

Multimédia e Novos Serviços

Lab 2

João Seixas - up201505648

José Pedro Borges - up201503603

Vicente Espinha - up201503764

**Projeto realizado no âmbito do
Mestrado Integrado em Engenharia Informática e
Computação**

13/03/2018

Índice

Índice	2
1ª Parte	2
Conversão de RGB para HSV	2
Conversão de RGB para YCbCr	6
Conversão de RGB para YUV	9
2ª Parte	10
REDUÇÃO DE IMAGEM	10
AMPLIAÇÃO DE IMAGEM	11
3ª Parte	12
3.1 “filtro2019.m”	12
3.2 “filtroMediana.m”	16

1ª Parte

Nesta primeira parte deste trabalho, é pedido que se use scripts Matlab, que ajudem a analisar ficheiros bitmap nos diferentes espaços de cor (modelos que representam sinais visuais de formas pré-definidas). Nesta experiência, utilizaram-se os seguintes espaços de cor : RGB, HSV, YCbCr e YUV.

Conversão de RGB para HSV

Nesta parte é pedido um script que:

1. importe uma imagem com o formato bitmap (espaço de cores RGB) e apresente essa imagem no écran;
2. separe cada componente RGB numa matriz diferente e apresente no écran cada uma delas ;
3. converta essa imagem para o espaço de cores HSV e apresente essa imagem no écran;
4. separe cada componente HSV numa matriz diferente e apresente no écran cada uma delas

O script criado foi o seguinte:

```
function resultado = RGB2HSV (filename);

% ii) importe uma imagem com o formato bitmap (espaço de cores RGB) e
apresente essa
% imagem no écran;

im = imread (filename);

if size(im,3) ~= 3
    im = cat(3,im,im,im);
end
figure(1);imshow(im);title('imagem original');

fprintf('\n Prima uma tecla para continuar\n'); pause

% iii) separe cada componente RGB numa matriz diferente e apresente no
écran cada uma
% delas
    r = im(:, :, 1);
    g = im(:, :, 2);
```

```

    b = im(:, :, 3);
    figure(2);imshow(r);title('imagem original (R)');
    figure(3);imshow(g);title('imagem original (G)');
    figure(4);imshow(b);title('imagem original (B)');

fprintf('\n Prima uma tecla para continuar\n'); pause

% iv) converta essa imagem para o espaço de cores HSV e apresente essa
imagem no
% écran;

im_hsv = rgb2hsv(im);
figure(5);imshow(im_hsv);title('imagem HSV');

fprintf('\n Prima uma tecla para continuar\n'); pause

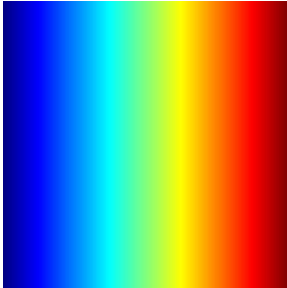
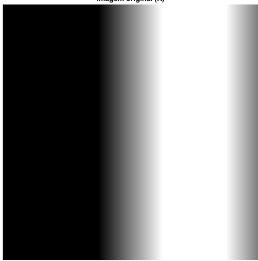
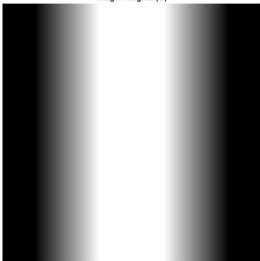
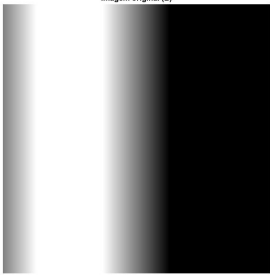












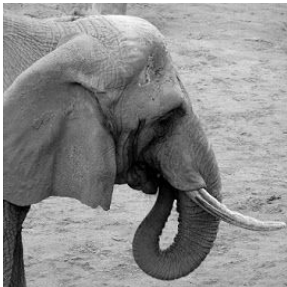
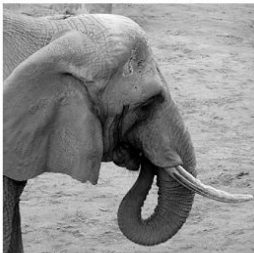
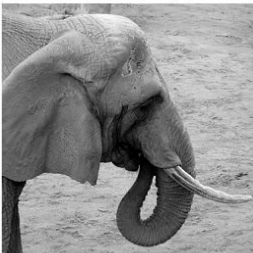

% v) separe cada componente HSV numa matriz diferente e apresente no écran
cada uma
% delas
h = im_hsv(:, :, 1);
s = im_hsv(:, :, 2);
v = im_hsv(:, :, 3);

    figure(6);imshow(h);title('imagem HSV (H)');
    figure(7);imshow(s);title('imagem HSV (S)');
    figure(8);imshow(v);title('imagem HSV (V)');

resultado = 0;
end

