Multimedia Services and Applications (SAM)
2021/2022

Second Lab Assignment Report
Fundamentals of visual signals processing

João Carlos Machado Rocha Pires (UP201806079)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Contents

1. Color spaces

1.1 RGB to HSV

The following Matlab script starts by importing a picture that receives through the only argument associated with the function *rgb_hsv* and then divides it into 3 matrices, each one for one of the RGB components. Then, it converts the original picture into HSV and divides again into 3 matrices, one for each HSV component, showing the results for both conversions in two separate pictures.

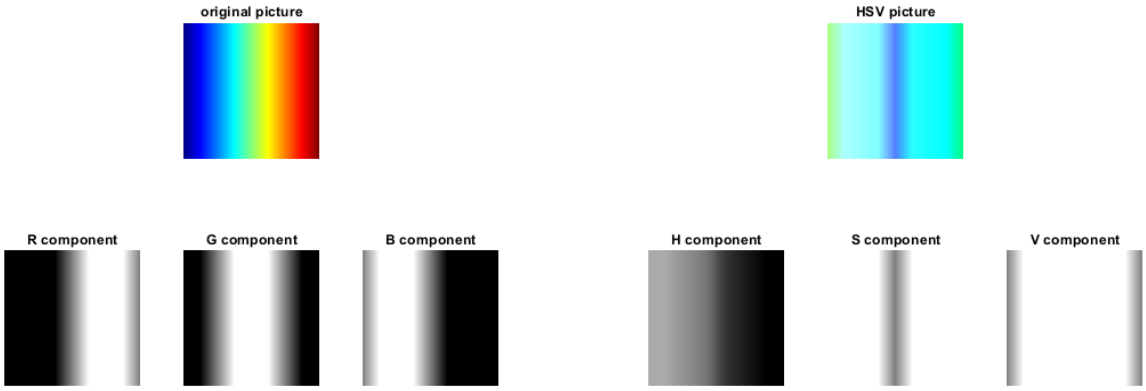
```
function rgb_hsv(originalPicture);

O=imread(originalPicture);
if contains(originalPicture,"elephant.bmp",'IgnoreCase',true)
    O = cat(3, O, O, O);
end
I=im2double(O);

R=I(:,:,1);
G=I(:,:,2);
B=I(:,:,3);

figure(1);
subplot(2,3,2),imshow(I); title('original picture');
subplot(2,3,4),imshow(R); title('R component');
subplot(2,3,5),imshow(G); title('G component');
subplot(2,3,6),imshow(B); title('B component');

% conversion to HSV

HSV = rgb2hsv(O);
[h,s,v] = imsplit(HSV);

figure(2);
subplot(2,3,2),imshow(HSV); title('HSV picture');
subplot(2,3,4),imshow(h); title('H component');
subplot(2,3,5),imshow(s); title('S component');
subplot(2,3,6),imshow(v); title('V component');
```

This script was tested with the pictures 'testRGB.bmp', 'floresVermelhas.bmp', 'folhasVerdes.bmp' and 'elephant.bmp'.

Note 1: Despite the fact that it's mentioned in the Lab guide, the file 'praia.bmp' was not in the folder at Moodle.

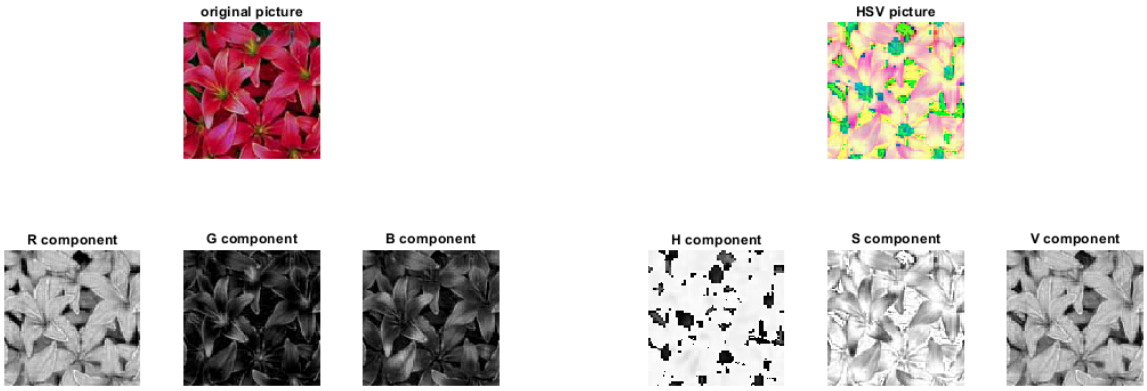
Note 2: For the picture 'elephant.bmp', it was necessary to add an extra line in the code to handle the fact that the picture is in grey scale.

The following pictures show the outputs of the script above for each one of the pictures used to test it.



