Practicum 2

Computer Vision

University of Barcelona

November 19, 2015

Authors:
Leyva, María
Camps, Julià

Contents

1 Image smoothing and simple geometric operations in Matlab

1.1 Creating images of 3 channels (color images)

We create a first matrix, size 200x200, full with zeros (a black square)

```
A=zeros(200,200);
```

Using the original matrix, we set as 255(white) the columns from 100 to 200 (second half) and create an b&w image

```
M1=A;
M1(1:200,100:200)=255;
im1=mat2gray(M1);
```

Using the original matrix, we set as 255(white) the rows from 100 to 200 (second half) and create an b&w image

```

M2=A;
M2(100:200,1:200)=255;
im2=mat2gray(M2);
```

Using the original matrix, we set as 255(white) the cells from 1x1 to 100x100, the first cuadrant and create an b&w image

```
M3=A;
M3(1:100,1:100)=255;
im3=mat2gray(M3);
```

We create a RGB image from the other three matrixes (Red, Green and Blue)

```

im4 = cat(3, M1, M2, M3);
```

We show all the images

```

subplot(2, 2, 1);
imshow(im1);
subplot(2, 2, 2);
imshow(im2);
subplot(2, 2, 3);
imshow(im3);
subplot(2, 2, 4);
imshow(im4);
```

We write the 4th image to a file called 3channels.jpg

```

imwrite(im4, '3channels.jpg');
```

In the next image we can see the three layers we created (R,G,B) and the resulting image.

1.2 Displaying color images

Read the image sillas.jpg

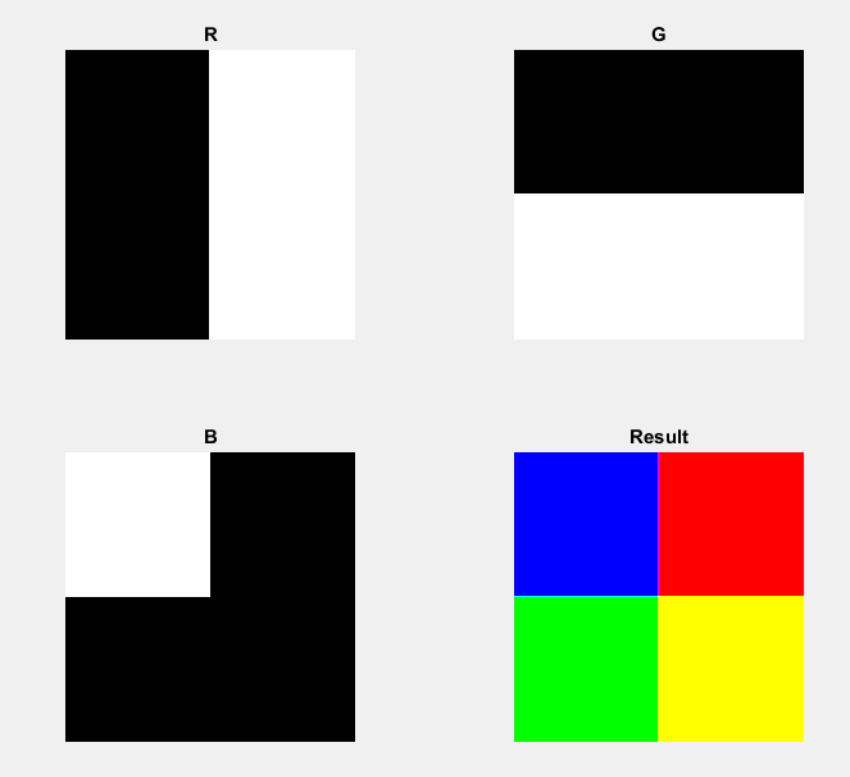

Figure 1: The three layers and the resulting image.

```
figure;
im=imread('images/sillas.jpg');
size(im);
im1=im(10,20); % grey imatge
R=im(:,:,1); % R channel
G=im(:,:,2); % G channel
B=im(:,:,3); % B channel
subplot(2, 3, 2);
imshow(im);
```

As we can see, in the blue and green layers, the chairs appear black (no green nor blue color there), and the red layer appears with white chairs much red color there.

The rest of the image is basically gray so it remains more or less the same in the same images (gray color means having the same red, blue and green level).

```

title('Original image')


subplot(6, 3, 4);
imshow(R);
title('R')

subplot(6, 3, 5);
imshow(G);
title('G')

subplot(6, 3, 6);
imshow(B);
title('B')
```

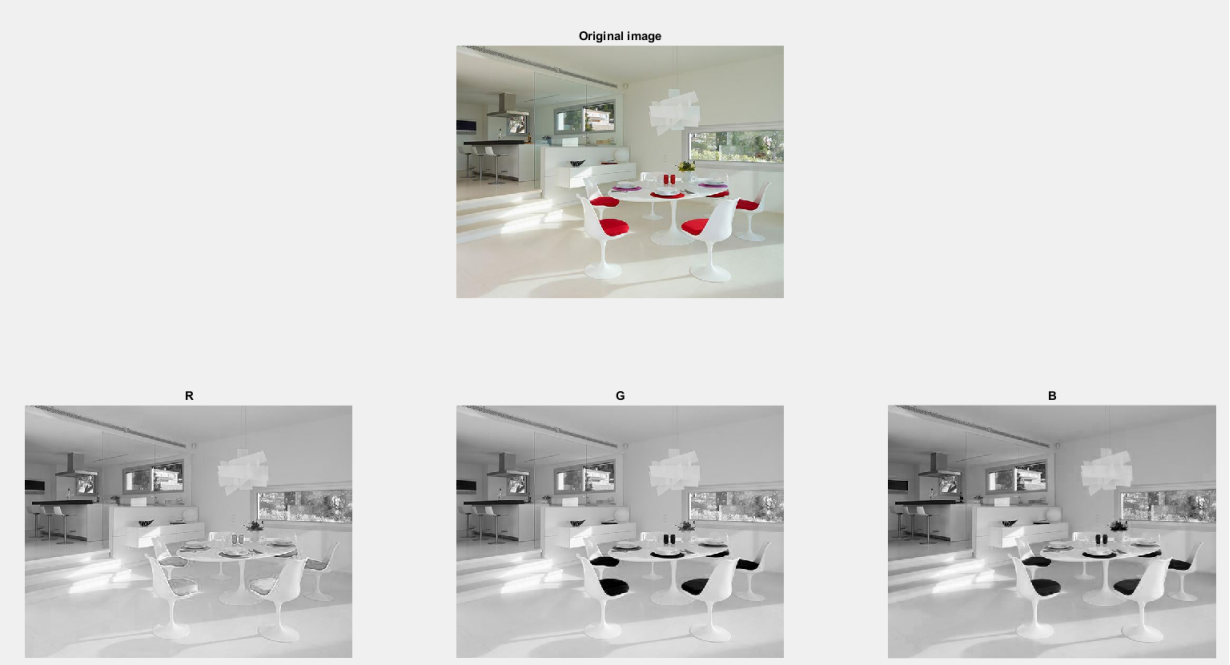

Figure 2: The original image and the R G B layers.

What would happen if we interchange the channels?

As we can see in figure 3, if we change the layers Red and Green the chairs are now green instead of red, therefore the red layer now appears with the chairs black (no red color there), and the green layer appears with white chair (much green color there).

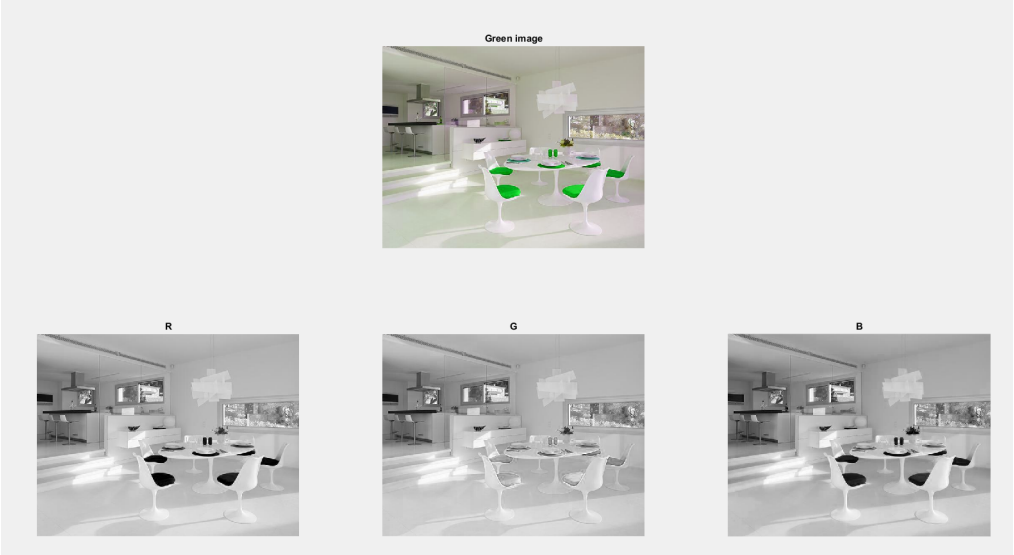

Figure 3: The original image with interchanged layers and the R G B layers.

```
imGreen=im;
 imGreen(:,:,1)=im(:,:,2);
 imGreen(:,:,2)=im(:,:,1);

RimGreen=imGreen(:,:,1); % R channel
GimGreen=imGreen(:,:,2); % G channel
BimGreen=imGreen(:,:,3); % B channel