```

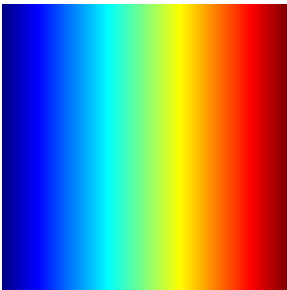
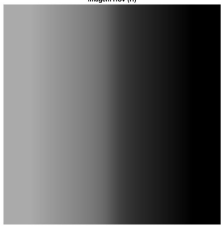

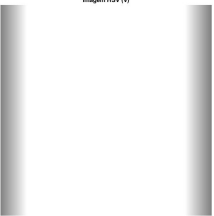
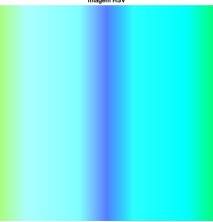

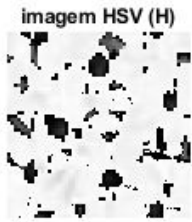


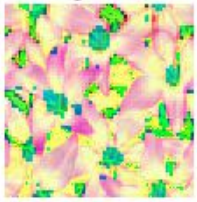
Na primeira parte é lida a imagem através da instrução *imread()* e a imagem é mostrada com *imshow()*. Também é testado se a imagem está no formato grayscale (imagem na qual o valor de cada pixel é uma única amostra de um espaço de cores), convertendo-a para RGB, sendo os valores de todas as componentes iguais. Seguidamente, estão os resultados observados, em relação às suas componentes RGB.




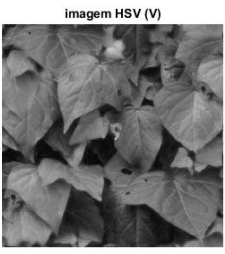
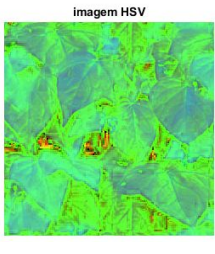



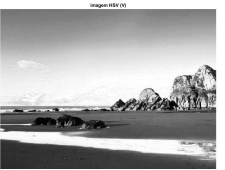
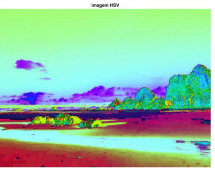
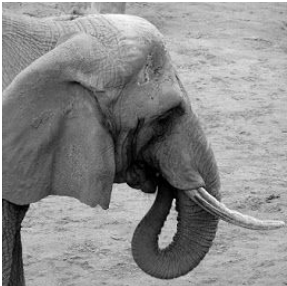
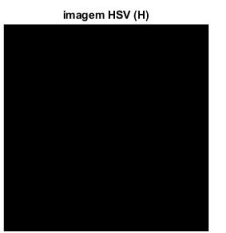
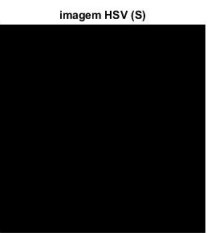
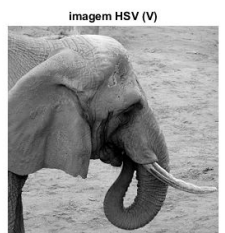
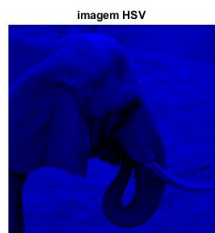
Original	R	G	B
	<small>imagem original (R)</small> 	<small>imagem original (G)</small> 	<small>imagem original (B)</small> 
	<small>imagem original (R)</small> 	<small>imagem original (G)</small> 	<small>imagem original (B)</small> 
	<small>imagem original (R)</small> 	<small>imagem original (G)</small> 	<small>imagem original (B)</small> 
	<small>imagem original (R)</small> 	<small>imagem original (G)</small> 	<small>imagem original (B)</small> 
	<small>imagem original (R)</small> 	<small>imagem original (G)</small> 	<small>imagem original (B)</small> 

Neste formato, são usados 3 bytes por cada pixel, sendo que cada um deles corresponde a uma cor: vermelho (Red), verde (Green) ou azul (Blue). Todas as cores são obtidas através da mistura destas 3 cores, variando as suas intensidades. Por exemplo, quando as três cores têm o valor mínimo (zero), correspondem à cor preta. Contrariamente, a cor branca resulta da combinação de valores máximos (255) nas três componentes.

