INF209B - Tópicos Especiais em Processamento de Sinais: Visão Computacional

Prática 02 - Experimentos Básicos de Processamento de Imagens - v01

Prof. Filipe Ieda Fazanaro

Sala 929 - Bloco B - filipe.fazanaro@ufabc.edu.br

12 de março de 2020

CONSIDERAÇÕES GERAIS	
Data limite para entrega do relatório:	**22/03/2020**
Entrega do relatório após data limite:	**Rel01 = 0,0**
Nota final dessa prática:	**R1 = Rel01**

Descrição

Este laboratório está dividido em 05 partes. Leia atentamente as instruções, execute os códigos de demonstração e elabore os programas solicitados nos exercícios descritos ao final.

Atividades em Aula - Parte 01: Operações Básicas

Objetivos:

- Aprender como acessar os valores de pixel e como modificá-los;
- Aprender como acessar as propriedades da imagem;
- Compreender a definição de Região de Interesse (ROI, do inglês *Region Of Interest*);
- Aprender a dividir/separar e mesclar imagens.

Observações:

1. Quase todas as operações nesta seção estão relacionadas principalmente à biblioteca `Numpy` ao invés da OpenCV. Essa biblioteca foi instalada durante o procedimento de instalação do OpenCV descrito no apêndice da prática 01.
2. É interessante que você leia com atenção os comentários relacionados aos comandos na sequência em que são apresentados. Somente após isso ter sido feito é que você deverá executar o código exemplo.
3. Os arquivos de imagens foram disponibilizados juntamente com esse roteiro experimental.

(a) **Acessando e Modificando Valores de Pixels:**

Primeiramente, deve-se carregar uma imagem colorida:

```
import numpy as np
import cv2 as cv

img = cv.imread('messi5.jpg')

cv.imshow('Original image',img)
cv.waitKey(0)
```

É possível acessar um valor de pixel por suas coordenadas de linha e coluna. Para a imagem BGR, ela retorna uma matriz de valores Azul, Verde e Vermelho. Para imagens em escala de cinza, apenas a intensidade correspondente é retornada:

```
## Acessando e modificando os valores dos pixels ##
px = img[100,100]
print( px )
# [157 166 200]

print("px[100,100,0] = ")
blue = img[100,100,0]
print( blue )
# 157
```

É possível acessar os valores de um determinado pixel da mesma maneira:

```
img[100,100] = [255,255,255]
print( img[100,100] )
# [255 255 255]
```

Nota: Para o acesso individual ao pixel, os métodos de matriz do `Numpy`, `array.item()` e `array.itemset()` são considerados melhores, mas sempre retornam um escalar. Se você quiser acessar todos os valores B, G, R, você precisa chamar `array.item()` separadamente para todos.

```
## Acessando um pixel da camada RED
red = img.item(10,11,2)
print( red )

## Modificando o valor do pixel
img.itemset((10,11,2), 100)
red = img.item(10,11,2)
print( red )
```

(b) **Acessando as Propriedades da Imagem:**

As propriedades da imagem incluem o número de linhas, de colunas e de canais (de cores), tipo de dados da imagem, número de pixels, etc.

O formato de uma imagem é acessado pelo comando `img.shape`. Ele retorna uma tupla de número de linhas, colunas e canais (se a imagem for colorida):

```
## Acessando as propriedades da imagem ##
print( img.shape )
# (342, 548, 3)
```

O número total de pixels é acessado por `img.size`:

```
print( img.size )
# 562248
```

O tipo de dados da imagem é obtido por `img.dtype`:

```
print( img.dtype )
# uint8
```

(c) **Acessando uma Região de Interesse (ROI) da Imagem:**

Em algumas situações, é necessário trabalhar com determinada região de imagens. Por exemplo, para a detecção de olhos em imagens, obter a região referente ao rosto é feita em toda a imagem. Quando uma face é identificada, a região da face sozinha é selecionada e, em seguida, busca-se pelos olhos dentro dela, ao invés de procurar na imagem inteira. Esse procedimento melhora a precisão (porque os olhos estão sempre nas faces) e o desempenho (porque a pesquisa é feita em uma área pequena).

A ROI é obtido novamente usando a indexação do Numpy. Aqui é feita a seleção da bola e, em seguida, ela é copiada para outra região da imagem:

```
## ROI ##
ball = img[280:340, 330:390]
img[273:333, 100:160] = ball
```

(d) **Separando e Mesclando os Canais de Cores da Imagem:**

Em determinadas situações, torna-se necessário trabalhar separadamente nos canais B, G, R da imagem. Nesse caso, as imagens do BGR são divididas em canais únicos. Em outros casos, talvez seja necessário associar esses canais individuais a uma imagem BGR.

Para separar os canais de cores, os seguintes comandos podem ser empregados:	`## Separando e misturando os canais de cores da imagem ##`
	`b,g,r = cv.split(img)`
	`img = cv.merge((b,g,r))`

Ou ainda:	`b = img[:,:,0]`
Suponha que seja de interesse definir todos os pixels vermelhos para zero. Nesse caso, não é necessário dividir os canais primeiro. A indexação Numpy é mais rápida:	`img[:,:,2] = 0`

Atenção: é importante salientar que `cv.split()` é uma operação dispendiosa (em termos de tempo de computação). Assim sendo, faça isso apenas se for muito importante ao contexto do seu problema. A indexação do Numpy tende a ser uma solução mais interessante.

(e) **Exemplo para executar em aula:**

```
import numpy as np
import cv2 as cv
import os # usado para indicar diretorios especificos

path = './'

img = cv.imread('messi5.jpg')
cv.imshow('Original image',img)
cv.waitKey(0)

## Acessando e modificando os valores dos pixels ##
px = img[100,100]
print( px )
# [157 166 200]

print("px[100,100,0] = ")
blue = img[100,100,0]
print( blue )
# 157

img[100,100] = [255,255,255]
print( img[100,100] )
# [255 255 255]

## Acessando um pixel da camada RED
red = img.item(10,11,2)
print( red )

## Modificando o valor do pixel
img.itemset((10,11,2), 100)
red = img.item(10,11,2)
print( red )

## Acessando as propriedades da imagem ##
print( img.shape )
# (342, 548, 3)

print( img.size )
# 562248

print( img.dtype )
# uint8

## ROI ##
ball = img[280:340, 330:390]
img[273:333, 100:160] = ball

cv.imwrite(os.path.join(path,'messi_bola.png'),img)
cv.imshow('ROI',img)
cv.waitKey(0)

## Separando e misturando os canais de cores da imagem ##
b,g,r = cv.split(img)
img = cv.merge((b,g,r))

#b = img[:,:,0]

cv.imshow('Blue channel',b)
cv.waitKey(0)

img[:,:,2] = 0

cv.imshow('image minus Red',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Atividades em Aula - Parte 02: Operações Aritméticas e Lógicas em Imagens

Objetivos:

- Aprender os procedimentos para realização de operações aritméticas em imagens, tais como adição e subtração;
- Aprender os procedimentos para a realização de operações lógicas em imagens, tais como bit a bit;
- Estudar a sintaxe e o uso de funções específicas, tais como `cv.add()`, `cv.addWeighted()`, etc.

(a) **Adição de Imagens:**

Pode-se adicionar duas imagens pela função OpenCV, `cv.add()` ou simplesmente por uma operação Numpy, `res = img1 + img2`. Ambas as imagens devem ter as mesmas dimensões, profundidade e tipo, ou a segunda imagem pode ser apenas um valor escalar.

Há uma diferença entre a adição do OpenCV e a adição do Numpy: a primeira é uma operação saturada, enquanto que a segunda é uma operação de módulo. O exemplo apresentado na Figura 1 ilustra essas diferenças.

