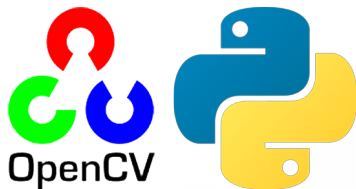


Processamento Digital de Imagens Usando OpenCV + Python



Adilmar C. Dantas - Doutorando em Ciência da Computação

Universidade Federal de Uberlândia - Faculdade de Computação (FACOM)

20 de abril de 2019

Sumário

- 1 Introdução
 - Conceitos
 - Preparando o Ambiente
- 2 Processamento de Imagens e Vídeos
 - Conversão em escala de cinza - Imagem
 - Capturando frames de vídeo pela webcam
 - Manipulando Vídeos - Escrever
 - Desenhando, escrevendo em imagens
 - Operações em imagens
 - Detecção de Bordas e Gradientes
 - Filtro de cores OpenCV
 - Transformações Morfológicas
 - Correspondência de Modelos OpenCV
 - Detecção de ROI's (face e olhos) usando Haar-Cascade
- 3 Referencias

Introdução

- **Processamento de imagens:** é qualquer forma de processamento de dados no qual a entrada e saída são imagens tais como fotografias ou quadros de vídeo;
- **OpenCV:** é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão computacional;
- **Python:** é uma linguagem de programação de alto nível, interpretada, de script, orientada a objetos.

Passo 1: instalando o Python

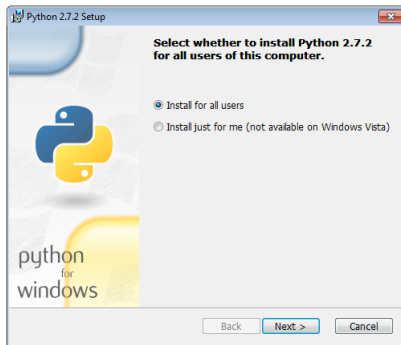


Figura: Instalação do Python

O primeiro passo a ser realizado é a instalação do Python versão 3.5. **Link Python - Windows X64**

Passo 2: instalando o OpenCV

Para instalar o OpenCV iremos utilizar o instalador de pacotes do próprio Python, executando os passos a seguir.

- Navegue pelo CMD até o diretório: **c:/Python3.5/Scripts/**
- Execute o comando **pip install opencv-python**
- Execute o comando **pip install numpy**
- Execute o comando **pip install matplotlib**

Pronto OpenCV e suas dependências instalar, agora vamos aos códigos.

Conversão em escala de cinza

Converter algo para escala de cinza e substituir a presença de cor por branco e preto variando suas intensidade na imagem de acordo com o padrão de cores.

- O primeiro passo é importa as dependências, logo em seguida importar a imagem a ser convertida **img** como **cv2.read (arquivo de imagem, parms)**.
- O padrão será **IMREAD_COLOR**, que é cor sem nenhum canal alfa. Se você não estiver familiarizado, alfa é o grau de opacidade (o oposto da transparência). Se você precisar manter o canal alfa, também poderá usar **IMREAD_UNCHANGED**.

- Para o segundo parâmetro, você pode usar -1, 0 ou 1. A cor é 1, a escala de cinza é 0 e o inalterado é -1. Assim, para tons de cinza, basta fazer **img = cv2.imread ('images/01.jpg', 0)**.
- Uma vez carregado, usamos **cv2.imshow (title, image)** para mostrar a imagem. A partir daqui, usamos o **cv2.waitKey(0)** para esperar até que qualquer tecla seja pressionada. Feito isso, usamos **cv2.destroyAllWindows()** para fechar tudo.