Nestes resultados, pode verificar-se que, tal como esperado, as cores predominantes nas imagens originais são mais claras na componente correspondente. Isto acontece, pois as componentes têm valores mais elevados, e, quanto mais intenso, mais claro aparecerá na imagem da sua componente. Por exemplo, na imagem das folhas verdes, observa-se que a imagem correspondente à componente verde (Green) é a mais clara em comparação com as outras duas componentes, podendo assim verificar-se a predominância de cor verde na imagem original. Em relação à imagem do elefante, a imagem original está em grayscale, ou seja todas as componentes têm o mesmo valor.

De seguida, a imagem é convertida para o formato HSV e os resultados observados são os seguintes:

Original	H	S	V	HSV
				
				

	 imagem HSV (H)	 imagem HSV (S)	 imagem HSV (V)	 imagem HSV
	 imagem HSV (H)	 imagem HSV (S)	 imagem HSV (V)	 imagem HSV
	 imagem HSV (H)	 imagem HSV (S)	 imagem HSV (V)	 imagem HSV

O formato HSV está consideravelmente mais próximo da forma como o sistema visual humano percebe e descreve as cores. O espaço de cores é definido através dos seguintes parâmetros:

- Hue (tonalidade) define a cor assumindo valores em $[0^\circ ; 360^\circ]$
- Saturation oferece uma medida de grau de pureza da cor, ou do grau pelo qual a cor está poluída de branco, assumindo valores em $[0 ; 100\%]$
- Value indica o brilho da cor e varia entre $[0 ; 100\%]$

Neste formato, a informação de cada um dos parâmetros referidos acima está contida, por ordem, num byte.

Em todas as imagens, em relação à tonalidade(Hue), observa-se que a cor azul tem um valor alto (ângulo elevado), enquanto a cor vermelha tem um valor mais baixo (ângulo menor), com excepção da imagem das flores vermelhas, em que a sua componente de Hue é predominante clara, concluindo que a cor das flores não é um vermelho puro, pois mistura-se com cores de um valor mais elevado (rosa).

Os valores de Saturação e Brilho(Value) estão de acordo com o que era expectável, sendo que, na saturação, os valores revelam se a cor é mais ou menos pura, por exemplo, o céu da praia. Em relação ao brilho, a imagem é mais brilhante onde as imagens são mais claras, como se pode verificar na primeira imagem, em que se vê as cores mais e menos brilhantes. No que diz respeito à imagem do elefante, o valor

da sua tonalidade(H) é 0, visto que, como os valores de *vermelho*, *verde* e *azul* são iguais, o $MAX=MIN$. A sua Saturação também será 0, pois é o resultado da seguinte fórmula $(MAX - MIN)/MAX$. Tendo em conta, ao valor do Brilho(V) será igual ao MAX, ou seja, será igual à imagem original.