```
1  import numpy as np
2  import cv2 as cv
3
4  x = np.uint8([250])
5  y = np.uint8([10 ])
6
7  # 250 + 10 = 260 -> 255
8  print( cv.add(x,y) )
9  # [[255]]
10
11 # 250 + 10 = 260 % 256 -> 4
12 print( x+y )
13 # [4]
```

Figura 1: Exemplo dos diferentes resultados obtidos quando se realiza a soma empregando funções OpenCV e Numpy.

Em se tratando de operações com imagens, o função OpenCV retorna resultados melhores. Por que? Reflita sobre isso.

(b) **Mistura de Imagens:**

A mistura de imagens nada mais é do que uma soma ponderada entre imagens, i.e., diferentes pesos são dados às imagens (i.e. aos seus pixels). O resultado é uma sensação de mistura ou transparência. Em geral, as imagens são adicionadas conforme a equação (1):

$$\mathrm{I}_{\text{saida}}(x, y) = (1 - \alpha)\mathrm{I}_0(x, y) + \alpha\mathrm{I}_1(x, y) \tag{1}$$

onde $\mathrm{I}_{\text{saida}}$ é a imagem de saída misturada. I_0 e I_1 são as imagens de entrada. (x, y) é a posição espacial do pixel na imagem. Note que variando α entre 0 e 1 é possível obter um efeito de transição de uma imagem para outra.

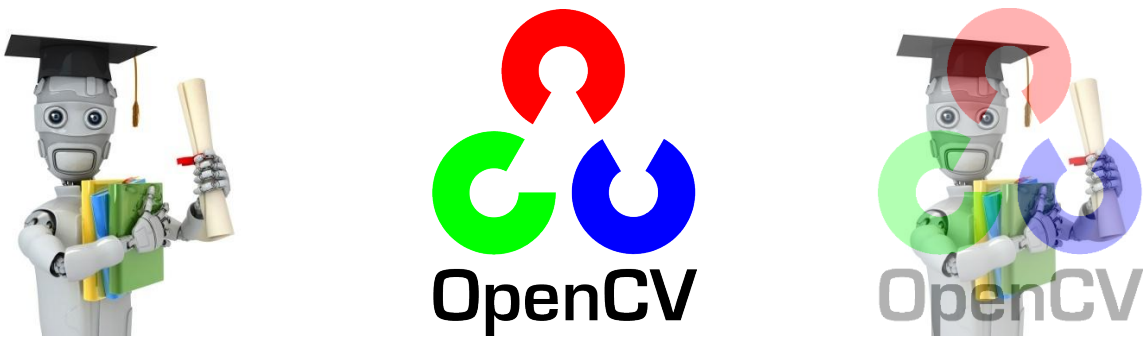
A aplicação da função `cv.addWeight()` respeita a equação (2),

$$\text{dst} = \alpha\mathrm{I}_0 + \beta\mathrm{I}_1 + \gamma, \tag{2}$$

e pode-se escolher $\alpha = 0,7$, $\beta = 0,3$ e $\gamma = 0,0$. O código apresentado na Figura 2 ilustra a aplicação dessa função e na Figura 3 tem-se o resultado esperado.

```
1  import numpy as np
2  import cv2 as cv
3
4  img1 = cv.imread('ml.png')
5  img2 = cv.imread('opencv-logo.png')
6
7  dst = cv.addWeighted(img1,0.7,img2,0.3,0)
8
9  cv.imshow('dst',dst)
10 cv.waitKey(0)
11 cv.destroyAllWindows()
```

Figura 2: Código em Python que executa a mistura (soma ponderada) entre duas imagens empregando a função `cv.addWeight()`.



(a) (b) (c)

Figura 3: Em (a) e (b) tem-se as imagens de entrada. Em (c) tem-se a imagem misturada de saída.

(c) **Operações Bitwise:**

Operações bitwise ou operações lógicas incluem operações bit a bit, tais como AND, OR, NOT e XOR. Essas operações serão bastante úteis para extrair qualquer parte da imagem, definindo e trabalhando com ROI não retangular, etc. A seguir, é considerado um exemplo de como alterar uma região específica de uma imagem.

O código apresentado na Figura 4 faz com que o logo OpenCV seja posicionado no canto superior esquerdo da imagem. Se simplesmente adicionar as duas imagens, o resultado será uma imagem degradada, i.e., haverá mudança de cor. Se realizar uma mistura, obtém-se um efeito de transparência. Se fosse uma região retangular, poderia utilizar a ROI como feito anteriormente. Contudo, o logotipo OpenCV não é uma forma retangular. Nesse contexto, operações bit a bit resolvem o problema.

```
import numpy as np
import cv2 as cv

## Carrega as imagens
img1 = cv.imread('messi5.jpg')
img2 = cv.imread('opencv-logo-white.png')

## O objetivo eh colocar o logo no canto superior esquerdo. Para isso,
## emprega-se o conceito de ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols]

## Cria uma mascara do logo e a sua inversa
img2gray = cv.cvtColor(img2,cv.COLOR_BGR2GRAY)
ret, mask = cv.threshold(img2gray, 10, 255, cv.THRESH_BINARY)
mask_inv = cv.bitwise_not(mask)

## "Zera" o fundo
img1_bg = cv.bitwise_and(roi,roi,mask = mask_inv)

## Considera a informacao relevante (foreground)
img2_fg = cv.bitwise_and(img2,img2,mask = mask)

## Posiciona o logo na ROI e modifica a imagem principal
dst = cv.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst

cv.imshow('res',img1)

cv.waitKey(0)
cv.destroyAllWindows()
```

Figura 4: Código em Python que executa operações bit a bit entre duas imagens de modo a modificar uma ROI retangular.

Figura 5: Máscara definida a partir do logo OpenCV e o resultado de sua superposição na imagem original.

(d) **Exemplo para executar em aula:**

Modifique o código da Figura 4 de modo que sejam exibidas todas as imagens intermediárias, especialmente as imagens `img1_bg` e `img2_fg`.

Atividades em Aula - Parte 03: Cálculo de Histograma de uma Imagem

Objetivos:

- Aprender a calcular os histogramas usando as funções OpenCV `cv.calcHist()`;
- Plotar histogramas usando as funções OpenCV e Matplotlib.

Considerações Iniciais:

- Considera-se um histograma como sendo um gráfico ou função, que passa uma ideia geral sobre a distribuição de intensidade de uma imagem. Trata-se de um gráfico com valores de pixel (por exemplo, para uma imagem em escala de cinza com resolução de bit igual à 8, a variação será de 0 a 255) no eixo x e número correspondente de pixels na imagem no eixo y.
- Um histograma pode ser compreendido como sendo uma maneira alternativa de entender a imagem. Ao olhar para o histograma de uma imagem, obtém-se, por exemplo, intuições sobre contraste, brilho e distribuição de intensidade dessa imagem. Quase todas as ferramentas de processamento de imagens atuais são capazes de fornecer recursos baseados em histogramas.

(a) **Terminologias Relacionadas aos Histogramas:**

Bins: O histograma mostra o número de pixels para cada valor de pixel (?!). Considere novamente o caso de uma imagem em escala de cinza com resolução de bit igual a 8. Para essa imagem, o histograma equivalente contém 256 valores (0 a 255) que representam todas os níveis de cinza possíveis de serem obtidos por uma palavra de 8 bits. Cada um desses 256 valores representa um *bin* cuja altura indica quantas vezes esse valor (ou seja, essa intensidade de cinza) aparece na imagem. Note que essa mesma imagem pode ser representada por um histograma alternativo: ao invés de encontrar o número de pixels para todos os valores de pixel separadamente, pode-se considerar o número de pixels em um intervalo de valores de pixel. Por exemplo, um *bin* passa a indicar o número de pixels entre as intensidades de cinza no intervalo de 0 a 15, depois de 16 a 31, ..., 240 a 255. Nesse caso, serão necessários apenas 16 valores (ou *bins*) para representar o histograma. *Bins* são representados pelo termo `histSize` em documentos OpenCV.

Dims: É o número de parâmetros para os quais os dados são coletados. Neste caso, são coletados dados referentes a apenas uma grandeza, o valor de intensidade. Então aqui vale dims = 1.

Range: É o intervalo de valores de intensidade que se deseja medir. No caso do exemplo usado anteriormente, o intervalo é [0, 256], ou seja, todos os valores de intensidade de cinza.

(b) **Cálculo do Histograma:**

A função `cv.calHist()` é empregada para encontrar o histograma. A chamada da função é dada da seguinte maneira:

