FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Multimedia and New Services

Lab 2 - Principles of Visual Processing in Matlab

Mário Santos
up201503406

Mestrado Integrado em Engenharia Informática e Computação

February 1, 2021

Multimedia and New Services
Lab 2 - Principles of Visual Processing in Matlab

Mário Santos
up201503406

Mestrado Integrado em Engenharia Informática e Computação

February 1, 2021

Contents

List of Figures

List of Tables

Chapter 1

1. Color Spaces

In this chapter we convert a bitmap image with RGB colorspace to HSV, YCbCr and YUV colorspaces.

1.1 RGB to HSV

In this section a script was developed to serve the following purposes:

- Import a bitmap image with RGB colorspace and display it
- Separate each RGB component and display them
- Convert the image to HSV colorspace and display it
- Separate each HSV component and display them

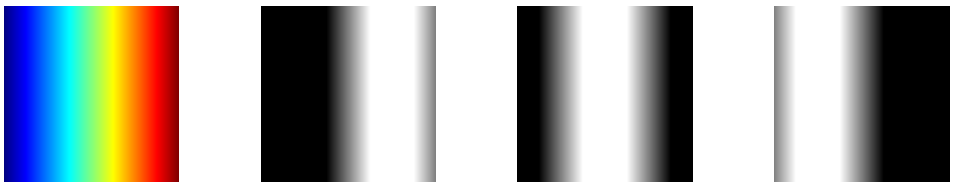
The following script was developed:

```
function rgbToHsv = rgbToHsv(filename)

% i) Import a bitmap image and presents that image on the screen

im = imread(filename);

if size(im,3)~= 3
    im = cat(3,im,im,im);
end

figure(1);imshow(im);title('RGB image');

% Save image
filename = extractBefore(filename,".");
path = sprintf('figures/RGB/%s_RGB.png', filename);
imwrite(im,path)
fprintf('\n Press any key view RGB components'); pause
```

```
18
19 fprintf('\n Press any key view RGB components'); pause
20
21 % ii) Separate each RGB component in a different matrix and visualise each one on
       the screen
22
23 r = im(:,:,1);
24 g = im(:,:,2);
25 b = im(:,:,3);
26
27 figure(2);imshow(r); title('RGB image - R');
28 path = sprintf('figures/RGB/%s_R.png', filename);
29 imwrite(r,path)
30
31 figure(3);imshow(g); title('RGB image - G');
32 path = sprintf('figures/RGB/%s_G.png', filename);
33 imwrite(g,path)
34
35 figure(4);imshow(b); title('RGB image - B');
36 path = sprintf('figures/RGB/%s_B.png', filename);
37 imwrite(b,path)
38
39 fprintf('\n Press any key to view HSV image'); pause
40
41 % iii) Convert the RGB image to the HSV color space and present it on the screen;
42
43 im_hsv = rgb2hsv(im);
44 figure(5);imshow(im_hsv);title('HSV image');
45 path = sprintf('figures/HSV/%s_HSV.png', filename);
46 imwrite(im_hsv,path)
47
48 fprintf('\n Press any key to view separate HSV components'); pause
49
50 % iv) Separate each HSV component in a different matrix and visualise each
51 % one on the screen
52
53 h = im_hsv(:,:,1);
54 s = im_hsv(:,:,2);
55 v = im_hsv(:,:,3);
56
57 figure(6);imshow(h); title('HSV image - H');
58 path = sprintf('figures/HSV/%s_H.png', filename);
59 imwrite(h,path)
60 figure(7);imshow(s); title('HSV image - S');
61 path = sprintf('figures/HSV/%s_S.png', filename);
62 imwrite(s,path)
63 figure(8);imshow(v); title('HSV image - V');
64 path = sprintf('figures/HSV/%s_V.png', filename);
65 imwrite(v,path)
```

```
66
67 rgbToHsv = 0; % Terminated successfully
68 end
```

Using the built-in Matlab function imread() we read the image and then displayed it using another built-in function, imshow(). We also checked if the image was in grayscale format, and converted it to RGB, assigning equal values to each RGB component. The results were as follows:



(a) Original image (RGB) (b) Original image (R) (c) Original image (G) (d) Original image (B)

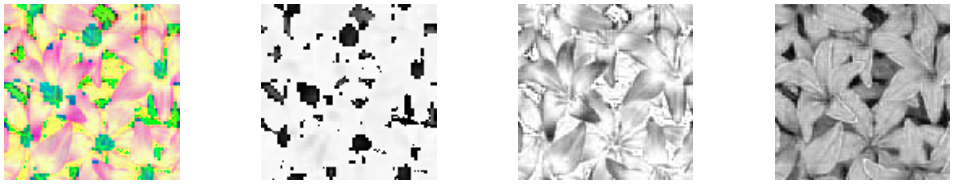
Figure 1.1: Original image testRGB.bmp and its separate RGB components

(a) HSV image (b) HSV image (H) (c) HSV image (S) (d) HSV image (V)

Figure 1.2: HSV converted image testRGB.bmp and its separate HSV components

(a) Original image (RGB) (b) Original image (R) (c) Original image (G) (d) Original image (B)

Figure 1.3: Original image floresVermelhas.bmp and its separate RGB components

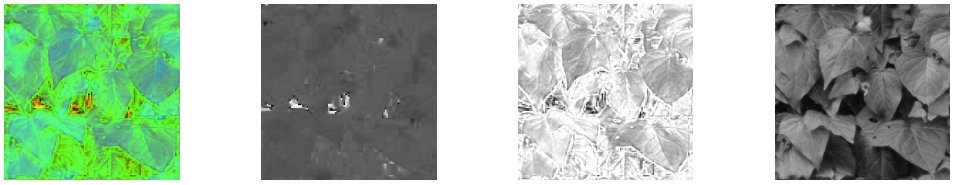


(a) HSV image (b) HSV image (H) (c) HSV image (S) (d) HSV image (V)

Figure 1.4: HSV converted image floresVermelhas.bmp and its separate HSV components

(a) Original image (RGB) (b) Original image (R) (c) Original image (G) (d) Original image (B)

Figure 1.5: Original image folhasVerdes.bmp and its separate RGB components



(a) HSV image (b) HSV image (H) (c) HSV image (S) (d) HSV image (V)

Figure 1.6: HSV converted image folhasVerdes.bmp and its separate HSV components

(a) Original image (RGB) (b) Original image (R) (c) Original image (G) (d) Original image (B)

Figure 1.7: Original image praia.bmp and its separate RGB components

(a) HSV image (b) HSV image (H) (c) HSV image (S) (d) HSV image (V)

Figure 1.8: HSV converted image praia.bmp and its separate HSV components

(a) Original image (RGB) (b) Original image (R) (c) Original image (G) (d) Original image (B)

Figure 1.9: Original image elephant.bmp and its separate RGB components

(a) HSV image (b) HSV image (H) (c) HSV image (S) (d) HSV image (V)

Figure 1.10: HSV converted image elephant.bmp and its separate HSV components

Considering the RGB colorspace, each pixel uses 3 bytes, each corresponding to one of the RGB componentes (Red, Green, Blue). Every other color is obtained by a combination of these.