Conversão de RGB para YCbCr

Nesta parte, foi desenvolvido um script semelhante ao anterior, mas em vez de converter para HSV, converta para YCbCr. O código desenvolvido foi o seguinte:

```
function resultado = RGB2YCbCr (filename);

% ii) importe uma imagem com o formato bitmap (espaço de cores RGB) e
apresente essa
% imagem no écran;

im = imread (filename);

if size(im,3) ~= 3
    im = cat(3,im,im,im);
end
figure(1);imshow(im);title('imagem original');

fprintf('\n Prima uma tecla para continuar\n'); pause

% iii) separe cada componente RGB numa matriz diferente e apresente no
écran cada uma
% delas

    r = im(:, :, 1);
    g = im(:, :, 2);
    b = im(:, :, 3);
    figure(2);imshow(r);title('imagem original (R)');
    figure(3);imshow(g);title('imagem original (G)');
    figure(4);imshow(b);title('imagem original (B)');

fprintf('\n Prima uma tecla para continuar\n'); pause

% iv) converta essa imagem para o espaço de cores HSV e apresente essa
imagem no
```



```

% écran;

im_ycbcr = rgb2ycbcr(im);
figure(5);imshow(im_ycbcr);title('imagem YCbCr');

fprintf('\n Prima uma tecla para continuar\n'); pause

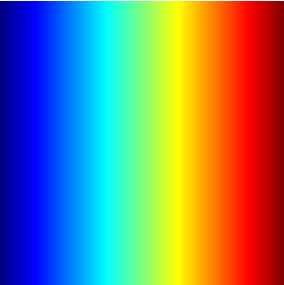
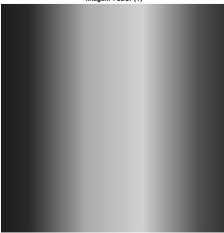

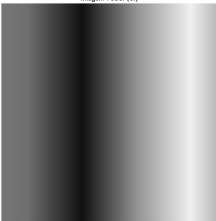
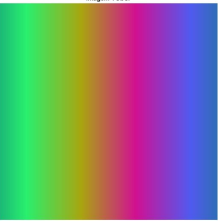

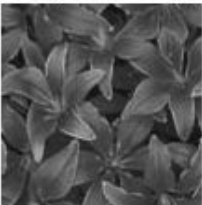
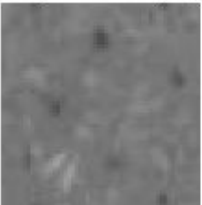
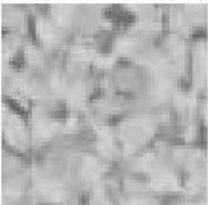

% v) separe cada componente HSV numa matriz diferente e apresente no écran
cada uma
% delas
y = im_ycbcr(:, :, 1);
cb = im_ycbcr(:, :, 2);
cr = im_ycbcr(:, :, 3);

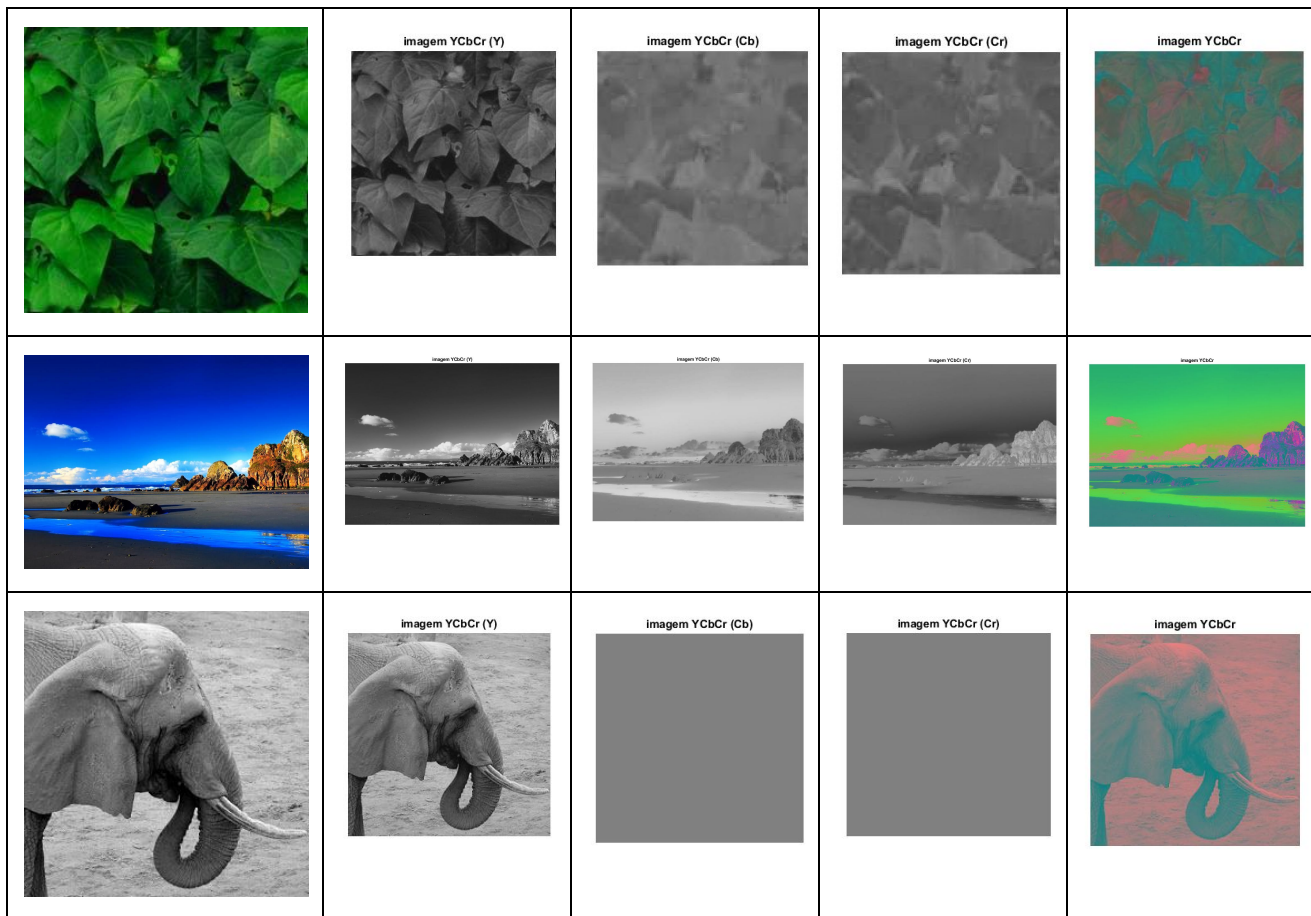
figure(6);imshow(y);title('imagem YCbCr (Y)');
figure(7);imshow(cb);title('imagem YCbCr (Cb)');
figure(8);imshow(cr);title('imagem YCbCr (Cr)');

resultado = 0;
end