```
cv.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
```

onde:

images (*imagens*): é a imagem de origem do tipo `uint8` ou `float32`. Deve ser dado entre colchetes, ou seja, "[img]".

channels (*canais*): também é dado em colchetes. É o índice do canal para o qual o histograma é calculado. Por exemplo, se a entrada for uma imagem em tons de cinza, seu valor será [0]. Para a imagem colorida, pode passar [0], [1] ou [2] para calcular o histograma do canal azul, verde ou vermelho, respectivamente.

mask (*máscara*): máscara da imagem. Para encontrar o histograma da imagem completa, é colocado como "None". Mas se quiser encontrar o histograma de uma determinada região da imagem, deve-se criar uma máscara de imagem para isso e passar como o parâmetro "mask".

histSize : Representa a contagem de *bins*. Precisa ser dado entre colchetes. Para a escala completa (escala de cinza, 8 bits), deve ser passado [256].

ranges (*intervalos*): esta é a faixa do intervalo. Normalmente, é [0, 256].

Exemplo de uso:

Carregar uma imagem no modo de escala de cinza e encontrar seu histograma completo:

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

img = cv.imread('home.jpg',0)
histr = cv.calcHist([img],[0],None,[256],[0,256])
```

Interpretação: hist resultante é uma matriz 256×1, cada valor corresponde ao número de pixels nessa imagem com seu valor de pixel correspondente.

(c) **Gráfico do Histograma em OpenCV:**

A biblioteca Matplotlib disponibiliza uma função para plotagem de histograma definida como `matplotlib.pyplot.hist()`. Esta função encontra diretamente o histograma e o grava. Observe que não há necessidade de se empregar a função `calcHist()` para encontrar o histograma previamente. A Figura 6 ilustra como isso pode ser realizado. O resultado é apresentado na Figura 7.

```
1 import numpy as np
2 import cv2 as cv
3 from matplotlib import pyplot as plt
4
5 img = cv.imread('home.jpg',0)
6 histr = cv.calcHist([img],[0],None,[256],[0,256])
7
8 plt.hist(img.ravel(),256,[0,256])
9 plt.show()
```

Figura 6: Código em Python que calcula e mostra o histograma de uma dada figura.

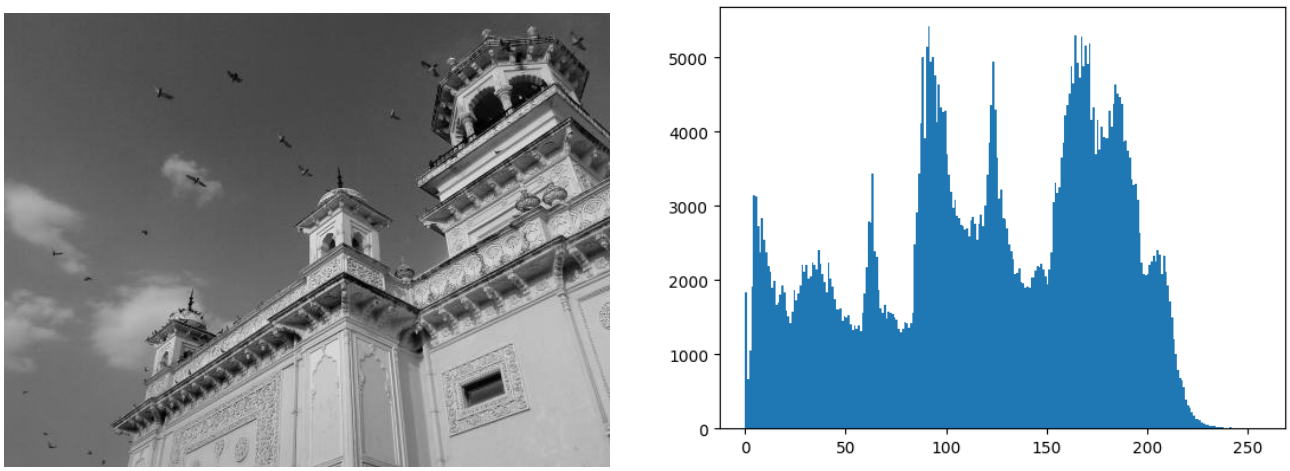


Figura 7: Figura de entrada e o respectivo histograma.

Outra possibilidade consiste em usar o gráfico normal do matplotlib, o que seria bom para o gráfico BGR. Para isso, é necessário encontrar os dados do histograma primeiro. O código ilustrado na Figura 8 indica como isso pode ser feito. O resultado é apresentado na Figura 9.

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

img = cv.imread('home.jpg')
color = ('b','g','r')

for i,col in enumerate(color):

    histr = cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])

