

# **Processamento de imagens**

Enzo Borges Segala      Jonathan Gabriel Nunes Mendes  
Matheus Machado Cesar      Marcos Henrique Volpato de Moraes  
Miguel Predebon Abichequer      Nicolas Rosenthal Dal Corso  
Rafael Silveira Bandeira

11/11/2025

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Tarefas</b>	<b>3</b>
2.1	Interpolação em imagens coloridas . . . . .	3
2.1.1	Arquivo principal . . . . .	3
2.1.2	Funções auxiliares . . . . .	4
2.2	Realce de imagens no domínio espacial (da imagem) . . . . .	6
2.2.1	Arquivo principal . . . . .	6
2.3	Filtragem espacial . . . . .	7
2.3.1	Arquivos principais . . . . .	7
2.4	Dithering . . . . .	13
2.4.1	Arquivo principal . . . . .	13
2.5	Operações Geométricas . . . . .	13
2.5.1	Arquivo principal . . . . .	13

# 1 Introdução

...

## 2 Tarefas

### 2.1 Interpolação em imagens coloridas

#### 2.1.1 Arquivo principal

```
%% Interpolação em imagens coloridas

addpath(genpath(strcat(pwd, '/mfunctions')));

scale = 1.8;
baseimgs = {'peppers.png', 'onion.png'};

imgs = cell(1, 8);
names= cell(1, 8);

for k = 1:2
    original = im2double(imread(baseimgs{k}));

    imgs(4*(k-1)+1:4*k) = { ...
        original, ...
        nearest_neighbour_resize(original, scale), ...
        bilinear_resize(original, scale), ...
        bicubic_resize(original, scale) };
    names(4*(k-1)+1:4*k) = {'Original', 'Nearest Neighbor', 'Bilinear', ...
        'Bicubic'};
end

imgs_in_docked_figures(imgs, names)
```

### 2.1.2 Funções auxiliares

```
function out = bicubic_channel(channel, scale)
    [rows, cols] = size(channel);
    new_rows = round(rows * scale);
    new_cols = round(cols * scale);
    out = zeros(new_rows, new_cols);
    [X, Y] = meshgrid(1:new_cols, 1:new_rows);
    X = X / scale; Y = Y / scale;

    for i = 1:new_rows
        for j = 1:new_cols
            x = X(i,j); y = Y(i,j);
            x1 = floor(x); y1 = floor(y);
            dx = x - x1; dy = y - y1;
            patch = zeros(4,4);
            for m = -1:2
                for n = -1:2
                    xm = min(max(x1 + m, 1), cols);
                    yn = min(max(y1 + n, 1), rows);
                    patch(n+2,m+2) = channel(yn, xm);
                end
            end
            out(i,j) = bicubic_interpolate(patch, dx, dy);
        end
    end
end

function val = bicubic_interpolate(patch, dx, dy)
% Cubic kernel
function w = cubic(t)
    a = -0.5; % Catmull-Rom
    abs_t = abs(t);
    if abs_t <= 1
        w = (a+2)*abs_t^3 - (a+3)*abs_t^2 + 1;
    elseif abs_t < 2
        w = a*abs_t^3 - 5*a*abs_t^2 + 8*a*abs_t - 4*a;
    else
        w = 0;
    end
end

wx = zeros(1,4); wy = zeros(1,4);
for i = 1:4, wx(i) = cubic(dx + 1 - i); end
for j = 1:4, wy(j) = cubic(dy + 1 - j); end
val = wy * patch * wx';
end
```

```

function out = bicubic_resize(img, scale)
    [rows, cols, ch] = size(img);
    new_rows = round(rows * scale);
    new_cols = round(cols * scale);
    out = zeros(new_rows, new_cols, ch);
    for k = 1:ch
        out(:, :, k) = bicubic_channel(img(:, :, k), scale);
    end
end

function out = bilinear_resize(img, scale)
    [rows, cols, ch] = size(img);
    new_rows = round(rows * scale);
    new_cols = round(cols * scale);
    [X, Y] = meshgrid(1:new_cols, 1:new_rows);
    X = round(X / scale); Y = round(Y / scale);
    X(X < 1) = 1; X(X > cols) = cols;
    Y(Y < 1) = 1; Y(Y > rows) = rows;

    out = zeros(new_rows, new_cols, ch);
    for k = 1:ch
        for i = 1:new_rows
            for j = 1:new_cols
                x = X(i, j); y = Y(i, j);
                x1 = floor(x); y1 = floor(y);
                x2 = min(x1 + 1, cols); y2 = min(y1 + 1, rows);
                a = x - x1; b = y - y1;
                Q11 = img(y1, x1, k);
                Q12 = img(y2, x1, k);
                Q21 = img(y1, x2, k);
                Q22 = img(y2, x2, k);
                out(i, j, k) = Q11*(1-a)*(1-b) + Q21*a*(1-b) + Q12*(1-a)*b
                    + Q22*a*b;
            end
        end
    end
end