```

Os resultados observados são os seguintes:

Original	Y	Cb	Cr	YCbCr
				
				



No modelo YCbCr, Y representa a luminância e as duas outras componentes correspondem às crominâncias. Estas componentes são calculadas através das seguintes fórmulas:

$$\begin{aligned}
 Y &= 0,2989 R + 0,58766 G + 0,1145 B \\
 Cb &= -0,1687 R - 0,3312 G + 0,500 B \\
 Cr &= 0,500 R - 0,4183 G - 0,0816 B
 \end{aligned}$$

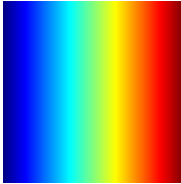
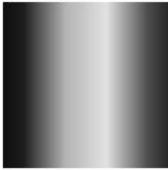
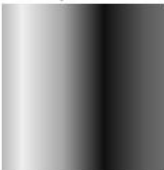


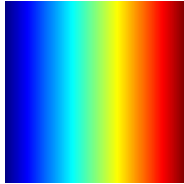








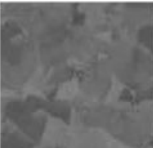









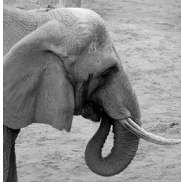




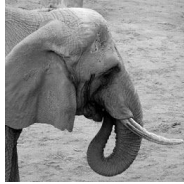
Este formato separa a *luminância* dos valores das crominâncias, o que pode levar a que estas duas componentes (Cb e Cr) possam ser sub-amostradas e comprimidas de forma independente para aumentar a eficiência do sistema, sem grandes perdas de qualidade, em relação ao olho humano.

Nos resultados obtidos, pode-se verificar que as imagens correspondentes à luminância são mais nítidas do que as de crominância. Também se observa que, por exemplo, na imagem das flores vermelhas, a componente de Cr é mais clara, e, na praia, a componente Cb, também é mais clara na zona do mar e céu. Por outro lado, na imagem das folhas verdes as componentes de crominância possuem valores baixos.

Conversão de RGB para YUV

Na terceira parte, foi utilizado um script fornecido no moodle, que converte uma imagem de formato RGB para o formato YUV, e, posteriormente, reverte para o formato inicial.

Os resultados observados foram os seguintes:

Original	Y	U	V	YUV	Nova Imagem RGB
	componente Y 	componente U 	componente V 	imagem YUV 	
	componente Y 	componente U 	componente V 	imagem YUV 	
	componente Y 	componente U 	componente V 	imagem YUV 	
	componente Y 	componente U 	componente V 	imagem YUV 	
	componente Y 	componente U 	componente V 	imagem YUV 	

O espaço de cor YUV funciona de uma forma semelhante ao YCbCr, mas utiliza coeficientes diferentes para a distribuição de cores (componentes U e V) e os mesmos para a luminância(Y).

