VISÃO COMPUTACIONAL

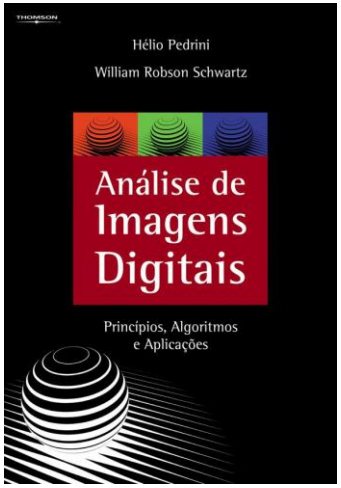
RELACIONAMENTO BÁSICO ENTRE PIXELS

Prof. Msc. Giovanni Lucca França da Silva
E-mail: giovanni-lucca@live.com

SOBRE A DISCIPLINA

- Bibliografia principal:
  - GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens**. Pearson, 2011.

- Bibliografia complementar:
  - PEDRINI, Hélio; SCHWARTZ, William Robson. **Análise de imagens digitais: princípios, algoritmos e aplicações**. Thomson Learning, 2008.



NA AULA PASSADA...

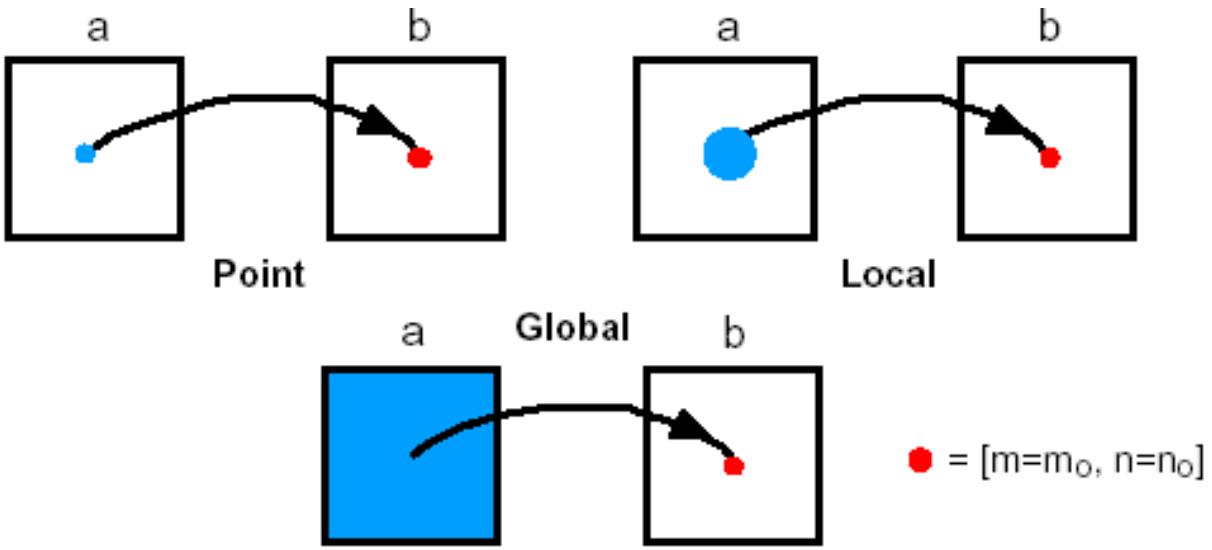
- Representação da imagem digital.
- Passos fundamentais do processamento de imagem.
- Amostragem e quantização.

ROTEIRO

- Introdução.
- Vizinhança.
- Conectividade.
- Componentes conexos.
- Operações lógico-aritméticas.

INTRODUÇÃO

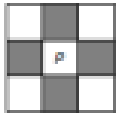
- Relacionamento entre pixels.



VIZINHANÇA

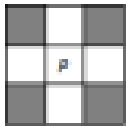
- Um pixel p nas coordenadas (x,y) possui 4 vizinhos horizontais e verticais, $N_4(p)$.

$$(x+1,y),(x-1,y),(x,y+1),(x,y-1)$$

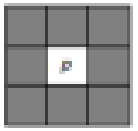


- Os 4 vizinhos diagonais, $N_D(p)$, possuem as seguintes coordenadas.

$$(x+1,y+1),(x+1,y-1),(x-1,y+1),(x-1,y-1)$$



- A vizinhança de 8 de p, $N8(p) = N_4(p) \cup N_D(p)$.