```

```
subplot(6, 3, 8);
imshow(imGreen);
title('Green image')

subplot(6, 3, 10);
imshow(RimGreen);
title('R')

subplot(6, 3, 11);
imshow(GimGreen);
title('G')

subplot(6, 3, 12);
imshow(BimGreen);
title('B')
```

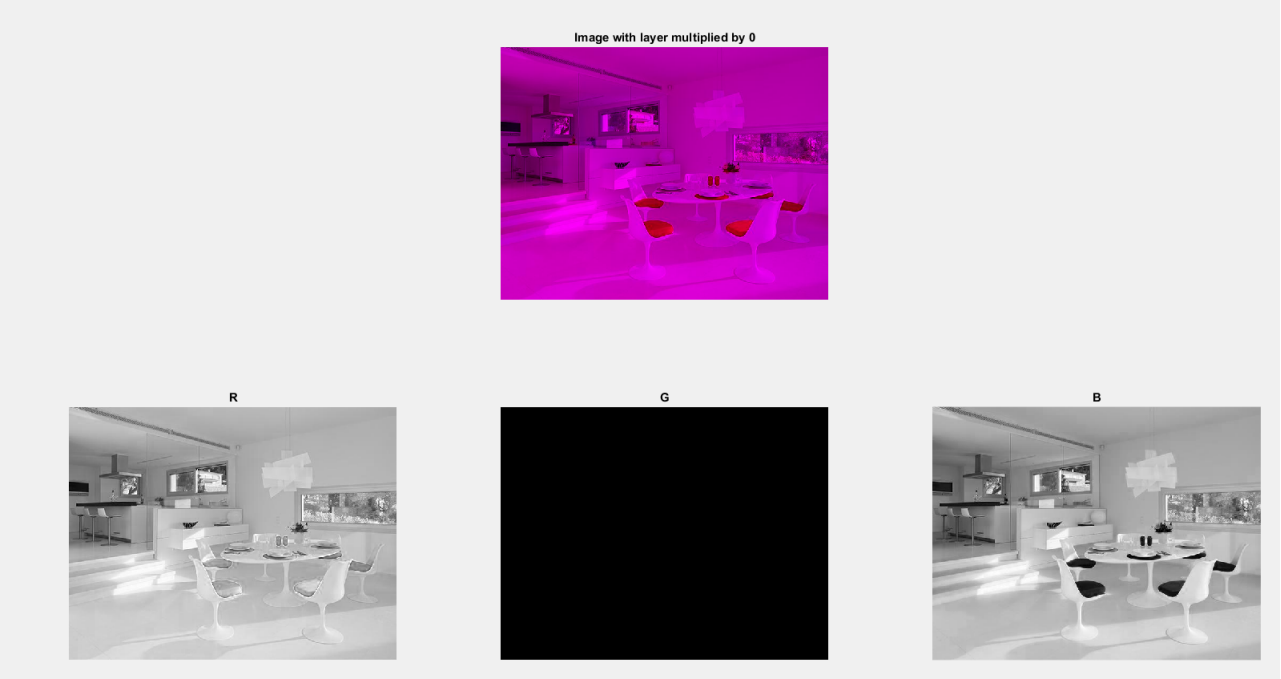

Figure 4: The original image with a layer multiplied by zero and the R G B layers.

What would happen if we multiply one of them by 0? We multiply the Green layer by 0

```
imByZero=im;
imByZero(:,:,2)=im(:,:,2)*0;


RimByZero=imByZero(:,:,1); % R channel
GimByZero=imByZero(:,:,2); % G channel
BimByZero=imByZero(:,:,3); % B channel

subplot(6, 3, 14);
```

As we can see in figure 4, now there is no green in the image therefore the green layer appears totally black and as a result we have a magenta image If we did this with the red layer, the image would be yellow. If we did the same with the blue layer, the image would be cyan.

```
imshow(imByZero);
title('Image with layer multiplied by 0')

```

```
subplot(6, 3, 16);
imshow(RimByZero);
title('R')

subplot(6, 3, 17);
imshow(GimByZero);
title('G')

subplot(6, 3, 18);
imshow(BimByZero);
title('B')
```

1.3 Managing different size and filters

1.3.1 a) Read one of the images in the images.rar.

To read one of the images we have implemented the following matlab code:

```
%Read original image
im_orig = imread(imagename);
```

1.3.2 b) Show how details of the image disappear when rescaling

```
%RESCALE
im_small = imresize(im_orig,0.1);%image resizing to 10 times smaller
figure;%figure1
subplot(1,2,1); imshow(im_orig); title('Original');%will plot on the left ...
    side of figure 2
subplot(1,2,2); imshow(im_small); title('Rescaled');%will plot on the right side
```

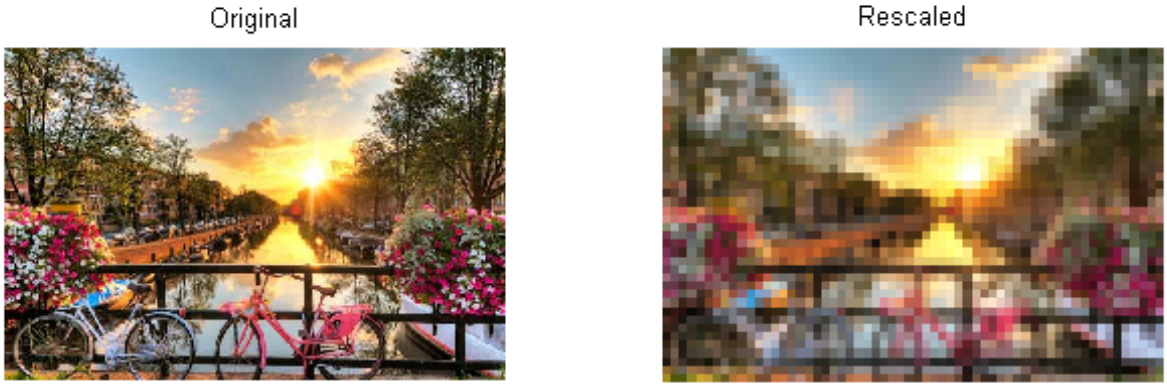

Figure 5: Figure containing the original image reed and the rescaled to smaller size.

In figure 5 we can appreciate that lots of details are lost when rescaling the selected image.

Does the histogram change of both images?

```
%HISTOGRAMS
figure;%figure2
%(histograms only work with one color channel at a time)
%histogram for the original image
subplot(1,2,1); imhist(rgb2gray(im_orig)); title('Original');%will plot on ...
    the left side of figure 2
%histogram for the rescaled image
subplot(1,2,2); imhist(rgb2gray(im_small)); title('Rescaled');%will plot on ...
    the right side
```

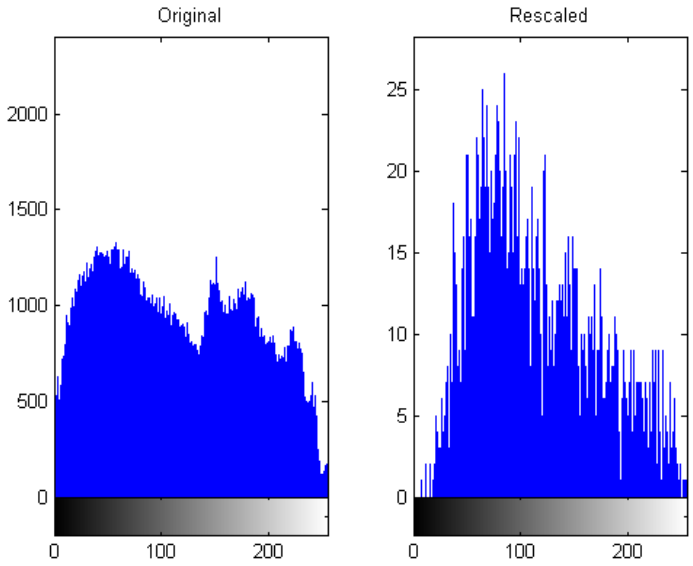

Figure 6: Figure containing the histograms of the original image reed and the resized to smaller size.

In figure 6 we can appreciate that lots of information is lost when rescaling the selected image.

Return back the smaller image to the original size.

```
%RETURN TO ORIGINAL SIZE
%Return back the smaller image to the original size. Compare to the original ...
    one.
im_small_re = imresize(im_small,10);
figure;%figure3
subplot(1,2,1); imshow(im_orig); title('Original');%will plot on the left ...
    side of figure 2
subplot(1,2,2); imshow(im_small_re); title('Restored size');%will plot on ...
    the right side
%Whe can see that the pixel density is better than in the without returning
%to the original size image, but, the information that was lost when we
%rescaled first time hasn't been recovered with simply returning to the
%original size.
```

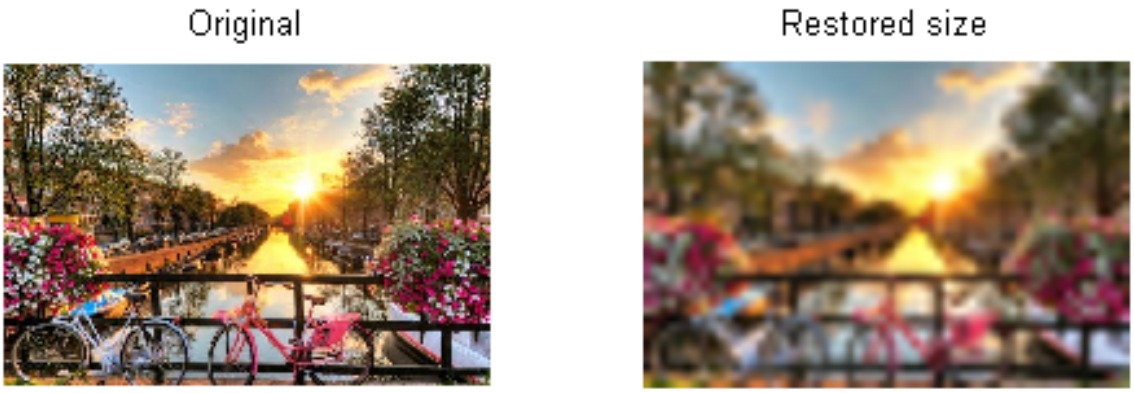

Figure 7: Figure containing the original image reed and the resized to smaller size now restored to its original size.

