

# Processamento de Imagens



Cruzeiro do Sul Virtual  
Educação a distância



# Material Teórico



**Restauração, Reconstrução e Morfologia em Imagens**

**Responsável pelo Conteúdo:**

Prof. Me. Josivan Pereira da Silva

**Revisão Textual:**

Prof.<sup>a</sup> M.<sup>a</sup> Sandra Regina Fonseca Moreira



# UNIDADE

## Restauração, Reconstrução e Morfologia em Imagens



- Morfologia;
- Restauração em Imagens.



### OBJETIVOS DE APRENDIZADO

- Entender os filtros morfológicos dos tipos: Erosão e Dilatação, Abertura e Fechamento, para realizar pré-processamento, melhorias em imagens e pós-processamento;
- Aprender de forma teórica e prática a realizar restauração, reconstrução e melhorias em imagens.





# Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



## Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

# Morfologia

Morfologia é um termo diferente e que pode assustar no início, mas não se trata de algo difícil e complexo. A Morfologia empregada em imagens é até bem simples.

Tem uma relação com a Matemática, mas é uma parte tranquila e relativamente simples.

A Morfologia é um conceito matemático que é empregado em Processamento de Imagens, consta do estudo da decomposição de operadores em reticulados (ou *pixels*) e os principais tipos são a Erosão, a Dilatação, a Abertura e o Fechamento (RODRIGUES; LEONARDI, 2009).

Vamos estudar este assunto de uma forma mais prática e voltada para a execução de códigos em *Python* com *OpenCV*, ainda assim, um pouco de teoria, de vez em quando, vai nos ajudar bastante.

As operações morfológicas contêm uma matriz especial para a realização do processamento, que é chamada de elemento estruturante.

As operações morfológicas funcionam melhor e são entendidas mais facilmente quando operadas em imagens de um único canal com cores em preto e branco, é assim que vamos utilizar as imagens aqui.

Em *OpenCv* as matrizes de elementos estruturantes disponíveis são dos tipos **MORPH\_RECT**, **MORPH\_ELLIPSE** e **MORPH\_CROSS**. Essas matrizes têm elementos formados por zeros e uns (0 e 1) que definem cores pretas (0) e brancas (1). Elas têm os formatos de retângulo, elipse ou cruz, como os nomes sugerem, respectivamente.

Ao pegar a forma do elemento estruturante com o código *Python* com *OpenCV*, para a forma de retângulo:

```
cv.getStructuringElement(cv.MORPH_RECT,(5,5))
```

O resultado após um comando de *print* para verificação é:

```
array([[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]], dtype=uint8)
```

Ao pegar a forma do elemento estruturante com o código *Python* com *OpenCV*, para a forma de elipse:

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

O resultado após um comando de *print* para verificação é:

```
array([[0, 0, 1, 0, 0],  
       [1, 1, 1, 1, 1],  
       [1, 1, 1, 1, 1],  
       [1, 1, 1, 1, 1],  
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Ao pegar a forma do elemento estruturante com o código *Python* com *OpenCV*, para a forma de cruz:

```
cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
```

O resultado após um comando de print para verificação é:

```
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Esses elementos estruturantes podem ser aplicados nas operações morfológicas de Erosão, Dilatação, Abertura ou Fechamento.

Vamos utilizar a Figura 1 como exemplo base para os códigos a serem explicados:



Figura 1 – Imagem de exemplo

Fonte: Reprodução

## Erosão

---

A Erosão refere-se a utilizar alguma matriz chamada de elemento estruturante para reduzir as partes mais claras da imagem e aumentar as regiões com partes mais escuras. Na prática, ela escurece a imagem e pode ser utilizada estratégicamente para reduzir alguns tipos de ruídos e restaurar imagens (ANDRADE, 2014).

Podemos ver que a imagem da Figura 1 é constituída por regiões pretas (escuas) e regiões brancas (claras). Aplicando a Erosão na Figura 1, teremos um resultado que irá diminuir as regiões brancas e, consequentemente, escurecer a imagem.

Ao aplicar a Erosão sobre a Figura 1, teremos a Figura 2 como resultado, a parte da esquerda é a imagem original, e a da direita é o resultado da Erosão. Como podemos ver, a imagem da direita teve a região branca reduzida e a preta ampliada.



