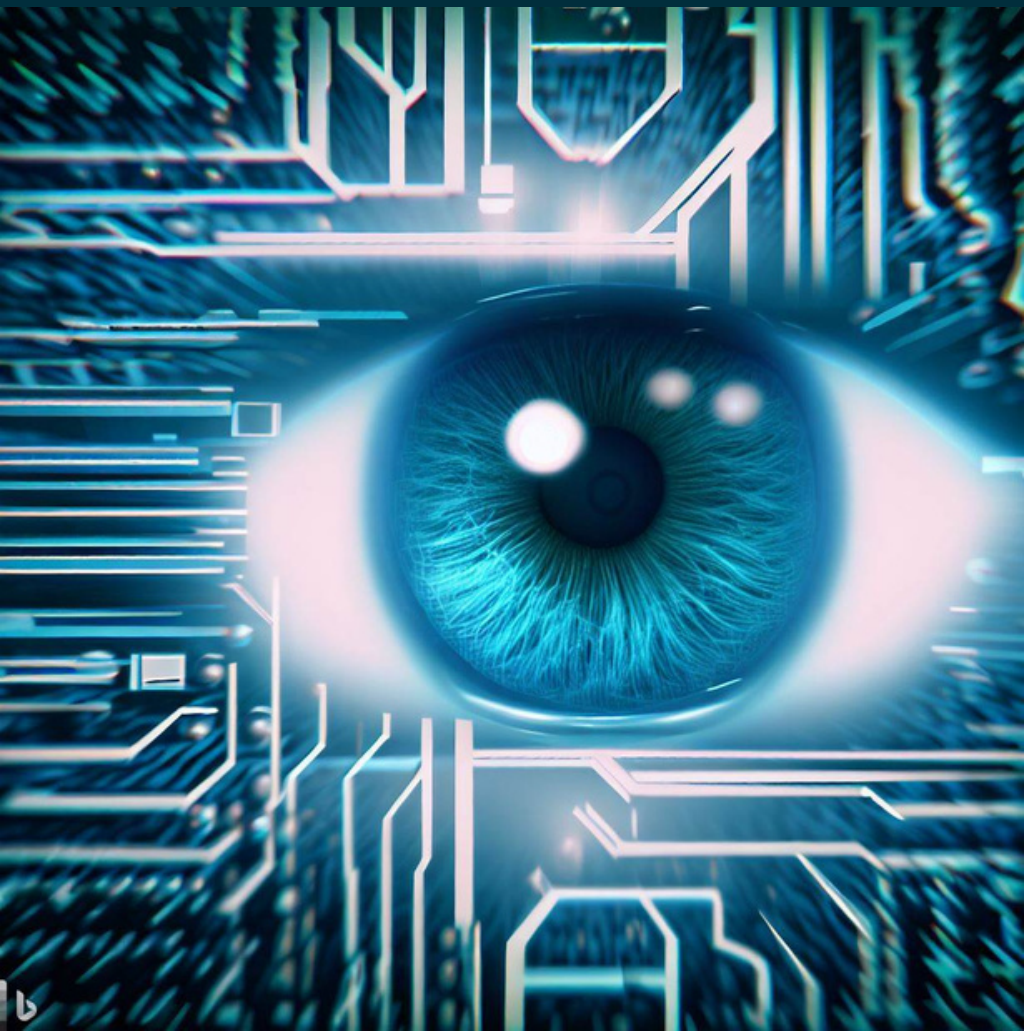
VISÃO COMPUTACIONAL COM PYTHON E OPENCV

AUTOR: GABRIEL RIBEIRO RODRIGUES DESSOTTI



SUMÁRIO

ORIENTAÇÕES

- **O treinamento fornecido nesse documento usa, em suma, a biblioteca OpenCV da linguagem de programação Python;**
- **A documentação dessa biblioteca pode ser encontrada nesse link;**
- **Nesse link, encontra-se as orientações para instalação da biblioteca;**
- **Orienta-se o uso de VS Code para testes e realização de exercícios, mas também há a possibilidade, com algumas adaptações, de uso de Jupyter Notebook ou Google Colab;**
- **Recomenda-se que, a cada linha de código explicada, faça o teste dela no próprio computador.**

A BIBLIOTECA OPENCV

- **Ferramenta usada para trabalhar com imagens e vídeos;**
- **Open source;**
- **Lançada no ano 2000 pela Intel;**
- **Funcionalidades: pré-processamento de imagens, filtros, segmentação de objetos, extração de características, detecção de movimentos, detecção de faces, comandos de vídeo, entre outros;**
- **Presente em várias linguagens de programação, como Python, C++ e Java.**

O QUE É UMA IMAGEM?

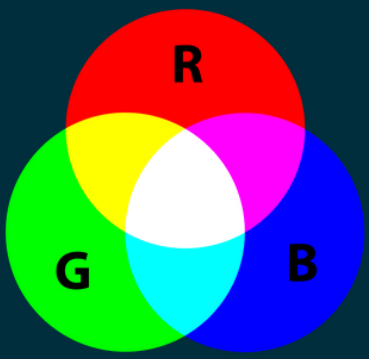
Para o OpenCV, uma imagem é vista como uma matriz. Cada elemento da matriz é um pixel, que possui n valores para representar a cor do pixel.

Para imagens na escala de cinza, ou grayscale, cada pixel possui apenas um canal de cores. O valor é entre 0 e 255 e representa a tonalidade de cinza que o pixel tem. O valor 0 representa o preto, enquanto o valor 1 representa o branco.

Para imagens coloridas, temos mais de um canal de cor. Usualmente, usamos o espaço de cor RGB (red, green, blue).

- **R: intensidade do vermelho (0 a 255);**
- **G: intensidade do verde (0 a 255);**
- **B: intensidade do azul (0 a 255).**

Assim, cada pixel é formado por um vetor [R, G, B].

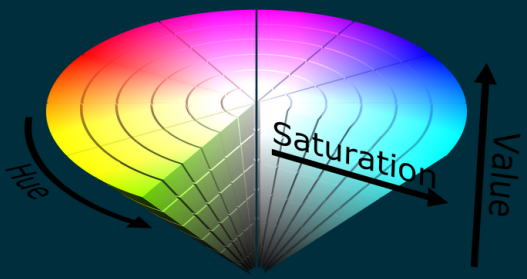


O QUE É UMA IMAGEM?