Conversão em escala de cinza

Nessa etapa iremos aprender como carregar uma **imagem** e convertê-la em escala de cinza.

```
exer1.py ●  
  
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('images/01.jpg',cv2.IMREAD_GRAYSCALE)  
cv2.imshow('image',img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```




Figura: Carregando e convertendo uma imagem para escala de cinza.

Conversão em escala de cinza - Vídeo

Agora ao invés de carregamos uma imagem iremos carregar um stream da webcam, veja como:

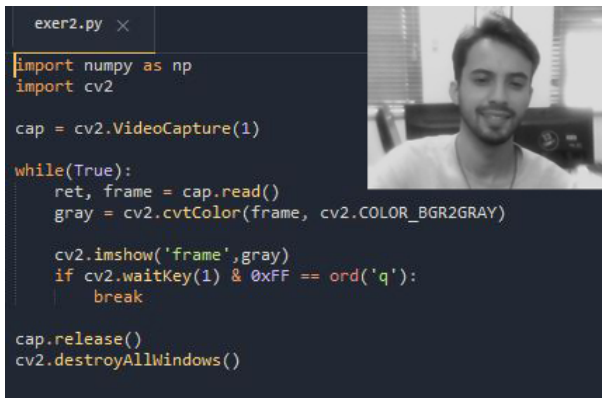


Figura: Carregando e convertendo stream da webcam para escala de cinza.

Conversão em escala de cinza - Vídeo

Nesse exemplo agora iremos realizar as mesmas operações da imagem, porem em streams de video da webcam.

- Nessa código definimos uma nova variável para a conversão em escla de cinza **BGR2GRAY**. É importante observar que o OpenCV lê as cores como BGR (Blue Green Red), onde a maioria dos aplicativos de computador são lidos como RGB (Red Green Blue). Lembre-se disso.

Manipulando vídeos

Nesse exercício iremos pegar o stream da câmera, conforme o exercício anterior e grava-lo em uma pasta no formato .avi, seguindo os seguintes passos.

- Primeiramente devemos passar o codec de vídeo, utilize **`cv2.VideoWriter_fourcc(*'XVID')`**.
- Algumas informações como dimensão, etc:
`cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))`

Manipulando vídeos

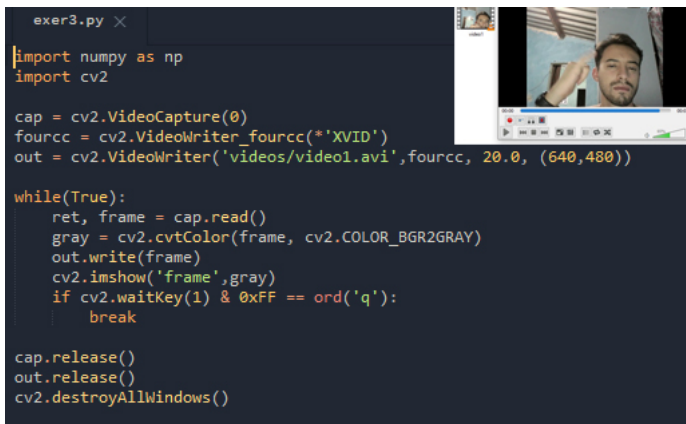


Figura: Salvando stream de vídeo em arquivo .AVI

Desenhando, escrevendo em imagens

Nessa parte abordaremos como desenhar vários tipos de formas em suas imagens e vídeos. É bastante comum querer marcar objetos detectados de alguma forma, como veremos mais a seguir.

Para esse exercício iremos desenhar em uma imagem os seguintes elementos: uma linha, um retângulo, um círculo, pelignos e por fim escrever um texto.

Desenhando, escrevendo em imagens

Desenhando, escrevendo em imagens

```
exer4.py X
import numpy as np
import cv2

img = cv2.imread('images/01.jpg',cv2.IMREAD_COLOR)
#cv2.line(img,(0,0),(200,300),(255,255,255),50)
#cv2.rectangle(img,(500,250),(1000,500),(0,0,255),15)
cv2.circle(img,(280,220), 30, (0,255,0), -1)
#pts = np.array([[100,50],[200,300],[700,200],[500,100]], np.int32)
#pts = pts.reshape((-1,1,2))
#cv2.polylines(img, [pts], True, (0,255,255), 3)
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'UFU',(5,500), font, 3, (255,0,0), 13, cv2.LINE_AA)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Figura: Desenhando e escrevendo em imagens com Python e OpenCV.

Operações em imagens

Nessa parte abordaremos alguns dos princípios básicos das **operações em imagem**. Todo vídeo se divide em quadros. Cada quadro, como uma imagem, divide-se em **pixels** armazenados em **linhas e colunas** dentro do quadro / imagem. Cada pixel tem uma localização de coordenadas e cada pixel é composto por valores de cores. Vamos elaborar alguns exemplos de acesso e manipulação a esses bits.