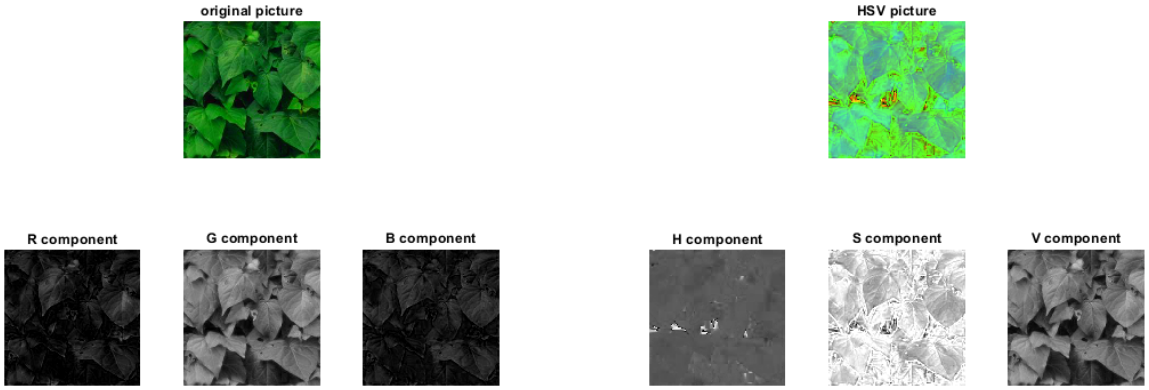
(a) RGB (b) HSV

Figure 1.1: 'testRGB.bmp' - RGB and HSV components



(a) RGB (b) HSV

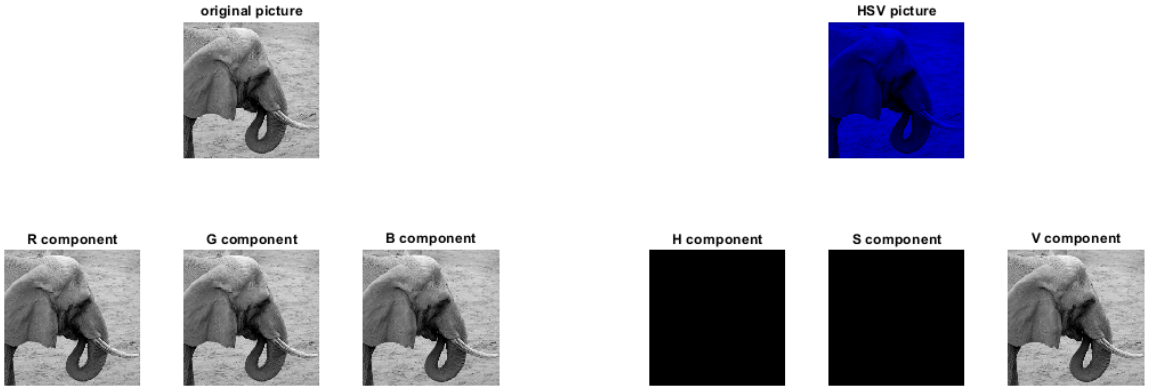
Figure 1.2: 'floresVermelhas.bmp' - RGB and HSV components



(a) RGB

(b) HSV

Figure 1.3: 'folhasVerdes.bmp' - RGB and HSV components



(a) RGB

(b) HSV

Figure 1.4: 'elephant.bmp' - RGB and HSV components

For the 'testRGB.bmp' picture, the RGB components have a white band in the corresponding color space. On the other hand, the HSV conversion of the picture shows an almost complete white representation for the Value and Saturation, which was expected since the picture is itself an spectrum of the visible colors and the Hue component ranges from grey to black.

For the 'floresVermelhas.bmp' and 'folhasVerdes.bmp', the components R and G, respectively and as expected, contain a grey representation of the picture, since this color is the most represented one. On the other side, the HSV conversion of this two pictures shows a lighter Hue component for the 'floresVermelhas.bpm' picture, while the lighter component for the 'folhasVerdes.bmp' is the Satura-

tion. This is justified by the fact that Red has 0 Hue and Green has 0 Saturation (the exact opposite to the RGB representation, where those values are 255 for Red and 255 for Green, respectively).

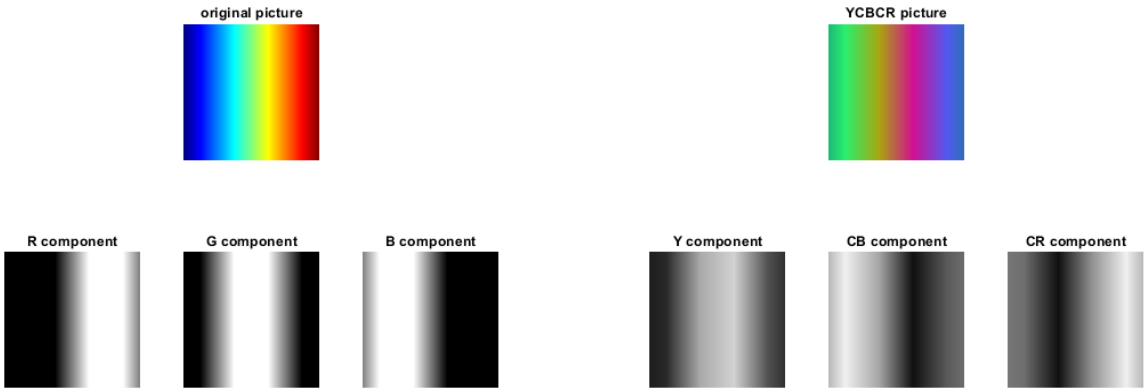
Last but not least, the 'elephant.bmp' picture has 3 equal RGB components, since it's a grey scale picture. In the HSV conversion, the only non-black component is the Value, since both the Hue and the Saturation are 0.

1.2 RGB to YCbCr

The following script is similar to the one above, but, this time, the conversion is not for HSV, but for YCbCr, i.e. luminance (Y) and chrominance (Cb and Cr).

