

Relatório 2 - Codificação de Informação Multimédia

CIM 2022/2023

Pedro Silva | Tiago Ribeiro

Class: 1MEEC_T02

Introdução

O presente relatório foi proposto no âmbito da unidade curricular de "Codificação de Informação Multimédia" e tem como objetivo aplicar e consolidar os conhecimentos obtidos sobre representação de sinais visuais no domínio digital. Serve também para familiarização com a técnica de processamento de imagens utilizando a ferramenta MATLAB - a utilização de diversos espaços de cor, o filtro de imagens e convolução, bem como a deteção de cantos e contornos.

Exercício 1

Inicialmente, utilizaram-se 3 imagens obtidas em formato *bitmap*, correspondente ao espaço de cor **RGB** (*Red, Green, Blue*), separando a imagem original nos seus componentes constitutivos, que podem ser visualizados na figura 1.

Posteriormente, codificou-se a conversão das imagens deste espaço de cor para o espaço de cor **HSV** (*Hue, Saturation, Value*), separando cada nos seus componentes constitutivos, como é possível verificar na figura 2.

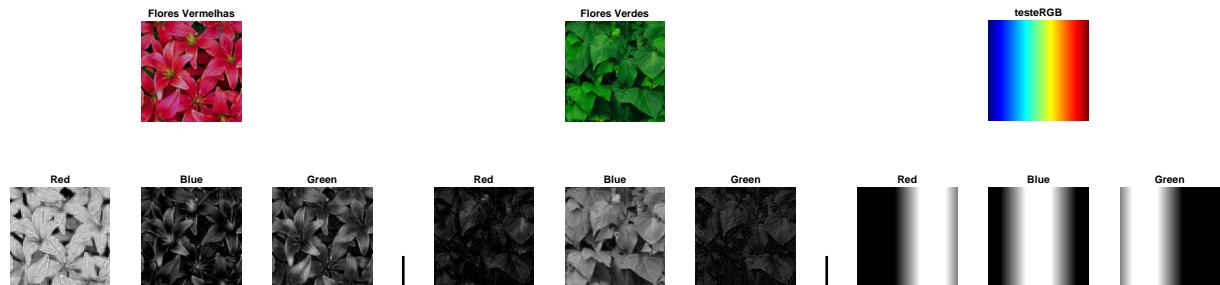


Figura 1: Imagens em formato bitmap e separação nos seus componentes RGB

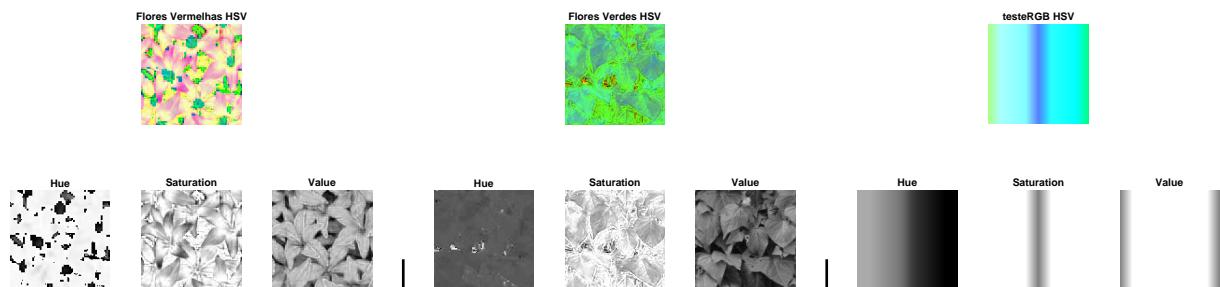


Figura 2: Imagens convertidas para HSV e separação nos seus componentes

Como é possível comprovar pelas imagens estudadas e acima referidas, o espaço de cor RGB segue um modelo de cor aditivo, em que uma dada cor é obtida pela adição das 3 componentes fundamentais. Por outro lado, o espaço de cor HSV segue um modelo cilíndrico, onde uma cor é obtida pelas suas componentes fundamentais, onde *Hue* representa factualmente a cor, *Saturation* a intensidade da cor e *Value* o brilho associado a esta.

Gerando as imagens noutra ferramenta de *software*, verificou-se também que as cores das imagens utilizando o espaço de cor RGB variam ligeiramente, enquanto que quando se utiliza o espaço de cor HSV, as cores nunca variam, o que vai de encontrar ao resultado empírico que dita que o espaço de cor RGB é dependente do dispositivo utilizado, ao contrário de HSV que é independente.

