

# Relatório de MNSE : Lab 2

Elaborado por Nuno Jorge Dias Carneiro Martins / up201405079

- [1. Espaços de cor](#)
  - [1.1. Conversão de RGB para HSV](#)
  - [1.2. Conversão de RGB para YCbCr](#)
  - [1.3 Conversão de RGB para YUV](#)
- [2. Variação das dimensões espaciais de imagem usando ou não filtros com imagem de teste "imzoneplate"](#)
  - [Redução de imagem](#)
  - [Ampliação de imagem](#)
- [3. Experiências de filtragem](#)
  - [Filtro motion](#)
  - [Filtros Average e Gaussian](#)
  - [Filtros Prewitt e Sobel](#)
  - [Filtro Unsharp](#)

## 1. Espaços de cor

Na primeira parte deste trabalho, é pedido que sejam usadas scripts `Matlab` que leiam ficheiros `.bmp` (bitmap), e que seja analisada a representação destes em diferentes espaços de cor.

Um espaço de cor é um modelo que representa sinais visuais de formas previamente definidas.

Os espaços de cor utilizados são RGB, HSV, YCbCr e YUV. De notar que não foram colocadas as imagens que eram imprimidas nos formatos HSV, YCbCr e YUV, já que estas eram imprimidas como se fossem imagens RGB e, por isso, não trariam nada de mais construtivo do que as suas componentes que serão mostradas à frente.

### 1.1. Conversão de RGB para HSV

Nesta parte, é pedida a criação de uma script que:

- 1 - leia um ficheiro no formato `.bmp` e mostre a imagem;
- 2 - separe a imagem nas suas componentes RGB e mostre cada uma separadamente;
- 3 - converta a imagem para o espaço de imagem HSV e apresente a imagem;
- 4 - separe a imagem nas suas componentes HSV e mostre cada uma separadamente.