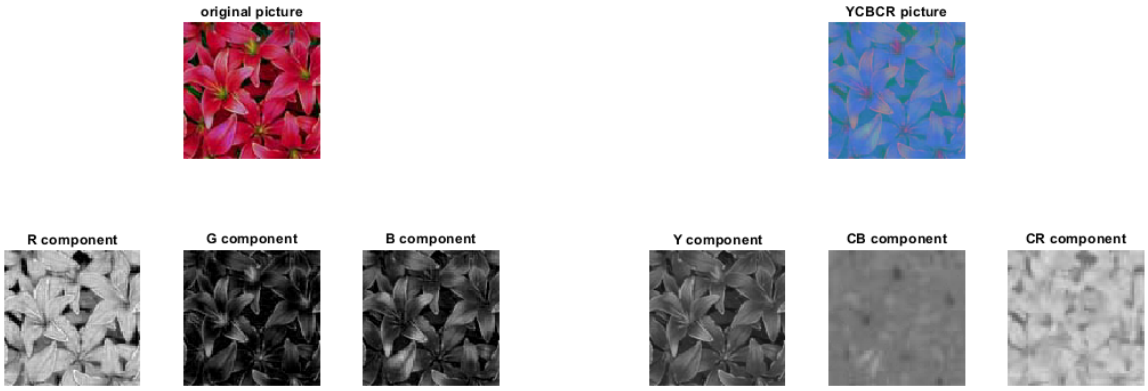
```
function rgb_ycbcr(originalPicture);

O=imread(originalPicture);
if contains(originalPicture,"elephant.bmp",'IgnoreCase',true)
    O = cat(3, O, O, O);
end
I=im2double(O);

R=I(:,:,1);
G=I(:,:,2);
B=I(:,:,3);

figure(1);
subplot(2,3,2),imshow(I); title('original picture');
subplot(2,3,4),imshow(R); title('R component');
subplot(2,3,5),imshow(G); title('G component');
subplot(2,3,6),imshow(B); title('B component');

% conversion to YCBCR

YCBCR = rgb2ycbcr(O);
[y,cb,cr] = imsplit(YCBCR);

figure(2);
subplot(2,3,2),imshow(YCBCR); title('YCBCR picture');
subplot(2,3,4),imshow(y); title('Y component');
subplot(2,3,5),imshow(cb); title('CB component');
subplot(2,3,6),imshow(cr); title('CR component');
```

Using the same pictures as above, the following pictures show on the left the original picture and the RGB components and, on the right, the YCbCr components.



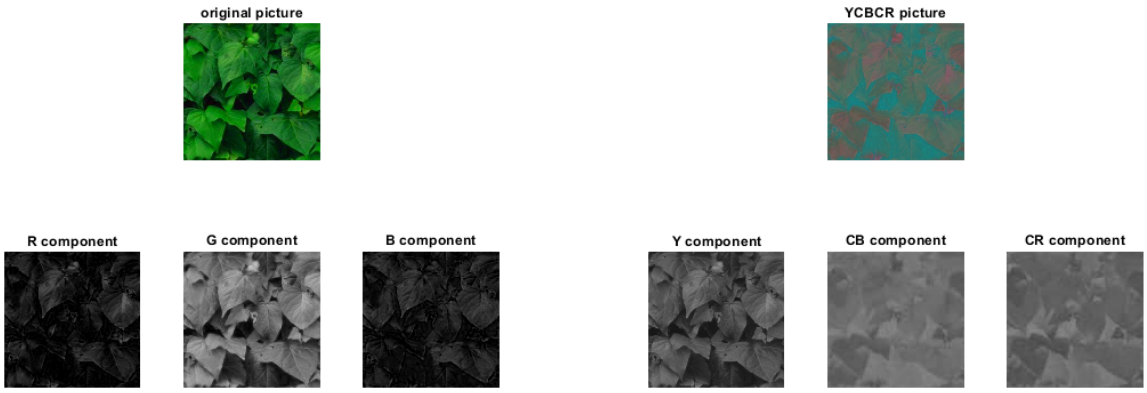
(a) RGB (b) YCbCr

Figure 1.5: 'testRGB.bmp' - RGB and YCbCr components



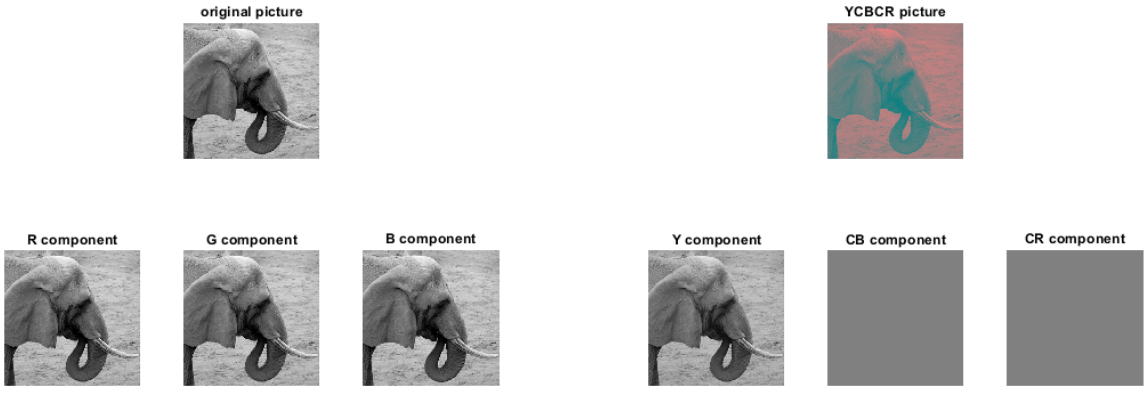
(a) RGB (b) YCbCr

Figure 1.6: 'floresVermelhas.bmp' - RGB and YCbCr components



(a) RGB (b) YCbCr

Figure 1.7: 'folhasVerdes.bmp' - RGB and YCbCr components



(a) RGB (b) YCbCr

Figure 1.8: 'elephant.bmp' - RGB and YCbCr components

For the 'testRGB.bmp' picture, the YCbCr conversion shows a similar representation in each one of the components, considering that the white band is now somewhere between the grey and white.

For the pictures 'floresVermelhas.bmp' and 'folhasVerdes.bmp', the conversion is almost imperceptible, with the luminance component being the only recognizable one, even that in grey scale.

Last but not least, the 'elephant.bmp' conversion led to a single Y component, since the chrominance is null due to the original picture not having a single color.