Além do RGB, usa-se muito o espaço de cor HSV. Assim como o anterior, possui três canais de cor: matiz (hue), saturação e valor.

- **H: representa o tipo de cor, descrevendo a cor dominante (0 a 360);**
- **S: mede a vivacidade da cor. Um valor próximo a 0 reflete uma cor semelhante a um tom de cinza, enquanto um valor próximo a 100 reflete uma cor vibrante (0 a 100);**

V: representa o brilho. Quanto maior o valor, maior o brilho (0 a 100).



Além desses três espaços citados, existem outros, como o CMYK (ciano, magenta, amarelo e preto). No entanto, os três explicados com mais detalhes serão os mais usados.

OBSERVAÇÃO: o OpenCV, por padrão, lê uma imagem em BGR, ou seja, um RGB com a ordem invertida. Existe uma função que faz conversão de espaços de cor, que será vista posteriormente.

ABRINDO UMA IMAGEM

Primeiro, devemos fazer a importação da biblioteca no nosso código em Python.

```
import cv2 as cv
```

Após isso, usamos a função imread() dessa biblioteca para ler a imagem.

```
img = cv.imread(destino, param)
```

O 'destino' é o diretório que a imagem se localiza nos arquivos do computador.

O 'param' pode ser cv.IMREAD_GRAYSCALE (ou 0), cv.IMREAD_COLOR (ou 1) ou cv.IMREAD_UNCHANGED (ou -1).

- **cv.IMREAD_GRAYSCALE: lê a imagem em escalas de cinza;**
- **cv.IMREAD_COLOR: lê a imagem em BGR;**
- **cv. IMREAD_UNCHANGED: lê a imagem incluindo um canal de transparência.**

Para exibir a imagem em uma janela, usa-se o código abaixo.

```
cv.imshow(nome, img)
```

ABRINDO UMA IMAGEM

A função a seguir determina um tempo para a janela exibir a imagem. O tempo é dado em milissegundos e, quando inserido o valor 0, a imagem fica aberta até ser executada alguma ação.

```
cv.waitKey(tempo)
```

Por fim, é deve-se fechar as janelas.

- **Fechar todas:**

```
cv.destroyAllWindows()
```

- **Fechar a janela 'nome':**

```
cv.destroyWindow(nome)
```

Outrossim, tendo a matriz da imagem armazenada na variável img, podemos extrair algumas características básicas dessa imagem. O comando ".shape" extrai de uma imagem dados sobre sua altura, largura e quantidade de canais de cor.

```
altura, largura, canais = img.shape
```

O comando ".dtype" fornece o tipo de dados que forma a imagem, podendo CV_8U (unsigned int 8 bits), CV_8S (signed int 8 bits), CV_32F (float 32 bits), entre outros.

```
tipo = img.dtype
```

CONVERSÃO DE ESPAÇOS DE CORES

O OpenCV carrega uma imagem na escala BGR como já dito anteriormente. No entanto, podemos converter para qualquer outra escala.

Para tanto, usa-se a seguinte função.

```
cv.cvtColor(entrada, saida, param)
```

A 'entrada' é a imagem original, a 'saida' é a imagem convertida e 'param' é o parâmetro de conversão, no qual é informado de qual espaço para qual espaço a imagem deve ser convertida.

O 'param' pode ser:

- **cv.COLOR_BGR2RGB: BGR para RBG;**
- **cv.COLOR_BGR2HSV: BGR para HSV;**
- **cv.COLOR_BGR2GRAY: BGR para Grayscale;**
- **cv.COLOR_RGB2BGR: RGB para BGR;**
- **cv.COLOR_RGB2HSV: RGB para HSV;**
- **cv.COLOR_RGB2GRAY: RGB para Grayscale.**

Com os parâmetros acima, percebe-se o padrão para fazer as conversões entre espaços.

SALVANDO UMA IMAGEM

Para salvar uma imagem criada em código nos arquivos do seu computador é usada a seguinte função.

```
cv.imwrite(nome, img)
```

O nome da imagem deve incluir o formato que ela será salva e img é a matriz da imagem.

FILTROS

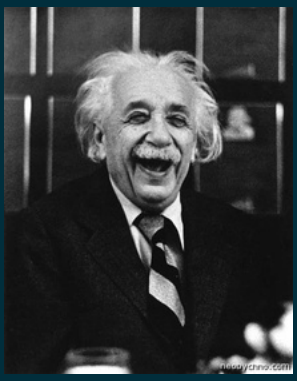
Filtros são máscaras que percorrem toda a imagem, alterando os valores do pixels. Isso é denominado convolução.

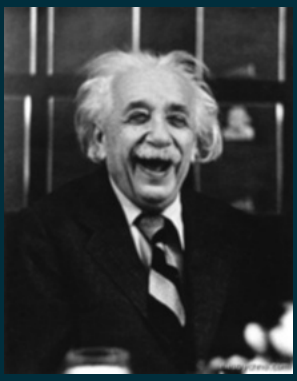
FILTROS DE SUAVIZAÇÃO

Filtro Média

- **Substitui cada pixel da imagem pelo valor médio da sua vizinhança.**

```
img_media = cv.blur(img, mascara)
```