Figura 2 – Aplicação da Erosão na imagem de exemplo

Fonte: Reprodução

O código utilizado para aplicar a Erosão foi:

```
import cv2
import numpy as np

img = cv2.imread('j.png',0)
kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
erosion = cv2.erode(img,kernel,iterations = 1)

res = np.concatenate((img,erosion), axis=1)

cv2.imshow('Imagen Comum J, Com Erosão', res)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Dilatação

A Dilatação refere-se a utilizar alguma matriz chamada de elemento estruturante para ampliar as partes mais claras da imagem e diminuir as regiões com partes mais escu- ras. Na prática, ela clareia a imagem e pode ser utilizada estratégicamente para reduzir alguns tipos de ruídos e restaurar imagens (SILVA, 2015).

Podemos ver que a imagem da Figura 1 é constituída por regiões pretas (escu- ras) e regiões brancas (claras). Aplicando a Dilatação na Figura 1, teremos um resultado que irá aumentar as regiões brancas e, consequentemente, clarear a imagem.

Ao aplicar a Dilatação sobre a Figura 1, teremos a Figura 3 como resultado, a parte da esquerda é a imagem original, e a da direita é o resultado da Dilatação. Como pode- mos ver, a imagem da direita teve a região branca ampliada e a preta reduzida.



Figura 3 – Aplicação da Dilatação na imagem de exemplo

Fonte: Reprodução

Podemos concluir que a operação de Dilatação é oposta a da Erosão, pois ela faz o inverso do que a Erosão faz, e vice-versa.

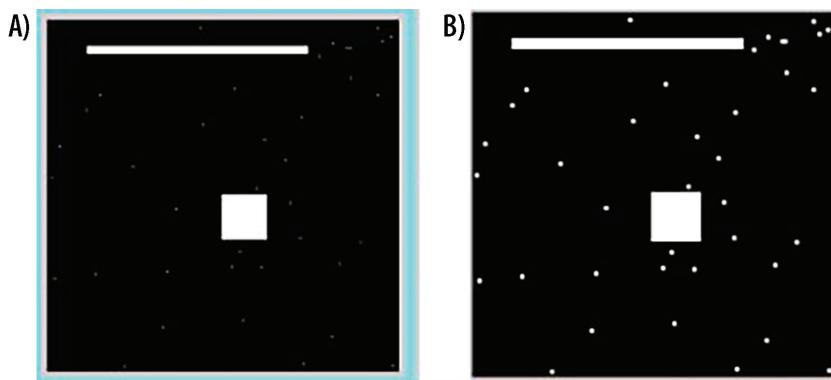


Figura 4 – Em a) um imagem de entrada contendo ruído.  
 Em b) a imagem de saída após a dilatação (ruído destacado)  
 Fonte: Reprodução

Essas operações podem ser combinadas, com ordens de combinação distintas e com resultados diferentes, para compor duas novas operações. As operações de Abertura e Fechamento.

O código utilizado para aplicar a Dilatação foi:

```
import cv2
import numpy as np

img = cv2.imread('j.png',0)
kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))

dilate = cv2.dilate(img,kernel,iterations = 1)

res = np.concatenate((img,dilate), axis=1)

cv2.imshow('Imagen Comum J, Com Dilatação', res)

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

Os programas demonstrados neste material serão melhor explicados na vídeo-aula da unidade.

## Abertura

---

A operação de Abertura de uma imagem de entrada por um elemento estruturante é definida pela aplicação de uma operação de Erosão, que gera uma imagem de saída, e aplicação de uma operação de Dilatação na imagem de saída, respectivamente nessa ordem: Erosão e depois Dilatação (NETO; MESQUITA, 2010).



