre a imagem, como na convolução 2D. Um pixel na imagem resultante será 1 se todos os pixels no kernel forem 1, caso contrário, será erodido (intensidade 0).

Desse modo, os pixels próximos às bordas serão descartados. Quanto? Depende do tamanho do kernel. Essa operação é útil, por exemplo, para remover ruídos isolados de pequeno tamanho e separar dois objetos conectados.

Há vários exemplos em <http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm>

Observe o exemplo com um elemento estruturante 5x5.

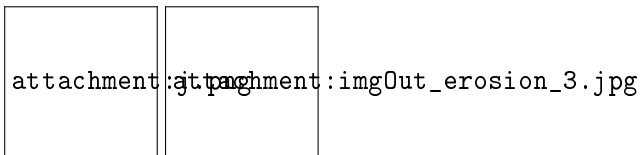
```
In [1]: #Exemplo de http://docs.opencv.org/3.2.0/d9/d61/tutorial_py_morphological_ops.html
import cv2
import numpy as np

img = cv2.imread('j.png',0)
cv2.imshow('imgJ', img)

# -----
kernel = np.ones((5, 5), np.uint8)
erosion = cv2.erode(img, kernel, iterations = 1)

cv2.imshow('res',erosion)
cv2.imwrite( "imgOut_erosion_3.jpg", erosion)

# -----
cv2.waitKey(0)
cv2.destroyAllWindows()
```



4.2 Dilatação

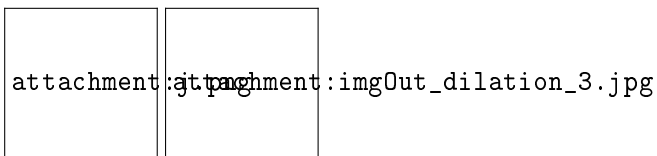
É o oposto da erosão. Aqui, um pixel resultado será 1 se pelo menos um dos pixels sob análise for 1. Desse modo, há um aumento do tamanho do objeto (aumento da região branca).

Dilatação é comumente aplicada após uma operação de erosão para redução de ruído. Também é útil para unir partes de um objeto.

```
In [2]: # -----
dilation = cv2.dilate(img,kernel,iterations = 1)

cv2.imshow('res',dilation)
cv2.imwrite( "imgOut_dilation_3.jpg", dilation)

# -----
cv2.waitKey(0)
cv2.destroyAllWindows()
```



4.3 Abertura e fechamento

São operações que usam erosão e dilatação.