Operação de corte (crop)

```
exer5.py X
import cv2
import numpy as np

img = cv2.imread('images/01.jpg',cv2.IMREAD_COLOR)

#capture pixel
px = img[55,55]

#modify pixel
img[55,55] = [255,255,255]

#modify pixel
px = img[55,55]
print(px)

px = img[100:150,100:150]
print(px)

#set white pixel matrix
img[100:150,100:150] = [255,255,255]

print(img.shape)
print(img.size)
print(img.dtype)

#crop image
crop = img[37:111,107:194]
img[0:74,0:87] = crop
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Operações em imagens

Nesse exercício iremos aprender algumas operações aritméticas e de lógica importantes e que podem fazer muita diferença no seu projeto. Para esse exercício usaremos de base as duas imagens (p_1 , p_2). O primeiro passo que iremos fazer é uma simples **adição**.

Operações aritméticas em imagens

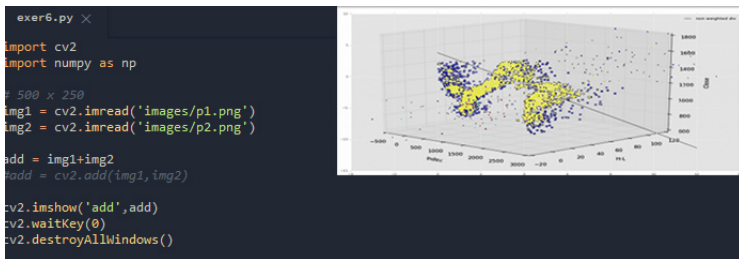


Figura: Somando duas imagens usando OpenCV.

Operações aritméticas em imagens

Certamente o resultado não ficou bom, mas podemos melhorar, trocando a operação de adição anterior por está: **cv2.add(img1,img2)**.

```
import cv2
import numpy as np

# 500 x 250
img1 = cv2.imread('images/p1.png')
img2 = cv2.imread('images/p2.png')

#add = img1+img2
add = cv2.add(img1,img2)

cv2.imshow('add',add)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Figura: Somando duas imagens usando OpenCV.

Operações aritméticas em imagens

Novamente o resultado não é dos melhores, isso ocorre devido ao fato de que a escala de cores é **0-255** onde 255 é “Branco - luz” e 0 preto. Assim, por exemplo, $(50, 170, 200) = 205, 381, 279$ normalizando teremos $(205, 255, 255)$. Para contornar esse problema podemos ponderar as imagens, veja.

Operações aritméticas em imagens



Figura: Somando e normalizando duas imagens usando OpenCV.

Thresholding OpenCV Python

Para tentar interpretar uma imagem com baixa iluminação, convertê-la em escalas de cinza usando limiares parece uma boa estratégia, mas para aperfeiçoar ainda mais nosso resultado podemos usar o *Thresholding*. Essa operação permite criar **imagens binárias** a partir de imagens em escala de cinza, conforme veremos a seguir.

Thresholding OpenCV Python

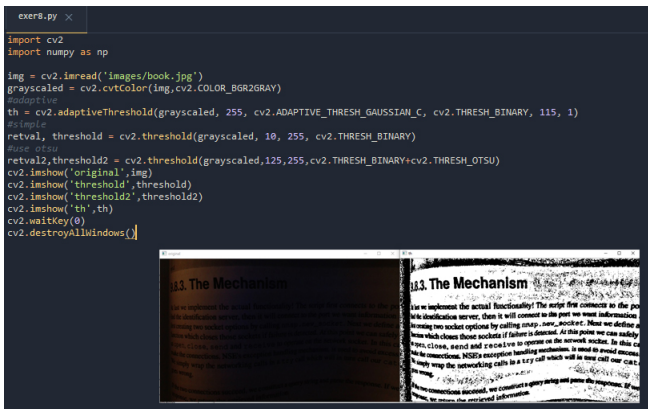


Figura: Exemplo de Thresholding adaptativo.