```

```

function out = nearest_neighbour_resize(img, scale)
    [rows, cols, ch] = size(img);
    new_rows = round(rows * scale);
    new_cols = round(cols * scale);
    [J, I] = meshgrid(1:new_cols, 1:new_rows);
    I = ceil(I / scale); J = ceil(J / scale);
    I(I < 1) = 1; I(I > rows) = rows;
    J(J < 1) = 1; J(J > cols) = cols;

    out = zeros(new_rows, new_cols, ch);
    for k = 1:ch
        for i = 1:new_rows
            for j = 1:new_cols
                ii = I(i,j); jj = J(i,j);
                out(i,j,k) = img(ii, jj, k);
            end
        end
    end
end

```

## 2.2 Realce de imagens no domínio espacial (da imagem)

### 2.2.1 Arquivo principal

```
%% Realce de imagens no domínio espacial (da imagem)
```

## 2.3 Filtragem espacial

### 2.3.1 Arquivos principais

#### 2.3.1.1 Ruídos Salt & Pepper e Gaussian



Figura 1: Imagens originais

### Adição de Ruídos

Para iniciar esta etapa, deve-se inserir diferentes níveis de ruído sal e pimenta (salt and pepper) e ruído Gaussiano simultaneamente nas imagens selecionadas: “raposa.jpg” e “borboleta.jpg”. Para isso, utiliza-se a função imnoise, responsável por contaminar a imagem com ambos os tipos de ruídos.

```

img1 = imread('borboleta.jpg');
% Adiciona ruído sal e pimenta
img_noisy1 = imnoise(img1, 'salt & pepper', 0.05); % 5% de ruído
% Adiciona ruído gaussiano
img_noisy1 = imnoise(img_noisy1, 'gaussian', 0, 0.01); % média 0, var
    0.01

img2 = imread('raposa.jpg');
% Adiciona ruído sal e pimenta
img_noisy2 = imnoise(img2, 'salt & pepper', 0.05); % 5% de ruído
% Adiciona ruído gaussiano
img_noisy2 = imnoise(img_noisy2, 'gaussian', 0, 0.01); % média 0, var
    0.01

if usejava('desktop')
    imshow(img_noisy1);
    imshow(img_noisy2);
else
    mkdir('.out');
    imwrite(img_noisy1, '.out/borboleta_ruido.png');
    imwrite(img_noisy2, '.out/raposa_ruido.png');
end

```

Como resultado, temos as imagens originais contaminadas com ruído Gaussiano de variância 0.01 e com ruído salt & pepper de densidade 0.05.



Figura 2: Imagens com ruído

### Filtragem Alpha Trimmed Mean

Em seguida, as imagens passam pelo filtro Alpha Trimmed Mean. Para isso, foi desenvolvida a função `alpha_trimmed_mean_filter 3x3`, que aplica o filtro aos três canais de cor: vermelho, verde e azul,