CONECTIVIDADE

- A conectividade entre pixels é um conceito importante usado no estabelecimento das bordas de objetos e componentes de regiões em uma imagem.

- Dois pixels estão conectados se eles são de alguma forma adjacentes e possuem níveis de cinza similares.

CONECTIVIDADE

- Seja V o conjunto de valores de níveis de cinza usados para definir conectividade.
  - Exemplo: V = {1}, V = {32,33,34,36}.

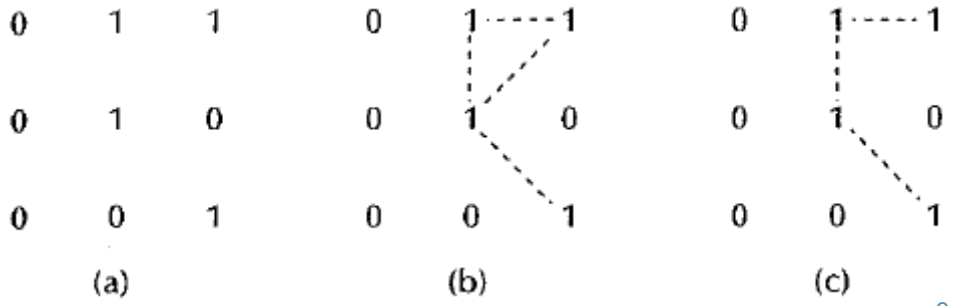
- Conectividade de 4: p e q, assumindo valores em V, são conectados de 4 se q está no conjunto de $N_4(p)$.

- Conectividade de 8: p e q, assumindo valores em V, são conectados de 8 se q está no conjunto de $N_8(p)$.

CONECTIVIDADE

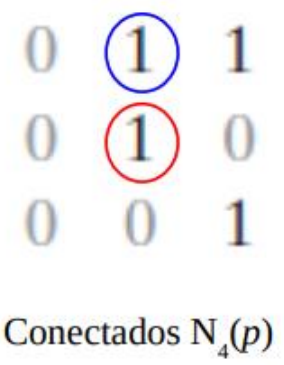
- Conectividade mista: p e q, assumindo valores em V, são conectados de m se
  - q está no conjunto de $N_4(p)$, ou
  - q está no conjunto de $N_D(p)$ e o conjunto $N_4(p) \cap N_4(q)$ for vazio.

- A conectividade mista é uma modificação da conectividade de 8 para eliminar múltiplos caminhos.

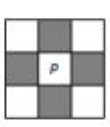


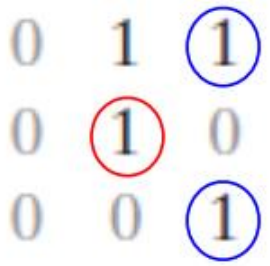
CONECTIVIDADE

- Estabelece uma relação de adjacência entre pixels e seus respectivos níveis de cinza.
  - Exemplo: V = {1}.