In figure 7 we can appreciate that the lost information is not recovered after restoring the image to its original size.

Compare to the original one

```
%HISTOGRAMS
figure; subplot(1,3,1); imhist(rgb2gray(im_orig)); title('Original Image ...
    Histogram');
subplot(1,3,2); imhist(rgb2gray(im_small)); title('Small image Histogram');
subplot(1,3,3); imhist(rgb2gray(im_small_re)); title('Small image returned ...
    histogram');
```

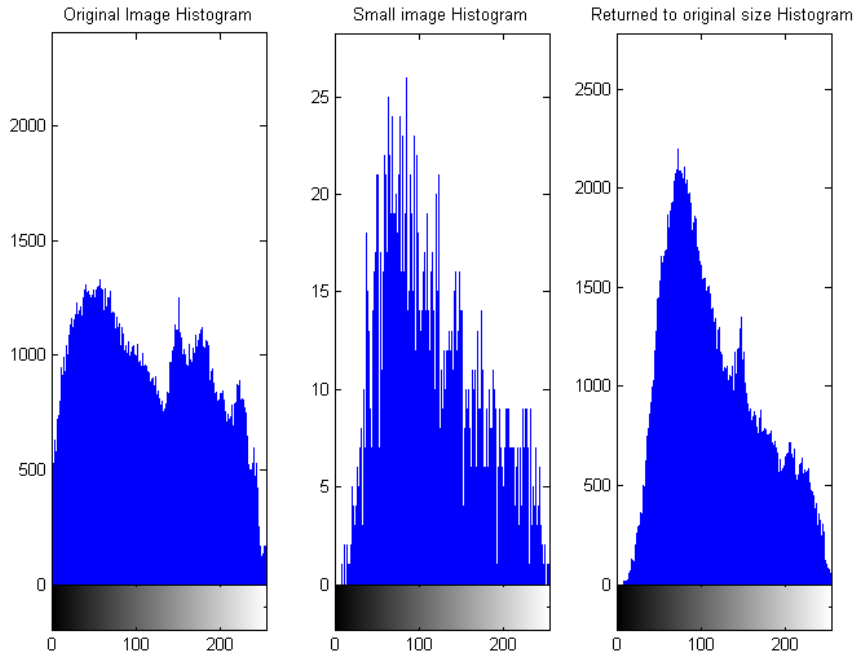

Figure 8: Histograms of the original image, the rescaled and the resized to smaller size now restored to its original size.

In figure 8 we can appreciate that as we already said when restoring the image, the final histogram is explaining that after rescaling the size to a smaller size, some information is lost, and can't be recovered by the inverse procedure. We just can get a smooth on the information gaps.

1.3.3 As an alternative of removing image details, apply different smoothing filters

For this exercise we have done the following Matlab code:

```
figure;
subplot(3,3,1); imshow(im_orig); title('Original');
mask=[1 1 1 1 1 1 1 1 1 1]/10;%vector mask
subplot(3,3,2); imshow(mask); title('Mask: [1 1 1 1 1 1 1 1 1 1]/10');
hsize = 10; sigma = 3;
filter = fspecial('gaussian',hsize,sigma);%gaussian filter
subplot(3,3,3); imagesc(filter); axis equal tight; title('Gausian Filter');
%first iteration
img_smooth_mask=imfilter(im_orig,mask);%vector mask filtering
img_smooth_gausian = imfilter(im_orig, filter);%gausian filering
subplot(3,3,4); imshow(img_smooth_mask); title('1 M. Smooth');
subplot(3,3,7); imshow(img_smooth_gausian); title('1 G. Smooth');
for i=1 : 2
    img_smooth_mask=imfilter(img_smooth_mask,mask);%vector mask filtering
    img_smooth_gausian = imfilter(img_smooth_gausian, filter);%gausian filering
    subplot(3,3,i+4); imshow(img_smooth_mask); title(strcat(num2str(i+1),' ...
        M. Smooth'));
    subplot(3,3,i+7); imshow(img_smooth_gausian); ...
        title(strcat(num2str(i+1),' G. Smooth'));
end
```

The previous code is applying the required filters several times to an image in order to be able to see how the smooth is increasing, but in a different way, depending on the filter applied. This can appreciate this in figure 9, that corresponds to the plot of the previous code shown.

Figure 9: Figure containing an RGB image with a Gaussian filter applied on it.

Discuss how the size of the filter or mask affect the final outcome. *As bigger is the size of a mask from more far a pixel will be affected by other pixels values when we apply the smoothing, and this one will affect at the same time to more pixels. (Example: So if we have a single pixel was black on a white image, and we have a huge mask, it will nearly disappear, leaving behind him a dim grey zone, because all other pixels are affecting this one to resemble more to the others.)*

(Conclusion:) *Therefore, the resulting image of a smoothing, will be more homogeneous, in the direction that we are applying the filter, as larger is the mask that we use for filtering.*

Can you apply the filter on the rgb image? *Yes you can. (Demostration:)*

```
1      hsize = 10; sigma = 3;
2      filter = fspecial('gaussian',hsize,sigma);%gaussian filter
3      img_smooth_gausian = imfilter(im_orig, filter);%gausian filering
4      figure; imshow(img_smooth_gausian); title('RGB Image Smoothed with Gausian ...
           Filter');
```

Figure 10: Figure containing an RGB image with a Gaussian filter applied on it.

In figure 10 we have an example of applying a filter to a RGB image.

What type should be the image before convolving and why? *Should be a 2D image, because the conv2 is for 2D Matrixes.*

What dimensions should be the mask? *Should be at least a 1D mask, in orther to have horitzontal or vertical mask smooting (the direction of the smoothing depends on the direction on that the mask is moving when the mask has more than one dimension, but for 1D masks, the shape of it is determining the direction of the smoothing, because it can only take influence in the direction of the mask orientation). Or it can be a matrix based mask,2D then, like Gaussian masks.*

What is the difference using the following three masks:

- **a) [1 1 1 1 1]:** *It's only Horitzontal Smoothing mask*
- **b) [1;1;1;1;1]:** *It's only Vertical Smoothing mask*
- **c) [[1 1 1 1 1];[1 1 1 1 1];[1 1 1 1 1];[1 1 1 1 1];[1 1 1 1 1]]:** *It's using information from horizontal and vertical neighbours to smooth the image.*

Do we need to normalize the mask (divide by the sum of all its numbers)? *Yes, because otherwise we may obtain values that are out of range (more than 255).*

1.3.4 Note 1: Repeat these experiments with a couple of other images

For doing this Note1 we implemented the following short matlab function:

```
ej_13_function('amsterdam.jpg');
%Note1 & Note2)
ej_13_function('corals.jpg');
ej_13_function('dog.jpg');
```

1.3.5 Note 2: You can subtract the original and smoothed images in order to illustrate the difference between them

For doing this Note2 we implemented the following short matlab function:

```
figure;
subplot(1,3,1); imshow(im_orig); title('Original Image');
subplot(1,3,2); imshow(img_smooth_gausian); title('Gaussian Smoothed Image');
subplot(1,3,3); imshow(imsubtract(im_orig, img_smooth_gausian)); ...
    title('Substraction from both images');
```

Figure 11: Figure containing the original image, one with a Gaussian filter applied some times on it and the substraction from this first two images.

In figure 11 we can appreciate the differences between the original image and the resulting one after applying some times the Gaussian filter to the first one.

1.4 Simple geometric operations

We want to swap the two halves of a given image, selecting the swapping point using an interface tool (we have implemented this additional part in order to be able to try different cut points for the swapping). In order to achieve this, we implemented the following matlab function.

```
function  ej_14( original_image )
%ej_14 this function swaps the two halves of a given image
%   First we read the image
im_orig=imread(original_image);

%We select the cutting point manually (to ensure its well done even if its
%not in the middle of the original image
[im_sel,rect] =imcrop(im_orig);

%Now we check if the previous selected part is the first part or the second
%one, and depending on this we rejoin both images to create an inverted
%parts one
if (rect(1) > 1) && (rect(1) > 1) %The selection was taken from the first part ...
    of the image
    %We fix the selection to ensure no problems happen
    im_sel=im_orig(1:size(im_orig,1),size(im_sel,2):size(im_orig,2),:);
    %We obtain the unselected part of the image
    im_unsel=im_orig(1:size(im_orig,1),1:size(im_sel,2),:);
    %We invert the order of the both parts
    Res = cat(2,im_sel,im_unsel);
else %The selection was taken from the second part of the image
    %We fix the selection to ensure no problems happen
    im_sel=im_orig(1:size(im_orig,1),1:size(im_sel,2),:);
    %We obtain the unselected part of the image
    im_unsel=im_orig(1:size(im_orig,1),size(im_sel,2):size(im_orig,2),:);
    %We invert the order of the both parts
    Res = cat(2,im_unsel,im_sel);
end
%We display the original and the new image
figure; imshow(Res); %figure1
%Original vs Inverted
figure; %figure2
subplot(1,2,1);imshow(im_orig); title('Original');
subplot(1,2,2); imshow(Res); title('Inverted');

end
```

We can see the results with different images.