O código da script criada é o seguinte:

```
function [] = ex11(imagePath);

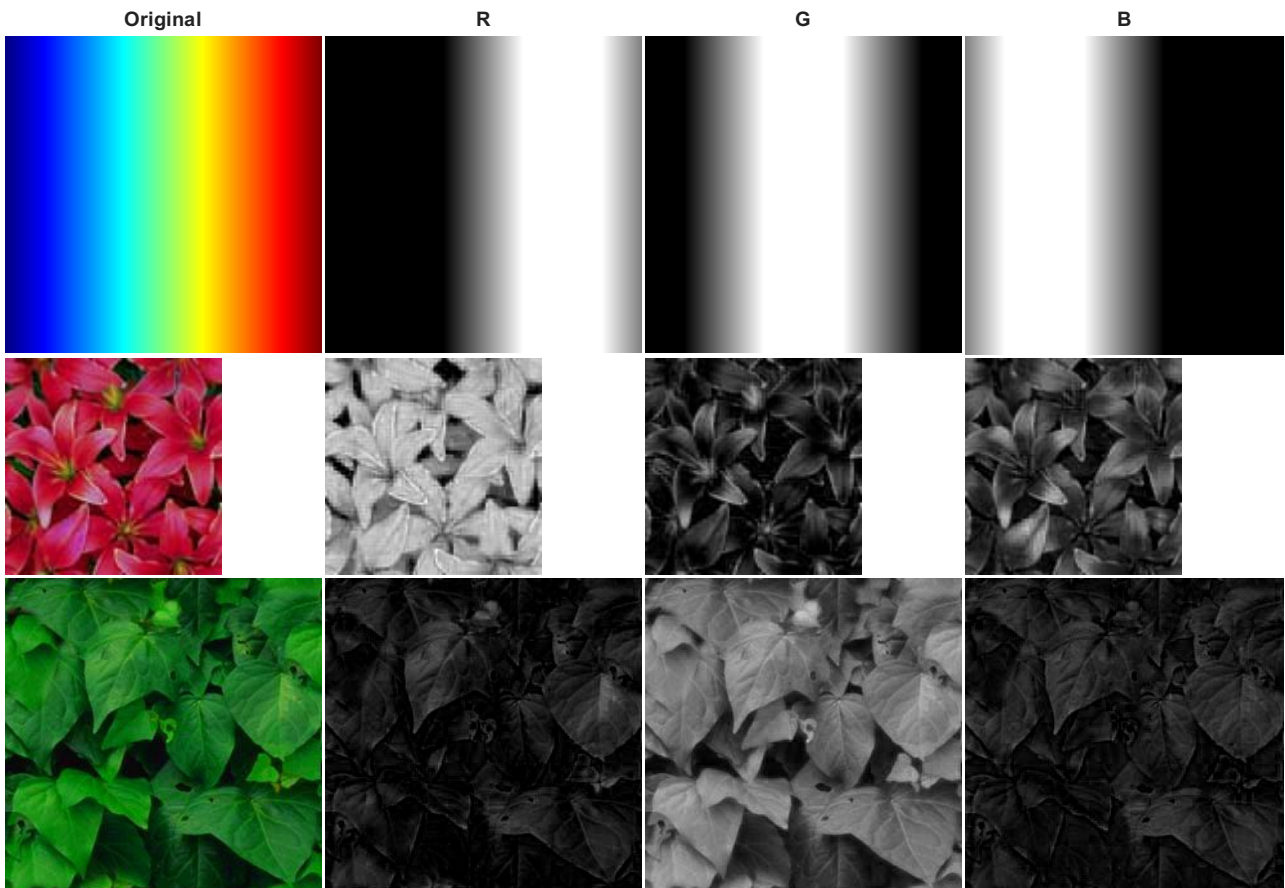
%parte 1.1 i)
image = imread(imagePath);
if size(image,3) ~= 3
    image = cat(3,image,image,image);
end
disp('Imagem:');
figure(1),imshow(image),title('imagem original');
%parte 1.1 ii)
r = image(:,:,1);
g = image(:,:,2);
b = image(:,:,3);
figure(2),imshow(r),title('red');
figure(3),imshow(g),title('green');
figure(4),imshow(b),title('blue');

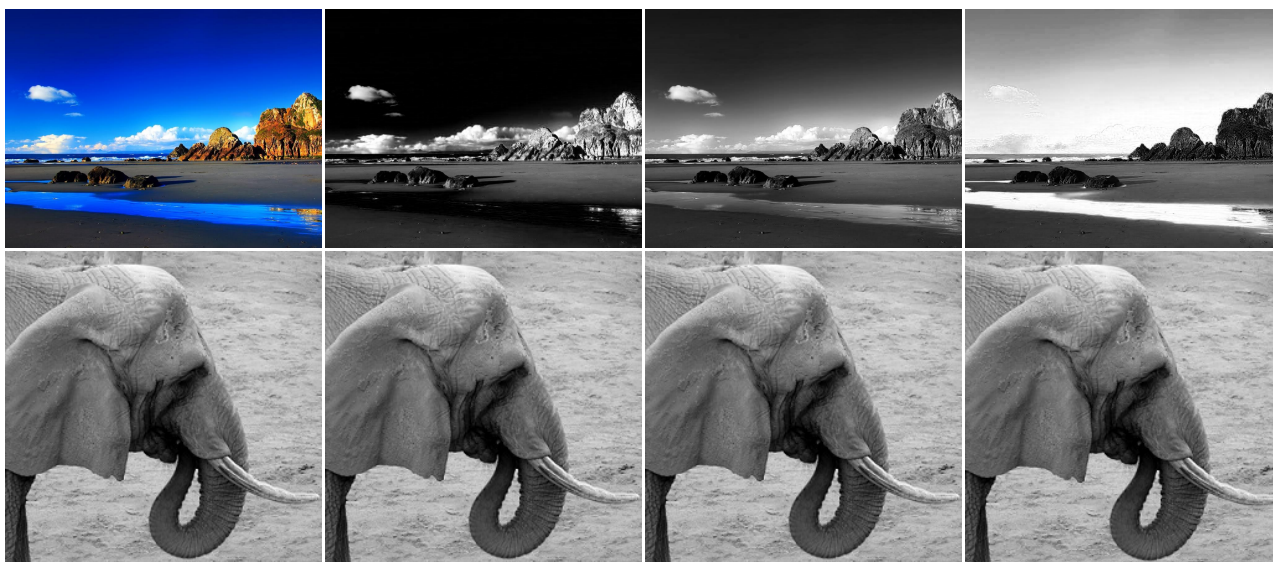
%parte 1.1 iii)
image2 = rgb2hsv(image);
disp('HSV');
figure(5),imshow(image2),title('imagem em hsv');

%parte 1.1 iv)
h = image2(:,:,1);
s = image2(:,:,2);
v = image2(:,:,3);
figure(6),imshow(h),title('hue');
figure(7),imshow(s),title('saturation');
figure(8),imshow(v),title('brightness');
```

Na primeira parte, é simplesmente lida a imagem através da utilização da função `imread()` e a imagem é mostrada com `imshow()`. Também é testado se a imagem está no formato `grayscale`, convertendo-a para `RGB` caso isto se verifique, sendo os valores de todas as componentes iguais.