Posteriormente, converteu-se as mesmas imagens de RGB para o espaço de cor YCbCr, cujos resultados podem ser visualizados abaixo:

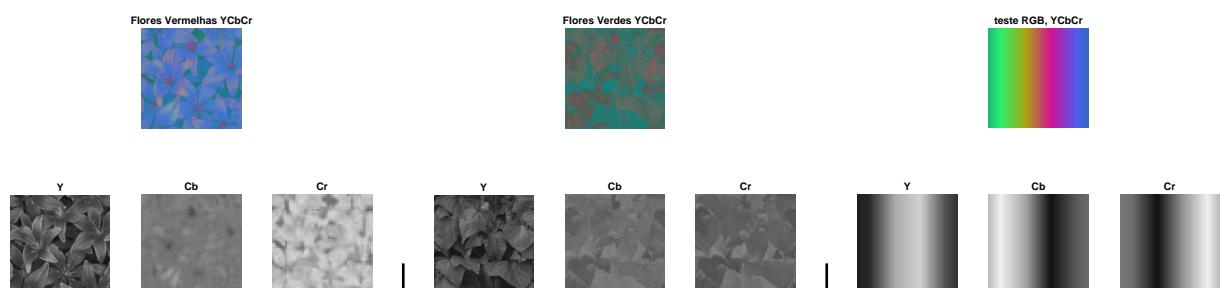


Figura 3: Imagens convertidas para YCbCr e separação nos seus componentes

Este é um espaço de cor que procura separar a informação do brilho e cor da imagem. Y refere-se à luminância e corresponde à média pesada das componentes RGB. Já o Cb e Cr referem-se à diferença entre a cor azul e vermelha face à luminância, respetivamente.

Assim, analisando as imagens com este espaço de cor, bem como os seus componentes, permite distinguir melhor a distribuição e contraste das cores em questão, bem como a separação mais clara das componentes de crominância.

Por fim, converteram-se as mesmas imagens para o espaço de cor YUV. Este espaço de cor é semelhante ao YCbCr, onde Y representa na mesma a luminância, mas as componentes U e V referem-se à diferença entre as cores azul e amarelo face a esta. Deste modo, em algumas imagens, a diferença entre estes dois espaços de cor não é elevada, existindo apenas pequenas distinções face à separação da crominância

Exercício 2

Neste exercício, utilizaram-se diferentes métodos/algoritmos de forma a proceder à ampliação ou redução de uma determinada imagem, que para o contexto desta atividade, pode ser gerada através da função Matlab fornecida `imzoneplate.m`, que recebe como argumento o valor N , gerando assim uma imagem com dimensões $N \times N$.

Desta forma, recorreu-se à função fornecida `ampliaReduz.m`, tendo feito algumas alterações de modo a facilitar a obtenção dos dados. Nesse sentido, gerou-se um vasto número de imagens para diferentes valores de N , diferentes fatores de ampliação ou redução, para além de terem sido testados outros métodos que serão abordados de seguida.

NOTA: Juntamente com a submissão deste documento, foram também fornecidas todas as imagens criadas no contexto deste exercício.