Detecção de Bordas e Gradientes

A detecção de gradientes é utilizado geralmente para medir intensidades em imagens e vídeos. Enquanto a detecção de bordas faz exatamente o que o nome diz, podendo ser utilizado ate mesmo para reconhecer, demarcar objetos.

Detecção de Bordas e Gradientes

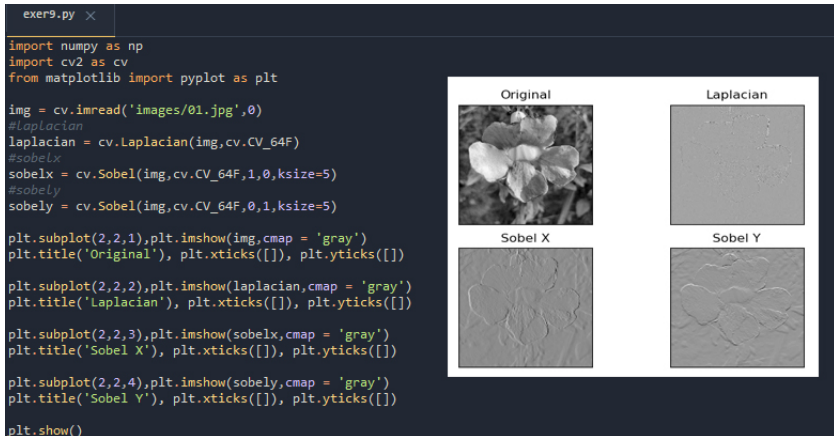


Figura: Gradientes em imagens (Laplacian, Sobel X, Sobel Y).

Detecção de Bordas e Gradientes

Você certamente está se perguntando o que é esse **cv2.CV64F**, essa informação é o tipo de dados do *grid* que no nosso caso é 5x5. Embora possamos usar os gradientes para convertê-los em arestas puras, podemos usar a função de bordas para fazer isso por nós com mais eficiência, como veremos a seguir.

Detecção de Bordas e Gradientes

```
exer10.py x
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask= mask)

    cv2.imshow('Original',frame)
    edges = cv2.Canny(frame,100,200)
    cv2.imshow('Edges',edges)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

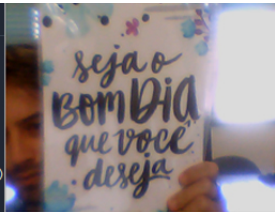


Figura: Detecção de bordas com OpenCV.

Filtro de cores OpenCV

Nessa parte do curso abordaremos como criar uma espécie de filtro, revisitando as operações bit a bit, onde filtraremos especificamente uma determinada cor, e então mostrá-la para o usuário. Para consultar as principais cores no padrão HSV iremos acessar este link **Cores no padrão HSV**.

Filtro de cores OpenCV

```
exer11.py x
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

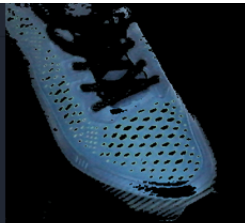
    lower_blue = np.array([61,68,61])
    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    mask = cv2.inRange(hsv, lower_blue, upper_red)
    res = cv2.bitwise_and(frame,frame, mask= mask)

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

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```



Transformações Morfológicas

Transformações morfológicas são operações que podemos executar em imagens com varias finalidades, como por exemplo, para realçar uma determinada área.

Esse tipo de operação geralmente é feita em pares, para esse exemplo usaremos as seguintes técnicas:

- **Erosão:** O processo de erosão ocorre nas bordas, para isso criamos um *grid* 5 x 5 que varre a imagem e se todos os pixel's forem brancos seta branco caso contrario preto;
- **Dilatação:** Na dilatação ocorre o inverso, os pixel's são substituídos por preto.