De seguida, as imagens são separadas nas suas componentes RGB, sendo estes os resultados observados:

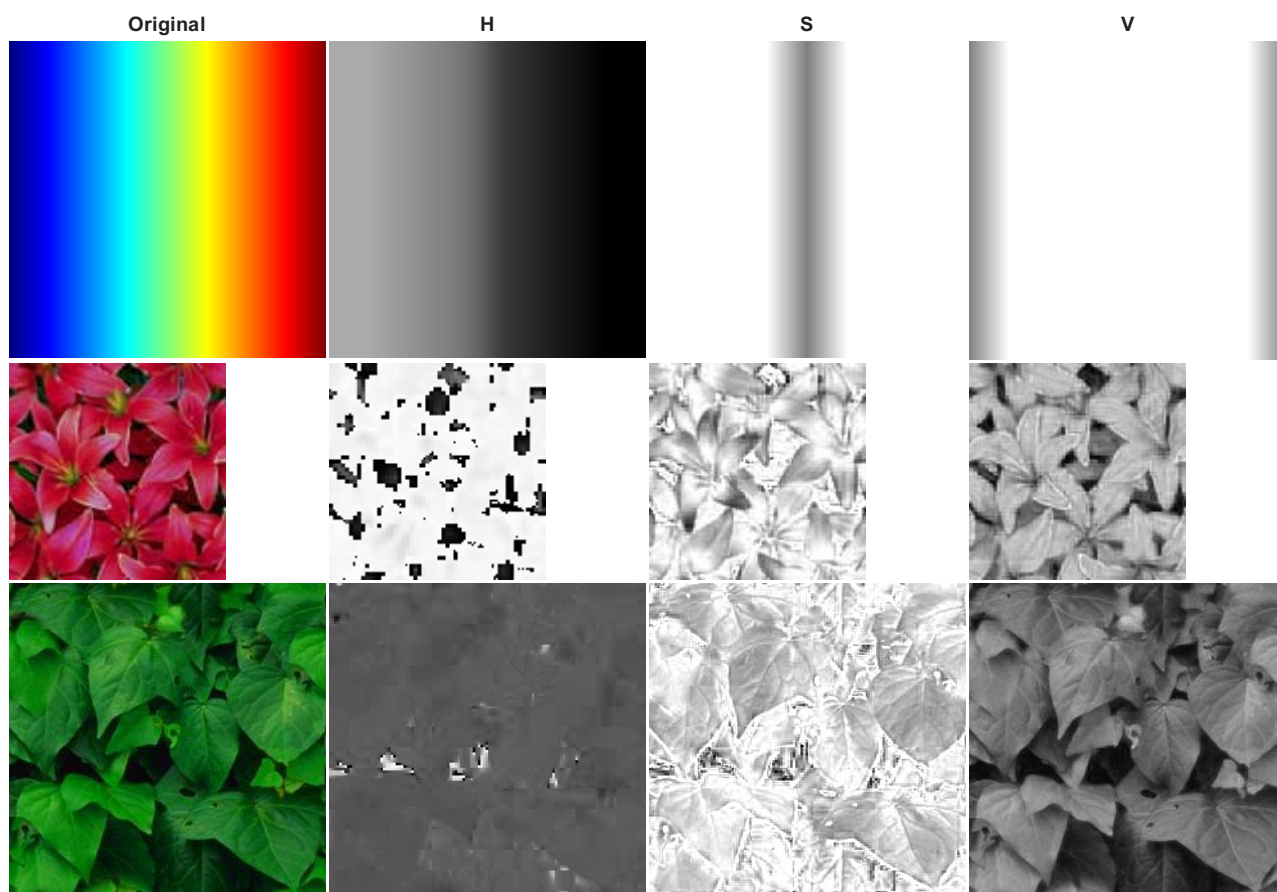




No formato RGB, são usados 3 bytes para guardar cada pixel, sendo cada byte correspondente a uma cor: Vermelho, Verde ou Azul. As cores são obtidas através da mistura destas 3 cores, em quantidades determinadas. Quando a mistura das três cores está no valor mínimo (0, 0, 0), o resultado é a cor preta. Quanto está no máximo (255, 255, 255), resulta na cor branca. A utilização de valores menores gera cores mais escuras, e valores maiores geram cores claras. Este é o formato usado por monitores para mostrar imagem.

Os resultados obtidos são os esperados. As cores mais predominantes nas imagens originais são mais claras na componente correspondente. Por exemplo, a componente vermelha das flores vermelhas é de longe a mais intensa, enquanto que a componente verde é a mais clara nas folhas verdes. Isto deve-se ao facto de estes componentes possuírem valores mais elevados, e quanto mais forte, mais claro é imprimido no ecrã. No caso do elefante, todas as componentes têm o mesmo valor, já que a imagem original está em grayscale.

Na parte seguinte, a imagem é convertida para o formato HSV, e os seus componentes são separados. Os resultados obtidos são os seguintes:





No formato **HSV**, sigla para (Hue,Saturation,Value), o primeiro **byte** representa o ângulo no círculo das cores do pixel(ou seja, a cor), o 2º **byte** representa a pureza desta cor e o 3º **byte** define o brilho do pixel. Este é um formato cuja vantagem reside na maior facilidade na manipulação de imagem.

Na primeira imagem, observa-se que **vermelho** possui um valor menor, pois tem um ângulo menor no círculo das cores, enquanto que o **azul** tem um valor alto, pelo que possui um ângulo elevado no círculo.