The results show that the most abundant colors in the original image are the brightest in their corresponding component. As an example, when looking at the RGB components from the image floresVermelhas.bmp, which is almost entirely red, we can clearly see the Red component is the brightest. We can also see there is no difference between the different components for the elephant.bmp image, this is because the image is in grayscale and thus all components have equal values.

Now considering the HSV colorspace, its behavior is much closer to how the human eye perceives colors. The colorspace is defined by the following parameters;

- Hue determines the color from red (0º) to blue (360º)
- Saturation defines a range from pure color (100%) to gray (0%).
- Value defines the brightness (0% to 100%)

Looking at the first image, testRGB.bmp, we can tell that a the color red corresponds to white in the Hue component, and blue to black. The Saturation values are as expected, the higher the saturation of a given pixel, the purer the color and the closer to white is its Saturation component, a good example is the sky from the praia.bmp image. Finally the Brightness values are also as expected, the brighter the image the whiter the Brightness component. Regarding the elephant.bmp image, it's Hue value is 0, since the RGB values are the same, the MAX = MIN. Its Saturation is also 0, as it results from (MAX-MIN)/MAX. The brightness value, MAX, is the same as in the original image.

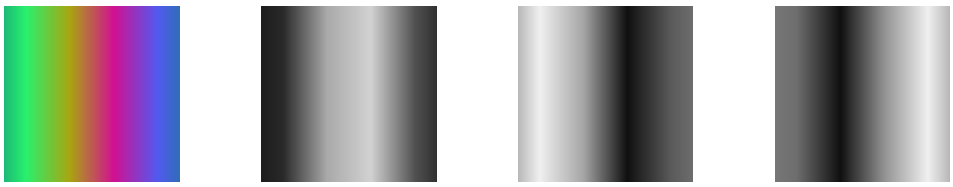
1.2 RGB to YCbCr

In this section a similar script was developed but it converts images in RGB colorspace to YCbCr. The following is the code developed:

```
function rgbToYcbcr = rgbToYcbcr(filename)

% i) Import a bitmap image and presents that image on the screen

im = imread(filename);

if size(im,3)~= 3
    im = cat(3,im,im,im);
end

figure(1);imshow(im);title('RGB image');

% Save image
filename = extractBefore(filename,".");
path = sprintf('figures/RGB/%s_RGB.png', filename);
imwrite(im,path)

% ii) Separate each RGB component in a different matrix and visualise each one on
    the screen

r = im(:,:,1);
g = im(:,:,2);
b = im(:,:,3);

figure(2);imshow(r); title('RGB image - R');
path = sprintf('figures/RGB/%s_R.png', filename);
imwrite(r,path)

figure(3);imshow(g); title('RGB image - G');
path = sprintf('figures/RGB/%s_G.png', filename);
imwrite(g,path)

figure(4);imshow(b); title('RGB image - B');
path = sprintf('figures/RGB/%s_B.png', filename);
imwrite(b,path)

% iii) Convert the RGB image to the YCbCr color space and present it on the screen;

im_ycbcr = rgb2ycbcr(im);
figure(5);imshow(im_ycbcr);title('YCbCr image');
path = sprintf('figures/YCbCr/%s_YCbCr.png', filename);
imwrite(im_ycbcr,path)

```

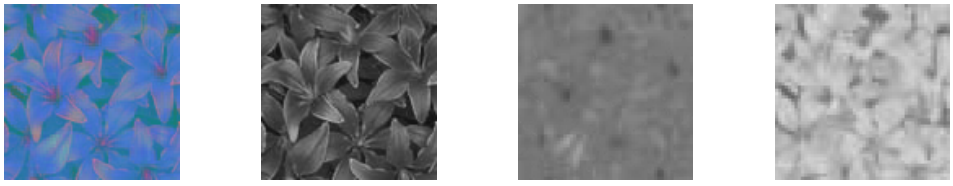
```
43 % iv) Separate each YCbCr component in a different matrix and visualise each
44 % one on the screen
45
46 y = im_ycbcr(:,:,1);
47 cb = im_ycbcr(:,:,2);
48 cr = im_ycbcr(:,:,3);
49
50 figure(6);imshow(y); title('YCbCr image - Y');
51 path = sprintf('figures/YCbCr/%s_Y.png', filename);
52 imwrite(y,path)
53 figure(7);imshow(cb); title('YCbCr image - Cb');
54 path = sprintf('figures/YCbCr/%s_Cb.png', filename);
55 imwrite(cb,path)
56 figure(8);imshow(cr); title('YCbCr image - Cr');
57 path = sprintf('figures/YCbCr/%s_Cr.png', filename);
58 imwrite(cr,path)
59
60 rgbToYcbcr = 0; % Terminated successfully
61 end
```

And the results were as follows:



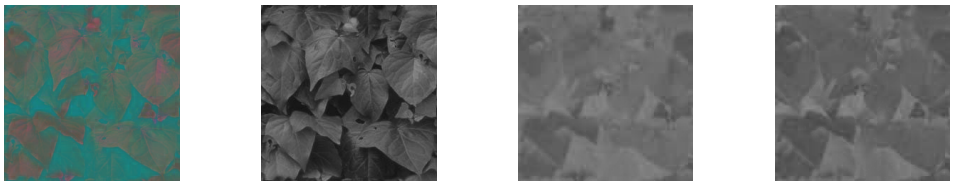
(a) YCbCr image (b) YCbCr image (Y) (c) YCbCr image (Cb) (d) YCbCr image (Cr)

Figure 1.11: YCbCr converted image testRGB.bmp and its separate YCbCr components



(a) YCbCr image (b) YCbCr image (Y) (c) YCbCr image (Cb) (d) YCbCr image (Cr)