Observando as imagens, verifica-se que YUV guarda menos informação nas componentes de cromaticidade, visível nas imagens da coluna “YUV”, mas, todavia, não se confirma que haja uma perda de informação quando reverte para o formato RGB, tendo ficado semelhantes às originais.

2ª Parte

Para esta experiência é pedido aos alunos que utilizem o script “*ampliaReduz.m*” para ampliar e reduzir imagens utilizando diferentes algoritmos de interpolação. Este script utiliza uma imagem de teste “zone plate” criada pelo script “*imzoneplate.m*”. Os diferentes algoritmos de interpolação são os seguintes:

1. **Nearest Neighbor** : Cada pixel da imagem redimensionada terá o valor do pixel mais próximo correspondente na imagem original.
2. **Bilinear** : Cada pixel da imagem redimensionada terá o valor de uma média ponderada de 4 pixels que rodeiam o pixel na imagem original.
3. **Bicúbica** : Semelhante ao algoritmo bilinear, só que com uma média ponderada de 16 pixels em redor do pixel da imagem original.

Redução de Imagem

Foram utilizados os seguintes argumentos para a função *ampliaReduz*:

```
>> ampliaReduz(400,0.5,1);  
>> ampliaReduz(400,0.5,2);  
>> ampliaReduz(400,0.5,3);  
>> ampliaReduz(600,0.5,1);  
>> ampliaReduz(600,0.5,2);  
>> ampliaReduz(600,0.5,3);
```

Com o primeiro conjunto de argumentos, é possível observar que a imagem original, ao ser reduzida, é corrompida. Surge um efeito de aliasing em redor do centro de imagem, tornando-a assim bastante diferente da original. Uma imagem “zone plate” tem frequência zero no centro e frequência máxima nos cantos, sendo que a frequência aumenta linearmente a partir do centro. O efeito de aliasing surge porque a regra de Nyquist não é respeitada. Assim, dever-se-ia previamente efetuar uma filtragem das frequências que fossem mais baixas que metade da taxa de amostragem. Este efeito é menos observável quando utilizadas as proporções de 600x600 na imagem original.

Quando é utilizado o algoritmo de interpolação Bilinear, o efeito de aliasing praticamente desaparece. Isto porque o algoritmo permite um redimensionamento mais suave da imagem original, fazendo assim com que haja menos discrepância entre imagens. Utilizando o algoritmo de interpolação Bicúbica não há grandes alterações face ao Bilinear. A imagem redimensionada fica ainda mais suave, não se notando diferença entre imagem original e atual.

Ampliação de Imagem

Foram utilizados os seguintes argumentos para a função *ampliaReduz*:

```
>> ampliaReduz(100,3,1);  
>> ampliaReduz(100,3,2);  
>> ampliaReduz(100,3,3);
```

As diferenças entre as imagens originais e as redimensionadas são menos óbvias quando se trata de ampliação. O algoritmo de nearest neighbor é o que redimensiona com pior qualidade. Este contém alguns “quadrados” na imagem, que fazem com que pareça menos suave. Os algoritmos de interpolação bilinear e bicúbica permitem um redimensionamento mais preciso da imagem, com contornos leves e sem pixelização desnecessária.

Em qualquer um destes algoritmos o efeito de aliasing é muito pouco observável e praticamente inexistente.