plt.show()
```

Figura 8: Código em Python que calcula e mostra o histograma de uma dada figura colorida (RGB). Preste atenção como o espaçamento delimita o que é executado pelo laço de repetição (`for`).

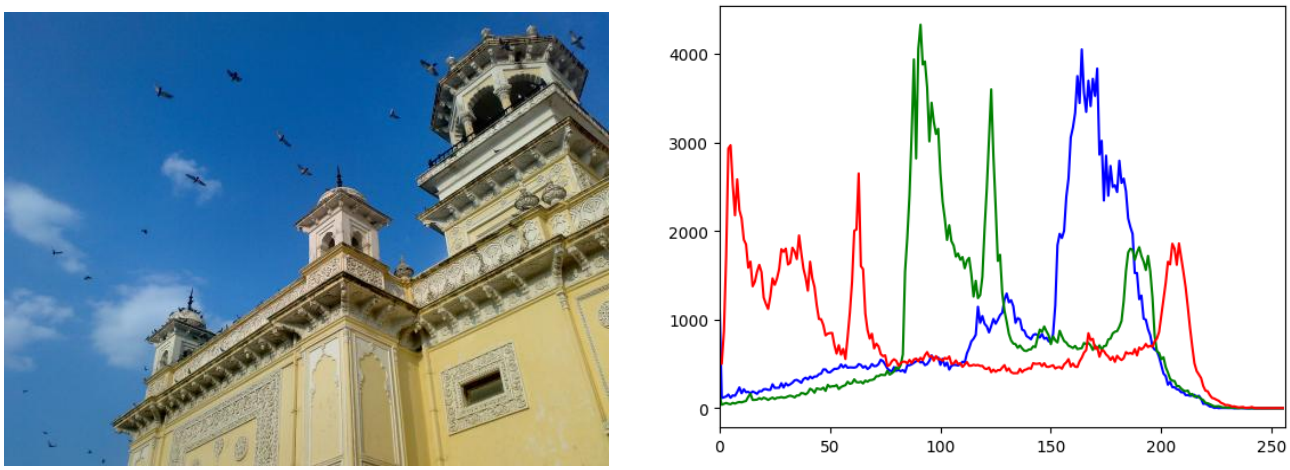


Figura 9: Figura de entrada e o respectivo histograma.

Nota: Para obter o histograma usando apenas o OpenCV (sem o matplotlib) deve-se ajustar os valores dos histogramas junto com seus valores de *bins* para se parecer com as coordenadas (x, y) e, com isso, fazer o gráfico usando a função `cv.line()` ou `cv.polyline()` para gerar a mesma imagem da Figura 7. Isso já está disponível com amostras oficiais do OpenCV-Python3. Verifique o código em `samples / python / hist.py` e/ou nesse link.

(d) **Histograma com Aplicação de Máscara:**

Para encontrar o histograma da imagem completa, pode-se empregar a função `cv.calcHist()`. Se for de interesse encontrar histograma de Regiões Específicas da Imagem, basta criar uma imagem de máscara com cor branca na região que deseja encontrar histograma e preta caso contrário. Então passe isso como a máscara. O código da Figura 10 ilustra como isso pode ser feito e o resultado pode ser observado na Figura 11.

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

img = cv.imread('home.jpg',0)

## Criacao da mascara
mask = np.zeros(img.shape[:2], np.uint8)
mask[100:300, 100:400] = 255
masked_img = cv.bitwise_and(img,img,mask = mask)

## Calcula os histogramas com e sem mascara
## Note que o terceiro argumento da funcao indica quando eh mascara
hist_full = cv.calcHist([img],[0],None,[256],[0,256])
hist_mask = cv.calcHist([img],[0],mask,[256],[0,256])


plt.subplot(221), plt.imshow(img, 'gray')
plt.subplot(222), plt.imshow(mask,'gray')
plt.subplot(223), plt.imshow(masked_img, 'gray')
plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask)
plt.xlim([0,256])

plt.show()
```

Figura 10: Código em Python que calcula e mostra o histograma de uma dada figura e a sua respectiva máscara.

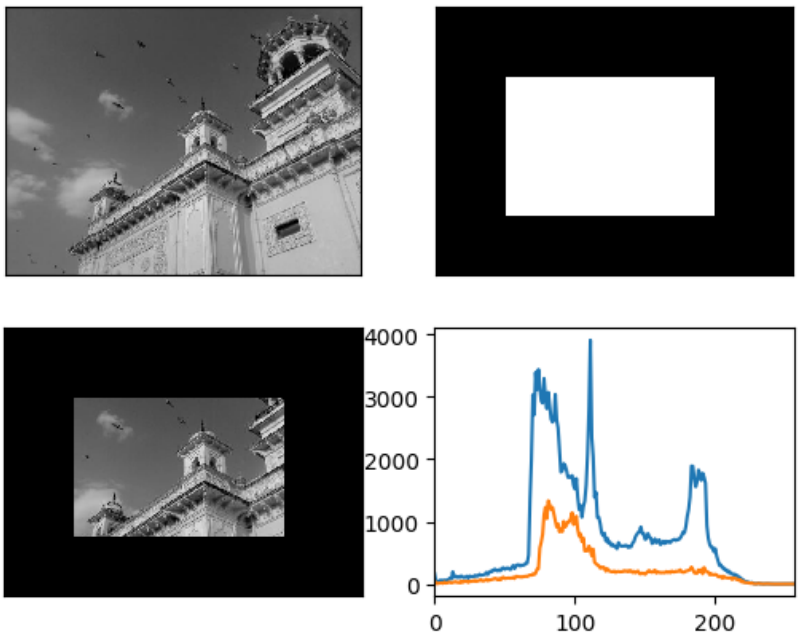


Figura 11: Figura de entrada, a máscara e os respectivos histogramas.

(e) **Equalização de Histogramas em OpenCV:**

Em linhas gerais, a equalização do histograma consiste em ajustar o contraste e o brilho de uma data imagem a partir do seu histograma. A proposta é que haja uma distribuição mais igualitária das intensidades[1]. O OpenCV tem uma função para equalização de histogramas: `cv.equalizeHist()`. Essa função tem como entrada apenas imagem em tons de cinza e a saída é a imagem equalizada do histograma. Na Figura 12 tem-se o resultado da equalização aplicada à uma dada imagem. A Figura 13 ilustra o código implementado para a obtenção da Figura 12.

Figura 12: Figura de entrada, a mascara e os respectivos histogramas.

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

img = cv.imread('wiki.jpg',0)
equ = cv.equalizeHist(img)

## Monta uma imagem com a original e a equalizada
res = np.hstack((img,equ))

cv.imwrite('res.png',res)

cv.imshow('original',img)
cv.imshow('equalization',equ)
cv.imshow('final result',res)

cv.waitKey(0)
cv.destroyAllWindows()
```

Figura 13: Código em Python para a equalização do histograma.

[1]Maiores informações podem ser encontradas, por exemplo, na página da Wikipedia e nos livros da bibliografia da disciplina os quais estão descritos no plano de ensino.

Atividades em Aula - Parte 04: Limiarização

Objetivos:

- Aprender a realizar a limiarização simples empregando a função OpenCV `cv.threshold()`;
- Estudo do algoritmo de limiarização de Otsu.

(a) **Limiarização Simples:**

Conforme discutido em aula, a limiarização consiste em obter uma imagem binária a partir de uma imagem em escala de cinza. Para isso, deve-se comparar o valor do pixel com um limiar (*threshold*): se for maior que esse valor limite, atribuí-se um valor (pode ser branco); caso contrário, é atribuído outro valor (pode ser preto).

Em OpenCV, a função que realiza esse procedimento é `cv.threshold()`. O primeiro argumento dessa função é a imagem de origem, que deve ser uma imagem em tons de cinza. O segundo argumento é o valor limite usado para classificar os valores de pixel. O terceiro argumento é o `maxVal` que representa o valor a ser dado se o valor do pixel for maior que (às vezes menor que) o valor do limite. O OpenCV fornece diferentes estilos de limiar e é decidido pelo quarto parâmetro da função. Os diferentes tipos são[2]:

i. `cv.THRESH_BINARY`

ii. `cv.THRESH_BINARY_INV`

iii. `cv.THRESH_TRUNC`

iv. `cv.THRESH_TOZERO`

v. `cv.THRESH_TOZERO_INV`

Duas saídas são obtidas. A primeira é `retVal`, utilizada na Binarização de Otsu, e a segunda saída é a imagem limiarizada. Na Figura 14 tem-se um código em Python que permite identificar como são implementados os diferentes tipos de binarização. Na Figura 15 podem ser observados os respectivos resultados.

[2]Para melhor compreensão do significado de cada um dos tipos, sugere-se a leitura da documentação.

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

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

retVal,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
retVal,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
retVal,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
retVal,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
retVal,thresh5 = cv.threshold(img,127,255,cv.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()
```

Figura 14: Código em Python para identificar os diferentes tipos de limiarização.