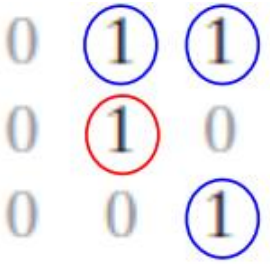


Conectados $N_4(p)$

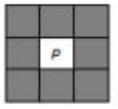




Conectados $N_D(p)$

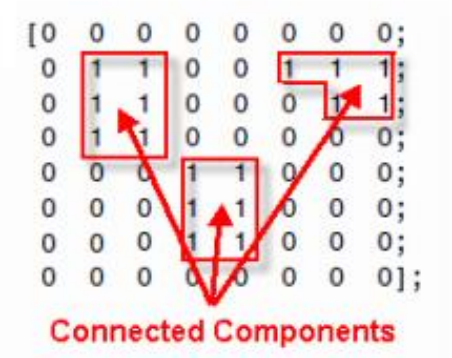


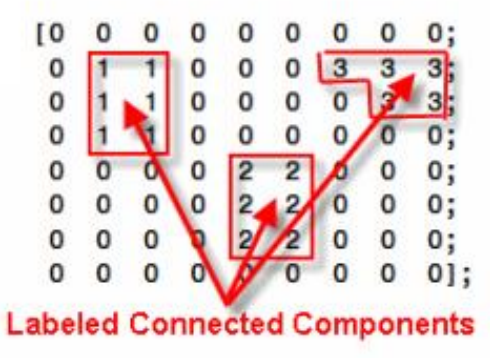
Conectados $N_8(p)$

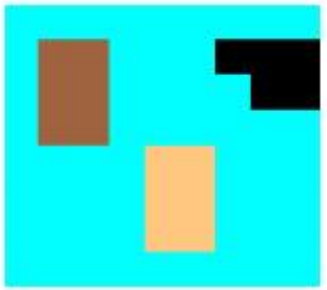


COMPONENTES CONEXOS

- Se p e q forem pixels de um subconjunto S de uma imagem, então p está conectado a q em S se existir um caminho de p a q consistindo inteiramente de pixels de S.

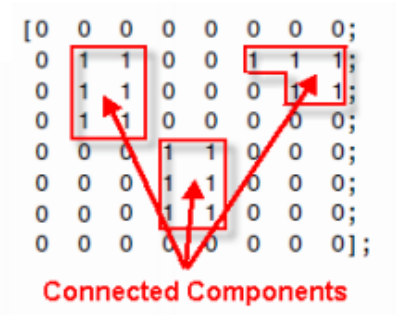


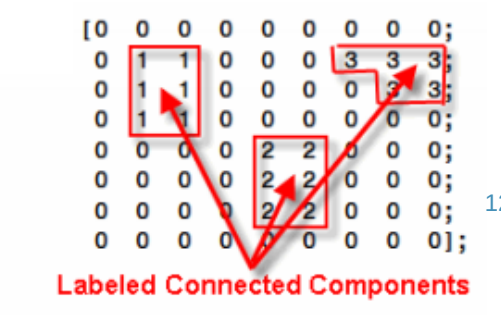




COMPONENTES CONEXOS

- Seja p o pixel em qualquer passo no processo de varredura e sejam r e t, respectivamente, os vizinhos superior e esquerdo.
  - Se o valor de p é 0, mova para a próxima posição.
  - Se o valor de p é 1, examine r e t.
    - Se ambos forem 0, atribua um novo rótulo a p.
    - Se apenas um dos dois forem 1, atribua o seu rótulo a p.
    - Se ambos forem 1 e possuem o mesmo rótulo, atribua a p aquele rótulo.
    - Se ambos forem 1 e possuem rótulos diferentes, atribua um dos rótulos a p e anote que esses rótulos são equivalentes.

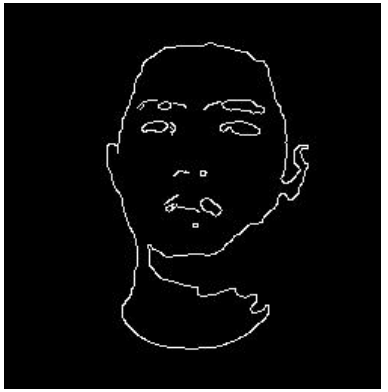


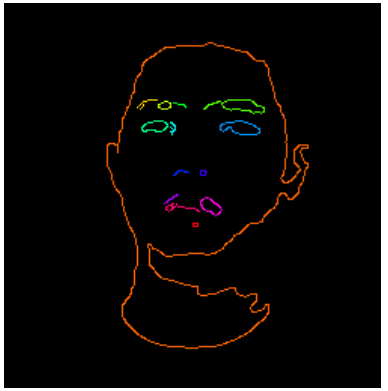


COMPONENTES CONEXOS

- Função cv2.connectedComponents().

```
import cv2
import numpy as np

img = cv2.imread('eGaIy.jpg', 0)
img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]
ret, labels = cv2.connectedComponents(img)

# Map component labels to hue val
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])

# cvt to BGR for display
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)

# set bg label to black
labeled_img[label_hue==0] = 0

cv2.imshow('labeled.png', labeled_img)
cv2.waitKey()
```

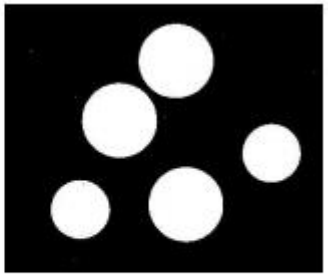




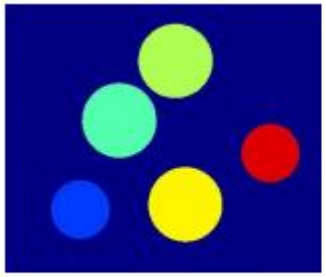
COMPONENTES CONEXOS

- Exemplo de aplicação:
  - Remoção de objetos com área maior que T.