3ª Parte

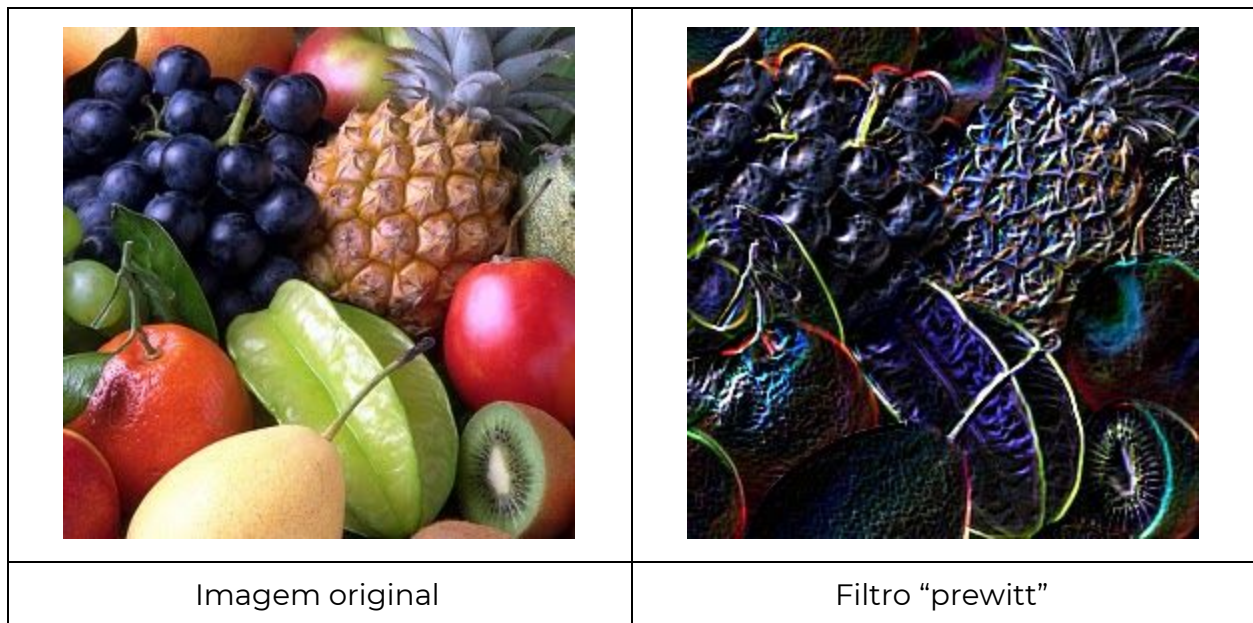
3.1 “filtro2019.m”


O seguinte código foi adicionado de forma a guardar a imagem resultante do filtro aplicado:

```
name=input(' nome para guardar?');  
imwrite(Ifiltrada,name);
```

Os filtros **Prewitt**, **Sobel** e **LoG** (*Laplacian of Gaussian*) servem para detectar os contornos da imagem. Quer o Sobel, quer o Prewitt, apenas dão ênfase aos contornos verticais ou horizontais, sendo preciso processar o filtro para cada uma das direções e depois juntar as duas.

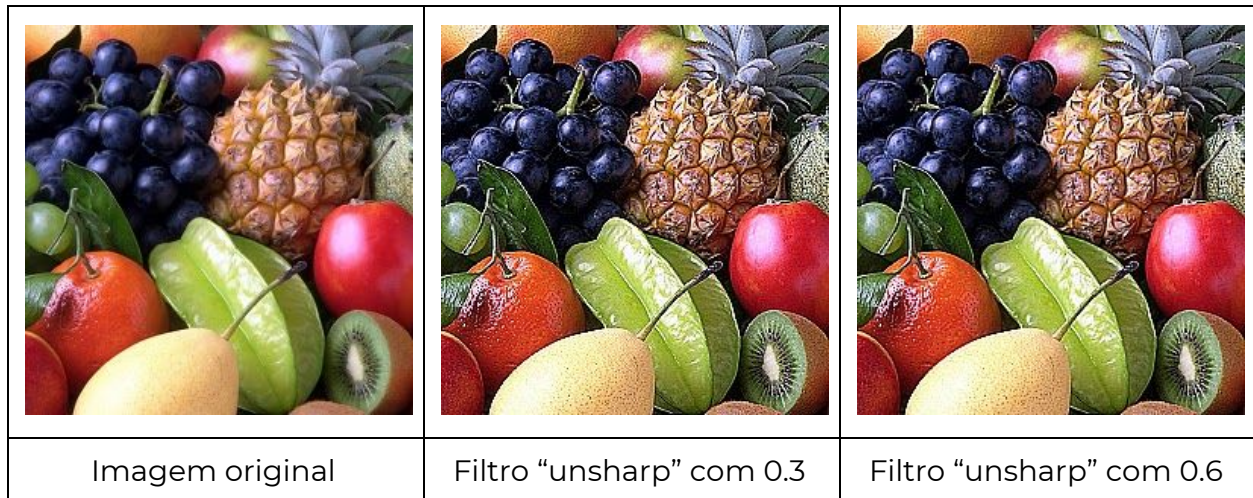
O filtro Prewitt não sofre suavização, enquanto o Sobel e o LoG sofrem suavização e este último ainda sofre suavização conforme a dimensão passada.