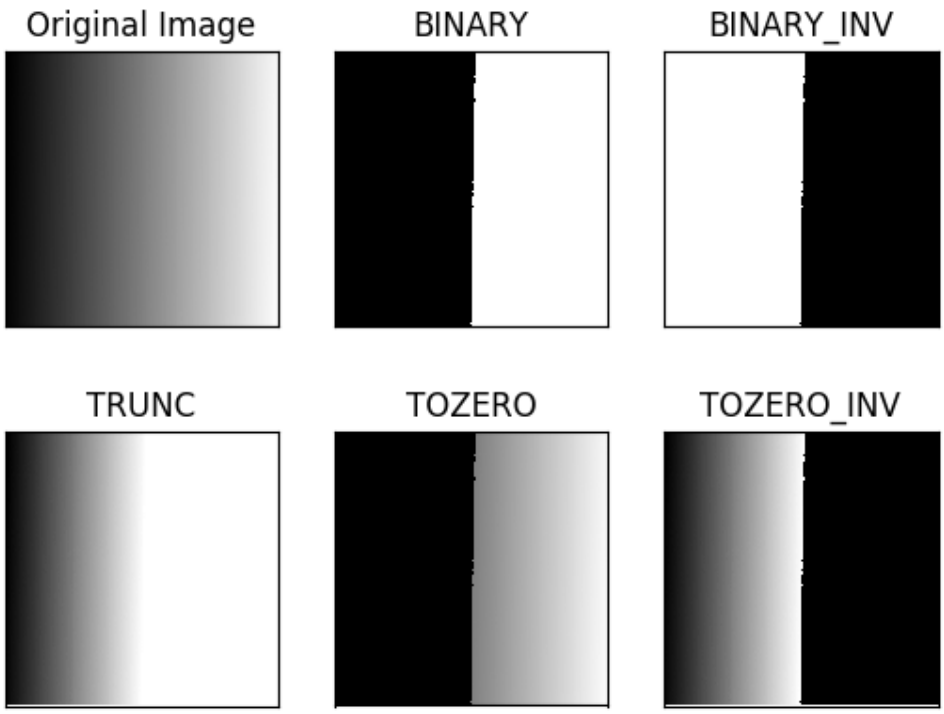


Figura 15: Resultado do código da Figura 14.

(b) **Binarização de Otsu:**

No limiar global, em geral, é utilizado um valor arbitrário para o valor limite. Normalmente, a percepção de que esse limiar é ótimo ou não é obtida apenas após a obtenção da binarização, ou seja, o limiar é ajustado na base da metodologia "da tentativa e do erro".

Considere uma imagem bimodal, i.e., uma imagem cujo histograma tem dois picos distintos. Para essa imagem, pode-se obter aproximadamente um valor no meio desses picos como valor

limite. Isso é o que a binarização Otsu faz, ou seja, o algoritmo de Otsu calcula automaticamente e de maneira não supervisionada um valor limite do histograma da imagem para uma imagem bimodal[3].

Para isso, a função `cv.threshold()` é usada, juntamente com o argumento `cv.THRESH_OTSU`. Para o valor limite, coloca-se zero. Em seguida, o algoritmo encontra o valor de limite ideal e retorna como a segunda saída, `retVal`. Se o limite de Otsu não for usado, `retVal` será o mesmo que o valor limite usado.

Na Figura 16 tem-se um exemplo de código que produz diferentes comportamentos de binarização para limiares distintos, conforme pode ser visto na Figura 17. A imagem de entrada é uma imagem ruidosa. No primeiro caso, aplicou-se o limiar global para um valor de 127. No segundo caso, aplicou-se o limiar calculado pelo algoritmo de Otsu. No terceiro caso, filtra-se a imagem com um kernel gaussiano 5×5 para remover o ruído e, em seguida, aplica-se o limiar Otsu. Pode-se perceber que o resultado da binarização após o procedimento da filtragem de ruído é bastante superior em comparação aos resultados anteriores.

```
1  img = cv.imread('noisy2.png',0)
2
3  ## Obtencao do threshold global
4  ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
5
6  ## Threshold a partir do algoritmo de Otsu
7  ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
8
9  ## Binarizacao de Otsu apos a aplicacao de um filtro gaussiano
10 blur = cv.GaussianBlur(img,(5,5),0)
11 ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
12
13 ## Plot de todas as imagens e seus respectivos histogramas
14 images = [img, 0, th1,
15         img, 0, th2,
16         blur, 0, th3]
17
18 titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
19         'Original Noisy Image','Histogram',"Otsu's Thresholding",
20         'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
21
22 for i in range(3):
23     plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
24     plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
25     plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
26     plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
27     plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
28     plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
29
30 plt.show()
```

Figura 16: Código em Python para identificar o resultado da binarização de uma imagem segundo a aplicação de diferentes valores de limiares.

[3]Para imagens que não são bimodais, a binarização poderá não ser tão precisa.