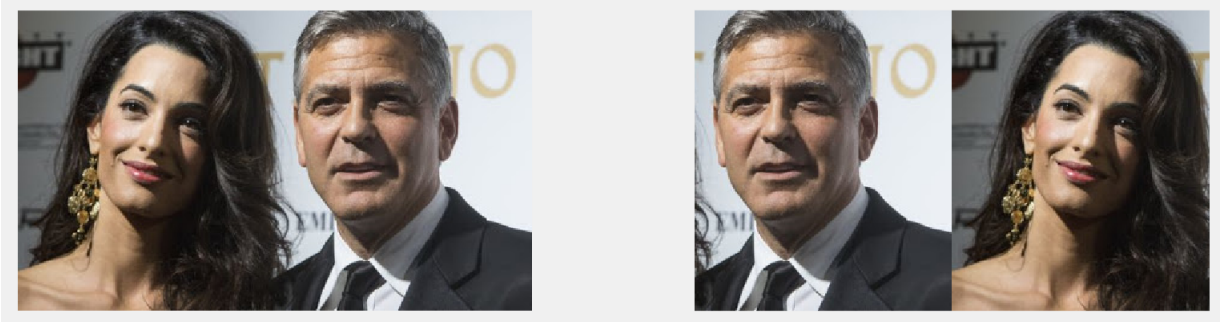

Figure 12: Swapped clooney.jpg.

Figure 13: Swapped fabulous.jpg.

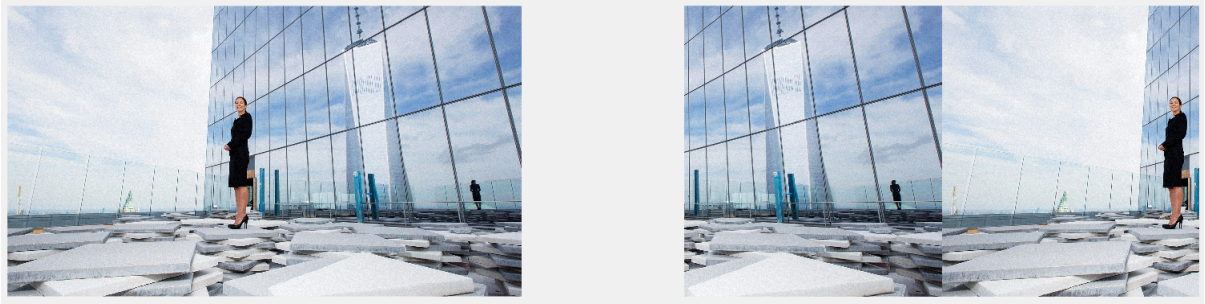

Figure 14: Swapped towerWoman.jpg.

Figure 15: Swapped house.jpg.

1.5 Image binarization

For this function we have implemented the function thresholdImg, which creates the binary version of a given image, using a given threshold. The threshold function is the following:

$$B_i(x,y) = \begin{cases} 0, & \forall (x,y) : I(x,y) < th \\ 1, & \forall (x,y) : I(x,y) \geq th \end{cases}$$

```
function im = thresholdImg( threshold )
%THRESHOLDING This function creates a binary image of car_gray.jpg, given a
%threshold
im_orig=imread('images/car_gray.jpg');
im=im2bw(im_orig, threshold/255);
end
```

We create and display the binary image with different thresholds: 20, 30, 150, 255.

```
figure;
subplot(2,2,1);
imshow(thresholdImg(20));
title('Threshold 20');
subplot(2,2,2);
imshow(thresholdImg(30));
title('Threshold 30');
subplot(2,2,3);
im_150=thresholdImg(150);
imshow(im_150);
title('Threshold 150');
subplot(2,2,4);
imshow(thresholdImg(255));
title('Threshold 255');
```

As we can see, when we have a lower threshold we have an whiter image (we're ignoring all the greys lighter than the threshold), and it get's darker as we increase it. With threshold=255 we get a completely black image, since we don't ignore any pixel.

We store the image with threshold=150 as car_binary.jpg.

```

imwrite(im_150, 'car_binary.jpg');
```

We multiply the original image by the binary image (th=150) and display the original, the binary and the result.

```

orig=imread('images/car_gray.jpg');
mult=immultiply(orig,im_150);
figure;
subplot(1,3,1);
imshow(orig);
title('Original');

subplot(1,3,2);
imshow(im_150);
title('Threshold 150');

subplot(1,3,3);
imshow(mult);
title('Binary * Original');
```

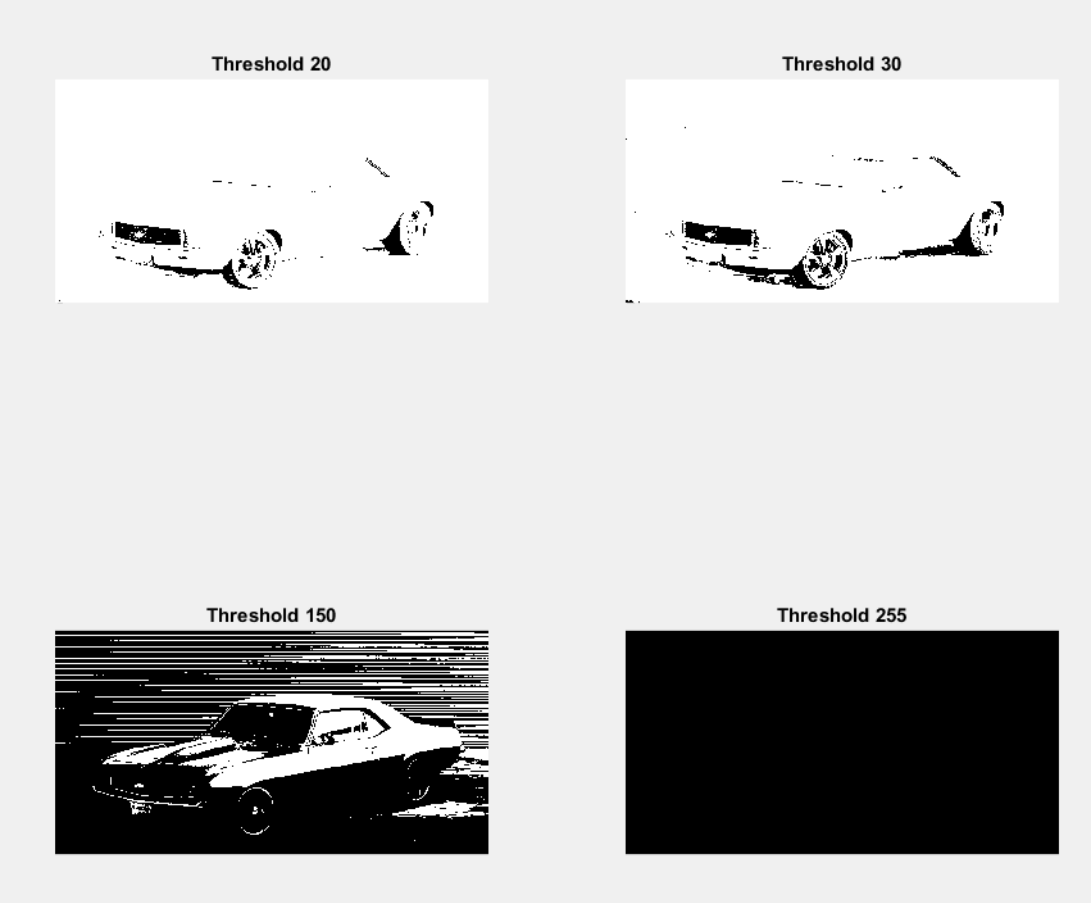

Figure 16: Binarization with different thresholds.

As seen in the picture, when we multiply the binary image by the original one, we get the same image but with the pixels that were black in the binary image also turned black.

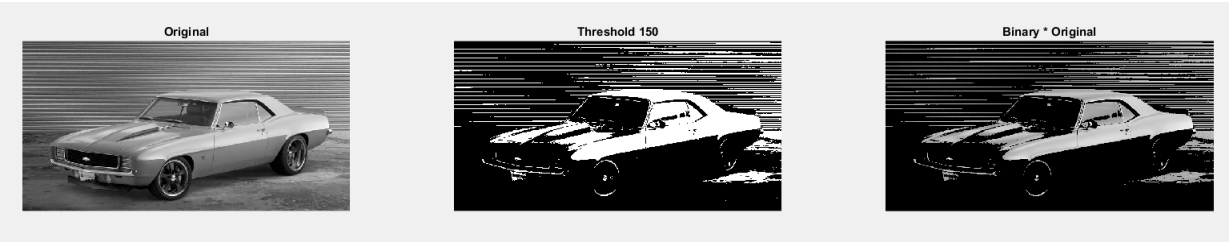

Figure 17: Original, binary and result of the multiplication.