5.1 CandyCounter

Contador de doces de determinada cor:

```
In [1]: # -----
        # Candy Counter

        # Adaptado de http://docs.opencv.org/3.2.0/d9d/tutorial_py_colorspaces.html
        import numpy as np
        import cv2

        # img_BGR = cv2.imread("mms.jpg") #obtida de http://blogs.mathworks.com/images/steve/201
        img_BGR = cv2.imread("balas_s.png")
        img_buffer = img_BGR

        # Convert BGR to HSV
        img_HSV = cv2.cvtColor(img_buffer, cv2.COLOR_BGR2HSV)
        # -----

        # define range of blue color in HSV
        # importante: faixa da componente H no OpenCV: [0, 179]

        red = 0

        green = red + 60
        if green > 179:
            green = green - 180

        blue = green + 60
        if blue > 179:
            blue = blue - 180

        # -----
        thres=34

        lower_blue = np.array([blue-thres, 50, 50])
        upper_blue = np.array([blue+thres, 255, 255])

        lower_green = np.array([green-thres, 50, 50])
        upper_green = np.array([green+thres, 255, 255])

        lower_red = np.array([red-thres, 50, 50])
        upper_red = np.array([red+thres, 255, 255])

        # Threshold the HSV image to get only blue colors
```

```

mask_blue = cv2.inRange(img_HSV, lower_blue, upper_blue)
mask_green = cv2.inRange(img_HSV, lower_green, upper_green)
mask_red = cv2.inRange(img_HSV, lower_red, upper_red)

# mask = mask_blue;
mask = mask_green;
# mask = mask_red;

# -----
kernel = np.ones((5,5), np.uint8)
mask = cv2.erode(mask, kernel, iterations = 3)
kernel = np.ones((3,3), np.uint8)
mask = cv2.dilate(mask, kernel, iterations = 1)

# Bitwise-AND mask and original image
imgRes = cv2.bitwise_and(img_BGR, img_BGR, mask)

# -----
# find all blobs and label them
n, labels, stats, centroids = cv2.connectedComponentsWithStats(mask)
print centroids
print stats
font = cv2.FONT_HERSHEY_SIMPLEX

for n in range(1, len(stats)): #o primeiro (índice 0) é a imagem toda, então, desconsidere
    cv2.rectangle(imgRes, (stats[n][0], stats[n][1]), (stats[n][0]+stats[n][2], stats[n][1]+stats[n][3]),
    cv2.putText(imgRes, str(n), (stats[n][0], stats[n][1]), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.circle(imgRes, (int(centroids[n][0]), int(centroids[n][1])), 3, (0, 0, 255), -1)

cv2.putText(imgRes, 'Total: '+str(n), (0, stats[0][3]), font, 1, (255, 255, 255), 2, cv2.LINE_AA)

# -----
# cv2.imshow('img_BGR', img_BGR)
cv2.imshow('img_buffer', img_buffer)
cv2.imshow('mask', mask)
cv2.imshow('imgRes', imgRes)

cv2.imwrite( "imgOut_BGR_4.jpg", img_BGR);
cv2.imwrite('img_buffer_4.jpg', img_buffer)
cv2.imwrite( "imgOut_mask_4.jpg", mask);
cv2.imwrite( "imgOut_imgRes_4.jpg", imgRes);
# -----
cv2.waitKey(0)
cv2.destroyAllWindows()

```

```

[[249.66842271 249.09780387]
 [156.98      66.38      ]
 [115.008     88.792     ]

```

```

[ 80.94736842  86.11578947]
[170.73584906  96.13207547]
[335.          95.         ]
[389.8         125.56       ]
[ 68.36065574 128.22131148]
[ 83.80246914 142.95061728]
[140.44444444 147.96296296]
[206.38983051 158.08474576]
[291.125       175.95833333]
[334.62121212 190.15151515]
[141.47826087 207.32608696]
[375.62295082 210.62295082]
[398.05084746 306.74576271]
[183.48623853 337.19266055]
[306.87640449 352.84269663]
[151.94915254 355.96610169]
[129.93333333 364.62962963]
[308.3125      372.6484375 ]
[438.8372093   378.18604651]
[389.35433071 386.07086614]
[116.73076923 402.5        ]
[146.44        425.74666667]
[356.656       425.736      ]
[398.04195804 447.14685315]
[346.1954023   453.44827586]]
[[ 0      0      500      500 247526]
 [ 152    62     11     10    50]
 [ 106    81     16     15   125]
 [  74    82     15      9    95]
 [ 167    91      8     11    53]
 [ 334    94      3      3     9]
 [ 388   123      5      6    25]
 [  60   124     16     11   122]
 [  80   135     10     15    81]
 [ 138   145      6      7    27]
 [ 203   152      8     13    59]
 [ 285   170     13     13   120]
 [ 330   186     10      9    66]
 [ 134   204     15      8    92]
 [ 372   206      8     11    61]
 [ 392   302     13     10    59]
 [ 178   331     12     12   109]
 [ 301   346     11     13    89]
 [ 146   350     12     13   118]
 [ 124   358     12     14   135]
 [ 302   361     12     21   128]
 [ 436   371      7     15    86]
 [ 382   381     16     11   127]

```

attachment

Page 10 of 10

skinAggImg_imgRes_4.jpg