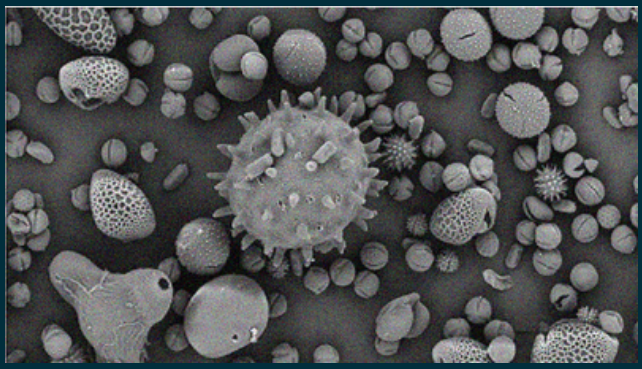
```
img_media = cv.blur(img, (5,5))
```

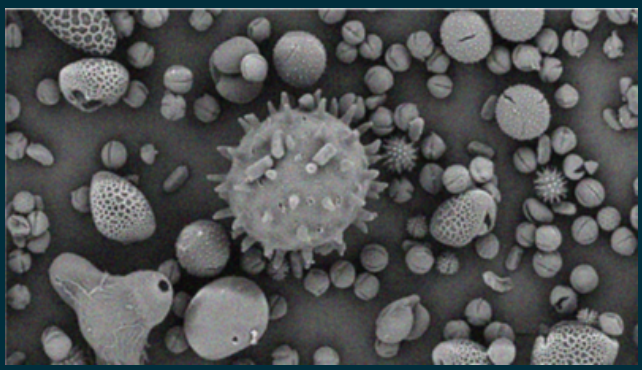
FILTROS

Filtro Gaussiano

- **Usa máscaras ponderadas com diferentes valores baseados em uma distribuição gaussiana. Trata ruídos gaussianos;**
- **O parâmetro 'suav' refere-se ao grau de suavização.**

```
img_gaussiana = cv.GaussianBlur(img, mascara,
                                suav)
```





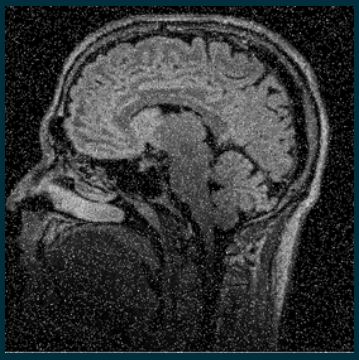
```
img_gaussiana = cv.GaussianBlur(img, (5,5), 0)
```

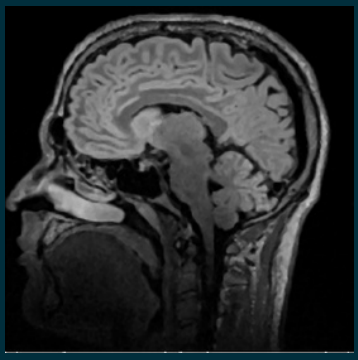
FILTROS

Filtro Mediana

- **Substitui o valor central da área compreendida pela máscara pela mediana dos valores da matriz da imagem dentro dessa área. Trata ruídos do tipo sal e pimenta.**

```
img_mediana = cv.medianBlur(img, intensidade)
```





```
img_mediana = cv.medianBlur(img, 5)
```

Filtro Bilateral

- **Aplica uma combinação ponderada de suavização baseada na distância espacial e de intensidade dos pixels. Reduz ruídos e preserva contornos.**

```
img_bilateral = cv.bilateralFilter(img, tam)
```

FILTROS

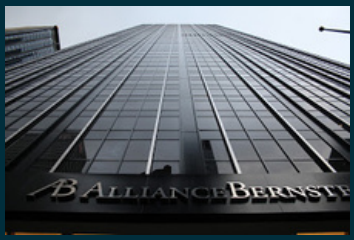
FILTROS DE REALCE

Filtro de Sobel

- **Realiza uma operação para realçar contornos em imagens;**
- **Realça as linhas verticais e horizontais mais escuras que o fundo, sem realçar pontos isolados;**
- **Resulta em uma imagem com bordas mais grossas.**

```
img_sobel = cv.Sobel(img, prof, dx, dy, ksize)
```

Os parâmetros que essa função recebe são, respectivamente, a imagem original, a profundidade da imagem de saída (cv.CV_8U ou cv.CV_64F), o realce horizontal, o realce vertical e o tamanho da máscara.



```
cv.Sobel(img,
cv.CV_8U, 0, 1,
  ksize = 7)
```

```
cv.Sobel(img,
cv.CV_8U, 1, 0,
  ksize = 7)
```

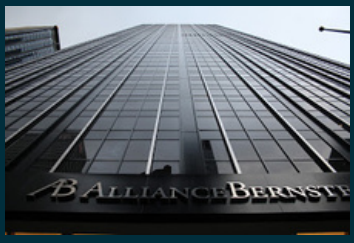
FILTROS

Filtro Laplaciano

- **Máscara de ordem 3 que percorre toda a imagem, alterando o pixel-alvo pela média ponderada dos pixels vizinhos e, depois, elevando ao quadrado;**
- **Produz bordas mais finas que o filtro de Sobel.**

```
img_laplaciano = cv.Laplacian(img, prof)
```

Os parâmetros que essa função recebe são, respectivamente, a imagem original e a profundidade da imagem de saída (cv.CV_8U ou cv.CV_64F).



```
img_laplaciano = cv.Laplacian(img, cv.CV_8U)
```

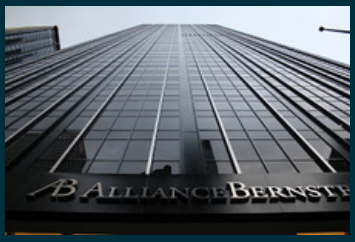
Filtro de Canny

- **Um dos algoritmos mais eficientes para detectar bordas em imagens;**
- **Deve ser capaz de identificar todas as bordas possíveis.**

FILTROS

```
img_canny = cv.Canny(img, det1, det2)
```

Os parâmetros que essa função recebe são, respectivamente, a imagem original e dois inteiros que ajustam a detecção de bordas.



```
img_canny = cv.Canny(img, 75, 5Ø)
```

Filtro edgePreservingFilter

- **suaviza ruídos e, ao mesmo tempo, mantém as bordas nítidas;**
- **usado onde as bordas das imagens são essenciais e devem ser preservadas nítidas e sem distorções na suavização dos ruídos.**

```
img_edge = cv.edgePreservingFilter(img, flags,
                sigma_s, sigma_r)
```

Os parâmetros são, respectivamente, a imagem original, uma flag que indica o método de preservação de bordas que será usado, um parâmetro que controla a sensibilidade ao detalhe da imagem e outro que controla a sensibilidade à intensidade das bordas.

FILTROS

```
img_edge = cv.edgePreservingFilter(img,
    flags=2, sigma_s=50, sigma_r=0.4)
```

FILTROS MORFOLÓGICOS

É uma transformação numa imagem binarizada usando uma segunda imagem menor, chamando de elemento estruturante, que é usada para ressaltar ou remover aspectos específicos da imagem original.