We multiply the the original image by the inverse of the binary image (th=150) and display the original, the inverse and the result.

```
1
2  inv=imcomplement(im_150);
3  multinv=immultiply(orig,inv);
4  figure;
5  subplot(1,3,1);
6  imshow(orig);
7  title('Original');
8
9  subplot(1,3,2);
10 imshow(inv);
11 title('Threshold 150 Inverse');
```

```
12
13 subplot(1,3,3);
14 imshow(multinv);
15 title('Binary Inverse * Original');
```

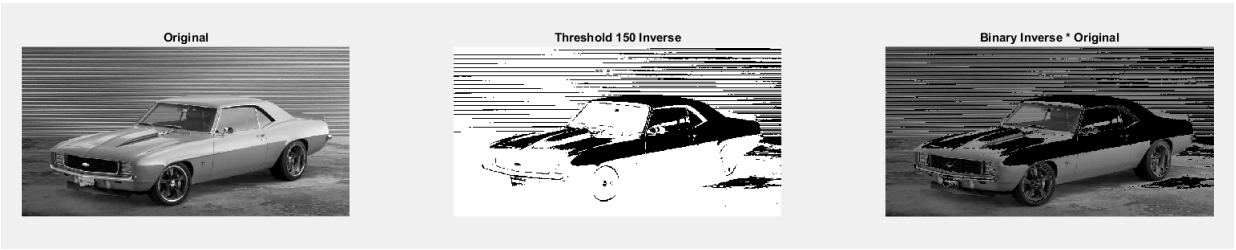

Figure 18: Original, inverse of the binary and result of the multiplication.

As seen in the picture, the inverse of the binary is the same but with inverted colors. When we multiply the inverse binary image by the original one, we get the same image but with the pixels that were black in the inverse binary image also turned black.

1.6 Treating color images

For this exercise we implemented the FuseImg function.

```
1  function fuseImg()
2  %FUSEIMG Summary of this function goes here
3  %   Detailed explanation goes here
4  original_mapfre=imread('images/mapfre.jpg');
5
6  % a)    Open    hand.jpg    and convert it  in  gray    scale   image.
7  original_hand=imread('images/hand.jpg');
8  gray_hand=rgb2gray(original_hand);
9
10 % b)     Perform     a   binarization    to  obtain  a   binary  image   of  2   ...
       regions:    the     hand    (called
11 % foreground)    and     the     rest    (called     background).    Create  ...
       the     inverse     binary  image
12 % changing  the areas   of  foreground  and background.
13 binary_hand=im2bw(gray_hand, 0.099);
14 inv_binary_hand=imcomplement(binary_hand);
15
16 % c)    use  the    binary  matrices    created in  b)   to merge    the    ...
       images  hand    and mapfre  (Fig. 3(c))
17 foreground_mask = cat(3,binary_hand,binary_hand,binary_hand);
18 background_mask = cat(3,inv_binary_hand,inv_binary_hand,inv_binary_hand);
19
20 background=immultiply(original_mapfre,background_mask);
21 foreground=immultiply(original_hand,foreground_mask);
22
23 figure;
24 subplot(1,3,1);
25 imshow(background);
26
27 subplot(1,3,2);
28 imshow(foreground);
29
30 hand_mapfre=background+foreground;
31
32 subplot(1,3,3);
33 imshow(hand_mapfre);
34 % e)    Save    the image   as  hand_mapfre_3C.jpg.
```

```
imwrite(hand_mapfre,'hand_mapfre_3C.jpg');
end
```

We display the obtained results in the followig picture:

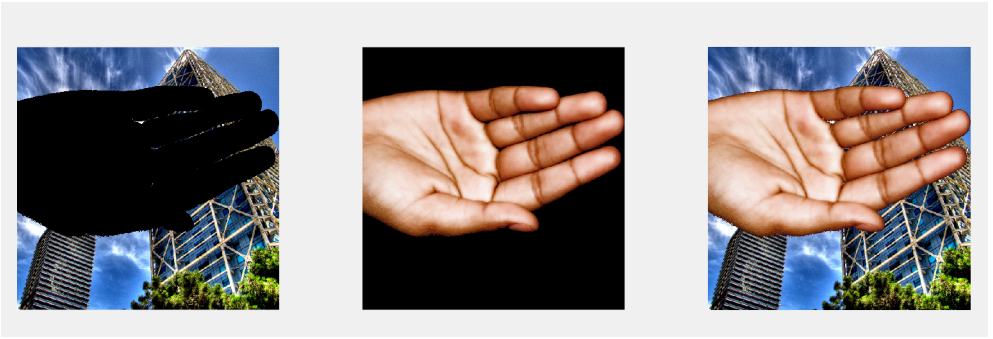

Figure 19: Background, foreground, and result

2 Edges and contours

2.1 Hybrid images construction

2.1.1 Given the images Einstein.jpg and Monro.jpg, build a hybrid image from them

For solving this exercise we implemented this matlab code:

```
    im1=imread(im1_name);
    im2=imread(im2_name);

    hsize = 30; sigma = 9; sigma_high=25;
    filter_low = fspecial('gaussian',hsize,sigma);%gaussian filter
    filter_high = fspecial('gaussian',hsize,sigma_high);%gaussian filter
    im1_low = imfilter(im1, filter_low);%obtain the lowpass filtered
    im2_high= im2- imfilter(im2, filter_high);%obtain the highpass filtered
    im_hybrid = imadd(im1_low,im2_high); %generate the hybrid image

    %plot all
    figure;
    subplot(3,3,1); imshow(im1);
    subplot(3,3,2); imshow(im2);
    for i=3:1:9
        subplot(3,3,i); imshow(im_hybrid);
        im_hybrid((size(im_hybrid,1)+ceil(size(im_hybrid,1)/5)),
        (size(im_hybrid,2)+ceil(size(im_hybrid,2)/5)))=0;
    end
```

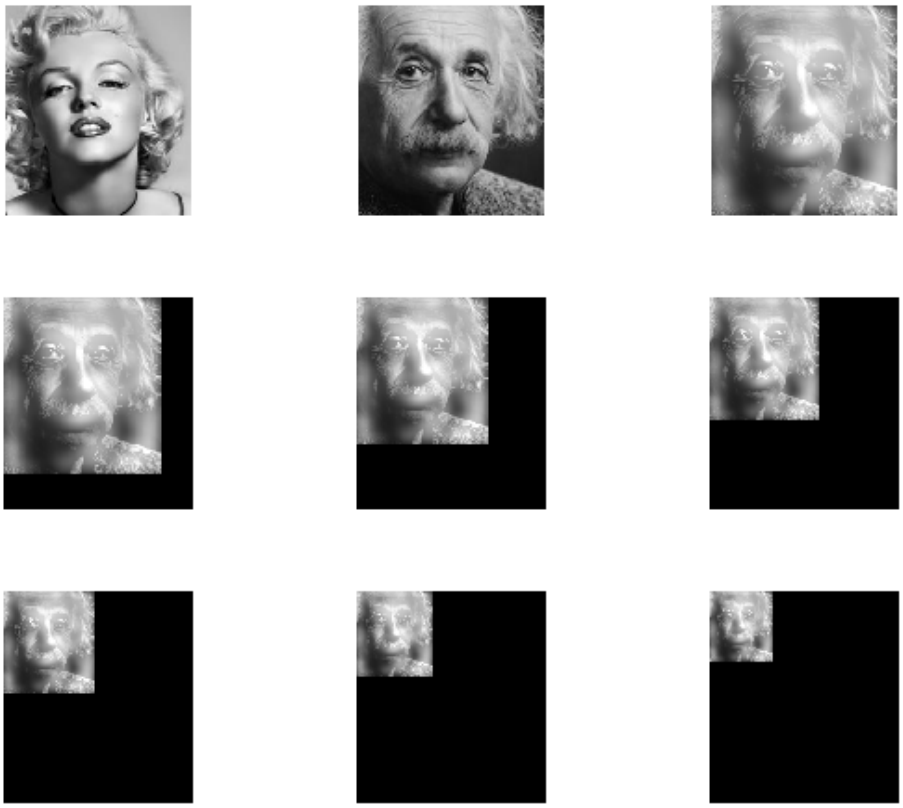

Figure 20: Figure containing many times the hybrid image from Einstein and Monro, decreasing size.

In figure 20 we can appreciate that as we just display the hybrid image smaller, the silhouette of Monro gains definition and ends by taking the principal role on the image, when at the beginning we had that it was a pretty clear photo of Einstein. This is due that we added both images, but just the filtered lowpass of Monro.jpg but the highpass filtered of Einstein.jpg.

2.2 Determine the optimal edges

We have used three different methods (Sobel, Canny and Prewitt) in order to obtain the edges of different images.

As we can see, with this image the method that seems to work best is Canny, as it detects almost all edges in a very precise way.

This image was a bit more complicated, it didn't have clear contours. Again, Canny seems to work fine, and Prewitt does not work at all.

Testing the function with another quite difficult image (a white dog laying in an almost white floor), we conclude again that the best working method is Canny. Prewitt keeps not working at all.

At last, we tried with buffet.jpg, and again, best method is Canny and the worst is Prewitt.

Figure 21: Edges in starbuck.jpg.