Ampliação

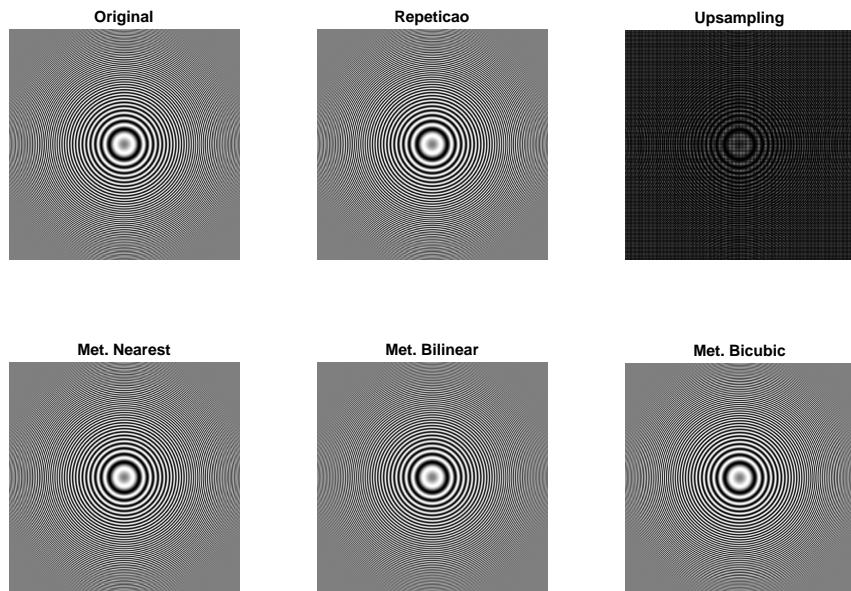


Figura 4: Ampliação de uma Imagem 500x500 com fator 2

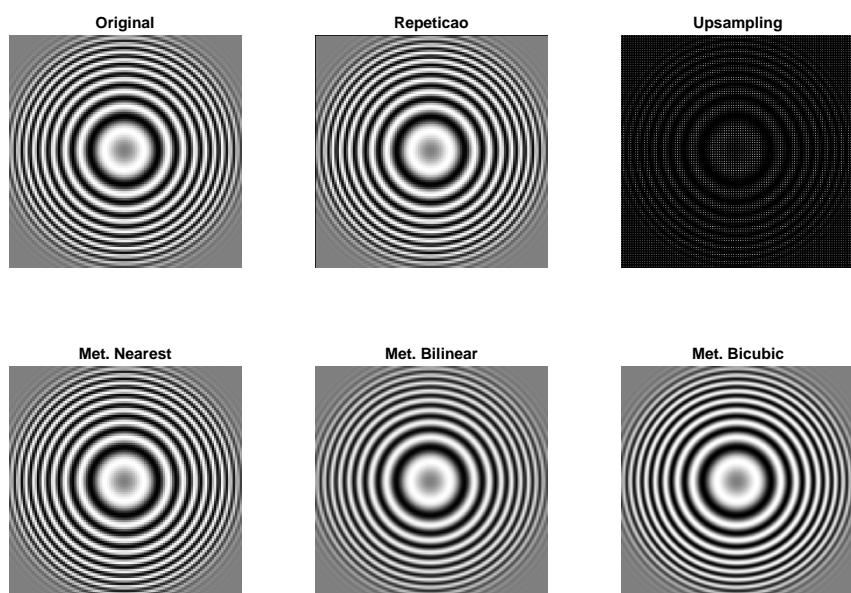


Figura 5: Ampliação de uma Imagem 100x100 com fator 3

Redução

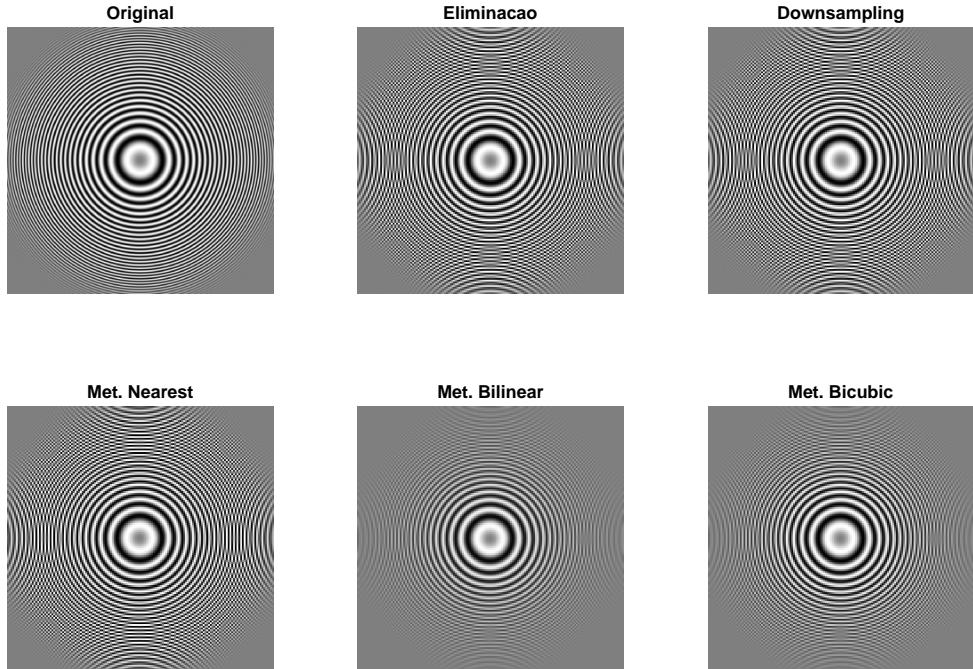


Figura 6: Redução de uma Imagem 300x300 com fator 2 (1/2)