Este padrão verifica-se nos valores de **Hue** das outras imagens, com o caso especial das flores vermelhas, que possui um valor elevado embora a cor seja vermelha. Isto deve-se ao facto de ser um vermelho rosado, pelo que tem um ângulo ligeiramente inferior ao 0, e, por isso, um valor elevado.

Os valores de **Saturação** e **Brilho** também estão de acordo com o previsto, sendo a 1ª mais elevada em zonas cuja cor é mais ou menos pura(por exemplo, o céu da praia em contraste com a areia) e a 2ª mais brilhante onde as imagens são mais claras(como por exemplo nas folhas verdes em contraste com os pontos vazios com sombra).

Também se verifica que na imagem do elefante os valores de **Hue** e **Saturation** são 0. O facto de a saturação ser 0 deve-se ao facto do cálculo deste valor ser igual a  $(MAX-MIN) / MAX$  (sendo **MAX** o maior valor no espaço **RGB** e **MIN** o menor). Como a imagem está em escala cinza, estes valores serão sempre iguais, pelo que a saturação será sempre 0. De notar também que o valor de **Hue** é 0 pois  $MAX = MIN$ . Como o valor do **Brilho** é igual a **MAX**, a componente brilho será igual à imagem original nesta imagem, pois todas as componentes **RGB** têm um valor igual.

## 1.2. Conversão de RGB para YCbCr

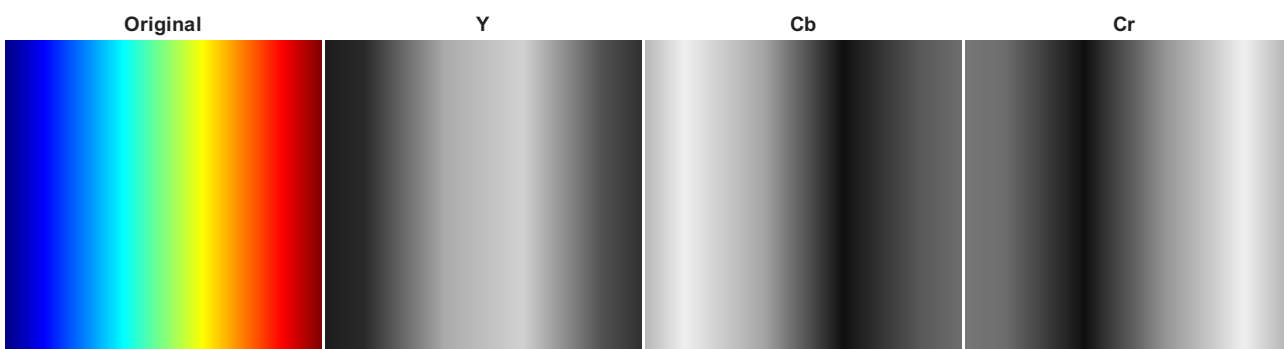
Nesta secção, foi desenvolvida uma script que converte uma imagem do formato **RGB** para uma imagem no formato **YCbCr**.