Figure 1.12: YCbCr converted image floresVermelhas.bmp and its separate YCbCr components



(a) YCbCr image (b) YCbCr image (Y) (c) YCbCr image (Cb) (d) YCbCr image (Cr)

Figure 1.13: YCbCr converted image folhasVerdes.bmp and its separate YCbCr components

(a) YCbCr image (b) YCbCr image (Y) (c) YCbCr image (Cb) (d) YCbCr image (Cr)

Figure 1.14: YCbCr converted image praia.bmp and its separate YCbCr components

(a) YCbCr image (b) YCbCr image (Y) (c) YCbCr image (Cb) (d) YCbCr image (Cr)

Figure 1.15: YCbCr converted image elephant.bmp and its separate YCbCr components

In the YCbCr model, Y is the luma component and Cb and Cr are the blue-difference and red-difference chroma components. They are calculated using the following formulas:

$$Y = 0.2989 * R + 0.58766 * G + 0.1145 * B \tag{1.1}$$

$$Cb = -0.1687 * R - 0.3312 * G + 0.500 * B \tag{1.2}$$

$$Cr = 0.500 * R - 0.4183 * G + 0.0816 * B \tag{1.3}$$

The results show that the Luma component is clearer than the Chroma components. We can also observe that when an image has more Red color, the Cr component becomes clearer and conversely, when an image has more Blue color, the Cb component becomes clearer.

1.3 RGB to YUV

In this section we used the script supplied, rgb2yuv.m, to convert an RGB image to YUV colorspace, and then reconvert it back to RGB. The following is the scripts code:

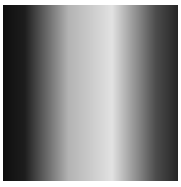
```
function rgb2yuv(imagemEntrada)

I=imread(imagemEntrada);

%I=im2double(I);

if size(I,3)~= 3
    I = cat(3,I,I,I);
end

figure(1);imshow(I);title('RGB image');

% Save image
filename = extractBefore(imagemEntrada,".");
path = sprintf('figures/RGB/%s_RGB.png', filename);
imwrite(I,path)

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


figure(2);imshow(R);title('RGB image - R');
path = sprintf('figures/RGB/%s_RGB_R.png', filename);
imwrite(R,path)
figure(3);imshow(G);title('RGB image - G');
path = sprintf('figures/RGB/%s_RGB_G.png', filename);
imwrite(G,path)
figure(4);imshow(R);title('RGB image - B');
path = sprintf('figures/RGB/%s_RGB_B.png', filename);
imwrite(B,path)
%figure(1);
%subplot(2,3,2),imshow(I); title('imagem original');
%subplot(2,3,4),imshow(R); title('componente R');
%subplot(2,3,5),imshow(G); title('componente G');
%subplot(2,3,6),imshow(B); title('componente B');

%conversao para YUV

Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = .5-0.14713 * R - 0.28886 * G + 0.436 * B;
V = .5+0.615 * R - 0.51499 * G - 0.10001 * B;

```

```
YUV = cat(3,Y,U,V);

figure(5);imshow(YUV);title('YUV image');
path = sprintf('figures/YUV/%s_YUV.png', filename);
imwrite(YUV,path)

figure(6);imshow(Y);title('YUV image - Y');
path = sprintf('figures/YUV/%s_Y.png', filename);
imwrite(Y,path)
figure(7);imshow(U);title('YUV image - U');
path = sprintf('figures/YUV/%s_U.png', filename);
imwrite(U,path)
figure(8);imshow(V);title('YUV image - V');
path = sprintf('figures/YUV/%s_V.png', filename);
imwrite(V,path)
%figure(2);
%subplot(2,3,2),imshow(YUV); title('imagem YUV');
%subplot(2,3,4),imshow(Y); title('componente Y');
%subplot(2,3,5),imshow(U); title('componente U');
%subplot(2,3,6),imshow(V); title('componente V');

%converter de novo para RGB

R = Y + 1.139834576 * (V-0.5);
G = Y -.3946460533 * (U-0.5) -.58060 * (V-0.5);
B = Y + 2.032111938 * (U-0.5);

RGB = cat(3,R,G,B);

figure(3),imshow(RGB), title(' imagem RGB recuperada');
```

And the results were as follows:

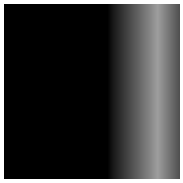
(a) Original

(b) YUV image



(c) YUV image (Y)

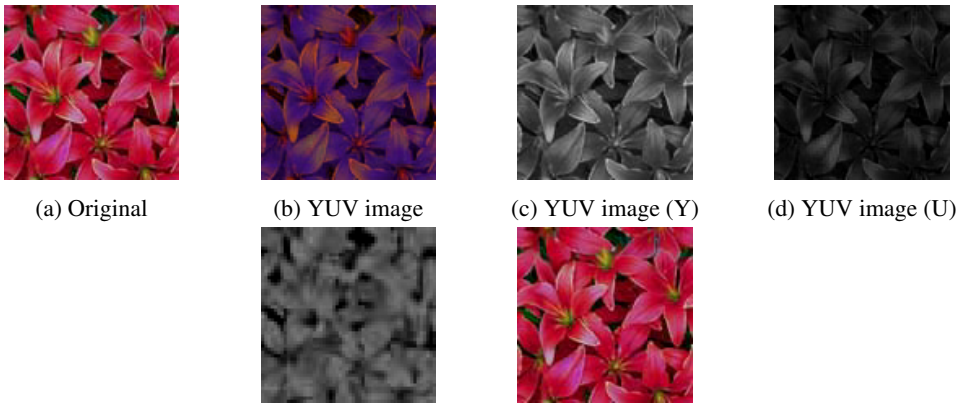
(d) YUV image (U)



(e) YUV image (V)