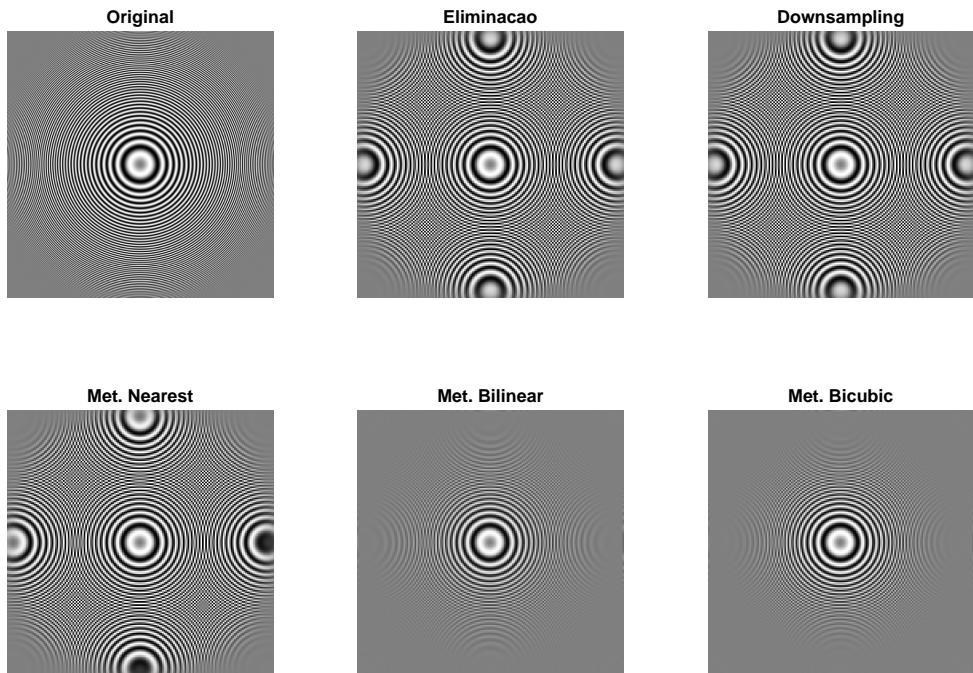


Figura 7: Redução de uma Imagem 500x500 com fator 3 (1/3)

Assim como é possível observar, foram utilizados 5 métodos diferentes:

- **Repetição - Eliminação** - Este método é o mais simples, consistindo apenas na repetição de amostras preexistentes, caso se pretenda ampliar, ou na remoção de amostras, no caso de se pretender reduzir, criando uma nova imagem de nova dimensão, porém mais pixelizada e de menor qualidade.

Para a obtenção destes resultados, utilizou-se a função fornecida em `ampliaReduz.m`;

- **Downsampling** - Este método é idêntico ao anterior de 'Eliminação', conduziu aos mesmos resultados, no entanto este é aplicado exclusivamente para a redução de imagem. Nesse sentido, difere apenas no facto de os resultados terem sido obtidos através da biblioteca já fornecida `resample.m`;

- **Upampling** - A função `upsample(img, fator)` é também fornecida pela biblioteca `resample.m`, aplicando-se apenas no aumento de imagem. Este método foi o que conduziu a piores resultados, pois no lugar de novas amostras este limita-se a atribuir-lhe o valor 0, deteriorando significativamente a qualidade da imagem.

Os seguintes métodos foram executados através da função `imresize.m`, uma função interna do Matlab.

- **Nearest** - À semelhança dos métodos já referido - **Repetição - Eliminação**, este limita-se também a repetir ou eliminar os valores dos pixels vizinhos de forma a criar uma nova imagem, pelo que conduz a resultados indistinguíveis dos anteriores;
- **Bilinear** - Este método difere das abordagens anteriores. Neste caso, um novo pixel é obtido através da média pesada dos quatro pixels vizinhos. Conduzindo a contornos mais suaves que, na maioria dos casos, se traduz em resultados mais favoráveis;
- **Bicubic** - Neste método, o novo pixel é calculado através da média pesada dos 16 mais próximos, que à semelhança do 'Bilinear', conduz a contornos suaves, porém acentuando ainda mais esse efeito, o que permite preservar mais detalhes da imagem original, com a desvantagem de ser computacionalmente o mais exigente;

Exercício 3

Primeiramente, escolheu-se alguns dos filtros pré-instalados no MATLAB. Utilizaram-se os seguintes filtros:

- *Average*;
- *Disk*;
- *Laplacian*;
- *Gaussian*;
- *Motion*.