O OpenCV possui três elementos estruturantes: retangular, elíptico e em cruz.

```
elem_estr = cv.getStructuringElement(metodo,
                    dimensao)
```

Os métodos podem ser cv.MORPH_RECT, cv.MORPH_ELLIPSE e cv.MORPH_CROSS, que geram, respectivamente, os elementos estruturantes retangular, elíptico e em cruz.

FILTROS

Filtro de Erosão

- **Realiza a corrosão das arestas do objeto de interesse, encolhendo-o;**

```
img_erosao = cv.erode(img, elem_estr, iterations)
```

Os parâmetros são, respectivamente, a imagem original, o elemento estruturante e o número de iterações.

Filtro de Dilatação

- **A área do objeto de interesse será dilatada, ou seja, ficará maior.**

```
img_dilatacao = cv.dilate(img, elem_estr, iterations)
```

Os parâmetros são, respectivamente, a imagem original, o elemento estruturante e o número de iterações.

`img`

```
elem_estr = cv.getStructuringElement(cv.MORPH_RECT, (7, 7))
```

FILTROS


```
cv.erode(img,
elem_estr,
iterations=2)
```

```
elem_estr2 =
cv.getStructuringElement(cv.MORPH_ELLIPSE,
(7, 7))
```


```
cv.dilate(img,
elem_estr2,
iterations=2)
```

OUTROS TIPOS:

- **Gradiente Morfológico: diferença entre dilatação e erosão, apresentação a borda do objeto;**
- **TopHat: diferença entre a imagem original e a abertura da imagem. Realça objetos brilhantes em fundos escuros e corrige variação de luminosidade;**
- **BlackHat: diferença entre o fechamento da imagem e a imagem original.**

```
img_processada = cv.morphologyEx(img,
metodo, elem_estr)
```

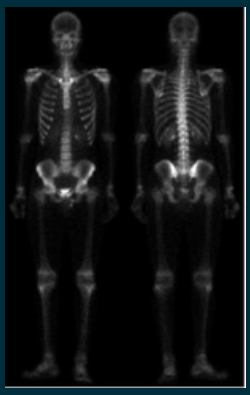
FILTROS

Os parâmetros são, respectivamente, a imagem original, o método que indicará qual filtro será aplicado (cv.MORPH_GRADIENT, cv.MORPH_TOPHAT e cv.MORPH_BLACKHAT).

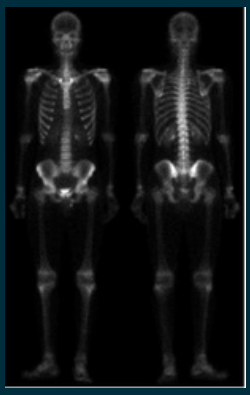
```
elem_estr3 =
cv.getStructuringElement(cv.MORPH_ELLIPSE,
(5,5))

elem_estr4 =
cv.getStructuringElement(cv.MORPH_ELLIPSE,
(21,21))
```

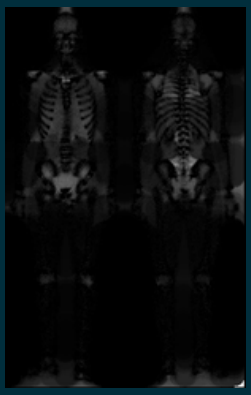

```
img_gradiente = cv.morphologyEx(img1,
    cv.MORPH_GRADIENT, elem_estr3)
```

FILTROS



```
img_tophat = cv.morphologyEx(img,
  cv.MORPH_TOPHAT, elem_estr4)
```





```
img_blackhat = cv.morphologyEx(img,
  cv.MORPH_BLACKHAT, elem_estr4)
```

DESENHOS

Para criar uma imagem inicial, usamos o código a seguir.

```
img = np.zeros((altura, largura, 3), dtype=tipo)
```

As variáveis altura e largura são a quantidade de pixels para a altura e a largura da imagem, respectivamente. A variável tipo é o tipo de dados que compõe a imagem, que usualmente é np.uint8.

Podemos definir a cor desse fundo com:

```
img[:] = (B, G, R)
```

sendo B, G e R os canais de cor do espaço BGR.

Para criar um retângulo, usamos a seguinte função.

```
cv.rectangle(img, ponto_inicial,
ponto_final, cor, espessura)
```

Os parâmetros dessa função são, respectivamente, a imagem na qual se vai inserir o retângulo, uma tupla (x, y) com o ponto inicial desse retângula, uma tupla (x, y) com o ponto final, uma tupla (B, G, R) que determina a cor e um valor que determina a espessura da borda do retângulo.

Para criar um círculo, usamos a seguinte função.

```
cv.circle(img, centro, raio, cor, espessura)
```

DESENHOS

Os parâmetros são, respectivamente, a imagem na qual se vai inserir o círculo, uma tupla (x, y) com o centro do círculo, um valor para o raio do círculo, uma tupla (B, G, R) que representa a cor do círculo e um valor para a espessura do círculo.

Para desenhar uma linha, usamos:

```
cv.line(img, ponto_inicial, ponto_final,
            cor, espessura)
```

Os parâmetros são, respectivamente, a imagem na qual se vai inserir o círculo, uma tupla (x, y) com o ponto inicial da linha, uma tupla (x, y) com o ponto final da linha, uma tupla (B, G, R) que representa a cor e um valor para a espessura dessa linha.

Para criar um polígono convexo, usa-se a função a abaixo.

```
cv.fillConvexPoly(img, pontos, cor)
```

Os parâmetros são, respectivamente, a imagem que se quer inserir o polígono, um numpy array com os vértices do polígono e a cor dele.
O numpy array é definido como:

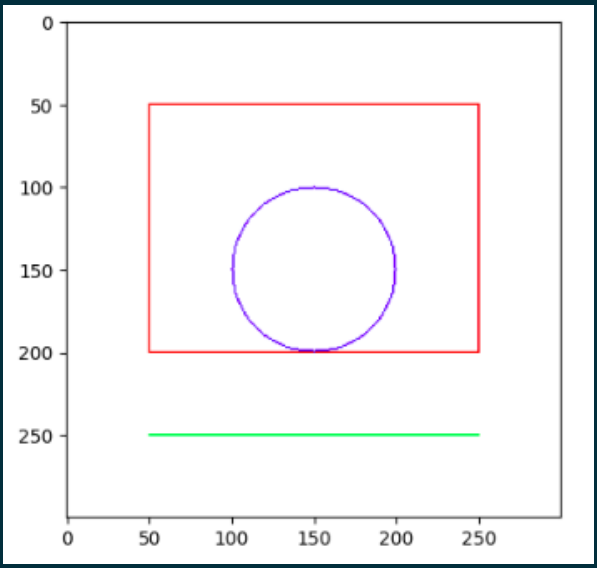
```
pontos = np.array([[x1, y1], [x2, y2], [x3,
 y3], [x4, y4], [x5, y5], ..., [xn, yn]])
```

DESENHOS

Existem outras funções possíveis, como cv.ellipse(), cv.fillPoly(), cv.putText(), que podem ser consultadas a partir da documentação da biblioteca OpenCV.