Imagem de entrada



Resultado da segmentação



Rotulação dos componentes conexos

Imagem processada

COMPONENTES CONEXOS

- Exemplo de aplicação:
  - Análise de forma.

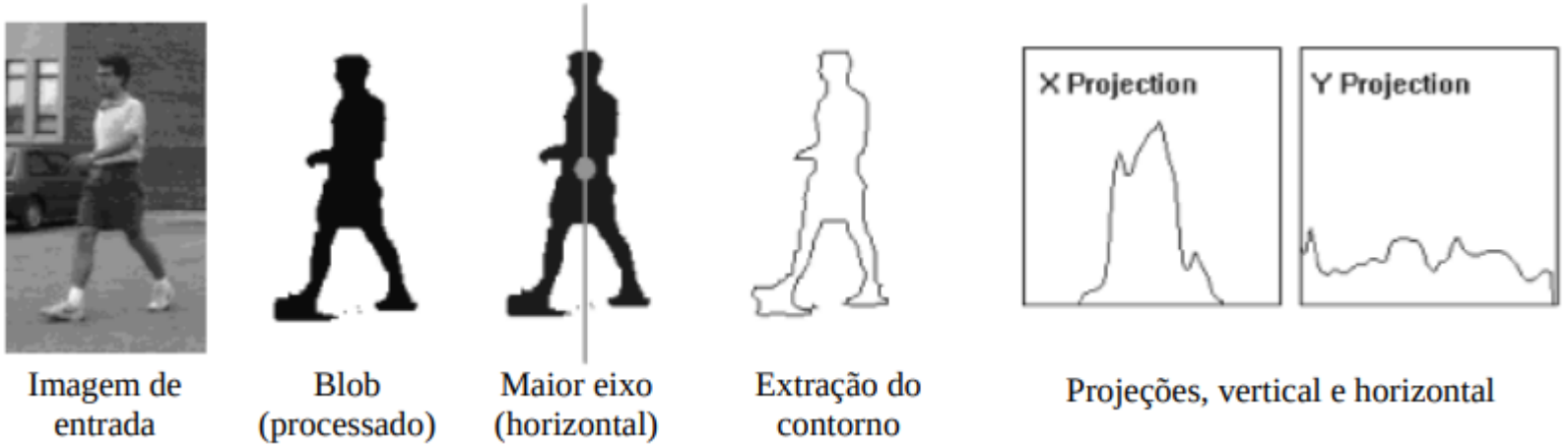
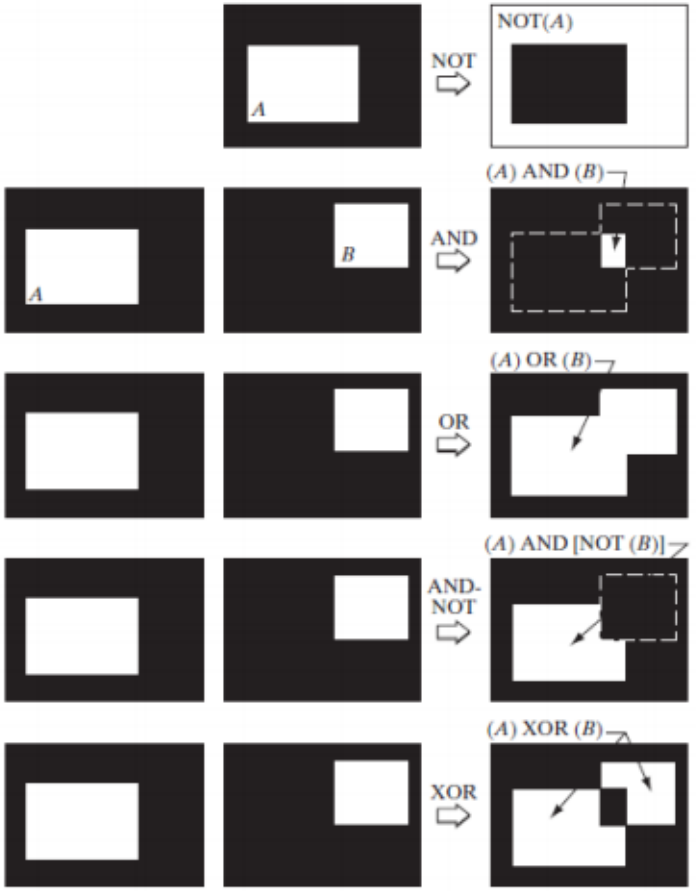