Com recurso às funções presentes na ferramenta de *software*, nomeadamente a função **fspecial** e **imfilter**. Assim, aplicou-se cada um dos filtros às imagens escolhidas. O resultado pode ser visto abaixo:

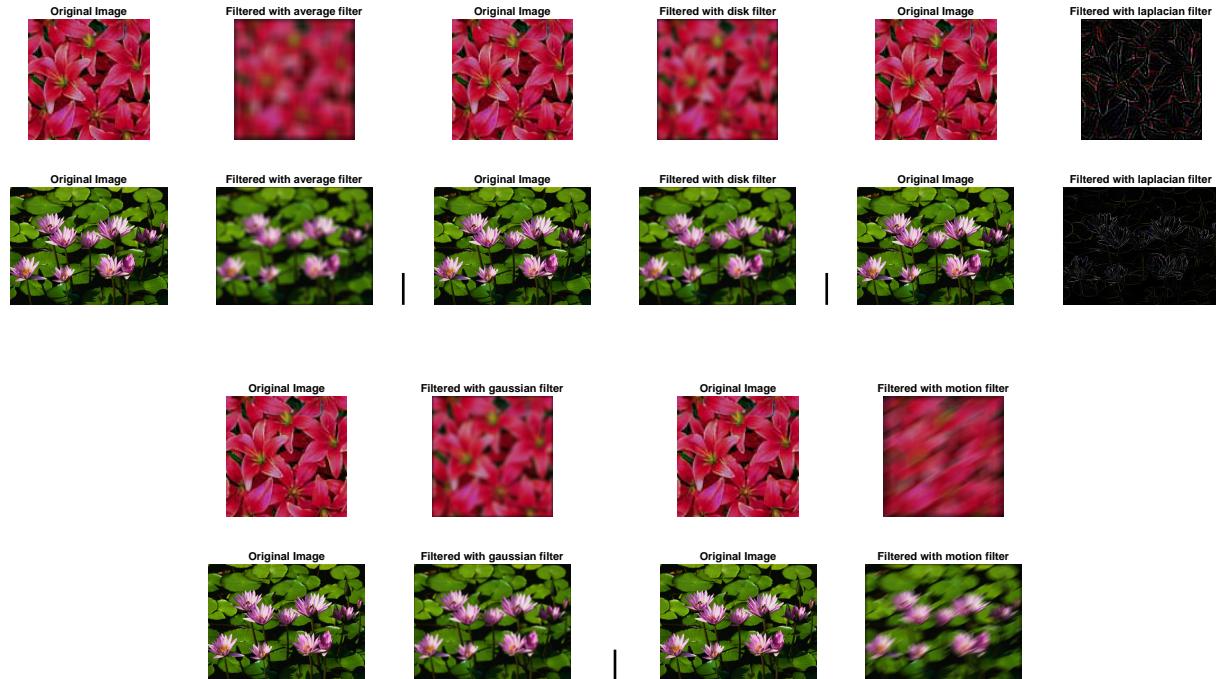


Figura 8: Imagens filtradas com diversos filtros

Como se pode verificar, constatam-se diferenças com a utilização dos vários tipos de filtros.

O filtro **Average** substitui cada pixel da imagem pela média do valor dos pixels vizinhos. Este filtro foi codificado da seguinte maneira:

```
chosen_filter = fspecial(filters, 2);
img_filtered = filter_apply(img, chosen_filter, 0);
```

O filtro **Disk** substitui cada pixel da imagem pela média ponderada do valor de cada pixel vizinho em função da distância. Este filtro foi codificado da seguinte maneira:

```
chosen_filter = fspecial(filters, 3);
img_filtered = filter_apply(img, chosen_filter, 0);
```

O filtro **Laplacian** salienta as elevadas frequências das imagens, nomeadamente os cantos e contornos. Este filtro foi codificado da seguinte maneira:

```
chosen_filter = fspecial(filters, 0.5);
img_filtered = filter_apply(img, chosen_filter, 0);
```

O filtro **Gaussian** substitui cada pixel da imagem pela média ponderada do valor de cada pixel vizinho em função da Função Gaussiana. Este filtro foi codificado da seguinte maneira:

```
chosen_filter = fspecial(filters, [5 5], 2);
img_filtered = filter_apply(img, chosen_filter, 0);
```

O filtro **Motion** substitui cada pixel da imagem pela média ponderada do valor de cada pixel vizinho em função de uma direção que simula movimento. Este filtro foi codificado da seguinte maneira:

```

chosen_filter = fspecial(filters, 20, 30);
img_filtered = filter_apply(img, chosen_filter, 0);

```

Todos as imagens filtradas permitem comprovar o comportamento empírico esperado com a utilização de cada filtro.