(f) Reconverted

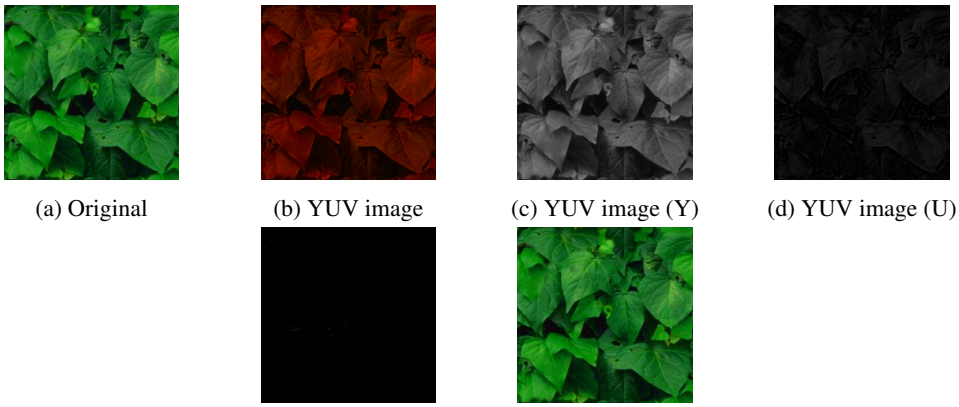
Figure 1.16: YUV converted image testRGB.bmp and its separate YUV components



(a) Original (b) YUV image (c) YUV image (Y) (d) YUV image (U)

(e) YUV image (V) (f) Reconverted

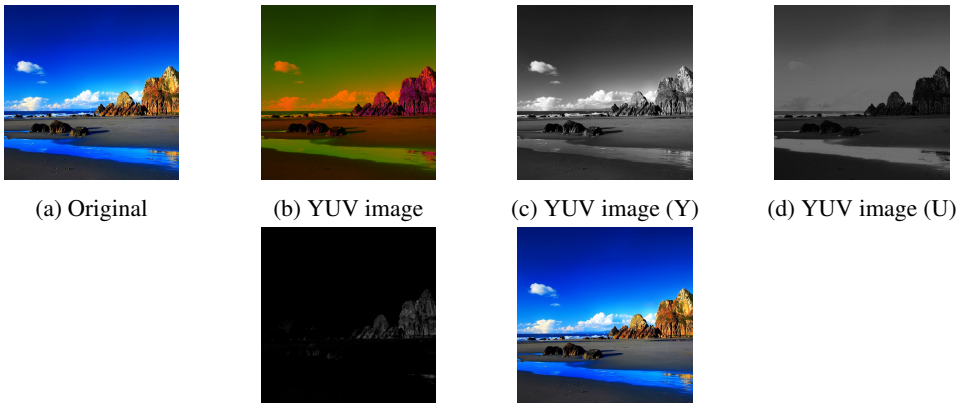
Figure 1.17: YUV converted image floresVermelhas.bmp and its separate YUV components



(a) Original (b) YUV image (c) YUV image (Y) (d) YUV image (U)

(e) YUV image (V) (f) Reconverted

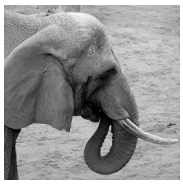
Figure 1.18: YUV converted image folhasVerdes.bmp and its separate YUV components



(a) Original (b) YUV image (c) YUV image (Y) (d) YUV image (U)

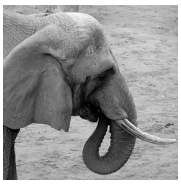
(e) YUV image (V) (f) Reconverted

Figure 1.19: YUV converted image praia.bmp and its separate YUV components



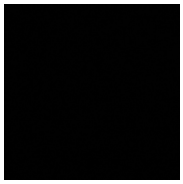
(a) Original

(b) YUV image

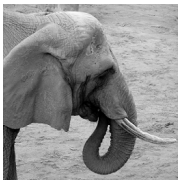


(c) YUV image (Y)

(d) YUV image (U)



(e) YUV image (V)



(f) Reconverted

Figure 1.20: YUV converted image elephant.bmp and its separate YUV components

The YUV colorspace has a Luma component, Y, like the YCbCr colorspace, however instead of blue difference and red difference chroma components we have U (blue projection) and V (red projection). Similarly to our YCbCr conversion, results show that most of the information is stored in the Luma component, and the remaining is stored in the chroma components. And also similarly, we can observe that when the image contains more blue the U value is clearer than V and conversely for when an image contains more red. Additionally we note that we can see no discernible loss in quality when reconverting back to RGB.

Chapter 2

2. Varying Spatial Dimensions

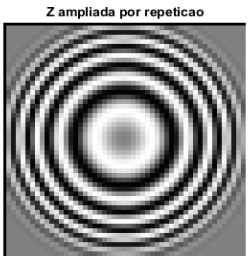
In this chapter we used the supplied script 'ampliaReduz.m', to increase and decrease images using multiple interpolation algorithms. This script uses a test image 'zone plate' generated by the 'imzoneplate.m' script. The available interpolation algorithms were the following:

- **Nearest Neighbor:** Each pixel of the resized image will have the value of the closes corresponding pixel from the original image
- **Bilinear:** Each pixel of the resized imaged will have the value of weighted average of the attributes (color, transparency, etc.) of only the 4 nearest pixels, located in diagonal directions from a given pixel
- **Bicubic:** Each pixel of the resized imaged will have the value of weighted average of the attributes (color, transparency, etc.) of the 16 nearest pixels

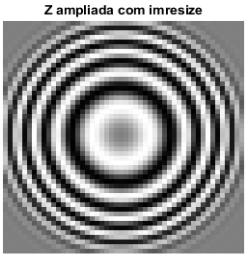
The script "ampliaReduz.m" was used to generate two "zone plate" test images, with 50 by 50 and 400 by 400 pixels respectively. The 50 by 50 image was increased by a factor of 5 (250 by 250), using both duplication and the built-in function "imresize.m". The 400 by 400 image was decreased by a factor of 0.5 (200 by 200), using both elimination and again the built-in function "imresize.m".