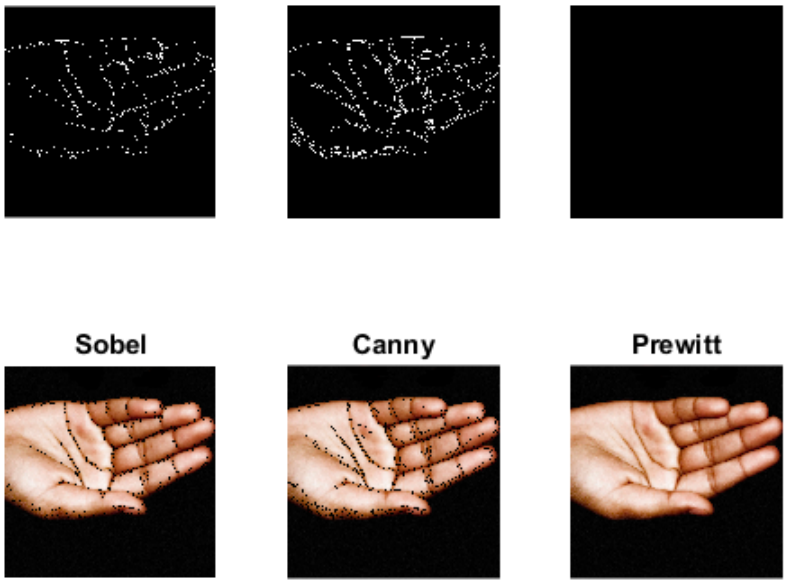

Figure 22: Edges in hand.jpg.

2.3 Enhancing images with edges

2.3.1 Enhance the edges and overlap them on the video Maldives.mp4

In order to enhance the edges we implemented the following code:

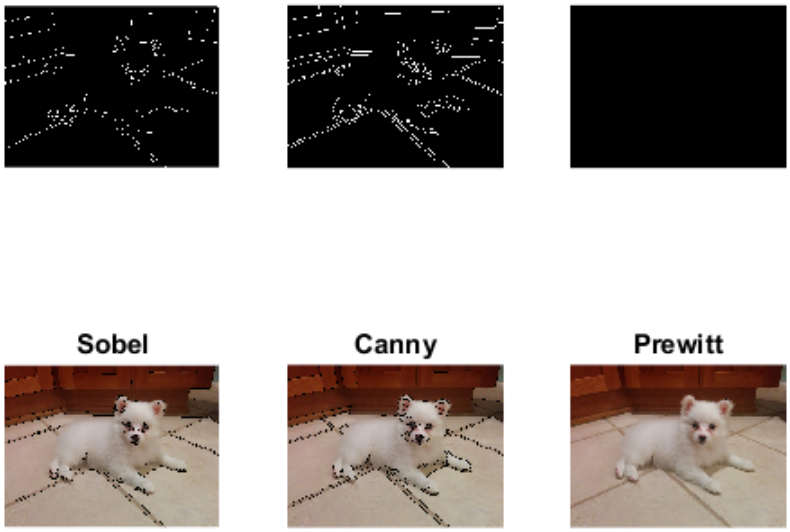

Figure 23: Edges in dog.jpg.

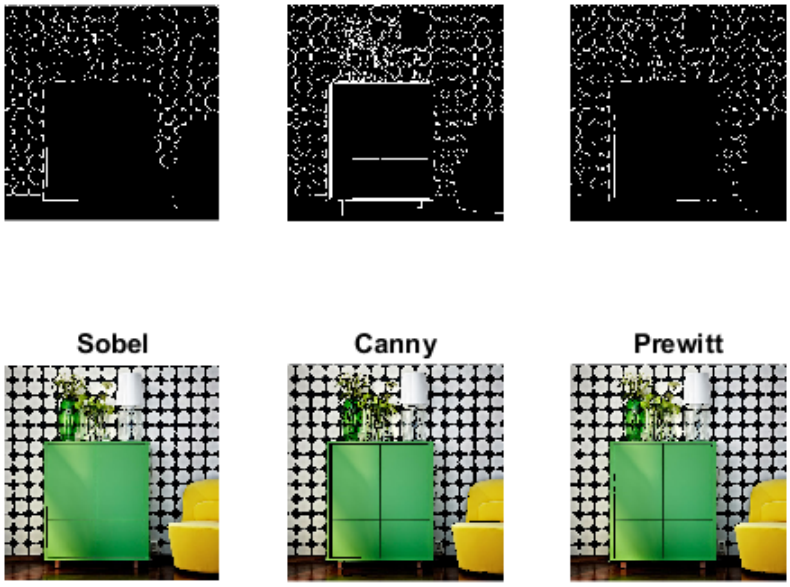

Figure 24: Edges in buffet.jpg.

```
function ej_23( filename )
    videoFReader = vision.VideoFileReader(filename);
    videoPlayer1 = vision.VideoPlayer('Name','Sobel');
    videoPlayer11 = vision.VideoPlayer('Name','Sobel');
    videoPlayer2 = vision.VideoPlayer('Name','Canny');
```

```
    videoPlayer22 = vision.VideoPlayer('Name','Canny');
    videoPlayer3 = vision.VideoPlayer('Name','Perwitt');
    videoPlayer33 = vision.VideoPlayer('Name','Perwitt');
    while ¬isDone(videoFReader)
        videoFrame = step(videoFReader);
        I = rgb2gray(videoFrame);
        BW1 = edge(I,'sobel',0.063);
        BW2 = edge(I,'canny',0.31,0.95);
        BW3 = edge(I,'prewitt',0.065);

        edges_green1 = cat(3, BW1*0, BW1, BW1*0);
        edges_green2 = cat(3, BW2*0, BW2, BW2*0);
        edges_green3 = cat(3, BW3*0, BW3, BW3*0);

        videoFrame1 = videoFrame+edges_green1;
        videoFrame2 = videoFrame+edges_green2;
        videoFrame3 = videoFrame+edges_green3;
        videoFrame11 = edges_green1;
        videoFrame22 = edges_green2;
        videoFrame33 = edges_green3;

        step(videoPlayer1, videoFrame1);
        step(videoPlayer2, videoFrame2);
        step(videoPlayer3, videoFrame3);
        step(videoPlayer11, videoFrame11);
        step(videoPlayer22, videoFrame22);
        step(videoPlayer33, videoFrame33);
    end
    release(videoPlayer1);
    release(videoPlayer2);
    release(videoPlayer3);
    release(videoPlayer11);
    release(videoPlayer22);
    release(videoPlayer33);
    release(videoFReader);
end
```

This code displays a window for each of the enhancing methods tested in exercise 2.2 as we can see on figure 25.

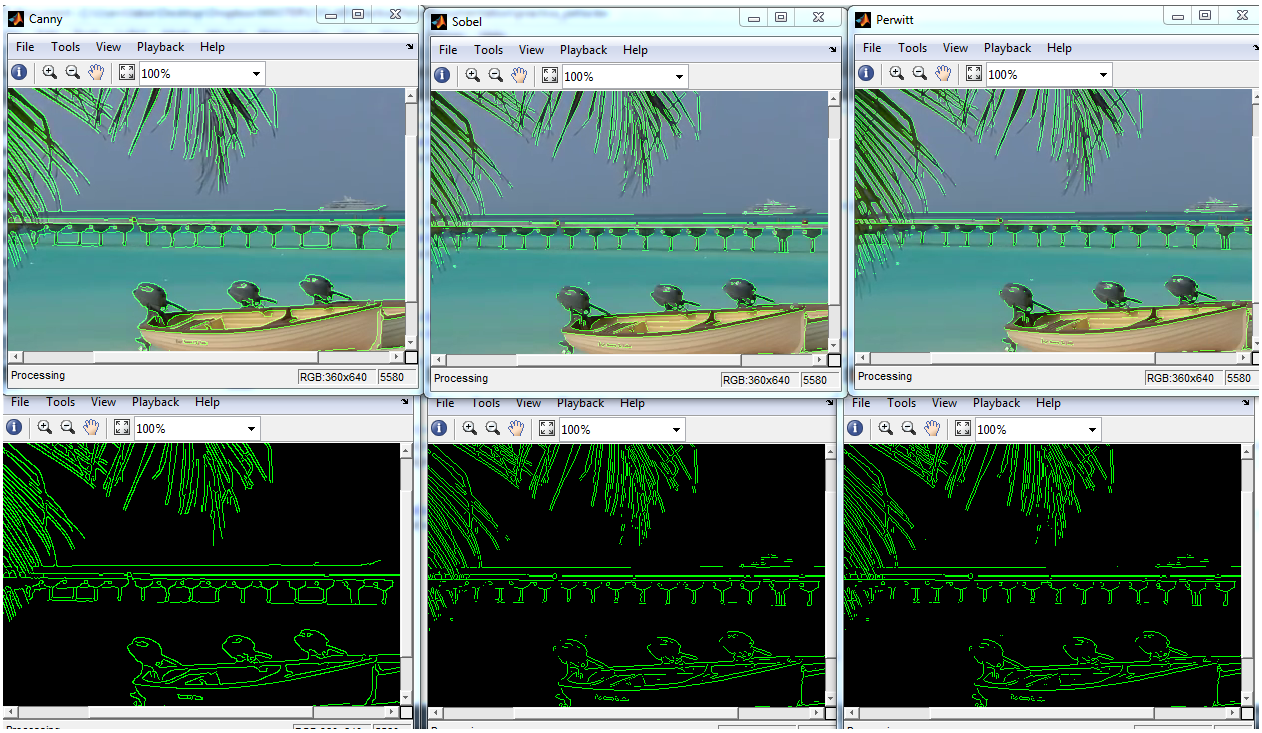

Figure 25: Figure the result of executing the previous code, were we can find two windows of display for each of the methods tested in this exercise for the edge detection on video. One for the edges detected, in green, and one for the edges overlapped on the video frames.