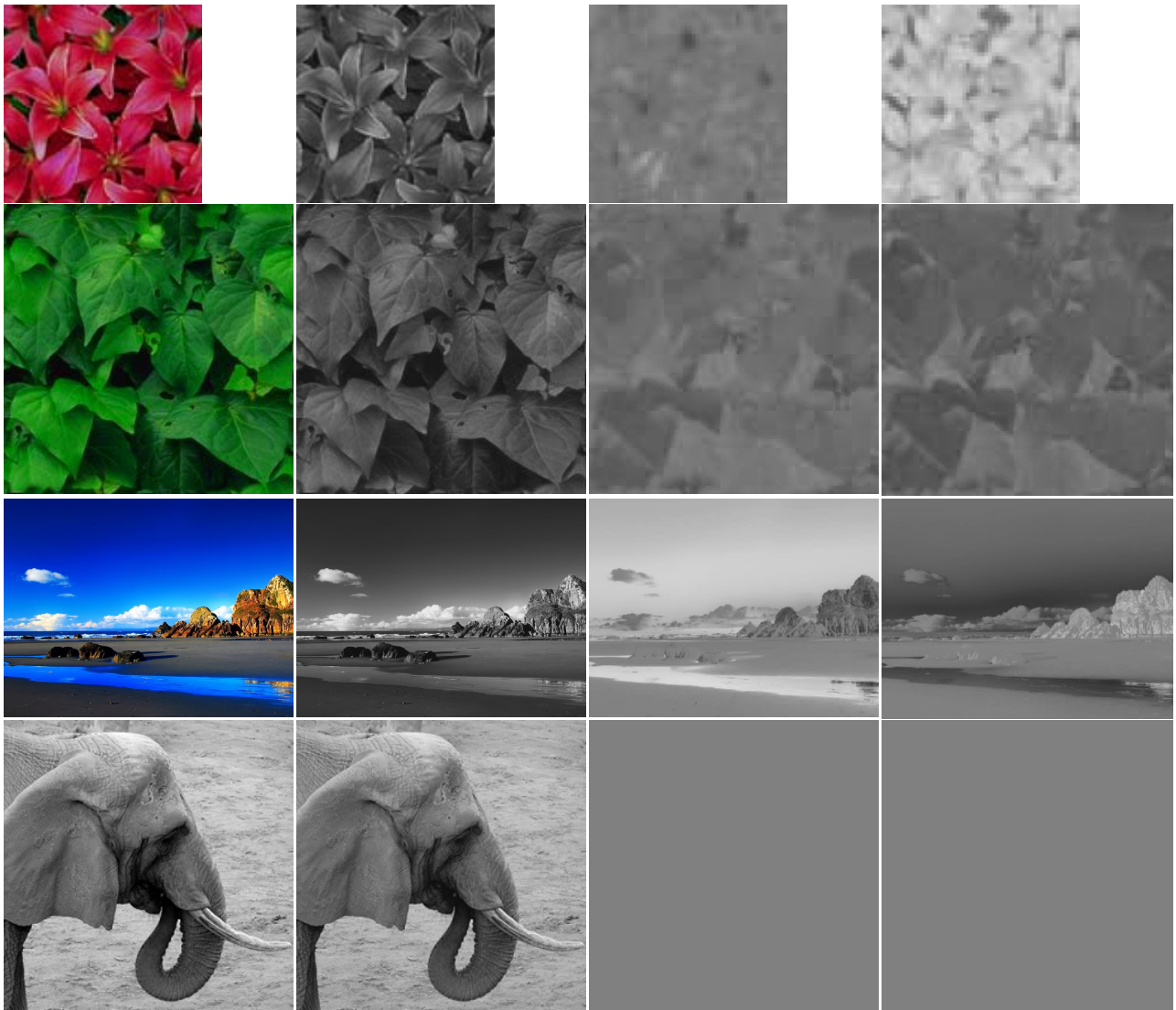
O código desenvolvido é o seguinte(trata-se de uma extensão do código desenvolvido para a parte anterior):

```
%parte 1.2
image3 = rgb2ycbcr(image);
figure(9),imshow(image3),title('imagem em ycbcr');

y = image3(:,:,1);
cb = image3(:,:,2);
cr = image3(:,:,3);
figure(10),imshow(y),title('luminance');
figure(11),imshow(cb),title('cb');
figure(12),imshow(image),title('cr');
```

O resultados obtidos foram os seguintes:





No modelo  $YCbCr$ ,  $Y$  representa a luminância. Esta é uma soma pesada das 3 componentes RGB, tendo maior intensidade na componente Verde pois os olhos humanos são mais sensíveis a esta e menos a Azul.  $Cb$  e  $Cr$  representam as crominâncias azul e vermelha, que são a subtração da luminância pelos valores de  $B$  e  $R$ . Estas componentes guardam informação menos imperativa à representação da imagem relativa principalmente às cores Azul e Vermelha.

Este formato possui a vantagem de separar a luminância, que é o fator mais importante na representação das imagens, dos valores de crominância que são menos importantes, permitindo a realização de subsampling a estas componentes sem grande perda de qualidade, pois os olhos não são tão sensíveis a estas.

Nas imagens produzidas, observa-se que as que representam a luminância possuem uma maior nitidez do que as de crominância. As primeiras são muito próximas do que seria uma imagem em greyscale na escala RGB, no entanto tem maior intensidade nas zonas verdes do que vermelhas e azuis. Por exemplo, na imagem das folhas verdes, observa-se que as crominâncias possuem um valor baixo, enquanto que a crominância vermelha apresenta um valor ligeiramente elevado para as flores vermelhas, e a crominância azul para o céu na praia.