Results were as follows:

(a) Original image 50x50 image



(b) 50x50 image increased by a factor of 5 (250x250) by repetition



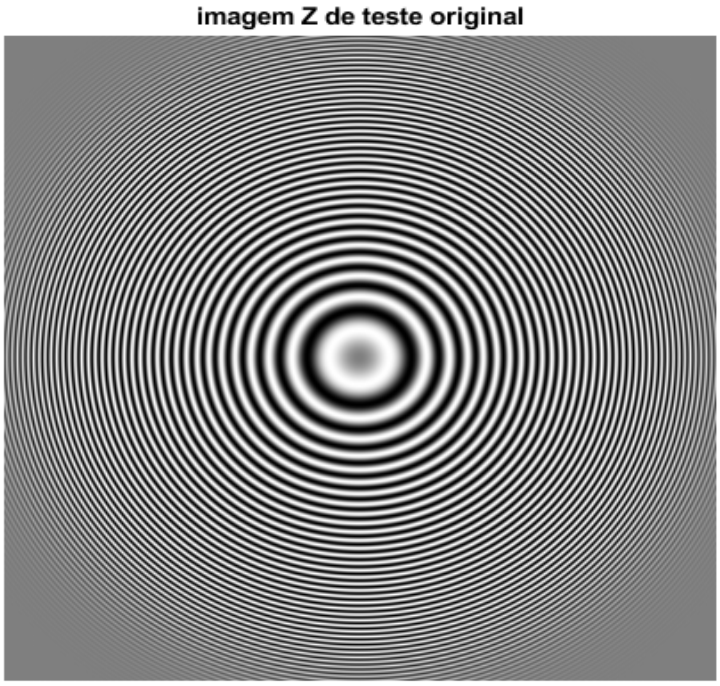
(c) 50x50 image increased by a factor of 5 (250x250) using imresize with 'nearest' method

(d) 50x50 image increased by a factor of 5 (250x250) using imresize with 'bilinear' method

(e) 50x50 image increased by a factor of 5 (250x250) using imresize with 'bicubic' method

Figure 2.1: 50x50 "zone plate" images

When increasing an image, we can see that when using the repetition method, we get the worst results, and the lowest quality image. Using the nearest neighbor interpolation method, there is also a clear aliasing effect. The zone plate image has the smallest frequencies closest to the center and it increases linearly towards the borders. The Nyquist rule is not complied with, resulting in this aliasing effect. When using the Bilinear method, we can see that the aliasing effect is now only partially visible and closer to the borders. Finally, when using the Bicubic method, the aliasing effect is further reduced, resulting in an image which is closest to the original.



(a) Original image 400x400 image

(b) 400x400 image decreased by a factor of 5 (200x200) by repetition

(c) 400x400 image decreased by a factor of 5 (200x200) using imresize with 'nearest' method

(d) 400x400 image decreased by a factor of 5 (200x200) using imresize with 'bilinear' method

(e) 400x400 image decreased by a factor of 5 (200x200) using imresize with 'bicubic' method

Figure 2.2: 400x400 "zone plate" images

When decreasing an image, we can see that when using the elimination method, we get the worst results, and the lowest quality image with the most prominent aliasing effect. Using the nearest neighbor interpolation method, there is also a clear aliasing effect. The zone plate image has the smallest frequencies closest to the center and it increases linearly towards the borders. The Nyquist rule is not complied with, resulting in this aliasing effect. When using the Bilinear method, we can see that the aliasing effect is now only partially visible and closer to the borders. Suprisingly, when using the Bicubic method, even tho the difference is small, it appears that the aliasing effect is more prominent when compared to the Bilinear method.

Chapter 3

3. Filtering Experiments

In this chapter we will address multiple filters that can be applied to images with multiple purposes, from edge-detection to noise elimination.

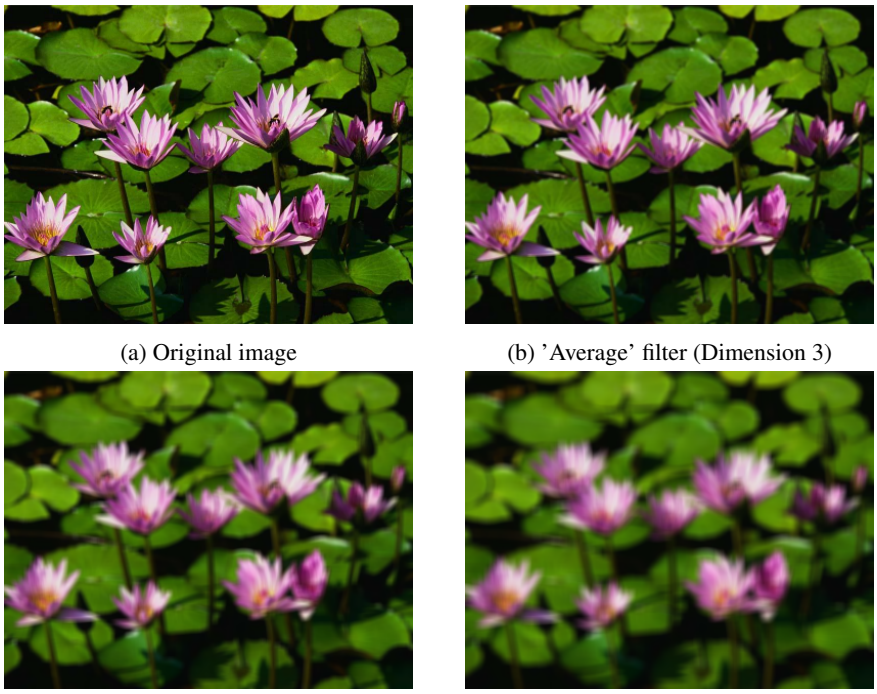
3.1 Different filtering operations

In these experiments the script "filtro2019.m" was used to perform different filtering operations to an image.

The 'Average' filter uses the average of adjacent pixels to reduce high frequencias in images, and makes the images smoother. It also results in some blurriness, which increases with larger dimension values.

The 'Gaussian' filter uses using the weighs of adjacent pixels with Gauss's curve. Like the 'Average' filter it reduces the high frequencies in the image, resulting in a smoother image however, unlike the 'Average' filter it does not blur it.