Figura 5 – À esquerda, uma imagem de entrada com ruído e, à direita, o resultado após a operação de Abertura

Fonte: Reprodução

O código utilizado para aplicar a Abertura foi:

```
import cv2
import numpy as np

img = cv2.imread('j_noisy.png',0)
kernel = np.ones((5,5),np.uint8)

openingD = cv2.erode(img,kernel,iterations = 1)
openingE = cv2.dilate(openingD,kernel,iterations = 1)

res1 = np.hstack((img, openingE))

cv2.imshow('J com ruído, Sem ruído após abertura', res1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Fechamento

A operação de Fechamento de uma imagem de entrada por um elemento estruturante é definida pela aplicação de uma operação de Dilatação, que gera uma imagem de saída, e aplicação de uma operação de Erosão na imagem de saída, respectivamente nessa ordem: Dilatação e depois Erosão, ou seja, é a ordem contrária da Abertura, sendo também a operação inversa (NETO; MESQUITA, 2010).



Figura 6 – À esquerda, uma imagem de entrada com ruído interno e, à direita, o resultado após a operação de Fechamento

Fonte: Reprodução

O código utilizado para aplicar o Fechamento foi:

```
import cv2
import numpy as np

img = cv2.imread('j_inside.png',0)
kernel = np.ones((5,5),np.uint8)

closing1 = cv2.dilate(img,kernel,iterations = 1)
closing2 = cv2.erode(closing1,kernel,iterations = 1)

res1 = np.hstack((img, closing2))

cv2.imshow('J com ruído interno, Fechamento corrigiu', res1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Restauração em Imagens

Em se tratando de Restauração de imagens, podemos dividir o assunto em duas partes: o realce de imagens e a recuperação de imagens corrompidas/degradadas. Para a parte de Realce de Imagens, podem ser aplicados filtros para eliminação de ruídos.

### Eliminação de Ruídos

Para estudarmos a eliminação de ruídos para o objetivo de restaurar uma imagem, vamos considerar a seguinte imagem com ruído como imagem de entrada (Figura 7).



Figura 7 – Imagem com ruído do tipo Salt and Pepper  
Fonte: KOLI, 2012

Para eliminar esses ruídos da imagem de entrada podemos utilizar um filtro do tipo Mediana.

O filtro de Mediana é um filtro do tipo passa-baixa, não linear no domínio espacial. Sua característica é a de suavizar a imagem, porém, minimizando bastante o efeito de “borramento”. O processo consiste em ordenar os *pixels* da vizinhança e o valor do *pixel* central passa a ser o valor mediano dos vizinhos. Este filtro é ideal para o tratamento de ruídos do tipo “sal e pimenta” (GUILHON, 2013).

Aplicando o filtro de Mediana na Figura 7, a saída é como na Figura 8.



Figura 8 – À esquerda, a imagem com ruído do tipo Salt and Pepper e, à direita, a saída com ruído amenizado pelo filtro de Mediana

Fonte: KOLI, 2012

O código utilizado para aplicar o filtro de Mediana foi:

```
import cv2
import numpy as np

img = cv2.imread('lena_noisy.png',0)

median = cv2.medianBlur(img,3)

res1 = np.hstack((img, median))

cv2.imshow('Imagen Entrada, Imagen mediana', res1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Equalização de Histograma

A equalização de histograma é uma operação muito utilizada em Processamento de Imagens para que seja possível ajustar os níveis de cinza de uma imagem automaticamente, resultando em um brilho e contraste balanceados de forma rápida e fácil (CARNEIRO, 2010).

É bom para gerar um realce em imagens e permitir melhor visualização das informações.

O histograma de uma imagem é bastante utilizado em processamento de imagens digitais por ser de fácil operação. O histograma de níveis de cinza de uma imagem digital

é uma função que apresenta, para cada nível, o número de *pixels* da imagem que tem aquele valor de cinza. Na abscissa (no eixo X) estão os níveis de cinza e na ordenada (no eixo Y) a frequência de ocorrência (ESTRELA *et al.*, 2012).

A equalização de histograma é uma técnica de transformação de intensidade que tem por objetivo balancear os níveis de cinza em uma imagem de forma automática, sem a necessidade de muitos ajustes em parâmetros e configurações adicionais. Imagens com um nível de brilho desbalanceado, ou seja, claras ou escuras demais, atingem uma distribuição normalizada com a Equalização de Histograma, o que garante um melhor contraste e visualização dos detalhes presentes na cena (CARNEIRO, 2010).

Uma característica interessante é que a soma de todos os níveis de cinza da imagem tem o mesmo valor da área da imagem, ou seja, número total de *pixels*. Outra é que se um objeto está numa tonalidade acima de um determinado valor de cinza, o somatório do histograma para valores acima desse limiar é a área do objeto (ESTRELA *et al.*, 2012).

As operações sobre histograma permitem obter informações que não estavam facilmente visíveis na imagem, inicialmente. Entre essas informações podemos destacar, a transformação de intensidade, a equalização, especificação e a limiarização. A transformação de intensidade é útil quando desejamos ressaltar uma determinada faixa de valores de tons de cinza para melhor definir uma parte da cena (ESTRELA *et al.*, 2012).

Na Figura 9, por exemplo, não é possível ver nitidamente importantes informações da imagem, pois a fotografia parece ser tirada em um local e momento em que havia muita neblina. Para realçar as informações dessa imagem e tornar os detalhes dela mais perceptíveis, a Equalização de Histograma pode ser uma boa opção de processamento.



Figura 9 – Imagem com prédios e árvores apresentando uma baixa nitidez, por conta de neblina

Fonte: Reprodução

Para aplicarmos a Equalização de Histograma, podemos executar o seguinte código Python com OpenCV para obter o resultado da Figura 10:

```
import numpy as np
import cv2 as cv

img = cv.imread('tempo.jpg',0)
```

```

equ = cv.equalizeHist(img)

res = np.concatenate((img, equ), axis=1)

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

cv.waitKey(0)
cv.destroyAllWindows()

```



Figura 10 – À esquerda, temos a imagem de entrada com pouca nitidez, à direita, temos a imagem de saída, melhorada, após a Equalização de Histograma

Fonte: Reprodução

## *Telea Inpaint*

---

O Algoritmo chamado *Telea Inpaint* é uma técnica de Processamento de Imagens que permite reconstruir uma imagem danificada/rasurada. Dessa maneira, podemos pegar fotografias degradadas e restaurá-las. Para que isso seja possível, é necessário ter uma imagem degradada e uma máscara (imagem auxiliar) em cores preto e branco apenas (1 canal), e que tenha foco no padrão de degradação da fotografia. A Figura 11 é um bom exemplo de aplicação do algoritmo de *Telea Inpaint*:



Figura 11

Fonte: Reprodução

À esquerda, temos a imagem degradada por riscos em cor preta, no centro, temos a máscara com o mesmo padrão da degradação, só que em preto e branco e, à direita, temos a imagem restaurada.

Há casos em que a imagem pode estar com degradação, rasuras e também ruídos, por ser tratar de fotografias muito antigas, que foram molhadas etc. Nesses casos, somente o processamento do *Tela Inpaint* pode não bastar, provavelmente, além do *Inpaint*, pode ser necessário algum filtro para remoção de ruídos, uma Equalização de Histogramas, ou outras técnicas. É possível e recomendável combinar técnicas sempre que possível para atingir um melhor resultado.

Certamente que, sempre que houver rasuras ou degradações na imagem, o processamento do *Telea Inpaint* já vai melhorar bastante a imagem, ainda assim, será necessário utilizar mais técnicas de forma combinada.

A Figura no link abaixo é um bom exemplo de uma imagem antiga que, além do processamento do *Telea Inpaint*, vai necessitar de outras técnicas combinadas.



À esquerda, temos a imagem degradada por riscos em cor preta, ao lado, temos a máscara com o mesmo padrão da degradação, só que em preto e branco e, abaixo, temos a imagem restaurada, disponível em: <https://bit.ly/35UecDw>



### Importante!

É bom lembrar que para testar todos os códigos mostrados e disponibilizados neste material, você deve ter as imagens apropriadas, com os mesmos nomes e extensões que aparecem nos códigos, caso não tenha essas imagens com os nomes e extensões corretas, o seu código irá falhar na execução.

O código utilizado para realizar o processamento do *Telea Inpaint* é disponibilizado a seguir:

```
import numpy as np
import cv2 as cv

img = cv.imread('OpenCV_Logo_B.png')

mask = cv.imread('OpenCV_Logo_C.png',0)

dst = cv.inpaint(img,mask,3,cv.INPAINT_TELEA)

mask = cv.merge((mask,mask,mask))

res1 = np.concatenate((img, mask), axis=1)
res2 = np.concatenate((res1,dst), axis=1)

cv.imshow('dst',res2)

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

A Figura 12 é o resultado da execução do código acima.