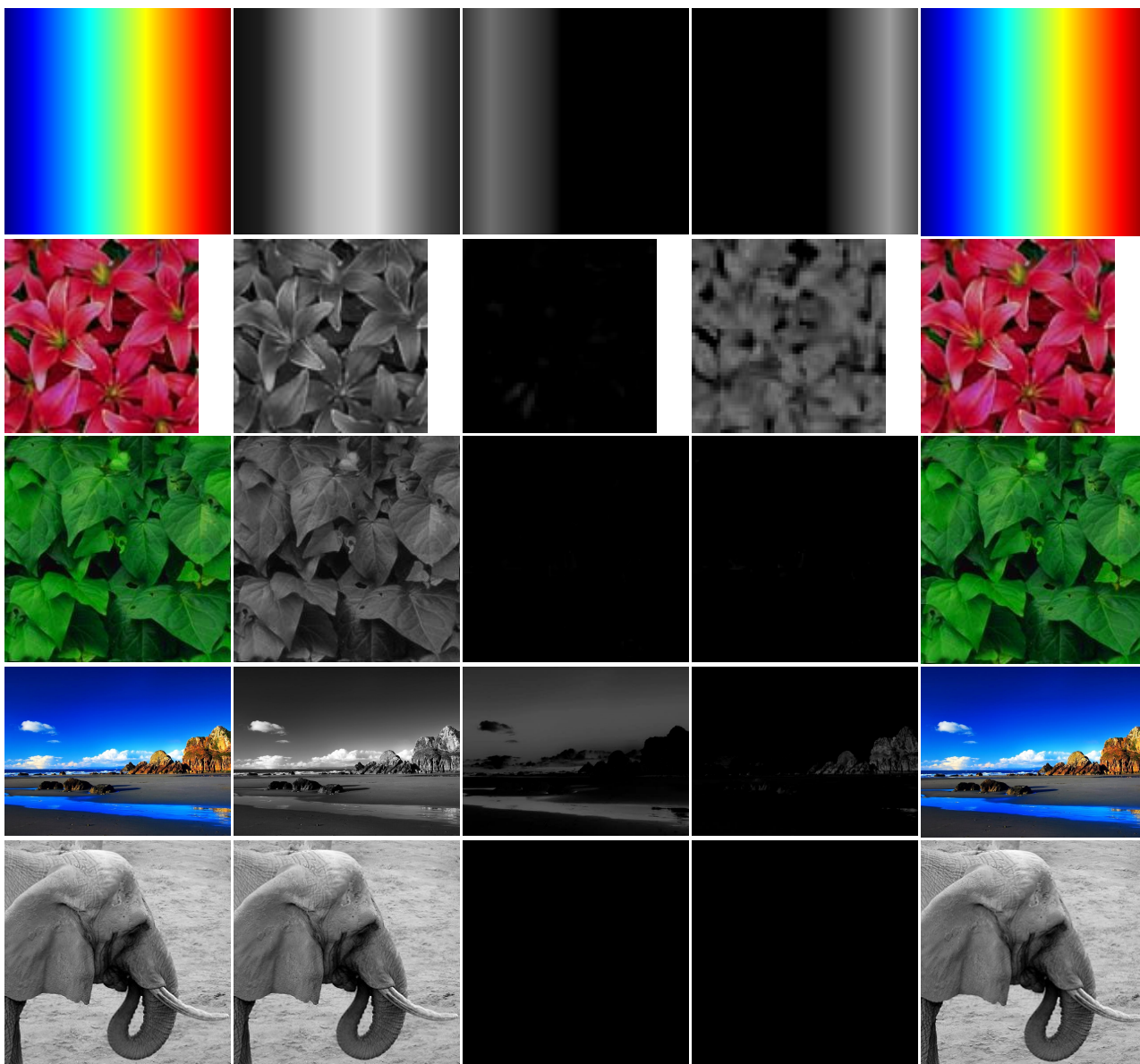
### 1.3 Conversão de RGB para YUV

Na terceira e última secção, foi utilizada uma script fornecida que converte uma imagem do formato RGB para o formato YUV, mostrando as suas componentes, e, de seguida, reverte de volta para RGB.

Os resultados obtidos foram os seguintes:

Original	Y	U	V	RGB(Reconstrução)
----------	---	---	---	-------------------





YUV é um espaço de cor que funciona da mesma forma de YCbCr, que no entanto usa coeficientes diferentes para a distribuição de cores.

Pode-se confirmar ao observar as imagens que YUV guarda menos informação nas componentes de *crominância* relativo às cores fora do azul e vermelho respetivamente, mas que no entanto não há perda de informação e que as imagens foram reconstruídas como esperado, e que os fundamentos do espaço de cor são iguais aos de YCbCr.

## 2. Variação das dimensões espaciais de imagem usando ou não filtros com imagem de teste “imzoneplate”

O objetivo desta experiência é utilizar a script `ampliaReduz.m` que, como o nome indica, pode reduzir ou ampliar uma imagem aplicando diferentes algoritmos e relatar/comentar os resultados. A imagem utilizada é uma *Zone Plate* produzida pela script `imzoneplate.m`.

Os algoritmos de interpolação usados nas experiências são os seguintes:

- *Nearest Neighbor* - neste algoritmo, a cor dos texels é igual à cor do pixel central do mesmo na imagem original.
- *Bilinear* - a cor dos texels é igual a uma média ponderada dos 4 pixels que estão à volta do central.
- *Bicubica* - funciona da mesma forma que *bilinear*, mas utiliza os 16 pixels mais próximos

Devido ao facto de no formato `.pdf` as imagens serem automaticamente suavizadas, foi impossível incluir os resultados observados.