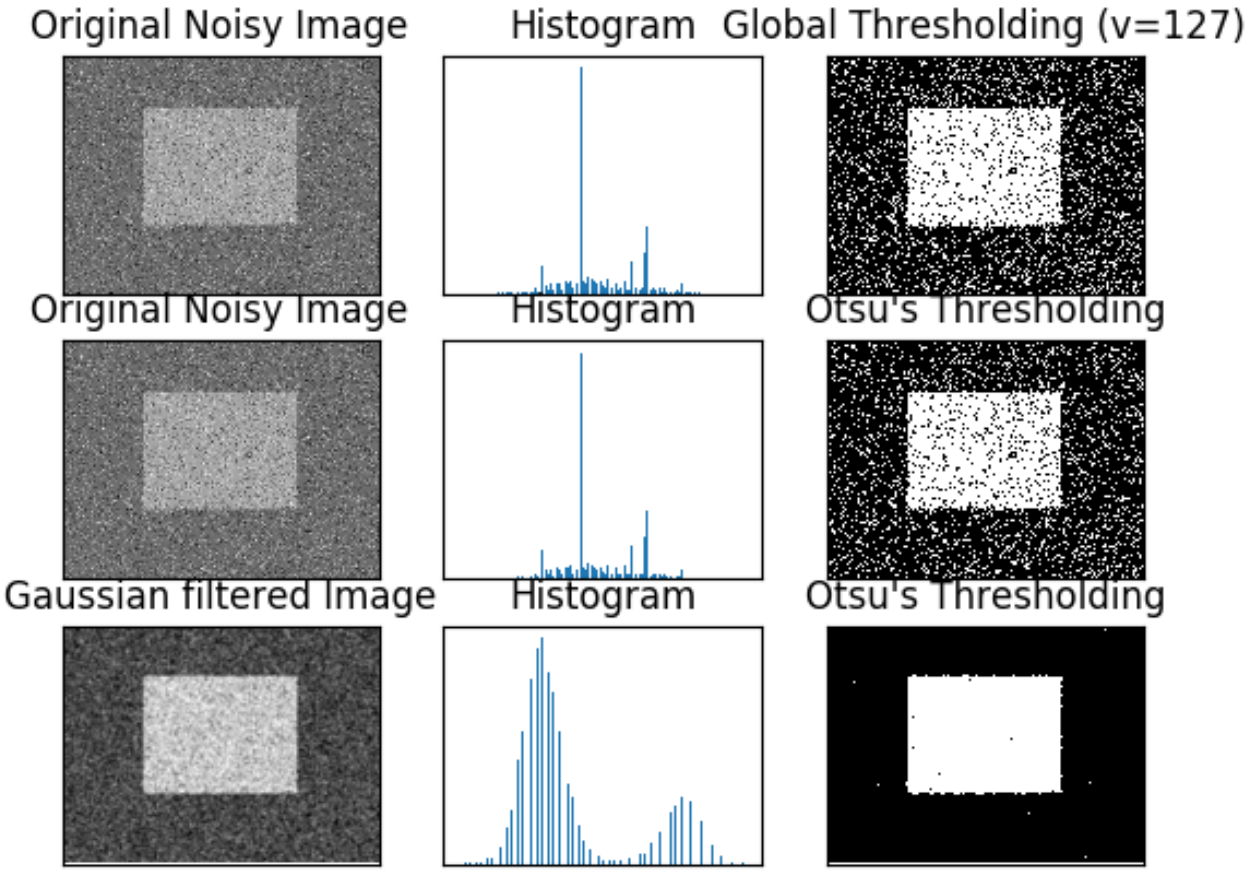


Figura 17: Resultado do código da Figura 16.

Atividades em Aula - Parte 05: Suavização de Imagens

Objetivos:

- Desfocar as imagens com vários filtros passa-baixa;
- Aplicar filtros personalizados às imagens (ou seja, realizar a convolução 2D).

(a) **Convolução 2D (ou Filtragem de Imagens):**

Como nos sinais unidimensionais, as imagens também podem ser filtradas com vários filtros passa-baixa (LPF), filtros passa-alta (HPF), etc. Filtros passa-baixa ajudam, por exemplo, a remover ruídos e borrar as imagens. Já os filtros passa-alta ajudam a encontrar bordas no imagens.

O OpenCV fornece a função `cv.filter2D()` para fazer a convolução de um kernel com uma imagem. Por exemplo, considere um filtro de média aplicado a uma imagem. Um *kernel* de filtro de média 5×5 pode assumir o formato descrito na equação (3):

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3}$$

A operação de filtragem é bastante simples: posiciona-se o *kernel* (3) de modo que o seu elemento central coincida com o pixel-alvo da imagem. Em seguida, deve-se fazer a combinação linear dos pesos do *kernel* com os 24 vizinhos do pixel-alvo e substituir o pixel central pelo novo valor médio. Esse novo valor é o valor de pixel da imagem filtrada. Essa operação é repetida para todos os pixels na imagem. Na Figura 18 tem-se o resultado esperado pela execução do código ilustrado na Figura 19.

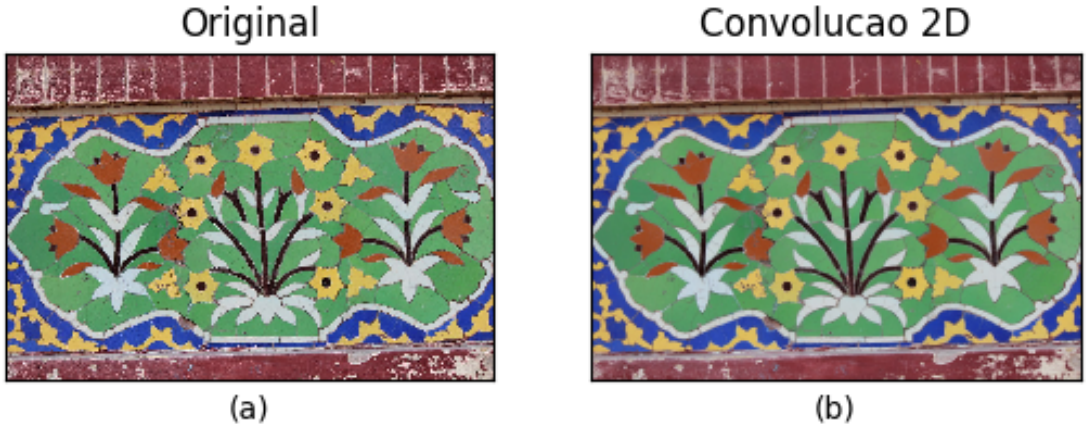


Figura 18: Resultado do código da Figura 19.

```python
import numpy as np
import cv2 as cv
import os # usado para indicar diretorios especificos
from matplotlib import pyplot as plt

path = './'

# 01. Carrega a imagem
img = cv.imread('Tile_WazirKhanmosque.jpg', cv.IMREAD_UNCHANGED)
#img = cv.imread('bowl_fruit.png', cv.IMREAD_UNCHANGED)

# 02. OpenCV usa BGR; matplotlib usa RGB; esse comando corrige isso
# de modo que as cores da figura nao saiam invertidas
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
#img = cv.cvtColor(img, cv.COLOR_RGB2GRAY)

# 03. Convolucao 2D
kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)

# 04. Plot
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.xlabel('(a)'), plt.ylabel('')

plt.subplot(122),plt.imshow(dst),plt.title('Convolucao 2D')
plt.xticks([]), plt.yticks([])
plt.xlabel('(b)'), plt.ylabel('')

# Salva em disco
plt.savefig(os.path.join(path,'Smooth_fig01a.png'), bbox_inches='tight')

# Mostra na tela
plt.show()
```

Figura 19: Código em Python para realizar a convolução 2D de uma imagem segundo um filtro de média.

(b) **Suavização de Imagem (Desfocamento e Borragem):**

O desfoque da imagem é obtido pela convolução da imagem com um *kernel* de filtro passa-baixa. Conforme discutido em aula, essa abordagem é útil para remover ruídos. Na verdade, remove conteúdo de alta frequência, ou seja, além do ruído, as bordas dos elementos contidos na cena. Dessa forma, as bordas são borradas um pouco nesta operação[4].

(a) *Filtragem pela Média*:

Segue a descrição apresentada anteriormente.

A implementação se dá pelo uso das funções `cv.blur()` ou `cv.boxFilter()`. Devem ser especificadas a largura e a altura do *kernel*. Maiores informações podem ser obtidas na documentação oficial. Um exemplo de implementação é apresentado na Figura 20.

(b) *Filtragem Gaussiana*:

Neste filtro, é empregado um *kernel* gaussiano pelo uso da função `cv.GaussianBlur()`. É necessário especificar a largura e altura do *kernel*. Esses valores devem ser positivos e ímpares. Também deve ser especificado o desvio padrão nas direções x e y, `sigmaX` e `sigmaY`, respectivamente. Se apenas `sigmaX` for especificado, `sigmaY` será considerado igual

[4]Existem técnicas de desfoque que também não embaçam as bordas.

```python
import cv2 as cv
import numpy as np
import os # usado para indicar diretorios especificos
from matplotlib import pyplot as plt

path = './'

# 01. Carrega a imagem
img = cv.imread('Tile_WazirKhanmosque.jpg', cv.IMREAD_UNCHANGED)
#img = cv.imread('bowl_fruit.png', cv.IMREAD_UNCHANGED)

# 02. OpenCV usa BGR; matplotlib usa RGB; esse comando corrige isso
# de modo que as cores da figura nao saiam invertidas
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
#img = cv.cvtColor(img, cv.COLOR_RGB2GRAY)

# 03. Configura o filtro
KernelSize = 5
blur = cv.blur(img, (KernelSize,KernelSize))

# 04. Plot
plt.subplot(121), plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.xlabel('(a)'), plt.ylabel('')

plt.subplot(122), plt.imshow(blur),plt.title('Filtro de Media')
plt.xticks([]), plt.yticks([])
plt.xlabel('(a)'), plt.ylabel('')

# Salva no disco
plt.savefig(os.path.join(path,'Smooth_fig02a.png'), bbox_inches='tight')

# Mostra na tela
plt.show()
```

Figura 20: Código em Python que implementa a filtragem pela média usando funções específicas do OpenCV. Note que o *kernel* possui tamanho 5.