Para exemplificar, vamos mostrar uma código que faz o uso de algumas funções apresentadas.

```
img = np.zeros((300, 300, 3), dtype=np.uint8)
img[:] = (255, 255, 255)

cv.rectangle(img, (50,50), (250, 200), (0, 0, 255), 1)
cv.circle(img, (150, 150), 50, (255, 0, 0), 1)
cv.line(img, (50, 250), (250, 250), (0, 255, 0), 1)
```



TRANSFORMAÇÕES

TRANSFORMAÇÃO AFIM

A Transformação Afim é uma transformação linear que preserva paralelismo de linhas. Pode ser usada para rotações, translações e escalas.

```
img_saida = cv.warpAffine(img, matriz,
                tamanho)
```

A função warpAffine recebe como parâmetro a imagem original, a matriz que realiza a transformação e uma tupla (largura, altura) que representa o tamanho para a imagem de saída.

Rotação

Para obter a matriz de rotação, usa-se a seguinte função.

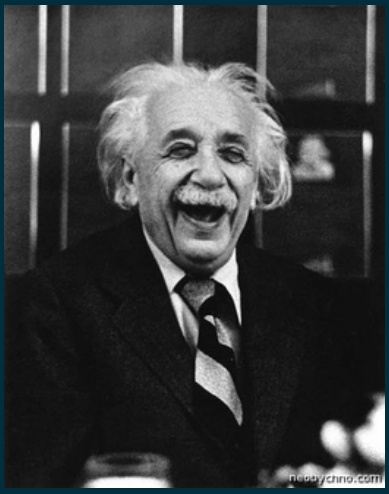
```
matriz = cv.getRotationMatrix2D(centro,
            angulo, escala)
```

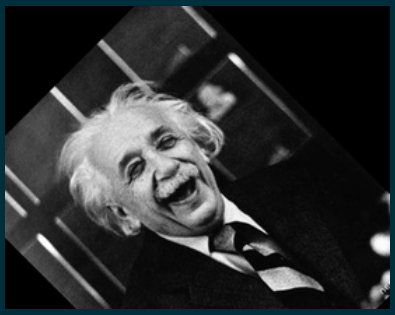
O parâmetro centro é uma tupla (x, y) com os pontos do centro de rotação. Normalmente, se rotaciona ao centro da imagem e, portanto, essa tupla é (largura/2, altura/2).

O parâmetro angulo é o valor do ângulo que se deseja girar a imagem. Por fim, escala é um opcional que se usa para redimensionar a imagem. Normalmente, não se deseja alterar a escala e, portanto, usa-se o valor 1.

TRANSFORMAÇÕES

Por exemplo, com o comando a seguir, obtemos essa rotação em 45 °.





```
linhas, colunas = img.shape
mat = cv.getRotationMatrix2D((linhas/2,
colunas/2), 45, 1)
imgRotacionada = cv.warpAffine(img, mat,
(linhas, colunas))
```

Translação

A matriz de translação é feita do modo clássico: criando um array numpy com tipo de dados float32.

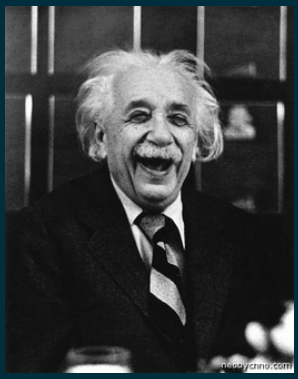
```
matriz = np.float32([[1, 0, dx], [0, 1, dy]])
```

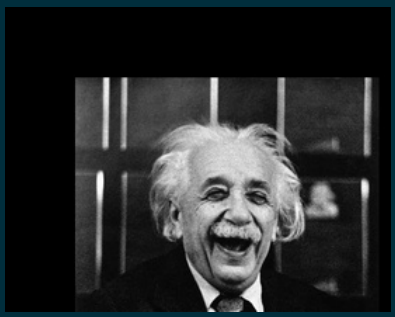
O valor dx é o deslocamento em x e o valor dy é o deslocamento em y.

TRANSFORMAÇÕES

Por exemplo, para deslocar a imagem em 100 em x e em 100 em y, usamos:

```
linhas, colunas = img.shape
mat = np.float32([[1, 0, 100], [0, 1, 100]])
imgDeslocada = cv.warpAffine(img, mat, (linhas, colunas))
```





Ajuste de Escala

O ajuste de escala de uma imagem sem distorção é feito pela função resize().

```
img_redimensionada = cv.resize(img,
  tamanho_final, interpolation)
```

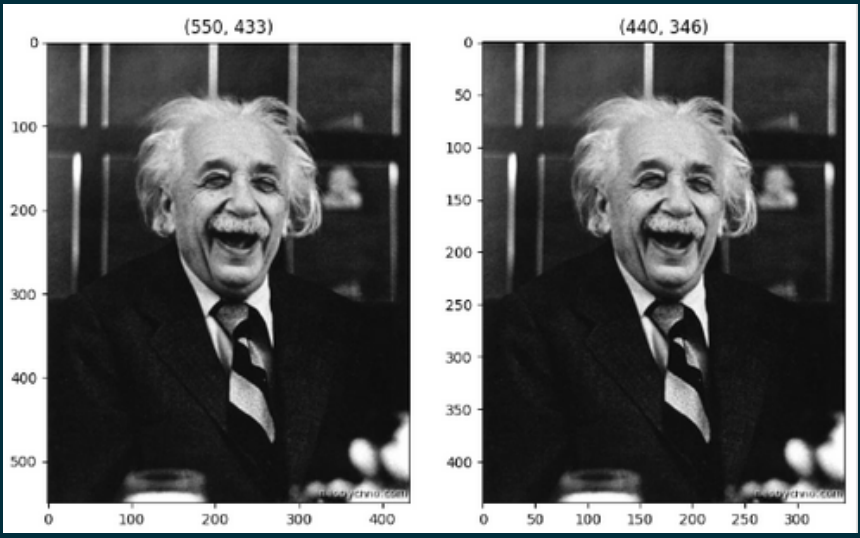
TRANSFORMAÇÕES

Os parâmetros são, respectivamente, a imagem original, uma tupla (x, y) com os tamanhos que se deseja em cada dimensão e o método de interpolação.