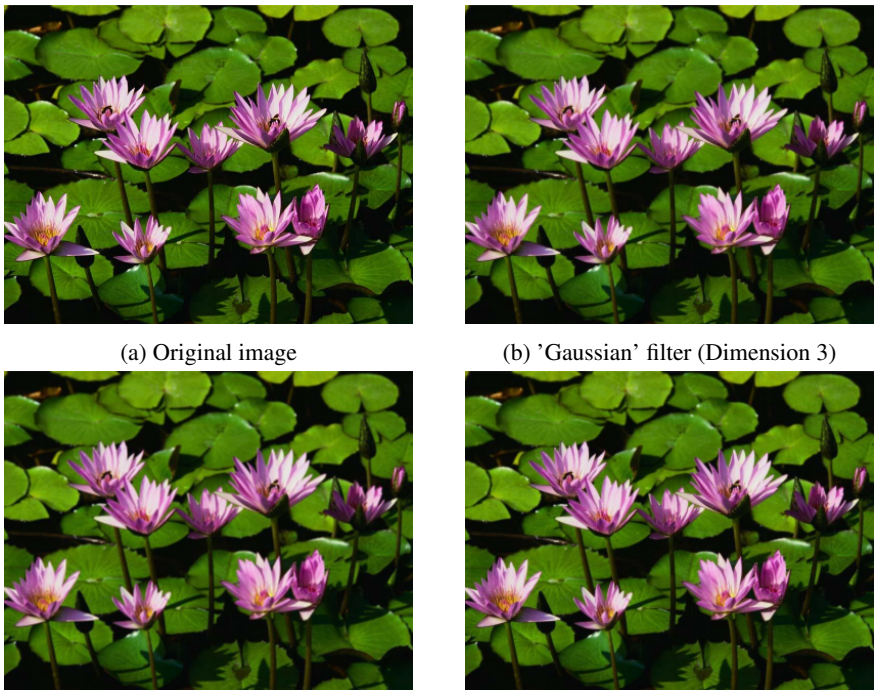
The Motion filter simulates motion, resulting in a blurring effect with direction. The larger the specified 'length' the more motion (blurriness). The direction of motion can also be specificied using the angle parameter, however in this experiment the default value of 0 was used (horizontal motion).



(a) Original image (b) 'Average' filter (Dimension 3)

(c) 'Average' filter (Dimension 6) (d) 'Average' filter (Dimension 9)

Figure 3.1: 'Average' filter



(a) Original image (b) 'Gaussian' filter (Dimension 3)

(c) 'Gaussian' filter (Dimension 6) (d) 'Gaussian' filter (Dimension 9)

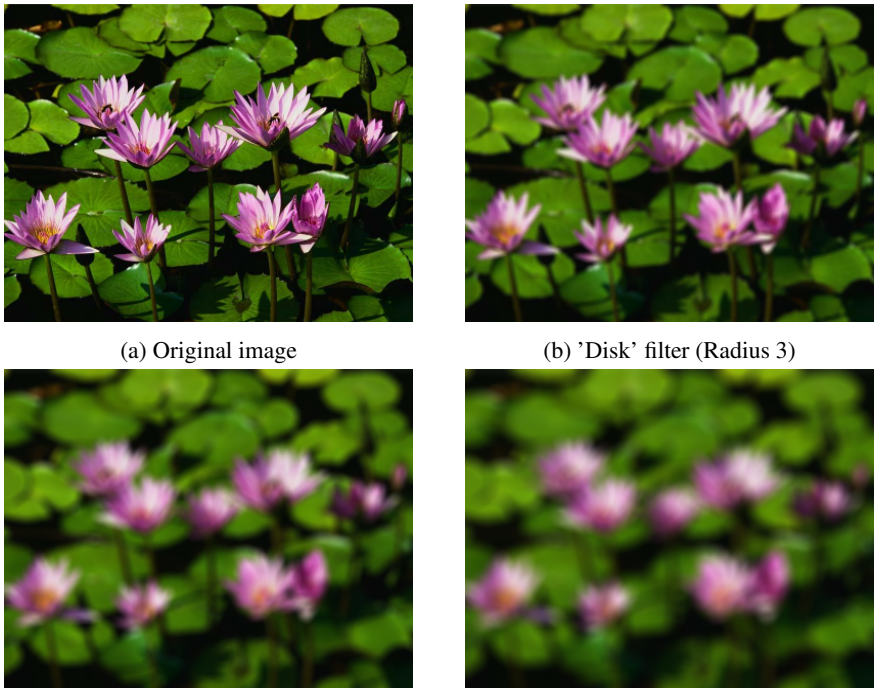
Figure 3.2: 'Gaussian' filter

(a) Original image (b) 'Motion' filter (Length 3)

(c) 'Motion' filter (Length 6) (d) 'Motion' filter (Length 9)

Figure 3.3: 'Motion' filter

The Disk filter applies a cicular average filter to the image. This results in blurry image. The higher the radius, the blurrier the resulting image.



(a) Original image

(b) 'Disk' filter (Radius 3)

(c) 'Disk' filter (Radius 6)

(d) 'Disk' filter (Radius 9)

Figure 3.4: 'Disk' filter

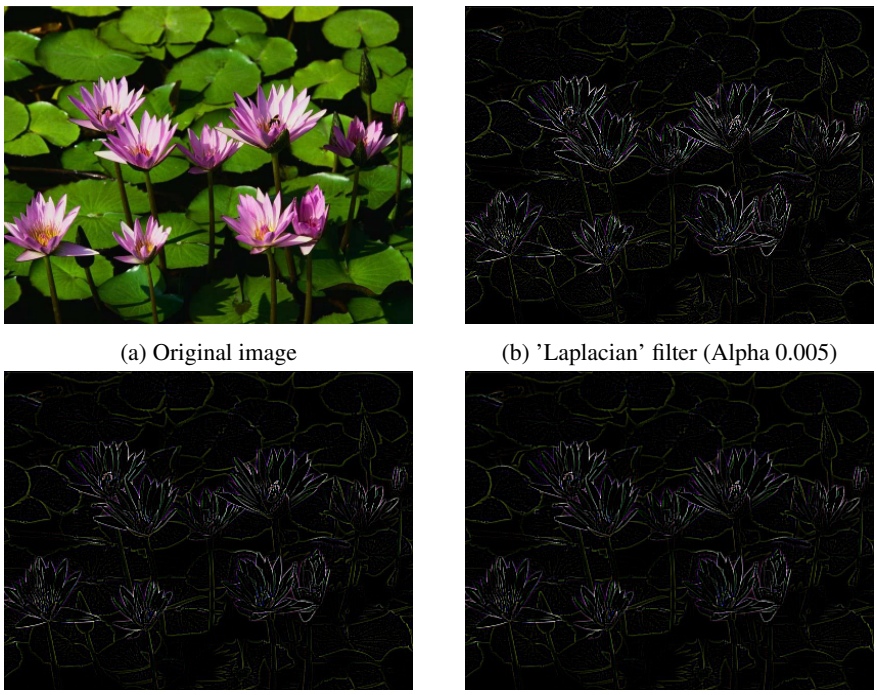
The Sobel, Prewitt and Laplacian filters are used for edge detection. The Prewitt filter has no smoothing, both Sobel and Laplacian have smoothing and the latter allows for the specification of the smoothing level (alpha).