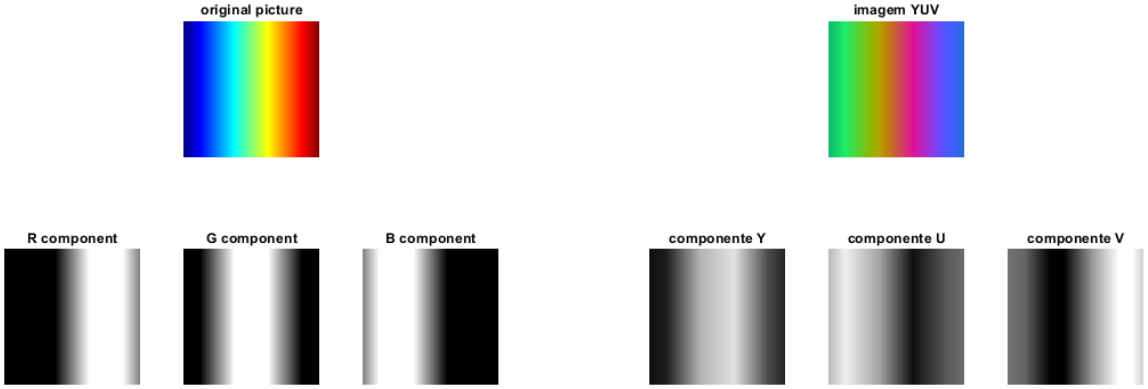
1.3 RGB to YUV

To finish the first experiment, the same process as above was performed, this time using the script *rgb2yuv.m*.

Note 3: In order to test the script with the 'elephant.bmp' picture, the following piece of code was added:

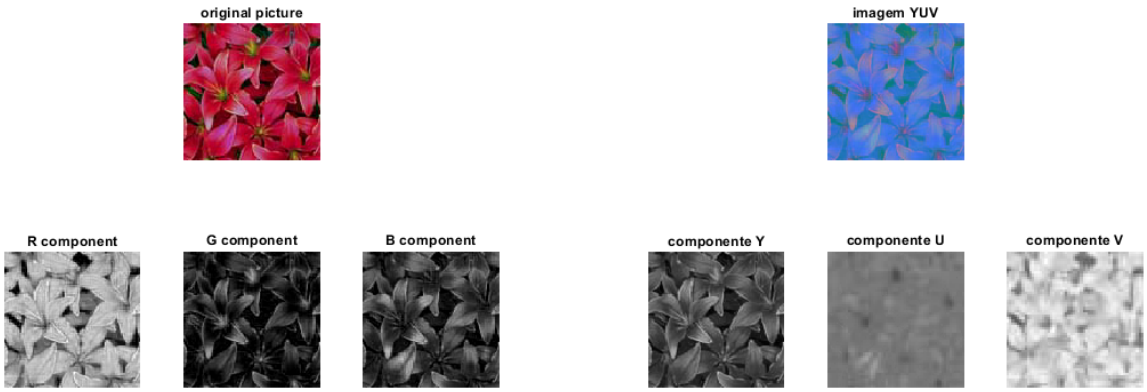
```
if contains(imagemEntrada,"elephant.bmp",'IgnoreCase',true)
I = cat(3, I, I, I);
end
```

The following pictures show the results obtained:



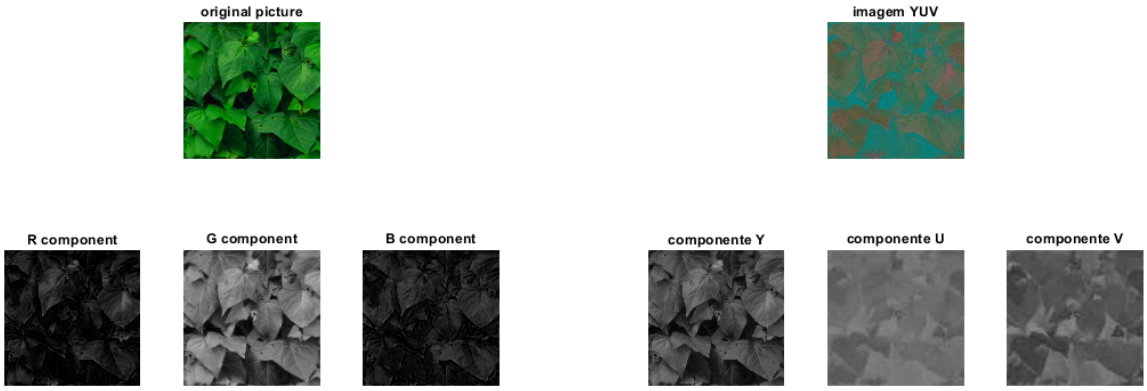
(a) RGB (b) YUV

Figure 1.9: 'testRGB.bmp' - RGB and YUV components



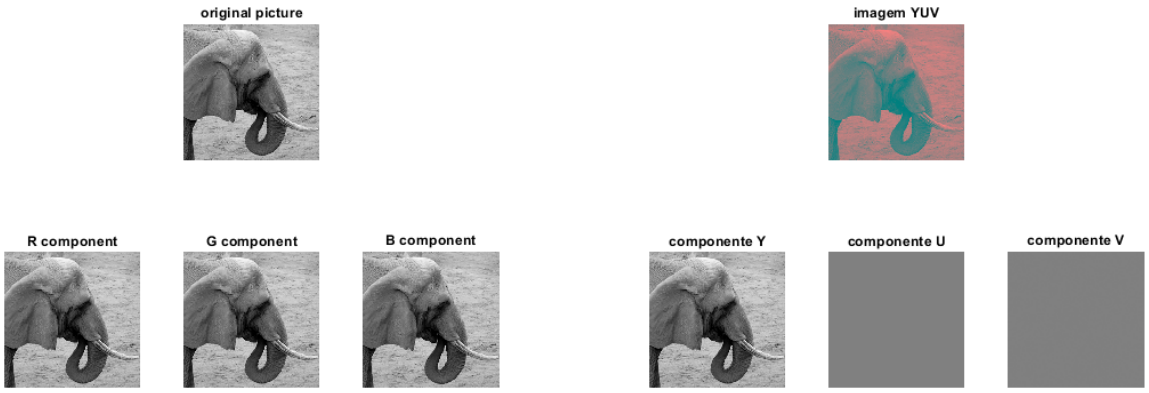
(a) RGB (b) YUV

Figure 1.10: 'floresVermelhas.bmp' - RGB and YUV components



(a) RGB (b) YUV

Figure 1.11: 'folhasVerdes.bmp' - RGB and YUV components



(a) RGB (b) YUV

Figure 1.12: 'elephant.bmp' - RGB and YUV components

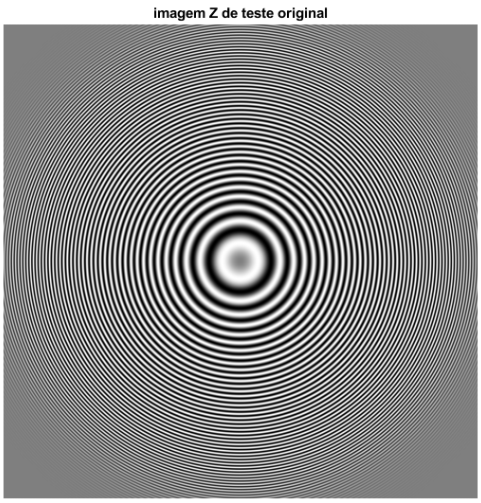
Comparing the results of this conversion with the conversion to YCbCr, the converted pictures and its components are similar.

2. Changing the spatial dimensions of a picture (w/ and w/o filters)

For this experiment, I've decided to test the script *ampliaReduz.m* with N equal to 100 and 500 (obtaining pictures generated on runtime with sizes 100x100 and 500x500) and, for each one of the values of N, I've tested the enlargement with a factor equal to 2 and also for the combination with the 3 possible methods: nearest, bilinear and bicubic.

The original pictures (for N=100 and N=500) are the following:

(a) N=100



(b) N=500

Figure 2.1: Original pictures for N=100 and N=500

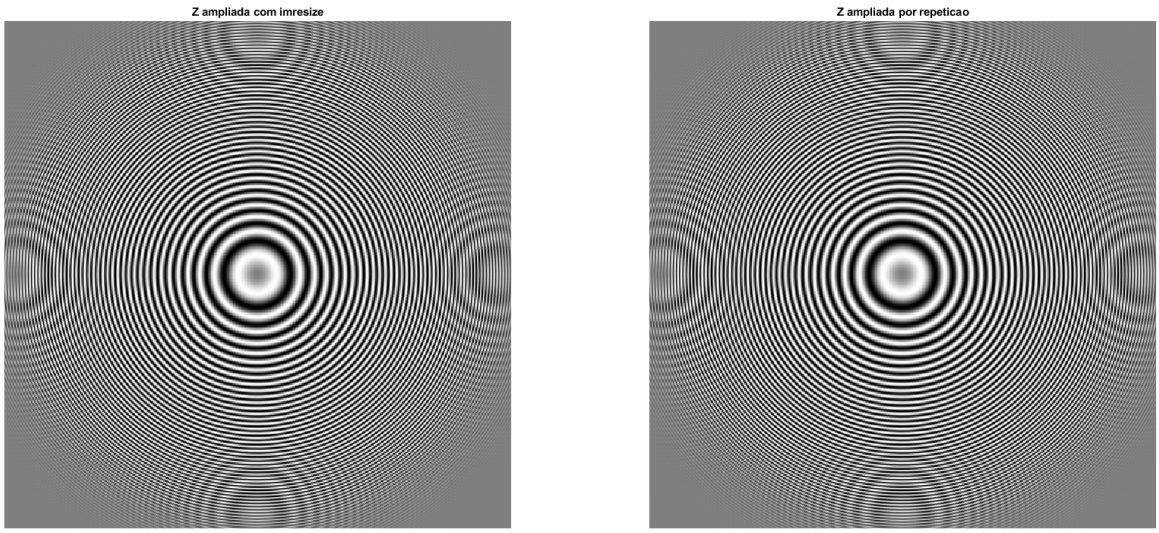
2.1 Nearest

Starting with the nearest, with the pixel calculation depending on the nearest neighbour, for the enlargement with a factor 2, I obtained the following results:

(a) imresize

(b) Repetition

Figure 2.2: N=100 - Transformation using imresize and repetition



(a) imresize (b) Repetition

Figure 2.3: N=500 - Transformation using imresize and repetition

For this method, I didn't noticed a difference between the use of imresize and repetition. However, while for N=100 the enlargement is more or less lossless, for N=500 we can see some "duplicates" of the center circles in the periphery.

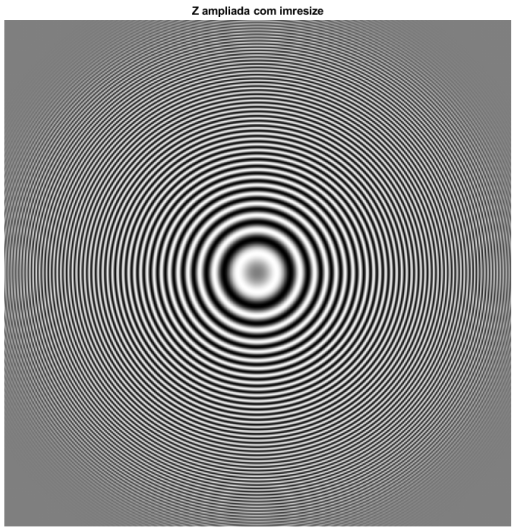
2.2 Bilinear

Secondly, for the method bilinear, with the pixel calculation depending on the bilinear interpolation, for the enlargement with a factor 2, I obtained the following results:

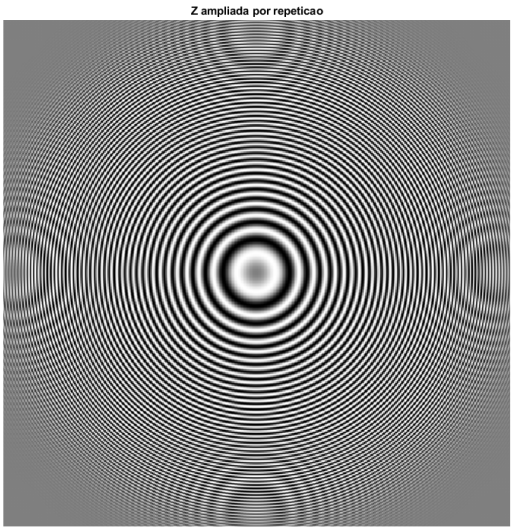
(a) imresize

(b) Repetition

Figure 2.4: N=100 - Transformation using imresize and repetition



(a) imresize



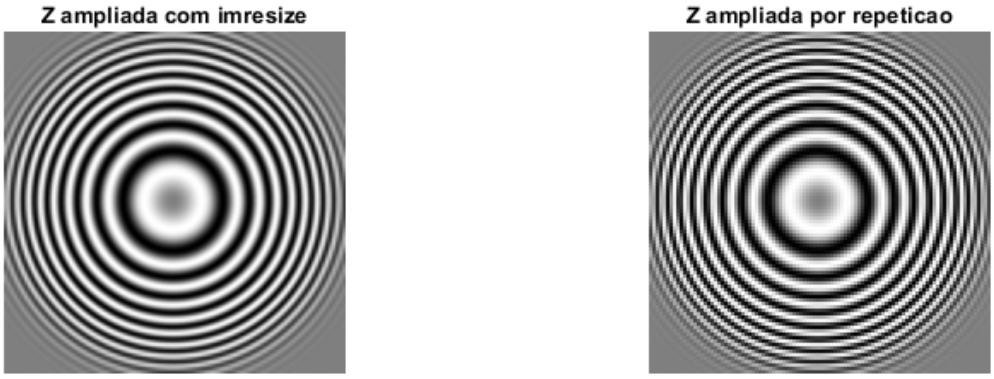
(b) Repetition

Figure 2.5: N=500 - Transformation using imresize and repetition

The conclusions here are similar as above for the N=100. However, for N=500, the difference between the imresize and the repetition becomes now notorious, with this method producing slightly better results.

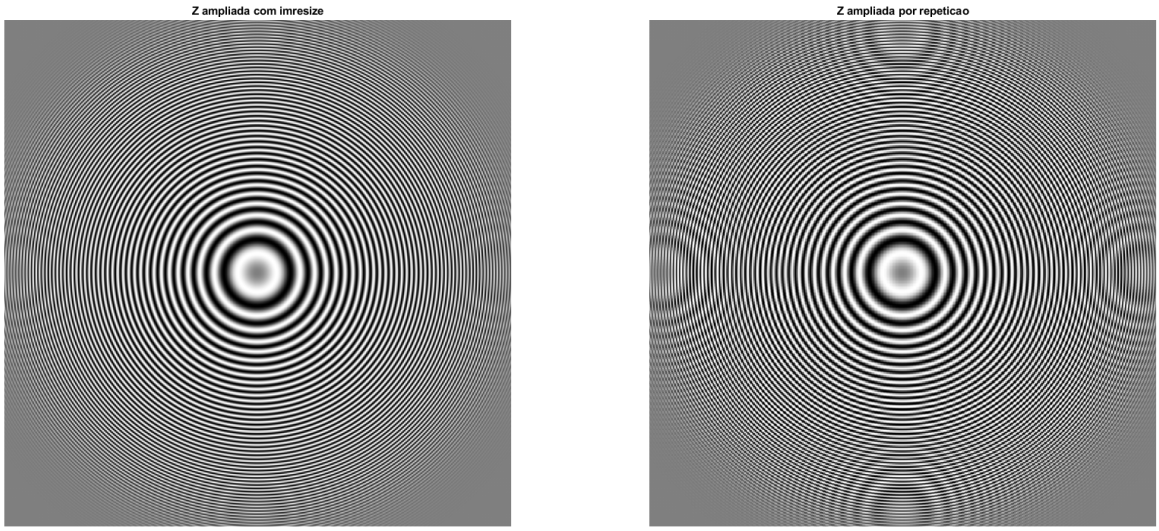
2.3 Bicubic

Last but not least, using the bicubic method, with the pixel calculation depending on the bicubic interpolation, for the enlargement with a factor 2, I obtained the following results:



(a) imresize (b) Repetition

Figure 2.6: N=100 - Transformation using imresize and repetition



(a) imresize (b) Repetition

Figure 2.7: N=500 - Transformation using imresize and repetition

The conclusion here is the same as above (for the bilinear method).

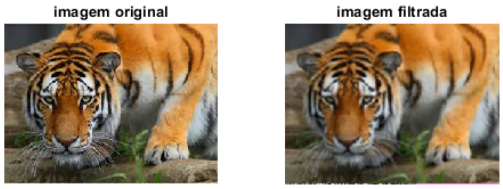
3. Filter experiences

3.1 Testing different filters

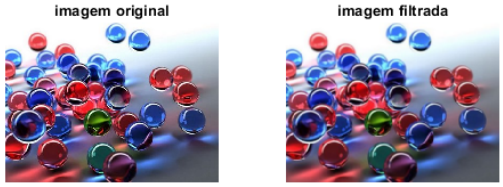
In this first part of the experiment, I used the pictures 'tigre.bmp' and 'berlindesTransparentes.bmp' to test different filters.

3.1.1 Constant dimension (default=3)

The following pictures are the results of the filters applications for each type of filter used.