Os métodos podem ser `cv.INTER_AREA`, `cv.INTER_CUBIC` ou `cv.INTER_LINEAR`. As diferenças entre eles podem ser consultadas na documentação oficial da biblioteca.

Por exemplo, para reduzir o tamanho da imagem em 20% em x e em y, usamos:

```
imgModificada = cv.resize(imgOriginal, None,
        fx=0.8, fy=0.8, interpolation =
                cv.INTER_CUBIC)
```



TRANSFORMAÇÕES

TRANSFORMAÇÃO EM PERSPECTIVA

A Transformação em Perspectiva é usada quando a imagem precisa simular um ponto de vista diferente.

```
img_saida = cv.warpPerspective(img, matriz, tamanho)
```

A função warpPerspective recebe como parâmetro a imagem original, a matriz que realiza a transformação e uma tupla (largura, altura) que representa o tamanho para a imagem de saída.

Para obter a matriz de uma transformação em perspectiva, pode-se usar a seguinte função.

```
matriz = cv.getPerspectiveTransform(origem, destino)
```

Os parâmetros são, respectivamente, arrays numpy que definem quatro vértices de origem e de destino.

COMANDOS DE VÍDEO

Um vídeo é uma sequência de imagens em movimento. Cada imagem em um vídeo é chamada de frame. A quantidade de frames exibidos em um segundo é chamada de FPS (frames per second) e, quando ela é suficientemente alta, percebemos a sequência de imagens como um movimento contínuo.

WEBCAM

O comando cv.VideoCapture(0) retorna a captura do vídeo da WebCam frame por frame.

```
cap = cv.VideoCapture(0)

if not cap.isOpened():
  print('Erro!')
  exit()
```

Para exibir todos os frames na tela, precisamos criar um laço while.

```
while True:

  ret, frame = cap.read()
  if ret == True:

    cv.imshow('WebCam', frame)
    if cv.waitKey(1) & 0xFF == ord('q'):
      break

cap.release()
cv.destroyAllWindows()
```

COMANDOS DE VÍDEO

A função .read() faz a leitura da WebCam e armazena em ret um booleano que indica se a leitura foi bem sucedida e em frame o frame lido.

O if colocado no código aguarda 1 ms para capturar a entrada do teclado e, quando essa entrada for a letra q, entra no if e quebra o loop.

Após o laço, usamos a função .release(), que serve para liberar a WebCam, e cv.destroyAllWindows() para fechar a janela criada.

VÍDEOS

Para ler um video, a função cv.VideoCapture() também é usada. No entanto, aqui o parâmetro usado é o nome do arquivo do vídeo.

```
cap = cv.VideoCapture('video1.mp4')

if not cap.isOpened():
  print('Erro!')
  exit()
```

Para exibir esse vídeo, usa-se o mesmo while do tópico anterior.

Inserção de Formas no Vídeo

Por exemplo, caso queiramos inserir um retângulo no vídeo, basta usar a função de cv.rectangle() vista anteriormente, lembrando que a imagem é o frame.

COMANDOS DE VÍDEO

Dentro do while, coloca-se:

```
 cv.rectangle(frame, (x, y), (x + w, y + h),
(0, 0, 255), 4)
```

Com isso, cria-se um retângulo vermelho de espessura 4 iniciando no ponto (x, y) e terminando no ponto (x + w, y + h), em que w e h são a largura e a altura do retângulo, respectivamente.

Nesse caso, obviamente, o retângulo está estático dentro do vídeo. Posteriormente, serão mostradas técnicas para inserir formas geométricas que se movimentam de acordo com pontos detectados em um frame.

SEGMENTAÇÃO DE OBJETOS

A segmentação de objetos consiste em representar somente os objetos de interesse em uma nova imagem.

Segmentação por Binarização

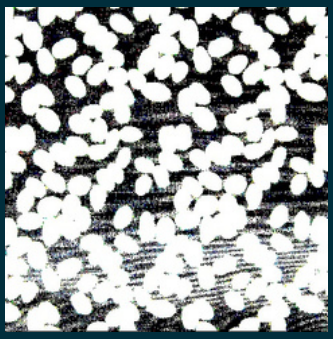
- **Conhecida pela aplicação do limiar de intensidade. A separação do objeto de interesse ocorre pela definição de um limiar;**
- **Os pixels com valores maiores do que o limiar são determinados objetos de interesse, sendo redefinidos para a cor preta ou branca;**
- **Os pixels com valores menores que o limiar são estabelecidos como segundo plano, sendo redefinidos para a cor oposta a do objeto de interesse;**
- **Assim, ao final, obtemos aquilo que denominados por imagem binária: contida apenas por pixels brancos e pretos.**

```
ret, img_binaria = cv.threshold(img, limiar,
                    max_valor, metodo)
```

A função acima possui como parâmetros, respectivamente, a imagem original, o valor limiar abaixo do qual o pixel é definido como parte do objeto de interesse, o valor a ser atribuído aos pixels que passarem o limiar e o método de binarização. Retorna o valor limiar usadoe a imagem binária.