(a) Original image (b) 'Sobel' filter

Figure 3.5: 'Sobel' filter

(a) Original image (b) 'Prewitt' filter

Figure 3.6: 'Prewitt' filter



(a) Original image

(b) 'Laplacian' filter (Alpha 0.005)

(c) 'Laplacian' filter (Alpha 0.6)

(d) 'Laplacian' filter (Alpha 1)

Figure 3.7: 'Laplacian' filter

3.2 Eliminating Punctual Noise

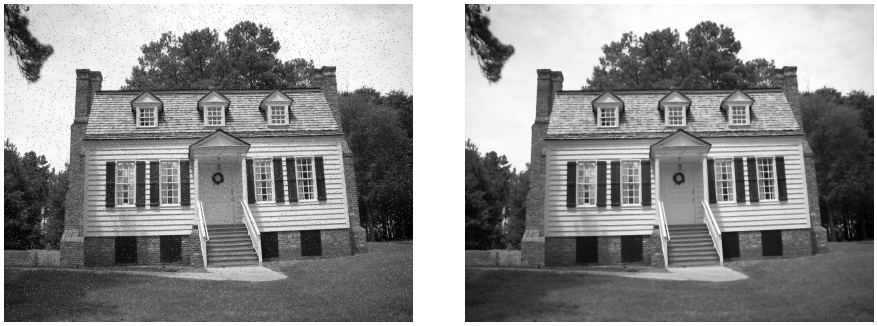
In this section we apply filters with the purpose of removing punctual noise from images. We used the intended images ruido1.jpg and ruido2.jpg, and additionally we used the supplied script to add noise to one of the images, 'nenufares.bmp' and then denoised it aswell.

3.2.1 Median Filter

In this experiment the median filter was used and the results were as follows:

(a) Original image (b) Denoised image using median filter

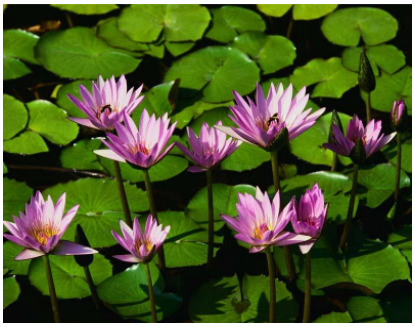
Figure 3.8: Median Filtering 'ruido1.jpg'



(a) Original image (b) Denoised image using median filter

Figure 3.9: Median Filtering 'ruido2.jpg'

We can observe that the Median filter sucessfully eliminated the noise from both supplied images and the modified image.



(a) Original image

(b) Image with added noise

(c) Denoised using 'median' filter (Dimension 3)

Figure 3.10: 'Median' filtering image with added noise

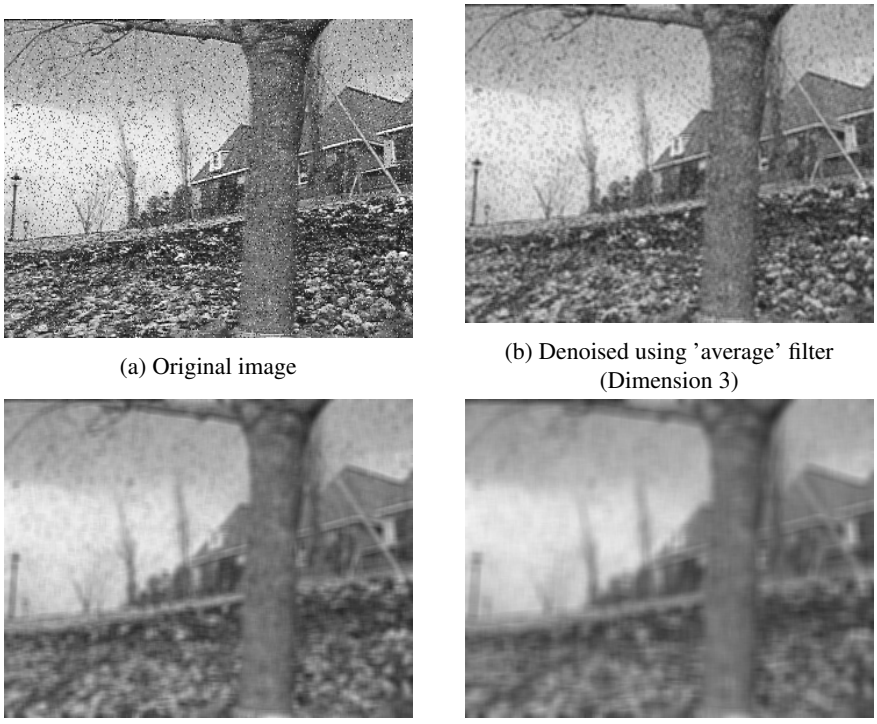
3.2.2 Average and Gaussian Filtering

The following code was added to save the image with added noise.

```
fprintf('Saving image with added noise with format filename_noise_value');
newfilename = append(filename,'_noise_',num2str(r));
path = sprintf('figures/filters/%s.png', newfilename);
imwrite(IR,path);
```

Addiotionally, the following code was written to save the denoised image.

```
fprintf('Saving denoised image with format filename_denoised_median');
newfilename = append(filename,'_denoised_median');
path = sprintf('figures/filters/%s.png', newfilename);
imwrite(Idenoised,path);
```



(a) Original image

(b) Denoised using 'average' filter (Dimension 3)

(c) Denoised using 'average' filter (Dimension 6)

(d) Denoised using 'average' filter (Dimension 9)

Figure 3.11: 'Average' filtering image 'ruido1.jpg'

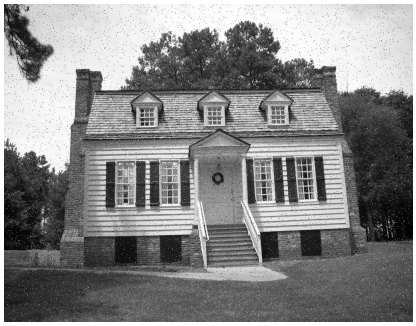
(a) Original image

(b) Denoised using 'gaussian' filter (Dimension 3)

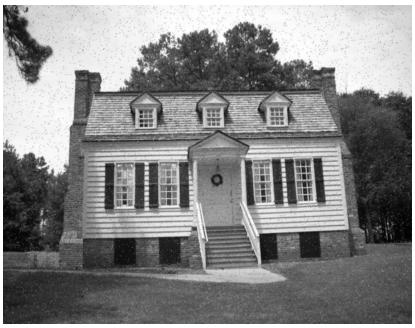
(c) Denoised using 'gaussian' filter (Dimension 6)