Posteriormente a esta análise, utilizou-se na mesma os filtros **Average** e **Gaussian**, mas com outros valores intrínsecos, tal que:

```

filters2 = {"average", "gaussian"};
chosen_filter = fspecial(filters, 10); %average filter
chosen_filter = fspecial(filters, [10 10], 2); % gaussian filter

applied1 = filter_apply(im1, chosen_filter, 0);
applied2 = filter_apply(im2, chosen_filter, 0);

```

Os resultados podem ser visualizados abaixo:

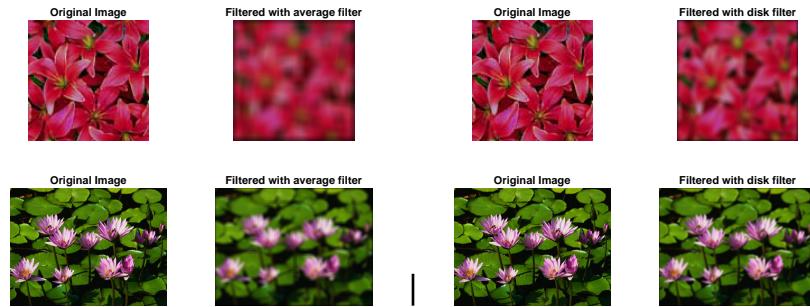


Figura 9: Imagens filtradas com diferentes parâmetros

Comparando os resultados obtidos com os acima citados, verifica-se que, aumentando o tamanho do *kernel* no filtro **Average**, a imagem fica mais distorcida, dado que se está a aumentar o número de vizinhos que são usados para calcular a média do pixel final. Já no filtro **Gaussian**, aumentou-se o tamanho do *kernel*, mas manteve-se o desvio padrão, o que resulta numa imagem mais suave mas também mais distorcida.

De seguida, pautou-se por aplicar um **Mean Filter** e um **Median Filter** a duas imagens em *grayscale*, com objetivo de reduzir o ruído presente nestas. No primeiro filtro, escolheram-se dois tamanhos para o *kernel*, uma matriz 3×3 e uma matriz 5×5 .

Na primeira, um pixel é substituído pela média dos vizinhos presentes numa matriz 3×3 com o pixel original no centro dessa matriz. Na segunda, um pixel é substituído pela média dos vizinhos presentes numa matriz 5×5 com o pixel original no centro dessa matriz.

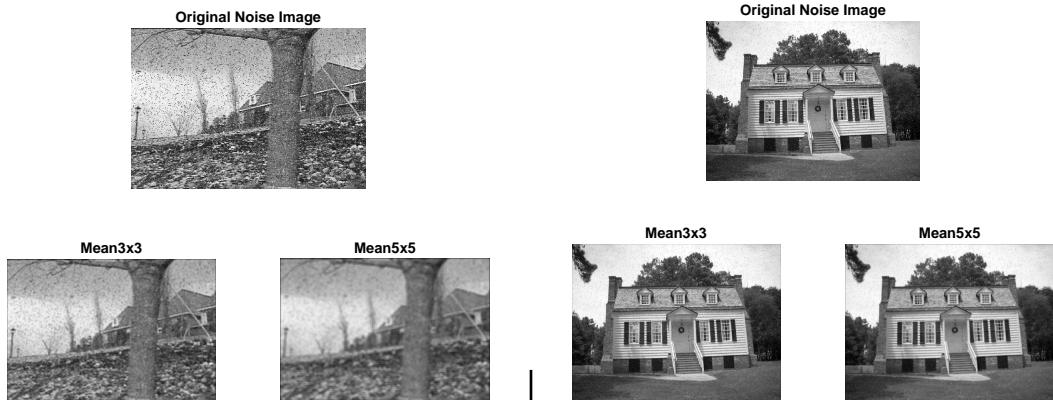


Figura 10: Imagens filtradas com *Mean Filter*

Como é possível verificar pelas imagens supracitadas, ambos os tamanhos definidos para o *kernel* permitem reduzir o ruído. No caso do filtro aplicado com matriz 3×3 , verifica-se uma menor atenuação do ruído, mas menos suavização e distorção da imagem. Com um filtro 5×5 , a atenuação do ruído é superior (no céu e telhado p.ex), mas a suavização e distorção é superior, dado que cada pixel está a ser substituído pela média de um maior número de vizinhos.