### Redução de imagem

Os testes de redução foram realizados com:

Tamanho Factor Métodos de interpolação

300	0.4	Nearest Neighbor
300	0.4	Bilinear
300	0.4	Bicubic
600	0.4	Nearest Neighbor
600	0.4	Bilinear
600	0.4	Bicubic

Foi observado que, ao utilizar uma imagem com tamanho 300 com o algoritmo Nearest Neighbor, surgia um efeito de `aliasing` com repetição do padrão nos cantos, assim como uma grande deformação do formato da imagem. Isto deve-se principalmente à imagem ser pequena, pois com a imagem de tamanho 600, embora o efeito de `aliasing` fosse observável, não existia deformação.

Este efeito de `aliasing` deve-se à redução da imagem, pois, longe do centro da imagem, a frequência de alteração de preto para branco é igual à densidade de pixels presentes. Ao reduzir a imagem, `Nyquist` é quebrado, levando ao surgimento de um efeito de `aliasing`, que neste caso se manifesta com repetição do padrão da imagem simetricamente nos cantos.

No entanto, ao utilizar o algoritmo `Bilinear`, este efeito de `aliasing` já é vastamente menos observável. Isto deve-se ao facto de este algoritmo realizar uma média dos 4 pixels que rodeavam o pixel original, eliminando assim quase inteiramente o efeito de `aliasing`, assim como as deformações às imagens. Isto deve-se a que o algoritmo, ao realizar uma suavização de cada pixel da imagem produzida, suaviza também a frequência de alteração de preto para branco, e, consequentemente, esta aparenta a ser menor do que a imagem original, não sendo assim observável que `Nyquist` não foi respeitado.

Com o algoritmo `Bicubic`, a diferença foi muito ligeira em relação a `Bilinear`, quase não perceptível. A única diferença observável foi que a imagem ficava com a textura uniforme um pouco mais próximo do centro em `Bicubic`.

## Ampliação de imagem

Os testes de redução foram realizados com:

Tamanho	Factor	Métodos de interpolação
100	3	Nearest Neighbor
100	3	Bilinear
100	3	Bicubic

Utilizando o algoritmo `Nearest Neighbor`, foi observável que cada pixel era ampliado para 3x o seu tamanho, passando a ser cada pixel da imagem original um bloco com a mesma exata cor, mas o triplo do tamanho. Ao contrário do observado na redução da imagem, não surgiu `aliasing` e não houve deformação da imagem. Isto deve-se principalmente a que na ampliação não é quebrado `Nyquist`.

Ao utilizar o algoritmo `Bilinear`, a imagem já era mais suave em relação à anterior, não possuindo o mesmo padrão de quadrados mas uma distribuição de cores mais uniforme.

A diferença entre `Bilinear` e `Bicubic` já foi mais perceptível durante a ampliação, sendo os contornos ainda mais leves.

## 3. Experiências de filtragem

Nesta última experiência, o objetivo é aplicar diferentes filtros a imagens e relatar os efeitos observados. Os filtros a serem utilizados são os seguintes:

- `Motion` : efeito de movimento horizontal na imagem
- `Average` : cria um efeito de "blur" na imagem, realizando uma suavização desta de forma a remover frequências altas
- `Gaussian` : mesma função de filtro `Average`, mas utiliza uma curva de Gauss para calcular os pesos de cada pixel, dando assim maior influência aos pixels centrais
- `Prewitt` : realça os contornos horizontais ou verticais de uma imagem
- `Sobel` : faz o mesmo que `prewitt`, mas realiza alguma suavização
- `Unsharp` : realça todos os contornos das imagens

### Filtro motion

Para o filtro `Motion`, os resultados observados foram os seguintes:

Original

Motion(Dimensão 3)

Motion(Dimensão 10)