SEGMENTAÇÃO DE OBJETOS

O método pode ser cv.THRESH_BINARY, que atribui cor preta ao objeto de interesse e o valor de 'max_valor' ao segundo plano. Alternativamente, temos o oposto: cv.THRESH_BINARY_INV, que atribui cor definida em 'max_valor' ao objeto de interesse e preto ao segundo plano.



```
ret, img_binaria = cv.threshold(img, 200, 255,
cv.THRESH_BINARY_INV)
```

Segmentação Adaptativa

- **Usada quando a imagem não possui iluminação adequada ao procedimento de binarização;**
- **Calcula diferentes valores de limiar para cada região da imagem.**

```
img_binaria = cv.adaptativeThreshold(img,
max_valor, metodo, tipo_threshold, mascara,
cte)
```

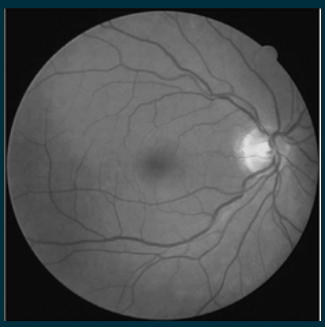
SEGMENTAÇÃO DE OBJETOS

A função possui como parâmetros, respectivamente, a imagem original, o valor a ser atribuído aos pixels que passarem o limiar, o método de cálculo do limiar adaptativo, o tipo de limiarização (são válidos os mesmos da cv.threshold), o tamanho da máscara para calcular o limiar adaptativo e uma constante que subtrai o valor do limiar adaptativo, permitindo ajustá-lo.

Os métodos para cálculo do limiar são:

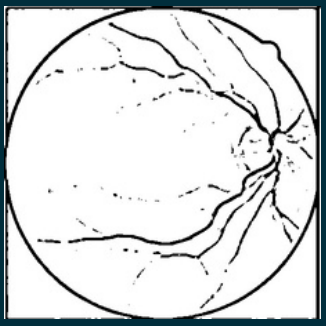
- **cv.ADAPTIVE_THRESH_MEAN_C: o valor de limiar é a média dos valores na vizinhança do pixel definida pelo tamanho do bloco;**

- **cv.ADAPTIVE_THRESH_GAUSSIAN_C: o valor de limiar é calculado usando uma média ponderada de valores na vizinhança do pixel, em que os pesos são determinados pela distribuição gaussiana.**

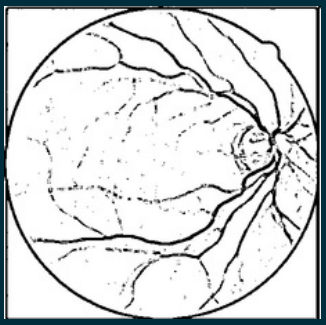


img

SEGMENTAÇÃO DE OBJETOS



```
cv.adaptiveThreshold(img, 255,
  cv.ADAPTIVE_THRESH_MEAN_C,
   cv.THRESH_BINARY, 11, 5)
```

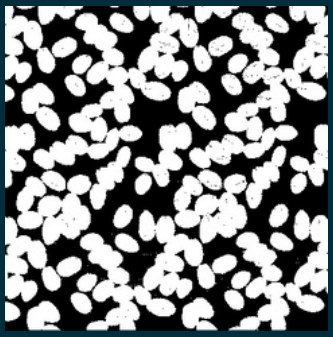


```
cv.adaptiveThreshold(img, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C,
   cv.THRESH_BINARY, 11, 2)
```

Obs.: Binarização por Otsu

- **Define um limiar baseado no histograma da imagem;**
- **Basta usar a função threshold(), definindo o valor do limiar como nulo e somando a constante cv.THRESH_OTSU no método.**

SEGMENTAÇÃO DE OBJETOS



```
metodo = cv.THRESH_BINARY_INV + cv.THRESH_OTSU
ret, img_binaria = cv.threshold(img, 0, 255,
metodo)
```

Transformada de Hough

- **Trata-se de uma técnica popular para detecção de qualquer forma que pode ser representada matematicamente;**

- **Dentre as funções do OpenCV que usam essa técnica, temos, por exemplo, HoughLines(), HoughLinesP(), HoughCircles(), HoughEllipses() e HoughRectangles(). Para mais detalhes sobre cada uma delas, deve-se consultar a documentação da biblioteca;**

- **Vamos exemplificar a função HoughLinesP(), que serve para detectar segmentos de linhas em uma imagem.**

```
linhas = cv.HoughLinesP(img, rho, theta,
            limiar, min, max)
```

SEGMENTAÇÃO DE OBJETOS

Os parâmetros dessa função são, respectivamente, uma imagem binária, o parâmetro rho (usualmente 1), o parâmetro theta (usualmente 1º), o valor limiar, o comprimento mínimo para ser considerado linha e o valor máximo para ser considerado linha. O retorno dessa função é uma matriz numpy, sendo que cada elemento dela é um vetor que contém os 4 pontos que definem a linha: x1, y1, x2, y2, sendo (x1, y1) o ponto inicial e (x2, y2) o ponto final.

Por exemplo, queremos identificar as linhas dessa estrada para conseguir reconhecer seus limites.

O primeiro passo é torná-la binária. A função Canny() nos fornece uma imagem binária e detecta bordas.

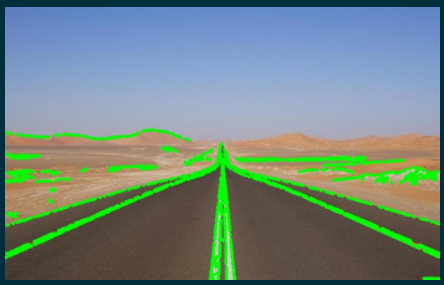
```
bordas = cv.Canny(img, 70, 255)
```

SEGMENTAÇÃO DE OBJETOS

Após isso, podemos usar a função que aplica a transformada de Hough. O valor limiar definido para detecção das linhas é 10 e o comprimento mínimo para ser dito uma linha é 200.

```
linhas = cv.HoughLinesP(bordas, 1, np.pi/180,
                        10, 200)
```

Assim, temos um vetor de pontos que são coordenadas para linhas na imagem. Assim, basta que usemos a função lines() para desenhar essas linhas na imagem.



```
for linha in linhas:
 x1, y1, x2, y2 = line[0]
 cv.line(img, (x1, y1), (x2, y2), (0, 255, 0), 3)
```

EXTRAÇÃO DE CARACTERÍSTICAS

O OpenCV nos permite obter informações sobre um objeto. Podemos obter algumas características, como aspectos de cor, dimensionais e inerciais.