Figura 21: Resultado do código da Figura 20.

ao `sigmaX`. Se ambos forem dados como zeros, eles serão calculados a partir do tamanho do *kernel*. O desfoque gaussiano é altamente eficaz na remoção do ruído gaussiano da imagem. O código descrito na Figura 20 pode ser facilmente modificado para empregar um filtro gaussiano, conforme pode ser visto na Figura 22. Na Figura 23 tem-se o resultado esperado.

```
1 # 03. Configura o filtro gaussiano
2 GaussianKernlSize = 5
3 gaussian = cv.GaussianBlur(img, (GaussianKernlSize,GaussianKernlSize), 0)
```

Figura 22: Código em Python que implementa a filtragem gaussiana usando funções específicas do OpenCV.

Figura 23: Resultado do código da Figura 22.

(c) *Filtragem pela Mediana*:

Aqui, a função `cv.medianBlur()` toma a mediana de todos os pixels sob a área do *kernel* e o elemento central é substituído por este valor mediano. Isso é altamente eficaz contra o ruído "sal e pimenta" nas imagens. O interessante é que, nos filtros anteriores, o elemento central é um valor recém-calculado, que pode ser um valor de pixel na imagem ou um novo valor. Contudo, na filtragem de mediana, o elemento central é sempre substituído por algum valor de pixel na imagem. Dessa forma, o ruído é reduzido de maneira bastante eficiente. O tamanho do *kernel* deve ser um inteiro ímpar positivo. Na Figura 26 tem-se o trecho de código que deve ser substituído em relação ao código da Figura 20. Nas Figuras 24 e 25 tem-se o resultado da filtragem pela mediana em uma figura variando-se a intensidade do ruído aplicado (linhas 2 e 4 do código).

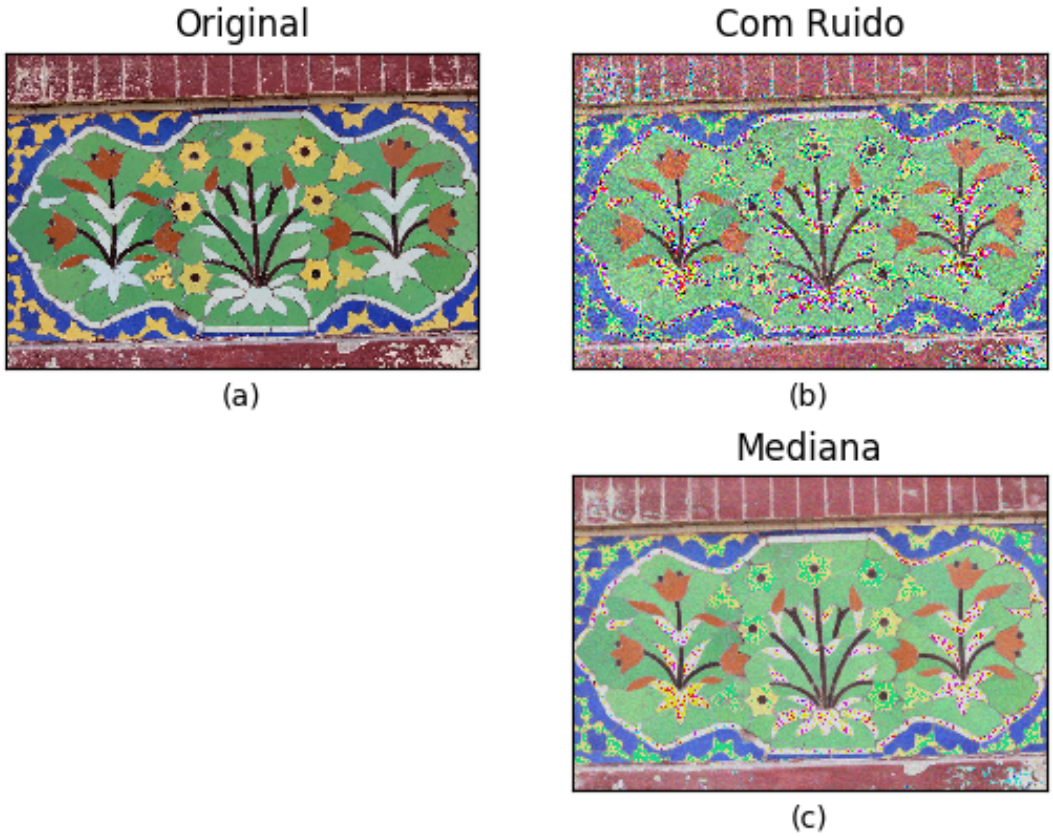


Figura 24: Resultado do código da Figura 26.