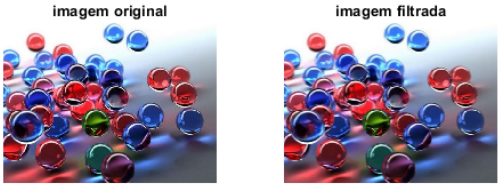
(a) 'average' filter applied to 'tigre.bmp'



(b) 'average' filter applied to 'berlindesTransparentes.bmp'

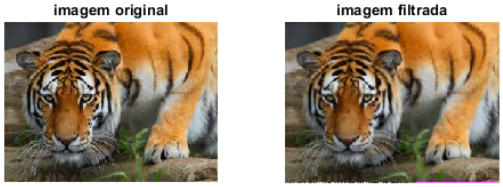
Figure 3.1: 'average' filter

(a) 'motion' filter applied to 'tigre.bmp'

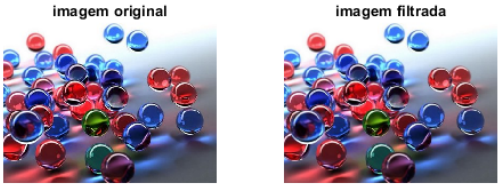


(b) 'motion' filter applied to 'berlindesTransparentes.bmp'

Figure 3.2: 'motion' filter

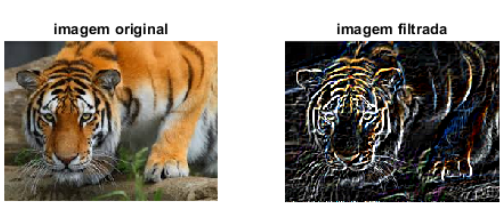


(a) 'gaussian' filter applied to 'tigre.bmp'

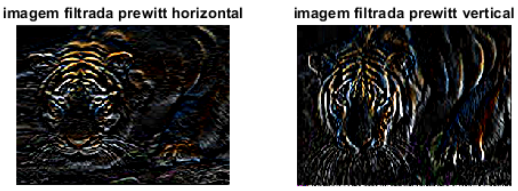


(b) 'gaussian' filter applied to 'berlindesTransparentes.bmp'

Figure 3.3: 'gaussian' filter

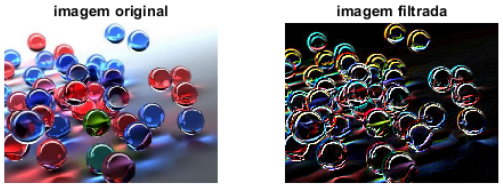


(a) 'prewitt' filter applied to 'tigre.bmp'

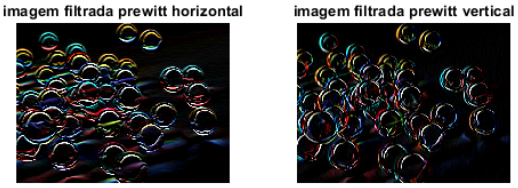


(b) 'prewitt' filter applied to 'tigre.bmp' - vertical and horizontal

Figure 3.4: 'prewitt' filter

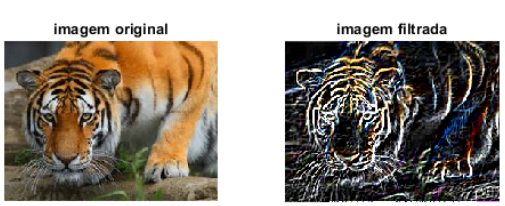


(a) 'prewitt' filter applied to 'berlindesTransparentes.bmp'

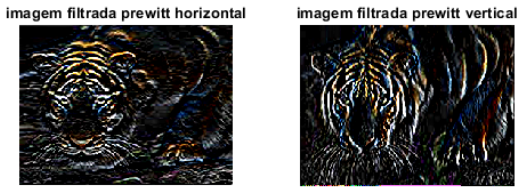


(b) 'prewitt' filter applied to 'berlindesTransparentes.bmp' - vertical and horizontal

Figure 3.5: 'prewitt' filter

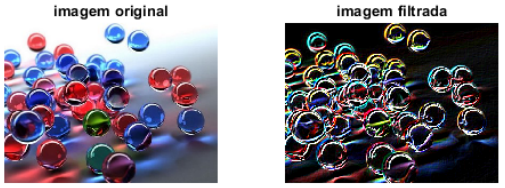


(a) 'sobel' filter applied to 'tigre.bmp'

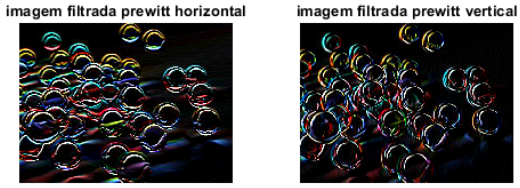


(b) 'sobel' filter applied to 'tigre.bmp' - vertical and horizontal

Figure 3.6: 'sobel' filter



(a) 'sobel' filter applied to 'berlindesTransparentes.bmp'

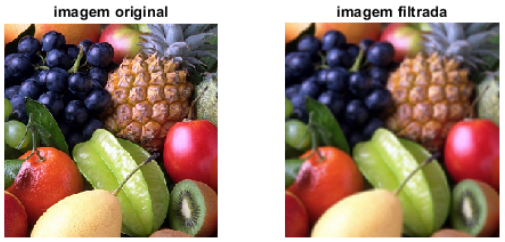


(b) 'sobel' filter applied to 'berlindesTransparentes.bmp' - vertical and horizontal

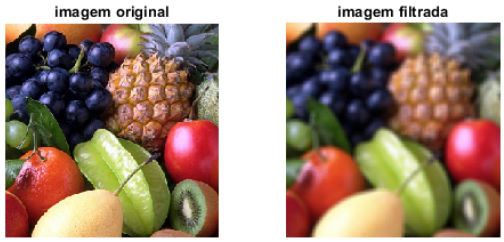
Figure 3.7: 'sobel' filter

3.1.2 Different dimensions (only for average and gaussian filters)

After the previous experiment, with fixed dimension (3), I tested this time only for the 'fruta.bmp' picture the average and gaussian filters for dimensions 4 and 7. The following pictures show the results.