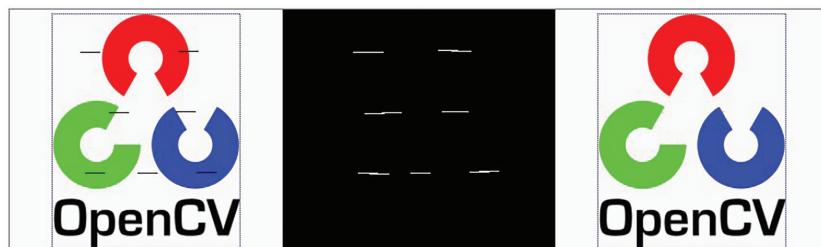


Figura 12

Fonte: Reprodução

À esquerda, temos a imagem degradada por riscos em cor preta, no centro, temos a máscara com o mesmo padrão da degradação, só que em preto e branco e, à direita, temos a imagem restaurada.

É importante ressaltar que os programas demonstrados neste texto serão melhor explicados na videoaula da unidade.

# Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

## Leitura

*Guidefill: Gpu Accelerated, Artist Guided Geometric Inpainting for 3D Conversion Of Film*

HOCKING, L. R.; RUSSELL MACKENZIE, R.; SCHONLIEB, C. B. *Guidefill: Gpu Accelerated, Artist Guided Geometric Inpainting for 3D Conversion Of Film*, 2017.

<https://bit.ly/35UuigB>

*Smoothing Images*

MORDVINTSEV, A.; ABID, K. *Smoothing Images*. 2013.

<https://bit.ly/2M23tQv>

*Morfologia Matemática*

<https://bit.ly/3nZGPoU>

*Image inpainting with OpenCV and Python*

<https://bit.ly/38VbrDW>

# Referências

- ANDRADE, A. O. **Sistema de Contagem com Morfologia Matemática Fuzzy**. Tese (doutorado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica. Natal-RN, 2014.
- CARNEIRO, A. L. C. **Equalização de histograma em Python**. 2010. Disponível em: <<https://medium.com/data-hackers/equaliza%C3%A7%C3%A3o-de-histograma-em-python-378830368d60>>. Acesso em: 17/11/2020.
- ESTRELA, V.; ASSIS, J. **Processamento de Imagens: Histogramas**. 10.13140/2.1.5025.0564, 2012. Disponível em: <[https://www.researchgate.net/publication/270816861\\_Processamento\\_de\\_Imagens\\_Histogramas](https://www.researchgate.net/publication/270816861_Processamento_de_Imagens_Histogramas)>.
- GUILHON, R. J. **Fundamentos de Computação Gráfica**. 2013. Disponível em: <<http://webserver2.tecgraf.puc-rio.br/~raquelg/compgrafica/trabalho1.html>>. Acesso em: 17/11/2020.
- KOLI, M. A. *Robust Algorithm for Impulse Noise Detection*. Research Scholar, Department of Computer Science Tumkur university, 2012.
- NETO, R. A. V.; MESQUITA, M. E. R. V. **Introdução à Morfologia Matemática Binária e em Tons de Cinza**. 2010. Disponível em: <<http://www.ime.unicamp.br/~valle/PDFfiles/valente10.pdf>>. Acesso em: 17/11/2020.
- RODRIGUES, T. G.; LEONARDI, F. **O Uso de Morfologia Matemática na Detecção de Pistas em Autódromo**. 2009. Disponível em: <<http://www.civil.uminho.pt/revista/artigos/n35/Pag.%2033-44.pdf>>. Acesso em: 17/11/2020.
- SILVA, B. S. **Detecção automática de células via técnicas de morfologia matemática e processamento digital de imagens**. 2015. Disponível em: <<http://monografias.polli.ufrj.br/monografias/monopoli10013465.pdf>>. Acesso em: 17/11/2020.





**Cruzeiro do Sul**  
Educacional