(d) Denoised using 'gaussian' filter (Dimension 9)

Figure 3.12: 'Gaussian' filtering image 'ruido1.jpg'



(a) Original image

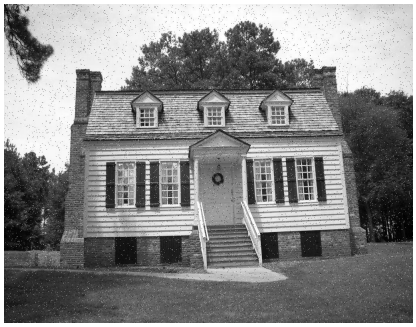


(b) Denoised using 'average' filter (Dimension 3)

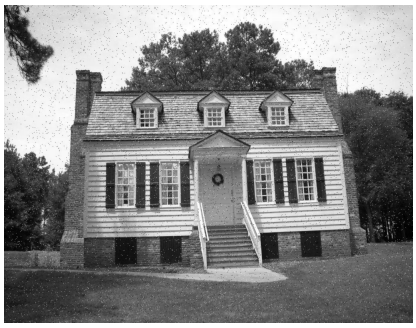
(c) Denoised using 'average' filter (Dimension 6)

(d) Denoised using 'average' filter (Dimension 9)

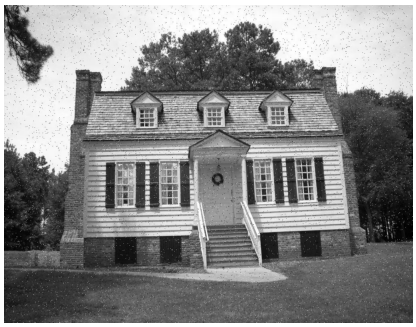
Figure 3.13: 'Average' filtering image 'ruido2.jpg'



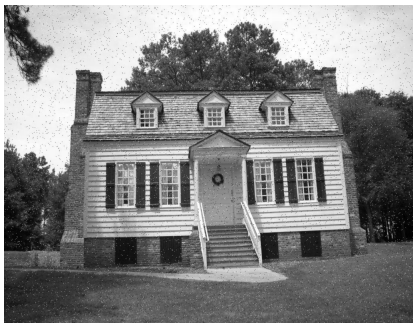
(a) Original image



(b) Denoised using 'gaussian' filter (Dimension 3)



(c) Denoised using 'gaussian' filter (Dimension 6)

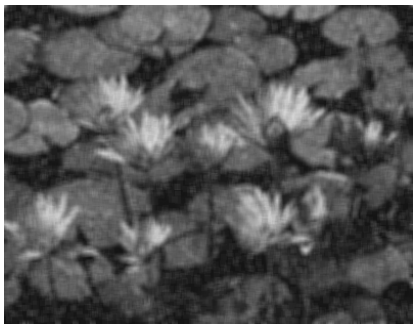


(d) Denoised using 'gaussian' filter (Dimension 9)

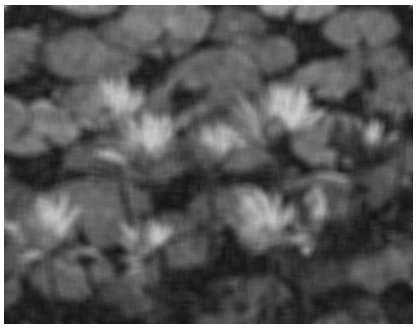
Figure 3.14: 'gaussian' filtering image 'ruido2.jpg'

(a) Image with added noise

(b) Denoised using 'average' filter (Dimension 3)

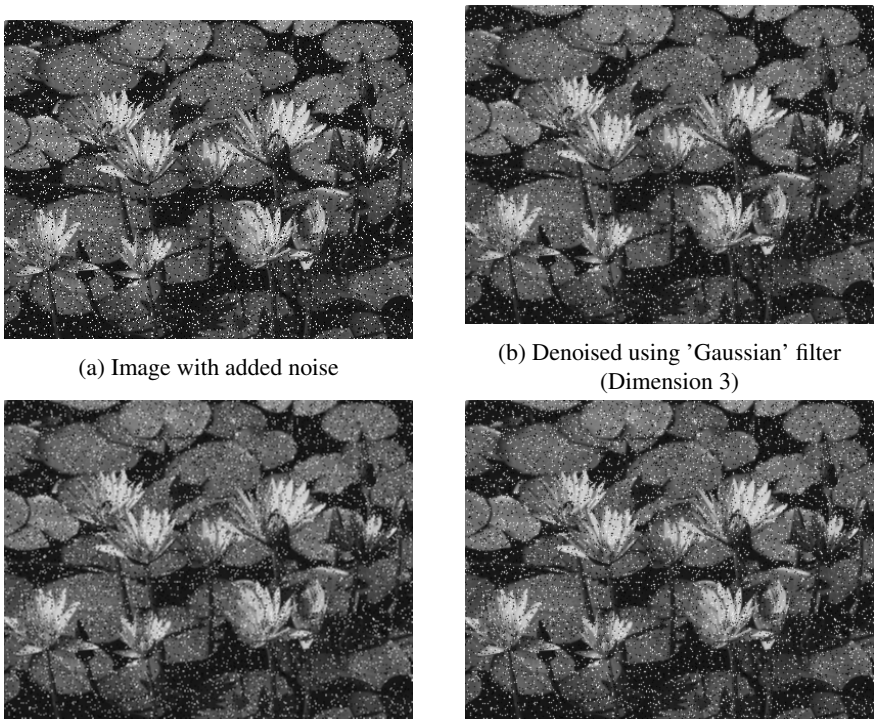


(c) Denoised using 'average' filter (Dimension 6)



(d) Denoised using 'average' filter (Dimension 9)

Figure 3.15: 'Average' filtering image with added noise



(a) Image with added noise

(b) Denoised using 'Gaussian' filter (Dimension 3)

(c) Denoised using 'Gaussian' filter (Dimension 6)

(d) Denoised using 'Gaussian' filter (Dimension 9)

Figure 3.16: 'Gaussian' filtering image with added noise

We can observe that both the Gaussian and Average filters were able to remove some noise from the images, however in the case of the Average filter, it resulted in a very blurry image, and in the case of the Gaussian filter, it removed little noise. The median filters displayed the best performance at eliminating punctual noise while maintaining image quality.