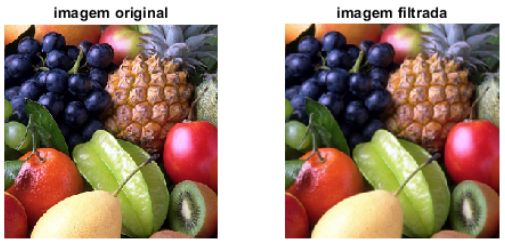


(a) 'average' filter with dimension 4

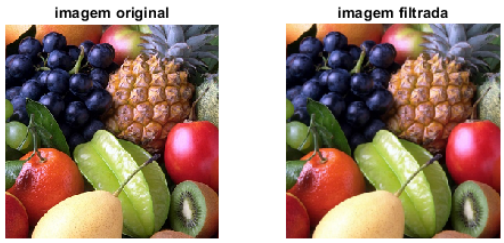


(b) 'average' filter with dimension 7

Figure 3.8: 'average' filter



(a) 'gaussian' filter with dimension 4



(b) 'gaussian' filter with dimension 7

Figure 3.9: 'gaussian' filter

Comparing the results obtained for dimension equal to 4, the average filter presents a more unfocused picture when comparing to the gaussian filter. The same happens to the dimension equal to 7, but the blur is even more accentuated for the average filter.

3.2 Working with noise

In this last experiment, I've used the pictures 'ruido1.jpg' and 'ruido2.jpg' to test the reduction of noise and the pictures 'berlindes.bmp' and 'dollarsTexture.bmp' to introduce noise with two values of

noise quantity: 0.3 and 0.7.

3.2.0.1 Reduce noise

The following two pictures show the reduction of noise for the pictures 'ruido1.jpg' and 'ruido2.jpg'.

(a) Original Picture (b) Reduction of noise

Figure 3.10: Original vs Noise reduction for 'ruido1.jpg'

(a) Original Picture (b) Reduction of noise

Figure 3.11: Original vs Noise reduction for 'ruido2.jpg'

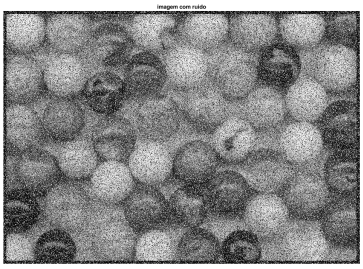
This experience reduces the noise points in the original pictures, making the resultant picture looking like it has no noise points. Perceptually, the resultant pictures looks a little bit unfocused, but it's significantly better than the original one.

3.2.0.2 Introduce noise

To introduce noise, I've used the the pictures 'berlindes.bmp' and 'dollarsTexture.bmp' and two values of noise quantity: 0.3 and 0.7.

The following pictures show the result of the application of such values to each picture. On the left, the original picture. In the middle, the picture with the noise introduced. On the right, the resultant picture of the reduction of noise introduced.

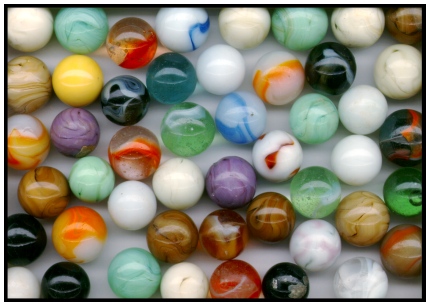
(a) Original Picture



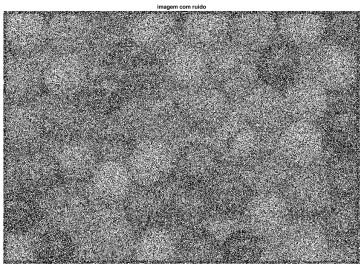
(b) Introduction of noise

(c) Reduction of introduced noise

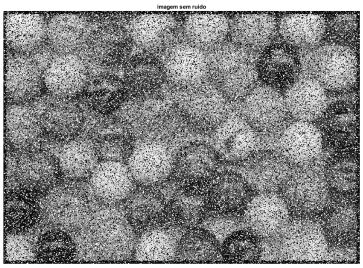
Figure 3.12: Introduction of noise to 'berlindes.bmp' picture with noise quantity equal to 0.3



(a) Original Picture



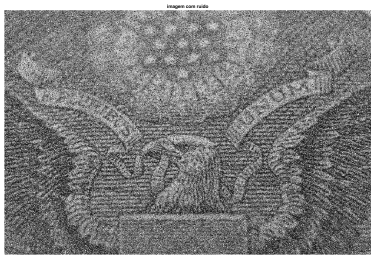
(b) Introduction of noise



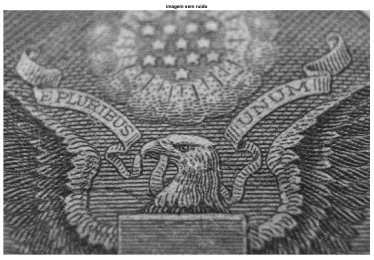
(c) Reduction of introduced noise

Figure 3.13: Introduction of noise to 'berlindes.bmp' picture with noise quantity equal to 0.7

(a) Original Picture



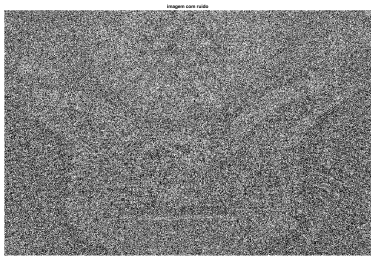
(b) Introduction of noise



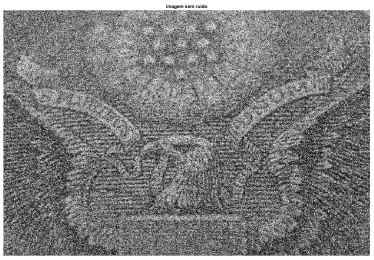
(c) Reduction of introduced noise

Figure 3.14: Introduction of noise to 'dollarsTexture.bmp' picture with noise quantity equal to 0.3

(a) Original Picture



(b) Introduction of noise



(c) Reduction of introduced noise

Figure 3.15: Introduction of noise to 'dollarsTexture.bmp' picture with noise quantity equal to 0.7

As expected, the higher the value for the noise quantity, the more imperceptible the noise picture becomes, with the reduction to that one being even more difficult as that value increases, meaning that as the value tends to become closer to 1, the more difficult it is to obtain the original picture again.