Transformações Morfológicas

```
exer12.py X
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([60,70,60])
    upper_red = np.array([255,255,180])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask= mask)

    kernel = np.ones((5,5),np.uint8)
    erosion = cv2.erode(mask,kernel,iterations = 1)
    dilation = cv2.dilate(mask,kernel,iterations = 1)

    cv2.imshow('Original',frame)
    cv2.imshow('Mask',mask)
    cv2.imshow('Erosion',erosion)
    cv2.imshow('Dilation',dilation)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```



Correspondência de Modelos OpenCV

Nessa etapa do curso iremos abordar uma versão bastante básica de reconhecimento de objetos, a correspondência de modelos. A ideia aqui é encontrar regiões idênticas de uma imagem que correspondam a um modelo que fornecemos, dando um certo limite, ou seja, dado um recorte de uma imagem encontrar esse recorte em qualquer imagem.

Correspondência de Modelos OpenCV

```
exer13.py x
import cv2
import numpy as np

img_rgb = cv2.imread('images/coins.jpg')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

template = cv2.imread('images/coin.jpg',0)
w, h = template.shape[::-1]

res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)

for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)

cv2.imshow('Detected',img_rgb)
```



Figura: Correspondência de modelos usando Python e OpenCV.

Detecção de ROI's (face e olhos) usando Haar-Cascade

Para detectar regiões de interesse (ROI's) em imagens existem atualmente diversas técnicas, dentre elas uma bastante usada e bem sucedida é a utilização da técnica Haar-Cascade. Essa técnica consiste em utilizar a integral de imagens combinada com algoritmos de otimização como o Adabost (efeito cascata) para detectar rapidamente regiões de interesse em imagens [2].

Detecção de ROI's (face e olhos) usando Haar-Cascade

Para começar o algoritmo os seguintes passos devem ser seguidos:

- Treinar um classificador em cascata ou utilizar algo pronto, pois esse processo demanda tempo e uma grande quantidade de imagens positivas e negativas;
- Criar o frame de entrada da informação (imagem ou vídeo);
- Chamar o classificador para varrer a imagem;
- Demarcar as regiões identificadas.

Detecção de ROI's (face e olhos) usando Haar-Cascade

Para o nosso curso eu mesmo providenciei os arquivos xml (classificador em cascata) para vocês no **Link**. Um dos arquivos é responsável por detectar faces (frontais) e o outro por detectar a região dos olhos. Feito isso devemos carregar nossos arquivos no script a procura de regiões de interesse.

Detecção de ROI's (face e olhos) usando Haar-Cascade

Detecção de ROI's (face e olhos) usando Haar-Cascade

```
face.py X
import numpy as np
import cv2

# multiple cascades: https://github.com/Itseez/opencv/tree/master/data/haarcascades

face_cascade = cv2.CascadeClassifier('haar/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haar/haarcascade_eye.xml')

cap = cv2.VideoCapture(0)

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

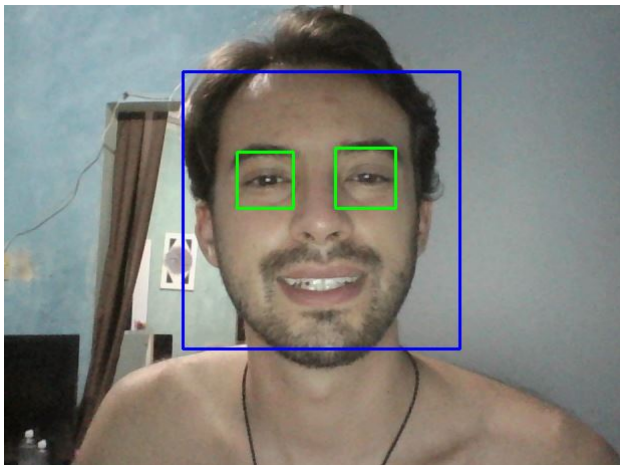
        eyes = eye_cascade.detectMultiScale(roi_color)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0), 2)

    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Detecção de ROI's (face e olhos) usando Haar-Cascade

Detecção de ROI's (face e olhos) usando Haar-Cascade



Esse curso usou como referência o seguinte livro [1].

Referencias I



J. Howse.

OpenCV computer vision with python.

Packt Publishing Ltd, 2013.



P. Viola and M. Jones.

Rapid object detection using a boosted cascade of simple features.

In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I-I. IEEE, 2001.

Contatos e social



WebSite: www.adilmar.com.br

E-mail: akanehar@gmail.com

Lattes: <http://lattes.cnpq.br/2462384793631673>

FB/IG @adilmarcoelho