Imagem de entrada | Blob (processado) | Maior eixo (horizontal) | Extração do contorno | Projeções, vertical e horizontal

OPERAÇÕES LÓGICO-ARITMÉTICAS

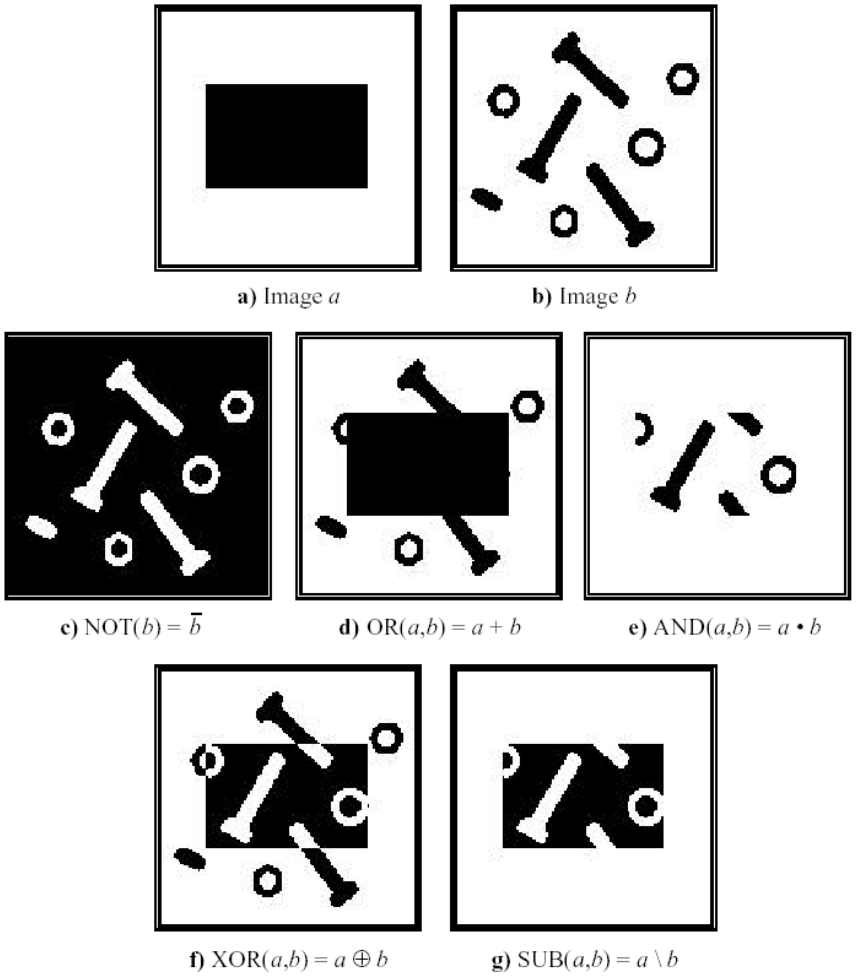
- Operações lógicas:
  - Aplicadas à imagens binárias.
  - AND: p E q.
    - cv2.bitwise_and(src1, src2)
  - OR: p OU q.
    - cv2.bitwise_or(src1, src2)
  - XOR: p OU EXCLUSIVO q.
    - cv2.bitwise_xor(src1, src2)
  - NOT: NÃO q.
    - cv2.bitwise_not(src1, src2)



OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações lógicas:
  - Aplicadas à imagens binárias.

```
img1 = cv2.imread("drawing_1.png")
img2 = cv2.imread("drawing_2.png")

bit_and = cv2.bitwise_and(img2, img1)
bit_or = cv2.bitwise_or(img2, img1)
bit_xor = cv2.bitwise_xor(img1, img2)
bit_not = cv2.bitwise_not(img1)
bit_not2 = cv2.bitwise_not(img2)


cv2.imshow("img1", img1)
cv2.imshow("img2", img2)

cv2.imshow("bit_and", bit_and)
cv2.imshow("bit_or", bit_or)
cv2.imshow("bit_xor", bit_xor)
cv2.imshow("bit_not", bit_not)
cv2.imshow("bit_not2", bit_not2)


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



OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações lógicas:

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);
        int pixel2 = image2->GetPixel(pixelIndex);

        int newPixel;

        if(pixel == 0 && pixel2 == 0){
            newPixel = 0;
        }else if(pixel == 0 && pixel2 == 255){
            newPixel = 0;
        }else if(pixel == 255 && pixel2 == 0){
            newPixel = 0;
        }else{
            newPixel = 255;
        }

        logicImage->SetPixel(pixelIndex, newPixel);
    }
}
```

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);
        int pixel2 = image2->GetPixel(pixelIndex);

        int newPixel;

        if(pixel == 0 && pixel2 == 0){
            newPixel = 0;
        }else if(pixel == 0 && pixel2 == 255){
            newPixel = 255;
        }else if(pixel == 255 && pixel2 == 0){
            newPixel = 255;
        }else{
            newPixel = 255;
        }

        logicImage->SetPixel(pixelIndex, newPixel);

    }
}
```

OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações lógicas:

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);

        int newPixel;

        if (pixel == 0) {
            newPixel = 255;
        }else{
            newPixel = 0;
        }

        logicImage->SetPixel(pixelIndex, newPixel);

    }
}
```

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);
        int pixel2 = image2->GetPixel(pixelIndex);

        int newPixel;

        if(pixel == 0 && pixel2 == 0){
            newPixel = 0;
        }else if(pixel == 0 && pixel2 == 255){
            newPixel = 255;
        }else if(pixel == 255 && pixel2 == 0){
            newPixel = 255;
        }else{
            newPixel = 0;
        }

        logicImage->SetPixel(pixelIndex, newPixel);

    }
}
```

OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Adição: p + q.
    - Operação linear pixel a pixel.
    - N imagens, sendo N >= 2.
    - Soma os pixels e divide por N.
    - Reduzir ruídos.
    - Exemplo: Sensoriamento remoto.

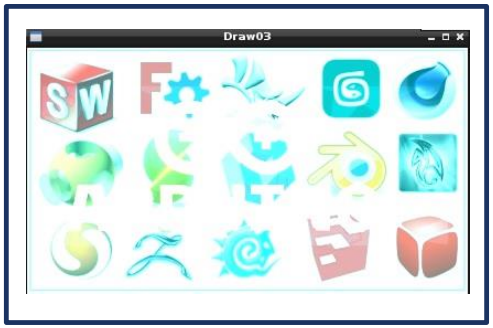
OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Adição: p + q.

OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Adição: p + q.


```
import cv2
import numpy as np

# 500 x 250
img1 = cv2.imread('3D-Matplotlib.png')
img2 = cv2.imread('mainsvmimage.png')

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

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



OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Adição: p + q.

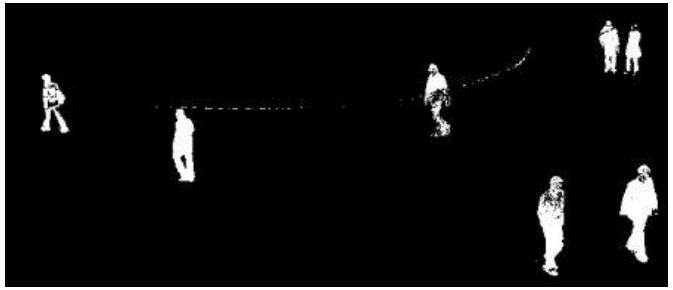
```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);

        int pixel2 = image2->GetPixel(pixelIndex);

        int add = pixel + pixel2;

        if (add > 255) add = 255;

        operationImage->SetPixel(pixelIndex, add);

    }
}
```

OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Subtração: p – q.
    - Operação linear pixel a pixel mais utilizada.
    - N imagens, sendo N == 2.
    - Soma 255 e divide por 2 cada pixel.
    - Remover características estáticas.
    - Exemplo: Controle de qualidade.

OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Subtração: p – q.



```
>>> import cv2
>>> import numpy as np

>>> a = np.arange(9, dtype=np.uint8).reshape(3,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]], dtype=uint8)
>>> b = np.full((3,3), 4, np.uint8)
>>> b
array([[4, 4, 4],
       [4, 4, 4],
       [4, 4, 4]], dtype=uint8)

>>> np.subtract(b,a)
array([[  4,   3,   2],
       [  1,   0, 255],
       [254, 253, 252]], dtype=uint8)

>>> cv2.subtract(b,a)
array([[4, 3, 2],
       [1, 0, 0],
       [0, 0, 0]], dtype=uint8)
```

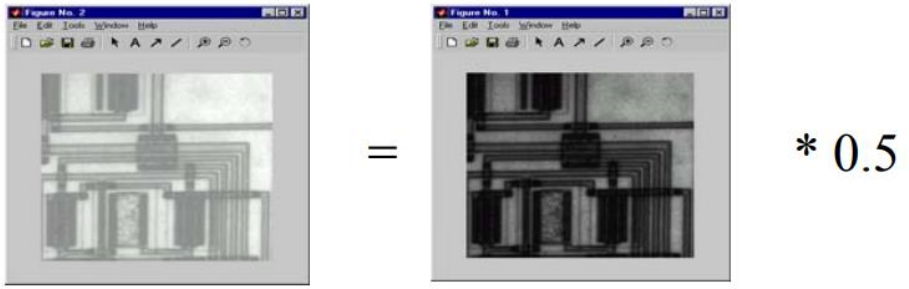
OPERAÇÕES LÓGICO-ARITMÉTICAS

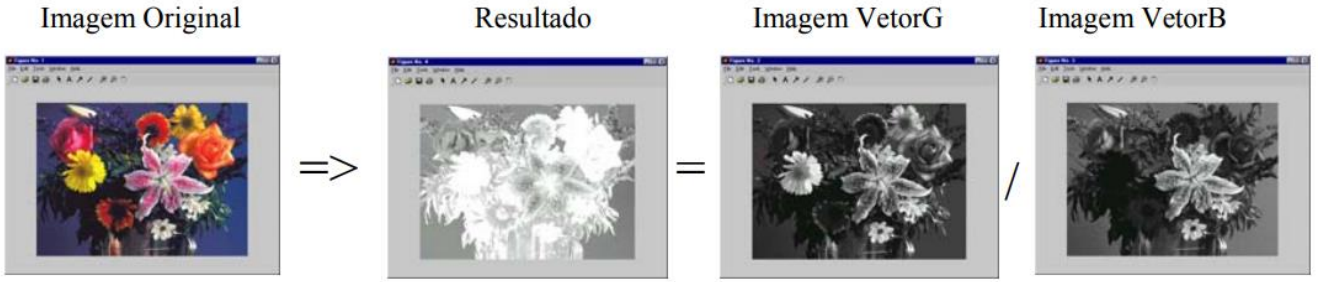
- Operações aritméticas:
  - Subtração: p – q.

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);

        int pixel2 = image2->GetPixel(pixelIndex);

        int sub = pixel - pixel2;

        if (sub < 0) sub = 0;

        operationImage->SetPixel(pixelIndex, sub);

    }
}
```

OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Multiplicação: p * q.
    - Operação menos utilizada.
    - Realce das baixas magnitudes.
    - Corrigir sombra.
  - Divisão: p/q.
    - Identificar diferenciação de contrastes.





OPERAÇÕES LÓGICO-ARITMÉTICAS

- Operações aritméticas:
  - Multiplicação: p * q.
  - Divisão: p/q.

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);

        int pixel2 = image2->GetPixel(pixelIndex);

        int mult = pixel * pixel2;

        if (mult > 255) mult = 255;

        operationImage->SetPixel(pixelIndex, mult);

    }
}
```

```
for(int i = 0; i < width; i++){
    for(int j = 0; j < height; j++){

        ImageType::IndexType pixelIndex={{i,j}};

        int pixel = image->GetPixel(pixelIndex);

        int pixel2 = image2->GetPixel(pixelIndex);

        if(pixel == 0) pixel = 1;

        if(pixel2 == 0) pixel2 = 1;

        float div = pixel/pixel2;

        round(div);

        operationImage->SetPixel(pixelIndex, div);

    }
}
```