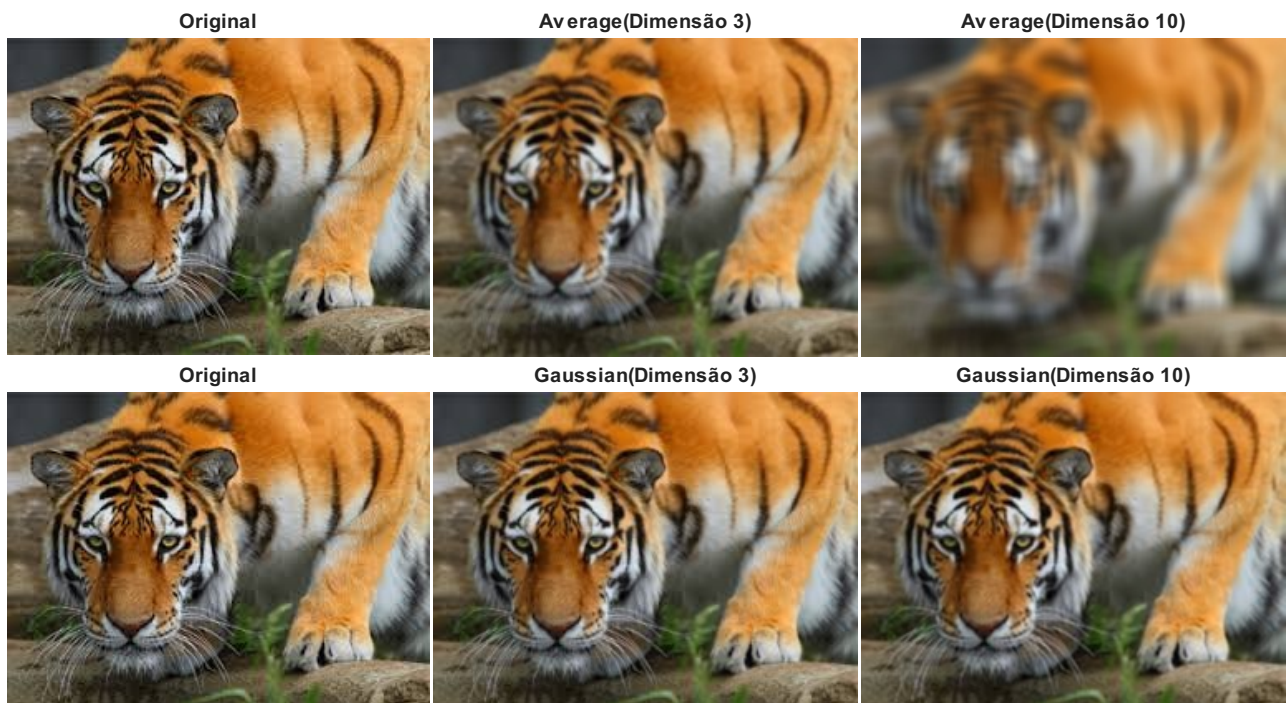


Este filtro cria uma sensação de movimento ou vibração da imagem na direção horizontal. Observa-se principalmente do resultado com dimensão 10 que parece que o tigre está em movimento.

Este efeito é conseguido ao criar uma média com dimensão N dos píxeis presentes horizontalmente, conseguindo assim um efeito de "blur" linear nessa direção. Ao a imagem ser replicada horizontalmente, cria-se este efeito devido à semelhança ao efeito de persistência nos olhos humanos que resulta de movimentos reais.

### Filtros Average e Gaussian

Com os filtros *Average* e *Gaussian*, os resultados foram os seguintes:



Estes filtros criam um efeito de "blur" na imagem, com objetivo de reduzir altas frequências. No entanto, utilizam algoritmos diferentes.

O primeiro(Average) utiliza a média dos píxeis à volta do pixel em questão para o efeito, enquanto que o segundo(Gaussian) utiliza uma curva de Gauss para calcular a influência dos píxeis que rodeiam cada pixel da imagem original.

Pode-se confirmar que os efeitos do filtro Average são excessivos, causando demasiado "blur", enquanto que Gaussian blur criou contornos muito mais suaves, sendo esta mudança mais notifiável, por exemplo, nos bigodes do Leão, e eliminando mesmo assim altas frequências.

### Filtros Prewitt e Sobel

Para os filtros *Prewitt* e *Sobel*, os resultados observados foram os seguintes:

<b>Original</b>	<b>Prewitt Horizontal</b>	<b>Prewitt Vertical</b>
-----------------	---------------------------	-------------------------





O objetivo deste filtro é realçar os contornos, tanto horizontalmente como verticalmente, das imagens de teste.

Este efeito é conseguido ao ser calculado o gradiente da imagem usando uma matriz 3x3.

Para o filtro `Prewitt`, a matriz usada é:

```
[ 1  1  1
  0  0  0
 -1 -1 -1 ]
```

enquanto que para o filtro `Sobel`, a matriz usada é:

```
[ 1  2  1
  0  0  0
 -1 -2 -1 ]
```

Observa-se que os contornos horizontais e verticais adquiriram uma cor branca, existindo ligeiras diferenças entre o filtro `Prewitt` e `Sobel`.

A principal diferença reside na deteção de altas frequências nos contornos. Devido ao filtro `Prewitt` não realizar suavização, é muitas vezes difícil encontrar contornos em situações com altas frequências, enquanto que o filtro `Sobel` realiza suavização, tendo maior facilidade em encontrar contornos em altas frequências. Este efeito é facilmente observável novamente nas barbas do Tigre, nas qual é mais facilmente ver os contornos com `Sobel` do que `Prewitt`.

### Filtro Unsharp

Finalmente, para o filtro `unsharp`, os resultados foram os seguintes:



O efeito deste filtro é realçar contornos das imagens. Este aumenta a variação da frequência na imagem. O resultado disto é que em zonas com variações de frequência altas, como por exemplo nas barbas do tigre, a variação de frequência aumenta e é torna-se fácil identificar cada contorno