e, posteriormente, os resultados são combinados novamente para gerar a imagem final com os ruídos atenuados.

```

function imgFiltrada = alpha_trimmed_mean_filter(imgRuidosa, windowSize,
    trimAmount)
% imgRuidosa = imagem colorida de entrada
% windowSize = tamanho da janela
% trimAmount = número de pixels a remover das extremidades após ordenar

    % Inicializa imagem de saída
    imgFiltrada = zeros(size(imgRuidosa));

    % Processa cada canal RGB separadamente
    for ch = 1:3
        % Extrai canal e converte para double
        canal = double(imgRuidosa(:,:,ch));
        [rows, cols] = size(canal);
        outputCanal = zeros(rows, cols);

        % Limites da vizinhança
        maxOffset = ceil(windowSize / 2);
        minOffset = floor(windowSize / 2);

        % Varre a imagem
        for r = maxOffset:(rows - minOffset)
            for c = maxOffset:(cols - minOffset)
                % Extrai janela local
                localRegion = canal(r - minOffset:r + minOffset, c -
                    minOffset:c + minOffset);

                % Coloca em vetor, ordena e remove extremos
                values = sort(localRegion(:));
                values = values(trimAmount + 1 : windowSize * windowSize
                    - trimAmount);

                % Calcula média dos valores restantes
                outputCanal(r, c) = mean(values);
            end
        end

        % Mantém bordas originais
        outputCanal(1:maxOffset-1, :) = canal(1:maxOffset-1, :);
        outputCanal(rows-maxOffset+2:end, :) = canal(rows-maxOffset+2:end
            , :);
        outputCanal(:, 1:maxOffset-1) = canal(:, 1:maxOffset-1);
        outputCanal(:, cols-maxOffset+2:end) = canal(:, cols-maxOffset+2:
            end);

        % Atribui canal processado à imagem de saída
        imgFiltrada(:,:,ch) = outputCanal;
    end

    % Converte para uint8
    imgFiltrada = uint8(imgFiltrada);
end

```



Figura 3: Imagens filtradas

#### Medições PSNR e SNR

A avaliação da qualidade da filtragem é feita utilizando os parâmetros SNR (Signal-to-Noise Ratio) e PSNR (Peak Signal-to-Noise Ratio), por meio de suas respectivas funções. Como esses indicadores medem a relação entre o sinal original e o ruído, valores mais altos de SNR ou PSNR correspondem a menor presença de ruído, indicando melhor qualidade da imagem.

```



```

	Borboleta	Raposa
Filtrada PSNR	26.16 dB	24.87 dB
Filtrada SNR	19.84 dB	19.14 dB
Ruidosa PSNR	16.26 dB	16.58 dB
Ruidosa SNR	10.25 dB	11.14 dB

### 2.3.1.2 Unsharp Masking

Ao aplicar este filtro, os detalhes da imagem original são realçados sem a introdução de ruído. Inicialmente, a imagem é suavizada por meio de convoluções Gaussianas, que funcionam como filtros passa-baixa, removendo componentes de alta frequência, como texturas finas e bordas. Em seguida, a imagem suavizada é subtraída da original, e os detalhes resultantes são ampliados através da multiplicação por um fator de ganho “amount”. Por fim, esse resultado é somado à imagem original, proporcionando o realce final de detalhes e contornos.

OBS.: 2 parâmetros no código: Sigma: Define o grau de suavização na máscara Gaussiana; Fator de ganho(amount): Define quanto os detalhes serão amplificados;

```

% Leitura da imagem original (colorida)
img = imread('raposa.jpg');
img = im2double(img); % converte para double

% Parâmetros do filtro Gaussiano
sigma = 0.5; % desvio padrão da Gaussiana (controla suavização)
amount = 2.5; % fator de realce

% Criar imagem suavizada
h = fspecial('gaussian', [5 5], sigma); % filtro Gaussiano 5x5
img.blur = imfilter(img, h, 'replicate');

% Unsharp masking
img.sharp = img + amount*(img - img.blur);

% Garantir que os valores fiquem entre 0 e 1
img.sharp = max(min(img.sharp, 1), 0);

% Mostrar resultados
figure;
subplot(1,2,1); imshow(img); title('Original');
subplot(1,2,2); imshow(img.sharp); title('Realce com Unsharp Masking');

```

## 2.4 Dithering

### 2.4.1 Arquivo principal

```
%% Dithering
```

## 2.5 Operações Geométricas

### 2.5.1 Arquivo principal

```
%% Operações Geométricas
```