2.3.2 Discuss which is the best way and parameters to obtain the edges

After some testing we decided that the best attributes for the thresholds where:

- Sobel: 0.063
- Canny: 0.31, 0.95
- Prewitt: 0.065

We just adjusted them using the first frames of the video, like the one shown in figure 26, because it was better for replaying several times. And also because it had elements that we considered interesting. We adjusted the parameters of threshold in order to detect the edges only of the houses and the sea border, but to avoid detecting edges in waves or clouds.

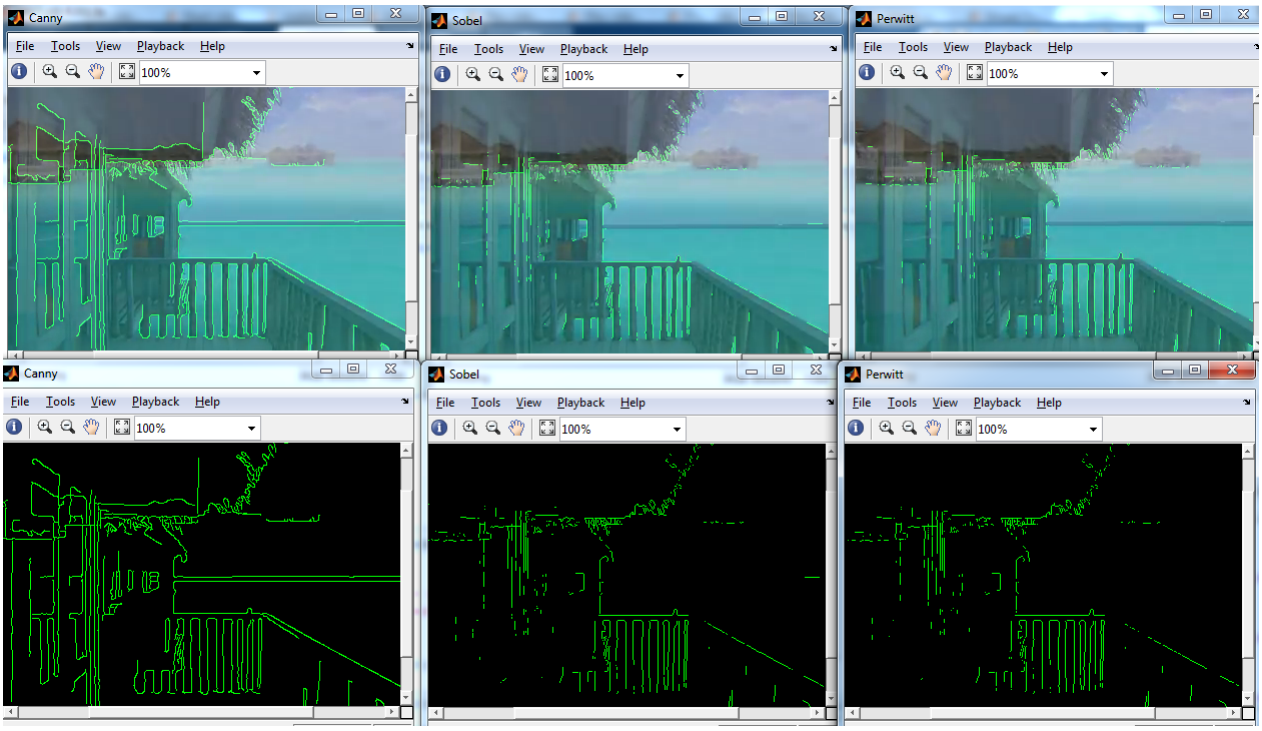

Figure 26: Figure showing one of the first frames of the video, used for defining the parameters.

From this three methods, we considered that canny was detecting better the edges, more clear and precise than others, that were detecting discontinuous edges with interferences, this can be appreciated on figure 27. But on the other hand, canny method was changing allot the edges detected on each single frame, when others tended to give a more stable output. So even that we considered canny, with the given parameters, to be the best method to solve this detection edges exercise, we are aware that it depends on what is considered more important when evaluating a solution, because each of them has different drawbacks, and none is perfect when the frames are changing continuously.

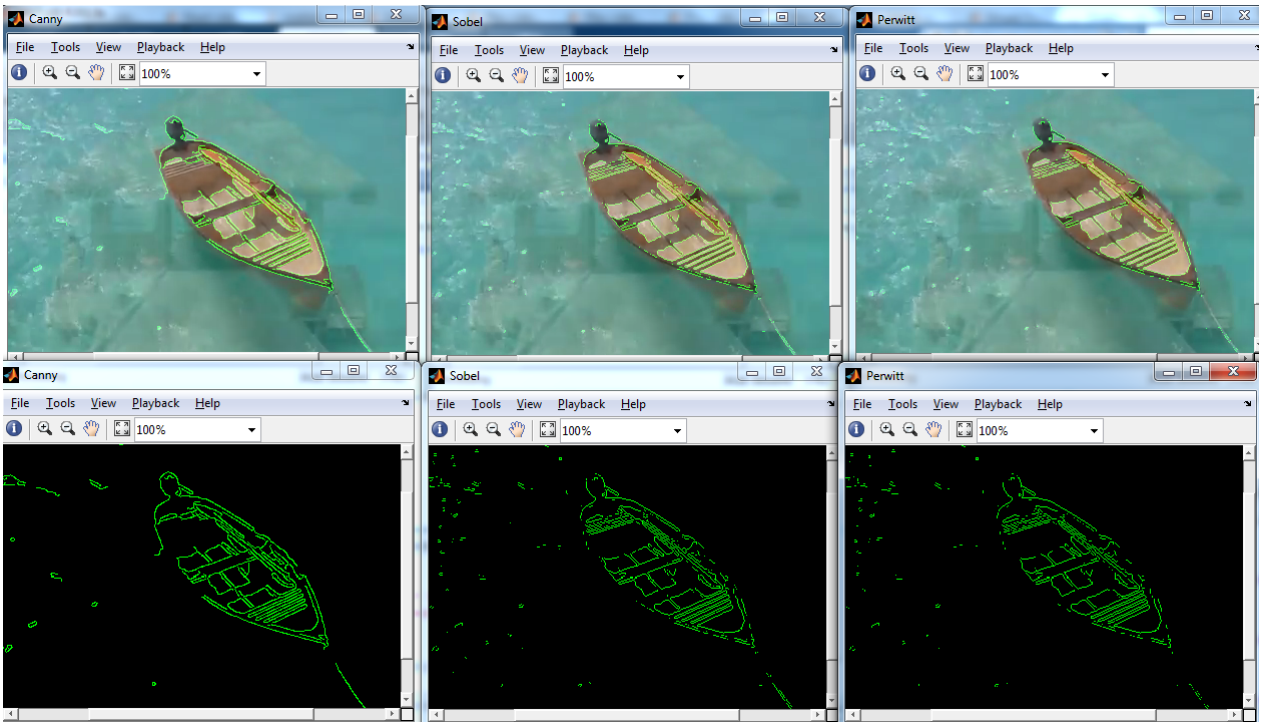

Figure 27: Figure showing one of the first frames of the video, used for defining the parameters.

2.3.3 What advantages and disadvantages do you see on the obtained effect?

The advantages is that we are able to define object contours in video images. So we are enhancing the important information for the image treatment if we are intending to do movement recognition. But the problem is that we are generalizing the threshold for the whole video, and different frames may need different thresholds.

In fact we found that canny was behaving very well in general but at some point missed important edges as we can see in the figure 28 bellow.

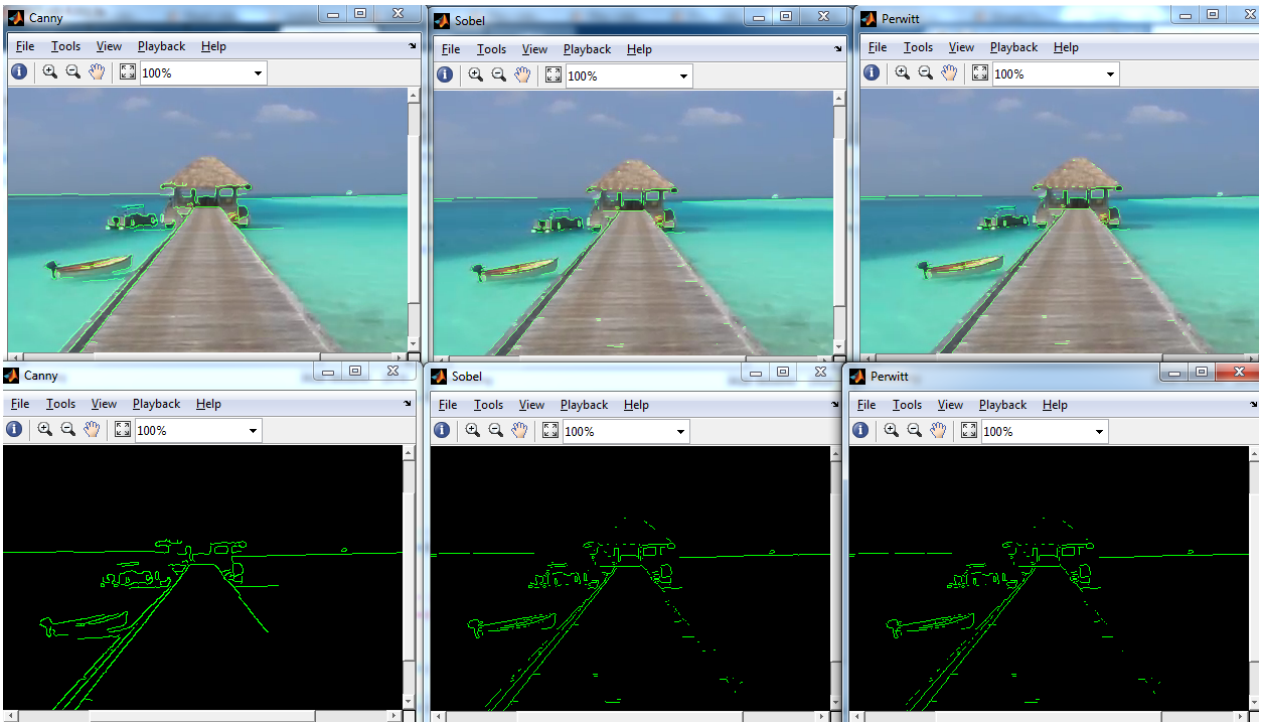

Figure 28: Figure showing that canny, that was doing better in some other frames, here is loosing relevant information of the main edges, giving a weak description of the frame.