	
<p>Imagem original</p>	<p>Filtro "sobel"</p>

		
<p>Imagem original</p>	<p>Filtro "log" (Dimensão 3)</p>	<p>Filtro "log" (Dimensão 10)</p>

O filtro **Unsharp** é um filtro que realça a nitidez dos contornos na imagem, e verifica-se isso mesmo nos resultados obtidos.




O filtro **Motion** aplica um filtro que "simula" movimento, notando-se uma imagem mais tremida. Quanto maior a dimensão, mais tremida será a imagem. Neste exemplo apenas se variou a dimensão do movimento, e não o ângulo, representando assim um movimento horizontal.



O filtro **Disk** aplica um filtro de média circular à imagem.

		
Imagem original	Filtro "disk" (Dimensão 3)	Filtro "disk" (Dimensão 10)

O filtro **Average** aplica um filtro de média dos pixels adjacentes, de forma a reduzir as frequências altas das imagens, e como tal, aumentando a dimensão, aumenta o desfoque da imagem, notando-se especialmente com dimensão 6 e 10.

	
Imagem original	Filtro "average" (Dimensão 3)

	
Filtro “average” (Dimensão 6)	Filtro “average” (Dimensão 10)

O filtro **Gaussian** é também um filtro de média, usando o peso dos pixels adjacentes com a curva de Gauss. Reduz as frequências altas da imagem, mas sem a tornar muito desfocada comparativamente ao filtro Average.

	
Imagem original	Filtro “gaussian” (Dimensão 3)

	
Filtro “gaussian” (Dimensão 6)	Filtro “gaussian” (Dimensão 10)

3.2 “filtroMediana.m”

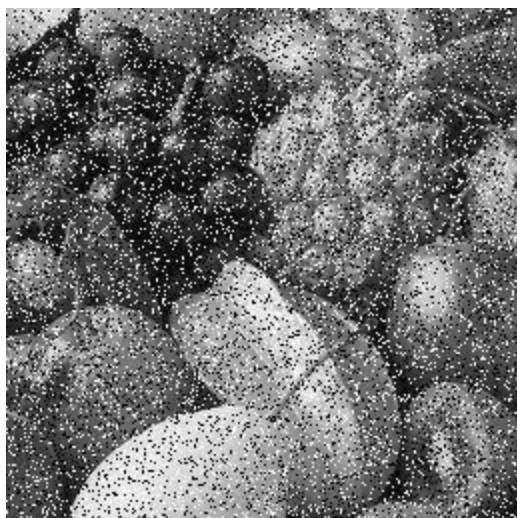
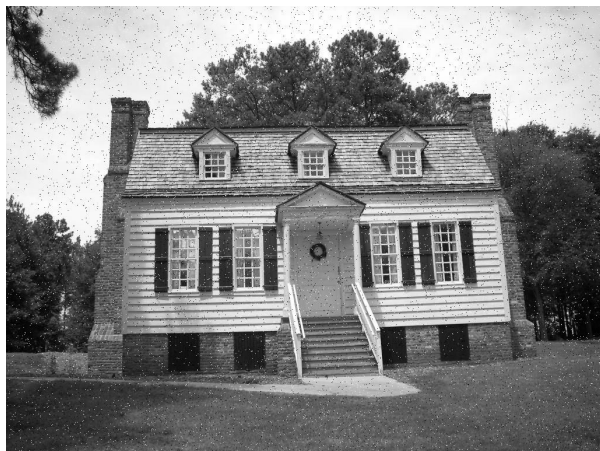
O seguinte código foi adicionado de forma a guardar a imagem resultante da aplicação de ruído à imagem (na situação em que a imagem não tenha ruído):

```
imwrite(IR, strcat(strcat('with_noise_', num2str(r)), strcat('_', filename))));
```







O seguinte código foi adicionado de forma a guardar a imagem resultante da remoção do ruído:

```
imwrite(Idenoised, strcat('noise_fixed_', filename));
```

Usando o filtro mediana podemos ver que o ruído das imagens desaparece quase por completo, sendo os resultados visíveis nas seguintes imagens:



Usando o filtro **Gaussian** nota-se uma redução de ruído, mas apresenta um desfoque ligeiro, enquanto no filtro **Average**, o desfoque é muito mais acentuado de forma a remover o ruído.

		
Imagem com ruído	Gaussian 5	Gaussian 10
		
Imagem com ruído	Average 5	Average 10