OPERAÇÕES LÓGICO-ARITMÉTICAS

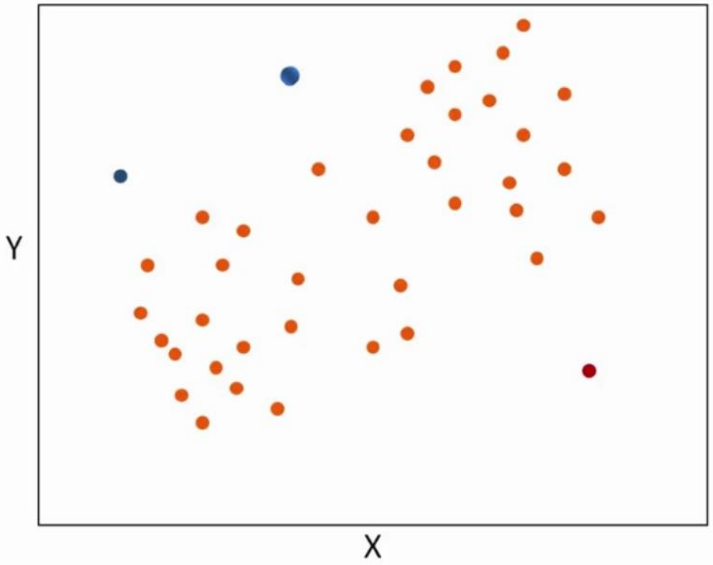
- Operações aritméticas:
  - Mistura de imagens.
    - Operação da soma, porém ponderada.

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

```
img1 = cv2.imread('ml.png')
img2 = cv2.imread('opencv-logo.png')

dst = cv2.addWeighted(img1,0.7,img2,0.3,0)

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

MEDIDAS DE DISTÂNCIAS

- Considere os pixels p, q, e z, com coordenadas (x,y), (s,t) e (v,w), respectivamente. D é uma medida de distância se:
  - D(p,q) >= 0 (D(p,q) == 0, se p == q).
  - D(p,q) == D(q,p).
  - D(p,z) <= D(p,q) + D(q,z).



MEDIDAS DE DISTÂNCIAS

- Distância Euclidiana entre p e q:

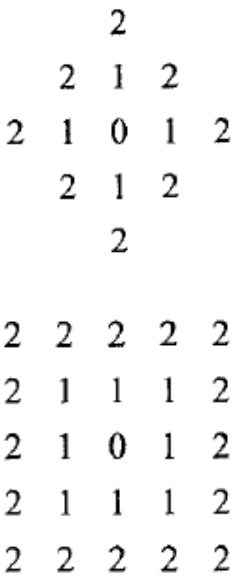
$$D_e(p,q) = \left[(x-s)^2 + (y-t)^2\right]^{\frac{1}{2}}$$

- Distância D_4 (quarteirão) entre p e q:

$$D_4(p,q) = |x-s| + |y-t|$$

- Distância D_8 (xadrez) entre p e q:

$$D_8(p,q) = \max(|x-s|, |y-t|)$$



MEDIDAS DE DISTÂNCIAS

- Distância Euclidiana entre p e q:

```
import math
''
def dist_euclidiana(v1, v2):
        dim, soma = len(v1), 0
        for i in range(dim):
                soma += math.pow(v1[i] - v2[i], 2)
        return math.sqrt(soma)

print('%.2f' % dist_euclidiana(v1, v2))
```

```
import numpy as np
''
def dist_euclidiana_np(v1, v2):
        v1, v2 = np.array(v1), np.array(v2)
        diff = v1 - v2
        quad_dist = np.dot(diff, diff)
        return math.sqrt(quad_dist)

print('%.2f' % dist_euclidiana_np(v1, v2))
```

REFERÊNCIAS

- GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens**. Pearson, 2011.

- PEDRINI, Hélio; SCHWARTZ, William Robson. **Análise de imagens digitais: princípios, algoritmos e aplicações**. Thomson Learning, 2008.

- SILVA, Aristófanes. **Notas de aula da disciplina Processamento de Imagens da Universidade Federal do Maranhão**. 2018.

- BRAZ Jr, Geraldo. **Notas de aula da disciplina Visão Computacional da Universidade Federal do Maranhão**. 2018.