3 Retrieval of images based on texture

3.1 Cresting the filters

Imagesc: this command, given an array C, displays it as a full color image. Each element of C specifies a pixel of the image.

What are the different filters? What values do?

The filters are supossed to detect textures with its forms when applied to an image. For this exercise we have edges shapes (green-blue background), bars (greenish background)and spots (dark blue and yellow backgrounds). Also, for each kind, we have different directions (edges and bars) and sizes, so they different features.

It is necessary to have different shapes, orientations and sizes in our bank of filters in order to detect all the possible textures.

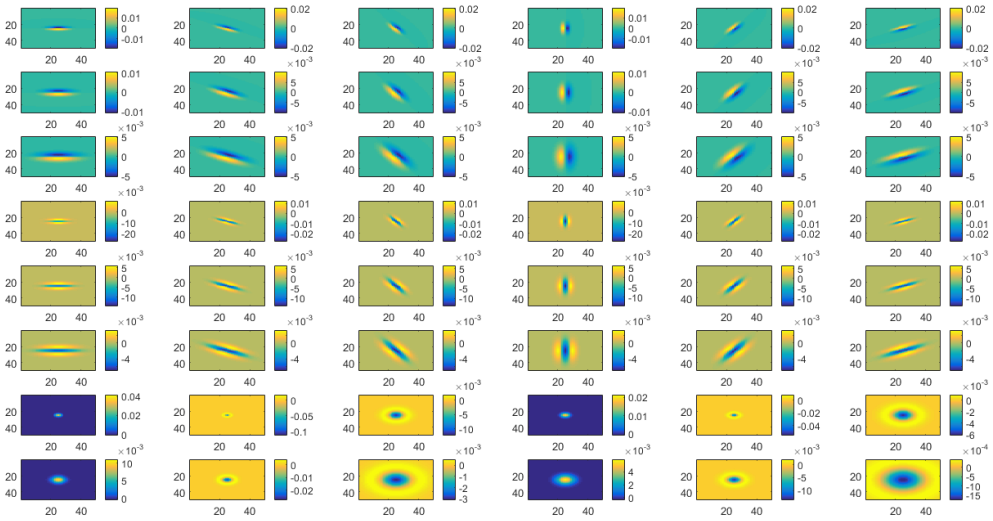

Figure 29: Filters

Colorbar: plots a vertical colorbar to the right of the axes when plotting a graph or an image.

3.2 Implement a function getFeatures() in Matlab

This is the matlab code that we implemented:

```
function [ texturesDescriptor ] = getFeatures(Image,Filters)
% ej_3.2
texturesDescriptor = zeros(1,size(Filters,3));

    for i=1:size(Filters,3)
        texturesDescriptor(1,i)=mean(mean(abs(imfilter(rgb2gray(Image),
            Filters(:,:,i)))));
    end
end
```

Figure 30: Result of the convolution with all the filters

As we can observe in the image, the filters that appear to get better results are the ones highlighted in ted. They get the biggest value of the mean of the convolution, and, as is it shown, detect many features of the image. For instance, the first one has detected almost all the shape of the trees.

The descriptors are an array 1x48, having one value for each of the 48 filters we have got.

3.3 Write a function getClassFeatures() in Matlab

Fullfile: this command given some string parameters will find a file with the name equal to all the parameters concatenated. *For example: fullfile('a/bf','d','.jpg') will get the file named 'a/bfd.jpg').*

For processing one directory we have implemented the next function. It returns both the feature matrix and the array of images we've used, as we will need them later.

```
1 function [ classFeatures, images ] = getClassFeatures( dirPath, extension )
2 files=dir(fullfile(dirPath,strcat('*.',extension)));
3 F=makeLMfilters(); % generating the filters
4 classFeatures=zeros(1,size(F,3));
5 imsize=size(imread(fullfile(dirPath, files(1).name)));
6 images=zeros(size(files,1),imsize(1),imsize(2),imsize(3));
7 for i=1:size(files,1)
8     im=imread(fullfile(dirPath, files(i).name));
```

```
9       images(i,:,:,:)=im;
10      classFeatures(i,:)=getFeatures(im, F);
11  end;
12
13  end
```

For retreiving the features from all the directories, we have implemented an aditional function that will simply call getClassFeatures() and store all the results in the same matrix.

```
1  function [ textureDescriptors, images ] = ...
       getClassFeaturesAllDirectories(extension, firstDirectory, varargin )
2  [textureDescriptors,images]=getClassFeatures(firstDirectory,extension);
3
4  for k = 1:length(varargin)
5      [textureDescriptors1,images1]=getClassFeatures(varargin{k},extension);
6      textureDescriptors=vertcat(textureDescriptors,textureDescriptors1);
7      images=vertcat(images,images1);
8  end
9  end
```

3.4 For each image in the three classes, write a function retrieveKImages() in Matlab

Using the feature matrix and the images obtained by calling getClassFeatures() or getClassFeaturesAllDirectories, we can compare the images and order them by similarity, considering texture descriptors.

In order to achieve this, we have used the matlab function knnsearch(), which executes the K-Nearest Neighbor algorithm.

We have to define a K (tipically an odd number), the image we want to compare (image), its features (imageFeatures), the images we want to compare against (images) and their features (textureDescriptors).

The function will plot the original image with the k most similar found, and will return the most similar images.

```
1  function [ moreSimilarImages ] = retrieveKImages( image, imageFeatures, images, ...
       textureDescriptors, k )
2  found=knnsearch(textureDescriptors,imageFeatures,'K',k);
3  figure;
4  subplot(k/3+1,3,2);
5  imshow(image);
6  title('Original image');
7
8  for i=1:k
9      im=found(i);
10     subplot(k/3+1,3,3+i);
11     imshow(uint8(squeeze(images(im,:,:,:))));
12     title('Retrieved image');
13 end
14 end
```

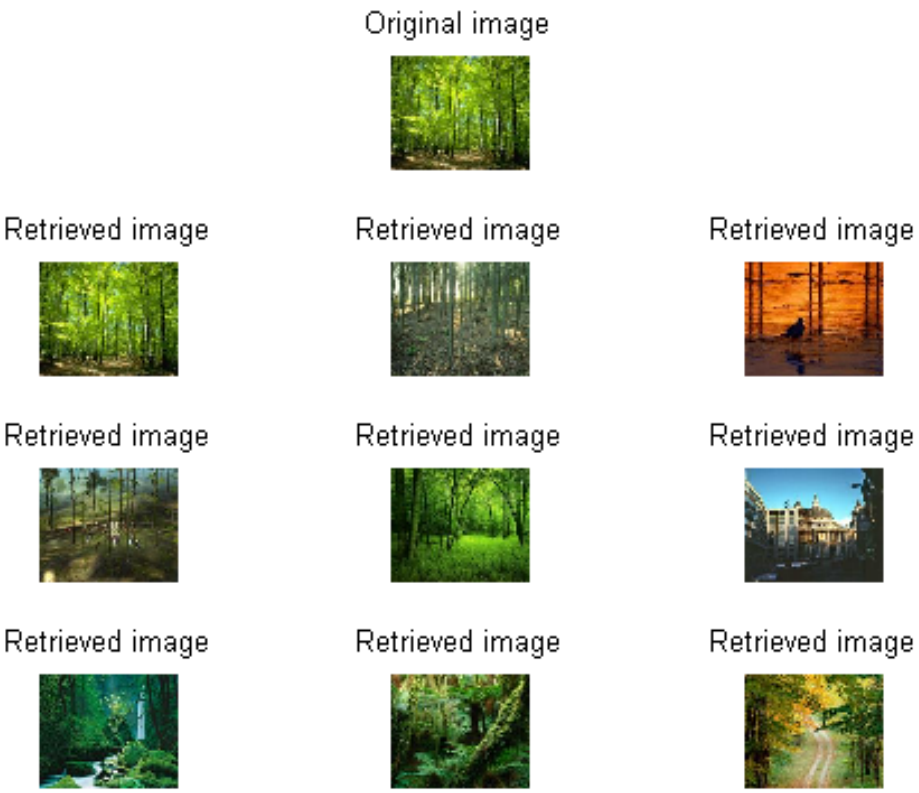

Figure 31: 9 most similar images to forest_9.png