Já no caso de se aplicar um **Median Filter**, cujos resultados podem ser vistos abaixo, verifica-se uma maior capacidade de remoção de ruído, nomeadamente o ruído *salt and pepper*, sem incorrer numa suavização e distorção exagerada da imagem. Isto porque este filtro tem a capacidade de preservar os cantos e *features* importantes.

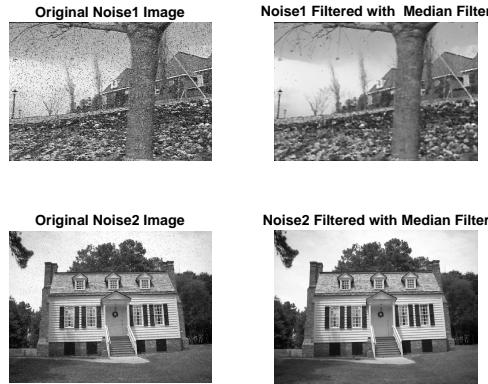


Figura 11: Imagens filtradas com *Median Filter*

Sobel

Posteriormente, aplicou-se os operadores **Sobel** e **Prewitt** a 3 imagens distintas.

Para o operador **Sobel**, foram fornecidas as seguintes matrizes:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (1)$$

Os resultados da sua aplicação nas imagens pretendidas são os seguintes:

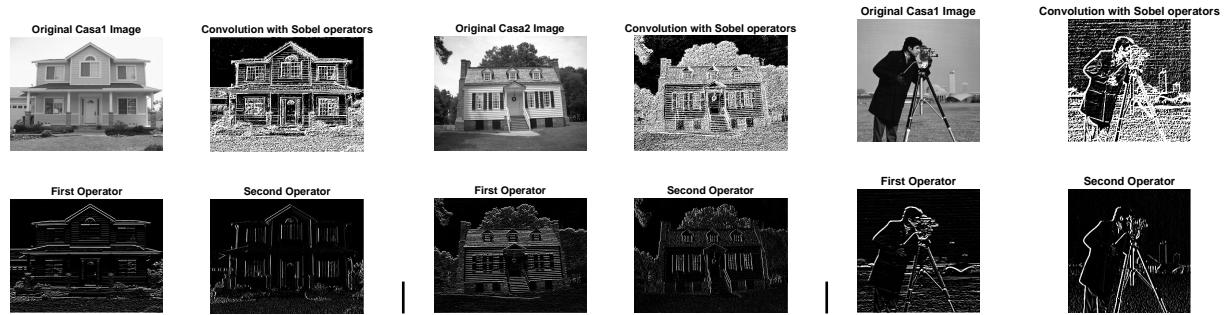


Figura 12: Imagens filtradas com operador Sobel

Como é possível verificar, a primeira matriz do operador permite delimitar os contornos horizontais da imagem. Já a segunda matriz do operador delimita os contornos verticais.

A conjugação das duas matrizes permite assim definir com precisão os contornos das imagens, ou seja, as suas *features* mais importantes.

Prewitt

No operador **Prewitt**, são fornecidas as seguintes matrizes:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad (2)$$

Por oposição ao primeiro operador, a primeira matriz delimita os contornos verticais das imagens e a segunda matriz delimita os contornos horizontais.

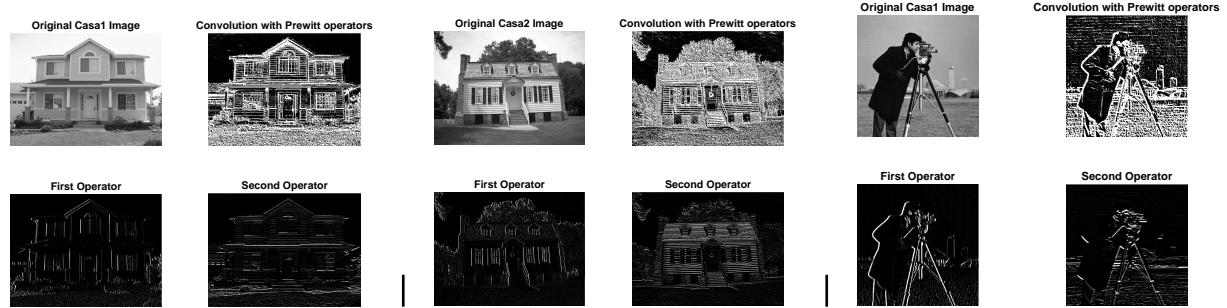


Figura 13: Imagens filtradas com operador Prewitt

Do mesmo modo que no operador anterior, a conjugação dos contornos permite também retirar as *features* principais das imagens.