COR

A função mean() retorna a média de todos os canais BGR. Além da média dos canais de cor, retorna-se um quarto parâmetro, que é o valor alpha da imagem, que representa sua transparência.

```
B, G, R, alpha = cv.mean(img)
```

A função meanStdDev() calcula a média e o desvio padrão de forma independente para cada canal de cor e os retorna na saída.

```
mean, stdDev = cv.meanStdDev(img)
```

ASPECTOS DIMENSIONAIS

Após fazer a segmentação por binarização, podemos obter dados de área, perímetro e diâmetro.

A função findContours() extrai da imagem binária os pontos que representam os contornos dos objetos segmentados.

```
contours = cv.findContours(img, modo, metodo)
```

EXTRAÇÃO DE CARACTERÍSTICAS

Os parâmetros são, respectivamente, a imagem que contém o objeto, a defanação de como os contornos são organizados e a definição de como os contornos são aproximados. O retorno é uma lista de arrays numpy, representando os contornos encontrados na imagem.

Com essa lista, podemos usar a função contourArea() para obter a área dos objetos. A área é definida como a quantidade de pixels na região delimitada pelo contorno.

```
area = cv.contourArea(contour)
```

Além disso, podemos obter os perímetros das formas com a função arcLength(). A entrada é um array numpy do contorno do objeto e um booleano que indica se o contorno é fechado ou aberto.

```
perimetro = cv.arcLength(contour, bool)
```

ASPECTOS INERCIAIS

A extração de inerciais em uma imagem refere-se ao processo de calcular propriedades que descrevem a distribuição de massa ou intensidade de pixels na imagem.

É possível reconhecer objetos a partir dessas características, mesmo que eles tenham sofrido alterações de escala, rotação e translação.

EXTRAÇÃO DE CARACTERÍSTICAS

A função moments() retorna os 24 momentos de uma imagem. É aconselhável fazer uma binarização da imagem antes de aplicar essa função.

```
momentos = cv.moments(img)
```

Com os elementos da variável momentos, podemos encontrar, por exemplo, a área e o centro de massa.

```
area = momentos['m00']

cx = momentos['m10'] / momentos['m00']
cy = momentos['m01'] / momentos['m00']
```

Além dessa função, existe a função HuMoments(), que calcula os Momentos de Hu, os quais são momentos invariantes usados para representar as características de uma imagem.

```
momentos_hu = cv.HuMoments(momentos).flatten()
```

Obs.: a função flatten() apenas transforma um array multidimensional para um array unidimensional.

DETECÇÃO DE CANTOS

Um canto é a intersecção de duas arestas e representa um ponto no qual as direções dessas duas arestas mudam.

EXTRAÇÃO DE CARACTERÍSTICAS

A técnica Harris Corner Detector cria uma matriz 2x2 a partir das derivadas parciais da imagem em escala de cinza e análise dos autovalores dessa matriz.

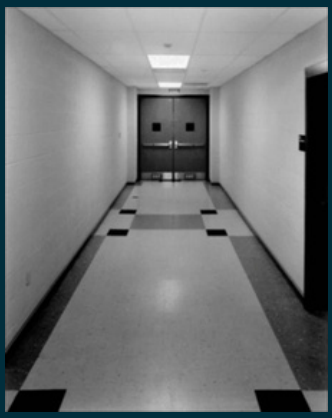
```
img_corner = cv.cornerHarris(img, mascara,
                             ksize, k)
```

Os parâmetros são, respectivamente, a imagem original em escalas de cinza, o tamanho da máscara usada para calcular a matriz de covariância dos gradientes (usualmente 2,3 ou 5), o tamanho da janela do filtro de Sobel usado para calcular os gradientes (usualmente 3) e o parâmetro k que é uma constante da fórmula de Harris para detecção de cantos (usualmente um valor entre 0,04 e 0,06).

O retorno é uma imagem de mesma dimensão da original com os cantos realçados. Normalmente, fazemos uma dilatação para identificar esses cantos com precisão.

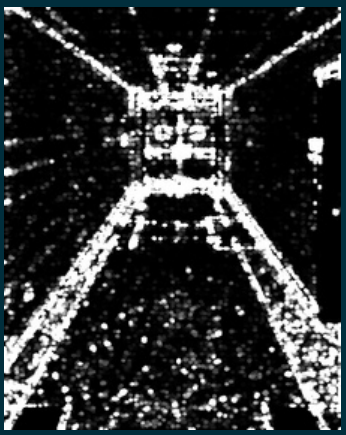
Por exemplo, vamos detectar os cantos da imagem ao lado. Começamos convertendo para a escala de cinzas.

```
gray =
cv.cvtColor(img,
cv.COLOR_BGR2GRAY)
```



EXTRAÇÃO DE CARACTERÍSTICAS

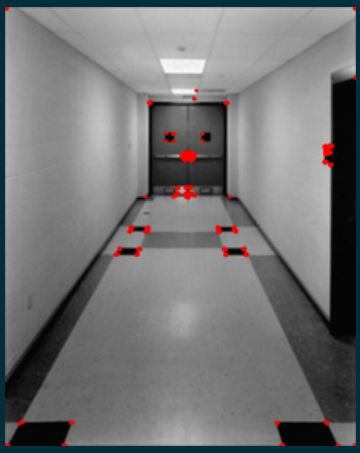
A seguir, aplica-se a função de Harris Corner e uma dilatação, obtendo a seguinte imagem.



```
img_corner = cv.cornerHarris(gray, 2, 3,
Ø.Ø1)

elem_estr =
cv.getStructuringElement(cv.MORPH_ELLIPSE,
(5,5))

img_corner = cv.dilate(img_corner,
elem_estr)
```

Na sequência, precisamos fazer a correspondência dos pontos detectados na imagem original. Assim, obtemos uma imagem com os cantos detectados colocados em vermelho.

EXTRAÇÃO DE CARACTERÍSTICAS



```
img[img_corner > 0.05*img_corner.max()] =
[0,0,255]
```

Isso significa que os pontos da imagem obtida por Harris Corner que tiverem valores maiores do que 5% do valor máximo dessa imagem são cantos e, portanto, serão pintados de vermelho.