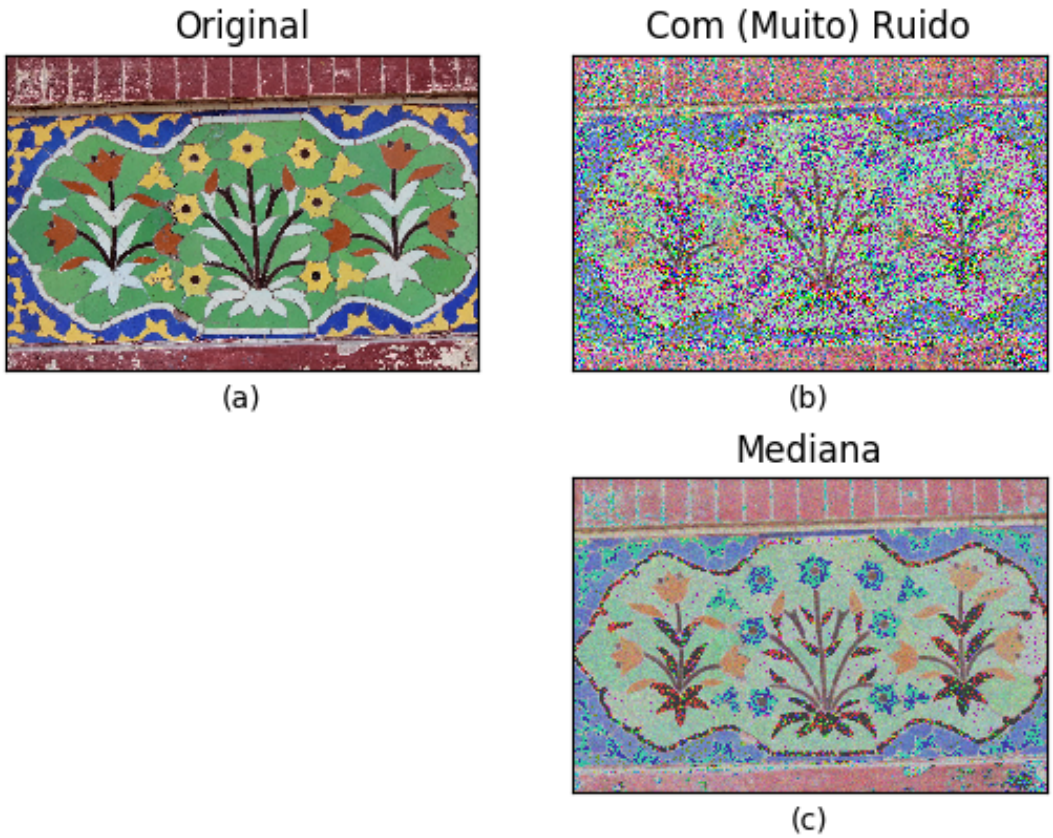


Figura 25: Resultado do código da Figura 26.

```
# 03. Adiciona o ruido ah imagem
noise_01 = ( 1.5*img.std() * np.random.random(img.shape) )
#noise_02 = ( 0.5*img.max() * np.random.random(img.shape) )
noise_02 = ( 2.5*img.std() * np.random.random(img.shape) )

noise_img_01 = img + noise_01
noise_img_02 = img + noise_02

# Antes de passar pelo filtro, tem que garantir que os pixels possuem
# valores uint8
#    - Quando as operacoes anteriores foram realizadas, os pixels foram
#    passados para "float64' (por causa da numpy)
noise_img_01 = noise_img_01.astype('uint8')
noise_img_02 = noise_img_02.astype('uint8')

# 04. Filtra a imagem
KernelSize = 5

median_noise_img_01 = cv.medianBlur(noise_img_01, KernelSize)
median_noise_img_02 = cv.medianBlur(noise_img_02, KernelSize)
```

Figura 26: Código em Python que implementa a filtragem pela mediana usando funções específicas do OpenCV.

Exercícios para serem Analisados no Relatório:

1. Elabore um programa que carrega a sua foto obtida na aula anterior e, utilizando os comandos da Parte 01, separa (por tentativa e erro) uma ROI de seu rosto apenas (olhos, nariz, e boca). Insira esta ROI numa imagem do objeto capturada na aula anterior, e salve a imagem resultante.

2. Elabore um programa utilizando os comandos aprendidos nesta aula para executar as operações aritméticas e reproduzir as imagens mostradas nos slides da aula "Operações Aritméticas em Pixels".

   Imagens da aula: cameraman.tif, toycars1.png, toycars2.png, toycars3.png, cola1.png, cola2.png.

3. Elabore um programa utilizando os comandos aprendidos nesta aula para executar as operações lógicas e reproduzir as imagens mostradas nos slides da aula "Operações Lógicas em Pixels: Limiarização".

   Imagens da aula: rice.png, peppers.png.

4. Utilizando os comandos aprendidos nesta aula, elabore um programa para executar as operações aritméticas e reproduzir as imagens mostradas nos slides da aula "Distribuição de Pixels: Histograma".

   Imagens da aula: coins.png, rice.png.

5. Elabore um programa para calcular o histograma de uma imagem colorida. Deverá ser apresentado um histograma para cada canal (R-G-B). Uma equalização deverá ser feita para cada canal separadamente. Os canais equalizados deverão ser re-agrupados e a imagem colorida composta deverá ser apresentada.

   Imagem da aula: peppers.png

6. Escolha uma das imagens obtidas na aula anterior, que tenha característica bimodal. Elabore um programa para realizar a binarização de Otsu, tanto no formato em escala de cinzas quanto na imagem colorida.

7. Escolha uma das imagens obtida na aula anterior com sua foto. Elabore um programa para realizar as filtragens de média, gaussiana, e mediana, após aplicação de ruídos na imagem original.

Observação: Em todos os casos, sempre que necessário, não esqueça de explicitar o caminho onde as imagens e os vídeos serão salvos.

Observações

1) **Os relatórios deverão ser entregues única e exclusivamente via e-mail. Envie um ou mais links de compartilhamento (não anexe no corpo do e-mail) referentes ao relatório e ao arquivo compactado que contém as imagens, vídeos e os códigos em Python**.

   - Cada código deve estar comentado, contendo um cabeçalho com a sua identificação, breve descrição do que o código faz e comentários pertinentes ao seu completo entendimento. Veja o código ilustrado na Figura 27 apresentado como exemplo de como isso deve ser feito. Na dúvida, pergunte.

```
# INF209B - TOPICOS ESPECIAIS EM PROCESSAMENTO DE SINAIS:
# VISAO COMPUTACIONAL
#
# PRATICA 02
#
# RA: <seu RA>
# NOME: <seu nome>
#
# E-MAIL: <o email que voce estah utilizando para comunicacao>
#
# DESCRICAO:
#    - Descreva em linhas gerais, o que o seu codigo esta fazendo;
#    - Nao esqueca de inserir comentarios pertinentes ao completo
#    entendimento do codigo.
#    - Voce pode "modularizar" os comentarios do seu codigo.
import numpy as np
import cv2 as cv

img = cv.imread('messi5.jpg',0)

cv.imshow('image',img)

k = cv.waitKey(0)

if k == 27:    # wait for ESC key to exit
    cv.destroyAllWindows()

elif k == ord('s'): # wait for 's' key to save and exit
    cv.imwrite('./messigray.png',img)
    cv.destroyAllWindows()
```

Figura 27: Exemplo de como os códigos realizados devem estar comentados.

2) **O relatório deverá estar em formato .pdf. Arquivos enviados em quaisquer outros formatos não serão considerados.**

3) Para facilitar a correção dos relatórios, pede-se que seja adotado o seguinte padrão:

   - `inf209b_<seu RA>_<seu nome>_rel01.pdf`
     - Exemplo: `inf209b_002_Steve_Vai_rel01.pdf`

4) Para auxiliar o filtro de mensagens, ao enviar o e-mail para mim, peço que adote o seguinte:

   - Título do e-mail: [INF209B] <RA> - <Nome> - Prática 02
   - Normalmente, no dia seguinte após a finalização do prazo de entrega, você receberá uma resposta confirmando se o relatório foi recebido.

Sobre o relatório

O relatório pode ou estar formatado no padrão IEEE de conferência (especificado no plano de ensino e discutido em aula) ou descrito usando o Jupyter notebook. Em ambos os casos, o texto deve estar organizado da seguinte maneira:

- Título com a sua identificação (nome da instituição, nome do aluno e o respectivo e-mail de contato);
- Resumo:
    - Autoexplicativo. Resumo das atividades realizadas e considerações sobre os resultados obtidos.
- Metodologia:
    - Explicação da metodologia empregada.
- Resultados:
    - Imagens obtidas, tabelas de análises comparativas de resultados;
    - **Análise dos resultados (interpretação)**.
- Conclusões e discussões;
- Bibliografia (se houver);

Notas da Versão

v01: versão inicial, disponibilizada em 11/03/2020.

Referências

MINICHINO, J., HOWSE, J.. Learning OpenCV 4 Computer Vision with Python 3, 3rd Edition, Packt Publishing, 2020.

Tutorial: Basic Operations on Images

Tutorial: Arithmetic Operations on Images

Tutorial: Camera Histograms: Tones and Contrast

Tutorial: Histogram Begins

Tutorial: Histogram Equalization

Tutorial: Image Thresholding

Tutorial: Smoothing Images

Sintaxe do comando "cv::addWeighted":

Sintaxe do comando "cv::bitwise_and":

Sintaxe do comando "cv::resize":

Sintaxe do comando "cv::threshold":