Existe, no entanto, diferenças entre os dois operadores que podem ser constatadas pela visualização das imagens.

Verifica-se experimentalmente que o operador **Sobel** é mais sensível aos cantos e contornos principais das imagens, detetando-as com maior precisão que o operador **Prewitt**. Não obstante, é mais sensível há existência de ruído que este último.

Em conclusão, a escolha entre estes operadores deverá ser feita com base na precisão das *features* que se pretende obter com estes e se nas imagens sobre as quais se aplicam os operadores está presente elevado ruído.

Canny Detector

Por fim, fez-se uma experiência com **Canny's Contour Detector**, utilizando a função `edge` do Matlab para diferentes valores de `sigma` e `threshold` onde se obtiveram os seguintes resultados:

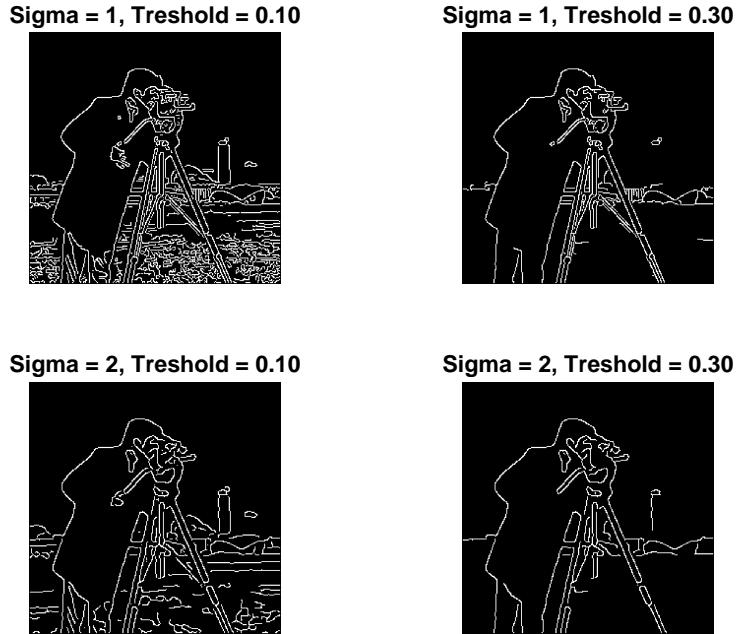


Figura 14: Resultados obtidos para diferentes valores de Sigma e Threshold

Analisando a figura 14 é possível observar como estes parâmetros afetam de diferente forma a imagem.

Relativamente ao valor de **Sigma**, este é responsável por suavizar a imagem, pelo que valores elevados de sigma levam à deteção de menos contornos. Este fenómeno é particularmente evidente comparando imagens onde o único parâmetro alterado é o de **Sigma**, tal como é possível observar comparando as duas imagens da esquerda ou as duas da direita.

No caso do **Threshold**, ao aumentar o seu valor diminui-se a sensibilidade do algoritmo à deteção de contornos, o que se pode confirmar comparando a primeira imagem com a segunda, bem como a terceira com a última.

Deste modo, ao aplicar **Canny's Contour Detector**, a escolha dos valores de **Sigma** e **Threshold** é dependente dos requisitos da aplicação desejada. Caso se pretenda detetar um maior número de contornos, estes parâmetros devem comportar valores mais baixos, acontecendo o oposto caso se pretenda filtrar detalhes mais ténues.

Conclusão

Esta atividade laboratorial tinha como objetivo introduzir o processamento de imagem em ambiente MATLAB, tendo sido abordados os diferentes espaços de cor, as diferentes técnicas de ampliação e redução de imagem, bem como a utilização de filtros de imagem.

Com as experiências realizadas, conclui-se que a escolha adequada do espaçado de cor, do método de ampliação ou redução, bem como do tipo de filtro a utilizar têm um grande impacto no resultado final, o que é fundamental para aplicações em visão computacional e processamento de imagem, pelo que é crucial estar familiarizado com estas ferramentas.

Em suma, pode-se aferir que se concluiu este trabalho com sucesso, tendo sido realizadas todas as atividades propostas.